

Statistical Machine Learning

Ulrike von Luxburg

Summer 2022

Department of Computer Science, University of Tübingen

(Version as of July 21, 2022)

Table of contents

Introduction to Machine Learning

What is machine learning?	7
Motivating examples and applications	8
Machine learning as inductive inference	18
(*) Warmup: The kNN algorithm	38
This lecture	65
Formal setup	68
Standard setup for supervised learning	69
Statistical and Bayesian decision theory	80
Optimal prediction functions in closed form	99
... for classification under 0-1 loss	100
... for regression under L_2 loss	112
Basic learning principles: ERM, RRM	119
Linear methods for supervised learning	
Linear methods for regression	142
Linear least squares regression	143
Feature representation of data	169
Feature normalization	179
Least squares with linear combination of basis functions	182
Ridge regression: least squares with L_2 -regularization	190
Lasso: least squares with L_1 -regularization	209
Selecting parameters by cross validation	226
Probabilistic interpretation of linear regression	240
Linear methods for classification	245

Table of contents (2)

Intuition	246
(*) Linear discriminant analysis	253
Logistic regression	276
(*) Probabilistic interpretation of linear classification	290
Evaluation of classification results	297
Linear support vector machines	330
Intuition and primal	331
Excursion: convex optimization, primal, Lagrangian, dual	356
Deriving the dual problem	357
Important properties of SVMs	373
Kernel methods for supervised learning	
Positive definite kernels	383
Intuition	384
Definition and properties of kernels	396
Reproducing kernel Hilbert space and feature maps	417
Support vector machines with kernels	440
Regression methods with kernels	460
Kernelized least squares	461
Kernel ridge regression	469
How to center and normalize in the feature space	474
Randomized methods: bagging, boosting and friends	
Random Forests	487
Boosting	515
(*) Gradient Boosting	534
Unsupervised learning	
Dimensionality reduction and embedding	537

Table of contents (3)

Classical PCA	538
Kernel PCA	565
(*) Multi-dimensional scaling	591
(*) Random projections and the Theorem of Johnson-Lindenstrauss	608
(*) Isomap	639
(*) t-SNE	653
Clustering	674
(*) K-means and kernel k-means	684
Standard k -means algorithm	685
(*) Linkage algorithms for hierarchical clustering	709
Graph-based machine learning algorithms: introduction	718
A glimpse on spectral graph theory	726
Unnormalized Laplacians	728
Normalized Laplacians	742
Cheeger constant	748
Spectral clustering	761
Unnormalized spectral clustering	771
Normalized spectral clustering	794
Regularized spectral clustering	799
Introduction to learning theory	
The classic theory for supervised learning	807
Classic Learning Theory: setup and main questions	808
Controlling the estimation error: generalization bounds	817
Capacity measures for function classes	834
Finite classes	835
Shattering coefficient	847
VC dimension	857

Table of contents (4)

Rademacher complexity	869
Controlling the approximation error	875
Getting back to Occam's razor	883
(*) The No-Free-Lunch Theorem	892
 A glimpse on modern results in learning theory	904
What is wrong with classic learning theory?	905
Why DNNs might work: Overfitting and the double descent curve	912
Why is there no contradiction to classic SLT?	926
Why we might really need large models: Smooth interpolation	930
 Machine Learning in the context of society	
The issues with ML	942
 Fairness	960
Why can ML be unfair?	961
Data and measurement	986
Some basic notions of fairness	996
Technical approaches to improve fairness	1020
Tradeoffs	1034
 Explainability	1040
 Energy footprint of ML	1071
 Low rank matrix methods	
Introduction: recommender systems, collaborative filtering	1085
Matrix factorization basics	1089
Low rank matrix completion	1097

Table of contents (5)

Compressed sensing	1140
(*) Ranking from pairwise comparisons	
Introduction	1185
Simple but effective counting algorithm	1194
Learning to rank	1214
Application: distance completion problem	1226
Spectral ranking	1239
Google page rank	1243
Meta ML: How does research work? In general, and in Tübingen	
How does research funding work? In general, and in Tübingen	1262
How to find a good PhD position?	1272
Publications and reviewing in ML	1279
Mathematical Appendix	
Recap: Probability theory	1288
Discrete probability theory	1289
Continuous probability theory	1327
Recap: Linear algebra	1341
The maths	1342
Some numerical procedures you should know	1381
Excursion to convex optimization: primal, dual, Lagrangian	1384
Convex optimization problems: intuition	1385
Lagrangian: intuitive point of view	1395
Lagrangian: formal point of view	1416

Introduction to Machine Learning

What is machine learning?

Motivating examples and applications

Hand-written digit recognition

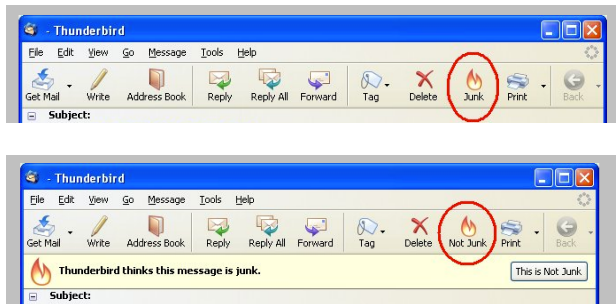
One of the “founding problems” of pattern recognition and machine learning:

Fachbereich Informatik

Saal 14

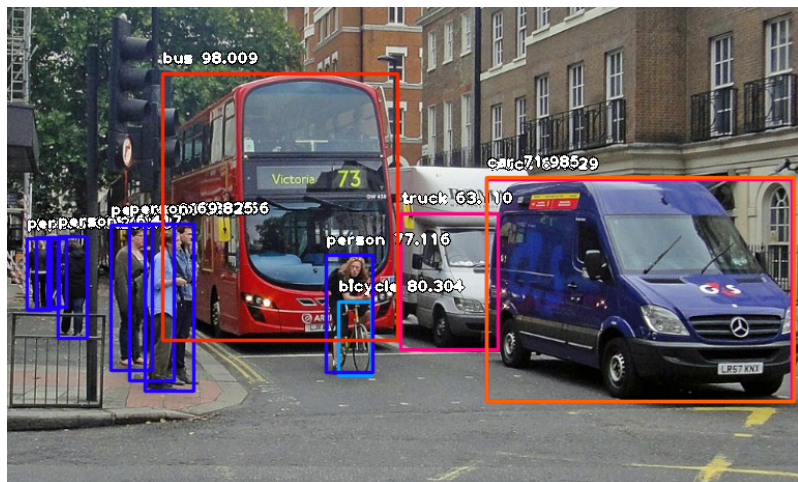
72076 Tübingen

Spam filtering



This is a typical “online learning problem”: training arrives in an online stream, rules have to be updated all the time

Object detection in general



Self-driving cars

- 2005, first breakthroughs in the DARPA grand challenge: Build an autonomous car that can find its way 100 km through the desert.



- By now: all major companies realize self-driving cars.



AlphaGo

Deep mind (google) constructed an computer player for the board game GO. In March 2016 it defeated the 18-times world champion, Lee Se-dol.



Machine learning in life sciences: Bioinformatics

Machine learning is used all over the place in bioinformatics:

- Classify different types of diseases based on microarray data

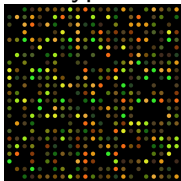
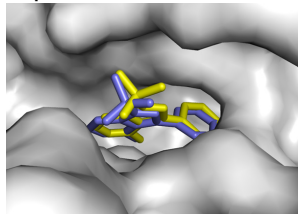
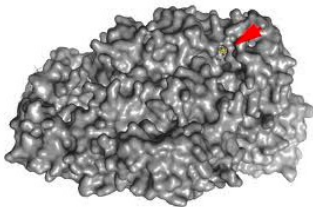


Image: Wikipedia

- Want to find drugs that can bind to a protein:



Images: BiochemLabSolutions.com

Machine learning in life sciences: Medical image analysis

Skin cancer detection:

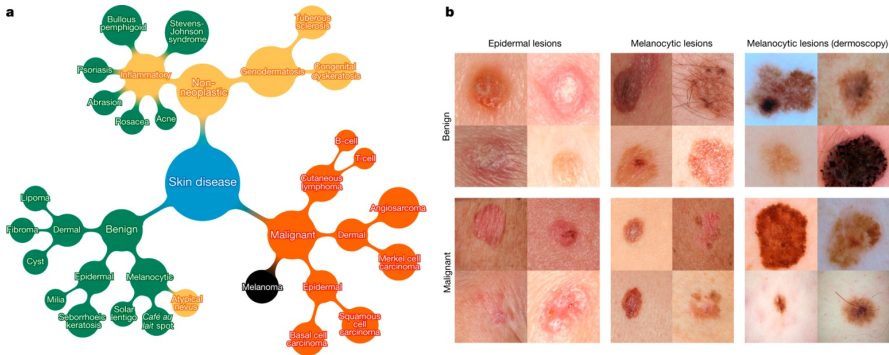


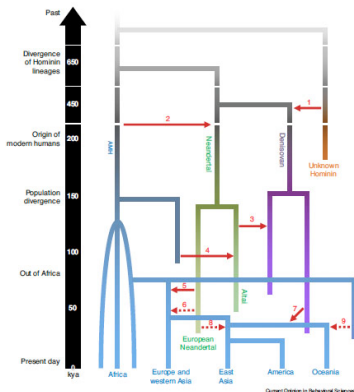
Figure taken from Esteva et al, Nature 2017

Machine learning in life sciences: Medical image analysis (2)

Machine learning achieves level comparable to best human experts.

Machine Learning in Science: Archaeology

ML approach to analyze the human genome finds evidence for unknown human ancestor:



Nature communications, 2019

An example of unsupervised learning.

Machine learning as inductive inference

What is machine learning?

First explanation:

- ▶ Development of algorithms which allow a computer to “learn” specific tasks from training examples.
- ▶ Learning means that the computer can not only memorize the seen examples, but can generalize to previously unseen instances
- ▶ Ideally, the computer should use the examples to extract a general “rule” how the specific task has to be performed correctly.

Deduction vs. Induction

WHO KNOWS WHAT INDUCTION AND DEDUCTION MEAN?

Deduction vs. Induction (2)

Deductive inference is the process of reasoning from one or more general statements (premises) to reach a logically certain conclusion.

Example:

- ▶ Premise 1: every person in this room is a student.
- ▶ Premise 2: every student is older than 10 years.
- ▶ Conclusion: every person in this room is older than 10 years.

If the premises are correct, then all conclusions are correct as well.

Nice in theory. Mathematics is based on this principle.
But no natural way to deal with uncertainty regarding the premises.

Deduction vs. Induction (3)

Inductive inference: reasoning that constructs or evaluates general propositions that are derived from specific examples.

Example:

- ▶ We drop lots of things, very often.
- ▶ In all our experiments, the things fall downwards, not upwards.
- ▶ So we conclude that likely, things always fall downwards when we drop them.

Very important: we can never be sure, our conclusion can be wrong!

Humans do inductive reasoning all the time: we draw uncertain conclusions from our relatively limited experiences.

Machine learning as inductive inference

Here comes now our second, more abstract description of what machine learning is:

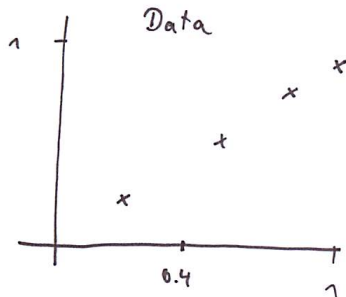
Machine learning tries to automate the process of inductive inference.

Why should machine learning work at all?

Consider the following simple regression example:

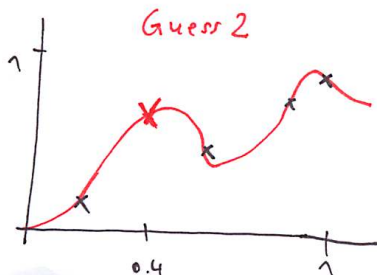
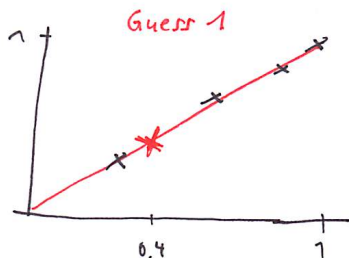
- ▶ Given: input-output pairs (X_i, Y_i) , $X_i \in \mathcal{X}$, $Y_i \in \mathcal{Y}$.
- ▶ Goal: learn to predict the Y -values from the X -values, that is we want to “learn” a suitable function $f : \mathcal{X} \rightarrow \mathcal{Y}$.

EXAMPLE 1: WHAT DO YOU BELIEVE IS THE VALUE $f(0.4)$?



Why should machine learning work at all? (2)

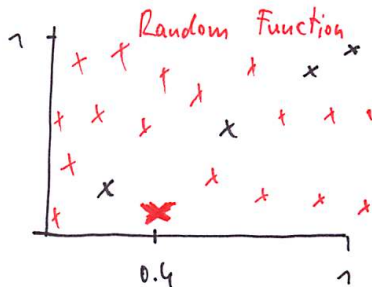
Here are two guesses:



WHICH ONE IS BETTER?

Why should machine learning work at all? (3)

Now I tell you that, in fact, the function values Y_i have been generated by a uniform random number generator.



WHAT DO YOU PREDICT NOW AS THE OUTPUT TO THE INPUT 0.4?

Why should machine learning work at all? (4)

Consequence 1: we will only be able to learn if “there is something we can learn”.

- ▶ Output Y “has something to do” with input X
- ▶ “Similar inputs” lead to “similar outputs”
- ▶ There is a “simple relationship” or “simple rule” to generate the output for a given input
- ▶ The function f is “simple” (but caution, this is not the end of the story, see later in the section on learning theory)

These assumptions are rarely made explicit, but something along this line has to be satisfied, otherwise ML is doomed.

Why should machine learning work at all? (5)

Consequence 2: We need to have an idea what we are looking for. This is called the “inductive bias”. Learning is impossible without such a bias.

Let's try to get some intuition for what this means.

Inductive bias: very simple example

Discrete input space $\mathcal{X} = \{0.01, 0.02, \dots, 1\}$.

Output space: $\mathcal{Y} = \{0, 1\}$

Given: training examples $(X_i, Y_i)_{i=1, \dots, n} \subset \mathcal{X} \times \mathcal{Y}$, assume there is no label noise (all training labels are correct).

Goal: Learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ based on the examples

Case 1: no inductive bias, every function $f : \mathcal{X} \rightarrow \mathcal{Y}$ can be the correct one.

Formally:

- we want to find a function out of $\mathcal{F} := \mathcal{Y}^{\mathcal{X}}$ (the space of all functions). This space contains 2^{100} functions.

Inductive bias: very simple example (2)

- ▶ Now assume we have already 5 training points and their labels.
- ▶ This means that we can rule out all functions from \mathcal{F} which do not satisfy $f(X_i) = Y_i$.
So we are left with 2^{95} possible functions.
- ▶ Now we want to predict the value at a previously unseen point $X' \in \mathcal{X}$.
- ▶ There are 2^{94} remaining functions with $f(X') = 0$ and the same number of functions with $f(X') = 1$.
And there is no way we can decide which one is going to be the best prediction.
- ▶ In fact, no matter how many data points we get, our prediction on unseen points will be as bad as random guessing.

Without any further restrictions or assumptions, the problem of machine learning would be ill posed!

Inductive bias: very simple example (3)

A more formal way of stating this result is called the “No free lunch theorem”. See the slides in the chapter on learning theory (we won’t discuss it in the lecture).

Inductive bias: very simple example (4)

Case 2: model with an inductive bias.

- ▶ Assume we know that the true function is one out of two functions: either the constant one function **1** or the constant zero function **0**.
Our hypothesis space is $\mathcal{F} = \{0, 1\}$.
- ▶ Again we assume that no noise exists.
- ▶ Then, after we have observed one training example, we know exactly which function is the correct one and can make predictions without any error.

If we have a (strong enough) inductive bias, we can predict based on few training examples.

Inductive bias: very simple example (5)

A bit too simplistic?

- ▶ Yes, the hypothesis \mathcal{F} seems too restricted to be useful for practice. The problem of selecting a good hypothesis class is called **model selection**.
- ▶ And yes, we did not take noise into account (yet).
- ▶ And yes, we did not talk about what happens if the true function is not contained in \mathcal{F} after all.

The details of all this are quite tricky. **It is the big success story of classical machine learning theory to work out how exactly all these things come together.**

At the end of the course you will know all this, at least roughly 😊

Excursion: Inductive bias in animal learning

Any “system” that learns has an inductive bias. Consider learning in animals:

Rats get two choices of water. One choice makes them feel sick, the other one doesn't.

Experiment 1:

- ▶ Two types of water taste differently (neutral / sweet taste).
- ▶ Rats learn very fast not to drink the water that makes them sick.

Excursion: Inductive bias in animal learning (2)

Experiment 2:

- ▶ Same taste, but one type of water is presented together with “audio-visual stimuli” (certain sounds and light conditions), while the other type of water is presented without these audio-visual stimuli.
- ▶ In this setting, rats did NOT learn to avoid the water that makes them sick.
- ▶ Apparently, they cannot make a connection between “sound of the food” and “sickness”.

Excursion: Inductive bias in animal learning (3)

Explanation:

- From the point of view of evolution, it makes a lot of sense that the taste of food is related to whether it makes sick or not, whereas this does not seem so useful for sounds coming with food.

In our words: the rat has an inductive bias!

In psychology, this effect is called the “Garcia effect” (published in a line of papers by John Garcia and co-workers in the 1960ies).

Reference: Garcia, John and Brett, Linda Phillips and Rusiniak, Kenneth W. Limits of Darwinian conditioning. In S.B. Klein and R.R. Mowrer, editors, Contemporary learning theories: instrumental conditioning theory and the impact of biological constraints, pages 181-204, 1989.

Inductive bias, bottom line for now

Any successful learning algorithm has an inherent inductive bias.

- ▶ We prefer to select a hypothesis from some “restricted” or “small” function space \mathcal{F} .
- ▶ Whether this function is “close to the truth” depends on whether the model class \mathcal{F} is “selected well” for the problem at hand.
- ▶ Note: for some algorithms it is obvious what the inductive bias is. For some algorithms it is hard to understand what exactly the bias is. **But if the algorithm works, there HAS TO BE a bias. This is very important to keep in mind.**

All these things will be made precise during the course of this lecture.

(*) Warmup: The kNN algorithm

Literature:

For the algorithm:

Hastie, Tibshirani, Friedman Section 2.3.2

Duda, Hart Section 4.5

For theory (not covered in this lecture): Devroye, Györfi, Lugosi: A Probabilistic Theory of Pattern Recognition

A simple machine learning experiment

c Data:

- ▶ Take a set of **training points and labels** $(X_i, Y_i)_{i=1, \dots, n}$. The machine learning algorithm has access to this training input and can use it to generate a classification rule $f : \mathcal{X} \rightarrow \{0, 1\}$.
- ▶ Take a set of **test points** $(X_j, Y_j)_{j=1, \dots, m}$. This set is independent from the training set (“previously unseen points”) and will be used to evaluate the success of the training algorithm.

A simple machine learning experiment (2)

Assume our machine learning algorithm has used the training data $(X_i, Y_i)_{i=1, \dots, n}$ to construct a rule f_{alg} for predicting labels.

Training error:

- ▶ Predict the labels of all training points: $\hat{Y}_i := f_{alg}(X_i)$.
- ▶ Compute the error (“loss”) of the classifier on each training point:

$$\ell(X_i, Y_i, \hat{Y}_i) := \begin{cases} 0 & \text{if } \hat{Y}_i = Y_i \\ 1 & \text{otherwise} \end{cases}$$

This is called the “pointwise 0-1-loss”.

A simple machine learning experiment (3)

- Define the training error of the classifier (“risk of the classifier”) as the average error over all training points:

$$R_{train}(f_{alg}) = \frac{1}{n} \sum_{i=1}^n \ell(X_i, Y_i, f_{alg}(X_i))$$

Later we will call this quantity the “empirical risk” of the classifier (with respect to the 0-1-loss).

A simple machine learning experiment (4)

Test error:

- ▶ Predict the labels of all test points: $\hat{Y}_j := f_{alg}(X_j)$.
- ▶ Compute the error (loss) of the classifier on each test point:

$$\ell(X_j, Y_j, \hat{Y}_j) := \begin{cases} 0 & \text{if } \hat{Y}_j = Y_j \\ 1 & \text{otherwise} \end{cases}$$

- ▶ Define the test error (risk) of the classifier as the average error over all test points:

$$R_{test}(f_{alg}) = \frac{1}{m} \sum_{j=1}^m \ell(X_j, Y_j, f_{alg}(X_j))$$

A simple machine learning experiment (5)

Technical remark:

- The quantity R_{test} as defined above is an empirical quantity (it depends on the test set). Later, we will define the true risk R of the classifier, which is the expectation over this quantity.

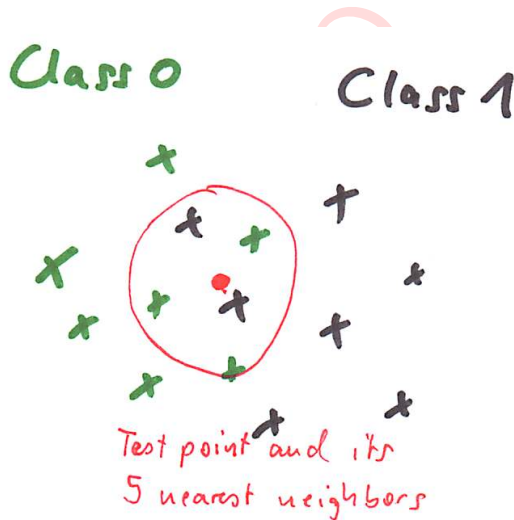
A simple machine learning experiment (6)

Remarks:

- ▶ Obviously, it is not so much of a challenge for an algorithm to correctly predict the training labels (after all, the algorithm gets to know these labels).
- ▶ Still, machine learning algorithms usually make training errors, that is they construct a rule f_{alg} that does not perfectly fit the training data.
- ▶ But the crucial measure of success is the performance of the classifier on an independent test set.
- ▶ In particular, it is not the case that a low training error automatically indicates a low test error or vice versa.

The kNN classifier

Informally, the idea is this:



The kNN classifier (2)

Given: Training points $(X_i, Y_i)_{i=1, \dots, n} \subset \mathcal{X} \times \{0, 1\}$ and a distance function $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.

Goal: Construct a classifier f that predicts the labels from the inputs.

- ▶ Given a test point X' , compute all distances $d(X', X_i)$ and sort them in ascending order.
- ▶ Let X_{i_1}, \dots, X_{i_k} be the first k points in this order (the k nearest neighbors of X'). We denote the set of these points by $\text{kNN}(X')$.
- ▶ Assign to Y' the majority label among the corresponding labels Y_{i_1}, \dots, Y_{i_k} , that is define

$$Y' = \begin{cases} 0 & \text{if } \sum_{j=1}^k Y_{i_j} \leq k/2 \\ 1 & \text{otherwise} \end{cases}$$

Influence of the parameter k

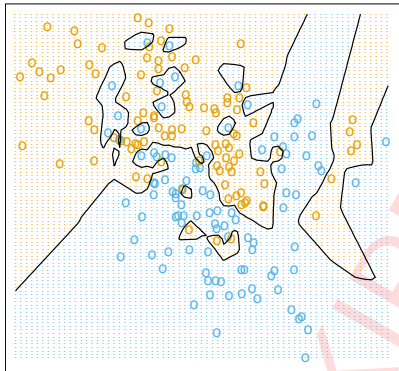
The classification result will depend on the parameter k .

WHAT DO YOU THINK, IS IT BETTER TO HAVE k SMALL OR LARGE?

SKIPPED

Influence of the parameter k (2)

1-Nearest Neighbor Classifier



15-Nearest Neighbor Classifier

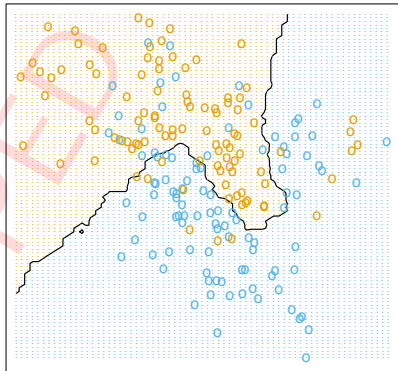


Figure from Hastie/Tibshirani/Friedman:

- ▶ Yellow/blue circles: training points and their labels
- ▶ Yellow/blue little dots: if this were a test point, the kNN classifier would classify the point as yellow/blue.

Influence of the parameter k (3)

Generally:

- ▶ k too small \leadsto overfitting
(Extreme case: $k = 1$, very wiggly and prone to noise, zero training error)
- ▶ k too large \leadsto underfitting
(Extreme case: $k = n$, then every point gets the same label, namely the overall majority label)
- ▶ Theoretical analysis can reveal: k should be roughly of order $\log n$ as $n \rightarrow \infty$ (we won't prove it in this course, if you are interested you might want to consider the book by Devroye et al., see literature list).

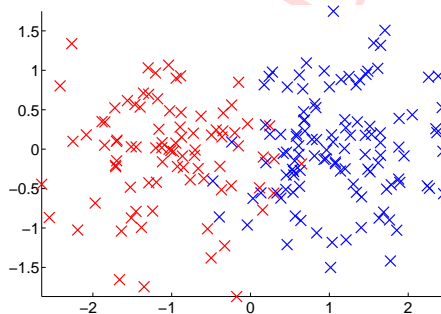
Application: simple mixture of Gaussians

Recap:

- ▶ Normal distribution in 1 dimension
- ▶ Multivariate normal distribution
- ▶ Mixture of Gaussians

Application: simple mixture of Gaussians (2)

We draw 100 points randomly from a mixture of two Gaussian distributions. The figure shows a typical training set:



Application: simple mixture of Gaussians (3)

Conduct the following experiment:

- 1 **for** $rep = 1, \dots, 10$
- 2 Draw n training points $(X_i, Y_i)_{i=1, \dots, n}$
- 3 Draw m test points $(X'_i, Y'_i)_{i=1, \dots, m}$
- 4 **for** $k = k_1, \dots, k_s$
- 5 Predict the labels of all training points, using the k nearest training points
- 6 $ErrTrain(k, rep)$ = the training error, averaged over all training points
- 7 Predict the labels of all test points, using the k nearest training (!) points
- 8 $ErrTest(k, rep)$ = the test error, averaged over all test points
- 9 **return** For each k , return the average train and test error (where the average is taken over the repetitions)

Application: simple mixture of Gaussians (4)

... see matlab demo: `demo_knn_classifier()`
(play with size of training set, and with separation of classes)

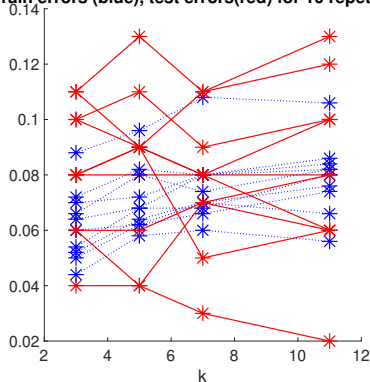
The following figure shows the results:

- ▶ Left figures: errors in each individual repetition
- ▶ Right figure: errors averaged over all repetitions

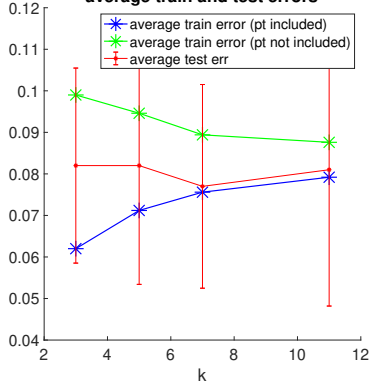
Application: simple mixture of Gaussians (5)

mixture of gaussians, 500 train and 100 test points

Train errors (blue), test errors(red) for 10 repetitions



average train and test errors



(x -Axis: parameter k , y -axis: error)

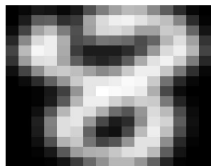
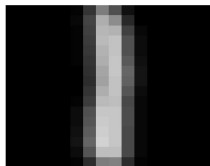
Application: simple mixture of Gaussians (6)

Note:

- ▶ For the kNNclassifier, the training error and test error are about the same (it does not really “train” in the sense that it selects a function that is particularly good on the training data).
- ▶ Depending on whether a point is considered to be part of its own kNN neighborhood or not, the train error differs a bit.

Application: hand written digits

Data:



Represented as 16×16 greyscale image. That is, each digit corresponds to a vector of length 256 with entries in $[0, 1]$.

Task 1: Learn to distinguish between 1 and 8

Task 2: Learn to distinguish between 3 and 8

Application: hand written digits (2)

Setup:

- ▶ To apply the kNN rule we need to define a distance function between digits.
- ▶ For simplicity, we use the Euclidean distance between the vectors:

for $X = (X_1, \dots, X_{256})^t$ and $X' = (X'_1, \dots, X'_{256})^t$ we set

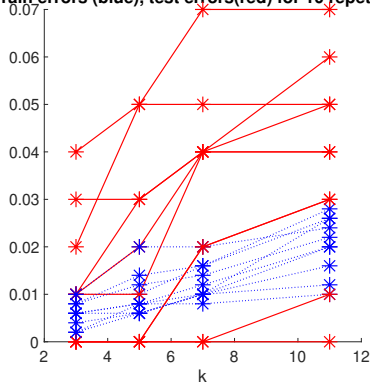
$$d(X, X') = \left(\sum_{s=1}^{256} (X_s - X'_s)^2 \right)^{1/2}$$

Application: hand written digits (3)

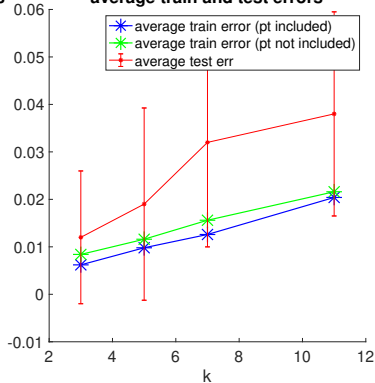
Results task 1 (digit 1 vs digit 8):

Digits 1 vs 8, 500 train and 100 test points

Train errors (blue), test errors (red) for 10 repetitions



average train and test errors



(x -Axis: parameter k , y -axis: error)

Application: hand written digits (4)

WHAT DO YOU THINK, IS THIS GOOD OR BAD?

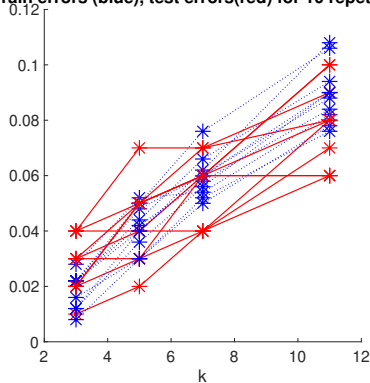
SKIPPED

Application: hand written digits (5)

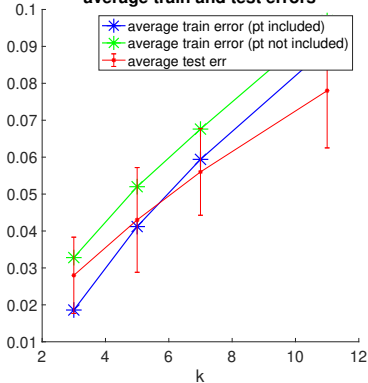
Results task 2 (digit 3 vs digit 8):

Digits 3 vs 8, 500 train and 100 test points

Train errors (blue), test errors (red) for 10 repetitions



average train and test errors



These results are surprisingly good!!!

Influence of the similarity function

The choice of the similarity function is crucial as well:

- The performance of kNNrules can only be good if the distance function encodes the “relevant information”.

Example: you want to classify mushrooms as “edible” or “not edible” and as distance function between mushrooms you use the difference in weight ...

- In many applications it is not so obvious how to define a good distance or similarity function

Example: you want to classify the genre of songs. How do you compute a similarity between different songs???

Inductive bias

WHAT DO YOU THINK IS THE INDUCTIVE BIAS IN THIS ALGORITHM? WHAT KIND OF FUNCTIONS ARE “PREFERRED” OR “LEARNED”?

SKIPPED

Inductive bias

WHAT DO YOU THINK IS THE INDUCTIVE BIAS IN THIS ALGORITHM? WHAT KIND OF FUNCTIONS ARE “PREFERRED” OR “LEARNED”?

Input points that are close to each other should have the same label

Extensions

- ▶ The kNN rule can easily be used for regression as well: As output value take the average over the labels in the neighborhood.
- ▶ kNN-based algorithms can also be used for many other tasks such as density estimation, outlier detection, clustering, etc. We will see more examples during the semester.

SKIPPED

Summary

In practice:

- ▶ The kNN classifier is about the simplest classifier that exists.
- ▶ But often it performs quite reasonably.
- ▶ Whatever your specific machine learning task is, you should always consider the kNN classifier as a baseline.

In theory:

- ▶ One can prove that in the limit of infinitely many data points, the kNN classifier is “consistent”, that is it learns the best possible function (see next lecture).

This lecture

Contents of the lecture

In this lecture: Statistical machine learning, the classical point of view

We do not cover

- ▶ Probabilistic / Bayesian ML
- ▶ Neural networks and deep learning
- ▶ Reinforcement learning

Organization

- ▶ Who am I?
- ▶ Who are you?
- ▶ Lecture: in person (videos exist, but small parts will change).
- ▶ Tutorials: please register in Ilias by tomorrow!!!
- ▶ Assignments: groups of 2 students. If you already have a buddy, indicate him/her in your tutorial registration.
- ▶ Exams and admissions
- ▶ Pre-requisites: maths for ML! Do check out the videos if you are unsure.
- ▶ Anything else?

Formal setup

Standard setup for supervised learning

The underlying space

- ▶ Input space \mathcal{X} , output space \mathcal{Y}
 - ▶ Sometimes, the spaces \mathcal{X} or \mathcal{Y} have some mathematical structure (topology, metric, vector space, etc), or we try to construct such a structure.
 - ▶ We assume that each space endowed with a σ -algebra, to be able to define a probability measure on the space. We ignore this issue in the following (for real world machine learning this is not an issue).
- ▶ Probability distribution P on the product space $\mathcal{X} \times \mathcal{Y}$ (with product sigma algebra)
 - ▶ no assumption on the form of the probability distribution
 - ▶ both input variables and output variables (!) are random quantities

A classifier / prediction function

A classifier or a prediction function is simply a function $f : \mathcal{X} \rightarrow \mathcal{Y}$.

- ▶ If \mathcal{Y} discrete: classification
- ▶ If $\mathcal{Y} = \mathbb{R}$: regression
- ▶ Other output spaces are possible as well, for example "structured prediction"

We now need to be able to measure how "good" a classifier / prediction function is:

Loss function

The loss function measures how “expensive” an error is:

A **loss function** is a function $\ell : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_{\geq 0}$.

Example:

- ▶ The **0-1-loss function** for classification is defined as

$$\ell(x, y, y') = \begin{cases} 0 & \text{if } y = y' \\ 1 & \text{otherwise} \end{cases}$$

- ▶ The **squared loss** for regression is defined as

$$\ell(x, y, y') = (y - y')^2$$

Note: the choice of a loss function influences the inductive bias.

Loss function (2)

Note:

- In some applications, it is important that the loss also depends on x .

CAN YOU COME UP WITH AN EXAMPLE?

- In some applications, it is important that the loss depends on the order of y and y' (the type of error)

CAN YOU COME UP WITH AN EXAMPLE?

True Risk

The **true risk** (or **true expected loss**) of a prediction function $f : \mathcal{X} \rightarrow \mathcal{Y}$ (with respect to loss function ℓ) is defined as

$$R(f) := E(\ell(X, Y, f(X)))$$

where the expectation is over the random draw of (X, Y) according to the probability distribution P on $\mathcal{X} \times \mathcal{Y}$.

The goal of machine learning is to use the training data to construct a function f_n whose **true risk is as small as possible**.

Bayes risk and Bayes classifier

What is the best function we can think of?

- The **Bayes risk** is defined as

$$R^* := \inf \{ R(f) \mid f : \mathcal{X} \rightarrow \mathcal{Y}, f \text{ measurable} \}$$

(we won't discuss measurability, if you've never heard of it then simply assume that f can be any function you want).

- In case the infimum is attained, the corresponding function

$$f^* := \operatorname{argmin} R(f)$$

is called the **Bayes classifier** / **Bayes predictor**.

The training data and learning

Assume we are given supervised training data:

- We draw n training points $(X_i, Y_i)_{i=1, \dots, n} \in \mathcal{X} \times \mathcal{Y}$ i.i.d. (independent and identically distributed) according to probability distribution P .

Note: “i.i.d.” is a strong assumption!!! Can you come up with a situation where this might not be satisfied?

The goal of learning is to construct a function f_n that has true risk close to the Bayes risk, that is $R(f_n) \approx R^*$.

Consistency of a learning algorithm

Consider an infinite sequence of data points $(X_i, Y_i)_{i \in \mathbb{N}}$ that have been drawn i.i.d. from distribution P over $\mathcal{X} \times \mathcal{Y}$. Denote by f_n the learning rule that has been constructed by an algorithm \mathcal{A} based on the first n training points.

- We say that the algorithm \mathcal{A} is **consistent** (for probability distribution P) if the risk $R(f_n)$ of its selected function f_n converges to the Bayes risk, that is

$$\forall \varepsilon > 0 : \lim_{n \rightarrow \infty} P(R(f_n) - R^* > \varepsilon) = 0.$$

(Note: Here convergence is “in probability”, for those who know what that means; if we have convergence almost surely, the algorithm is called **strongly consistent**. If you don't know these notions, don't worry)

Consistency of a learning algorithm (2)

- We say that algorithm \mathcal{A} is **universally consistent** if it is consistent for all possible probability distributions P over $\mathcal{X} \times \mathcal{Y}$.

Ultimately, what we want to find learning algorithms that are universally consistent: No matter what the underlying probability distribution is, when we have seen “enough data points”, then the true risk of our learning rule f_n will be arbitrarily close to the best possible risk.

Consistency of a learning algorithm (3)

For quite some time it was unknown whether universally consistent algorithms exist at all. The first positive answer was in 1977 when Stone proved that the kNN classifier is universally consistent.

Since then many algorithms have been found to be universally consistent, among them support vector machines, boosting, random forests, and many more.

Understanding the underlying principles behind these algorithms is the focus of this course, and in the learning theory part we will take a glimpse on how to get consistency statements.

Statistical and Bayesian decision theory

Literature:

- ▶ Hastie, Section 2.4 - 2.9 (parts only)
- ▶ Devroye, Section 2
- ▶ Duda/Hart, Section 2 (only parts of it, very technical)

What if we know all the underlying quantities?

Before we dive into machine learning principles, let's consider how we would solve classification if we had perfect knowledge of the probability distribution P .

Running example: Male or female?

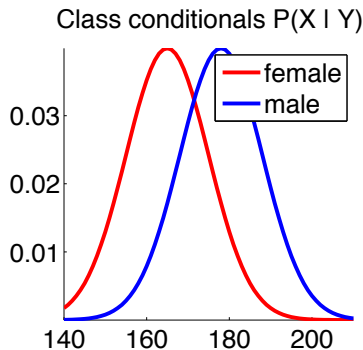
Predict gender of a person from body height:

HOW WOULD YOU PROCEED? ???

Running example: Male or female?

Predict gender of a person from body height:

HOW WOULD YOU PROCEED? ???



GIVEN THIS INFORMATION, HOW WOULD YOU LABEL THE INPUT $X = 160$?

Approach 1: just look at priors (a bit stupid)

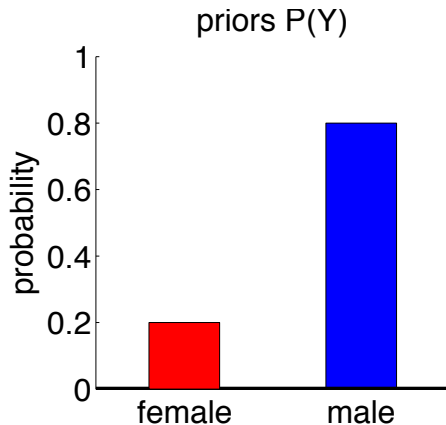
Decide based on class prior probabilities $P(Y)$.

- If you don't have any clue what to do, you could simply use the following rule:
You always predict the label of the “larger class”, that is

$$f_n(X) = \begin{cases} m & \text{if } P(Y = m) > P(Y = f) \\ f & \text{otherwise} \end{cases}$$

Approach 1: just look at priors (a bit stupid) (2)

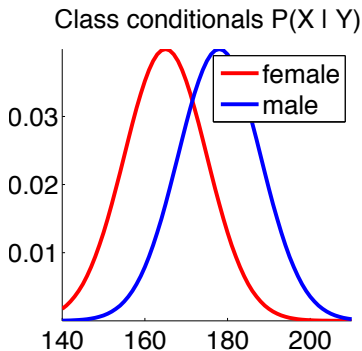
Visually: select the higher bar



Approach 2: maximum likelihood principle

Decide based on the likelihood functions $P(X|Y)$ (maximum likelihood approach).

- Consider the **class conditional distributions** $P(X|Y=m)$ and $P(X|Y=f)$.



Approach 2: maximum likelihood principle (2)

- Then predict the label with the higher likelihood:

$$f_n(x) = \begin{cases} m & \text{if } P(X = x|Y = m) > P(X = x|Y = f) \\ f & \text{otherwise} \end{cases}$$

Visually: select according to which curve is higher

Approach 3: Bayesian a posteriori criterion

Decide based on the posterior distributions $P(Y|X)$ (“Bayesian maximum a posteriori approach”):

- Compute the posterior probabilities

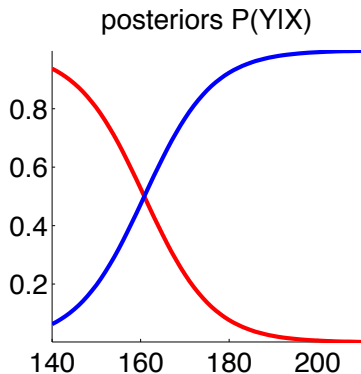
$$P(Y = m|X = x) = \frac{P(X = x|Y = m) \cdot P(Y = m)}{P(X = x)}$$

- Predict by the following rule:

$$f_n(x) = \begin{cases} m & \text{if } P(Y = m|X = x) > P(Y = f|X = x) \\ f & \text{otherwise} \end{cases}$$

Approach 3: Bayesian a posteriori criterion (2)

Visually: select according to which curve is higher



(figure is for uniform prior)

Approach: also take costs of errors into account

Take the “costs” of errors into account:

- ▶ Define a loss function $\ell(x, y, \hat{y})$ that tells you how much loss you incur by classifying the label of x as \hat{y} if the true label is y .
- ▶ The risk $R(\hat{y}|X = x) := E(\ell(x, Y, \hat{y}))$ is the expected loss we incur at point x when predicting \hat{y} (where the expectation is over the randomness in the sample, in this case only the randomness concerning the true label Y of x).
- ▶ Consider the expected conditional risk at point x

$$\begin{aligned} R(\hat{y}|X = x) &= \ell(x, m, \hat{y})P(Y = m \mid X = x) \\ &\quad + \ell(x, f, \hat{y})P(Y = f \mid X = x) \end{aligned}$$

- ▶ Use **Bayes decision rule**: Select the label $f_n(X)$ for which the conditional risk is minimal.

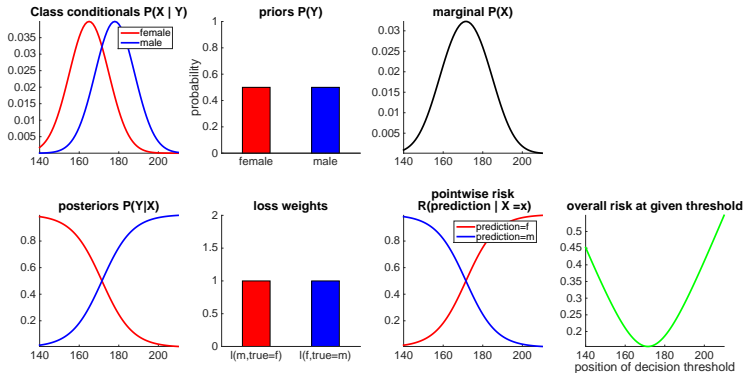
Example: male vs female

Run `demo_bayesian_decision_theory.m`

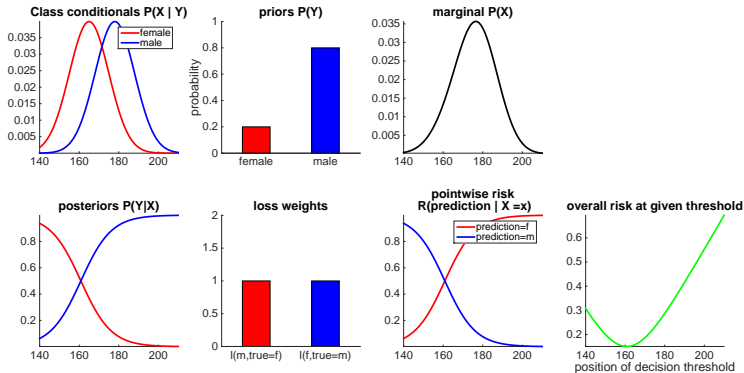
On the following slides you see results of a simulations for the following (stylized) input model: $X \in \mathbb{R}$ height of a person, $Y \in \{m, f\}$ gender of a person.

For your reference, the formulas for all plots follow after the plots.

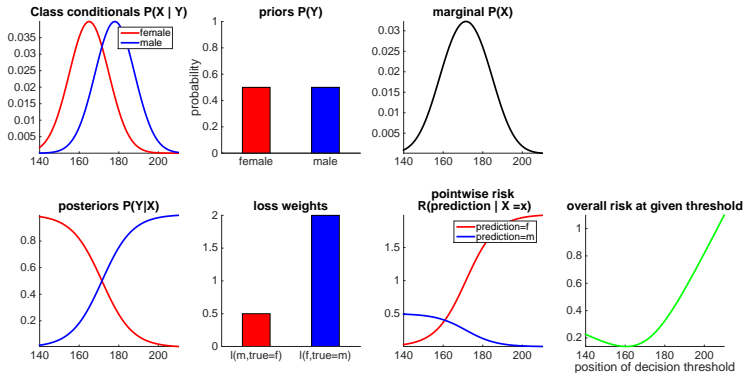
Example: male vs female (2)



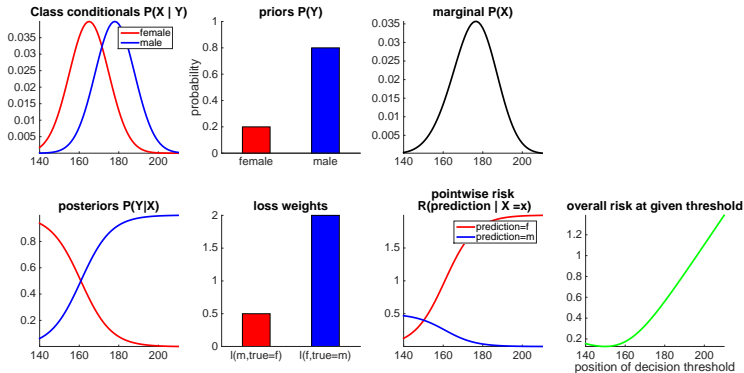
Example: male vs female (3)



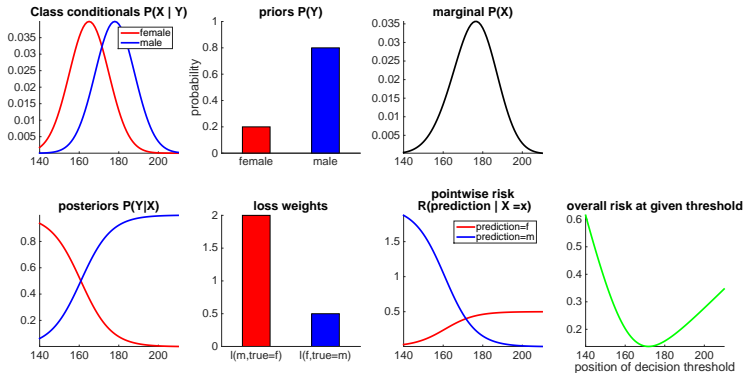
Example: male vs female (4)



Example: male vs female (5)



Example: male vs female (6)



Example: male vs female (7)

Formulas to produce the plots:

Plot 1 (Class conditionals): we plot $P(X = x \mid Y = m)$ (blue curve) and $P(X = x \mid Y = f)$ (red curve) as a function of $x \in \mathbb{R}$.

Plot 2 (Prior class weights): $\pi_f := P(Y = f)$ and $\pi_m := P(Y = m)$.

Plot 3 (Marginal of X): we plot $P(X = x)$ as a function of x , using $P(X = x) = P(X = x \mid Y = f)\pi_f + P(X = x \mid Y = m)\pi_m$

Plot 4 (Posteriors): We plot $P(Y = f \mid X = x)$ as a function of x (red curve), using the formula

$$P(Y = f \mid X = x) = \frac{P(Y=f, X=x)}{P(X=x)} = \frac{P(X=x|Y=f)P(Y=f)}{P(X=x)}.$$
 Similarly in blue for the case $Y = m$.

Example: male vs female (8)

Plot 5 (loss weights): We plot the weights of the losses:

$\ell(\hat{Y} = \hat{y}, Y = y)$. In particular, ℓ_{mf} is the loss if the true label is f and we predict m .

Plot 6 (pointwise risk): we plot in red the risk of predicting outcome “f” at input $X = x$. We only incur a loss for those people who are male but we predict female.

$$R(Y = f | X = x) = P(Y = m | X = x) \cdot \ell_{fm}$$

For male analogously in blue.

Plot 7 (overall risk at threshold): Assume that we use threshold t for predicting the gender: everybody with size $\leq t$ is predicted female, everybody with size $> t$ is predicted male. We now plot the overall risk of this classifier, as a function of the threshold t . To compute it from the

Example: male vs female (9)

given quantities, observe that if we threshold at t , then all males smaller than t and all females larger than t get misclassified. To compute the risk, we thus use

$$\begin{aligned} R_t &= P(Y = m \mid X \in]-\infty, t]) \cdot \ell_{fm} + P(Y = f \mid X \in]t, \infty]) \cdot \ell_{mf} \\ &= \ell_{fm} \int_{-\infty}^t P(Y = m \mid X = x) dx + \ell_{mf} \int_t^{\infty} P(Y = f \mid X = x) dx \end{aligned}$$

Optimal prediction functions in closed form

... for classification under 0-1 loss

Regression function (context of classification)

Consider (X, Y) drawn according to a probability distribution P on the product space $\mathcal{X} \times \{0, 1\}$. We want to describe the distribution P in terms of two other quantities:

- ▶ Let μ be the marginal distribution of X , that is $\mu(A) = P(X \in A)$.
- ▶ Define the so-called regression (!)-function:

$$\eta(x) := E(Y \mid X = x)$$

- ▶ In the special case of classification, the regression function can be rewritten as

$$\begin{aligned}\eta(x) &= 0 \cdot P(Y = 0 \mid X = x) + 1 \cdot P(Y = 1 \mid X = x) \\ &= P(Y = 1 \mid X = x)\end{aligned}$$

Regression function (context of classification) (2)

Intuition:

- ▶ If $\eta(x)$ is close to 0 or close to 1, then classifying x is easy.
- ▶ If $\eta(x)$ is close to 0.5, then classifying x is difficult.

WHY?

Regression function (context of classification) (3)

Proposition 1 (Unique decomposition)

The probability distribution P is uniquely determined by μ and η .

Intuition (discrete case): We can rewrite

$$\begin{aligned}P(X = x, Y = 1) &= P(Y = 1|X = x)P(X = x) \\ &= \eta(x)\mu(x)\end{aligned}$$

and similarly

$$\begin{aligned}P(X = x, Y = 0) &= P(Y = 0|X = x)P(X = x) \\ &= (1 - \eta(x))\mu(x)\end{aligned}$$

So we can express the probability of any event (x, y) in terms of η and μ .

Regression function (context of classification) (4)

Formal proof for the general case:

... see the book of Devroye, Györfi, Lugosi, first pages.

Explicit form of the Bayes classifier

Consider the 0-1-loss function. Recall:

- the risk of a classifier under the 0-1-loss counts “how often” the classifier fails, that is

$$R(f) = E(\ell(X, Y, f(X))) = E(\mathbb{1}_{f(X) \neq Y}) = P(f(X) \neq Y).$$

- The Bayes classifier f^* was defined as the classifier that minimizes the true risk. This is an implicit definition, we don't yet have a formula for it.

Now consider the following classifier:

$$f^\circ(x) := \begin{cases} 1 & \text{if } \eta(x) \geq 1/2 \\ 0 & \text{otherwise} \end{cases}$$

Explicit form of the Bayes classifier (2)

Theorem 2 (f° is the Bayes classifier)

Consider classification with 0-1-loss. Let $f : \mathcal{X} \rightarrow \{0, 1\}$ be any (measurable) classifier and f° the classifier defined above. Then $R(f) \geq R(f^\circ)$.

Before we prove it, digest what this means:

- ▶ The theorem shows that $f^\circ = f^*$ (WHY?)
- ▶ Consequence: in the particular case of classification with the 0-1-loss, we have an explicit formula for the Bayes classifier.
- ▶ In practice, this doesn't help, WHY?

Explicit form of the Bayes classifier (3)

Proof of Theorem 2:

Step 1: Consider any fixed classifier $f : \mathcal{X} \rightarrow \{0, 1\}$ and compute its error probability at some fixed point x :

$$\begin{aligned} P(f(x) \neq Y \mid X = x) &= 1 - P(f(x) = Y \mid X = x) \\ &= 1 - P(f(x) = 1, Y = 1 \mid X = x) - P(f(x) = 0, Y = 0 \mid X = x) \\ &\stackrel{(*)}{=} 1 - \mathbb{1}_{f(x)=1} P(Y = 1 \mid X = x) - \mathbb{1}_{f(x)=0} P(Y = 0 \mid X = x) \\ &= 1 - \mathbb{1}_{f(x)=1} \eta(x) - \mathbb{1}_{f(x)=0} (1 - \eta(x)) \end{aligned}$$

For step (*), observe that $f(x)$ is a deterministic function.

Explicit form of the Bayes classifier (4)

Step 2: Now compare the pointwise error of any particular classifier f to the one of f° :

$$\begin{aligned} P(f(X) \neq Y \mid X = x) - P(f^\circ(X) \neq Y \mid X = x) \\ &= \dots \text{ plug in the formula from last page and simplify } \dots \\ &= (2\eta(x) - 1)(\mathbb{1}_{f^\circ(x)=1} - \mathbb{1}_{f(x)=1}) \\ &\stackrel{(**)}{\geq} 0 \end{aligned}$$

To see the last step (**):

- ▶ if $f^\circ(x) = 1$, then $\eta(x) \geq 0.5$, so both terms ≥ 0 .
- ▶ if $f^\circ(x) = 0$, then $\eta(x) \leq 0.5$, so both terms ≤ 0 .

Explicit form of the Bayes classifier (5)

Step 3: We have seen that for all fixed values x , the probability of error satisfies

$$P(f(X) \neq Y \mid X = x) \geq P(f^\circ(X) \neq Y \mid X = x)$$

Because this holds for any individual value of x , it also holds in expectation over all x . This implies

$$R(f) \geq R(f^\circ).$$



Explicit form of the Bayes classifier (6)

Remarks:

- ▶ If we work with 0-1-loss and if we know the underlying probability distribution and hence the regression function, then we don't need to "learn", we can simply write down what the optimal classifier is.
- ▶ For many other loss functions one can also explicitly compute the optimal classifier. We will see one more example in a minute: regression with squared loss.
- ▶ Problem in practice: we don't know the regression function.

Plug-in classifier

Simple idea: If we don't know the underlying distribution, but are given some training data, simply estimate the regression function $\eta(x)$ by some quantity $\eta_n(x)$ and build the plugin-classifier

$$f_n := \begin{cases} 1 & \text{if } \eta_n(x) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

- ▶ In theory: It can be shown that if we use a suitable estimator η_n , the plugin-approach is universally consistent. That is, in the limit of infinitely many training points, the classifier is going to converge to the best one out there. 😊
- ▶ In practice: Estimating densities is notoriously hard, in particular for high-dimensional input spaces. We would need a ridiculous amount of training data. So unfortunately, the plugin-approach is useless for practice. ☹

... for regression under L_2 loss

Loss functions for regression

While in classification, there is a “natural loss function” (the 0-1-loss), there exist many loss functions for regression and it is not so obvious which one is the most useful one.

In the following, let's look at the classic case, the squared loss function:

Squared loss (L_2 -loss): $\ell(x, y, f(x)) = (f(x) - y)^2$

Regression function (for L_2 regression)

As in the classification setting, we define the regression function:

$$\eta(x) = E(Y \mid X = x)$$

We now want to show an explicit formula for the Bayes learner as well. As in the classification case, we fix a particular loss function, this time it is the squared loss.

We need one more intermediate result:

Regression function (for L_2 regression) (2)

Proposition 3 (Decomposition)

We always have

$$E(|f(X) - Y|^2) = E(|f(X) - \eta(X)|^2) + E(|\eta(X) - Y|^2).$$

Note: Getting a related inequality with \leq is trivial (by the triangle inequality), but the equality in this statement is not trivial.

Proof: just few lines, nearly identical to the proof of Proposition , but needs advanced treatment of conditional expectations, see also Györfi, Kohler, Krzyżak, Walk: Distribution-free theory for nonparametric regression, p.2. If you don't have the corresponding maths background, just ignore it.

Regression function (for L_2 regression) (3)

$$\begin{aligned}
 & E(|f(x) - \gamma|^2) \\
 &= E(|\underbrace{f(x) - \eta(x)} + \underbrace{\eta(x) - \gamma}|^2) \\
 &= E(|f(x) - \eta(x)|^2) + E(|\eta(x) - \gamma|^2) + 2E(\underbrace{(f(x) - \eta(x))(\eta(x) - \gamma)}_{\textcircled{\#} = 0}) \\
 &= E(|f(x) - \eta(x)|^2) + E(|\eta(x) - \gamma|^2)
 \end{aligned}$$

Regression function (for L_2 regression) (4)

$$\begin{aligned}
 \# &= E\left([f(x) - \eta(x)] [\eta(x) - \eta]\right) \stackrel{\textcircled{1}}{=} E(E(z)) \\
 &= E\left(E\left([f(x) - \eta(x)] [\eta(x) - \eta] \mid X\right)\right) \stackrel{\textcircled{2}}{=} E(g(x)g(\eta, X) \mid X) = g(x) \cdot E(g(\eta, X) \mid X) \\
 &= E\left([f(x) - \eta(x)]\right) \cdot E\left([\eta(x) - \eta] \mid X\right) = 0 \\
 &= \eta(x) - \underbrace{E(\eta \mid X)}_{=\eta(x)} \\
 &= 0
 \end{aligned}$$



Explicit form of optimal solution under L_2 loss

Define the following learning rule that predicts the real-valued output based on the regression function η :

$$f^\circ : \mathcal{X} \rightarrow \mathbb{R}, f^\circ(x) := \eta(x)$$

Theorem 4 (Explicit form of optimal L_2 -solution)

The function f° minimizes the L_2 -risk.

Proof. Follows directly from Proposition 3:

- ▶ Second expectation on the rhs does not depend on f .
- ▶ First expectation is always ≥ 0 , and it is $= 0$ for $f(X) = \eta(X)$.
- ▶ So the whole right hand side is minimized by $f(X) = \eta(X)$.



Basic learning principles: ERM, RRM

Two major principles

- ▶ Assume we operate in the standard setup, and are given a set of training points (X_i, Y_i) .
- ▶ Based on these points we want to “learn” a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that has as small true loss as possible.

There are two major approaches to supervised learning:

- ▶ Empirical risk minimization (ERM)
- ▶ Regularized risk minimization (RRM)

Empirical risk minimization

As we don't know P we cannot compute the true risk. But we can compute the **empirical risk** based on a sample $(X_i, Y_i)_{i=1, \dots, n}$

$$R_n(f) := \frac{1}{n} \sum_{i=1}^n \ell(X_i, Y_i, f(X_i))$$

The key point is that the empirical risk can be computed based on the training points only.

Empirical risk minimization (2)

Empirical risk minimization approach:

- ▶ Define a set \mathcal{F} of functions from $\mathcal{X} \rightarrow \mathcal{Y}$.
- ▶ Within these functions, choose one that has the smallest empirical risk:

$$f_n := \operatorname{argmin}_{f \in \mathcal{F}} R_n(f)$$

(might not be unique; for simplicity, let's assume the minimizer exists)

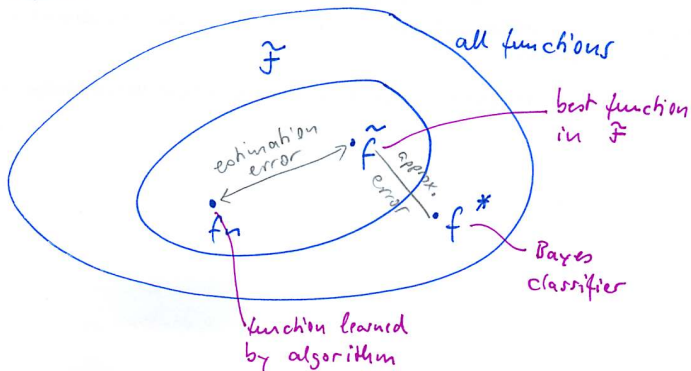
Estimation vs approximation error

With this approach, we can make two types of error:

- ▶ Denote by \tilde{f} the true best function in the set \mathcal{F} , that is $\tilde{f} = \operatorname{argmin}_{f \in \mathcal{F}} R(f)$.
- ▶ The quantity $R(f_n) - R(\tilde{f})$ is called the **estimation error**. It is a random variable that depends on the random sample.
- ▶ The quantity $R(\tilde{f}) - R(f^*)$ is called the **approximation error**. It is a deterministic quantity that does not depend on the sample, but on the choice of the space \mathcal{F} .

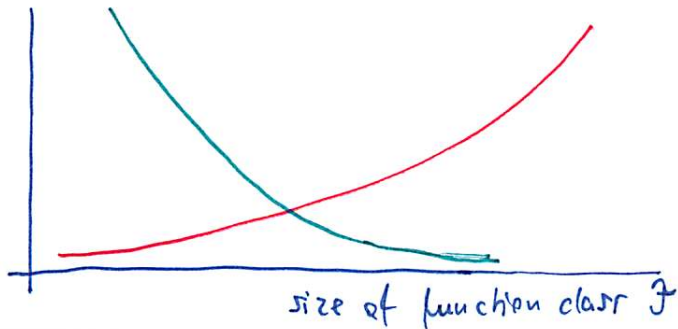
Estimation vs approximation error (classic regime)

Classic regime: the function class \mathcal{F} is reasonably small.



Estimation vs approximation error (classic regime) (2)

In the following sketch, one curve shows the approximation error, one the estimation error.



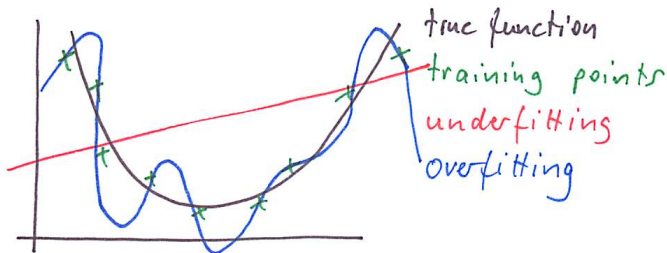
WHICH ONE IS WHICH? HOW WOULD THE CURVE OF THE TRUE RISK OF THE CLASSIFIER LOOK LIKE?

Overfitting, underfitting (classic regime)

Choosing a “reasonable function class \mathcal{F} ” is crucial.

Consider the following example:

- ▶ True function f : quadratic function
- ▶ Training points: $Y_i = f(X_i) + \text{noise}$
- ▶ Red curve: $\mathcal{F} = \text{all linear functions}$
- ▶ Blue curve: $\mathcal{F} = \text{all polynomial functions}$



Overfitting, underfitting (classic regime) (2)

Overfitting:

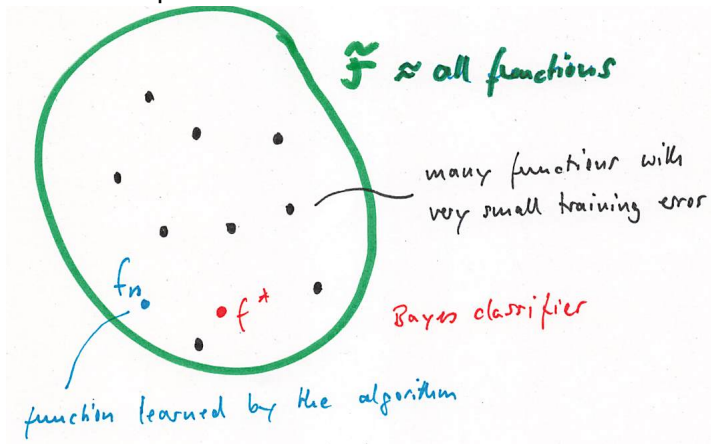
- ▶ We can always find a function that explains all training points very well or even exactly: \mathcal{F} is very large.
- ▶ But such a function tends to be very complicated and models the noise as well
- ▶ Predictions for unseen data points are poor (“large test error”)
- ▶ Low approximation error, high estimation error.

Underfitting:

- ▶ Model is too simplistic: \mathcal{F} is too small.
- ▶ But estimated functions are stable with respect to noise
- ▶ Large approximation error, low estimation error.

Overfitting, underfitting (modern regime)

Modern regime: the function class \mathcal{F} is huge and pretty much covers the whole space.



Overfitting, underfitting (modern regime) (2)

Many function with training error close to 0. Most of them would NOT generalize, some of them might generalize. The question is just how to find those.

Overfitting, underfitting (modern regime) (3)

In the modern regime, the approximation error is often close to 0.

Estimation error: traditional approaches would predict that it is huge, due to overfitting. But:

- ▶ Some algorithms often end up with a classifier that generalizes well (among all the many local and global optima...). Solutions often have a small estimation error.
- ▶ Overfitting can be beneficial (highly overparameterized regime as in deep learning).

Overfitting, underfitting (modern regime) (4)

The modern version of the approximation/estimation error figure is this one, describing the double descent phenomenon:

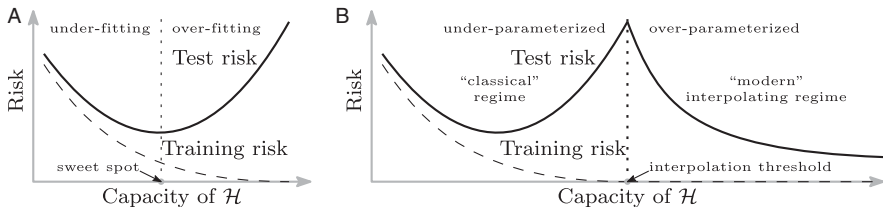


Fig. 1. Curves for training risk (dashed line) and test risk (solid line). (A) The classical U-shaped risk curve arising from the bias-variance trade-off. (B) The double-descent risk curve, which incorporates the U-shaped risk curve (i.e., the “classical” regime) together with the observed behavior from using high-capacity function classes (i.e., the “modern” interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Figure from Belkin et al, 2019, PNAS

See later.

Bias-Variance tradeoff in L_2 -regression

Sometimes another decomposition of the errors is used. Can be seen most easily for the case of regression with L_2 loss:

Let f_n be the function constructed by an algorithm on n points, and $f^* : \mathbb{R}^d \rightarrow \mathbb{R}$ the true best function (the regression function). Then we can decompose the pointwise expected L_2 risk in two terms:

$$\begin{aligned} E(|f_n(x) - f^*(x)|^2) \\ = \underbrace{E\left(\left(f_n(x) - E(f_n(x))\right)^2\right)}_{\text{Variance term}} + \underbrace{\left(E(f_n(x)) - f^*(x)\right)^2}_{\text{Bias term}} \end{aligned}$$

- Variance term = the variance of the random variable $f_n(x)$.
- Bias term: measures how much $E(f_n)$ and f^* deviate.

Bias-Variance tradeoff in L_2 -regression (2)

Note: we always have \leq (for any loss function), but for the L_2 -loss we get equality (as we have seen in Proposition 3).

Proof. skipped, nearly the same as the proof of Proposition 3 (see Györfi, Kohler, Krzyżak, Walk: Distribution-free theory for nonparametric regression, p.24)

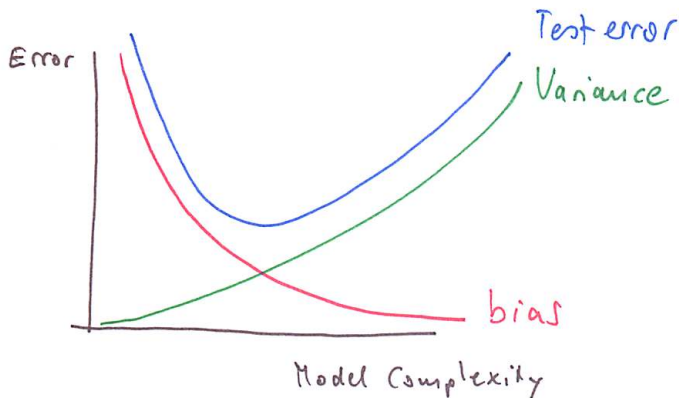
Bias-Variance tradeoff in L_2 -regression (3)

Intuition:

- ▶ Variance term: same intuition as estimation error, depends on random data and the capacity of the function class \mathcal{F} .
- ▶ The bias term: same intuition as the approximation error. Does not depend on the data, just on the capacity of the function class \mathcal{F} .

Bias-Variance tradeoff in L_2 -regression (4)

The classic version of this figure looks like this:



ERM, remarks

- ▶ From a conceptual/theoretical side, ERM is a straight forward learning principle.
- ▶ The key to the success / failure of ERM is to choose a “good” function class \mathcal{F}
- ▶ From the computational side, it is not always easy (depending on function class and loss function, the problem can be quite challenging: finding the minimizer of the 0-1-loss is often NP hard.) This is why in practice we use convex relaxations of the 0-1-loss function, see later.

Regularized risk minimization

Crucial problem in ERM: choose \mathcal{F}

Alternative approach:

- ▶ Let \mathcal{F} be a very large space of functions.
- ▶ Define a **regularizer** $\Omega : \mathcal{F} \rightarrow \mathbb{R}_{\geq 0}$ that measures how “complex” a function is. Examples:
 - ▶ \mathcal{F} = polynomials, $\Omega(f)$ = degree of the polynomial f
 - ▶ \mathcal{F} = differentiable functions, $\Omega(f)$ = maximal slope
- ▶ Define the **regularized risk**

$$R_{reg,n}(f) := R_n(f) + \lambda \cdot \Omega(f)$$

Here $\lambda > 0$ is called **regularization constant**.

- ▶ Then choose $f \in \mathcal{F}$ to minimize the regularized risk.

Regularized risk minimization (2)

Intuition:

- ▶ If we can fit the data reasonably well with a “simple function”, then choose such a simple function.
- ▶ If all simple functions lead to a very high empirical risk, then better choose a more complex function.

Regularized risk minimization (3)

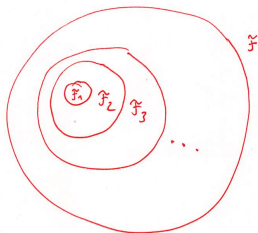
WHAT HAPPENS IF λ IS VERY SMALL? VERY LARGE? Relation to overfitting / underfitting, approximation / estimation error, relationship to ERM?

Regularized risk minimization (4)

Intuition: consider nested function spaces \mathcal{F}_t :

$$\mathcal{F}_t := \{f \in \mathcal{F} \mid \Omega(f) \leq t\}$$

- λ small \leadsto solution in \mathcal{F}_t with t large
- λ large \leadsto solution in \mathcal{F}_t with t small



Linear methods for supervised learning

Linear methods for regression

Linear least squares regression

Literature:

- ▶ Hastie/Tibshirani/Friedman Section 3
- ▶ Bishop Sec 3

Linear setup

- ▶ Assume we have training data (X_i, Y_i) with $X_i \in \mathcal{X} := \mathbb{R}^d$ and $Y_i \in \mathcal{Y} := \mathbb{R}$.
- ▶ We want to find the “best” *linear function*, that is a function of the form

$$f(x) = \sum_{i=1}^d w_i x^{(i)} + b$$

where $x = (x^{(1)}, \dots, x^{(d)})^t \in \mathbb{R}^d$.

The w_i are called “weights” and b the “offset” or “intercept” or “threshold”.

Linear setup (2)

As loss function, we want to use the squared loss (L_2 loss).

Formally, the linear least squares problem is the following:

(#) Find parameters $w_1, \dots, w_d \in \mathbb{R}$ and $b \in \mathbb{R}$ such that the empirical least squares error of the linear function f (as defined on the last slide) is minimal:

$$\frac{1}{n} \sum_{i=1}^n \left(Y_i - f(X_i) \right)^2$$

Example

Want to predict the shoe size of a person, based on many input values:

- ▶ For each person X , we have a couple of real-valued measurements: $X^{(1)} = \text{height}$, $X^{(2)} = \text{weight}$, $X^{(3)} = \text{income}$, $X^{(4)} = \text{age}$.
(Note: some measurements are useful for the question, some might not be useful)
- ▶ In this example, we might find that the following function is good for predicting the shoe size:

$$\text{shoesize} = \frac{2}{10} \text{height} + 0 \cdot \text{weight} + 0 \cdot \text{income} + 0 \cdot \text{age} + 1$$

Concise notation

- To write everything in a more concise form, we stack the training inputs into a big matrix (each point is one row) and the output in a big vector:

$$X = \left(\begin{array}{ccc} x_{11} & \dots & x_{1d} \\ \vdots & & \\ x_{n1} & \dots & x_{nd} \end{array} \right) \left. \vphantom{\begin{array}{ccc} x_{11} & \dots & x_{1d} \\ \vdots & & \\ x_{n1} & \dots & x_{nd} \end{array}} \right\}^d_n \quad y = \left(\begin{array}{c} y_1 \\ \vdots \\ y_n \end{array} \right) \left. \vphantom{\begin{array}{c} y_1 \\ \vdots \\ y_n \end{array}} \right\}^n_n \quad w = \left(\begin{array}{c} w_1 \\ \vdots \\ w_d \end{array} \right) \left. \vphantom{\begin{array}{c} w_1 \\ \vdots \\ w_d \end{array}} \right\}^d_d$$

- Notation: the i -th training point consists of the vector $X_i \in \mathbb{R}^d$, its entries are denoted as X_{i1}, \dots, X_{id} .
- Now we can write:

$$f(X_i) = \langle X_i, w \rangle + b = (Xw)_i + b$$

Concise notation (2)

- Formally, the linear least squares problem is the following:

(##) Determine $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ as to minimize the empirical least squares error

$$\frac{1}{n} \sum_{i=1}^n \left(Y_i - ((Xw)_i + b) \right)^2$$

Getting rid of b

We want to write the problem even more concisely.

- Define the matrices

$$\tilde{X} = \left(\begin{array}{ccc} x_{11} & \dots & x_{1d} & 1 \\ \vdots & & \vdots & \vdots \\ x_{n1} & \dots & x_{nd} & 1 \end{array} \right) \left. \vphantom{\begin{array}{ccc} x_{11} & \dots & x_{1d} \\ \vdots & & \vdots \\ x_{n1} & \dots & x_{nd} \end{array}} \right\}^n \quad \tilde{w} = \left(\begin{array}{c} w_1 \\ \vdots \\ w_d \\ b \end{array} \right) \left. \vphantom{\begin{array}{c} w_1 \\ \vdots \\ w_d \end{array}} \right\}^{d+1}$$

- Then we have

$$(\tilde{X}\tilde{w})_i = \sum_{k=1}^{d+1} \tilde{X}_{ik}\tilde{w}_k = \sum_{k=1}^d X_{ik}w_k + b = (Xw)_i + b$$

Getting rid of b (2)

- Hence, there is a unique correspondence between the original problem and the following new problem:

(###) determine $\tilde{w} \in \mathbb{R}^{d+1}$ as to minimize the empirical least squares error

$$\frac{1}{n} \sum_{i=1}^n \left(Y_i - (\tilde{X} \tilde{w})_i \right)^2 = \frac{1}{n} \|Y - \tilde{X} \tilde{w}\|^2$$

Getting rid of b (3)

Without loss of generality, from now on we consider the simplified problem that does not involve the intercept b . We also remove the twiddles on the letters \tilde{X} and \tilde{w} to make notation simpler. We still call the resulting problem (###).

(###) Determine $w \in \mathbb{R}^d$ as to minimize the empirical least squares error

$$\frac{1}{n} \sum_{i=1}^n \left(Y_i - (Xw)_i \right)^2 = \frac{1}{n} \|Y - Xw\|^2$$

In the following, we sometimes consider different factors in front of the norm (for example, we might drop the $1/n$, and include a factor $1/2$ for mathematical convenience). It doesn't change the solution, but the formulas then look nicer.

ML \leadsto Optimization problem

We can see:

- ▶ In order to solve (###), we need to solve an optimization problem
- ▶ In this particular case, we will see in a minute that we can solve it analytically.
- ▶ For most other ML algorithms, we need to use optimization algorithms to achieve this.

Least squares regression is convex

Recap:

- Convex optimization problem (maths appendix, page 1387)

Proposition 5 (Least squares is convex)

The least squares optimization problem (###) is a convex optimization problem.

Proof. Exercise

Solution, case of full rank

Recap:

- ▶ Inverse of a matrix
- ▶ Rank of a matrix

Solution, case of full rank (2)

Theorem 6 (Solution, case $\text{rank}(X) = d$)

Assume that X has rank d . Then the solution w of linear least squares regression (###) is given by $w = (X^t X)^{-1} X^t Y$.

Proof intuition.

- ▶ Want to find the minimum of the function $\|Y - Xw\|^2$
- ▶ Take the derivative and set it to 0.
- ▶ Then we have to check that what we get is indeed a minimum (in a 1-dimensional situation we would look at the second derivative for this).
- ▶ The minimum then has to be a global minimum because the objective function is convex.

We can either do all this by foot, coordinate-wise. Or we do it more elegantly as follows:

Solution, case of full rank (3)

Proof, formally. We write all equations in matrix form:

- ▶ Objective function: $Obj : \mathbb{R}^d \rightarrow \mathbb{R}$, $Obj(w) := \frac{1}{2} \|Y - Xw\|^2$
- ▶ Derivative: the gradient is a vector in \mathbb{R}^d consisting of all partial derivatives, it is given as
 $grad(Obj)(w) = -X^t(Y - Xw)$.
(To see this, either use matrix derivatives or compute all the partial derivatives by foot, see slide below.)
- ▶ Setting the gradient to zero gives the necessary condition:
 $X^tY = (X^tX)w$. Ideally, we would like to solve this for w .
- ▶ We always have $rank(X) = rank(X^tX) = rank(XX^t)$. In particular, under the assumption that X has rank d , the matrix X^tX is of full rank, hence invertible.
- ▶ So we can solve for w by $w = (X^tX)^{-1}X^tY$.

Solution, case of full rank (4)

- ▶ Now we need to figure out whether this is indeed a minimum. To this end, consider the Hessian matrix that contains all second derivatives: $H(Obj) = \frac{\partial^2 Obj}{\partial w \partial w'} = X^t X$.
- ▶ This matrix is positive semi-definite: obviously all eigenvalues ≥ 0 ,
(WHY ???)
and because of the rank condition we have > 0 . So the solution we computed above is indeed a local minimum.



Solution, case of full rank (5)

Side remark, here is the “derivative by foot”: To see that the gradient is indeed given by the expression on the previous slide, we again compute it, this time coordinate-wise:

- ▶ Objective function, written explicitly : $Obj : \mathbb{R}^d \rightarrow \mathbb{R}$,
$$Obj(w) = \frac{1}{2} \|Y - Xw\|^2 = \frac{1}{2} \sum_{i=1}^n \left(Y_i - \sum_k w_k X_i^{(k)} \right)^2.$$
- ▶ Take partial derivatives, for each w_k separately (attention, the variables of this function are the w_k , not the X_i)
$$\frac{\partial Obj(w)}{\partial w_k} = \sum_{i=1}^n \left(Y_i - \sum_k w_k X_i^{(k)} \right)^2 \left(-X_i^{(k)} \right)$$
- ▶ Now finally note that the right hand side agrees with the k -th coordinate of the vector $X^t(Y - Xw)$.

Solution, general case

Recap:

- ▶ Generalized inverse of a matrix (see maths appendix page 1369)

Solution, general case (2)

Theorem 7 (Solution, case $\text{rank}(X) < d$)

Assume that X has $\text{rank} < d$.

1. Then a solution w of linear least squares regression (###) is given by $w = (X^t X)^+ X^t Y$, where X^+ denotes the generalized inverse of the matrix X .
2. This solution is not unique. But even if w_1, w_2 are two different solutions, then their predictions agree on the training data, that is $\langle w_1, X_i \rangle = \langle w_2, X_i \rangle$ for all $i = 1, \dots, n$.

Proof (sketch).

- ▶ As above we get the necessary condition $X^t Y = (X^t X)w$.
- ▶ One can check that one particular vector w that satisfies this condition is given as $w = (X^t X)^+ X^t Y$ (EXERCISE!)

Solution, general case (3)

- ▶ So $w = (X^t X)^+ X^t Y$ is one solution to the problem, which proves part 1 of the theorem.
- ▶ However, w is not unique:
 - ▶ Let w be a solution and v any vector with $Xv = 0$ (exists because X has rank $< d$).
 - ▶ Then $w + v$ is a solution as well (EXERCISE: CHECK IT BY PLUGGING IT IN THE NECESSARY CONDITION).
 - ▶ So our problem has many solution vectors. Note that all of them lead to the same objective value.
- ▶ Moreover observe: all solutions give the same results on the training points:

Solution, general case (4)

- ▶ Let w_1 be a solution, and $w_2 = w_1 + v$ a solution as well. Then the vector of all predictions on the training points looks as follows:

$$\underbrace{Xw_2}_{\text{predictions by } w_2} = X(w_1 + v) = Xw_1 + Xv = \underbrace{Xw_1}_{\text{predictions by } w_1}$$

So all solutions to the problem predict the same values on the training points.

- ▶ But on the test points, the solutions will disagree. The question is then which one to prefer. One idea here is to use regularization, see below.



Relationship between n and d

n = number of points, d = dimension of the space.

WHAT DO YOU THINK IS EASIER IN A MACHINE LEARNING CONTEXT?

- ▶ n high, d low
- ▶ d high, n low
- ▶ $n \approx d$

??????????

Relationship between n and d (2)

Formally:

- ▶ the linear system we solve for least squares regression is $X^t Y = (X^t X) w$. It has d equations and d unknowns (independently of the value of n).
- ▶ So either there exists a unique solution (case $\text{rank}(X^t X) = d$) or many solutions (case $\text{rank} < d$).
- ▶ Note that the system always has a solution, no matter how large n is.

Intuition: we just want to find the best linear function, we don't require that it goes through the data points exactly.

Relationship between n and d (3)

Informally, we interpret d as the number of parameters and n as the number of constraints.

Case $d \ll n$: harmless

- ▶ This is the harmless case: we have many points in a low-dimensional space. Here linear functions are not very flexible, and we tend to not overfit (sometimes we underfit).
- ▶ This is the case that has been treated in traditional statistics since a century.

Relationship between n and d (4)

Case $d \gg n$: not quite as harmless

- Often, it is a bad idea to have much more parameters than constraints because it leads to overfitting.

Geometric reason: if we have few points (n) in a very high-dimensional space (d), then linear functions are very powerful. In machine learning terms: the size of the function class is too large.

- However, recently it has been understood that sometimes “benign overfitting” can take place, particularly in linear regression. Here $d \gg n$, but due to the inductive bias of the learning algorithm it doesn't hurt. See later in this lecture.

Relationship between n and d (5)

- Many other surprising things can happen in case $d \gg n$. See results in the field “High-dimensional statistics” (see textbooks by Martin Wainwright “High-dimensional statistics”, or Bühlmann and van der Geer “Statistics for High-Dimensional Data”.)

Summary: Linear least squares regression

- ▶ Regression problem, $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}$
- ▶ Loss function: L_2 -loss
- ▶ Function class \mathcal{F} : set of all linear functions over \mathcal{X}
- ▶ No regularization.
- ▶ Finding the linear function that minimizes the empirical L_2 -loss is a convex optimization problem, and we can compute its solution analytically.

Feature representation of data

Feature representation of data

On the first glance, the assumption that the data points are in \mathbb{R}^d looks pretty restrictive. What if our data is not “numbers”?

It turns out that in many cases it is a good idea to represent “objects” by “feature vectors”.

Feature representation of data (2)

Example: bag of words representation for texts

- ▶ Make a list of all words occurring in the text
- ▶ Throw away all words that are too common (“the”, “a”, “for”, “you”, ...)
- ▶ Use “stemming” to throw away word endings (like the plural “s”): we want to consider the word “horse” the same as “horses”)
- ▶ For each text, count how often each word occurs
- ▶ The represent each text as a vector: each dimension corresponds to one word, and the entry of the vector is how often this word occurs in the given text.

Feature representation of data (3)

T_1 : I like to play soccer.

T_2 : My soccer shoes are red.

T_3 : We moved to a new house.

	T_1	T_2	T_3
like	1	0	0
play	1	0	0
soccer	1	1	0
shoe	0	1	0
red	0	1	0
move	0	0	1
house	0	0	1

Feature representation of data (4)

Example: strings in a feature representation

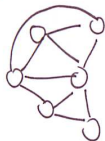
- ▶ Given a string
- ▶ Represent it by counting substrings (can also allow substrings with “gaps” in between)

Feature representation of data (5)

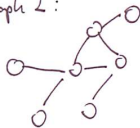
Example: motif representation of graphs (such as chemical molecules)

Count the occurrence of certain subgraphs (called motifs):

graph 1:



graph 2:



Motifs:

in graph 1

in graph 2



10

7



5

1



1

1



0

0

⋮

Feature representation of data (6)

Example: books and/or users in amazon.

- ▶ can describe a book by how often it was bought by each user
- ▶ or by how often it was bought together with each other book.

Representation of books by user data:

	user1	user2	user3
book 1	0	0	1
book 2	1	0	0
book 3	0	0	0
⋮	1	1	0
	0	0	0

how often did user 1 buy book 4

Representation of books by books

	book 1	book 2	book 3	...
book 1	2	1	3	
book 2	1	5	2	
book 3	3	2	3	
⋮				

how often was book 3 bought together with book 2

Feature representation of data (7)

Example: images

- Can obviously represent images as vectors of greyscale values, or RGB values or CYMK values ...

Feature representation of data (8)

General procedure that works very often:

- ▶ Given a set of “objects” (texts, graphs, images, emails, ...)
- ▶ Describe the objects by simple “features” that can be expressed as numbers
- ▶ Together, these objects give a feature vector $\in \mathbb{R}^d$.
- ▶ Note that often, the dimension d ends up very large! The incentive is: give as much information as possible to the learning algorithm, and hope that it is going to identify / extract the information that is helpful for classification.

Feature representation of data (9)

In machine learning, the mapping $\Phi : \mathcal{X} \rightarrow \mathbb{R}^d$ that takes an abstract object X to its feature representation is called the **feature map**. It is usually denoted by Φ .

All in all, the assumption that “data is in \mathbb{R}^d ” does make sense in very many applications.

Feature normalization

In practice: normalization

In regularized regression, it makes a difference how we scale our data. Example:

- Body height measured in mm or cm or even km

Different scales lead to different solutions, because they affect the regularization in a different way (WHY???)

Moreover, we typically want all coordinates to have “the same amount of influence” on the solution. This is not the case if our measurements have completely different orders of magnitude (for example, one coordinate is “body height in mm” and one is “shoe size”).

In practice: normalization (2)

In order to make sure that all basis functions “are treated the same” it is thus recommended to **standardize** your data:

1. Centering:

Replace Φ_i by $\Phi_i^{\text{centered}} := \Phi_i - \bar{\Phi}_i$ with $\bar{\Phi}_i := \frac{1}{n} \sum_{j=1}^n \Phi_i(X_j)$.

2. Normalizing the variance: rescale each basis function such that it has unit L_2 -norm (variance) on the training data:

$$\Phi_i^{\text{rescaled}} := \frac{\Phi_i^{\text{centered}}}{(\sum_{j=1}^n \Phi_i^{\text{centered}}(X_j)^2)^{1/2}}$$

In terms of the matrix Φ : you center and normalize **the columns** of the matrix to have center 0 and unit norm.

Least squares with linear combination of basis functions

Literature:

- ▶ Hastie/Tibshirani/Friedman Section 3
- ▶ Bishop Section 3

Using non-linear basis functions

Idea:

- ▶ Linear functions are often quite restrictive.
- ▶ Instead, want to learn a function of the form

$$f(x) = \sum_{i=1}^D w_i \Phi_i(x)$$

where the functions Φ_1, \dots, Φ_D are arbitrary “basis functions”.

- ▶ Note: f is linear in the parameters w , but if the functions Φ are non-linear in x , then so is f .

Examples

Example 1:

- Your data lives in \mathbb{R}^d , but it clearly cannot be described by a linear function of the original coordinates. Alternatively, you can fit a function of the form

$$f(x) = \sum_{i=1}^D w_i \Phi_i(x)$$

(where the number D of basis functions does not need to coincide with the original dimension d)

Examples (2)

- For example, if we want to learn a periodic function, the Φ_i might be the first couple of Fourier functions to fit a function of the form

$$g(x) = \sum_{k=1}^D w_k \sin(kx)$$

- In some other case, we might want to choose the basis functions Φ_i as polynomials x, x^2, \dots, x^D to fit a function of the form

$$h(x) = \sum_{k=1}^D w_k x^k$$

In this way you can use a linear algorithm (find the linear coefficients w_i) to fit non-linear functions (such as $g(x)$ or $h(x)$) to your data.

Examples (3)

Example 2: Feature spaces

the input X consists of a web page, the task is to predict how many seconds users stay on the page before they leave it again.

We might consider basis functions as the following ones:

- ▶ Φ_1 counts the number of occurrences of the word “soccer”
- ▶ Φ_2 counts the number of occurrences of the word “team”
- ▶ Φ_3 tells you how many words the text has in total
- ▶ Φ_4 counts the number of images on the page
- ▶ ... etc ...

How to solve it

It is easy to rewrite the “standard” least squares problem in this more general framework:

- Define the design matrix as follows:

$$\Phi = \begin{pmatrix} \phi_1(x_1) & \dots & \phi_D(x_1) \\ \vdots & & \\ \phi_1(x_n) & \dots & \phi_D(x_n) \end{pmatrix}$$

- Then the least squares problem is to find w as to minimize $\|Y - \Phi w\|^2$.
- This has the solution $w = (\Phi^t \Phi)^{-1} \Phi^t Y$ (with exact inverse or generalized inverse) as we have seen above.

Advantages and disadvantages

- ▶ Note that in the given scenario, we chose the basis functions before we got to see the data points.
- ▶ If we have prior domain knowledge about our data, we can select a “good” set of basis functions.
- ▶ (Side remark: there also exist approaches where we select particular basis functions AFTER we have seen our training data. This field is called dictionary learning; we won't cover it in this lecture).
- ▶ To avoid overfitting, the dimension D of the feature space should be small; To avoid underfitting, the dimension D should be large. To tackle both, a good choice of D is necessary.

Advantages and disadvantages (2)

- ▶ Note that achieving a small D can be quite restrictive. Just consider the case where our data is d -dimensional and we want to have a function space with polynomials of degree two. There are already of the order d^2 many basis polynomials of degree two ($x_i x_j$ for $i, j = 1, \dots, d$), so $D \approx d^2$.
- ▶ There is one way out of this trap, namely to regularize, in particular by enforcing sparsity. See Lasso below.

Ridge regression: least squares with L_2 -regularization

Literature: Hastie/Tibshirani/Friedman Section 3.4.3; Bishop
Section 3

Idea

Want to improve standard L_2 -regression. Two points of view:

1. Want to have a unique solution, no matter what the rank of the design matrix is. This is going to improve numerical stability.
2. In the standard problem, the coefficients w_i can become very large. This leads to a high variance of the results.

To avoid this effect, we want to introduce regularization to force the coefficients to stay “small”.

Ridge regression problem

Consider the following regularization problem:

- ▶ Input space \mathcal{X} arbitrary, output space $\mathcal{Y} = \mathbb{R}$.
- ▶ Fix a set of basis functions $\Phi_1, \dots, \Phi_D : \mathcal{X} \rightarrow \mathbb{R}$
- ▶ As function space choose all functions of the form $f(x) = \sum_i w_i \Phi_i(x)$.
- ▶ As regularizer use $\Omega(f) := \|w\|^2 = \sum_{i=1}^D w_i^2$. Choose a regularization constant $\lambda > 0$.
- ▶ Then solve the problem

$$w_{n,\lambda} := \operatorname{argmin}_{w \in \mathbb{R}^D} \frac{1}{n} \|Y - \Phi w\|^2 + \lambda \|w\|^2.$$

Solution

Theorem 8 (Solution of Ridge Regression)

The coefficients $w_{n,\lambda}$ that solve the ridge regression problem are given as

$$w_{n,\lambda} := \left(\Phi^t \Phi + n\lambda I_D \right)^{-1} \Phi^t Y$$

where I_D is the $D \times D$ identity matrix.

Solution (2)

Proof.

- ▶ Objective function is $Obj(w) := \frac{1}{n}\|Y - \Phi w\|^2 + \lambda\|w\|^2$.
- ▶ Note that this function is convex.
- ▶ Take the derivative with respect to w and set it to 0:

$$\begin{aligned} grad(Obj)(w) &= -\frac{2}{n}\Phi^t(Y - \Phi w) + 2\lambda w \stackrel{!}{=} 0 \\ \implies (\Phi^t\Phi + n\lambda I_D)w_{n,\lambda} &= \Phi^t Y \end{aligned}$$

- ▶ It is straight forward to see that the matrix $(\Phi^t\Phi + n\lambda I_D)$ has full rank whenever $\lambda > 0$ (see next slide). So we can take the inverse, and the theorem follows as in the standard L_2 -regression case.

Solution (3)

Before we continue with the proof, recap:

- ▶ WHAT DO YOU KNOW ABOUT THE EIGENVALUES OF A SYMMETRIC MATRIX?
- ▶ WHAT DO YOU KNOW ABOUT THE EIGENVALUES OF A MATRIX OF THE FORM $A = ZZ'$ WHERE Z IS ANY REAL-VALUED MATRIX?

This will occur millions of times in this lecture, please remember it!

Solution (4)

Proof that $(\Phi^t\Phi + n\lambda I_D)$ is invertible:

- ▶ The matrix $A := \Phi^t\Phi$ is symmetric, hence we can decompose it into eigenvalues: $A = V^t\Lambda V$ where Λ is the diagonal matrix with all eigenvalues of A
- ▶ Because of the special form $A := \Phi^t\Phi$, all eigenvalues are ≥ 0 (the matrix is positive semi-definite).
- ▶ A has full rank (is invertible) iff all its eigenvalues are > 0 .
- ▶ σ is an eigenvalue of A with eigenvector $v \iff \sigma + \lambda$ is an eigenvalue of $A + \lambda I$. Reason:

$$(A + \lambda I)v = Av + \lambda v = \sigma v + \lambda v = (\sigma + \lambda)v$$

- ▶ If $\lambda > 0$, then all eigenvalues of $A + \lambda I$ are > 0 :
 $\sigma \geq 0$ and $\lambda > 0$ implies $\sigma + \lambda > 0$
- ▶ So we know that $A + \lambda I$ has full rank and is invertible.



Example (by Matthias Hein)

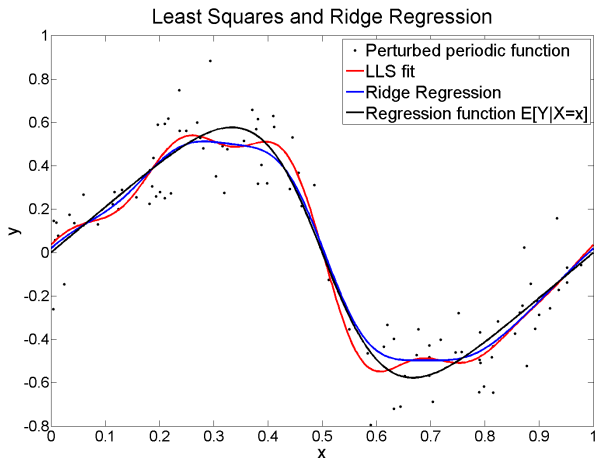
- ▶ True function: periodic function + noise
- ▶ Basis functions Φ : first 10 Fourier basis functions

$$x \mapsto \sin(kx), \quad (k = 1, \dots, 10)$$

So we want to determine the coefficients w_i for a function of the form

$$f(x) = \sum_{k=1}^{10} w_k \sin(kx)$$

Example (by Matthias Hein) (2)



Choice of the parameter λ

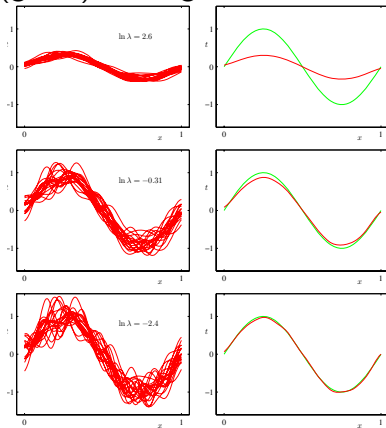
QUESTION: WHAT IS THE ROLE OF λ ? WHAT HAPPENS TO ESTIMATION AND APPROXIMATION ERROR IF IT IS HIGH / LOW?

Choice of the parameter λ (2)

Example (from Bishop's book):

Left: results for decreasing amount of regularization

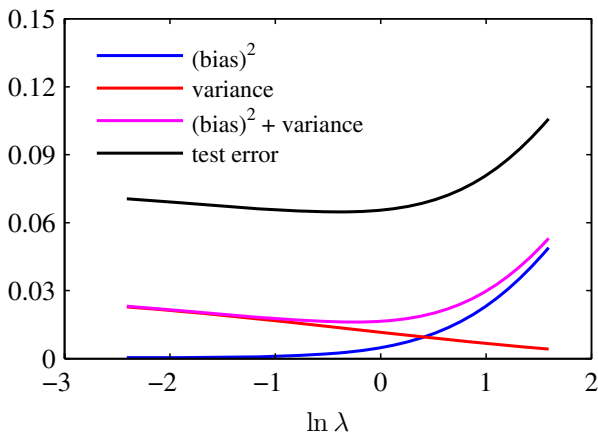
Right: True curve (green), average estimated curve (red)



Choice of the parameter λ (3)

Same example, bias-variance decomposition:

Larger regularization constant λ leads to less complex functions:



(*) Ridge regression as shrinkage method

Geometric interpretation of regularization via SVD:

- Consider the Singular Value Decomposition of the matrix $\Phi \in \mathbb{R}^{n \times d}$:

$$\Phi = U \Sigma V^t$$

- Plugging this into the formula for $w_{n,\lambda}$ leads to

$$w_{n,\lambda} = \dots = V \operatorname{diag} \left(\frac{\sigma_j}{\sigma_j^2 + \lambda} \right) U^t Y$$

- Standard least squares regression (without regularization) corresponds to $\lambda = 0$, and we have

$$\frac{\sigma_j}{\sigma_j^2 + \lambda} = \frac{1}{\sigma_j}$$

(*) Ridge regression as shrinkage method (2)

► Regularized case:

- Case σ_i large: not much difference to non-regularized case:

$$\frac{\sigma_j}{\sigma_j^2 + \lambda} \approx \frac{1}{\sigma_j}$$

- Case σ_i small: here it makes a lot of difference whether we have σ_i^2 or $\sigma_i^2 + \lambda$ in the denominator. In particular,

$$\frac{\sigma_j}{\sigma_j^2 + \lambda} \ll \frac{1}{\sigma_j}$$

This means that the regularization “shrinks” the directions of small variance. Intuitively, these are the directions that mainly contain noise, no signal.

(*) Ridge regression as shrinkage method (3)

In statistics, related methods are often called “shrinkage methods” (because we try to “shrink” the weights w_i).

From a statistics point of view, they can be justified by what is called “Stein’s paradox” (discovered in the 1950ies). Essentially, this paradox says that if we want to estimate at least three parameters jointly, then it is better to “shrink them”. Here is a simple example:

- ▶ Assume you want to estimate the mean of a normal distribution $N(\Theta, I)$ in \mathbb{R}^d , $d \geq 3$.
- ▶ Assume we have just a single data point $X \in \mathbb{R}^d$ from this distribution.
- ▶ Standard least squares estimator: $\hat{\Theta}_{LS} = X$.

(*) Ridge regression as shrinkage method (4)

- ▶ Now consider the following “shrinkage estimator” (it is called the James-Stein estimator): $\hat{\Theta}_{JS} = \left(1 - \frac{d-2}{\|X\|^2}\right)X$.
- ▶ One can prove that it outperforms the standard least squares error in terms of expected least squares error:

$$E(\|\Theta - \hat{\Theta}_{LS}\|) \geq E(\|\Theta - \hat{\Theta}_{JS}\|)$$

Read it on wikipedia if you are interested ☺

History and Terminology

- ▶ Invented by Andrey Tikhonov, 1943, in the context of integral equations.

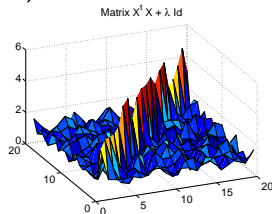
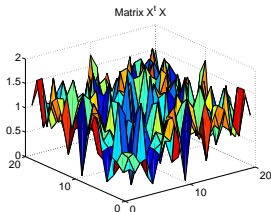
Original publication: *Tikhonov, Andrey Nikolayevich. On the stability of inverse problems. Doklady Akademii Nauk SSSR, 1943*

This type of regularization is often called **Tikhonov regularization** after its inventor.

- ▶ Introduced in statistics literature in the following paper: *Hoerl and Kennard. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 1970.*

History and Terminology (2)

- ▶ Originally, the intention was to make the solution of the least squares problem more stable and to achieve a unique solution.
 - ▶ Replace the matrix $\Phi^t \Phi$ in the least squares solution by the matrix $\Phi^t \Phi + \lambda Id$.
 - ▶ This is where the name “ridge” comes from (we add a little “ridge” on the diagonal of the matrix).



- ▶ The regularization interpretation we described above is more recent.

Summary: Ridge regression

- ▶ Regression problem, $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{R}$
- ▶ Loss function: L_2 -loss
- ▶ Function class: linear functions parameterized by w
 $\mathcal{F} := \{f_w : \mathbb{R}^d \rightarrow \mathbb{R}, f_w(x) = \langle w, x \rangle; w \in \mathbb{R}^d\}$
- ▶ Regularizer: $\Omega(f_w) = \|w\|^2$
- ▶ Finding the function that minimizes the regularized risk is a convex optimization problem, and we can compute its solution analytically.

Lasso: least squares with L_1 -regularization

Books:

- Hastie/Tibshirani/Friedman, Section 3.4.3;
- Bishop Section 3
- Hastie/Tibshirani/Wainwright, Section 2

Original paper: *Tibshirani: Regression shrinkage and selection via the lasso. J. Royal. Statist. Soc. B, 1996*

Sparsity

- ▶ Consider the setting of linear regression with basis functions Φ_1, \dots, Φ_D .
- ▶ It is very desirable to obtain a solution function $f_n := \sum_i w_i \Phi_i$ for which many of the coefficients w_i are zero. Such a solution is called “sparse”.
- ▶ Reasons:
 - ▶ Computational reasons: even if we have many basis functions, we just need to evaluate few of them.
 - ▶ Interpretability of the solution

A naive regularizer for sparsity

QUESTION: WHAT WOULD BE A GOOD REGULARIZER TO ENFORCE SPARSITY?

A naive regularizer for sparsity (2)

Need to find a function that is small if w is sparse:

Use the regularizer

$$\Omega_0(f) := \sum_{i=1}^D \mathbb{1}_{w_i \neq 0}.$$

It directly penalizes the number of non-zero entries w_i .

HOWEVER, USING THIS REGULARIZER IS NOT A GOOD IDEA. WHY?

A naive regularizer for sparsity (3)

Ω_0 is a discrete function, and optimizing discrete functions is typically NP hard.

Excursion: p -norms

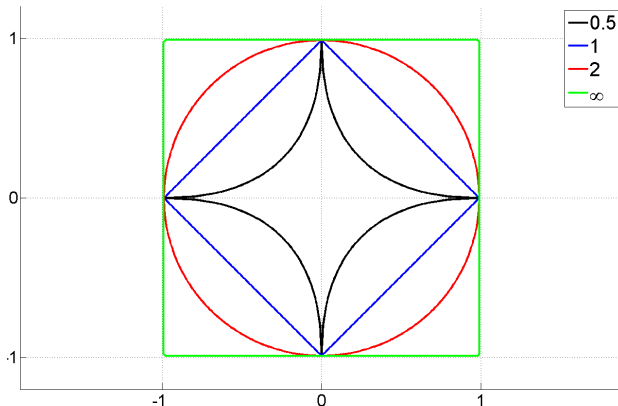
- For $p > 0$, define for a vector $w \in \mathbb{R}^D$

$$\|w\|_p := \left(\sum_{i=1}^D |w_i|^p \right)^{1/p}.$$

- For any $p \geq 1$, this is a norm and as such a convex function. It is called the p -norm.
- for $0 < p < 1$, it is not a norm (exercise!) and also not convex (exercise, and see figure on next slide)

Excursion: p -norms (2)

Unit spheres of p -balls for different values of p (e.g., the red line is the set of points $w \in \mathbb{R}^2$ for which $\|w\|_2 = 1$).



(Image by Matthias Hein)

Excursion: p -norms (3)

For $p = 0$, we can define the function

$$\|w\|_0 := \lim_{p \rightarrow 0} \|w\|_p^p = \lim_{p \rightarrow 0} \sum_{i=1}^d |w_i|^p = \sum_{i=1}^d |w_i|^0$$

(Note that we take the limit of $\|w\|_p^p$, not of $\|w\|_p$).

This is not a norm (it does not even satisfy the homogeneity condition $\|ax\| = a\|x\|$), but it is still called zero-norm in the literature.

It coincides with our regularizer: if we define $0^0 = 0$ and recall that $a^0 = 1$ (for $a \neq 0$), we get

$$\|w\|_0 = \sum_{i=1}^d |w_i|^0 = \sum_{i=1}^d \mathbb{1}_{w_i \neq 0} = \Omega_0(f)$$

Sparsity and the L_1 -norm

We now want to settle for $\|w\|_1$ as a regularizer: It is “as close” to the non-convex regularizer $\|w\|_0$ as possible while still being convex.

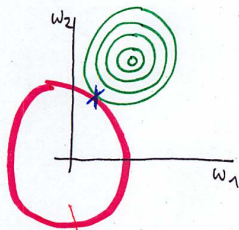
Question: Does it still tend to give sparse solutions?

Answer is yes, see the illustration on the next slide:

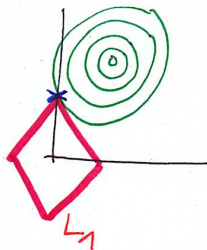
Sparsity and the L_1 -norm (2)

Illustration: Assume we restrict the search to functions with $\|w\| \leq \text{const}$. The blue cross shows the best solution $w = (w_1, w_2)^t$. It is not sparse for L_2 , but sparse for L_1 -norm regularization.

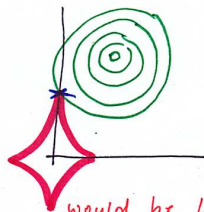
contour lines of the empirical error



L_2 regularization:
set with $\|w\|_2 \leq \text{const}$
 $w_1, w_2 \neq 0$, not sparse



$w_1 = 0$, sparse



would be L_p for
 $p < 1$ ~~||||~~
even sparser, but not
convex any more.

Sparsity and the L_1 -norm (3)

Another intuitive argument why solutions with L_1 -regularization might be sparser than L_2 -regularization:

- ▶ The L_2 -norm puts a particularly large penalty on large coefficients w_i . That is, to avoid a large L_2 -penalty, it is better to have many small w_i that are all non-zero than to have most w_i equal to 0 and a couple of large w_i .
- ▶ The L_1 -norm at least does not have this “preference” for many small weights. It punishes all weights linearly, not quadratic, and thus can afford to have a large weight if at the same time many small weights disappear.

The Lasso

Consider the following regularization problem:

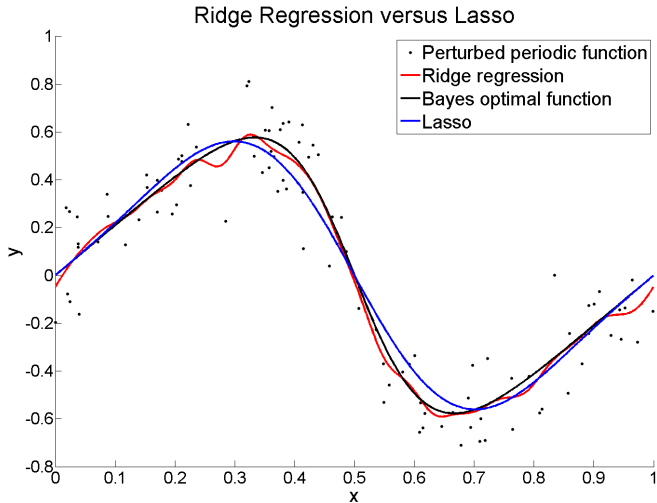
- ▶ Input space \mathcal{X} arbitrary, output space $\mathcal{Y} = \mathbb{R}$.
- ▶ Fix a set of basis functions $\Phi_1, \dots, \Phi_D : \mathcal{X} \rightarrow \mathbb{R}$
- ▶ As function space choose all functions of the form $f(x) = \sum_i w_i \Phi_i(x)$.
- ▶ As regularizer use $\Omega(f) := \|w\|_1 = \sum_{i=1}^D |w_i|$. Choose a regularization constant $\lambda > 0$.
- ▶ Then solve the problem

$$w_{n,\lambda} := \operatorname{argmin}_{w \in \mathbb{R}^D} \frac{1}{n} \|Y - \Phi w\|_2^2 + \lambda \|w\|_1.$$

Solution of the Lasso problem

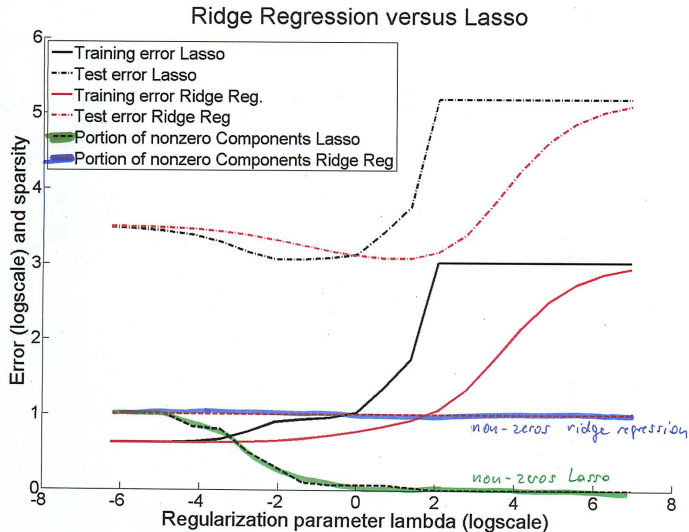
- ▶ The Lasso objective function is convex (it is a sum of two convex functions).
- ▶ However, there does not exist a closed form solution.
- ▶ Hence it has to be solved by a standard algorithm for convex optimization.
 - ▶ In general, any convex solver can be used, but might be slow.
 - ▶ Observing that the problem can be recast as a quadratic problem might help already.
 - ▶ But many faster approaches exist, for example coordinate descent algorithms. We are not going to discuss them in the lecture.

Example



(Figure by Matthias Hein)

Example (2)



(Figure by Matthias Hein)

History

- ▶ The name LASSO stands for “least absolute shrinkage and selection operator”
- ▶ First invented by *Tibshirani: Regression shrinkage and selection via the lasso. J. Royal. Statist. Soc. B, 1996*
- ▶ For a short retrospective and some important literature pointers, see *Tibshirani: Regression shrinkage and selection via the lasso: a retrospective. J. R. Statist. Soc. B (2011)*

Summary: the Lasso

- ▶ Regression problem, \mathcal{X} arbitrary space, $\mathcal{Y} = \mathbb{R}$
- ▶ Loss function: L_2 -loss
- ▶ Function class \mathcal{F} : a linear combination of a fixed set of basis functions.
- ▶ Regularizer: L_1 -norm $\|w\|_1$ to enforce sparsity.
- ▶ Convex optimization problem, no analytic solution, but efficient solvers exist.

Selecting parameters by cross validation

Literature:

- ▶ General: Chapter 7.10 of Hastie/Tibshirani/Friedman
- ▶ Experimental analysis: A meta-analysis of overfitting in machine learning. Roelofs et al, NeurIPS 2019.
- ▶ Y. Yang. Comparing learning methods for classification. Statistica Sinica, 2006.
- ▶ Sylvain Arlot and Matthieu Lerasle: Choice of V for V -fold cross-validation in least-square density estimation. Journal of Machine Learning Research, 2016
- ▶ Roelofs, R., Shankar, V., Recht, B., Fridovich-Keil, S., Hardt, M., Miller, J., Schmidt, L.: A meta-analysis of overfitting in machine learning. NeurIPS 2019

Cross validation - purpose

In all machine learning algorithms, we have to set parameters or make design decisions:

- ▶ Regularization parameter in ridge regression or Lasso
- ▶ Parameter C of the SVM
- ▶ Parameter σ in the Gaussian kernel
- ▶ Number of principle components in PCA
- ▶ But you also might want to figure out whether certain design choices make sense, for example whether it is useful to remove outliers in the beginning or not.

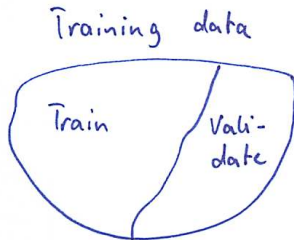
It is very important that all these choices are made appropriately. Cross validation is the method of choice for doing that.

K-fold cross validation

- 1 INPUT: Training points $(X_i, Y_i)_{i=1, \dots, n}$, a set S of different parameter combinations.
- 2 Partition the training set into K parts that are equally large. These parts are called “fold”
- 3 **for all** choices of parameters $s \in S$
- 4 **for** $k = 1, \dots, K$
- 5 Build one training set out of folds $1, \dots, k-1, k+1, \dots, K$ and train with parameters s .
- 6 Compute the validation error $err(s, k)$ on fold k
- 7 Compute the average validation error over the folds:
$$err(s) = \sum_{k=1}^K err(s, k) / K.$$
- 8 Select the parameter combination s that leads to the best validation error: $s^* = \operatorname{argmin}_{s \in S} err(s)$.
- 9 OUTPUT: s^*

K-fold cross validation (2)

①



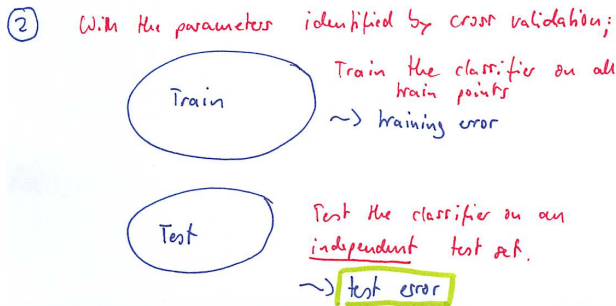
Cross validation
selects parameters

↓
(training error)

↓
validation error.

K-fold cross validation (3)

- Once you selected the parameter combination s^* , you train your classifier a final time on the whole training set. Then you use a completely new test set to compute the test error.



K-fold cross validation (4)

- ▶ Never, never use your test set in the validation phase. As soon as the test points enter the learning algorithm in any way, they can no longer be used to compute a test error. **The test set must not be used in training in any way!**
- ▶ In particular: you are NOT ALLOWED to first train using cross validation, then compute the test error, realize that it is not good, then train again until the test error gets better. As soon as you try to “improve the test error”, the test data effectively gets part of the training procedure and is spoiled.

K-fold cross validation (5)

What number of folds K ?

Not so critical, often people use 5 or 10.

In principle, you could also choose a large K (extreme case: $K = n$, called leave-one-out procedure in statistics). But the computational complexity increases linearly with K .

K-fold cross validation (6)

How to choose the set S ?

- ▶ If you just have to tune one parameter, say the regularization constant λ . Then choose λ on a logspace, say $\lambda \in \{10^{-3}, 10^{-2}, \dots, 10^3\}$.
- ▶ If you have to choose two parameters, say C and the kernel width σ , define, say, $S_C = \{10^{-2}, 10^{-1}, \dots, 10^5\}$, $S_\sigma = \{10^{-2}, 10^{-1}, \dots, 10^3\}$, and then choose $S = S_C \times S_\sigma$. That is, you have to try every parameter combination!
- ▶ You can already guess that if we have more parameters, then this is going to become tricky. Here you might want run several cross validations, say first choose C and σ (jointly) and fix them. Then choose the number of principle components, etc.
- ▶ Note that overfitting can also happen for cross-validation!

K-fold cross validation (7)

- There are also some advanced methods to “walk in the parameter space” (the idea is to try something like a gradient descent in the space of parameters).

A recent meta-study on test set reuse

- ▶ As I said above, you are not allowed to test on your test-set and then go back to improving parameters. In theory.
- ▶ In practice, this is what many people do.
- ▶ In the paper “A meta-analysis of overfitting in machine learning. Roelofs et al, NeurIPS 2019” the authors conducted a large-scale meta-analysis of Kaggle competitions and analyzed whether test set reuse is a big problem or not

The setup was as follows:

A recent meta-study on test set reuse (2)

Setup:

- ▶ The authors considered many kaggle competitions
- ▶ There the standard procedure is: you can test your results on a “public testset”, and the results are being announced on a public leaderboard. In the very end, after the competition time is over, the result is being evaluated on a “private testset” once and the final results are being announced.
- ▶ The difference in the public and private test errors can serve as an approximation to estimate the effect of overfitting.
- ▶ One might expect overfitting if people resubmit very often and really try hard to win the competitions. So the authors particularly considered the top 10% performers of each competitions.

A recent meta-study on test set reuse (3)

Results:

The authors found that overfitting not as big a problem as one might think. In most (but not all) studies, there was a surprisingly small difference between public and private testset.

Read the paper to find out more.

Advantages and disadvantages

Disadvantages of cross validation:

- ▶ Computationally expensive!!! In particular, if you have many parameters to tune, not just one or two.
- ▶ Note that the training size of the problems used in the individual cross validation training runs is $n(\cdot K - 1)/K$. If the sample size is small, then the parameters tuned on the smaller folds might not be the best ones on the whole data set (because the latter is larger).
- ▶ It is very difficult to prove theoretical statements that relate the cross-validation error to the test error (due to the high dependency between the training runs). In particular, the CV error is not unbiased, it tends to underestimate the test error.

Further reading: Y. Yang. *Comparing learning methods for classification*. *Statistica Sinica*, 2006. and references therein.

Advantages and disadvantages (2)

Advantages:

There is no other, systematic method to choose parameters in a useful way.

Always, always, always do cross validation!!! If possible, make sure the final test set is never touched while training (retraining for improving the test error is not allowed, then the data is spoiled). If you do repeated training/testing, be aware of the potential danger, and do report it in the paper / thesis / report you are writing.

Probabilistic interpretation of linear regression

The following slides just provide a sketch. If you want to know more or see exact formulas, please read this book chapter:

Kevin Murphy: Machine Learning, a probabilistic perspective, Chapter 7

Linear regression: ERM = maximum likelihood

- Assume the following probabilistic setup: the data is generated by the following linear model:

$$Y = Xw + \text{noise}$$

where w is unknown and the noise follows a (d -dim) normal distribution $N(0, \sigma^2 I)$ (σ unknown “meta-parameter”, considered fixed):

$$Y|X, w \sim N(Xw, \sigma^2 I)$$

Linear regression: ERM = maximum likelihood (2)

- **Maximum likelihood framework:** want to find the parameter w such that the likelihood of the observations is maximized:

$$\max_w P(Y|X, w)$$

$$\max_w \exp(-\|Y - Xw\|^2/\sigma^2)$$

$$\min_w \|Y - Xw\|^2$$

That is: Maximum likelihood regression with a Gaussian noise model corresponds to ERM with the L_2 loss function.

Linear regression: RRM = Bayesian MAP

- Assume that the observations are generated as above, but additionally assume that we have a prior distribution over the parameter w :

$$Y|X, w \sim N(Xw, \sigma^2 I) \quad \text{and} \quad w \sim N(0, \tau^2 I)$$

- **Bayesian maximum a posteriori approach (MAP)**: choose w that maximizes the posterior probability:

$$P(w|X, Y) = \frac{P(Y|X, w)P(w)}{P(Y|X)}$$

- Writing down all formulas, can see: Leads to **ridge regression** (with tradeoff constant $\lambda = \sigma^2/\tau^2$):

$$\min_w \|Xw - Y\|^2 + \lambda \|w\|^2$$

More generally: Bayesian interpretation of ERM and RRM

- ▶ The **noise model** in the probabilistic setup corresponds to the choice of a **loss function** in the ERM framework.
- ▶ The **prior distribution** of the parameter in the Bayesian model corresponds to a particular choice of **regularizer** in RRM.

Examples:

- ▶ If the data contains many outliers, one chooses a Laplace noise model (rather than a Gaussian one): $P(w) \approx \exp(-\|w\|/\tau)$. This then leads to the L_1 -loss function

$$\frac{1}{n} \sum_i |Y_i - \hat{Y}_i|$$

- ▶ Similarly, if we use a Laplace prior instead of a normal prior for the parameter, we end with Lasso regularization instead of Tikhonov/Ridge regression.

Linear methods for classification

Intuition

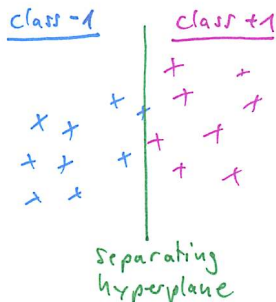
Intuition

Given:

- ▶ We assume that our data lives in \mathbb{R}^d (perhaps, through a feature space representation).
- ▶ Want to solve a classification problem with input space $\mathcal{X} = \mathbb{R}^d$ and output space $\mathcal{Y} = \{\pm 1\}$ (for simplicity we focus on the two-class case for now).

Intuition (2)

- Idea is to separate the two classes by a linear function:



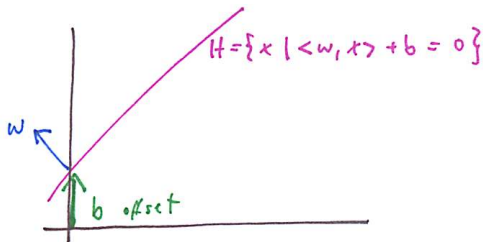
Hyperplanes in \mathbb{R}^d

Now let's consider linear classification with hyperplanes.

- A hyperplane in \mathbb{R}^d has the form

$$H = \{x \in \mathbb{R}^d \mid \langle w, x \rangle + b = 0\}$$

where $w \in \mathbb{R}^d$ is the normal vector of the hyperplane and b the offset.



Classification using hyperplanes

To decide whether a point lies on the right or left side of a hyperplane, we use the decision function

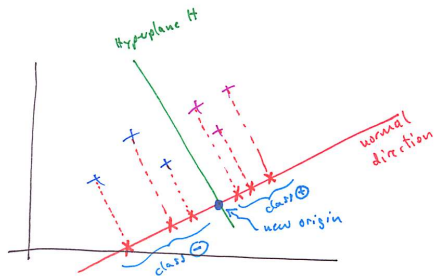
$$\text{sign}(\langle w, x \rangle + b) \in \{\pm 1\}$$

Note that it is a convenient convention to use the class labels $+1$ and -1 (because we can then simply use the sign function).

Projection interpretation

Here is another way to interpret classification by hyperplanes:

- ▶ The function $\langle w, x \rangle$ projects the points x on a real line in the direction of the normal vector of the hyperplane.
- ▶ The term b shifts them along this line.
- ▶ Then we look at the sign of the result and classify by the sign



Loss functions for classification

There exist quite a number of loss functions that are used in classification:

We are now going to see a number of basic approaches for linear classification based on various loss functions:

- ▶ Linear discriminant analysis (least squares loss)
- ▶ Logistic regression (logistic loss)
- ▶ Linear support vector machines (hinge loss)

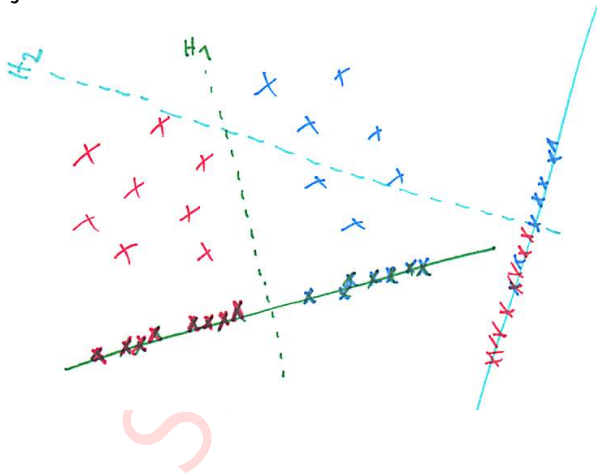
(*) Linear discriminant analysis

Literature:

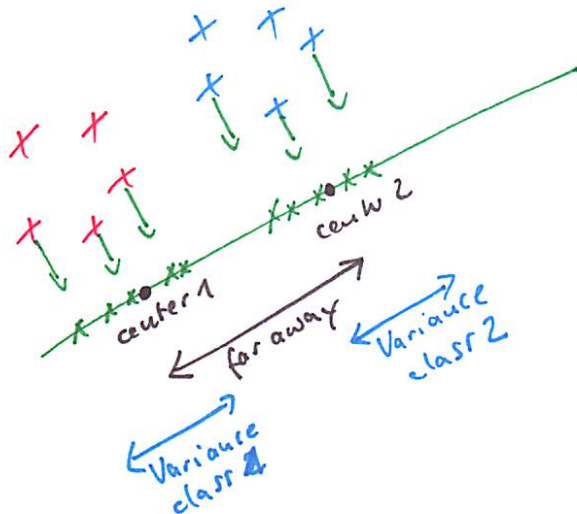
- ▶ Hastie/Tibshirani/Friedman Sec. 4.3
- ▶ Duda / Hart

LDA: Geometric motivation

Different projections: which one is better for classification?



LDA: Geometric motivation (2)

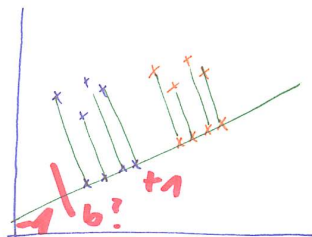
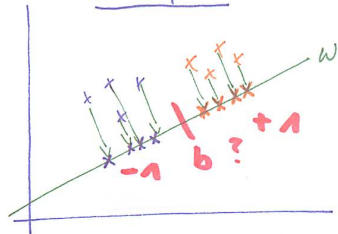


LDA: Geometric motivation (3)

- ▶ Linear classification amounts to a one-dimensional projection.
- ▶ LDA: Chooses the projection direction w such that ...
 - ▶ The class centers are as far away from each other as possible
 - ▶ The variance within each class is as small as possible.

LDA: Geometric motivation (4)

Role of b :



b decides where to "cut" the hyperplane to define the two classes

LDA: Geometric motivation (5)

Observe the different roles of w and b :

Step 1: finding a good separating direction $\leadsto w$

- ▶ All the intuition above is about how to find a good direction w (neither the separation of the two classes nor their variances are affected by b).
- ▶ So the first step of LDA will be to find a good direction w .

Step 2: given w , decide where to cut $\leadsto b$

- ▶ The parameter b only influences where we “cut” the two classes, after we projected them on w .
- ▶ The best parameter b is thus selected only once we know w .

Note: the label information is used in both steps!

Formally: the Fisher criterion

Define the following quantities for class +1:

- ▶ Let n_+ be the number of points in class 1
- ▶ Define the center of class 1 as

$$m_+ := \frac{1}{n_+} \sum_{\{i \mid Y_i = +1\}} X_i \in \mathbb{R}^d$$

Note that after projecting on w , the mean is given as $\langle w, m_+ \rangle$.

- ▶ Define the **within-class variance** after projecting on w as

$$\sigma_{w,+}^2 := \frac{1}{n_+} \sum_{\{i \mid Y_i = +1\}} \left(\langle w, X_i \rangle - \langle w, m_+ \rangle \right)^2$$

Make the analogue definitions for class -1: ...

Formally: the Fisher criterion (2)

Now define the **Fisher criterion** as

$$J(w) = \frac{\langle w, m_+ - m_- \rangle^2}{\sigma_{w,+}^2 + \sigma_{w,-}^2}.$$

The idea of linear discriminant analysis is now to select $w \in \mathbb{R}^d$ such that the **Fisher criterion** is maximized.

Fisher criterion in matrix form

We can write the Fisher criterion in matrix form as follows:

- Define the between-class scatter matrix as

$$C_B := (m_+ - m_-)(m_+ - m_-)^t \in \mathbb{R}^{d \times d}$$

- Define the total within-class scatter matrix as

$$\begin{aligned} C_W := & \frac{1}{n_+} \sum_{\{i \mid Y_i = +1\}} (X_i - m_+)(X_i - m_+)^t \\ & + \frac{1}{n_-} \sum_{\{i \mid Y_i = -1\}} (X_i - m_-)(X_i - m_-)^t \end{aligned}$$

- The Fisher criterion can now be rewritten as

$$J(w) = \frac{\langle w, C_B w \rangle}{\langle w, C_W w \rangle}$$

Solution vector w

Proposition 9 (Solution vector w^* of LDA)

If the matrix C_W is invertible, then the optimal solution of the problem $w^* := \operatorname{argmax}_{w \in \mathbb{R}^d} J(w)$ is given by

$$w^* = (C_W)^{-1}(m_+ - m_-).$$

Remark: it can happen that C_W is not invertible (in particular, if $d > n$. WHY?).

In this case, one can resort to the pseudo-inverse.

Proof (sketch).

► Take the derivative:

$$\frac{\partial J}{\partial w}(w) = 2 \frac{C_B w \langle w, C_W w \rangle - C_W w \langle w, C_B w \rangle}{\langle w, C_W w \rangle^2}$$

Solution vector w (2)

- Set it to 0:

$$\frac{\langle w, C_W w \rangle}{\langle w, C_B w \rangle} C_B w = C_W w$$

- Rewrite (plug in the definition of C_B):

$$\underbrace{\frac{\langle w, C_W w \rangle}{\langle w, C_B w \rangle}}_{\in \mathbb{R}} (m_+ - m_-) \underbrace{(m_+ - m_-)^t w}_{\in \mathbb{R}} = C_W w$$

- Additionally, observe that $J(w)$ is invariant under rescaling of w , that is $J(w) = J(\alpha w)$ for $\alpha \neq 0$.
- So the solution is

$$w^* \propto (C_W)^{-1} (m_+ - m_-)$$

- We can check that the Hessian of $J(w)$ at w^* is negative definite, so w^* is indeed a maximum.



Determining b

So far, we only discussed how to find the normal vector w . How do we set the offset b ? (Recall that the hyperplane is $\langle w, x \rangle + b$).

The standard is to choose b , once w is known, as to minimize the training error.

SKIPPED

LDA, alternative motivation by ERM

We can also start with the ERM framework and make the following assumptions:

- As **function class** we use the affine linear functions as above:

$$\mathcal{F} = \{f(x) = \langle w, x \rangle + b; w \in \mathbb{R}^d, b \in \mathbb{R}\}$$

- As **loss function** we use the squared loss between the real-valued output (!) of the function $f(x)$ and the actual class labels:

$$\ell(X, Y, f(X)) = (Y - f(X))^2$$

- **No assumption on the underlying distributions.**

Then we can prove the following nice theorem:

LDA, alternative motivation by ERM (2)

Theorem 10 (LDA as ERM)

Consider the following two optimization problems:

(1) Minimizing the least squares loss of affine linear functions:

$$(w', b') := \operatorname{argmin}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \sum_{i=1}^n (Y_i - \langle w, X_i \rangle - b)^2$$

(2) The LDA problem:

$$w^* = \operatorname{argmax}_{w \in \mathbb{R}^d} J(w)$$

Then the solutions w' and w^* coincide up to a constant, in particular they correspond to the same hyperplane.

LDA, alternative motivation by ERM (3)

Proof: skipped.

SKIPPED

LDA, alternative motivation by ERM (4)

Comments:

- Note: The least squares loss in problem (1) of the theorem is with respect to $\langle w, X_i \rangle + b$, not with respect to the sign of this expression (which is what we are ultimately interested in):

$$(Y_i - \underbrace{(\langle w, X_i \rangle + b)}_{\in \mathbb{R}})^2 \text{ versus } (Y_i - \underbrace{\text{sign}(\langle w, X_i \rangle + b)}_{\in \{\pm 1\}})^2$$

LDA, motivation by Bayesian approach

Let us make the following assumptions:

- ▶ The two class conditional distributions $P(X|Y = 1)$ and $P(X|Y = -1)$ follow a multivariate normal distribution with the same covariance matrix, but different means
- ▶ Classes have equal prior weights, that is $P(Y = 1) = P(Y = -1) = 0.5$.

Then we can argue as follows:

- ▶ Bayesian approach: under these assumptions the optimal classifier selects according to whether

$$P(Y = 1 \mid X = x) \stackrel{?}{>} P(Y = -1 \mid X = x).$$

- ▶ Equivalently:

$$\log \left(P(Y = 1 \mid X = x) / P(Y = -1 \mid X = x) \right) \stackrel{?}{>} 0.$$

LDA, motivation by Bayesian approach (2)

- ▶ If we compute this term for the normal distributions, one can see that the decision boundary between the two classes (i.e., the set where both classes have equal posteriors) is a hyperplane, and coincides with the LDA solution.
- ▶ Details skipped, see Hastie/Tibshirani/Friedman for details.

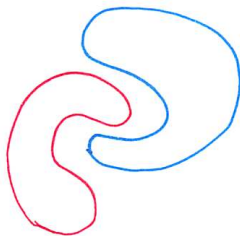
LDA, motivation by Bayesian approach (3)

Insights:

- ▶ Under the given assumptions (normal distributions, same weights, same covariance, etc), LDA should work nicely!
- ▶ We can also suspect that it does not such a good job if the assumptions are not satisfied.

Limitations and generalizations

- ▶ LDA does not work well if the classes are not “blobs”



- ▶ LDA does not work well if the variance of the two classes is very different from each other (remember, in the derivation of LDA based on Gaussian distributions we assumed equal variance for both classes).

Limitations and generalizations (2)

Generalizations:

- ▶ LDA tends to overfit (so far, we do not regularize). There also exist regularized versions, we'll skip it.
- ▶ LDA can be generalized to multiclass problems as well. We'll skip it.

History

- ▶ A variant of this was first published by R. Fisher in 1936:
Fisher, R. A. (1936). The Use of Multiple Measurements in Taxonomic Problems. Annals of Eugenics 7 (2): 179–188.
- ▶ LDA goes under various names: Linear discriminant analysis, Fisher's linear discriminant.
- ▶ R. Fisher is THE founder of modern statistics (design of experiments, analysis of variance, maximum likelihood, sufficient statistics, randomized tests, ...)

Summary: Linear discriminant analysis (LDA)

Three different motivations:

- ▶ Geometric motivation: project in a direction that separates the classes well
- ▶ ERM motivation: minimize the least squares loss on space of linear functions
- ▶ Model-based (probabilistic) motivation: Bayes classifier under assumption of normal distributions with equal variances

All the motivations lead to the same algorithm:

- ▶ Minimize the Fisher criterion
- ▶ Can compute solution vector w analytically

Logistic regression

Literature: Hastie/Tibshirani/Friedman Section 4.4

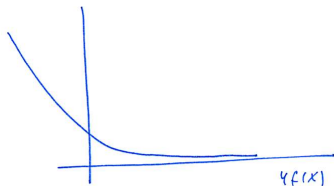
For the probabilistic point of view, see Chapter 8 in Murphy

Logistic regression problem as ERM

- ▶ Want to solve classification on \mathbb{R}^d with linear functions:
 - ▶ Given $X_i \in \mathbb{R}^d$, $Y_i \in \{\pm 1\}$.
 - ▶ $\mathcal{F} = \{f(x) = \langle w, x \rangle + b; w \in \mathbb{R}^d, b \in \mathbb{R}\}$
 - ▶ Use ERM
- ▶ Using L_2 -loss corresponds to Linear Discriminant Analysis (LDA)
- ▶ Now: use the logistic loss function:

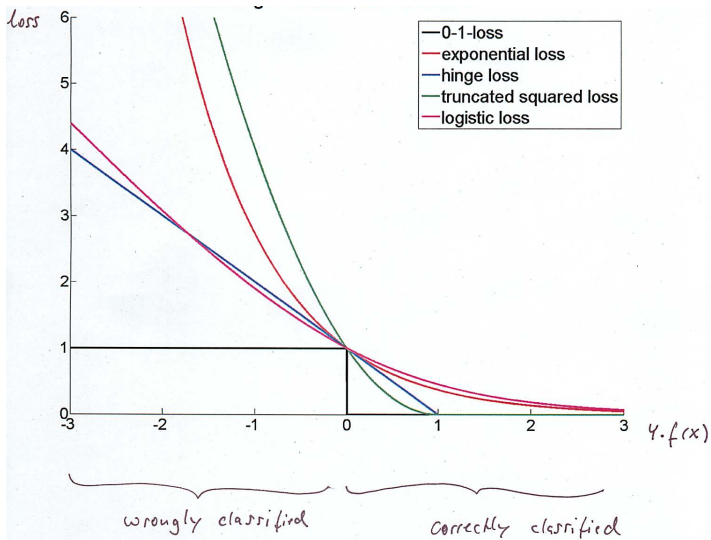
Logistic regression problem as ERM (2)

$$\ell(X, f(X), Y) = \log_2(1 + \exp(-Y f(X)))$$



- ▶ It already starts to “punish” if points are still on the correct side of the hyperplane, but get close to it.
- ▶ Once on the wrong side, it punishes “moderately” (close to linear)

Logistic regression problem as ERM (3)



Computing the ERM solution

Consider the problem of finding the best linear function under the logistic loss in the ERM setting.

- ▶ There is no closed form solution for this problem.
- ▶ Good news: the logistic loss function is convex.
This can be proved by showing that the Hessian matrix is positive definite.
- ▶ So we can use our favorite convex solver to obtain the logistic regression solution.
- ▶ The standard technique in this case is the Newton-Raphson algorithm, but we won't discuss the details.

But why would someone come up with the logistic loss function???

The answer comes from the following Bayesian approach to classification.

The logistic model

- ▶ We do NOT make a full model of the joint probability distribution $P(x, y)$ or the class conditional distributions $P(y|x)$ (generative approach).
- ▶ We just specify a model for the conditional posterior distributions (discriminative approach):

Assume that $f = \langle w, x \rangle + b$ (with parameters w and b) describe is given. For this f , we stipulate the following model:

$$P(Y = y \mid X = x) := \frac{1}{1 + \exp(-yf(x))}$$

The function $1/(1 + \exp(-t))$ is called the **logistic function** and looks as follows:

The logistic model (2)

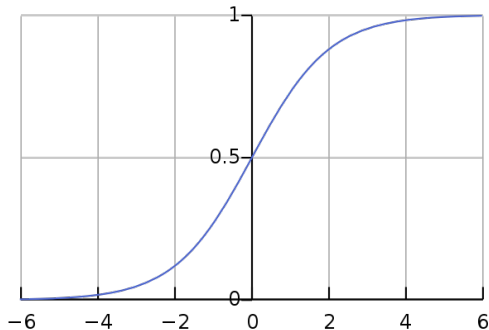
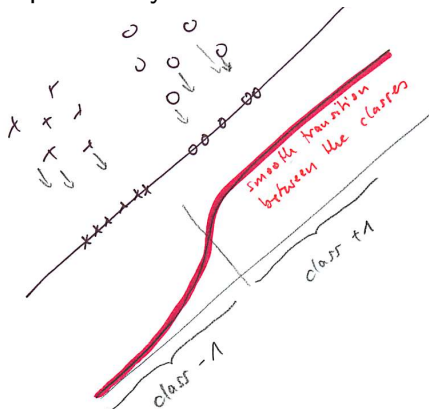


Fig from wikipedia)

The logistic model (3)

► Intuition:

Consider the projection scenario. Instead of a hard threshold (left = one class, right = other class) we have a smooth transition of the probability.



The logistic model (4)

The value of $P(Y = y|X = x)$ tells “how far” we are from the decision surface, that is “how sure” the classifier is about this class. If it is ≈ 0.5 this means that the classifier does not really have a good idea of the class itself. If $f(x)$ is close to 0 or 1, this means that the classifier is “pretty sure”.

The logistic model (5)

- Finding the f that maximizes $P(Y = y \mid X = x) = 1/(1 + \exp(-yf(x)))$ corresponds to minimizing the following loss function:

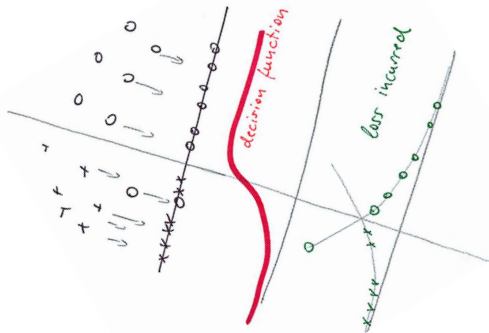
$$\ell(X, Y, f(X)) = \log(1 + \exp(-Y f(X)))$$

This is the logistic loss function.

Note that the logistic loss also punishes points that are correctly classified but are “too close” to the hyperplane.

For such points, the classifier “is not sure”, but ideally we would like to find a classifier that is “pretty sure” on which sides all points belong.

The logistic model



Decision function:

$$P(Y = \text{circle} | X = x, b) = 1 / (1 + \exp(-y f(x)))$$

Loss incurred for the respective decisions: logistic

Adding regularization

- ▶ As in linear regression, we can now use regularization to avoid overfitting.
- ▶ For example, we could use $\Omega(f) = \|w\|_2^2$ (as in ridge regression) or $\Omega(f) = \|w\|_1$ (as in Lasso).
- ▶ Then regularized logistic regression minimizes

$$\frac{1}{n} \sum_{i=1}^n \log \left(1 + \exp(-Y_i(\langle w, X \rangle + b)) \right) + \lambda \Omega(f).$$

- ▶ If the regularizer is convex in w , then so is the regularized logistic regression problem. It can be solved by standard convex solvers.
- ▶ More specialized (more efficient) solvers exist.

History of logistic regression

Very nice historic account: *Cramer: The origins of logistic regression. Tinbergen Institute Working Paper, 2002*

- ▶ Dates back to the 19th century to the work of Pierre-Francois Verhulst (published in several papers around 1845)
- ▶ Rediscovered in the 1920 by Pearl and Reed
- ▶ Many variants and adaptations (“probit” or “logit”)
- ▶ In 1973, Daniel McFaden draws the connections to decision theory; in 2000, he earns the nobel prize in economic sciences for his development of theory and methods for analyzing discrete choice!

Summary: logistic regression

- ▶ Loss function: logistic loss (a “smoothed” version of a step function)
- ▶ Function class: linear
- ▶ Either pure empirical risk minimization, or regularized risk minimization, for example with L_1 - or L_2 -regularizer
- ▶ Convex optimization problem, no closed form solution.

(*) Probabilistic interpretation of linear classification

Literature: Kevin Murphy: Machine Learning, a probabilistic perspective, Chapter 8

General idea

- ▶ In linear discriminant analysis (LDA): Minimizing the L_2 loss over the class of linear function is “the same” as finding the Bayesian decision theory solution for the probabilistic model with Gaussian class conditional priors, and Gaussian noise, and uniform class prior.
- ▶ In logistic regression: Minimizing the logistic loss function over linear functions can be interpreted as a probabilistic approach as well.

Let's briefly look at the general concept.

Excursion: Probabilistic interpretation of ERM

- ▶ Bayesian approach: choose $f(x)$ according to whether $P(Y = +1 \mid X = x)$ is larger or smaller than $P(Y = -1 \mid X = x)$.
- ▶ Assume that the conditional probability $P(Y = +1 \mid X = x)$ has a certain functional form, that is it can be described by some function $f \in \mathcal{F}$ (for some appropriate \mathcal{F}).
- ▶ The goal is to find the function $f \in \mathcal{F}$ that “best explains our training data”. That is, for each training point we would like to have $P(f(X_i) = Y_i)$ as large as possible.
- ▶ This amounts to selecting $f \in \mathcal{F}$ by

$$\operatorname{argmax}_{f \in \mathcal{F}} \prod_{i=1}^n P(f(X_i) = Y_i)$$

Excursion: Probabilistic interpretation of ERM

(2)

- This is equivalent to the following problem (simply take $-\log$):

$$\operatorname{argmin}_{f \in \mathcal{F}} \sum_{i=1}^n \underbrace{-\log P(f(X_i) = Y_i)}_{=: \ell(X_i, f(X_i), Y_i)}$$

- This approach can be interpreted as **empirical risk minimization** with respect to this newly defined loss function ℓ :

$$\operatorname{argmin}_{f \in \mathcal{F}} \sum_{i=1}^n \ell(X_i, f(X_i), Y_i)$$

Excursion: Probabilistic interpretation of ERM

(3)

What does this tell us?

Assume we start with an assumption how the probability distributions $P(Y \mid X = x)$ look like, and we follow the Bayesian approach of selecting according to $P(Y \mid X = x)$.

Then there always exists a particular loss function ℓ such that this approach corresponds to ERM with this particular loss function.

Note that it does not always work the other way round (if we start with a given loss function ℓ , it is not always possible to construct a corresponding model probability distribution)

Excursion: Probabilistic interpretation of ERM

(4)

Why is this insight useful?

It helps to get more intuition:

- For some loss functions, we can “understand” what the corresponding probabilistic model is. This gives insight into when a particular approach might or might not work.

Linear discriminant analysis is an example for this: you would not guess that the quadratic loss for a linear function class means that we assume that classes are round blobs with the same shape (normal distributions with the same covariance structure).

Excursion: Probabilistic interpretation of ERM

(5)

- ▶ Given a particular model, writing down the loss function helps to understand the behavior of the classifier: What are the errors that are punished most? So what does the classifier try to avoid at all costs?

Evaluation of classification results

Counting performance measures

There are many different ways to measure the error of a classifier, we are going to summarize many of them now.

The main difference between these performance measures is if the classes are very unbalanced.

Counting performance measures (2)

P = number of points whose true label is positive

N = number of points whose true label is negative

Confusion table (case of two classes):

predicted			
true	positive	true positive tp	false negative fn
	negative	false positive fp	true negative tn

For example, fp denotes the number of points that have wrongly been predicted to belong to the positive class.

Counting performance measures (3)

- ▶ **Error rate:** fraction of points that are wrongly classified:
 $(fn + fp)/(P + N)$
- ▶ **Accuracy:** fraction of examples that are correctly classified:
 $1 - errorrate$

Counting performance measures (4)

- ▶ True positive rate (**sensitivity**): tp / P
(“How many of the true positives did we find?”)
- ▶ False positive rate: fp / N
(“How many of the negative points have been wrongly classified positive”)?
- ▶ True negative rate (**specificity**): tn / N
- ▶ False negative rate: fn / P

Counting performance measures (5)

If the classes are highly unbalanced, one sometimes uses:

- ▶ Positive predictive value: $tp/(tp + fp)$
- ▶ Negative predictive value: $tn/(tn + fn)$

Counting performance measures (6)

In information retrieval the following measures are common (here we are mainly interested in the positive class, we want to retrieve documents from a collection that fit the search query):

- ▶ **Recall:** tp/P (how many positive examples can we find)
- ▶ **Precision:** $tp/(tp + fp)$ how many of all positively classified examples are indeed correct

These are in particular used in applications where discovering true negatives does not add much value to a classifier.

ROC and AUC

In many applications, in particular in information retrieval, there are many more negative examples than positive examples:

- ▶ Most webpages are irrelevant to a certain search query.
- ▶ Most possible links do not exist in a social network.
- ▶ etc

In such cases, classification accuracy is a very bad performance measure (WHY?).

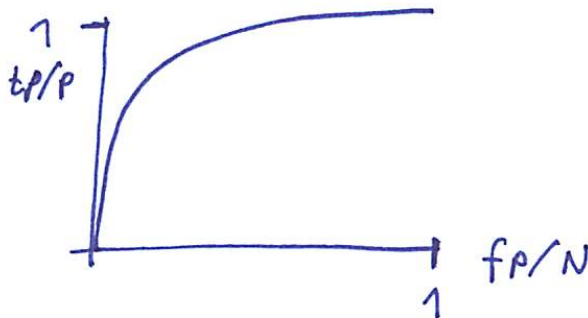
Instead, one is interested in true positives and false positives. To be able judge classifiers based on both criteria simultaneously, we can now use ROC curves.

ROC and AUC (2)

ROC (=Receiver-operator characteristic) curve:

- ▶ Consider a family of classifiers $class = \text{sign}(g(x) + \Theta)$
- ▶ Plots the false positive rate versus the true positive rate for varying decision threshold Θ :
 - ▶ Vary Θ from $-\infty$ to ∞
 - ▶ Evaluate $tp(\Theta)$ and $fp(\Theta)$
 - ▶ Then plot the points $(fp(\Theta)/N, tp(\Theta)/P)$.
 - ▶ Leads to a curve in $[0, 1]^2$:

ROC and AUC (3)

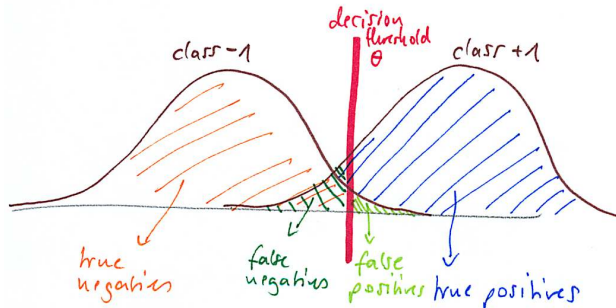


Note: indeed,

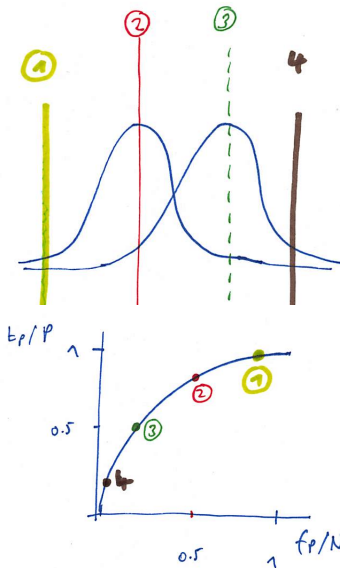
- ▶ $tp/P \in [0, 1]$
- ▶ $fp/N \in [0, 1]$

ROC and AUC (4)

Intuition with normal distributions:

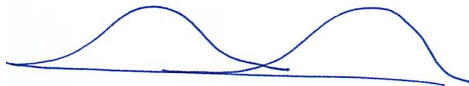


ROC and AUC (5)



ROC and AUC (6)

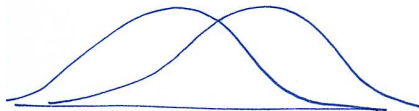
"good" Data



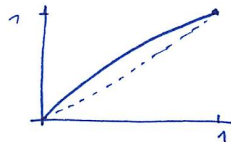
ROC



"bad" Data



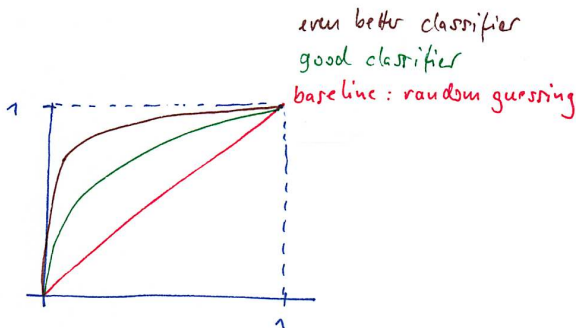
ROC



ROC and AUC (7)

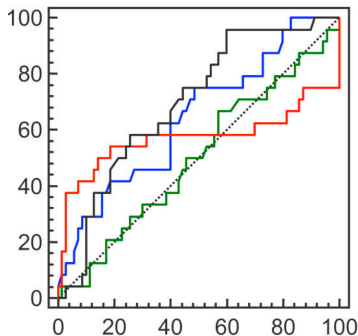
ROC for comparing classifiers:

- ▶ Assume you have two classifiers that depend on a certain parameter σ
- ▶ Plot the ROC curve of both classifiers
- ▶ If the curve of classifier 1 is always above the one of classifier 2, then classifier 1 is considered superior.



ROC and AUC (8)

- Often, such a clear picture is not true, the curves are going to intersect.

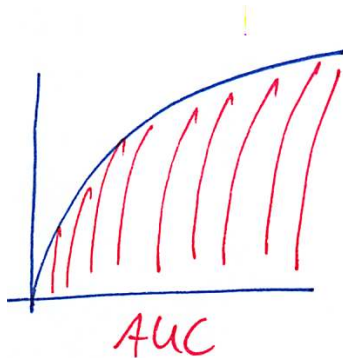


- In this case, you might still be able to say in what parameter range one classifier is better than the other.
- Or you might want to use AUC.

ROC and AUC (9)

AUC (Area under the ROC curve):

- ▶ To translate the ROC to a “number”, sometimes the area under the ROC curve is used as a performance measure.
- ▶ The larger the area, the “better” the classifier.



ROC and AUC (10)

ROC and AUC are used a lot in machine learning. However, there is also a lot of criticism related to these measures, see references in the end.

Multi-class performance measures

- ▶ Accuracy and error rate still can be defined, but the more classes the less informative are these numbers. WHY?
- ▶ In general, the more classes the harder it is to summarize the classification performance in one number.
- ▶ A good way to access the quality of multi-class classifiers is to discuss the confusion matrix directly. Only works if the number of classes is still small.

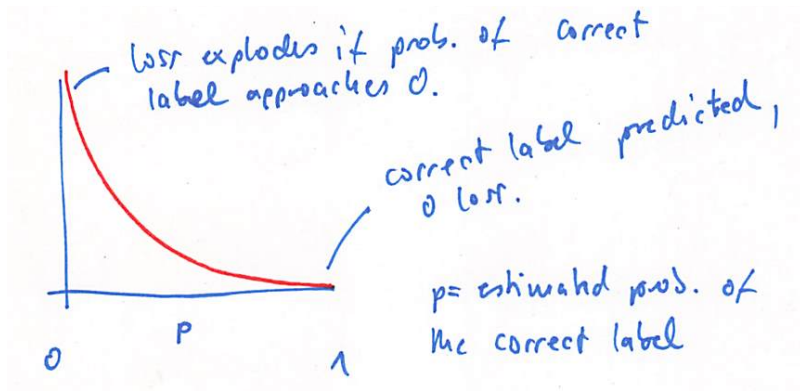
Multi-class performance measures (2)

- Alternatively, one can use a loss function that is tuned towards multi-class settings right away. For example, the cross-entropy loss function: it measures the difference of probability vectors p and q on a space of D classes via

$$H(p, q) = - \sum_{d=1}^D p_d \log(q_d)$$

If we want to visualize the term for one of the bins, it looks like this:

Multi-class performance measures (3)



Comparing many classifiers

- ▶ Below is a table I took from a random publication on classification (μ denotes the mean, σ the standard deviation of the result on several independent tests).
- ▶ You will find similar tables in very many publications.
- ▶ What can you read from this table???

Comparing many classifiers (2)

Table 1: Average classification error rates.

	SVM-R		MORF		LDAW		KNN		DANN		MACHETE		SCYTHE		C4.5	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Iris	4.9	3.11	4.6	2.88	5.4	2.88	4.9	3.58	6.4	3.83	6.0	4.21	4.8	2.95	8.3	3.54
Vote	3.7	1.78	3.5	1.96	7.6	3.79	8.4	2.35	3.9	1.90	5.4	2.12	5.4	1.65	3.5	1.60
Sonar	14.4	5.06	13.4	4.24	16.0	3.42	16.0	3.44	12.9	4.02	21.0	3.81	18.0	3.87	30.1	4.69
Ion	5.4	0.96	7.2	1.98	11.4	2.82	12.59	2.24	10.4	2.48	11.5	2.29	13.5	2.54	10.7	2.60
Liver	28.0	2.46	30.3	2.63	36.3	4.46	36.4	33.96	32.6	4.36	36.1	3.02	36.8	4.53	37.6	3.12
Hep	15.2	3.93	14.5	3.95	14.4	4.13	14.8	4.80	13.6	3.90	17.4	4.35	16.9	4.13	19.6	4.21
Cancer	2.98	0.62	2.9	0.89	3.2	0.89	3.1	0.91	2.8	0.78	3.6	1.04	3.2	0.95	4.0	1.31
Pima	23.5	2.41	24.6	2.57	26.6	2.91	27.1	3.35	26.4	2.47	25.7	3.00	25.7	3.00	19.1	1.33
OQ	3.1	1.35	4.3	2.11	6.1	2.16	6.4	2.04	4.5	1.88	8.0	1.69	6.3	2.01	3.5	0.81
Unstruct	29.6	2.70	7.0	5.92	26.1	11.78	34.0	3.56	30.0	3.39	9.0	2.25	13.6	3.07	10.1	7.40

Comparing many classifiers (3)

... hard to see anything. By all means, avoid tables!

Comparing many classifiers (4)

Often it is not so easy to decide which classifier is “better”:

- ▶ “Better in general” will be hard anyway (no free lunch theorem!)
- ▶ Depending on the choice of data sets! (here lots of cheating is possible)
- ▶ When would you say is a classifier “really better” than another one???

Statistical tests for comparing classifiers

You can try to do this in a more sound way using statistical tests:

- ▶ The null hypothesis is that both classifiers perform the same
- ▶ As test statistic use the difference in error rates: $err_i - \tilde{err}_i$ on many different data sets i (here err_i and \tilde{err}_i are the errors of the two classifiers on data set i)
- ▶ Assumption: These values are independent across data sets.
- ▶ Then you can use a t -test to test whether the performance of the classifiers is significantly different.

Permutation test:

- ▶ You can also use a permutation test.
- ▶ Here you compare the statistic $err_i - \tilde{err}_i$ against the statistic where you randomly exchange err_i and \tilde{err}_i .
- ▶ Read on permutation tests how this works ...

Statistical tests for comparing classifiers (2)

Comment:

- ▶ It is not extremely popular to use statistical tests, and it is also somewhat questionable whether it is really useful.
- ▶ A test has to make assumptions on the underlying distribution.
 - ▶ For example, the t -test assumes the “data” (in this case, the values err_i and \tilde{err}_i) to be normally distributed, independent, and from the same population.
 - ▶ This cannot really be true, it would not even hold for the Bayes errors (if we plotted a histogram of the Bayes errors in the data sets we use, it would not look like a normal distribution).
- ▶ But if the assumptions are not satisfied, the test is meaningless.

Some critical remarks

A quote from Duin (1996), see also Hand (2008), (full references below):

We are interested in the real performance for practical applications. Therefore, an application domain has to be defined. The traditional way to do this is by a diverse collection of datasets. In studying the results, however, one should keep in mind that such a collection does not represent any reality. It is an arbitrary collection, at most showing partially the diversity, but certainly not with any representative weight. It appears still possible that for classifiers showing a consistently bad behavior in the problem collection, somewhere an application exists for which they are perfectly suited.

Some critical remarks (2)

And one more (same source):

In comparing classifiers one should realize that some classifiers are valuable because they are heavily parameterized and thereby offer a trained analyst a large flexibility in integrating his problem knowledge in the classification procedure. Other classifiers, on the contrary, are very valuable because they are entirely automatic and do not demand any user parameter adjustment. As a consequence they can be used by anybody. It is therefore difficult to compare these types of classifiers in a fair and objective way.

Some critical remarks (3)

If you want to compare classifiers:

- ▶ Always try to compare them *for a particular task* (the “best classifier” does not exist).
- ▶ Always test on a variety of data sets, under many different conditions, on many different data sets.
- ▶ Be fair: try to implement all classifiers in the best way you can. If available, use implementations by the people who invented the algorithms (often a considerable amount of work goes into fine-tuning a classifier).
- ▶ Most people won't believe you if you say that your classifier “consistently outperforms” the other classifiers.

Some critical remarks (4)

- Try to assess the strengths / weaknesses of your classifier (and be open about it): the most helpful insight is to say that “in this situation, prefer classifier A, in that situation classifier B”. This also means to identify the situations where your approach does NOT work.

Some critical remarks (5)

If you read that “one classifier is better than the other one” always be a bit suspicious:

- ▶ Has the experiment been designed in a fair way? For example, have all parameters been chosen by cross validation?
- ▶ How were the data sets selected?
- ▶ Are there different data sets of different types (many/few sample points, high/low dimension, balanced/unbalanced classes, toy/real world data, ...)

Some references

There is lots of research on how to compare classifiers.

David Hand (London) did a lot of (critical) research on the topic of comparing classifiers, see for example:

- ▶ *Hand D.J. Assessing the performance of classification methods. International Statistical Review, 2012*
- ▶ Hand D.J. Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine Learning* 77, 2009.
- ▶ *Jamain, A., Hand, D. J. Mining supervised classification performance studies: a meta-analytic investigation. Journal of Classification, 25, 87-112. 2008.*

Some references (2)

A couple of other references:

- ▶ *Duin, R. A Note on Comparing Classifiers, Pattern Recognition Letters, 17:529-536.1996.*
- ▶ Demsar: Statistical comparisons of classifiers over multiple data sets. JMLR, 2006.
- ▶ *Yang: Comparing learning methods for classification. Statistica Sinica, 2006.*

Linear support vector machines

Literature:

- ▶ Schölkopf / Smola Section 7
- ▶ Shawe-Taylor / Cristianini

Intuition and primal

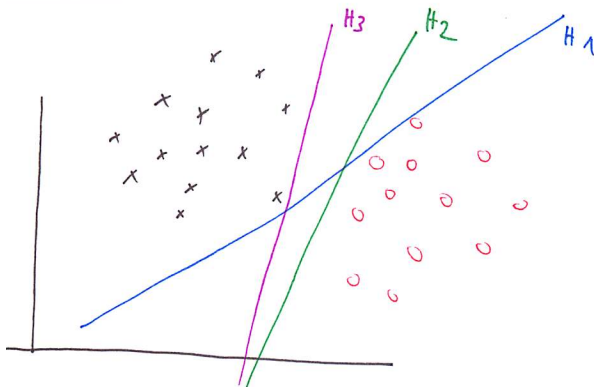
Prelude

The support vector machine (SVM) is the algorithm that made machine learning its own sub-discipline of computer science, it is one of the most important machine learning algorithms. It has been published in the late 1990ies (see later for more on history).

We are going to study the linear case first. The main power of the method comes from the “kernel trick” which is going to make them non-linear.

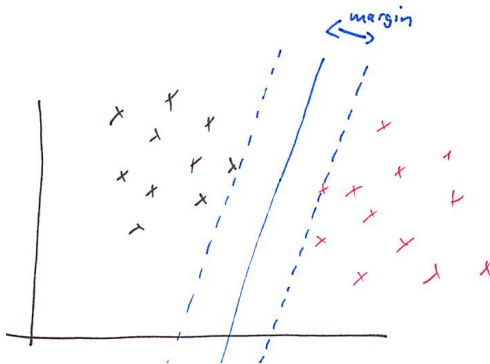
Geometric motivation

Given a set of linearly separable data points in \mathbb{R}^d . Which hyperplane to take???



Geometric motivation (2)

Idea: take the hyperplane with the largest distance to both classes ("large margin"):



Why might this make sense?

Geometric motivation (3)

Why might this make sense:

- ▶ Robustness: assume our data points are noisy. If we “wiggle” some of the points, then they are still on the same side of the hyperplane, so the classification result is robust on the training points.
- ▶ Later we will see: the size of the margin can be interpreted as a regularization term. The larger the margin, the “less complex” the corresponding function class.

Canonical hyperplane

- ▶ We are interested in a linear classifier of the form

$$f(x) = \text{sign}(\langle w, x \rangle + b)$$

- ▶ Note that if we multiply w and b by the same constant $a > 0$, this does not change the classifier:

$$\text{sign}(\langle aw, x \rangle + ab) = \text{sign}(a(\langle w, x \rangle + b)) = \text{sign}(\langle w, x \rangle + b)$$

- ▶ Want to remove this degree of freedom.
- ▶ For now, assume data can be perfectly separated by hyperplane.

We say that the pair (w, b) is in **canonical form** with respect to the points x_1, \dots, x_n if they are scaled such that

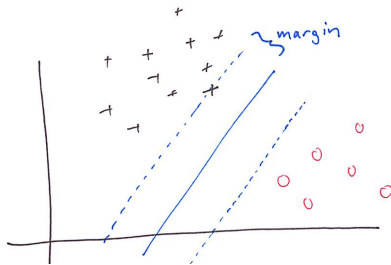
$$\min_{i=1, \dots, n} |\langle w, x_i \rangle + b| = 1$$

We also say that the **hyperplane is in canonical representation**.

The Margin

- ▶ Let $H := \{x \in \mathbb{R}^d \mid \langle w, x \rangle + b = 0\}$ be a hyperplane.
- ▶ Assume that a hyperplane correctly separates the training data.
- ▶ The **margin of the hyperplane H with respect to the training points** $(X_i, Y_i)_{i=1, \dots, n}$ is defined as the minimal distance of a training point to the hyperplane:

$$\rho(H, X_1, \dots, X_n) := \min_{i=1, \dots, n} d(X_i, H) := \min_{i=1, \dots, n} \min_{h \in H} \|X_i - h\|$$



The Margin (2)

Proposition 11 (Margin)

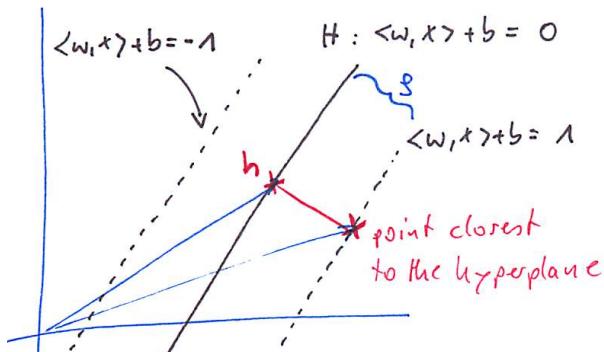
For a hyperplane in canonical representation, the margin ρ can be computed by $\rho = 1/\|w\|$.

First proof.

Observe:

- ▶ Points on the hyperplane itself satisfy $\langle w, x \rangle + b = 0$.
(Reason: definition of the hyperplane)
- ▶ Points that sit on the margin satisfy $\langle w, x \rangle + b = \pm 1$.
(Reason: canonical representation)
- ▶ Let x be the training point that is closest to the hyperplane (that is, the one that defines the margin). W.l.o.g. assume $\langle w, x \rangle + b = +1$ (the case with -1 works similarly). Let $h \in H$ the closest point to x on the hyperplane. Then $\|x - h\| = \rho$.

The Margin (3)



The Margin (4)

We also know that

$$x = h + \rho \frac{w}{\|w\|}$$

because the line connecting x and h is in the normal direction w and has length ρ .

Now we build the scalar product with w and add b on both sides:

$$\implies \langle w, x \rangle = \langle w, h + \rho \frac{w}{\|w\|} \rangle = \langle w, h \rangle + \rho \frac{\|w\|^2}{\|w\|}$$

$$\implies \underbrace{\langle w, x \rangle}_{=1} + b = \underbrace{\langle w, h \rangle}_{=0} + b + \rho \|w\|$$

$$\implies \rho = 1/\|w\|$$



The Margin (5)

Alternative proof:

By definition, the margin is $\rho = \|X - h\|$. In order to compute it, observe that

$$\langle w, x \rangle + b = 1$$

$$\langle w, h \rangle + b = 0$$

Subtracting these two equations and rescaling with $\|w\|$ gives

$$\langle w, x - h \rangle = 1$$

$$\langle w/\|w\|, x - h \rangle = 1/\|w\|$$

Now the proposition follows from the fact that w and $x - h$ point in the same direction and $w/\|w\|$ has norm 1.



Hard margin SVM

So here is our first formulation of the SVM optimization problem:

- Maximize the margin
- Subject to:
 - ▶ all points are on the correct side of the hyperplane
 - ▶ and outside the margin.

In formulas:

$$\begin{aligned} & \text{maximize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \frac{1}{\|w\|} \\ & \text{subject to } Y_i = \text{sign}(\langle w, X_i \rangle + b) \quad \forall i = 1, \dots, n \\ & \quad |\langle w, X_i \rangle + b| \geq 1 \quad \forall i = 1, \dots, n \end{aligned}$$

Hard margin SVM (2)

Usually, we consider the following equivalent optimization problem:

$$\begin{aligned} & \text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \frac{1}{2} \|w\|^2 \\ & \text{subject to} \quad Y_i(\langle w, X_i \rangle + b) \geq 1 \quad \forall i = 1, \dots, n \end{aligned}$$

This problem is called the (primal) hard margin SVM problem.

Hard margin SVM (3)

First remarks:

- ▶ This optimization problem is convex.
- ▶ In fact, it is a quadratic optimization problem (objective function is quadratic, constraints are linear).
- ▶ Observe that the solution will always be a hyperplane in canonical form. EXERCISE.
- ▶ The only reason to add constant $1/2$ in front of $\|w\|^2$ is for mathematical convenience (the derivative is then w and not $2w$). Sometimes we also drop it later.

Hard margin SVM (4)

However, big disadvantage:

This problem only has a solution if the data set is linearly separable, that is there exists a hyperplane H that separates all training points without error. This might be too strict ...

Soft margin SVM

- ▶ We want to allow for the case that the separating hyperplane makes some errors (that is, it does not perfectly separate the training data).
- ▶ To this end, we introduce “slack variables” ξ_i and consider the following new optimization problem:

$$\begin{aligned} \text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^n} \quad & \frac{1}{2} \|w\|^2 + \frac{C}{n} \sum_{i=1}^n \xi_i \\ \text{subject to} \quad & Y_i(\langle w, X_i \rangle + b) \geq 1 - \xi_i \quad \forall i = 1, \dots, n \\ & \xi_i \geq 0 \quad \forall i = 1, \dots, n \end{aligned}$$

Here C is a constant that controls the tradeoff between the two terms, see below.

This problem is called the **(primal) soft margin SVM problem**.

- ▶ Note that this is a convex (quadratic) problem as well.

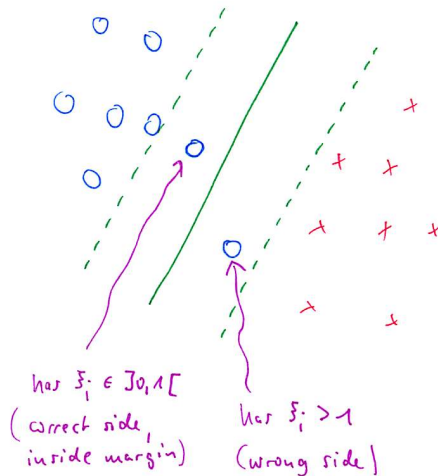
Soft margin SVM (2)

Interpretation:

- ▶ If $\xi_i = 0$, then the point X_i is on the correct side of the hyperplane, outside the margin.
- ▶ If $\xi_i \in]0, 1[$, then X_i is still on the correct side of the hyperplane, but inside the margin.
- ▶ If $\xi_i > 1$, then X_i is on the wrong side of the hyperplane.

Note that for soft SVMs, the margin is defined implicitly (the points on the margin are the ones that satisfy $\langle w, x \rangle + b = \pm 1$).

Soft margin SVM (3)



SVM as regularized risk minimization

We want to interpret the SVM in the regularization framework:

$$\begin{array}{ccc} \text{minimize} & \underbrace{\frac{1}{2} \|w\|^2}_{\leadsto \text{Regularization term}} & + \frac{C}{n} \underbrace{\sum_{i=1}^n \xi_i}_{\leadsto \text{Risk term}} \end{array}$$

To this end, we want to incorporate the constraints into the objective to form a new loss function:

SVM as regularized risk minimization (2)

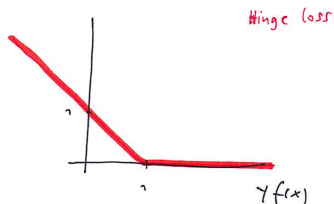
Consider the constraint $Y_i(\langle w, X_i \rangle + b) \geq 1 - \xi_i$. Exploiting $\xi_i \geq 0$ we can rewrite it as follows:

$$\xi_i \geq \max\{0, 1 - Y_i(\langle w, X_i \rangle + b)\}$$

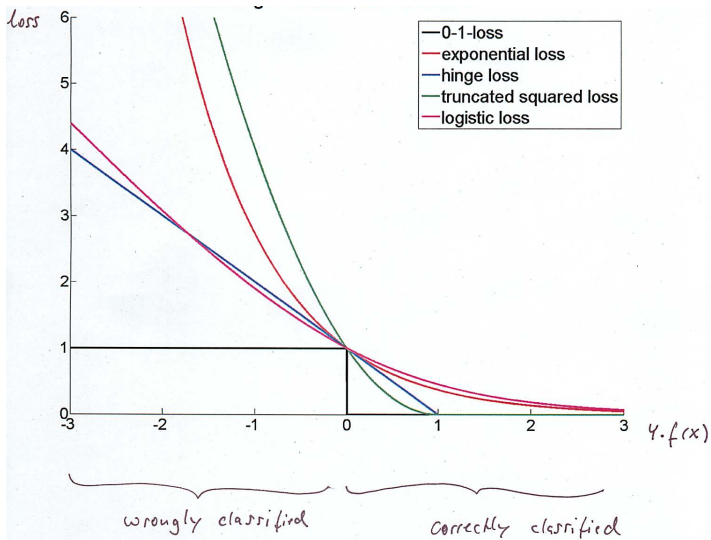
This is a loss function, the so called **Hinge loss**:

$$\ell(x, y, f(x)) = \max\{0, 1 - yf(x)\}$$

It looks as follows:



SVM as regularized risk minimization (3)



SVM as regularized risk minimization (4)

The hinge loss function of SVMs has a couple of interesting properties:

- ▶ It even punishes points if they have the correct label but are too close to the decision surface (the margin).
- ▶ For points on the wrong side it increases linearly, like an L_1 -norm, not quadratic.

SVM as regularized risk minimization

With the hinge loss function, we can now interpret the soft margin SVM as regularized risk minimization:

$$\underset{w,b}{\text{minimize}} \quad \underbrace{\frac{C}{n} \sum_{i=1}^n \max\{0, 1 - Y_i(\langle w, X_i \rangle + b)\}}_{\text{Empirical risk wrt Hinge loss}} + \underbrace{\|w\|^2}_{L_2\text{-regularizer}}$$

The constant C plays the “inverse role” of the regularization constant λ we used in the previous problems (just multiply the objective with $1/C$ and replace $1/C$ by λ).

It is a convention that we use C in SVMs, not λ ...

SVM as regularized risk minimization (2)

EXERCISE: what happens if C is chosen very small, what if C is chosen very large?

Summary so far: Linear SVM (primal)

What we have seen so far:

- ▶ The linear SVM tries to maximize the margin between the two classes.
- ▶ The hard margin SVM only considers solutions without training errors. The soft margin SVM can trade-off margin errors or misclassification errors with a large margin.
- ▶ Both hard and soft SVM are quadratic optimization problems (in particular, convex).
- ▶ The soft margin SVM can be interpreted as regularized risk minimization with the Hinge loss function and L_2 -regularization.

Excursion: convex optimization, primal, Lagrangian, dual

see slides 1387ff. in the appendix

Deriving the dual problem

Dual of hard margin SVM

It turns out that many of the important properties of SVM can easily be derived from the dual optimization problem.

So let us derive the dual problem:

Dual of hard margin SVM (2)

Primal problem (the one we start with):

$$\begin{aligned} & \text{minimize}_{w \in \mathbb{R}^d, b \in \mathbb{R}} \quad \frac{1}{2} \|w\|^2 \\ & \text{subject to} \quad Y_i(\langle w, X_i \rangle + b) \geq 1 \quad \forall i = 1, \dots, n \end{aligned}$$

Lagrangian: we introduce one Lagrange multiplier $\alpha_i \geq 0$ for each constraint and write down the Lagrangian:

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (Y_i(\langle w, X_i \rangle + b) - 1)$$

Dual of hard margin SVM (3)

Formally, the dual problem is the following:

Dual function:

$$g(\alpha) = \min_{w,b} L(w, b, \alpha)$$

Dual Problem:

$$\begin{aligned} & \underset{\alpha}{\text{maximize}} && g(\alpha) \\ & \text{subject to} && \alpha_i \geq 0, \quad i = 1, \dots, n \end{aligned}$$

But this is pretty abstract, we would need to first compute the dual function, but this seems non-trivial. We now show how to compute $g(\alpha)$ explicitly. Let's try to simplify the Lagrangian first.

Dual of hard margin SVM (4)

Saddle point condition: We know that at the solution of the primal, the saddle point condition has to hold:

In particular,

$$\frac{\partial}{\partial b} L(w, b, \alpha) = - \sum_{i=1}^n \alpha_i Y_i \stackrel{!}{=} 0 \quad (*)$$

$$\frac{\partial}{\partial w} L(w, b, \alpha) = w - \sum_i \alpha_i Y_i X_i \stackrel{!}{=} 0 \quad (**)$$

Dual of hard margin SVM (5)

Rewrite the Lagrangian: We plug (*) and (**) in the Lagrangian at the saddle point (w^*, b^*, α^*) :

- First exploit (*):

$$\begin{aligned} L(w, b, \alpha) &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (Y_i (\langle w, X_i \rangle + b) - 1) \\ &= \frac{1}{2} \|w\|^2 + \sum_i \alpha_i - \sum_i \alpha_i Y_i \langle w, X_i \rangle - b \underbrace{\sum_i \alpha_i Y_i}_{=0 \text{ by } (*)} \end{aligned}$$

- Now we replace w by formula (**) and get after simplification:

$$L(w^*, b^*, \alpha^*) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j Y_i Y_j \langle X_i, X_j \rangle$$

- Observe: $L(w, b, \alpha)$ does not depend on w and b any more!

Dual of hard margin SVM (6)

Dual function:

So at the saddle point (w^*, b^*, α^*) , the dual function is very simple:

$$\begin{aligned} g(\alpha) &:= \min_{w,b} L(w, b, \alpha) \\ &= \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j Y_i Y_j \langle X_i, X_j \rangle \end{aligned}$$

(we can drop the “ $\min_{w,b}$ ” because w, b have disappeared).

Dual of hard margin SVM (7)

To finally write down the dual optimization problem, we have to keep enforcing (*) and (**) (otherwise the transformation of the Lagrangian to its simpler form is no longer valid).

- ▶ By now, (**) is an empty condition, because the primal variables w (and b) have disappeared already (implicitly, this condition has already been “built in” the dual function).
- ▶ But we need to keep enforcing the condition (*) in the dual problem, because it involves the dual variables α_i .

Dual of hard margin SVM (8)

So finally we end up with the **dual problem** of the linear hard margin SVM:

$$\underset{\alpha \in \mathbb{R}^n}{\text{maximize}} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j Y_i Y_j \langle X_i, X_j \rangle$$

$$\text{subject to } \alpha_i \geq 0 \quad \forall i = 1, \dots, n$$

$$\sum_{i=1}^n \alpha_i Y_i = 0$$

Dual of the soft margin SVM

Analogously, one can derive the dual problem of the soft margin SVM, it looks nearly the same:

$$\begin{aligned} & \underset{\alpha \in \mathbb{R}^n}{\text{maximize}} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j Y_i Y_j \langle X_i, X_j \rangle \\ & \text{subject to} \quad 0 \leq \alpha_i \leq C/n \quad \forall i = 1, \dots, n \\ & \quad \quad \quad \sum_{i=1}^n \alpha_i Y_i = 0 \end{aligned}$$

Dual SVM in practice

- ▶ Given the input data, compute all the scalar products $\langle X_i, X_j \rangle$
- ▶ Solve the dual optimization problem (it is convex), this gives you the α_i .
- ▶ To compute the class label of a test point X , we need to understand how we can recover the primal variables w and b from the dual variables α . This works as follows.

Dual SVM in practice (2)

Recover the primal optimal variables w, b from the dual solution α :

- ▶ To compute w : directly use $(**)$: $w = \sum_i \alpha_i Y_i X_i$
- ▶ To compute b , we need to exploit the KKT conditions of the soft margin SVM. As we did not put all details of the soft margin derivation on the slides, here is the summary:
 - (i) $\alpha_i = 0$ implies that the slack variable $\xi_i = 0$ and that the point X_i is outside of the margin and correctly classified.
 - (ii) $0 < \alpha_i < C/n$ implies that the corresponding point sits exactly on the margin. In particular, we then have $Y_i(\langle w, X_i \rangle + b) = 1$.
 - (iii) $\alpha_i = C/n$ implies that the slack variable $\xi_i > 0$. The corresponding points sit either inside the margin (still on the correct side) or on the wrong side of the hyperplane.

Dual SVM in practice (3)

To compute b , we thus select a point X_j on the margin as in case (ii) above, that is an index j with $0 < \alpha_j < C/n$, and then solve $Y_j(\langle w, X_j \rangle + b) = 1$ for b .

To increase numerical stability, we might use all such points X_j and average the resulting values of b .

EXERCISE: USE THE LAGRANGE APPROACH TO DERIVE THE DUAL OF THE SOFT MARGIN SVM, AND USE THE KKT CONDITIONS TO VERIFY THE THREE CASES (i)-(iii) ABOVE.

Dual SVM in practice (4)

Now we can evaluate the label of a test point X :

$$Y := \text{sign}(\langle w, X \rangle + b)$$

with w and b as on the previous slide:

$$\begin{aligned}\langle w, X \rangle + b &= \left\langle \sum_i \alpha_i Y_i X_i, X \right\rangle + b \\ &= \sum_i \alpha_i Y_i \langle X_i, X \rangle + \\ &\quad + \frac{1}{|J|} \sum_{j \in J} \left(Y_j - \sum_i Y_i \alpha_i \langle X_i, X_j \rangle \right)\end{aligned}$$

where $J = \{j | 0 < \alpha_j < C/n\}$. Note that this formula just depends on the α_i .

In practice: solve the primal or the dual?

Because we know that for quadratic problems we have strong duality, we could either solve the primal or the dual problem. Which one?

Hard margin case:

- ▶ The primal problem has $d + 1$ variables (where d is the dimension of the space), and n constraints (where n is the number of training points). If d is small compared to n , then it makes sense to solve the primal problem.
- ▶ The dual problem has n variables and $n + 1$ constraints. If d is large compared to n , then it is better to solve the dual problem.

In practice: solve the primal or the dual? (2)

Soft margin case:

- ▶ Primal has $d + 1 + n$ variables and $2n$ constraints.
- ▶ Dual has n variables and $2n + 1$ constraints
- ▶ So one would typically choose the dual problem. In most SVM libraries, solving the dual is the default.

Important properties of SVMs

Solution as linear combination

Representation of the solution: From (**) we see immediately that the solution vector w can always be expressed as a linear combination of the input points: $w = \sum_i \alpha_i Y_i X_i$. This is very important for the kernel version of the algorithm (\leadsto representer theorem, see later).

Support vectors

Support vector property:

- KKT conditions in the hard margin case tell us: Only Lagrange multipliers α_i that are non-zero correspond to active constraints (the ones that are precisely met). Formally,

$$\alpha_i (Y_i f(X_i) - 1) = 0$$

A similar statement holds for the soft margin case, there the α_i are only non-zero for points on the margin, in the margin, or on the wrong side of the margin.

- In our context: Only those α_i are non-zero that correspond to points that lie exactly on the margin, inside the margin or on the wrong side of the hyperplane. The corresponding points are called **support vectors**.

Support vectors (2)

- ▶ So the solution can be expressed just by the coefficients of the support vectors.
- ▶ In low-dimensional spaces this property means that we have a **sparse solution vector** w . But note that sparsity is not necessarily true in very high-dimensional spaces (then essentially all points sit on the margin).

Scalar products

We can see that all the information about the input points X_i that enters the optimization problem is expressed in terms of scalar products:

- ▶ $\langle X_i, X_j \rangle$ in the dual objective function
- ▶ $\langle X, X_i \rangle$ and $\langle X_i, X_j \rangle$ in the evaluation of the target function on a new point X

This is going to be the key point to be able to apply the kernel trick.

Exercise

It might be instructive to solve the following exercise:

Input data: $x_1 = (1, 0)$; $y_1 = +1$; $x_2 = (-1, 0)$; $y_2 = -1$.

Primal problem:

- ▶ Write down the hard margin primal optimization problem and solve it using the Lagrange approach.
- ▶ Write down the soft margin primal optimization problem and solve it using the Lagrange approach.

Dual problem:

- ▶ Write down the dual hard margin optimization problem and solve it.
- ▶ Write down the dual soft margin primal optimization problem and solve it.

Exercise (2)

- ▶ Use the dual solution to recover the solution of the primal problem. Compare the values of the objective functions at the dual and primal solution.
- ▶ Determine the support vectors.

History

- ▶ **Vladimir Vapnik** is the “inventor” of the SVM (and, in fact, he laid the foundations of statistical learning theory in general).
- ▶ The hard margin SVM and the kernel trick was introduced by *Boser, Bernhard; Guyon, Isabelle; and Vapnik, Vladimir. A training algorithm for optimal margin classifiers. Conference on Learning Theory (COLT), 1992*
- ▶ This was generalized to the soft margin SVM by *Cortes, Corinna and Vapnik, Vladimir. "Support-Vector Networks", Machine Learning, 20, 1995.*

Summary: linear SVM

- ▶ Input data: $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{\pm 1\}$
- ▶ Function class: linear functions of the form $f(x) = \langle w, x \rangle + b$
- ▶ Want to select hyperplane as to maximize the margin
- ▶ Soft margin SVM has interpretation as regularized risk minimization with respect to the Hinge loss and with L_2 -regularization
- ▶ Is a quadratic optimization problem
- ▶ Convex duality leads to the following key properties of the solution:
 - ▶ Solution w^* can always be expressed as linear combination of input points
 - ▶ Sparsity: only points that are on, in or on the wrong side of the margin contribute to this linear combination
 - ▶ To compute and evaluate the solution, all we need are scalar products of input points.

Kernel methods for supervised learning

Positive definite kernels

Introductory literature:

- ▶ Schölkopf / Smola Section 2
- ▶ Shawe-Taylor / Cristianini Section 2 and 3

For a deeper mathematical treatment of kernels see the following book:

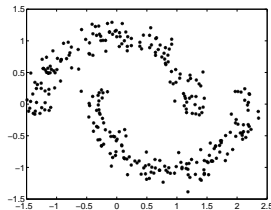
- ▶ Steinwart, Christmann: Support Vector Machines. Springer, 2008.

Intuition

Linear methods — disadvantages

We have seen several linear methods for regression and classification. Even though these methods are conceptually appealing, they have a number of disadvantages.

- ▶ Linear functions are restrictive. This can be of advantage to avoid overfitting, but often it leads to underfitting. For example, in classification we could not find any hyperplane to separate the following example:

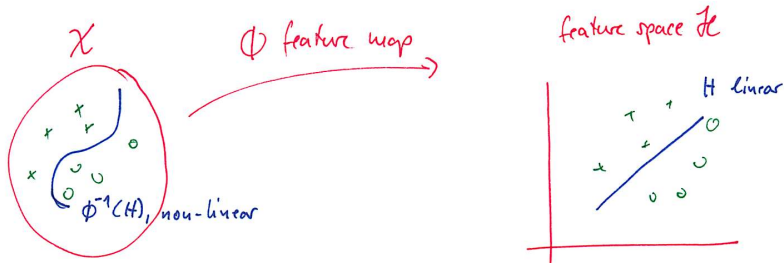


Linear methods — disadvantages (2)

- ▶ Alternatively, we could use a feature map with basis functions Φ_i to represent more complex functions, say polynomials. But:
 - ▶ It is not so obvious which are "good" basis functions.
 - ▶ We need to fix the basis before we see the data. This means that we need to have very many basis functions to be flexible. This leads to a very high dimensional representations of our data.

Linear methods — disadvantages (3)

The goal of kernel methods is to introduce a non-linear component to linear methods:



Starting point: Key observation for SVMs

To run the linear support vector machine algorithm, we do not need to compute $\Phi(X)$ explicitly — all we need to know are scalar products of the form $\langle \Phi(X_i), \Phi(X_j) \rangle$:

- ▶ The dual objective function only contains terms of the form $\langle X_i, X_j \rangle$, the X_i never occur “alone”.
- ▶ To evaluate the solution at the test point, again we only need to be able to compute scalar products of the input, we never need to know coordinates of the input points.

Starting point: Key observation for SVMs (2)

Let us be more explicit:

- ▶ Assume the data lives in \mathbb{R}^d .
- ▶ Introduce the shorthand notation $k(x, y) := \langle x, y \rangle$.

Then we can write the SVM optimization problem purely in terms of the function k (the X_i never occur outside the function k):

$$\begin{aligned} & \underset{\alpha \in \mathbb{R}^n}{\text{maximize}} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j Y_i Y_j k(X_i, X_j) \\ & \text{subject to} \quad 0 \leq \alpha_i \leq C/n \quad \forall i = 1, \dots, n \\ & \quad \quad \quad \sum_{i=1}^n \alpha_i Y_i = 0 \end{aligned}$$

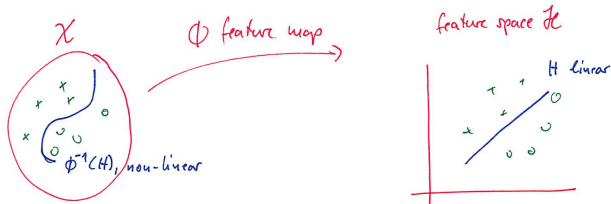
(and the same goes for the function that evaluates the results on test points).

Idea: Kernels replacing feature maps

Assume we are in a feature mapping scenario, but we know how to compute scalar products explicitly, that is we know a function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ with

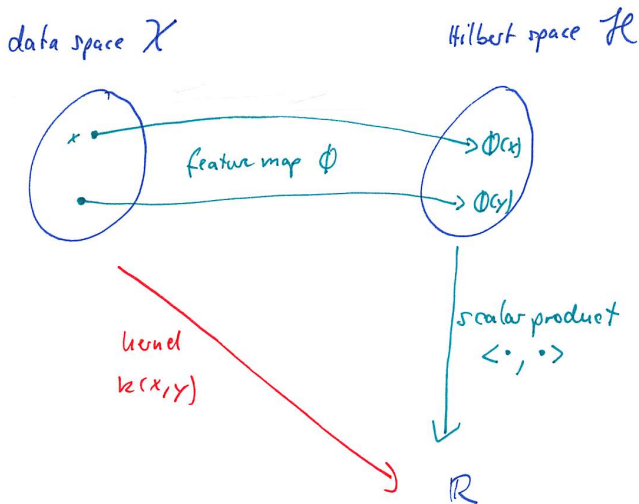
$$k(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle.$$

The idea is that it might even be possible to avoid computing the embeddings $\Phi(X_i)$ and compute the scalar products directly via the function k .



Idea: Kernels replacing feature maps (2)

This is what we want to achieve:



Intuition: kernels as similarity functions

- ▶ The scalar product can be interpreted as a measure of how similar two points are.
- ▶ We want to use the same intuition for a kernel: **The kernel is a measure of how “similar” two points in the feature space are.**

Intuition: kernels as similarity functions (2)

Intuition: direct features vs kernel approach:

Direct feature based approach:

describe each “entity” by individual features.

Example: person is described by age, income, family status, etc.

Kernel approach:

a kernel describes the similarity between two points.

Example: how close to persons are in a social network (see graph kernels below).

So the two approaches focus on different aspects. A direct feature approach treats each point individually, a kernel focuses on the relation between two points.

Kernel methods — the overall picture

What we want to do:

- ▶ Given points in some abstract space \mathcal{X}
- ▶ Would like to (implicitly) embed the points into some space \mathbb{R}^d via a (non-linear) feature map Φ
- ▶ In that space, we use a linear method like an SVM
- ▶ Ideally, we never compute the embedding directly.
- ▶ Instead we want to use a “kernel function ” to compute

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle.$$

This approach is called the “kernel trick” and the corresponding algorithms are called “kernel methods”.

Kernel methods — the overall picture (2)

In the following we try to make this idea formal.

- ▶ How do the functions k need to look like? (\leadsto kernels)
- ▶ Once we have an appropriate k , what is the corresponding feature map? (\leadsto RKHS)

Definition and properties of kernels

Kernel function — definition

Let \mathcal{X} be any space. A symmetric function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is called a **kernel function** if for all $n \geq 1$, $x_1, x_2, \dots, x_n \in \mathcal{X}$ and $c_1, \dots, c_n \in \mathbb{R}$ we have

$$\sum_{i,j=1}^n c_i c_j k(x_i, x_j) \geq 0.$$

Given a set of points $x_1, \dots, x_n \in \mathcal{X}$, we define the corresponding **kernel matrix** as the matrix K with entries $k_{ij} = k(x_i, x_j)$.

The condition above is equivalent to saying that $c' K c \geq 0$ for all $c \in \mathbb{R}^n$.

Kernel function — definition (2)

Remarks:

- It is NOT true that a function that satisfies $k(x, y) \geq 0$ for all $x, y \in \mathcal{X}$ is positive definite!!!

EXERCISE: FIND A COUNTEREXAMPLE (try to construct a matrix with positive entries that is not pd).

- In the maths literature, the above condition would be called “positive semi-definite” (and it would be called “positive definite” only if the inequality is strict).

Scalar products lead to kernels

Observe:

For any mapping $\Phi : \mathcal{X} \rightarrow \mathbb{R}^d$ the function defined (!) by

$$k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}, \quad k(x, y) = \langle \Phi(x), \Phi(y) \rangle$$

is a valid kernel!

Proof sketch:

- ▶ Symmetry: clear
- ▶ Positive definiteness: follows from the positive definiteness of the scalar product, EXERCISE!

Scalar products lead to kernels (2)

In this case, the kernel matrix is given as follows:

- ▶ Let $X_1, \dots, X_n \in \mathcal{X}$ be data points, $\Phi : \mathcal{X} \rightarrow \mathbb{R}^d$ a feature map.
- ▶ Denote by Φ the $n \times d$ -matrix that contains the data points $\Phi(X_i)$ as rows.
- ▶ Then the matrix $\Phi \cdot \Phi^t \in \mathbb{R}^{n \times n}$ coincides with the corresponding kernel matrix K with entries $k_{ij} = \langle \Phi(X_i), \Phi(X_j) \rangle = \Phi(X_i) \Phi(X_j)^t$.

Example: linear kernel

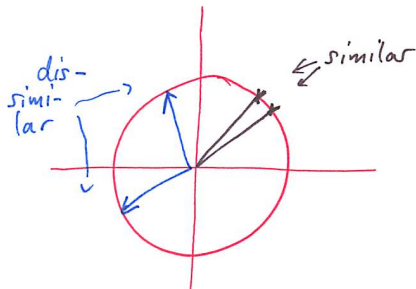
The linear kernel. The trivial kernel on \mathbb{R}^d defined by the standard scalar product:

$$k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}, \quad k(x, y) = \langle x, y \rangle$$

Is obviously a kernel.

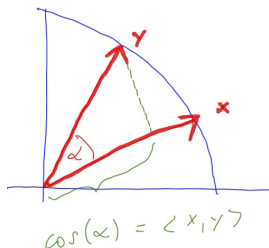
Example: cosine similarity

- ▶ Assume your data lives in \mathbb{R}^d and is normalized such that all data points have (roughly) norm 1. Then they sit on the hypersphere (surface of the ball of radius 1).
- ▶ Points are similar if the corresponding vectors “point to the same direction”.



Example: cosine similarity (2)

- ▶ As a measure how similar the points are, we use the cosine of the angle between the two points.
 - ▶ $\cosine = 1 \iff$ points agree
 - ▶ $\cosine = 0 \iff$ points are orthogonal
- ▶ If the x and y are normalized, then the cosine of the angle between x and y is given by the scalar product $\langle x, y \rangle$.



So the cosine similarity is obviously a kernel.

Example: cosine similarity (3)

Cosine similarity is typically used for counting data, for example a bag-of-words representation for texts. Then all points sit in the first quadrant (just positive counts, no negative entries). We normalize points because then we are not so much influenced by the length of the text. In particular, then the cosine similarity is always ≥ 0 .

Example: the Gaussian kernel

On \mathbb{R}^d , define the following kernel:

$$k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}, \quad k(x, y) = \exp \left(\frac{-\|x - y\|^2}{2\sigma^2} \right)$$

where $\sigma > 0$ is a parameter.

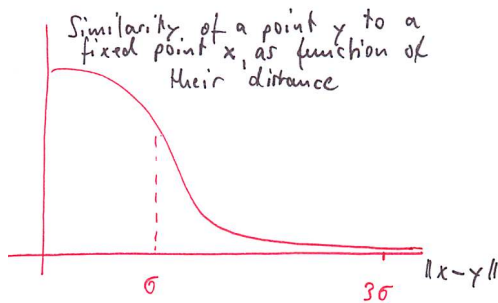
One can prove that this is indeed a kernel, this is not obvious at all!!! (WHAT DO WE NEED TO PROVE?).

See the text books if you are interested.

Note: the Gaussian kernel is also called **rbf-kernel** for “radial basis function”.

Example: the Gaussian kernel (2)

Induced notion of similarity:



Two points are considered “very similar” if they are of distance at most σ , “somewhat similar” if they are at distance (roughly) at most 3σ , and “pretty dissimilar” if they are further away than that.

Example: polynomial kernel

$$\mathcal{X} = \mathbb{R}^d.$$

$$k(x, y) = (x'y + c)^k$$

where $c > 0$ and $k \in \mathbb{N}$.

Not very useful for practice, but often mentioned, hence I put it on the slides.

Example: kernels based on explicit feature maps

- ▶ Assume we explicitly constructed a feature space embedding such as a bag-of-words representation for texts or a bag-of-motifs representation of graphs.
- ▶ Then simply use the linear kernel in the feature space \mathbb{R}^d .

Induced similarity functions:

- ▶ books are considered “similar” if they get bought by the same users.
- ▶ Graphs are considered “similar” if they contain the same motifs.
- ▶ etc ...

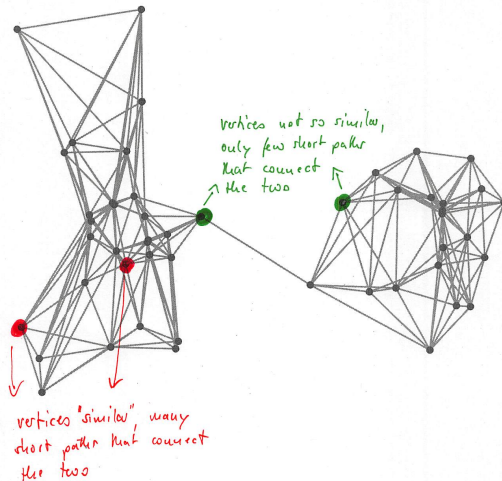
Example: kernel between vertices in a graph

Application scenario:

- ▶ Say we want to classify persons in a social network, whether they prefer samsung or apple phones. All we know about the persons are their friendships.
- ▶ We consider people as “similar” if they have similar sets of friends. We want to encode this notion of similarity by kernel function.
- ▶ Then we classify with an SVM.

Example: kernel between vertices in a graph (2)

There exists a large number of graph kernels. One big family is based on paths between vertices:



Example: kernel between vertices in a graph (3)

To define a kernel between vertices on a graph:

- ▶ Consider a directed graph with edge weights in $[0, 1]$, where this value encodes a similarity (high means very similar). For a directed path $\pi = v_1, \dots, v_k$ define the weight of the path as $w(\pi) = \prod_{j=1}^{k-1} w(v_j, v_{j+1})$
- ▶ For each pair of vertices v, \tilde{v} consider the set $\Pi_k(v, \tilde{v})$ which consists of all paths from v to \tilde{v} of lengths at most k
- ▶ Now define

$$s(v, \tilde{v}) = \begin{cases} \sum_{\pi \in \Pi_k} w(\pi) & \text{if } \Pi_k(v, \tilde{v}) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

and the symmetric kernel function

$$k(v, \tilde{v}) = s(v, \tilde{v}) + s(\tilde{v}, v)$$

Example: kernel between vertices in a graph (4)

- ▶ Note that in general this kernel cannot be interpreted in terms of a simple feature vector!
- ▶ This principle leads to the family of **diffusion kernels**, we won't discuss details.

Simple rules for dealing with kernels

- ▶ In general, it is really difficult to prove that a certain function k is indeed a kernel (WHAT DO WE HAVE TO PROVE?)
- ▶ In practice, it usually does not work to come up with a nice similarity function and “hope” that it is a kernel.
- ▶ But at least, there are some simple rules that can help to transform and combine elementary kernels:

Simple rules for dealing with kernels (2)

Assume that $k_1, k_2 : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ are kernel functions. Then:

- ▶ $\tilde{k} = \alpha \cdot k_1$ for some constant $\alpha > 0$ is a kernel.
- ▶ $\tilde{k} = k_1 + k_2$ is a kernel
- ▶ $\tilde{k} = k_1 \cdot k_2$ is a kernel
- ▶ The pointwise limit of a sequence of kernels is a kernel.
- ▶ For any function $f : \mathcal{X} \rightarrow \mathbb{R}$, the expression $\tilde{k}(x, y) := f(x)k(x, y)f(y)$ defines a kernel.

In particular, $\tilde{k}(x, y) = f(x)f(y)$ is a kernel.

Proof. EXERCISE.

(*) Kernel matrix: pd or psd?

Due to a common confusion, let me stress again:

- ▶ A scalar product is positive definite. This means that the property $\langle v, v \rangle > 0$ holds (with strict inequality!) for all $v \neq 0$
- ▶ The kernel matrix is positive semi-definite in the sense that $c'Kc \geq 0$ (greater or equal!).

WHY IS THIS NOT A CONTRADICTION?

(*) Kernel matrix: pd or psd? (2)

- ▶ Consider data $X_1, \dots, X_n \in \mathbb{R}^d$.
- ▶ Then the kernel matrix coincides with $K = XX^t$.
- ▶ Let v be an eigenvector of K . We have

$$v'XX'v = v'Kv = \lambda v'v = \lambda$$

- ▶ For eigenvectors with $\lambda > 0$: fine.
- ▶ For eigenvectors with $\lambda = 0$: Then $X'v = 0$, and the scalar product of the 0-vector with itself is 0. Fine as well.

In particular: The rank of the kernel matrix is at most the dimension of the underlying vector space. So if $n > d$, the kernel matrix must have eigenvalues 0. EXERCISE!

Reproducing kernel Hilbert space and feature maps

Literature:

Mohri et al, Foundations of machine learning, Chapter 5.2

Steinwart/Christmann: Support Vector Machines, Section 4

Kernels do what they are supposed to do

Here is the justification for why we defined kernels the way we did:

Theorem 12 (Kernel implies embedding)

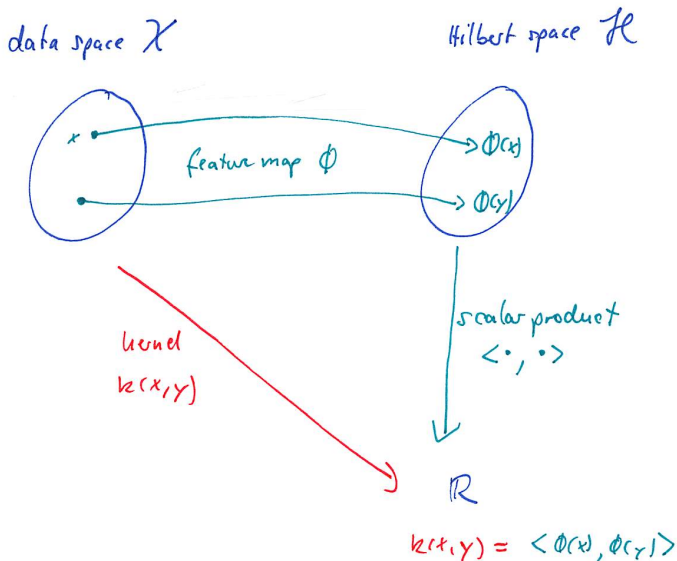
A function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ is a kernel if and only if there exists a Hilbert space \mathcal{H} and a map $\Phi : \mathcal{X} \rightarrow \mathcal{H}$ such that

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle.$$

If you have never heard of Hilbert spaces, just think of the space \mathbb{R}^d . The crucial properties are:

- ▶ \mathcal{H} is a vector space with a scalar product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$
- ▶ Space is complete (all Cauchy sequences converge)
- ▶ Scalar product gives rise to a norm: $\|x\|_{\mathcal{H}} := \langle x, x \rangle_{\mathcal{H}}$

Kernels do what they are supposed to do (2)



Kernels do what they are supposed to do (3)

WHICH DIRECTION OF THE THEOREM IS EASY, WHICH ONE IS DIFFICULT?

Kernels do what they are supposed to do (4)

Proof of " \Leftarrow "

Clear by definition of the kernel (we defined the kernel exactly such that this direction holds).

Kernels do what they are supposed to do (5)

Proof of “ \Rightarrow ”

We have to prove the following:

Given \mathcal{X} and k , there exists a vector space \mathcal{H} with a scalar product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$, and a mapping $\Phi : \mathcal{X} \rightarrow \mathcal{H}$ such that

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle_{\mathcal{H}}$$

for all $x, y \in \mathcal{X}$.

We now introduce the Reproducing Kernel Hilbert Space (RKHS), a scalar product on this space and a corresponding feature mapping Φ . This is going to conclude the proof. 😊

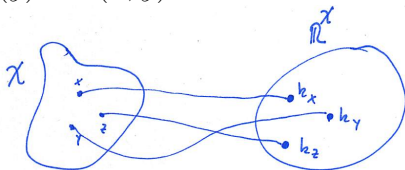
Reproducing kernel Hilbert space (RKHS)

As vector space we are going to use a space of functions:

- Let $\mathbb{R}^{\mathcal{X}}$ be the space of all real-valued functions from \mathcal{X} to \mathbb{R}
- Consider a mapping $\Phi : \mathcal{X} \rightarrow \mathbb{R}^{\mathcal{X}}$, defined as

$$x \mapsto \Phi(x) := k_x := k(x, \cdot)$$

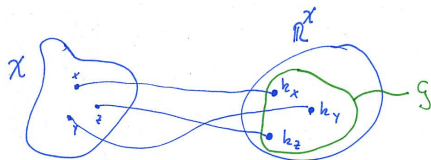
That is, the point $x \in \mathcal{X}$ is mapped to the function $k_x : \mathcal{X} \rightarrow \mathbb{R}$, $k_x(y) = k(x, y)$.



Reproducing kernel Hilbert space (RKHS) (2)

- Now consider the images $\{k_x | x \in \mathcal{X}\}$ as a spanning set of a vector space. That is, we define the space \mathcal{G} that contains all finite linear combinations of such functions:

$$\mathcal{G} := \left\{ \sum_{i=1}^r \alpha_i k(x_i, \cdot) \mid r \in \mathbb{N}, \alpha_1, \dots, \alpha_r \in \mathbb{R}, x_1, \dots, x_r \in \mathcal{X} \right\}$$



Reproducing kernel Hilbert space (RKHS) (3)

- ▶ Define a scalar product on \mathcal{G} as follows:
 - ▶ For the spanning functions we define

$$\langle k_x, k_y \rangle = \langle k(x, \cdot), k(y, \cdot) \rangle := k(x, y)$$

- ▶ For general functions in \mathcal{G} the scalar product is then given as follows: If $g = \sum_i \alpha_i k(x_i, \cdot)$ and $f = \sum_j \beta_j k(y_j, \cdot)$ then

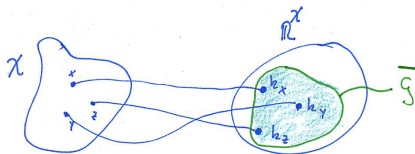
$$\langle f, g \rangle_{\mathcal{G}} := \sum_{i,j} \alpha_i \beta_j k(x_j, y_j)$$

To make sure that this is really a scalar product, we need to prove two things (EXERCISE!):

- ▶ Check that this is well-defined (not obvious because there might be several different linear combinations for the same function).
- ▶ Check that it satisfies all properties of a scalar product (crucial ingredient is the fact that k is positive definite.)

Reproducing kernel Hilbert space (RKHS) (4)

- ▶ Finally, to make \mathcal{G} a proper Hilbert space we need to take its topological completion $\overline{\mathcal{G}}$, that is we add all limits of Cauchy sequences.



- ▶ The resulting space $\mathcal{H} := \overline{\mathcal{G}}$ is called the **reproducing kernel Hilbert space**.
- ▶ By construction, it has the property that

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle.$$

(*) RKHS, further properties

The reproducing property:

Let $f = \sum_i \alpha_i k(x_i, \cdot)$. Then $\langle f, k(x, \cdot) \rangle = f(x)$.

Proof.

$$\begin{aligned}\langle k(x, \cdot), f \rangle &= \langle k(x, \cdot), \sum_i \alpha_i k(x_i, \cdot) \rangle \\ &= \sum_i \alpha_i \langle k(x_i, \cdot), k(x, \cdot) \rangle \\ &= \sum_i \alpha_i k(x_i, x) \\ &= f(x)\end{aligned}$$



(*) RKHS, further properties (2)

For those who know a bit of functional analysis:

- ▶ Let \mathcal{H} be a Hilbert space of functions from \mathcal{X} to \mathbb{R} . Then \mathcal{H} is a reproducing kernel Hilbert space if and only if all evaluation functionals $\delta_x : \mathcal{H} \rightarrow \mathbb{R}, f \mapsto f(x)$ are continuous.
- ▶ In particular, functions in an RKHS are pointwise well defined (as opposed to, say, function in an L_2 -space which are only defined almost everywhere).
- ▶ Given a kernel, the RKHS is unique (up to isometric isomorphisms). Given an RKHS, the kernel is unique.
- ▶ There is a close connection to the Riesz representation theorem.

The representer theorem

- ▶ In general, the RKHS is an infinite-dimensional vector space (a basis has to contain infinitely many vectors).
- ▶ The next theorem shows that in practice, we only have to deal with a finite-dimensional subspace.
- ▶ This subspace is still pretty large, later we discuss how to avoid overfitting.

The representer theorem (2)

Setup:

- ▶ Assume we are given a kernel k . Denote the corresponding RKHS with \mathcal{H} , and the norm and scalar product in the space by $\|\cdot\|_{\mathcal{H}}$ and $\langle \cdot, \cdot \rangle_{\mathcal{H}}$.
- ▶ Assume that we want to learn a linear function $f : \mathcal{H} \rightarrow \mathbb{R}$ that acts on the RKHS \mathcal{H} of a kernel k .
- ▶ Given input argument $z \in \mathcal{H}$, all such functions have the form $f(z) = \langle w, z \rangle_{\mathcal{H}}$ for some $w \in \mathcal{H}$, that is we can identify the function f with the corresponding vector $w \in \mathcal{H}$.
(for maths people: reason is that the dual of a real Hilbert space is isomorphic to this Hilbert space)

In this setup, we can prove the following theorem:

The representer theorem (3)

Theorem 13 (Representer theorem)

Consider a regularized risk minimization problem of the form

$$\underset{w \in \mathcal{H}}{\text{minimize}} R_n(w) + \lambda \Omega(\|w\|_{\mathcal{H}}) \quad (*)$$

where \mathcal{X} arbitrary input space, \mathcal{Y} output space, $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ a kernel, \mathcal{H} the corresponding RKHS. For a given training set $(X_i, Y_i)_{i=1, \dots, n} \subset \mathcal{X} \times \mathcal{Y}$ and classifier $f_w(x) := \langle w, \Phi(x) \rangle_{\mathcal{H}}$, let R_n be the empirical risk of the classifier with respect to a loss function ℓ , and $\Omega : [0, \infty[\rightarrow \mathbb{R}$ a strictly monotonically increasing function. Then problem $(*)$ always has an optimal solution of the form

$$w^* = \sum_{i=1}^n \alpha_i k(X_i, \cdot).$$

The representer theorem (4)

Proof intuition.

- ▶ Split the space \mathcal{H} into the subspace $\mathcal{H}_{data} := \text{span}\{k_{X_1}, \dots, k_{X_n}\}$ (induced by the data) and its orthogonal complement \mathcal{H}_{comp} . Then $\mathcal{H} = \mathcal{H}_{data} + \mathcal{H}_{comp}$.
- ▶ Now express each vector $w \in \mathcal{H}$ as $w = w_{data} + w_{comp}$.
- ▶ It is not difficult to see that the predictions of all functions with the same w_{data} agree on all training points, they do not depend on w_{comp} .
- ▶ So in particular, the loss w is not affected by w_{comp} .
- ▶ For fixed w_{data} , the norm of w is smallest if w_{comp} is 0.
- ▶ So if we had a solution w^* where w_{comp} would be non-zero, we could get a better solution by setting w_{comp} to zero.
- ▶ Thus we can always find an optimal solution with $w_{comp} = 0$.



The representer theorem (5)

Intuitively, this theorem implies the following:

- ▶ We have seen that for any given kernel k there exists a feature space \mathcal{H} .
- ▶ However, this space was a function space that usually is an infinite-dimensional Hilbert space.
- ▶ The representer theorem now says that for any finite data set with n points, we don't need to deal with all the infinitely many dimensions, but we are only confronted with a space of at most n dimensions.
- ▶ As any n -dimensional subspace of a Hilbert space is isomorphic to \mathbb{R}^n , we can simply assume that our feature map goes to \mathbb{R}^n .
- ▶ This makes our lives much easier.

(*) Injective feature map

Note that without any further assumptions, the feature map $\Phi : \mathcal{X} \rightarrow \mathcal{H}$ of a kernel k does not need to be injective!

(Simple counterexample: $k(x, y) = \langle x, y \rangle^2$).

However, a kernel for which the feature map is not injective might not be too useful (WHY?)

A particular class of “nice” kernels are univesal kernels:

(*) Universal kernels

A continuous kernel k on a compact metric space \mathcal{X} is called **universal** if the RKHS \mathcal{H} of k is dense in $C(\mathcal{X})$, that is for every function $g \in C(\mathcal{X})$ and all $\varepsilon > 0$ there exists a function $f \in \mathcal{H}$ such that $\|f - g\|_\infty \leq \varepsilon$.

Intuition: with a universal kernel, we can approximate pretty much any function we like: all continuous functions, and all functions that can be approximated by continuous functions (such as step functions). In particular, we can separate any pair of disjoint compact subsets from each other.

Example:

- The Gaussian kernel with fixed kernel width σ on a compact subset \mathcal{X} of \mathbb{R}^d is universal. Related statements can also be proved if we let $\sigma \rightarrow 0$ slowly as $n \rightarrow \infty$.

(*) Universal kernels (2)

- Polynomial kernels are not universal.

The kernel being universal is a necessary requirement if we want to construct learning algorithms that are uniformly Bayes consistent.

Universal kernels have many nice properties. For example, their feature maps are injective.

For details and proofs see the book by Steinwart / Christmann: Support Vector Machines. Springer 2008.

Kernels — history

- ▶ Reproducing kernel Hilbert spaces play a big role in mathematics, they have been invented by Aronszajn in 1950. He already proved all of the key properties.
Aronszajn. Theory of Reproducing Kernels. Transactions of the American Mathematical Society, 1950
- ▶ The feature space interpretation has first been published by Aizerman 1964, but in a different context. At that time the potential of the method had not been realized.
Aizerman, Braverman, Rozonoer: Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control, 1964.

Kernels — history (2)

- ▶ Then it was (re)discovered in the context of the SVM in 1992: *Boser, Bernhard E.; Guyon, Isabelle M.; and Vapnik, Vladimir N.; A training algorithm for optimal margin classifiers. Conference on Learning Theory (COLT), 1992*
- ▶ Since then, kernels and the kernel trick became extremely popular, the first text books already appeared pretty soon, e.g. Schölkopf / Smola 2002 and Shawe-Taylor / Cristianini 2004.

Kernel algorithms

In the following, we are going to see a couple of algorithms that all use the kernel trick. The roadmap is always the same:

- ▶ Start with a linear algorithm
- ▶ Try to write this algorithm (both training and testing parts) in such a way that the only access to training and testing points is in terms of scalar products (this is often possible but not always; sometimes it is simple, sometimes it is difficult).
- ▶ Then replace the scalar product by the kernel function.

In the machine learning lingo: we **kernelize** the algorithm.

Support vector machines with kernels

Literature:

- ▶ Schölkopf / Smola
- ▶ Shawe-Taylor / Cristianini
- ▶ A very theoretical / mathematically deep treatment of the theory of kernels and support vector machines is the following book:
Steinwart / Christmann: Support Vector Machines. Springer, 2008.

SVMs with kernels

- ▶ Consider the dual (!) SVM problem
- ▶ Have seen: the only way it accesses the training points in terms of scalar products
- ▶ So replace $\langle X_i, X_j \rangle$ by $k(X_i, X_j)$ everywhere
- ▶ The result is the dual of the “kernelized” SVM.

Formally, this looks as follows:

SVMs with kernels (2)

Given input training points $(X_i, Y_i)_{i=1, \dots, n}$ and a kernel function $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.

Kernelized dual SVM problem:

$$\begin{aligned} & \underset{\alpha \in \mathbb{R}^n}{\text{maximize}} \quad \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j Y_i Y_j k(X_i, X_j) \\ & \text{subject to} \quad 0 \leq \alpha_i \leq C/n \quad \forall i = 1, \dots, n \\ & \quad \quad \quad \sum_{i=1}^n \alpha_i Y_i = 0 \end{aligned}$$

Solving this problem gives the dual variables α .

SVMs with kernels (3)

Computing labels at new points: Have already seen how to compute the label of a test point for known α :

- $w = \sum_i \alpha_i Y_i X_i$
- $b = Y_j - \sum_i Y_i \alpha_i \langle X_i, X_j \rangle$ for some j such that $C/n > \alpha_j > 0$.
- Label of test point X given by $\langle w, X \rangle + b$

In kernel language:

$$\begin{aligned}\langle w, X \rangle + b &= \left\langle \sum_i \alpha_i Y_i X_i, X \right\rangle + b \\ &= \sum_i \alpha_i Y_i k(X_i, X) + \left(Y_j - \sum_i Y_i \alpha_i k(X_i, X_j) \right)\end{aligned}$$

This is the approach that is typically used in practice.

The power of kernels

Why is the kernel framework so powerful? Let's look at one particular example, the Gaussian kernel.

- Have seen that the decision function of a kernelized SVM has the form

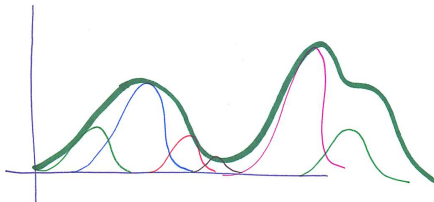
$$f(x) = \sum_i \beta_i k(x, X_i) + b$$

(with $\beta_i = \alpha_i Y_i$). Consider the case where k is a Gaussian kernel:

$$f(x) = \sum_i \beta_i \exp(-\|x - X_i\|^2 / (2\sigma^2))$$

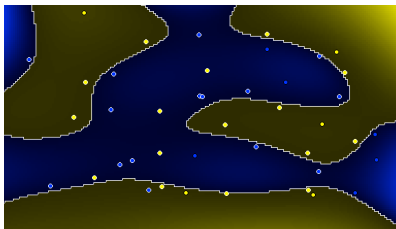
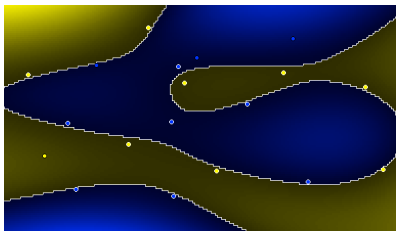
The power of kernels (2)

- Important property: we can approximate any arbitrary continuous function $g : \mathbb{R}^d \rightarrow \mathbb{R}$ by a sum of Gaussian kernels.



In particular, we can approximate any reasonable “decision surface” in \mathbb{R}^d by an SVM with Gaussian kernel:

The power of kernels (3)



The power of kernels (4)

- ▶ Kernels with this property are called “universal” kernels. One can prove that SVMs with universal kernels are universally consistent in the sense we defined in the very beginning of the lecture, that is they approximate the Bayes risk.
- ▶ Note: if the kernel is universal, the underlying function class is huge (all continuous functions can be approximated). All the more important is that we regularize!

Regularization interpretation

Recall that we interpreted the linear primal SVM problem in terms of regularized risk minimization where the risk was the Hinge loss and we regularized by L_2 -regularizer $\|w\|^2$.

How does it look for the kernelized SVM?

- ▶ The loss function is still the Hinge loss because the part with the variables ξ_i does not change.
- ▶ But the regularizer is now $\|w\|^2$ where w is a vector in the feature space, and the norm is taken in the feature space.
- ▶ By the representer theorem,

$$\|w\|^2 = \left\langle \sum_i \beta_i \Phi(X_i), \sum_j \beta_j \Phi(X_j) \right\rangle = \beta^t K \beta$$

Regularization interpretation (2)

- It is not so easy to gain intuition about this norm (obviously it depends on the kernel). But at least we can say that the regularization “restricts the size of the function space” (in the sense that there are fewer functions that can be expressed with w with low norm than with high norm).

Regularization interpretation (3)

This regularization interpretation is really important, otherwise the SVM could not work (in the classical, underparameterized regime):

- ▶ We implicitly embed our data in a very high-dimensional space.
- ▶ In high-dimensional spaces, it happens very easily that we overfit.
- ▶ The only way we can circumvent this is to regularize.
- ▶ This is what the kernelized SVM does.

Kernel SVMs in practice

If you want to use SVMs in practice, here is the vanilla approach:

- ▶ Come up with a good kernel that encodes a “natural notion” of similarity (sometimes easy, sometimes not).
- ▶ Train an SVM by some standard package.
- ▶ Make sure you set all parameters by cross validation! The results are very sensitive to the choice of the regularization parameter C and the kernel parameters (such as σ for the Gaussian kernel).

Later in the lecture we will look at more preprocessing steps that you should use (\leadsto non-vanilla-version).

(*) Kernelizing the SVM primal

AS AN EXERCISE: IT IS POSSIBLE TO EXPRESS THE PRIMAL OPTIMIZATION PROBLEM IN TERMS OF KERNELS?

$$\text{minimize}_{w \in \mathcal{H}} \|w\|_{\mathcal{H}}^2 + \frac{C}{n} \sum_{i=1}^n \xi_i$$

$$\text{subject to } Y_i \left(\langle w, \Phi(X_i) \rangle_{\mathcal{H}} \right) \geq 1 - \xi_i \quad (i = 1, \dots, n)$$

(*) Kernelizing the SVM primal (2)

- ▶ A priori, we cannot write the primal function just in terms of scalar products because it contains a scalar product between the variable we are looking for (w) and the input points (X_i).
- ▶ But according to the representer theorem, the solution vector w can always be written as a linear combination of input feature vectors, that is $w = \sum_i \beta_i \Phi(X_i)$.
- ▶ Consequently,

$$\|w\|^2 = \langle w, w \rangle = \sum_{i,j} \beta_i \beta_j k(X_i, X_j)$$

and

$$\langle w, \Phi(X_j) \rangle = \sum_i \beta_i \langle \Phi(X_i), \Phi(X_j) \rangle = \sum_i \beta_i k(X_i, X_j)$$

(*) Kernelizing the SVM primal (3)

- With this knowledge we can also kernelize the primal problem:

$$\begin{aligned} & \underset{\beta \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^d}{\text{minimize}} \quad \frac{1}{2} \sum_{i,j} \beta_i \beta_j k(X_i, X_j) + \frac{C}{n} \sum_{i=1}^n \xi_i \\ & \text{subject to} \quad Y_i \left(\sum_{j=1}^n \beta_j k(X_j, X_i) + b \right) \geq 1 - \xi_i \quad (i = 1, \dots, n) \end{aligned}$$

History / discussion

Before SVMs, there were “old-day” neural networks. They had a couple of drawbacks:

- ▶ Lots of parameters to tune (design choices to make: how many neurons, how many layers, etc)
- ▶ Training a neural network was complicated, due to the non-convex nature of the optimization problem, one got stuck in local optima (work did not yet happen in the overparameterized regime, see later).
- ▶ To be able to successfully work with neural networks one needed a large amount of experience.

History / discussion (2)

Then came SVMs, they revolutionized the field. Why?

- ▶ Convex optimization problem, easy to implement
- ▶ Very few variables to tune (C , and maybe a kernel parameter such as σ in the Gaussian kernel), this can be done by cross validation
- ▶ The kernel framework boosts the potential of the SVM to the non-linear regime, but does not lead to excessive overfitting.
- ▶ Appealing from a conceptual side (large margin principle) and also from the mathematical point of view (support vector property, representer theorem, etc).
- ▶ Statistical learning theory shows many nice guarantees about the SVM (consistency, etc).

History / discussion (3)

By now, neural networks are back in the form of deep networks:

- ▶ Deep networks are very successful in cases where there exists lots of highly structured data (speech, text, images).
- ▶ Main difference to 30 years ago: much more data available, computational power increased a lot, much larger networks, good heuristics have been worked out. Overparameterized regime.

Comparing SVMs and Deep networks, both tend to be successful in very different types of applications:

- ▶ SVMs are good in the underparameterized regime. In particular, cases where only few data is available and little prior knowledge is available.
- ▶ Deep networks focus on the overparameterized regime (see later) and require huge amounts of data.

Summary: SVM with kernels

- ▶ Given data points in some space \mathcal{X} and a kernel function k on this space
- ▶ Want to solve classification.
- ▶ By the kernel trick, we embed our data points into some abstract feature space and use a linear classifier in this space.
- ▶ The inductive principle is that the margin in this feature space should be large.
- ▶ All this leads to a convex optimization problem that can be solved efficiently.
- ▶ There are lots of important properties (support vector property, representer theorem, etc).
- ▶ The kernel SVM is equivalent to regularized risk minimization with the Hinge loss and regularization by the squared norm in the feature space.

Summary: SVM with kernels (2)

The kernel SVM is one of the most important classification algorithms that is out there. If you just remember one thing from this whole course, try to remember SVMs 😊

Regression methods with kernels

Kernelized least squares

Least squares revisited

We had already seen in the beginning how to solve least squares regression when we have an explicit feature mapping Φ :

Given data in some space \mathcal{X} , a mapping $\Phi : \mathcal{X} \rightarrow \mathbb{R}^d$, we considered the least squares problem in the feature space

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \frac{1}{n} \sum_{i=1}^n (Y_i - \langle \Phi(X_i), w \rangle)^2$$

and found its analytic solution $w^* = (\Phi^t \Phi)^{-1} \Phi^t Y$.

We are now going to rewrite everything using kernels.

Kernelizing least squares (first method via representer theorem)

- ▶ The representer theorem tells us that the least squares problem always has a solution of the form

$$w^* = \sum_{j=1}^n \alpha_j \Phi(X_j).$$

- ▶ Plugging this in the objective gives

$$\begin{aligned} & \underset{w \in \mathbb{R}^d}{\text{minimize}} \quad \frac{1}{n} \sum_{i=1}^n (Y_i - \langle \Phi(X_i), w \rangle)^2 \\ \iff & \underset{\alpha \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{n} \sum_{i=1}^n (Y_i - \sum_{j=1}^n \alpha_j \underbrace{\langle \Phi(X_i), \Phi(X_j) \rangle}_{k_{ij}})^2 \end{aligned}$$

Kernelizing least squares (first method via representer theorem) (2)

- In matrix notation:

$$\underset{\alpha \in \mathbb{R}^n}{\text{minimize}} \frac{1}{n} \|Y - K\alpha\|^2$$

- By taking the derivative with respect to α and exploiting that K is pd it is easy to see that the **solution is given as** $\alpha^* = K^{-1}Y$ (EXERCISE!).
- To evaluate the solution on a new data point x , we need to compute

$$f(x) = \langle \Phi(x), w^* \rangle = \sum_j \alpha_j^* \langle \Phi(x), \Phi(X_j) \rangle = \sum_j \alpha_j^* k(x, X_j)$$

- **So we can express the optimization problem, its solution and the evaluation function purely in terms of kernel functions. We have kernelized least squares.**

(*) Kernelizing least squares (second method via SVD)

Recap: the kernel matrix is $\Phi\Phi^t$

- ▶ Let $X_1, \dots, X_n \in \mathcal{X}$ be data points, $\Phi : \mathcal{X} \rightarrow \mathbb{R}^d$ a feature map.
- ▶ Denote by Φ the $n \times d$ -matrix that contains the data points $\Phi(X_i)$ as rows.
- ▶ Then the matrix $\Phi \cdot \Phi^t \in \mathbb{R}^{n \times n}$ coincides with the corresponding kernel matrix K with entries $k_{ij} = \langle \Phi(X_i), \Phi(X_j) \rangle = \Phi(X_i)\Phi(X_j)^t$.

(*) Kernelizing least squares (second method via SVD) (2)

Proposition 14 (Matrix Identities)

For any $n \times d$ -matrix Φ we have

$$(\Phi^t \Phi)^{-1} \Phi^t = \Phi^t (\Phi \Phi^t)^{-1}$$

Proof of the proposition.

- ▶ Let $\Phi = U \Sigma V^t$ the singular value decomposition of Φ .
- ▶ It is straightforward to prove that $(\Phi^t \Phi)^{-1} \Phi^t = V \Sigma^+ U^t$ (have seen this already when we derived least squares).
- ▶ It is even more straightforward to see that $\Phi^t (\Phi \Phi^t)^{-1} = V \Sigma^+ U^t$.



(*) Kernelizing least squares (second method via SVD) (3)

Using this proposition and the fact that the kernel matrix K is given as $\Phi\Phi^t$ we can rewrite the least squares solution as

$$w^* = (\Phi^t\Phi)^{-1}\Phi^tY = \Phi^t(\Phi\Phi^t)^{-1}Y = \Phi^tK^{-1}Y$$

Denote $\alpha := K^{-1}Y$. With this notation, the evaluation function is

$$\begin{aligned} f(x) &= \langle w^*, \Phi(x) \rangle = (w^*)^t \Phi(x) \\ &= (\Phi^t K^{-1} Y)^t \Phi(x) = Y^t K^{-1} \Phi^t \Phi(x) \\ &= \alpha^t \Phi^t \Phi(x) \\ &= \sum_{j=1}^n \alpha_j \Phi(X_j)^t \Phi(x) \end{aligned}$$

(*) Kernelizing least squares (second method via SVD) (4)

$$= \sum_{j=1}^n \alpha_j k(X_j, x)$$

So we can express the optimization problem, its solution and the evaluation function purely in terms of kernel functions. We have kernelized least squares.

Kernel ridge regression

Ridge regression

Recall ridge regression in feature space:

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \frac{1}{n} \sum_{i=1}^n (Y_i - \langle w, \Phi(X_i) \rangle)^2 + \lambda \|w\|^2$$

Again use the representer theorem to express w as a linear combination of input points:

$$w = \sum_{j=1}^n \alpha_j \Phi(X_j)$$

Ridge regression (2)

This leads to the following kernelized ridge regression problem:

$$\underset{\alpha \in \mathbb{R}^n}{\text{minimize}} \frac{1}{n} \|Y - K\alpha\|^2 + \lambda \alpha^t K \alpha$$

The solution is given by

$$\alpha = (n\lambda I + K)^{-1} Y$$

As before, we can compute the prediction for a new test point just using kernels.

A subtle difference: Ridge regression vs. kernel ridge regression

- ▶ Given points in space \mathcal{X} . Want to compare:
 - ▶ Ridge regression using basis functions $\Phi_i(x) = k(X_i, x)$
 - ▶ Kernel ridge regression using kernel k and feature map Φ
- ▶ In both cases, we work in the same function space, namely the one spanned by the functions Φ_i .
- ▶ So no matter which function we use, the least squares error is the same in both approaches.
- ▶ **However, the regularizers are different:**
 - ▶ In the standard case we regularize by $\|\alpha\|^2$.
 - ▶ In the kernel case we regularize by $\alpha^t K \alpha$.
- ▶ This is as for linear and kernel SVMs ...

Kernel version of LASSO

Note: the trick that we used to derive the kernel version of ridge regression does NOT work for Lasso (WHY????)

How to center and normalize in the feature space

Literature:

- ▶ Shawe-Taylor / Cristianini Section 5.1

What we want to do

- ▶ Have seen: many algorithms require that the data points are centered (have mean = 0) and are normalized.
- ▶ However, now we want to work in feature space, but without explicitly working with the coordinates in feature space.
- ▶ So how can we do this ???

Centering in the feature space

To center points in the feature space, we would need to perform the following calculations:

- ▶ Compute center: $\bar{\Phi} := 1/n \sum_i \Phi(x_i)$
- ▶ Replace $\Phi(x_i)$ by $\Phi(x_i) - \bar{\Phi}$

Not obvious that we can express this in terms of scalar products...

Centering in the feature space (2)

To proceed, assume that we can compute $\bar{\Phi}$ and let's compute the kernel values between the centered points:

$$\begin{aligned}\tilde{K}_{ij} &:= \langle \Phi(x_i) - \bar{\Phi}, \Phi(x_j) - \bar{\Phi} \rangle \\ &= \langle \Phi(x_i) - 1/n \sum_{s=1}^n \Phi(x_s), \Phi(x_j) - 1/n \sum_{t=1}^n \Phi(x_t) \rangle \\ &= k(x_i, x_j) - \frac{1}{n} \sum_{s=1}^n k(x_i, x_s) - \frac{1}{n} \sum_{t=1}^n k(x_j, x_t) \\ &\quad + \frac{1}{n^2} \sum_{s,t=1}^n k(x_s, x_t)\end{aligned}$$

Centering in the feature space (3)

In matrix notation, this means that we can compute the centered kernel matrix as follows:

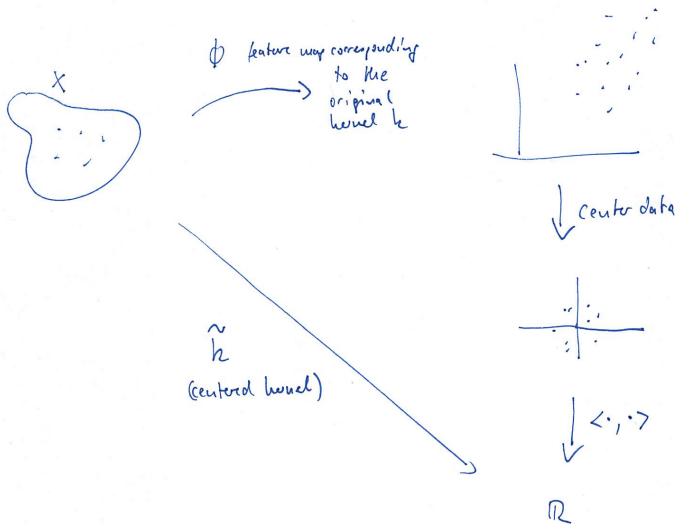
$$\tilde{K} = (K - \mathbb{1}_n K - K \mathbb{1}_n + \mathbb{1}_n' K \mathbb{1}_n)$$

where $\mathbb{1}_n$ is the $n \times n$ matrix containing $1/n$ as each entry.

Good news:

- ▶ We do not have to do the centering operation explicitly.
- ▶ We can implicitly center the data by replacing the “old” kernel matrix K by the new matrix \tilde{K} .

Centering in the feature space (4)



Normalizing in feature space

Assume our n data points are in \mathbb{R}^d and we stack them in a data matrix X as usual:

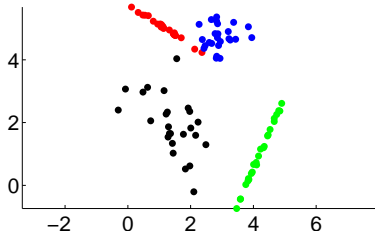
- ▶ Each row of X corresponds to one data point.
- ▶ The data matrix has dimensions $n \times d$

Two different ways to normalize data:

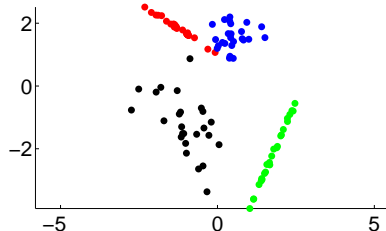
- ▶ **Normalize the data points**, that is rescale each data point such that it has norm 1. This is equivalent to normalizing the **rows** of the centered matrix to have unit norm.
- ▶ **Normalize the individual features**, that is rescale all **columns** of the centered matrix to have norm 1. This is just a rescaling of the coordinate axes such that the variance in each coordinate direction is 1.

Normalizing in feature space (2)

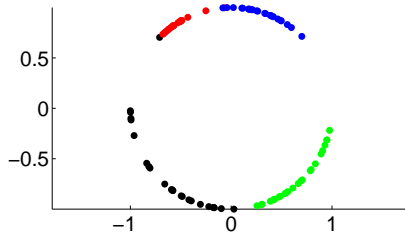
orig data



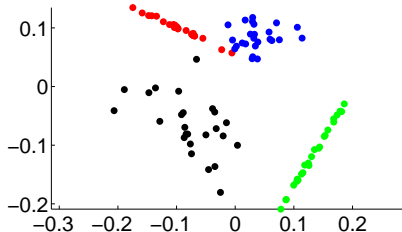
points centered



points centered, points normalized



points centered, features normalized



Normalizing in feature space (3)

Note:

- ▶ Normalize the points
 - ▶ We will see below that there is a way to normalize points in the feature space.
 - ▶ Sometimes it helps, sometimes it hurts.
 - ▶ If in doubt, use cross-validation to see whether your results improve if you normalize or not.
- ▶ Normalize features:
 - ▶ Typically this never hurts, and often helps.
 - ▶ For kernel methods it is impossible to normalize the features (we don't know the embedding Φ explicitly, in particular we don't know what the features are).

Normalizing in feature space (4)

To normalize the points such that they have unit norm in the feature space:

- ▶ Assume the data points are already centered in feature space.
- ▶ Then define the normalized data point
$$\hat{\Phi}(X) := \Phi(X) / \|\Phi(X)\|.$$
- ▶ Observe:

$$\begin{aligned}\langle \hat{\Phi}(X), \hat{\Phi}(Y) \rangle &= \left\langle \frac{\Phi(X)}{\|\Phi(X)\|}, \frac{\Phi(Y)}{\|\Phi(Y)\|} \right\rangle \\ &= \frac{\langle \Phi(X), \Phi(Y) \rangle}{\|\Phi(X)\| \|\Phi(Y)\|} \\ &= \frac{k(x, y)}{\sqrt{k(x, x)k(y, y)}}\end{aligned}$$

Normalizing in feature space (5)

So instead of first normalizing the points and then computing their kernels we can directly compute the kernels for the normalized data points.

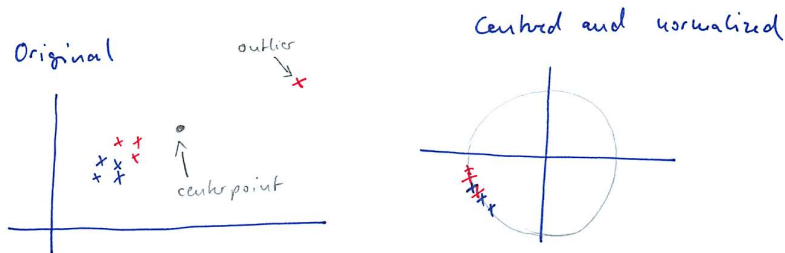
So to normalize the points in feature space, we simply replace the kernel function k by the normalized kernel function

$$\hat{k}(x, y) = \frac{k(x, y)}{\sqrt{k(x, x)k(y, y)}}.$$

(VERIFY THAT THIS IS INDEED A KERNEL)

When it can go wrong

Standardizing the data is a preprocessing step. As any such step, it often helps, but sometimes can go wrong:



Randomized methods: bagging, boosting and friends

Random Forests

Literature:

- ▶ Original paper: Breiman: Random forests. Machine Learning 2001.
- ▶ Nice recent overview paper: G. Biau and E. Scornet. A random forest guided tour. Test, 25(2):197-227, 2016.
- ▶ Text book: Chapters 8, 9 and 15 in Hastie/Tibshirani/Friedman (Elements of statistical learning)
- ▶ Textbook: Chapter 18 of Shalev-Shwartz/Ben-David (Understanding machine learning)
- ▶ Some of our own work:
 - ▶ Cheng Tang, Damien Garreau, Ulrike von Luxburg: When do random forests fail? NeurIPS 2018
 - ▶ Siavash Haghiri, Damien Garreau, Ulrike von Luxburg: Comparison-Based random forests. ICML 2018

Background: Ensemble methods

- ▶ Each individual classifier that we train might have some particular errors.
- ▶ If we train many different classifiers (an “ensemble” of classifiers), each of them might be “better” in some aspects.
- ▶ We now train many classifiers in parallel and in the end build a committee in which they jointly decide. In the simplest case, by majority vote, in other cases by more complicated weighted approaches.

Background: Bootstrap and bagging

Consider the method of cross validation:

- ▶ Want to estimate the test error of a classifier
- ▶ Take **random subsets of training set** for training, and another random set for testing.
- ▶ To get a more reliable result, we **repeat this procedure a number of times and average the result.**

In statistics, there exists a huge family of methods that follow a similar principle: bootstrap and bagging methods.

Background: Bootstrap and bagging (2)

The Bootstrap method:

- ▶ Assume you have a sample X_1, \dots, X_n of points and, say, an estimate $\hat{\Theta}$ of a true parameter Θ of this population. You would like to know the distribution of the estimate $\hat{\Theta}$ (for example, because you want to construct confidence sets).
- ▶ You now draw a subsample of m points of the original sample (with or without replacement), and on this subsample you compute an estimate of the parameter you are interested in.
- ▶ You repeat this procedure B times, resulting in B bootstrap estimates $\hat{\Theta}_1, \dots, \hat{\Theta}_B$.
- ▶ This set now gives an “indication” about how your estimate is distributed, and you can compute its mean, its variance, confidence sets, etc.

Background: Bootstrap and bagging (3)

- ▶ There exists a lot of theory, proving under which conditions this “indication” is statistically sound (consistent).

Be aware, the estimates $\hat{\Theta}_b$ are not independent, hence nothing here is trivial

There are many textbooks on the bootstrap, for example:

- ▶ An Introduction to the Bootstrap, by Efron and Tibshirani.
- ▶ The Jackknife and Bootstrap, by Shao and Tu

Background: Bootstrap and bagging (4)

Bagging: (short for **B**ootstrap **a**ggregation)

- ▶ As in bootstrap, you generate B bootstrap samples of your original sample, and on each of them compute the estimate you are interested in: $\hat{\Theta}_1, \dots, \hat{\Theta}_B$
- ▶ As your final estimate, you then take the average:
$$\hat{\Theta}_{bag} = \text{mean}(\hat{\Theta}_1, \dots, \hat{\Theta}_B).$$
- ▶ The advantage of this procedure is that the estimate Θ_{bag} can have a much smaller variance than each of the individual estimates $\hat{\Theta}_b$:
 - ▶ If the estimates $\hat{\Theta}_b$ were i.i.d. with variance σ^2 , then the variance of $\hat{\Theta}_{bag}$ would be σ^2/B .
 - ▶ If the estimates are identically distributed but have a (hopefully small) positive pairwise correlation ρ , then the variance of $\hat{\Theta}_{bag}$ is $\rho\sigma^2 + (1 - \rho)\frac{\sigma^2}{B}$. If ρ is small and B is large, this is good.

Background: Bootstrap and bagging (5)

We would now like to apply this principle to regression or classification:

- ▶ Given a sample of training points
- ▶ Repeatedly take a subsample, train some baseline algorithm on the subsample obtaining B classifiers/regressors f_1, \dots, f_B .
- ▶ For a test point x , compute the results of all baseline classifiers: $y_b = f_b(x)$, and then take the average:
 $y_{bag} = \text{mean}(y_1, \dots, y_B)$.

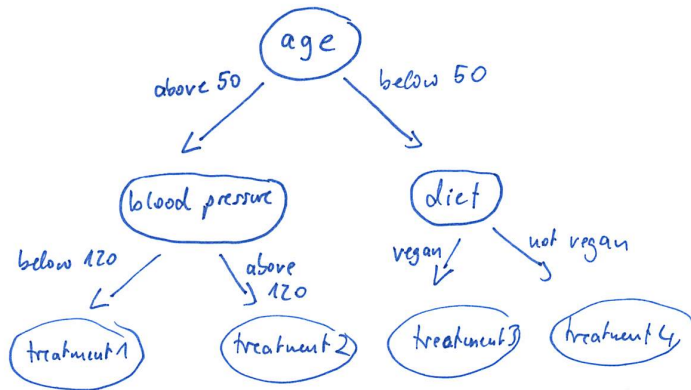
Background: Bootstrap and bagging (6)

First considerations for choosing the baseline classifier:

- ▶ As this mechanism is expensive from a computational point of view, it makes sense to use a reasonably simple baseline algorithm for training.
- ▶ Bagging reduces the variance most if there is only little correlation between the individual classifiers. This is what we need to achieve.
- ▶ If the classifiers have a strong bias, bagging cannot do anything about that.

A standard choice is to use decision trees, and then aggregate them to a “forest”.

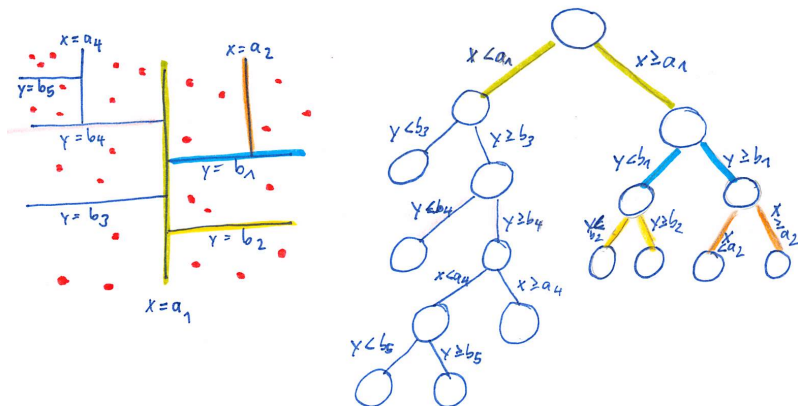
Decision trees: intuition



There exists a large variety of algorithms to build decision trees. Below we just consider the simplest of all setups, spatial decision trees, because this is what is typically used in random forests.

Spatial decision trees on \mathbb{R}^d : intuition

Assume our data lives in \mathbb{R}^d (typically, with a large dimension d). A spatial partition tree looks as follows:



Left side: the partition of the space; right side: the tree.

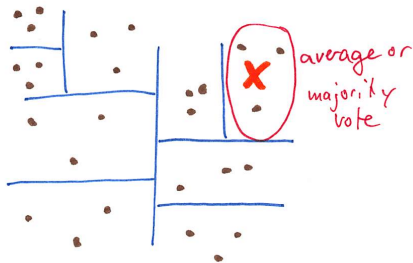
Spatial decision trees on \mathbb{R}^d : intuition (2)

To construct the tree, we proceed recursively. In each step, we select one dimension along which we are going to split the current cell. We keep on splitting cells until they contain only few data points.

Spatial decision trees on \mathbb{R}^d : intuition (3)

To predict the regression/classification output for a test point, we determine in which cell the test point is, consider the labels of all training points in this cell and then predict the average value (in case of regression) or the majority vote (in case of classification) for the test point.

Illustration (the red cross is the test point):



Spatial decision trees on \mathbb{R}^d : intuition (4)

Illustration in case of classification:

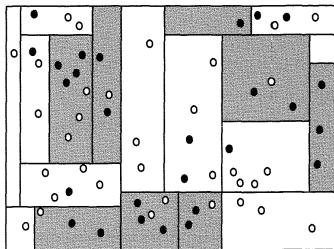


FIGURE 20.8. A natural classifier based on an ordinary binary tree. The decision is 1 in regions where points with label 1 form a majority. These areas are shaded.

Figure taken from "A Probabilistic Theory of Pattern Recognition"

Spatial decision trees: Quality of a split

To build the tree, we need a criterion to evaluate the quality of a proposed data split. There are many such criteria. As an example, consider the resulting training error on both sides:

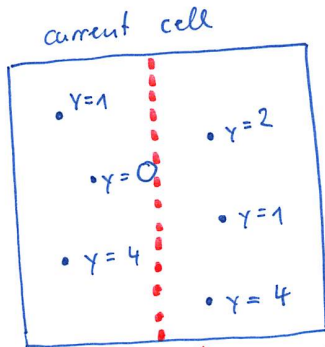
Consider a split of the data into two sets, called A and A^c .

- ▶ For all training points in A , we would predict the outcome $\hat{Y}_A = \text{mean}(\{Y_i \mid X_i \in A\})$.
- ▶ Similarly, for points in A^c we would predict the mean \hat{Y}_{A^c} .
- ▶ We can now compute the sum of the least squares error we would achieve on both sides:

$$\text{error}_{\text{split}} = \sum_{i \in A} (Y_A - Y_i)^2 + \sum_{i \in A^c} (Y_{A^c} - Y_i)^2$$

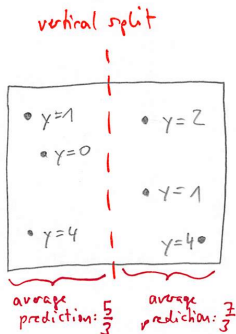
Spatial decision trees: Quality of a split (2)

Illustration: cell with 6 data points that have y -values as indicated. After the proposed split, the tree would predict the value $5/3$ on the left side and $7/3$ on the right side (the average y -values on the respective sides). The average L_2 training error with these predictions would be 2.2.



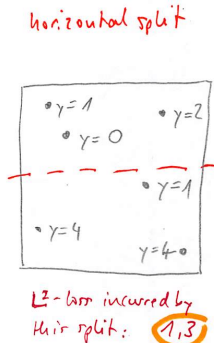
Spatial decision trees: Quality of a split (3)

If we have to decide which of two splits to implement next, we do so according to the the splitting error:



L^2 loss induced by this split:

$$\frac{1}{6} \left(\left(1 - \frac{5}{3}\right)^2 + \left(0 - \frac{5}{3}\right)^2 + 4\left(4 - \frac{5}{3}\right)^2 + \left(2 - \frac{7}{3}\right)^2 + \left(1 - \frac{7}{3}\right)^2 + 4\left(4 - \frac{7}{3}\right)^2 \right) = 2,2$$



we chose the horizontal split (has smaller error)



Spatial decision trees: Selecting the splitting dimension

Typically, spatial decision trees use axis-parallel splits. Given the current cell, they select one dimension along which to split next according to splitting error:

- ▶ For all dimensions $k = 1, \dots, d$:
 - ▶ Find the best splitting point s_k along this dimension k by optimizing the splitting error (typically, by a greedy procedure) and store the resulting splitting point s_k and splitting error $error_{split,k}$
- ▶ Select dimension with the smallest splitting error and use the corresponding split.

Spatial decision trees: Depth of the tree

Depth of the tree and size of the leafs:

Typically, we keep on splitting cells until they contain less than a pre-defined number n_{leaf} of points. Different rules of thumb exist:

- ▶ To achieve statistical consistency of the decision tree, the number of points per leaf needs to increase (slowly) with the number n of training points. For example, $n_{leaf} = \log n$ leads to consistency (see also later in the learning theory section). The resulting trees are sometimes called “**shallow trees**”.
- ▶ In practice, in random forests one often use “**deep trees**” that always contain only a constant number of points in the leaf (independent of n ; in the extreme case, $n_{leaf} = 1$). If we just used a single decision tree, this procedure would be a disaster (overfitting!), but in a random forests it can lead to a consistent forest in the end. See later.

Spatial decision trees: final remarks

- ▶ One advantage of spatial decision trees is their interpretability: because splits are axis-parallel, there is a simple interpretation (“blood pressure is below 120”) for each split, and ultimately also for a decision at a particular leaf.
- ▶ There are also many ways by which one can compute an “importance” score for each of the features for a random forest. See textbooks.
- ▶ There exist a more general class of decision trees. The main difference is that the choice of the dimensions along which we split and the choice of the splits themselves can be made by many different criteria. For example, the C4.5 algorithm uses criteria based on information theory. See the literature if you are interested.

Random forest

- ▶ A random forest uses bagging to combine many spatial decision trees to one big estimate.
- ▶ Each individual tree is constructed randomly: on a random sample of the input points, and by selecting the next splitting dimension from a random subset of all dimensions.

Random forest (2)

Input: data points in \mathbb{R}^p (p = dimension of the space)

Parameters: \mathcal{B} , N , n_{\min} , m

Algorithm 15.1 Random Forest for Regression or Classification.

1. For $b = 1$ to B :
 - (a) Draw a bootstrap sample \mathbf{Z}^* of size N from the training data.
 - (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{\min} is reached.
 - i. Select m variables at random from the p variables.
 - ii. Pick the best variable/split-point among the m .
 - iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x :

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the b th random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B$.

Random forest (3)

Figure from "Elements of statistical learning"

Random forest (4)

Parameters:

- ▶ The size of the subsample: reasonably large; can be with or without replacement (in the latter case, one often chooses the subsample size equal to the size of the original sample)
- ▶ The number m of dimension of which we pick the best one: typically, people choose something around $d/3$ where d is the original dim of the data
- ▶ The number B of trees, should be large
- ▶ The number n_{min} of points in the leafs: depending on whether you consider deep or shallow trees. In the extreme case of deep tree, $n_{min} = 1$. Then you definitely need many trees (B large). In the case of shallow trees, $n \approx \log n$.

Random forest (5)

In principle, many people report that the algorithm is not extremely sensitive to many of the parameters. However, if you are in a completely bad regime, the tree can under- or overfit. See discussion in Biau (Random forest guided tour) or our paper (When random forests fail).

Consistency of random forests

On a high level, here are the main results:

- ▶ A single spatial decision tree is consistent if the diameter of all cells converges to 0 and at the same time, the number of points in each of the cells tends to infinity, as the number n of data points goes to infinity (see chapter 20 of “Probabilistic theory of pattern recognition”)
- ▶ If all individual trees are consistent, so is the random forest (see G. Biau. Analysis of a random forests model. JMLR 2012)
- ▶ Curiously, a random forest can be consistent even if all its individual trees are not consistent. This is particularly the case for deep trees (see E. Scornet. On the asymptotics of random forests. Journal of Multivariate Analysis 2016).

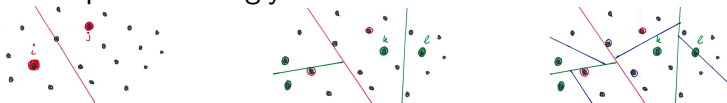
Consistency of random forests (2)

For more discussion about the consistency of tree classifiers and random forests see our paper “When do random forests fail” (Tang, Garreau, Luxburg 2018) and references therein, and Chapter 15 of Elements of Statistical learning.

Outlook: Comparison-based random forests

Random forests as described above require a Euclidean representation of data points. Alternative: comparison-tree:

- ▶ Given a current cell, we randomly select two of its data points x_1, x_2 .
- ▶ For every point x in the cell, we evaluate whether it is closer to x_1 or x_2 .
- ▶ Then we split accordingly:



- ▶ We could prove that this tree can also lead to consistent classification/regression (“Comparison-based random forests”, Haghir, Garreau, Luxburg 2018)
- ▶ Empirically, it works really well.

Discussion and History

- ▶ Random forests have been invented in Breiman: Random forests. Machine Learning 2001.
- ▶ Random forests are a very simple, yet very successful class of algorithms.
- ▶ They are used very widely in practice, not least due to their interpretability.
- ▶ Always consider Random Forests as a baseline when you work on a new machine learning problem.

Boosting

Literature:

- ▶ Shalev-Shwartz/Ben-David: Understanding Machine Learning, Section 10
- ▶ Hastie/Tibshirani/Friedman: Elements of Statistical Learning, Section 16
- ▶ A whole book: Boosting - Foundations and algorithms by Schapire and Freund
- ▶ Original paper: Robert Schapire, Yoav Freund, 1995.

Strong and weak learners

Intuitively, a “**strong learner**” is a classification algorithm that can approximate the true solution up to a small error ε . The goal of machine learning is to construct strong learners.

However, they are often hard to construct and computationally expensive.

A “**weak learner**” is an algorithm that is just slightly better than random guessing: for a classification problem with balanced classes, its 0-1-loss is just a tiny bit better than random guessing: $0.5 + \varepsilon$.

The idea of boosting is to combine many weak learners to obtain a strong learner.

Boosting, the outline

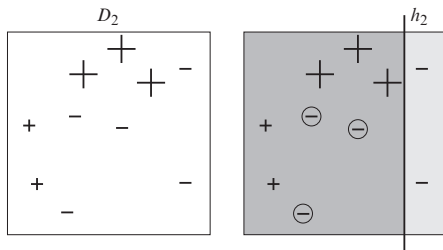
- ▶ Given a training set of n points $(x_i, y_i)_{i=1, \dots, n}$, the boosting algorithm proceeds in T rounds.
- ▶ Training points have weights that change from round to round. The weights always add up to 1.
- ▶ In each round, we train the weak classifier on the training points with the current weights. We then update the weights:
 - ▶ For point x_i that was mis-classified, we increase its weight w_i .
 - ▶ For point x_i that got correctly classified, we decrease its weight w_i .
- ▶ At the very end, the final classifier is a weighted sum of the weak classifiers of each round.

We now adapt the weights:

Boosting: toy example (2)

Left: data set with new weights (size of weight indicated by size of the plus/minus sign).

Right: a new weak classifier, trained on the weighted examples. It gets the points with the high weights right, but makes other mistakes.

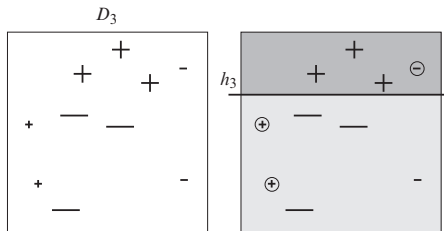


Again we reweight:

Boosting: toy example (3)

Left: data set with new weights.

Right: a new weak classifier, trained on the weighted examples.



Now we have three weak classifiers. We combine them to a final (strong) classifier:

Boosting: toy example (4)

We combine them by a linear combination (weighted majority vote):

$$H = \text{sign} \left(0.42 \begin{array}{|c|} \hline \text{[Diagram 1]} \\ \hline \end{array} + 0.65 \begin{array}{|c|} \hline \text{[Diagram 2]} \\ \hline \end{array} + 0.92 \begin{array}{|c|} \hline \text{[Diagram 3]} \\ \hline \end{array} \right)$$

$$= \begin{array}{|c|c|c|c|} \hline & & & \\ \hline & + & + & - \\ \hline + & - & + & \\ \hline + & - & - & - \\ \hline & - & & \\ \hline \end{array}$$

Boosting: toy example (5)

Some remarks:

- ▶ As opposed to random forests, which generate randomness through subsampling points and dimensions, the training set in boosting is always the same. We just change its weights.
- ▶ By re-weighting, the algorithm can focus on those examples that it finds difficult, or on aspects that have been overlooked so far.
- ▶ By the definition of a weak classifier, once a data point has accumulated weight (probability mass) larger than 0.5, the weak algorithm will get it right. But it is not obvious that this helps for the final classifier, because there are many points that we need to get right... So the question is how to combine all the evidence of the weak classifiers in such a way that we obtain a strong classifier in the end.

AdaBoost algorithm

sample weights denoted $D_i^{(t)}$

update factors and factor in final lin. comb: w_t

AdaBoost

input:

training set $S = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$

weak learner WL

number of rounds T

initialize $D^{(1)} = (\frac{1}{m}, \dots, \frac{1}{m})$.

for $t = 1, \dots, T$:

invoke weak learner $h_t = \text{WL}(D^{(t)}, S)$

compute $\epsilon_t = \sum_{i=1}^m D_i^{(t)} \mathbb{1}_{[y_i \neq h_t(\mathbf{x}_i)]}$ \rightarrow weighted 0-1-loss at step t

let $w_t = \frac{1}{2} \log \left(\frac{1}{\epsilon_t} - 1 \right)$ update factor

update $D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(\mathbf{x}_i))}{\sum_{j=1}^m D_j^{(t)} \exp(-w_t y_j h_t(\mathbf{x}_j))}$ for all $i = 1, \dots, m$

output the hypothesis $h_s(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T w_t h_t(\mathbf{x}) \right)$.

update weights of all sample points

weak learner gets weighted sample

\rightarrow final hypothesis after the T rounds:

a weighted combination of the weak learners of all previous rounds

Figure here and next page taken from "Understanding Machine Learning"

AdaBoost algorithm (2)

Intuition: why is it plausible that it works?

- ▶ Assume the final classifier h_s makes an error on some training point x .
- ▶ Because h_s is a (weighted) majority vote over all the weak classifiers, h_s can only get x wrong if most of the weak classifiers have classified x wrongly themselves.
- ▶ This means that the weight of x has been increased very often, so it must be still large after the final round.
- ▶ But there can only be few points with large weights, because all weights add up to 1.
- ▶ So there can only be few points that get classified wrong by the final classifier.

Theoretical results on the training error

There are many theoretical results for boosting. For example, the following theorem shows how the training error improves by boosting:

THEOREM 10.2 *Let S be a training set and assume that at each iteration of AdaBoost, the weak learner returns a hypothesis for which $\epsilon_t \leq 1/2 - \gamma$. Then, the training error of the output hypothesis of AdaBoost is at most*

$$L_S(h_s) = \frac{1}{m} \sum_{i=1}^m \mathbb{1}_{[h_s(\mathbf{x}_i) \neq y_i]} \leq \exp(-2\gamma^2 T) .$$

The proof is surprisingly simple, let's look at it:

Theoretical results on the training error (2)

Step 1: We look at the global function f_t that has been learned after step t and define a convenient random variable Z_t :

$$f_t := \sum_{p \leq t} \overset{\text{weight}}{w_p} \overset{\text{weak classifier}}{h_p} \quad (\text{thus, output is } f_t)$$

$$Z_t := \frac{1}{m} \sum_{i=1}^m \exp(-y_i f_t(x_i))$$

Note: for any classifier h , we have $\underbrace{\mathbb{1}(h(x) \neq y)}_{0-1\text{-loss}} \leq \underbrace{\exp(-y h(x))}_{\substack{0 \quad \text{if correctly classified} \quad 1/e \approx 0.3 \\ 1 \quad \text{if misclassified} \quad e \approx 2.7}}$

Thus: $\text{Training error} (f_t) \leq Z_t$
 0-1-loss on training set

So if we can bound Z_T by the term in the theorem, we are done.

Theoretical results on the training error (3)

Step 2: Telescope sum to bound Z_T :

Now we want to bound z_t .

Trick:
$$z_T = \frac{z_T}{z_0} = \frac{z_T}{z_{T-1}} \cdot \frac{z_{T-1}}{z_{T-2}} \cdot \dots \cdot \frac{z_1}{z_0} =: 1 \text{ initialization}$$

Want to show that each of these factors satisfies

$$\frac{z_{t+1}}{z_t} \leq \exp(-2\gamma^2)$$

Theoretical results on the training error (4)

Step 3: Show the bound on the factors (exploits the exact form of weights and update factors):

Theoretical results on the training error (5)

By a simple induction one can prove:

$$\mathbb{D}_i^{(t+1)} = \frac{\exp(-\gamma_i f_t(x_i))}{\sum_{j=1}^m \exp(-\gamma_j f_t(x_j))} \quad \text{formula for new weight}$$

Exploiting this, one can show in a couple of elementary computation steps that $\frac{Z_{t+1}}{Z_t} = 2 \sqrt{\epsilon_{t+1} (1 - \epsilon_{t+1})}$.

Exploiting that $\epsilon_{t+1} \leq \frac{1}{2} - \gamma$ (assumption!) we obtain

$$\dots \leq \sqrt{1 - 4\gamma^2} \leq \exp(-2\gamma^2)$$

\uparrow
 $1 - a \leq \exp(-a)$

(For the omitted proof details, see the Shalev-Schwartz/Ben-David book and the Schapire/Freund book)

Theoretical results on the training error (6)

Step 4: Combining everything:

Finally combining everything we get:

$$\text{Training error}(f_T) \leq z_T \leq \sum_{t=0}^{T-1} \left(\frac{z_{t+1}}{z_t} \right)$$

$$\leq \left(\exp(-2r^2) \right)^T = \exp(-2r^2 T)$$



Theoretical results on the test error

The key result that we are interested in: does the test error also improve during the boosting procedure?

To prove such results, one can use some of the standard techniques from statistical learning theory (see later in the lecture). For example, one can devise a large margin theory for boosting.

Check out the book by Schapire/Freund if you are interested, it is really well-written!

Bagging vs Boosting

Bagging, e.g. random forests:

- ▶ Generate many individual predictors in parallel, typically by subsampling
- ▶ Try to make them as independent as possible
- ▶ Average to obtain the final results

Boosting:

- ▶ Works sequentially, not in parallel.
- ▶ Individual classifiers are not independent: the next classifier explicitly tries to improve on the misclassified points of the previous one.
- ▶ Specific weighted average to obtain final result.

Final remarks on boosting

- ▶ As weak classifiers, one often uses decision stumps: decision trees of size 1. They induce axis-parallel splits as indicated in the toy example.
- ▶ There are many variants of boosting out there, we just scratched the surface.
- ▶ Boosting is well-understood from many different points of view, and it works well in practice.

(*) Gradient Boosting

Literature:

- ▶ Original paper: Friedman: Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics, 2001.
- ▶ Current implementation: Chen, Guestrin: XGBoost - A scalable Tree Boosting Algorithm. KDD 2016.

Gradient Tree Boosting

- ▶ ... is a combination ideas of random forests with variants of boosting.
- ▶ is extremely successful in practice, e.g. using the XGBoost implementation.

Skipped due to lack of time, but try it out. It has been substantial ingredient in a large number of kaggle competition winners!

Unsupervised learning

Dimensionality reduction and embedding

Classical PCA

Classical PCA is covered in many statistics books:

- ▶ A complete book on PCA is Jolliffe: Principal Component Analysis. Springer, 2002.
- ▶ Chapter 8 in Mardia, Kent, Bibby: Multivariate Analysis. Academic Press, 1979. A classic.

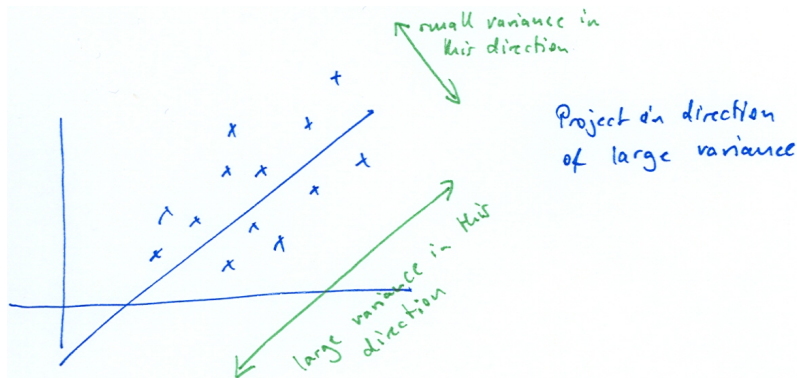
Principal component analysis (PCA)

... is a “traditional” method for unsupervised dimensionality reduction. Is based on linear principles.

Goal:

- ▶ Given data points $x_1, \dots, x_n \in \mathbb{R}^d$
- ▶ want to reduce the dimensionality of the data by throwing away “dimensions which are not important”.
- ▶ Result is set of new data points $y_1, \dots, y_n \in \mathbb{R}^\ell$ with $\ell < d$.

Principal component analysis (PCA) (2)



Principal component analysis (PCA) (3)

Two approaches in this lecture:

- ▶ Traditional approach: Maximize the variance of the reduced data \leadsto Covariance matrix approach
- ▶ Traditional approach: Minimize the quadratic error \leadsto SVD approach

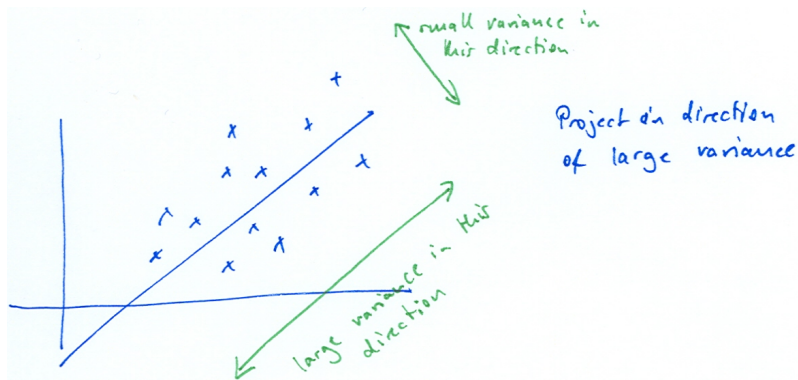
Recap: Projections

... see slides in the appendix (slides 1377 ff.)

Recap: Variance and Covariance

... see slides in the appendix (slides 1323 ff.)

PCA by max variance approach: Idea



Want to find a linear projection on a low-dim space such that the overall variance of the resulting points is as large as possible.

PCA by max variance approach: Idea (2)

Given: data points $x_1, \dots, x_n \in \mathbb{R}^d$, parameter $\ell < d$ (the dimension of the space we want to project to).

Goal: find a projection π_S on an affine subspace S such that the variance of the projected points is maximized: $\max_S \text{Var}_\ell(\pi_S(X))$.

For simplicity, let us assume that the data points are centered:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i = 0$$

(If this is not the case, we can center the data points by setting $\tilde{x}_i = x_i - \bar{x}$.)

PCA by max variance approach: Case $\ell = 1$

One-dimensional case:

We first of all assume that $\ell = 1$, that is we want to project the data points on a 1-dim space.

Have to solve the following optimization problem:

$$\begin{aligned} & \max_{a \in \mathbb{R}^d, \|a\|=1} \text{Var}(\pi_a(X)) \\ \iff & \max_{a \in \mathbb{R}^d} \sum_{i=1}^n (\pi_a(x_i))^2 \quad \text{subject to } a^t a = 1 \\ \iff & \max_{a \in \mathbb{R}^d} \sum_{i=1}^n (a^t x_i)^2 \quad \text{subject to } a^t a = 1 \\ \iff & \max_{a \in \mathbb{R}^d} \|Xa\|^2 \quad \text{subject to } a^t a = 1 \end{aligned}$$

PCA by max variance approach: Case $\ell = 1$ (2)

To solve this:

- Write the Lagrangian:

$$L(a, \lambda) = \|Xa\|^2 - \lambda(a^t a - 1) = a^t X^t X a - \lambda(a^t a - 1)$$

- Compute the partial derivatives wrt a :

$$\partial L / \partial a = 2X^t X a - 2\lambda a \stackrel{!}{=} 0$$

Thus necessary condition: a is an eigenvector of $X^t X$.

- Substitute $X^t X a = \lambda a$ in the original objective function:

$$a^t X^t X a = \lambda a^t a = \lambda$$

- This is maximal for a being the *largest* eigenvector of $X^t X$.

Solution: If the data points are centered, then projecting on the largest eigenvector of $C = X^t X$ solves the problem for $\ell = 1$.

PCA by max variance approach: Case $\ell > 1$

Case $\ell > 1$:

By similar arguments we can prove that we need to project the data on the space spanned by the ℓ largest eigenvectors of $X^t X$.

(and by “largest eigenvector” I mean the eigenvector corresponding to the largest eigenvalue).

PCA: algorithm using covariance matrix C

Input: Data points $x_1, \dots, x_n \in \mathbb{R}^d$, parameter $\ell \leq d$.

- ▶ Center the data points, that is compute $\tilde{x}_i = x_i - \bar{x}$ for all i .
- ▶ Compute the $n \times d$ data matrix X with the centered data points \tilde{x}_i as rows, and the $d \times d$ sample covariance matrix $C = X^t X$.
- ▶ Compute the eigendecomposition $C = V D V^t$.
- ▶ Define V_ℓ as the matrix containing the ℓ largest eigenvectors (i.e., the first ℓ columns of V if the eigs in D are ordered decreasingly).
- ▶ Compute the new data points:
 - ▶ View 2: $y_i = V_\ell^t \tilde{x}_i \in \mathbb{R}^\ell$
 - ▶ View 1: $z_i = P \tilde{x}_i + \bar{x} \in \mathbb{R}^d$ with $P = V_\ell V_\ell^t$

PCA: algorithm using covariance matrix C (2)

Notation:

- ▶ The eigenvectors are called **principal axes** or **principal directions**.
- ▶ In View 1: the distance between a point and its projection is called the **reconstruction error** or **projection error**.

Example: simple Gaussian toy data

`demo_pca.m`

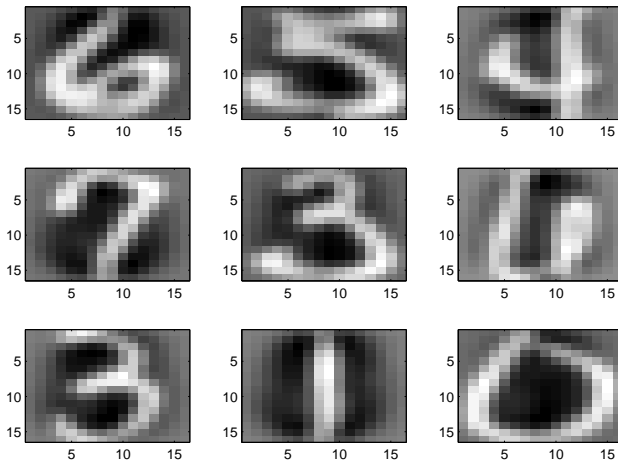
USPS example

USPS handwritten digits, 16 x 16 greyscale images.

→ `demo_pca_usps.m`

USPS example (2)

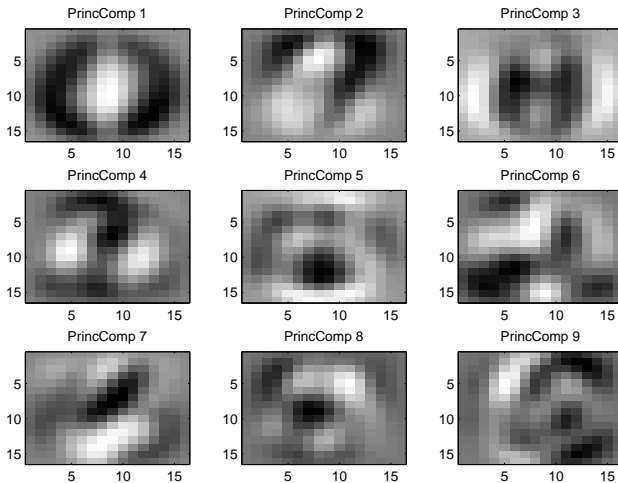
Some digits from the data set



USPS example (3)

The first principal components (computed based on 500 digits):

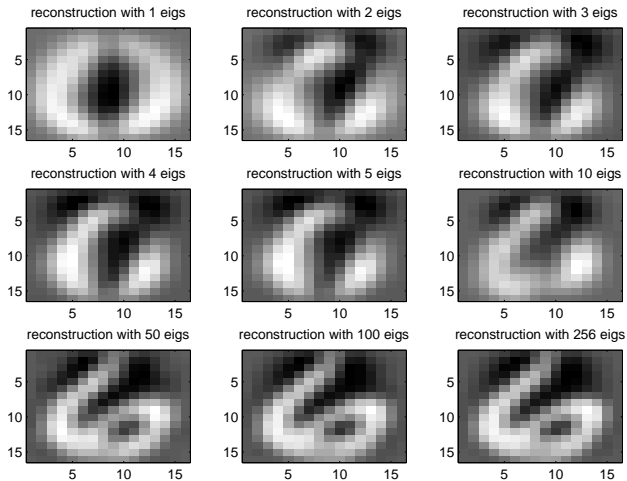
First principal components



USPS example (4)

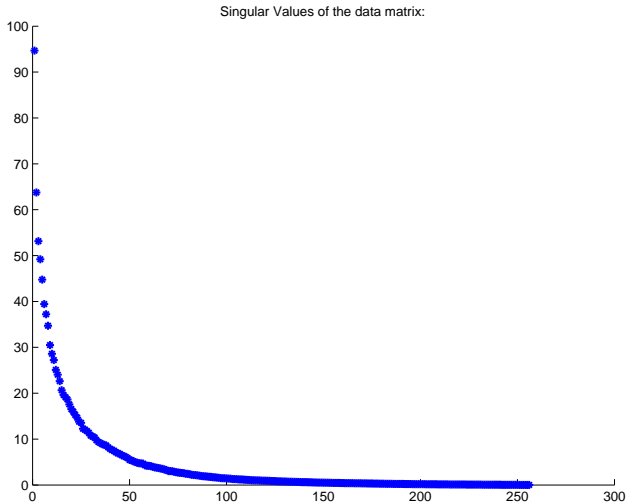
Reconstructing digits:

Reconstructed first digit



USPS example (5)

All eigenvalues:



Eigenfaces

Principal components for a data set of faces:



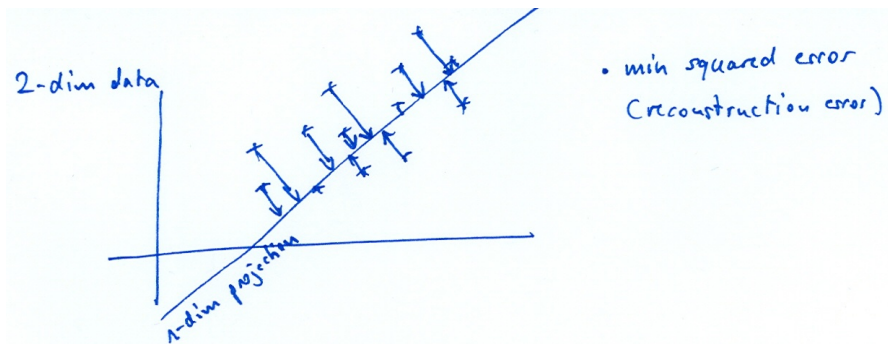
PCA – min squared error approach

Second approach to PCA:

Find a projection π_S on an affine subspace S such that the squared distance between the points and their projections is minimized:

$$\min_S \sum_{i=1}^n \|x_i - \pi_S(x_i)\|^2.$$

PCA – min squared error approach (2)

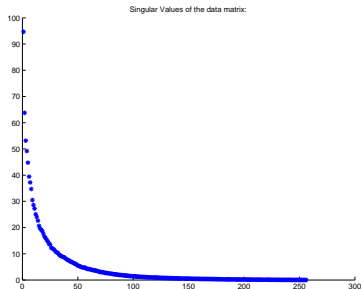


One can prove that this approach leads to exactly the same solution as the one induced by the max-variance criterion (we skip this derivation).

Choosing the parameter ℓ

- Heuristic: look at largest eigenvalues, and take the most “informative” ones. It also can be seen: the reconstruction error is bounded as

$$\sum_i \|x_i - \pi_\ell x_i\|^2 \leq \sum_{k=\ell+1}^n \lambda_k$$



Choosing the parameter ℓ (2)

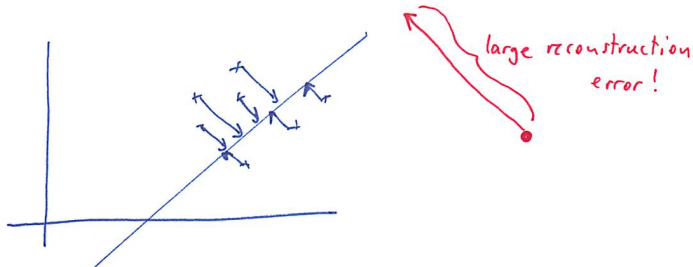
- If PCA is used as a preprocessing step for supervised learning, then use cross validation to set the parameter ℓ !

Note: It is not a priori clear whether it is better to choose ℓ large or small ...

Global!

Keep in mind that PCA optimizes global criteria.

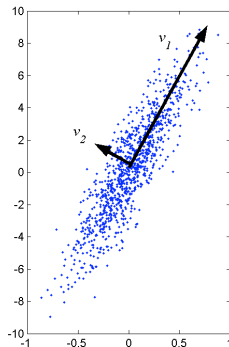
- No guarantees what happens to individual data points. This is different for some other dimensionality reduction methods (such as random projections and Johnson-Lindenstrauss).



- If the sample size is small, then outliers can have a large effect on PCA.

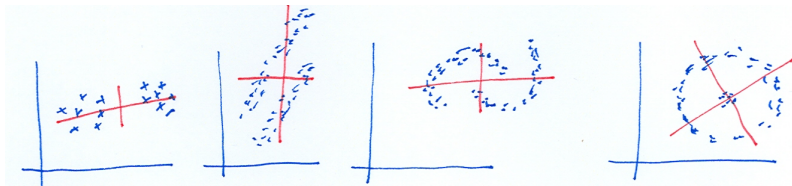
When does it (not) make sense?

- The PCA works best if the data comes from a Gaussian.



When does it (not) make sense? (2)

- But it can have very bad effects if the data is far from Gaussian:



Kernel PCA

Literature on kernel PCA:

- ▶ Chapter 14.2 of Schölkopf and Smola
- ▶ Chapter 6.2. of Shawe-Taylor and Cristianini

Towards kernel PCA

Now we want to kernelize the PCA algorithm to be able to have non-linear principal components.

Observe:

- ▶ PCA uses the covariance matrix — and this matrix inherently uses the actual coordinates of the data points.
- ▶ So how should we be able to kernelize PCA???
- ▶ The solution will be: there is a tight relationship between the covariance matrix and the kernel matrix.

Covariance matrix vs. kernel matrix

Consider centered data points x_1, \dots, x_n , stacked in a data matrix X as rows. Denote the k -th column of the matrix by $X^{(k)}$ (contains the k -th coordinate of all data points). Then:

- Covariance matrix is $C = X^t X$ because

$$C_{kl} = \text{Cov}_{1dim}(X^{(k)}, X^{(l)}) = \sum_{i=1}^n X_i^{(k)} X_i^{(l)} = (X^t X)_{kl}$$

Also note that because $X_i^{(k)} X_i^{(l)} = (x_i x_i^t)_{kl}$ this implies

$$(X^t X) = \sum_{i=1}^n \underbrace{(x_i x_i^t)}_{d \times d}$$

- Kernel matrix is $K = X X^t$

(because $(X X^t)_{ij} = \sum_{k=1}^d x_{ik} x_{jk} = x_i^t x_j = \langle x_i, x_j \rangle$).

Covariance matrix vs. kernel matrix (2)

Consider the SVD of the data matrix X :

$$X = U\Sigma V^t \text{ with } U \in \mathbb{R}^{n \times n}, \Sigma \in \mathbb{R}^{n \times d}, V \in \mathbb{R}^{d \times d}$$

Then:

$$C = X^t X = V \Sigma^t \Sigma V^t$$

$$K = X X^t = U \Sigma \Sigma^t U^t$$

- ▶ Can see that C and K are both psd.
- ▶ The eigenvalues of C and K are “pretty much the same ones” (CAN YOU MAKE IT MORE PRECISE)?
- ▶ The question is now whether this might also be the case for the eigenvectors.

Covariance matrix vs. kernel matrix (3)

We will also need the “inverse” $\Sigma^\#$ of the matrix Σ :

- ▶ For the $n \times d$ matrix Σ with non-zero diagonal values $\sigma_1, \sigma_2, \dots$ let $\Sigma^\#$ be the $d \times n$ matrix with diagonal values $1/\sigma_1, 1/\sigma_2, \dots$. Whenever a diagonal entry of Σ is 0, we also set the corresponding diagonal value of $\Sigma^\#$ to zero.
- ▶ Then $\Sigma^\# \Sigma$ is a $d \times d$ matrix with 1s and 0s on the diagonal (a one wherever there used to be a non-zero diagonal value in Σ). Similarly, the matrix $\Sigma \Sigma^\#$ is a $n \times n$ matrix with 1s and 0s on the diagonal.

See figure next page:

Covariance matrix vs. kernel matrix (4)

Case $d < n$:

$$\Sigma = \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_d \\ 0 & & & \end{pmatrix}, \quad \Sigma^\# = \begin{pmatrix} \sigma_1^{-1} & & \\ & \ddots & \\ & & \sigma_d^{-1} & 0 \\ & & & \ddots \end{pmatrix}, \quad \Sigma^\# \Sigma = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 & \\ & & & \ddots \end{pmatrix}, \quad \Sigma \Sigma^\# = \begin{pmatrix} 1 & & \\ & \ddots & \\ & & 1 & 0 \\ & & & \ddots \end{pmatrix}$$

$n \times d$ $d \times n$ $d \times d$ $n \times n$

If one of the $\sigma_i = 0$,
then we replace σ_i^{-1} by 0.

In this case,
we would have
a 0 on the diagonal
in this matrix.

Case $d > n$ analogously.

Eig of K implies eig of C

Proposition 15 (Eig of K implies eig of C)

Consider a set of points $x_1, \dots, x_n \in \mathbb{R}^d$. Let $\lambda \neq 0$ be an eigenvalue of K with eigenvector $u = (u_1, \dots, u_n)^t \in \mathbb{R}^n$ with $\|u\| = 1$, such that $Ku = \lambda u$. Define

$$\tilde{v} := \frac{1}{\sqrt{\lambda}} X^t u = \frac{1}{\sqrt{\lambda}} \sum_{j=1}^n u_j x_j \in \mathbb{R}^d.$$

Then $\tilde{v} \neq 0$, $\|\tilde{v}\| = 1$ and \tilde{v} is an eigenvector of C with eigenvalue λ , that is $C\tilde{v} = \lambda\tilde{v}$.

Eig of K implies eig of C (2)

Proof:

- ▶ Consider the SVD: $X = U\Sigma V^t$ where V are the eigenvectors of C and U the ones of K .
- ▶ Now compute:

$$X = U\Sigma V^t \quad (\text{now multiply with } \Sigma^\# U^t \text{ from left})$$

$$\implies \Sigma^\# U^t X = \Sigma^\# U^t U \Sigma V^t$$

$$\implies \Sigma^\# U^t X = \Sigma^\# \Sigma V^t \quad (\text{transpose, observe } \Sigma^\# \Sigma \text{ is sym})$$

$$\implies V(\Sigma^\# \Sigma) = X^t U (\Sigma^\#)^t$$

- ▶ The blue matrix is “nearly the identity”. In particular, if the i -th singular value $\sigma_i \neq 0$, then the blue matrix has entry 1 on at the i -th position of the diagonal. So it does not change the i -th column of V .

Eig of K implies eig of C (3)

- ▶ Then if u is the i -th column of U (that is, the i -th eigenvector of K), then the i -th column of the matrix V coincides with out chosen vector $\tilde{v} = \frac{1}{\sigma_i} X^t u$.
- ▶ Observing $\sigma_i = \sqrt{\lambda_i}$ shows the statement about eigenvector and eigenvalue.
- ▶ The statement about the norm:

$$\begin{aligned}
 \|\tilde{v}\|^2 &= \left\| \frac{1}{\sqrt{\lambda}} \sum_j u_j x_j \right\|^2 = \frac{1}{\lambda} \left\langle \sum_j u_j x_j, \sum_i u_i x_i \right\rangle \\
 &= \frac{1}{\lambda} \sum_{i,j} u_i u_j \langle x_i, x_j \rangle \\
 &= \frac{1}{\lambda} \sum_{i,j} u_i u_j k(x_i, x_j) = \frac{1}{\lambda} u^t K u = \frac{1}{\lambda} u^t \lambda u = 1
 \end{aligned}$$

Thus $\|\tilde{v}\| = 1$ (and in particular, $\tilde{v} \neq 0$).



Eig of C implies eig of K

Proposition 16 (Eig of C implies eig of K)

Assume that the points x_i are centered. Let v (with $\|v\| = 1$) and $\lambda \neq 0$ be eigenvector and eigenvalue of C , that is $Cv = \lambda v$. Then the vector $\tilde{u} := \frac{1}{\sqrt{\lambda}}Xv \in \mathbb{R}^n$ is an eigenvector of K with eigenvalue λ , that is $K\tilde{u} = \lambda\tilde{u}$, and it has norm $\|\tilde{u}\| = 1$.

Proof: Similar to the previous SVD proof, just use this derivation:

$$\begin{aligned} X &= U\Sigma V^t \quad (\text{now multiply with } V\Sigma^\# \text{ from right}) \\ \implies XV\Sigma^\# &= U\Sigma V^t V\Sigma^\# \\ \implies \textcolor{red}{XV\Sigma^\#} &= U\Sigma\textcolor{blue}{\Sigma^\#} \end{aligned}$$



(*) Sanity checks

Let's apply the two propositions one after the other:

- Assume $Cv = \lambda v$, set $\tilde{u} := \frac{1}{\sqrt{\lambda}}Xv$. Then by Prop. 16, $K\tilde{u} = \lambda\tilde{u}$. Now set $\tilde{v} = \frac{1}{\sqrt{\lambda}}X^t\tilde{u}$. Then by Prop 15, $C\tilde{v} = \lambda\tilde{v}$. We would now expect $\tilde{v} = v$, which is indeed the case:

$$\tilde{v} = \frac{1}{\sqrt{\lambda}}X^t\tilde{u} = \frac{1}{\lambda}X^tXv = \frac{1}{\lambda}Cv = v$$

(*) Sanity checks (2)

► Dimensions:

- C is a $d \times d$ -matrix, so its eigendecomposition has d eigenvalues.
- K is a $n \times n$ matrix with n eigenvalues.
- But intuitively spoken, we just showed that we can convert the eigenvalues of K to those of C and vice versa.

HOW CAN THIS BE, IF d AND n ARE DIFFERENT???

CONSIDER ALL CASES, $n > d$ and $d > n$...

First algorithm: eigs of C by eigs of K

Assume that the data points are centered in \mathbb{R}^d . Then to compute the ℓ -th eigenvector of C , we can proceed as follows:

- ▶ Compute the kernel matrix K and its ℓ -th eigenvector a
- ▶ Make sure a is normalized to $\|a\| = 1$.
- ▶ Then compute $v := \frac{1}{\sqrt{\lambda}} \sum_i a_i x_i$

Note: this “algorithm” still requires to know the original vectors x_i . However, in practice we don’t want to compute the eigenvectors themselves but just the projections on these eigenvectors. Let’s look at it.

Expressing the projection on eigs of C using K

- Assume we want to project on eigenvector v of C . Have already seen that we can write $v = \sum_i a_i x_i$. Thus:

$$\pi_v(x_i) = v'x_i = \left\langle \sum_j a_j x_j, x_i \right\rangle = \sum_j a_j \langle x_j, x_i \rangle$$

- If we want to project on a subspace spanned by ℓ vectors $v_1, \dots, v_\ell \in \mathbb{R}^d$, compute each of the ℓ coordinates by this formula as well.
- To compute the projections, we only need scalar products 😊. So we can write PCA as a kernel algorithm:

Finally: kernel PCA

Input: Kernel matrix K (computed from abstract data points X_1, \dots, X_n), parameter ℓ

- ▶ Center the data in feature space by computing the centered kernel matrix $\tilde{K} = K - \mathbb{1}_n K - K \mathbb{1}_n + \mathbb{1}_n K \mathbb{1}_n$
- ▶ Compute the eigendecomposition $\tilde{K} = A D A^t$. Let A_k denote the k -th column of A and λ_k the corresponding eigenvalue.
- ▶ Define the matrix V_ℓ which has the columns $A_k / \sqrt{\lambda_k}$, $k = 1, \dots, \ell$
- ▶ Compute the low dim representation points $y_i = (y_i^1, \dots, y_i^\ell)^t$ with the formula $y_i^s = \sum_{j=1}^n v_j^s \tilde{K}_{ji}$ (for all $s = 1, \dots, \ell$).

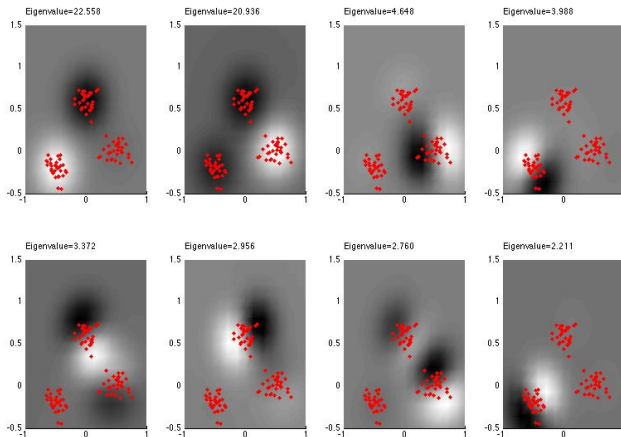
Output: $y_1, \dots, y_n \in \mathbb{R}^\ell$

kPCA toy example: three Gaussians

`demo_kpca_bernhard.m`

- ▶ Data drawn from 2-dim Gaussians (red crosses)
- ▶ kernel used is Gaussian kernel
- ▶ Now can compute the first eigenvectors in kernel feature space
- ▶ For test points, plot the coordinate which results when projecting on this eigenvector in the feature space (grey scale)

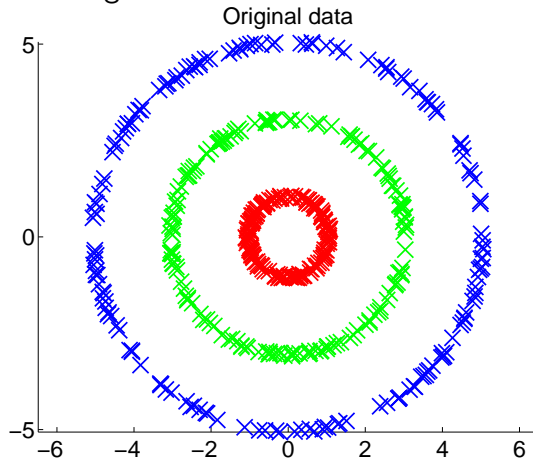
kPCA toy example: three Gaussians (2)



kPCA toy example: rings

demo_kpca_toy.m:

Consider the following three-dimensional data set:



kPCA toy example: rings (2)

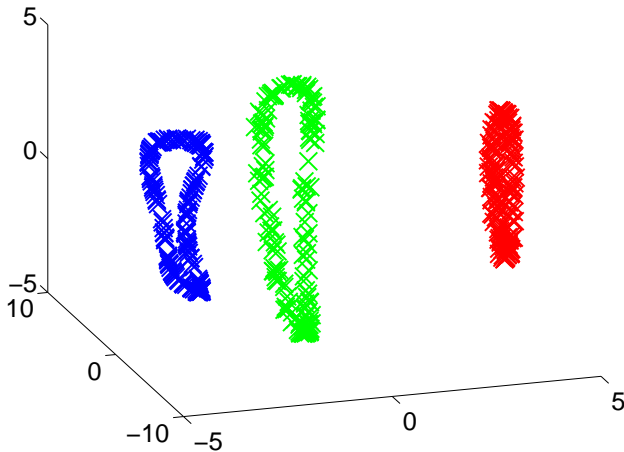
Now apply kPCA:

- ▶ Choose the Gaussian kernel with $\sigma = 2$.
- ▶ Note: we implicitly work in the RKHS, which has n dimensions
- ▶ So it makes sense to choose $\ell = 3$ (even though the original data set just had $d = 2$).

kPCA toy example: rings (3)

Here is the result:

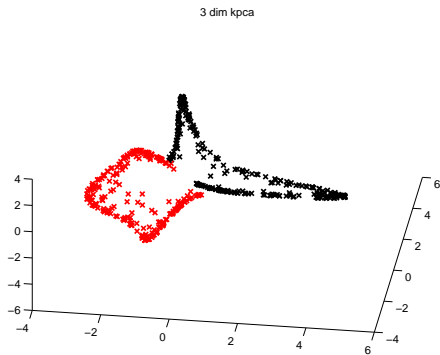
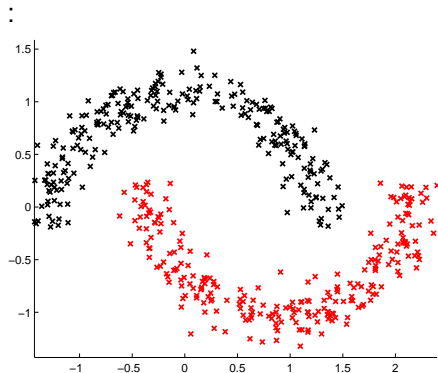
3 dim kpca



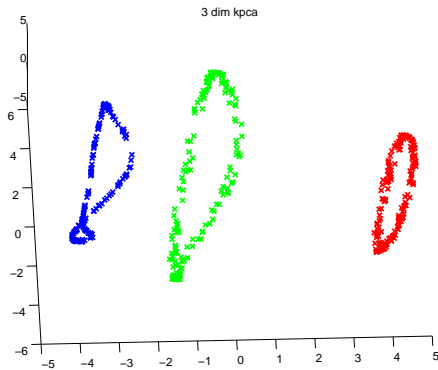
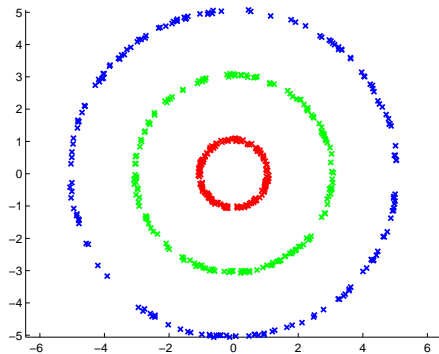
kPCA toy example: rings (4)

Surprising, isn't it???

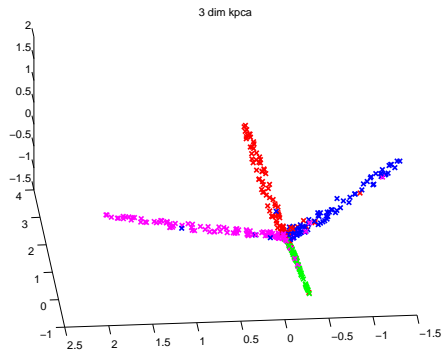
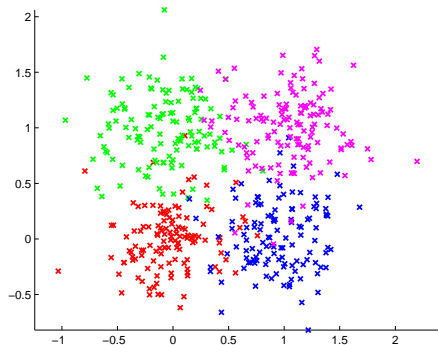
More toy examples



More toy examples (2)



More toy examples (3)



History

- ▶ Classical PCA was invented by Pearson:
On Lines and Planes of Closest Fit to Systems of Points in Space. Philosophical Magazine, 1901.
- ▶ It is one of the most popular “classical” techniques for data analysis.
- ▶ Kernel PCA was invented pretty much 100 years later 😊
B. Schölkopf, A. Smola, and K.-R. Müller. Kernel Principal component Analysis. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, Advances in Kernel Methods–Support Vector Learning, pages 327-352. MIT Press, Cambridge, MA, 1999.

Summary: PCA and kernel PCA

Standard PCA:

- ▶ Technique to reduce the dimension of a data set in \mathbb{R}^d by linear projections.
- ▶ First explanation: throw away dimensions with “low variance”
- ▶ Second explanation: minimize the squared error.
- ▶ Can be computed by an eigendecomposition of the empirical covariance matrix.

Kernel PCA:

- ▶ Use the kernel trick to make PCA non-linear.

(*) Multi-dimensional scaling

Literature:

Multi-dimensional scaling:

- ▶ Is a classic that is covered in many books on data analysis.
- ▶ A whole book on the subject: *Borg, Groenen: Modern multidimensional scaling. Springer, 2005.*

Isomap:

- ▶ The original paper is: *J. Tenenbaum, V. De Silva, J. Langford. A global geometric framework for nonlinear dimensionality reduction. Science, 2000.*

Embedding problem

- ▶ Assume we are given a distance matrix $D \in \mathbb{R}^{n \times n}$ that contains distances $d_{ij} = \|x_i - x_j\|$ between data points.
- ▶ Can we “recover” the points $(x_i)_{i=1, \dots, n} \in \mathbb{R}^d$?

This problem is called **(metric) multi-dimensional scaling**.

A more general way of asking: Given abstract “objects” $x_1, \dots, x_n \in \mathcal{X}$, can we find an **embedding** $\Phi : \mathcal{X} \rightarrow \mathbb{R}^d$ (for some d) such that $\|\Phi(x_i) - \Phi(x_j)\| = d_{ij}$?

DO YOU BELIEVE IT ALWAYS WORKS?

Embedding problem (2)

Answer will be:

- ▶ We can find a correct point configuration if the distances really come from points $\in \mathbb{R}^d$. In this case we say that D is a **Euclidean distance matrix**. See next slide for how this works.
- ▶ For general distance matrices D , we cannot achieve such an embedding without distorting the data. There is a huge bulk of literature on approximate embeddings, but we won't cover it in this lecture.

Embedding problem (3)

WHY DO YOU THINK SUCH AN EMBEDDING MIGHT BE USEFUL?

Embedding problem (4)

Why might we be interested in such an embedding?

- ▶ Visualization!
- ▶ Many algorithms are just defined for Euclidean data. If we want to apply them, we need to find a Euclidean representation of our data.
- ▶ Identify low-dimensional structure, see Isomap below.

What might be problematic about it?

- ▶ We might introduce distortion to the data ...

MDS in various flavors

- ▶ **Classic MDS:** we assume that the given distance matrix is Euclidean.
 - ▶ If the matrix is Euclidean, embedding will be exact.
 - ▶ If the matrix is not Euclidean, embedding will make some errors.
- ▶ **Metric MDS:** we are given any distance matrix (might be non-Euclidean). We try to find an embedding that approximately preserves all distances.
 - ▶ In case the original matrix is non-Euclidean, perfect reconstruction is impossible, so we definitely will make approximation errors.
 - ▶ In case the original matrix is Euclidean, we might still make errors due to the formulation of the problem, see below.
- ▶ **Non-metric MDS:** we are not given distances, but ordinal information, see below.

Classic MDS

Assume we are given a Euclidean distance matrix D . Will now see how to express the entries of the Gram matrix $S = (\langle x_i, x_j \rangle)_{ij=1, \dots, n}$ in terms of entries of D :

- By definition:

$$\begin{aligned} d_{ij}^2 &= \|x_i - x_j\|^2 = \langle x_i - x_j, x_i - x_j \rangle \\ &= \langle x_i, x_i \rangle + \langle x_j, x_j \rangle - 2\langle x_i, x_j \rangle \end{aligned}$$

- Rearranging gives

$$\langle x_i, x_j \rangle = \frac{1}{2} \left(\underbrace{\langle x_i, x_i \rangle}_{=d(0, x_i)^2} + \underbrace{\langle x_j, x_j \rangle}_{=d(0, x_j)^2} - d_{ij}^2 \right)$$

Classic MDS (2)

- ▶ We are free to choose the origin 0 as we want. For simplicity, we choose the first data point x_1 as the origin. This gives:

$$\langle x_i, x_j \rangle = \frac{1}{2} \left(d_{1i}^2 + d_{1j}^2 - d_{ij}^2 \right)$$

- ▶ So we can express the entries of the Gram matrix S with $s_{ij} = \langle x_i, x_j \rangle$ in terms of the given distance values.
- ▶ Because S is positive definite, we can decompose S in the form $S = XX^t$ where $X \in \mathbb{R}^{n \times d}$.
EXERCISE: HOW EXACTLY DO YOU DO THIS? WHAT IS THE DIMENSION d GOING TO BE?
- ▶ The rows of X are what we are looking for, that is we set the embedding of point x_i as the i -th row of the matrix X .

Classic MDS implementation

This is how it finally works:

- ▶ Compute the matrix S with $s_{ij} = \frac{1}{2} \left(d_{1i}^2 + d_{1j}^2 - d_{ij}^2 \right)$.
- ▶ Compute the eigenvalue decomposition $S = V \Lambda V^t$.
- ▶ Define $X = V \sqrt{\Lambda}$.

Alternatively, if you want to fix some dimension $d \leq n$, set V_d to be the first d columns of V and Λ_d the $d \times d$ diagonal matrix with the first d eigenvalues on the diagonal, and then set $X = V_d \sqrt{\Lambda_d}$.

- ▶ Row i of X then gives the coordinates of the embedded point x_i .

Classic MDS implementation (2)

How to choose d ?

- ▶ If the data points come from \mathbb{R}^d , then the matrix S is going to have rank d , that is there are d eigenvalues > 0 and $n - d$ eigenvalues equal to 0.
- ▶ Hence, looking at the spectrum of S gives you an idea to choose d . In case of classic MDS, you can just read off d from the matrix, in the more general case of metric MDS you simply “choose it reasonable” (as in PCA).

Demos

`demo_mds.m`

Metric MDS

Metric MDS refers to the problem where the distance matrix D is no longer Euclidean, but we still believe (hope) that a good embedding exists.

- ▶ If the distance matrix D is not Euclidean, we will not be able to recover an exact embedding.
- ▶ Instead, one defines a “stress function”. Below is an example for such a stress function:

$$\text{stress}(\text{embedding}) = \frac{\sum_{ij} (\|x_i - x_j\| - d_{ij})^2}{\sum_{ij} \|x_i - x_j\|}$$

Many more stress functions are considered in the literature.

- ▶ Then we try to find an embedding x_1, \dots, x_n with small stress by a standard non-convex optimization algorithm, say gradient descent.

Metric MDS (2)

When using metric MDS, there are two sources of error:

- ▶ The distance matrix is not Euclidean, so we will not be able to recover a perfect embedding.
- ▶ The optimization problems are highly non-convex and suffer all kinds of problems of local optima.

Using metric MDS only makes sense if the data is “nearly Euclidean”, and results should always be treated with care.

Non-metric MDS

- ▶ Instead of distance values, we are just given distance comparisons, that is we know whether $d_{ij} < d_{ik}$ or vice versa.
- ▶ The task is then to find an embedding such that these ordinal relationships are preserved.
- ▶ Our group is working on this problem 😊



Which of the bottom images is most similar to the top image?

History of MDS

- ▶ Metric MDS: Torgerson (1952) - The first well-known MDS proposal. Fits the Euclidean model.
- ▶ Non-metric MDS: Shepard (1962) and Kruskal (1964)

Outlook: general embedding problems

There exists a huge literature on embedding metric spaces in Euclidean spaces:

- ▶ Given certain assumptions on the metric ...
 - ▶ In what space can I embed (dimension???)
 - ▶ What are the guarantees on the distortion?

Some literature pointers:

- ▶ Theorem of Bourgain: Any n -point metric space can be embedded in Euclidean space with distortion $O(\log n)$. By the theorem of Johnson-Lindenstrauss, we can achieve dimensionality of $O(\log n)$ as well.
- ▶ An overview paper on the area is: *Piotr Indyk and Jiri Matousek. Low-distortion embeddings of finite metric spaces. In: Handbook of Discrete and Computational Geometry, 2004.*

Summary MDS

- ▶ Given a distance matrix D , MDS tries to construct an embedding of the data points in \mathbb{R}^d such that the distances are preserved as well as possible.
- ▶ If D is Euclidean, a perfect embedding can easily be constructed.
- ▶ If D is not Euclidean, MDS tries to find an embedding that minimizes the “stress”. The resulting problem is highly non-convex. Solutions should be treated with care.

(*) Random projections and the Theorem of Johnson-Lindenstrauss

Literature:

- ▶ We take the proof from the following paper:
S. Dasgupta, A. Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. Random Structures & Algorithms 22, 2003.
- ▶ The following paper has the “simpler” random projections: D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. Journal of Computer and System Sciences 66, 2003.

Random projections — the general idea

Given: points $X_1, \dots, X_n \in \mathbb{R}^d$, d large

We are going to construct a mapping $\pi : \mathbb{R}^d \rightarrow \mathbb{R}^k$, $k \ll d$ such that all distances are “nearly preserved”:

$$\|x_i - x_j\|_{\mathbb{R}^d} \approx \|\pi(x_i) - \pi(x_j)\|_{\mathbb{R}^k}$$

The mapping π will be a random projection.

This is cool because it means that we can work in much lower-dimensional spaces and don't lose much information by doing so.

Theorem of Johnson-Lindenstrauss

More precisely, we are going to prove the following theorem:

Theorem 17 (Johnson-Lindenstrauss, 1984)

Let $0 < \varepsilon < 1$, $n \in \mathbb{N}$. Assume that

$$k > \frac{4 \log(n)}{\varepsilon^2/2 - \varepsilon^3/3}.$$

Then, for any set of n points $x_1, \dots, x_n \in \mathbb{R}^d$ (and any dimension d) there exists a map $\pi : \mathbb{R}^d \rightarrow \mathbb{R}^k$ such that for all $i \neq j$:

$$(1 - \varepsilon) \|x_i - x_j\|_{\mathbb{R}^d}^2 \leq \|\pi(x_i) - \pi(x_j)\|_{\mathbb{R}^k}^2 \leq (1 + \varepsilon) \|x_i - x_j\|_{\mathbb{R}^d}^2 \quad (*)$$

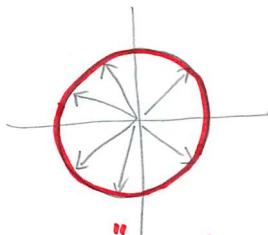
Theorem of Johnson-Lindenstrauss (2)

Intuition:

- ▶ Given a set of n points in \mathbb{R}^d , d arbitrarily large
- ▶ d arbitrarily large means: in the worst case, $d = n - 1$.
Reason: if $d > n - 1$, then the n points always sit in a subspace of dimension at most $n - 1$
- ▶ fix a deviation ε that you are going to tolerate
- ▶ Then you can always embed these points into a space of dimension $k \approx \log(n)/\varepsilon^2$ and keep distances close to the original ones
- ▶ That is, you can reduce the dimension from $\Omega(n)$ (general case) to $\Omega(\log(n))$!
- ▶ In particular, all algorithms that have running time exponential in d have running time polynomial in d after applying a random projection (at the cost of slight perturbations)

Random projections — definition

- ▶ We want to project onto a vector $z \in \mathbb{R}^d$ where z is random.
- ▶ We want that all directions z are equally likely, that is z has to be drawn from the uniform distribution on the sphere in \mathbb{R}^d .



"direction" = point on sphere

Random projections — definition (2)

Definition:

1. Let z be a vector drawn from the uniform distribution on the unit sphere in \mathbb{R}^d . Then

$$\pi_z : \mathbb{R}^d \rightarrow \mathbb{R}, \pi_z(v) = \langle z, v \rangle_{\mathbb{R}^d}$$

is called a random projection on the 1-dimensional space \mathbb{R} .

Random projections — definition (3)

2. Let z_1, \dots, z_k be random vectors drawn as above. Denote the subspace spanned by these vectors by S . Then the projection π_S on S is called a random projection on \mathbb{R}^k .

(To compute this projection, build the $k \times d$ -matrix Z that contains the vectors z_i as rows, and then compute $\pi(v) := Z \cdot v$.)

Random projections on \mathbb{R}^d — expectations

Proposition 18 (Expected length after random projection on \mathbb{R}^d)

Let $x \in \mathbb{R}^d$ a fixed vector with $\|x\| = 1$. Consider its random projection π_v on the random vector $v = (v_1, \dots, v_d)'$. Then:

$$E(|\pi_v(x)|^2) = 1/d.$$

Intuitive meaning: Projection on a random space decreases all lengths by a factor of $\sqrt{1/d}$, on average.

Proof. W.l.o.g. assume that $x = (1, 0, \dots, 0) =: e_1 \in \mathbb{R}^d$ (can do so because of rotational symmetry of the distribution of random vectors). Then:

Random projections on \mathbb{R} — expectations (2)

$$E(|\pi_v(x)|^2) = E(|\pi_v(e_1)|^2) = E(|\langle v, e_1 \rangle|^2) = E(|v_1|^2) \stackrel{!}{=} \frac{1}{d}$$

To see the last equality: By assumption, v is a random vector with length 1. Thus:

$$1 = \|v\|^2 = E\|v\|^2 = E\left(\sum_{j=1}^d v_j^2\right) = \sum_j E(v_j^2).$$

By rotational symmetry of the distribution of v , all components v_i have the same expectation, thus $E(v_i^2) = 1/d$ for all $i = 1, \dots, d$.



Random projections on \mathbb{R}^k — expectations

Theorem 19 (Expected length after random projection on \mathbb{R}^k)

Let $x \in \mathbb{R}^d$ a fixed vector with $\|x\| = 1$. Consider its random projection π_S on a k -dim random space S . Then $E\|\pi_S(x)\|^2 = k/d$.

Intuitive meaning: Projection on a random space decreases all lengths by a factor of $\sqrt{k/d}$, on average.

Proof. Let R be the rotation that maps S on the subspace E_k spanned by e_1, \dots, e_k . Because rotations do not change norms we have

$$E(\|\pi_S(x)\|^2) = E\|\pi_{RS}(Rx)\|^2$$

Random projections on \mathbb{R}^k — expectations (2)

We now define $v := Rx$. The trick is now:

- ▶ The length of a unit vector when projected on a random k -dim subspace has same distribution as length of a random unit vector when projected on a fixed k -dim subspace.
- ▶ So we consider v as a random vector (the image of a vector under a random rotation), whereas we consider $RS = E_k$ is fixed.

This leads to:

$$\begin{aligned} E_S(\|\pi_S(x)\|^2) &= E_S(\|\pi_{RS}(Rx)\|^2) = E_v(\|\pi_{E_k}(v)\|^2) \\ &= E_v\left(\left\|\sum_{i=1}^k v_i e_i\right\|^2\right) = E_v\left(\sum_{i=1}^k |v_i|^2\right) \\ &= \sum_{i=1}^k E_v |v_i|^2 = \sum_{i=1}^k \frac{1}{d} = \frac{k}{d} \end{aligned}$$

Concentration around the mean

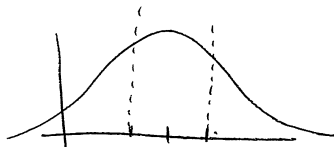
So far we know that the expected squared length of $\pi_S(x)$ is k/d . The question is now how much variance this length has (how much does it fluctuate around its mean?).

↪ concentration statement:

Concentration around the mean (2)

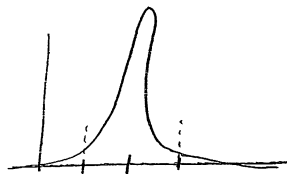
Intuitively:

A random variable U is concentrated if its deviation from the expectation $E(U)$ is very small with high probability.



$E(X)$

large deviations occur often
not concentrated



$E(X)$

large deviations are extremely
unlikely \leadsto concentrated

Concentration around the mean (3)

Typical concentration statements look as follows:

- Additive deviations decrease exponentially:

$$P(|U - E(U)| \geq \varepsilon) \leq \exp(-const \cdot \varepsilon)$$

- or multiplicative deviations decrease exponentially:

$$P(U \leq t \cdot E(U)) \leq \exp(-const_1 \cdot t) \text{ for } t < 1$$

$$P(U \geq t \cdot E(U)) \leq \exp(-const_2 \cdot t) \text{ for } t > 1$$

Random projections: concentration

Proposition 20 (Concentration properties of random projections)

For any $0 < \beta < 1$,

$$P\left(\|\pi_S(x)\|^2 \leq \beta \frac{k}{d}\right) \leq \exp\left(\frac{k}{2}(1 - \beta + \ln \beta)\right)$$

For any $\beta > 1$,

$$P\left(\|\pi_S(x)\|^2 \geq \beta \frac{k}{d}\right) \leq \exp\left(\frac{k}{2}(1 - \beta + \ln \beta)\right)$$

Intuitively: The probability that $\|\pi_S(x)\|^2$ deviates by more than a factor β from its expectation is exponentially small.

Random projections: concentration (2)

We skip the proof of this statement (it uses “concentration inequalities”).

Proof of Johnson-Lindenstrauss

Proof idea:

- ▶ Use a random projection to \mathbb{R}^k (where k is chosen as in the theorem) and rescale it: $\sqrt{d/k} \cdot \pi_S$
- ▶ By Proposition 19 (expectation): a vector of length 1 still has expected length 1 after the projection.
- ▶ By Proposition 20 (concentration): the actual length will be very close to 1, with high probability.

The details:

Proof of Johnson-Lindenstrauss (2)

Consider the two points $x_i, x_j \in \mathbb{R}^d$, and let $\pi : \mathbb{R}^d \rightarrow \mathbb{R}^k$ a random projection (where k is chosen as in the theorem). π will play the role of the mapping f in the theorem, that is we want to prove that $(*)$ from the theorem holds for $\pi(x_i) - \pi(x_j)$.

Observe that by linearity, $\|\pi(x_i) - \pi(x_j)\| = \|\pi(x_i - x_j)\|$.

We call $z := x_i - x_j$.

Step 1: we prove that

$$P\left(\|\pi(z)\|^2 \geq (1 + \varepsilon) \frac{k}{d} \|z\|^2\right) \leq 1/n^2$$

To see this, apply the concentration proposition:

Proof of Johnson-Lindenstrauss (3)

$$P(\|\pi(z)\|^2 \geq (1 + \varepsilon) \frac{k}{d} \|z\|^2) \leq \exp \left(\frac{k}{2} (1 - (1 + \varepsilon) + \ln(1 + \varepsilon)) \right)$$

Use $\ln(1 + \varepsilon) \leq \varepsilon - \varepsilon^2/2 + \varepsilon^3/3$ and plug in the formula for k from the theorem. Then we obtain

$$\begin{aligned} & \exp \left(\frac{k}{2} (1 - (1 + \varepsilon) + \ln(1 + \varepsilon)) \right) \\ & \leq \exp \left(\frac{2 \log(n)}{\varepsilon^2/2 - \varepsilon^3/3} (-\varepsilon + \varepsilon - \varepsilon^2/2 + \varepsilon^3/3) \right) \\ & = \exp(-2 \log n) = 1/n^2 \end{aligned}$$

Proof of Johnson-Lindenstrauss (4)

Consequence of Step 1:

$$P((*) \text{ does NOT hold for } x_i \text{ and } x_j) = 2/n^2$$

(factor 2 because of upper and lower deviations).

Proof of Johnson-Lindenstrauss (5)

Second step: Now want this to hold for all pairs x_i, x_j :

$$\begin{aligned} P((*) \text{ does NOT hold for at least one pair } x_i, x_j) \\ &= P((*) \text{ does NOT hold for } x_1, x_2) \text{ OR } (\dots \text{ NOT for } x_1, x_3) \text{ OR } \dots \\ &\leq \sum_{i \neq j} P((*) \text{ does NOT hold for } x_i, x_j) \\ &= \sum_{i \neq j} \frac{2}{n^2} = \frac{n(n-1)}{2} \frac{2}{n^2} = 1 - \frac{1}{n} \end{aligned}$$

Consequence:

$$P((*) \text{ holds for all pairs of points}) = \frac{1}{n}.$$

Proof of Johnson-Lindenstrauss (6)

Final step: existence

The probability that a single random projection does what we want is positive. Thus there *exists* a mapping with the desired properties. This proves the theorem.



Proof of Johnson-Lindenstrauss (7)

Comments:

- ▶ To actually *find* a suitable projection, we try random projections until we found one that does the job.
- ▶ Usually, we have to try $O(n)$ random projection until we find one that satisfies (*) for all pairs of points.

Implementing random projections — wrong way

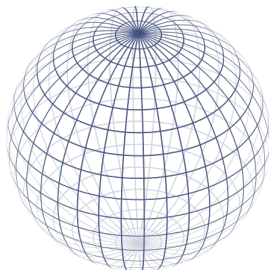
How can we implement drawing a vector uniformly from the sphere?

Note: the following method is WRONG:

- ▶ Consider points in polar coordinates
- ▶ Pick the two angles φ and ψ uniform from $[0, 2\pi[$.

WHY?

Implementing random projections — wrong way (2)

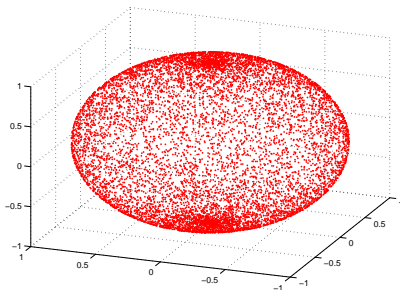


Reason:

- ▶ depending on where you are, the area that covers the same angle is small (close to the poles) or large (close to the equator)
- ▶ Thus, you would end up with more points close to the poles than close to the equator.

Implementing random projections — wrong way (3)

Here are some points from the sphere, sampled according to the wrong paradigm:



Samples accumulate at the poles!

Implementing random projections — correct ways

A simple method that is correct:

To project on a one-dimensional space:

- ▶ Draw a d -dim vector according to the d -dim Normal distribution: $z_1 = \text{randn}[1, d]$
- ▶ normalize it to have length 1: $z = z_1 / \text{norm}(z_1)$;

Draw k linearly independent vectors to implement a k -dim random projection:

- ▶ Draw k vectors z_1, \dots, z_k as described above and normalize them
- ▶ If d is large, these vectors are usually linearly independent (that is, they form a basis). If not, repeat the process. Very few repetitions should be enough.
- ▶ Then orthonormalize the basis for easier implementation.

Implementing random projections — correct ways (2)

A very elegant, simple method, discovered in Achlioptas (2003):

Construct a matrix R of size $d \times k$ with independent entries ± 1 with probability $1/2$ each (e.g., we throw a coin for each entry).

If d is large, the matrix has the following properties:

- ▶ All column vectors have nearly the same length \sqrt{d}
- ▶ All column vectors of R are nearly orthogonal to each other

We can use this matrix as a projection matrix in the Johnson-Lindenstrauss theorem. This leads to the following theorem:

Implementing random projections — correct ways (3)

Theorem 21 (Johnson-Lindenstrauss, Version by Achlioptas)

- ▶ Given $x_1, \dots, x_n \in \mathbb{R}^d$, let X be the $d \times n$ matrix with the vectors x_i as columns.
- ▶ Let $\varepsilon, \beta > 0$ and $k = \log(n) \cdot (4 + 2\beta) / (\varepsilon^2/2 - \varepsilon^3/3)$.
- ▶ Let R be a $k \times d$ matrix with independent entries ± 1 as described above.
- ▶ Construct the projected data points $Y := R \cdot X \cdot 1/\sqrt{k}$.
- ▶ Then, with probability at least $1 - n^{-\beta}$, property (*) from the Johnson-Lindenstrauss theorem holds for all pairs of points (x_i, x_j) .

Final comments

Why is Johnson-Lindenstrauss so cool?

- ▶ The running time of many computer science algorithms depends exponentially on the dimension d of the space.
- ▶ We can use JL to reduce the dimension to $\log d$. Afterwards, the algorithm runs in time polynomial in d .

Final comments (2)

Many generalizations of this theorem exist, let's just mention two of them:

- ▶ Instead of using the Euclidean distance on \mathbb{R}^d , we can also use an L_p distance. A very similar mechanism then works using p -stable distributions (the Gaussian distribution is just the special case $p = 2$).
- ▶ Bourgain's theorem: Any metric space of n points can be embedded into a Euclidean space of dimension $O(\log(n))$ with distortion at most $O(\log(n))$.

(*) Isomap

Literature:

- ▶ Original paper:
J. Tenenbaum, V. De Silva, J. Langford. A global geometric framework for nonlinear dimensionality reduction. Science, 2000.

Isomap

We often think that data is “inherently low-dimensional”:

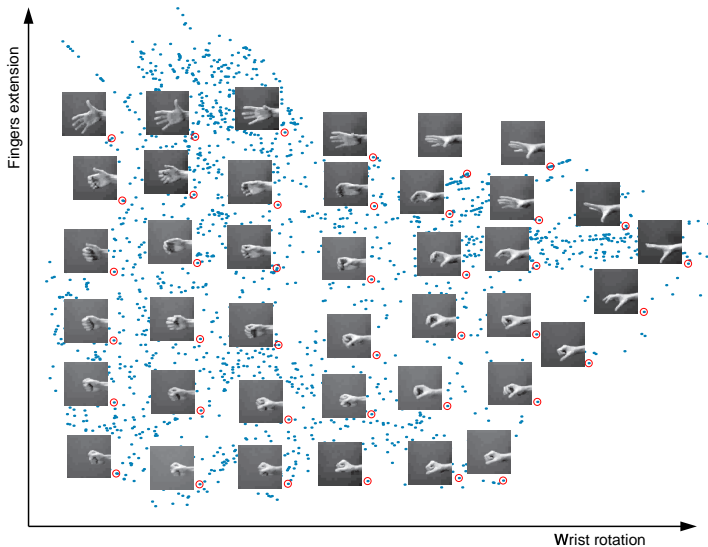
- Images of a tea pot, taken from all angles. Even though the images live in \mathbb{R}^{256} , say, we believe they sit on a manifold corresponding to a circle:



Isomap (2)

- ▶ A phenomenon generates very high-dimensional data, but the “effective number of parameters” is very low

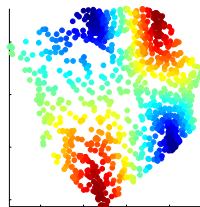
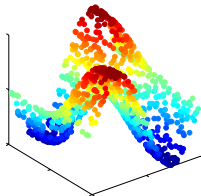
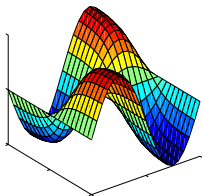
Isomap (3)



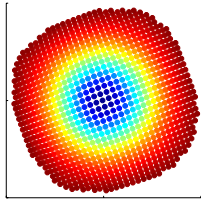
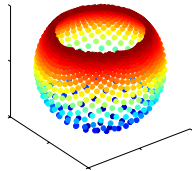
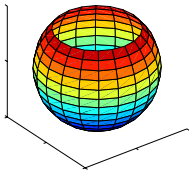
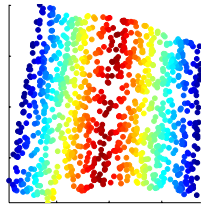
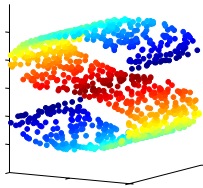
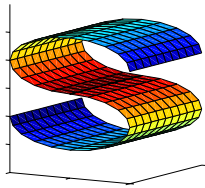
Isomap (4)

More abstractly:

- ▶ We assume that the data lives in a high-dimensional space, but effectively just sits on a low-dimensional manifold
- ▶ We would like to find a mapping that recovers this manifold.
- ▶ If we could do this, then we could reduce the dimensionality in a very meaningful way.



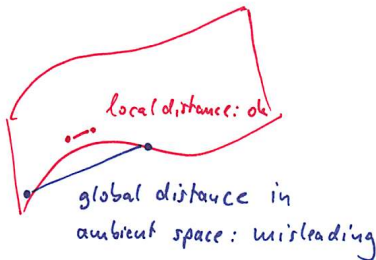
Isomap (5)



The Isomap algorithm

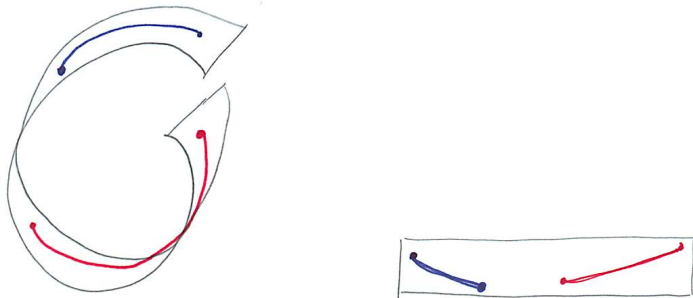
Intuition:

- ▶ In a small local region, Euclidean (extrinsic) distances between points on a manifold approximately coincide with the intrinsic distances. We want to keep the local distances unchanged.
- ▶ This is no longer the case for large distances: we want to keep the intrinsic (geodesic) distances rather than the ones in the ambient space.



The Isomap algorithm (2)

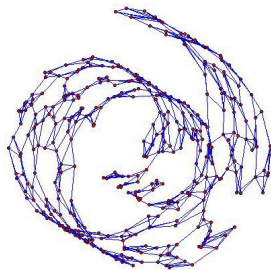
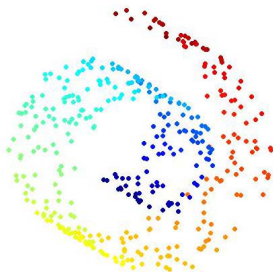
- If we want to “straighten” a manifold, we need to embed it in such a way that the Euclidean distance after embedding corresponds to the geodesic distance on the manifold.



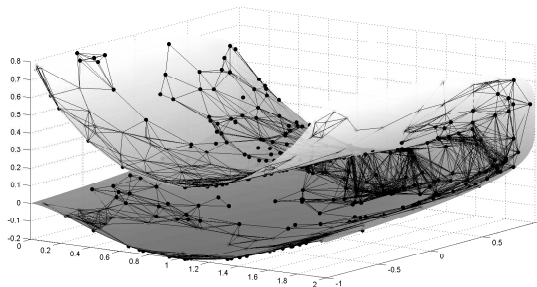
So we would like to discover the geodesic distances in the manifold.

The Isomap algorithm (3)

- ▶ To discover the geodesic distances in the manifold:
 - ▶ Build a kNN graph on the data
 - ▶ Use the shortest path distance in this graph.
 - ▶ Idea is: it goes “along” the manifold.



The Isomap algorithm (4)



(figure by Matthias Hein)

The Isomap algorithm (5)

The algorithm:

- ▶ Given some abstract data points X_1, \dots, X_n and a distance function $d(x_i, x_j)$.
- ▶ Build a k -nearest neighbor graph where the edges are weighted by the distances. These are the local distances.
- ▶ In the kNN graph, compute the shortest path distances d_{sp} between all pairs of points and write them in the matrix D . They correspond to the geodesic distances.
- ▶ Then apply metric MDS with D as input. Finds embedding that preserves the geodesic distances.

Theoretical guarantees

In the original paper (supplement) the authors have proved:

- ▶ If the data points X_1, \dots, X_n are sampled uniformly from a “nice” manifold, then as $n \rightarrow \infty$ and $k \approx \log n$, the shortest path distances in the kNN graph approximate the geodesic distances on the manifold.
- ▶ Under some geometric assumptions on the manifold, MDS then recovers an embedding with distortion converging to 0.

(Attention, the dimension is an issue here. Typically, we cannot embed a manifold without distortion in a space of the intrinsic dimension, we need to choose the dimension larger).

History

- ▶ Manifold methods became fashionable in machine learning in the early 2000s.
- ▶ Isomap was invented in 2000.
- ▶ Since then, a large number of manifold-based dimensionality reduction techniques has been invented:
Locally linear embedding, Laplacian eigenmaps, Hessian eigenmaps, Diffusion maps, Maximum Variance Unfolding, and many more ...

Summary Isomap

- ▶ Unsupervised learning technique to extract the manifold structure from distance / similarity data
- ▶ Intuition: local distances define the intrinsic geometry, shortest paths in a kNN graphs correspond to geodesics.
- ▶ MDS then tries to find an appropriate embedding.

(*) t-SNE

Literature:

- ▶ Original paper: van der Maaten, Hinton: Visualizing data using t-SNE, JMLR 2008
- ▶ An online tutorial: Wattenberg, Viegas, Johnson: How to Use t-SNE Effectively, Distill 2016
- ▶ Kobak, Berens: The art of using t-SNE for single cell transcriptomics. Nature communications, 2019.
- ▶ Some theoretical analysis:
 - ▶ Clustering with t-SNE, Provably. Linderman, Steinerberger, SIAM Journal on Mathematics of Data Science
 - ▶ Arora, Hu, Kothari: An Analysis of the t-SNE Algorithm for Data Visualization, COLT 2018

General setup

t-SNE (pronounced “tee-snee”), t-Stochastic Neighbor Embedding:

- ▶ Method for non-linear dimensionality reduction
- ▶ Used mainly for data visualization / exploration
- ▶ Supposed to “identify” cluster structure.

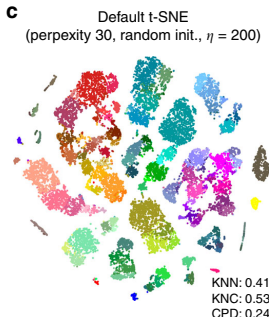


Figure from Kobak/Berens paper

General setup (2)

General hope: it brings out the cluster structure much better than MDS (left) or PCA (middle).

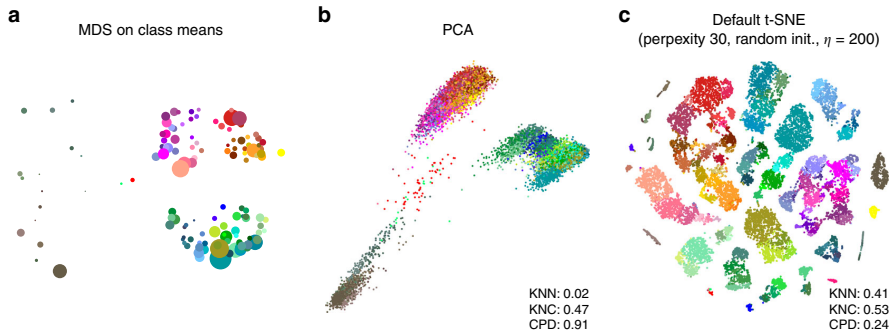


Figure from Kobak/Berens paper

(Question: at the cost of what? See later.)

General idea

- ▶ We start with points in a high-dim space \mathbb{R}^d , we typically embed in \mathbb{R}^2 (for visualization).
- ▶ We define a (non-symmetric) similarity function $p_{i|j}$ on the input space using a Gaussian kernel, see next slide. For each point i , we will interpret the similarities $p_{i|j}$ as a probability distribution.
- ▶ We define a (non-symmetric) similarity function $q_{i|j}$ on the output space using a heavy-tailed t -distribution.
- ▶ Then we minimize the Kullback-Leibler divergence between the two by a gradient descent algorithm.

The similarity in the input space

- ▶ The non-symmetric similarity between i and j will be modeled by a **Gaussian kernel**:

$$p_{i|j} = \frac{\exp(-\|x_i - x_j\|^2 / (2\sigma_i^2))}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / (2\sigma_i^2))}$$

- ▶ The similarities are normalized to sum to 1, so they can be interpreted as a probability distribution.
- ▶ For each point i , we **choose a different value of σ_i** . The rough idea is to choose σ_i such that the data point i is reasonably similar to a predefined, fixed number of neighbors (a bit like the parameter k in kNN methods).
- ▶ This is measured in a somewhat funny way (in my opinion):

The similarity in the input space (2)

- ▶ Fix a value of “perplexity” (\approx number of neighbors)
- ▶ Compute the entropy of the current input distribution at point i (depending on the current value of σ_i):

$$H = - \sum_{i \neq j} p_{j|i} \log_2 p_{j|i}$$

- ▶ Compute the “perplexity” $\mathcal{P} = 2^H$
- ▶ Choose the parameter σ_i of the Gaussian kernel such that the perplexity matches the pre-specified value (easy: use binary search, as perplexity monotonically depends on σ).

Similarity in the output space

In earlier versions, people used the Gaussian kernels on the output space as well:

$$q_{i|j} = \frac{\exp(-\|x_i - x_j\|^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2)}$$

However, this led to undesired effects (the algorithm was called SNE).

The breakthrough was to use a more heavy-tailed (more robust) similarity on the output space, turning SNE into t-SNE:

Similarity in the output space (2)

We model the similarity in the output space by a **heavy-tailed t -distribution** (with one degree of freedom; same as Cauchy distribution):

$$q_{i|j} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_i - y_k\|^2)^{-1}}$$

There is no parameter to tune here, at least in the vanilla version.

The objective function

Given the input similarities $p_{i|j}$, the goal is to find an embedding whose output similarities $q_{i|j}$ match the input ones as good as possible. This is measured by the **Kullback-Leibler divergence**:

$$\begin{aligned} \text{cost}(\text{embedding}) &= \sum_i KL(P_i || Q_i) \\ &= \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}} \end{aligned}$$

This cost function is minimized by gradient descent.

The objective function (2)

Interpretation:

- ▶ Large costs for points that are close in input space but far in output space; Small cost for points that are far in input space but close in output space
- ▶ Looking closer at the gradient that is being optimized, one can identify two components: attractive forces and repulsive forces, similarly to a system of springs.
 - ▶ All points are repelled from each other.
 - ▶ Points that are neighbors are also attracted to each other (governed by perplexity)

T-SNE can be cool

In many cases it has been observed that t-SNE produces plots that show true cluster structure:



Figure 1.1. *t-SNE* output (left) and colored by some known ground truth (right).

Figure from Lindermann et al.

T-SNE can be cool (2)

Theoretical results by Lindermann et al and Arora et al show that if there exists a ground truth clustering consisting of highly spherical clusters that are very well separated from each other, then a variant of t-SNE has a high likelihood of finding it:

Theorem 1.5 (Informal, see Theorem 3.1 for a formal version). *Let $\mathcal{X} = \{x_1, x_2, \dots, x_n\} \subset \mathbb{R}^d$ be γ -spherical and γ -well-separated clusterable data with $\mathcal{C}_1, \mathcal{C}_2, \dots, \mathcal{C}_k$ defining the individual clusters. Then, t-SNE with early exaggeration on input \mathcal{X} outputs a full visualization of \mathcal{X} with high probability.*

Theorem from Arora et al

Choice of parameters is crucial!

While the algorithm sounds really simple and can produce cool plots, the output really **depends a lot on the choice of parameters:**

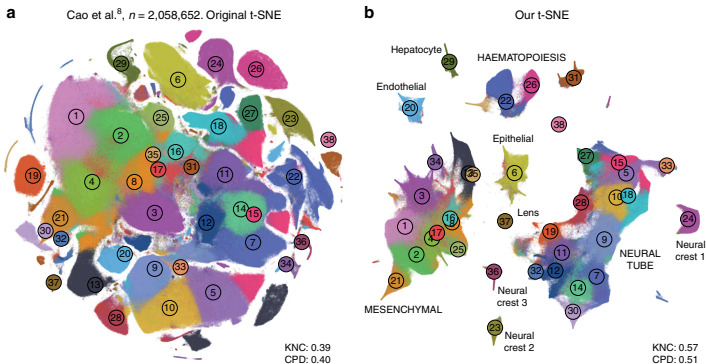


Figure from Kobak/Berens

Choice of parameters is crucial! (2)

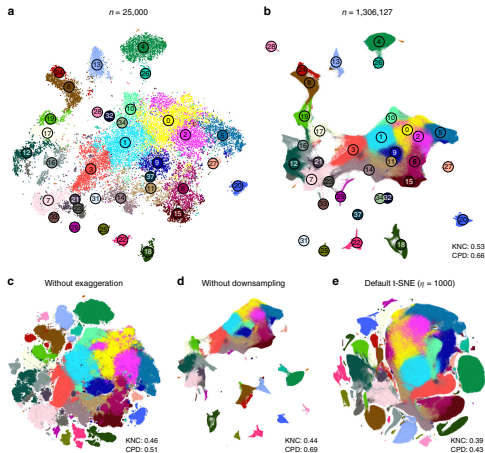
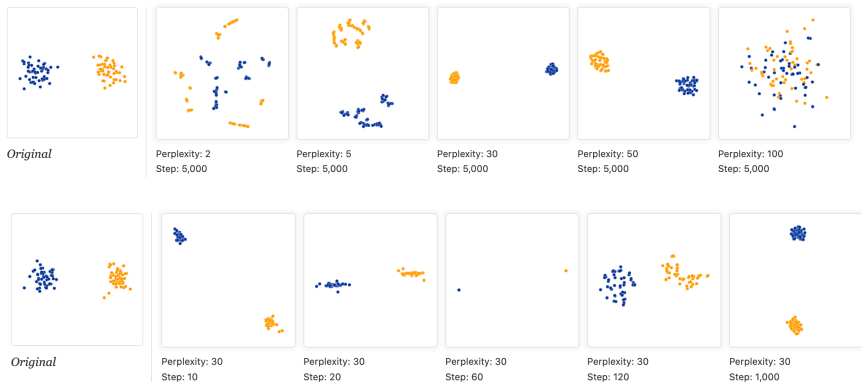


Fig. 7 10x Genomics data set. Sample size $n = 1,306,127$. Cluster assignments and cluster colours are taken from ref.²³. **a** T-SNE of a random subsample of 25,000 cells (PCA initialization, perplexity combination of 30 and 250, learning rate 25,000/12). Cluster labels for several small clusters (30, 35, 36, and 38) are not shown here and in **b** because these clusters were very dispersed in the embeddings. **b** T-SNE of the full data set. All cells were positioned on the embedding in panel **a** and this was used as initialisation. Perplexity 30, exaggeration 4, learning rate $\eta/12$. **c** The same as in **b** but without exaggeration. **d** The same as in **b** but with PCA initialisation, i.e. without using the downsampling step. **e** Default t-SNE with learning rate set to $\eta = 1000$: random initialisation, no exaggeration.

Figure from Kobak/Berens

Choice of parameters is crucial! (3)

Different values of perplexity (top row) and different values of early stopping (bottom row):



Figures from Wattenberg et al.

Choice of parameters is crucial! (4)

Some of the parameters to look at:

- ▶ Initialization: the algorithm starts with a set of points and then tries to improve the embedding by gradient descent. You can initialize with an MDS or PCA embedding, or randomly, or many other ways. The results depend a lot on the initialization:
- ▶ Perplexity
- ▶ Learning rate, early stopping, early exaggeration, momentum, ... and lots of other heuristics that are being applied

Be careful with interpretations!

Cluster sizes in a t-SNE plot mean nothing:

The size of a cluster in a t-SNE plot does not have anything to do with its “real” size. Consider the toy example below, which starts with two Gaussians with very different sizes. In all the embeddings, the clusters have the same sizes:

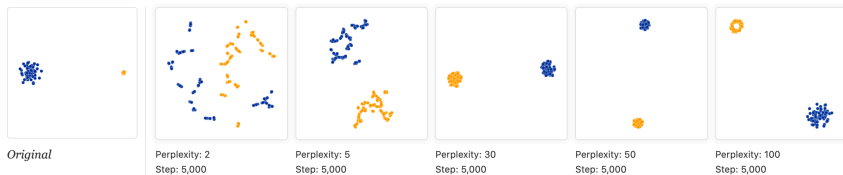
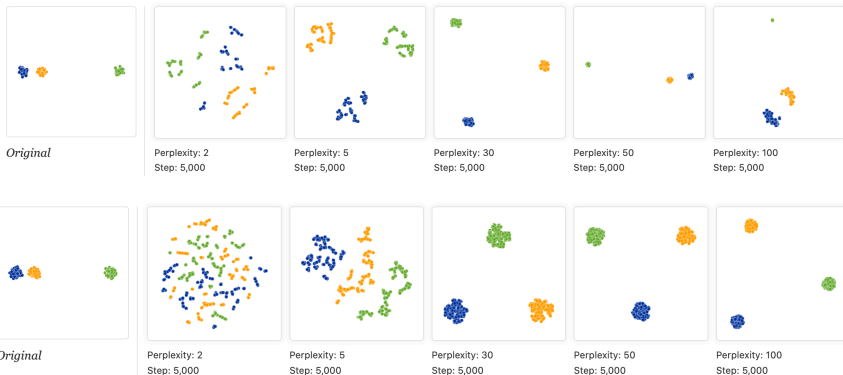


Figure from Wattenberg et al.

Be careful with interpretations! (2)

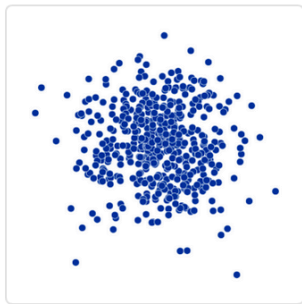
Distances between clusters might not mean anything. The relative position of clusters with respect to each other is arbitrary.



Figures from Wattenberg et al.

Be careful with interpretations! (3)

Random noise doesn't always look random



Original



Perplexity: 2

Step: 5,000

Figure from Wattenberg et al.

Be careful with interpretations! (4)

T-SNE might or might not preserve topology.

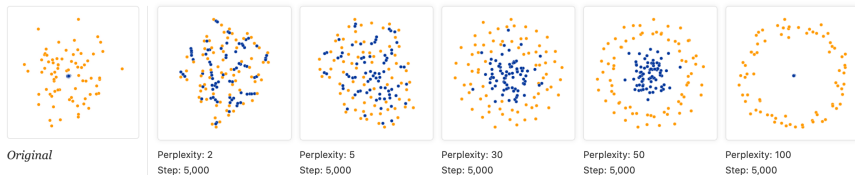


Figure from Wattenberg et al.

Summary

- ▶ t-SNE is very widely used!!!
- ▶ It is very sensible to choices of parameters. For me this means that you can play with the parameters until you see what you would like / hope / expect to see ...
- ▶ It is really difficult to interpret t-SNE plots. They look great, but see the previous slide for all the downsides. You cannot trust global geometry, sizes, distances, ...

My personal bottom line:

- ▶ Using t-SNE to explore data is fine. But **never trust t-SNE plots to reveal any truth**. When your explorative data analysis with t-SNE has suggested some new insights, then by all means go and validate your insights independently! Never trust a paper that presents a t-SNE plot as the final evidence. It can be a step on the way, but not more.

Clustering

Data clustering

Data clustering is one of the most important problems of unsupervised learning.

- ▶ Given just input data X_1, \dots, X_n
- ▶ We want to discover groups (“clusters”) in the data such that points in the same cluster are “similar” to each other and points in different clusters are “dissimilar” of each other.
- ▶ Important: a priori, we don't have any information (training labels) about these groups, and often we don't know how many groups there are (if any).

Data clustering (2)

Applications:

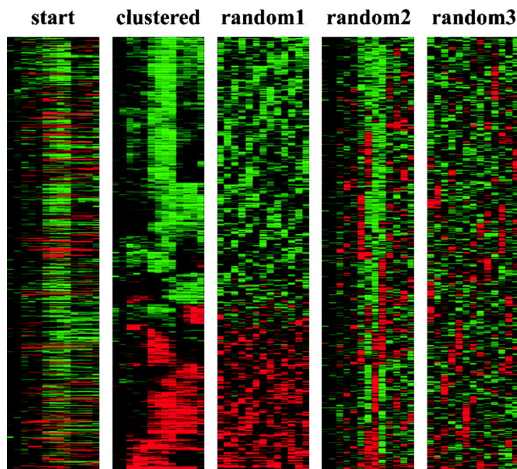
- ▶ Find “genres” of songs
- ▶ Find different “groups” of customers
- ▶ Find two different types of cancer, based on gene expression data
- ▶ Discover proteins that have a similar function
- ▶ ...

Data clustering (3)

Two main reasons to do this:

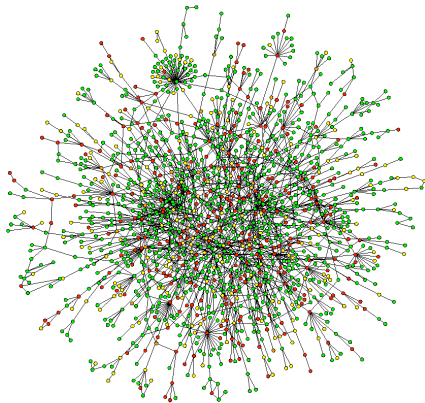
- ▶ Improve your understanding of the data! Exploratory data analysis.
- ▶ Reduce the complexity of the data. Vector quantization.
For example, instead of training on a set of 10^6 customers, use 1000 “representative” customers.
- ▶ Break your problem into subproblems and treat each cluster individually.

Example: Clustering gene expression data



M. Eisen et al., PNAS, 1998

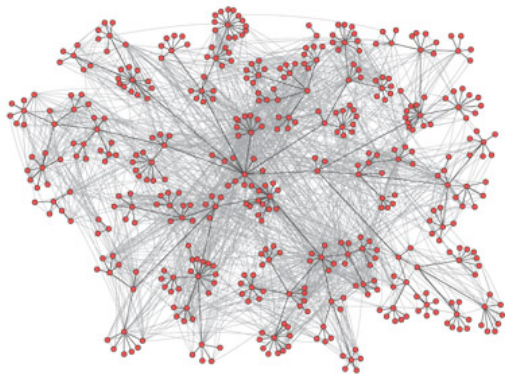
Example: Protein interaction networks



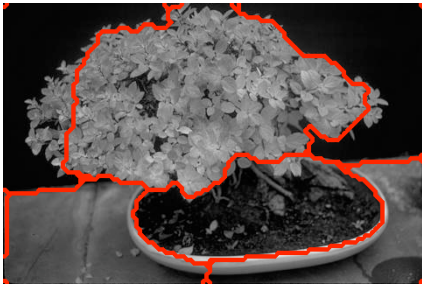
(from <http://www.math.cornell.edu/~durrett/RGD/RGD.html>)

Example: Social networks

Corporate email communication (Adamic and Adar, 2005)

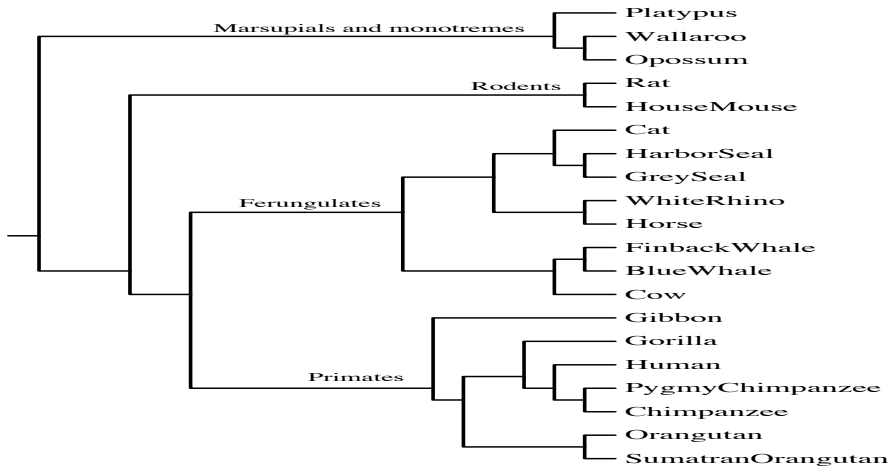


Example: Image segmentation



(from Zelnik-Manor/Perona, 2005)

Example: Genetic distances between mammals



cf. Chen/Li/Ma/Vitanyi (2004)

Most common approach

Have two goals:

- ▶ Want distances between points in the same cluster to be small
- ▶ Want distances between points in different clusters be large

Naive approach:

- ▶ Define a criterion that measures these distances and try to find the best partition with respect to this criterion: Example:

$$\text{minimize } \frac{\text{average within-cluster distances}}{\text{average between-cluster distances}}$$

Problem:

- ▶ Which objective to choose?
- ▶ Most such optimization problems are NP hard (combinatorial optimization).

(*) K-means and kernel k-means

Literature:

Tibshirani/Hastie/Friedmann

Standard k -means algorithm

k -means objective

- ▶ Assume we are given data points $X_1, \dots, X_n \in \mathbb{R}^d$
- ▶ Assume we want to separate it into K groups.
- ▶ We want to construct K class representatives (class means) m_1, \dots, m_K that represent the groups.
- ▶ Consider the following objective function:

$$\min_{\{m_1, \dots, m_K \in \mathbb{R}^d\}} \sum_{k=1}^K \sum_{i \in C_k} \|X_i - m_k\|^2$$

That is, we want to find the centers such that the sum of squared distances of data points to the closest centers are minimized.

k -means objective (2)



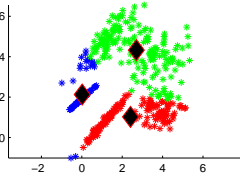
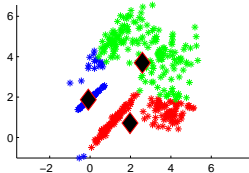
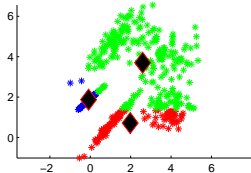
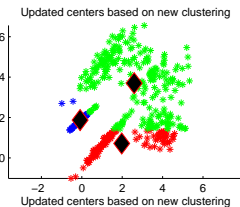
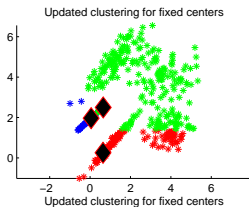
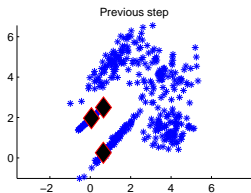
$x = \text{cluster centers}$

Lloyd's algorithm (k -means algorithm)

The following heuristic is typically used to find a local optimum of the k -means objective function:

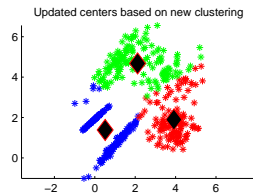
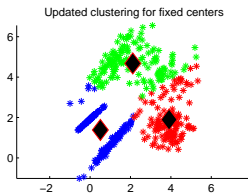
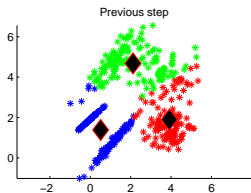
- ▶ Start with randomly chosen centers.
- ▶ Repeat the following two steps until convergence:
 - ▶ Assign all points to the closest cluster center.
 - ▶ Define the new centers as the mean vectors of the current clusters.

Lloyd's algorithm (k -means algorithm) (2)



...

Lloyd's algorithm (k -means algorithm) (3)



Lloyd's algorithm (k -means algorithm) (4)

The formal k -means algorithm:

- 1 **Input:** Data points $X_1, \dots, X_n \in \mathbb{R}^d$, number K of clusters to construct.
- 2 Randomly initialize the centers $m_1^{(0)}, \dots, m_K^{(0)}$.
- 3 **while** not converged
- 4 Assign each data point to the closest cluster center, that is define the clusters $C_1^{(i+1)}, \dots, C_K^{(i+1)}$ by

$$X_s \in C_k^{(i+1)} \iff \|X_s - m_k^{(i)}\|^2 \leq \|X_s - m_l^{(i)}\|^2, l = 1, \dots, K$$

- 5 Compute the new cluster centers by

$$m_k^{(i+1)} = \frac{1}{|C_k^{(i+1)}|} \sum_{s \in C_k^{(i+1)}} X_s$$

- 6 **Output:** Clusters C_1, \dots, C_K

Lloyd's algorithm (k -means algorithm) (5)

matlab demo: `demo_kmeans()`

K -means algorithm — Termination

Proposition 22 (Termination)

Given a finite set of n point in \mathbb{R}^d . Then the k -means algorithm terminates after a finite number of iterations.

Proof sketch.

- ▶ In each iteration of the while loop, the objective function decreases.
- ▶ There are only finitely many partitions we can inspect.
- ▶ So the algorithm has to terminate.



K -means algorithm — Solutions can be arbitrarily bad

Proposition 23 (Bad solution possible)

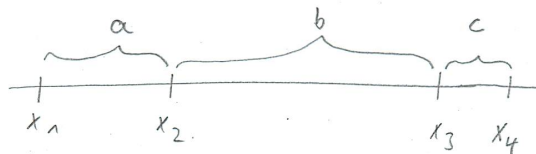
The algorithm ends in a local optimum which can be an arbitrary factor away from the global solution.

Proof.

- ▶ We give an example with four points in \mathbb{R} , see figure on the next slides.
- ▶ By adjusting the parameters a and b and c we can achieve an arbitrarily bad ratio of global and local solution.

K -means algorithm — Solutions can be arbitrarily bad (2)

Data set: 4 points on the real line:



Different solutions depending on the initialization:



Initialization 1



Solution 1, value $c^2/2$



Initialization 2



Solution 2, value $a^2/2$

If $b > \max\{a, c\}$ and $a > c$, then the local optimum Solution 2 is worse by a factor of $(a/c)^2$ than the global Solution 1.

K -means algorithm — Initialization

Methods to select initial centers:

- ▶ Most common: randomly choose some data points as starting centers.
- ▶ Much better: Farthest first heuristic:

- 1 $S = \emptyset$ # S set of centers
- 2 Pick x uniformly at random from the data points
 $S = \{x\}$
- 3 **while** $|S| < k$
- 4 **for all** $x \in \mathcal{X} \setminus S$
- 5 Compute $D(x) := \min_{s \in S} \|x - s\|^2$.
- 6 Select the next center y with probability proportional to $D(x)$ among the remaining data points and insert it in S .

K -means algorithm — Initialization (2)

The k -means algorithm with this heuristic is called `kmeans++` and satisfies nice approximation guarantees.

- ▶ Initialize the centers using the solution of an even simpler clustering algorithm.
- ▶ Ideally have prior knowledge, for example that certain points are in different clusters.

K -means algorithm — Heuristics for practice

As it is the standard procedure for highly non-convex optimization problems, in practice **we restart the algorithm many times with different initializations**. Then we use the best of all these runs as our final result.

K -means algorithm — Heuristics for practice (2)

Common problem:

- ▶ In the course of the algorithm it can happen that a center “looses” all its data points (no point is assigned to the center any more).
- ▶ In this case, one either restarts the whole algorithm, or randomly replaces the empty center by one of the data points.

K -means algorithm — Heuristics for practice (3)

Local search heuristics to improve the result once the algorithm has terminated:

- ▶ Restart many times with different initializations.
- ▶ Swap individual points between clusters.
- ▶ Remove a cluster center, and introduce a completely new center instead.
- ▶ Merge clusters, and additionally introduce a completely new cluster center.
- ▶ Split a cluster in two pieces (preferably, one that has a very bad objective function). Then reduce the number of clusters again, for example by randomly removing one.

k -means minimizes within-cluster distances

Another way to understand the k -means objective:

Proposition 24 (k -means and within-cluster distances)

The following two optimization problems are equivalent:

1. Find a discrete partition of the data set such that the within-cluster-distances are minimized:

$$\min_{\{C_1, \dots, C_K\}} \sum_{k=1}^K \frac{1}{|C_k|^2} \sum_{i \in C_k, j \in C_k} \|X_i - X_j\|^2$$

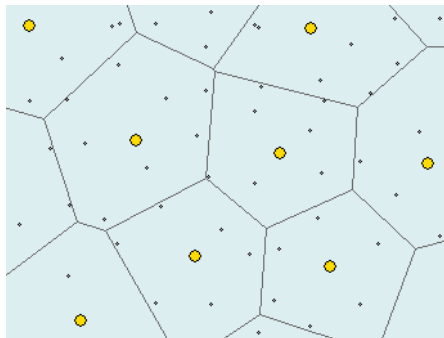
2. Find cluster centers such that the distances of the data points to these centers are minimized:

$$\min_{m_1, \dots, m_K \in \mathbb{R}^d} \sum_{k=1}^K \sum_{i \in C_k} \|X_i - m_k\|^2$$

Proof. Elementary, but a bit lengthy, we skip it.

k -means leads to Voronoi partitions

Observe that the partition induced by the k -means objective corresponds to a Voronoi partition of the space:

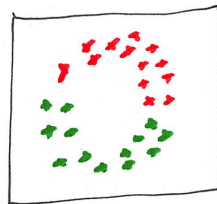
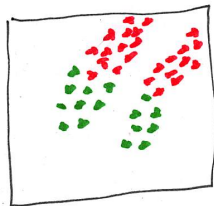
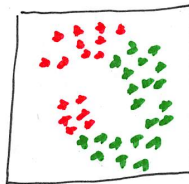
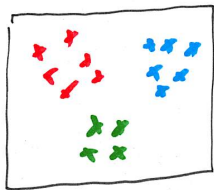


k -means leads to Voronoi partitions (2)

This has two important consequences:

- ▶ all cluster boundaries are linear (WHY MIGHT THIS BE INTERESTING?)
- ▶ The k -means algorithm always constructs convex clusters! This gives intuition about when it works and when it doesn't work:

k -means leads to Voronoi partitions (3)



K -means — computational complexity

- Finding the global solution of the k -means optimization problem is **NP hard** (both if k is fixed or variable, and both if the dimension is fixed or variable).

This is curious because one can prove that there only exist polynomially many Voronoi partitions of any given data set. The difficulty is that we cannot construct any enumeration to search through them.

See the following paper and references therein:

Mahajan, Meena and Nimbhorkar, Prajakta and Varadarajan, Kasturi: The planar k -means problem is NP-hard. WALCOM: Algorithms and Computation, 2009.

K -means — computational complexity (2)

- ▶ On the other hand, optimizing the k -means objective has **polynomial smoothed complexity**.
Arthur, David and Manthey, Bodo and Röglin: k -Means has polynomial smoothed complexity. FOCS 2009.
- ▶ With careful seeding, one can achieve **constant-factor approximations**:
 - ▶ Consider the random farthest first rule for initialization (kmeans with this initialization is called kmeans++).
 - ▶ Then, the expected objective value is at most a factor $O(\log k)$ worse than the optimal solution.
 - ▶ Reference:
Arthur, D., Vassilvitskii, S.: k -means++: the advantages of careful seeding. In: Symposium on Discrete Algorithms (SODA), 2007.

More variants of K -means

- ▶ K -median: here the centers are always data points. Can be used if we only have distances, but no coordinates of data points.
- ▶ weighted K -means: introduce weights for the individual data points
- ▶ kernel- K -means: the kernelized version of K -means (note that all boundaries between clusters are linear).
I. S. Dhillon, Y. Guan, and B. Kulis, Kernel k -means, spectral clustering and normalized cuts. KDD, 2004.
- ▶ soft K -means: no hard assignments, but “soft” assignments (often interpreted as “probability” of belonging to a certain cluster)
- ▶ Note: K -means is a simplified version of the EM-algorithm, which fits a Gaussian mixture model to the data.

Summary K -means

- ▶ Represent clusters by cluster centers
- ▶ Highly non-convex NP hard optimization problem
- ▶ Heuristic: Lloyd's k -means algorithm
- ▶ Very easy to implement, hence very widely used.
- ▶ In my opinion: k -means works well for vector quantization (if you want to find a large number of clusters, say 100 or so). It does not work so well for small k , here you should consider spectral clustering.

(*) Linkage algorithms for hierarchical clustering

cf. Chen/Li/Ma/Vitanyi (2004)

Simple idea

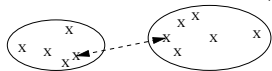
Agglomerative (bottom-up) strategy:

- ▶ Start: each point is its own cluster
- ▶ Then check which points are closest and “merge” them to form a new cluster
- ▶ Continue, always merge two “closest” clusters until we are left with one cluster only

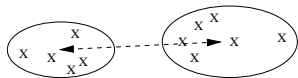
Simple idea (2)

To define which clusters are “closest”:

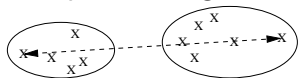
Single linkage: $\text{dist}(C, C') = \min_{x \in C, y \in C'} d(x, y)$



Average linkage: $\text{dist}(C, C') = \frac{\sum_{x \in C, y \in C'} d(x, y)}{|C| \cdot |C'|}$



Complete linkage: $\text{dist}(C, C') = \max_{x \in C, y \in C'} d(x, y)$



Linkage algorithms – basic form

Input:

- Distance matrix D between data points (size $n \times n$)
- function *dist* to compute a distance between clusters (usually takes D as input)

Initialization: Clustering $\mathcal{C}^{(0)} = \{C_1^{(0)}, \dots, C_n^{(0)}\}$ with $C_i^{(0)} = \{i\}$.

While the current number of clusters is > 1 :

- find the two clusters that have the smallest distance to each other
- merge them to one cluster

Output: Resulting dendrogram

Examples

... show matlab demos ...

```
demo_linkage_clustering_by_foot()
```

```
demo_linkage_clustering_comparison()
```

Linkage algorithms tend to be problematic

Observations from practice:

- ▶ Linkage algorithms are very vulnerable to outliers
- ▶ One cannot “undo” a bad link

Theoretical considerations:

- ▶ Linkage algorithms attempt to estimate the density tree
- ▶ Even though this can be done in a statistically consistent way, estimating densities in high dimensions is extremely problematic and usually does not work in practice.

History and References

- ▶ The original article: S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 2:241 - 254, 1967.
- ▶ A complete book on the topic: N. Jardine and R. Sibson. *Mathematical taxonomy*. Wiley, London, 1971.
- ▶ Nice, more up-to-date overview with application in biology: J. Kim and T. Warnow. Tutorial on phylogenetic tree estimation. ISMB 1999.

Linkage algorithms — summary

- ▶ Attempt to estimate the whole cluster tree
- ▶ There exist many more ways of generating different trees from a given distance matrix.
- ▶ Advantage of tree-based algorithms: do not need to decide on “the correct” number of clusters, get more information than just a flat clustering
- ▶ However, one should be very careful about the results because they are very unstable, prone to outliers and statistically unreliable.

Graph-based machine learning algorithms: introduction

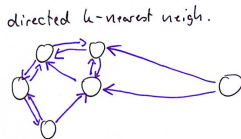
Neighborhood graphs

Given the similarity or distance scores between our objects, we want to build a graph based on it.

- ▶ Vertices = objects
- ▶ Edges between objects in the same “neighborhood”

Different variants:

- ▶ **directed k -nearest neighbor graph**: connect x_i by a directed edge to its k nearest neighbors (or to the k points with the largest similarity) .

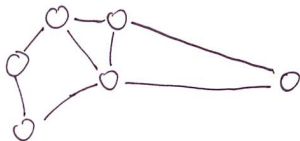


Neighborhood graphs (2)

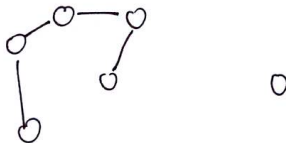
Note that this graph is not symmetric. Many algorithms need undirected graphs (in particular, spectral methods). To make it undirected:

- **Standard k -nearest neighbor graph:** put an edge between x_i and x_j if x_i is among the k nearest neighbors of x_j OR vice versa.
- **Mutual k -nearest neighbor graph:** put an edge between x_i and x_j if x_i is among the k nearest neighbors of x_j AND vice versa.

standard undirected



mutual undirected



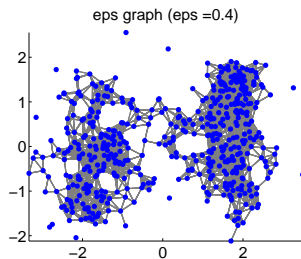
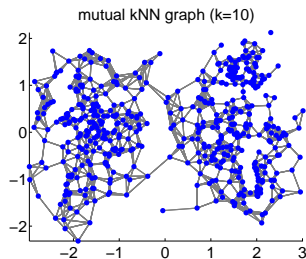
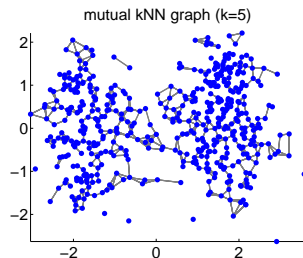
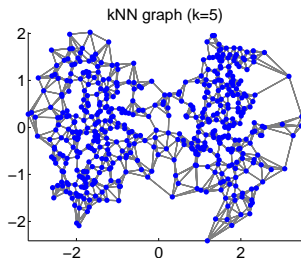
Neighborhood graphs (3)

Alternatively, we can use the ε -graph:

- Connect each point to all other points that have distance smaller than ε (or similarity larger than some threshold e)

Note: all these neighborhood graphs can be built based on similarities or based on distances.

Neighborhood graphs (4)



Neighborhood graphs (5)

Edge weights:

- ▶ A priori, all the graphs above are unweighted.
- ▶ On kNN graphs, it often makes sense to use **similarities as edge weights**.

(Reason: edges have very diverse “lengths”, and we want to tell this to the algorithm; e.g., spectral clustering is allowed to cut long edges more easily than short edges)

- ▶ **Never use distances as weights!** (this destroys the “logic” behind a neighborhood graph: no edge means “far away”, and no edge is the same as edge weight 0 ...)
- ▶ For ε -graphs, edge weights do not make so much sense because all edges are more or less “equally long”. The edge weights then do not carry much extra information.

Neighborhood graphs (6)

Matlab demo on similarity graphs:

```
/Users/u/e/matlab_u/e/not_in_path/demos_for_teaching/practical_session_summerschool_tuebingen07/
```

GraphDemos

Neighborhood graphs (7)

Why are we interested in similarity graphs?

- ▶ Sparse representation of the similarity structure
- ▶ Graphs are well-known objects, lots of algorithms to deal with them.
- ▶ Similarity graph encodes local structure, goal of machine learning (unsupervised learning) is to make statements about its global structure.
- ▶ There exist many algorithms for machine learning on graphs:
 - ▶ Clustering: Spectral clustering (see later)
 - ▶ Dimensionality reduction: Isomap, Laplacian eigenmaps, Maximum Variance Unfolding
 - ▶ Semi-supervised learning: label propagation (not treated in the lecture)

A glimpse on spectral graph theory

Literature:

- ▶ U. Luxburg. Tutorial on Spectral Clustering, Statistics and Computing, 2007.
- ▶ F. Chung: Spectral Graph Theory (Chapters 1 and 2).
- ▶ D. Spielman: Spectral Graph Theory, 2011.

What is it about?

General idea:

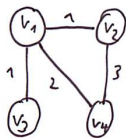
- ▶ many properties of graphs can be described by properties of the adjacency matrix and related matrices (“graph Laplacians”).
- ▶ In particular, the eigenvalues and eigenvectors can say a lot about the “geometry” of the graph.

Unnormalized Laplacians

Unnormalized Graph Laplacians: Definition

Consider an undirected graph with non-negative edge weights w_{ij} .
Notation:

- ▶ W := the weight matrix of the graph
- ▶ $D := \text{diag}(d_1, \dots, d_n)$ the **degree matrix** of the graph
- ▶ $L := D - W$ the **unnormalized graph Laplacian matrix**



$$W = \begin{pmatrix} 0 & 1 & 1 & 2 & 0 \\ 1 & 0 & 0 & 3 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 & 0 \end{pmatrix}$$

$$D = \begin{pmatrix} 4 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$L = \begin{pmatrix} 4 & -1 & -1 & -2 & 0 \\ -1 & 4 & 0 & -3 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ -2 & -3 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Unnormalized Laplacians: Key property

Proposition 25 (Key property)

Let G be an undirected graph. Then for all $f \in \mathbb{R}^n$,

$$f^t L f = \frac{1}{2} \sum_{i,j=1}^n w_{ij} (f_i - f_j)^2.$$

Proof. Simply do the calculus:

Unnormalized Laplacians: Key property (2)

$$\begin{aligned}f^t L f &= f^t D f - f^t W f \\&= \sum_i d_i f_i^2 - \sum_{i,j} f_i f_j w_{ij} \\&= \frac{1}{2} \left(\sum_i \left(\sum_j w_{ij} \right) f_i^2 - 2 \sum_{ij} f_i f_j w_{ij} + \sum_j \left(\sum_i w_{ij} \right) f_j^2 \right) \\&= \frac{1}{2} \sum_{ij} w_{ij} (f_i - f_j)^2\end{aligned}$$

□

Why is it called “Laplacian”?

Where does the name “graph Laplacian” come from?

$$f^t L f = \frac{1}{2} \sum w_{ij} (f_i - f_j)^2$$

Interpret $w_{ij} \sim 1/d(X_i, X_j)^2$

$$f^t L f = \frac{1}{2} \sum ((f_i - f_j)/d_{ij})^2$$

looks like a discrete version of the standard Laplace operator

$$\langle f, \Delta f \rangle = \int |\nabla f|^2 dx$$

Hence the graph Laplacian measures the variation of the function f along the graph: $f^t L f$ is low if points that are close in the graph have similar values f_i .

Unnormalized Laplacians: Spectral properties

Proposition 26 (Simple spectral properties)

For an undirected graph with non-negative edge weights, the graph Laplacian has the following properties:

- ▶ L is symmetric and positive semi-definite.
- ▶ Smallest eigenvalue of L is 0, corresponding eigenvector is $\mathbb{1} := (1, \dots, 1)^t$.
- ▶ Thus eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$.

Unnormalized Laplacians: Spectral properties (2)

Proof.

Symmetry: W is symmetric (graph is undirected), D is symmetric, so L is symmetric.

Positive Semi-Definite: by key proposition:

$$f^t L f = \frac{1}{2} \sum_{ij} w_{ij} (f_i - f_j)^2 \geq 0$$

Smallest Eigenvalue/vector: It is indeed an eigenvector because $L\mathbf{1} = D\mathbf{1} - W\mathbf{1} = 0$

It is the smallest because all eigs are ≥ 0 .



Unnormalized Laplacians and connected components

Proposition 27 (Relation between spectrum and clusters)

Consider an undirected graph with non-negative edge weights.

- ▶ Then the (geometric) multiplicity of eigenvalue 0 is equal to the number k of connected components A_1, \dots, A_k of the graph.
- ▶ The eigenspace of eigenvalue 0 is spanned by the characteristic functions $\mathbb{1}_{A_1}, \dots, \mathbb{1}_{A_k}$ of those components (where $\mathbb{1}_{A_i}(j) = 1$ if $v_j \in A_i$ and $\mathbb{1}_{A_i}(j) = 0$ otherwise).

Unnormalized Laplacians and connected components (2)

Proof, case $k=1$.

- ▶ Assume that the graph is connected.
- ▶ Let f be an eigenvector with eigenvalue 0.
- ▶ Want to show: f is a constant vector.

Here is the reasoning:

- ▶ By definition: $Lf = 0$.
- ▶ Exploiting this and the key proposition:

$$0 = f^t Lf = \sum_{ij} w_{ij} (f_i - f_j)^2$$

- ▶ The right hand side can only be 0 if all summands are 0.

Unnormalized Laplacians and connected components (3)

- ▶ Hence, for all pairs (i, j) :
 - ▶ either $w_{ij} = 0$ (that is, v_i and v_j are not connected by an edge in the graph),
 - ▶ or $f_i = f_j$.

Consequently: if v_i and v_j are connected in the graph, then $f_i = f_j$.

In particular, f is constant on the whole connected component.

Unnormalized Laplacians and connected components (4)

Proof, case $k > 1$.

- If the graph consists of k disconnected components, both the adjacency matrix and the graph Laplacian are block diagonal. In particular, each little block is the graph Laplacian of the corresponding connected component.

$$L = \begin{pmatrix} L_1 & & \\ & L_2 & \\ & & \ddots \\ & & & L_k \end{pmatrix}$$

Unnormalized Laplacians and connected components (5)

- For each block (= each connected component), by the case $k = 1$ we know that there is exactly one eigenvector for eigenvalue 0, and it is constant:

$$v_1(L_1) = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \quad v_1(L_2) = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \dots$$

Unnormalized Laplacians and connected components (6)

- For the matrix L we then know that there are k eigenvalues 0, each one coming from one of the blocks. Padding the eigenvectors with zeros leads to the cluster indicator vectors:

$$v_1(L) = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \quad v_2(L) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{pmatrix} \quad \dots$$



Unnormalized Laplacians and connected components (7)

QUESTION:

Consider a graph with k connected components:

WHY IS THERE NO CONTRADICTION BETWEEN
PROPOSITION 27 (SECOND STATEMENT) AND
PROPOSITION 26???

Normalized Laplacians

Normalized graph Laplacian

For various reasons (see below) it is better to normalize the graph Laplacian matrix.

Two versions:

- ▶ The “symmetric” normalized graph Laplacian

$$L_{sym} = D^{-1/2} L D^{-1/2}$$

(where the square root of the diagonal matrix D can be computed entry-wise).

- ▶ The “random walk graph Laplacian”

$$L_{rw} = D^{-1} L$$

We will now see that both normalized Laplacians are closely related, and have similar properties as the unnormalized Laplacian.

Normalized Laplacians: First properties

Proposition 28 (Adapted key property)

For every $f \in \mathbb{R}^n$ we have

$$f^t L_{\text{sym}} f = \frac{1}{2} \sum_{i,j=1}^n w_{ij} \left(\frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2.$$

Proof. Similar to the unnormalized case.



Normalized Laplacians: First properties (2)

Proposition 29 (Simple spectral properties)

Consider an undirected graph with non-negative edge weights.
Then:

1. λ is an eigenvalue of L_{rw} with eigenvector u
 $\iff \lambda$ is an eigenvalue of L_{sym} with eigenvector $w = D^{1/2}u$.
2. λ is an eigenvalue of L_{rw} with eigenvector u
 $\iff \lambda$ and u solve the generalized eigenproblem $Lu = \lambda Du$.
3. 0 is an eigenvalue of L_{rw} with the constant one vector $\mathbb{1}$ as eigenvector. 0 is an eigenvalue of L_{sym} with eigenvector $D^{1/2}\mathbb{1}$.
4. L_{sym} and L_{rw} are positive semi-definite and have n non-negative real-valued eigenvalues $0 = \lambda_1 \leq \dots \leq \lambda_n$.

Normalized Laplacians: First properties (3)

Proof.

Part (1): multiply the eigenvalue equation $L_{\text{sym}}w = \lambda w$ with $D^{-1/2}$ from the left and substitute $u = D^{-1/2}w$.

Part (2): multiply $L_{\text{rw}}u = \lambda u$ with D from the left.

Part (3): Just plug it in the corresponding eigenvalue equations.

Part (4): The statement about L_{sym} follows from the adapted key property, and then the statement about L_{rw} follows from (2). ☺

Normalized Laplacians and connected components

Proposition 30 (Relation between spectrum and clusters)

Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of both L_{rw} and L_{sym} equals the number of connected components A_1, \dots, A_k in the graph. For L_{rw} , the eigenspace of 0 is spanned by the indicator vectors $\mathbb{1}_{A_i}$ of those components. For L_{sym} , the eigenspace of 0 is spanned by the vectors $D^{1/2}\mathbb{1}_{A_i}$.

Proof.

Analogous to the one for the unnormalized case. □

Cheeger constant

Cheeger constant

Let G be an undirected graph with non-negative edge weights w_{ij} , $S \subset V$ be a subset of vertices, $\bar{S} := V \setminus S$ its complement. Define:

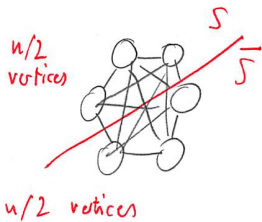
- ▶ Volume of the set: $\text{vol}(S) := \sum_{s \in S} d(s)$
- ▶ Cut value: $\text{cut}(S, \bar{S}) := \sum_{i \in S, j \in \bar{S}} w_{ij}$
- ▶ Cheeger constant:

$$h_G(S) := \frac{\text{cut}(S, \bar{S})}{\min\{\text{vol}(S), \text{vol}(\bar{S})\}}$$
$$h_G := \min_{S \subset V} h_G(S)$$

Cheeger constant (2)

Example: a clique (=fully connected graph, including self-loops) with n vertices has $h_G = \Theta(1)$:

- S that contains $n/2$ vertices:



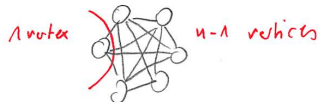
$$\text{cut}(S, \bar{S}) = \frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$$

$$vol(S) = vol(\bar{S}) = \frac{n}{2} \cdot n = \frac{n^2}{2}$$

$$\Rightarrow h_G(S) = \frac{1}{2} = \Theta(1)$$

Cheeger constant (3)

- S that contains 1 vertex:



$$\text{cut}(S, \bar{S}) = n-1$$

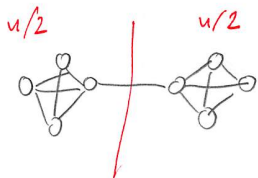
$$\text{vol}(S) = 1 \cdot n, \quad \text{vol}(\bar{S}) = (n-1) \cdot n$$

$$\Rightarrow h_G(S) = \frac{n-1}{n} = \Theta(1)$$

- Similarly for other sets S .

Cheeger constant (4)

Example: two cliques with $n/2$ vertices each, connected by a single edge. Results in $h_G = \mathcal{O}(1/n^2)$



$$\text{cut}(S, \bar{S}) = 1$$

$$\text{vol}(S) = \text{vol}(\bar{S}) = \frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$$

$$\Rightarrow h_G(S) = \frac{1}{n^2/4} = \Theta\left(\frac{1}{n^2}\right)$$

Cheeger constant (5)

Intuition:

small Cheeger cuts are achieved for cuts that split the graph into reasonably big, tightly connected subgraphs (so that denominator (Nenner) is large) which are well clustered (numerator (Zähler) small).

Relation between Cheeger constant and λ_2

Theorem 31 (Cheeger inequality for graphs)

Consider a connected, undirected, unweighted graph. Let λ_2 be the second-smallest eigenvalue of L_{sym} . Then

$$\frac{\lambda_2}{2} \leq h_G \leq \sqrt{2\lambda_2}$$

Intuition:

- ▶ The Cheeger constant describes the cluster properties of a graph.
- ▶ The Cheeger constant is controlled by the second eigenvalue.

Relation between Cheeger constant and λ_2 (2)

Proof (taken from Chung: Spectral Graph Theory Chapter 2):

Proof of the upper bound on the Cheeger constant:

Theorem: Let λ_2 be the second smallest eigenvector of L_{sym} and h_G the Cheeger constant of the graph. Then

$$2 \cdot h_G \geq \lambda_2.$$

Relation between Cheeger constant and λ_2 (3)

Proof: By Rayleigh's principle,

$$\lambda_2 = \inf_{v \perp v_1} \frac{v^t L_{\text{sym}} v}{\|v\|^2} = \inf_{v \perp D^{1/2} \underline{1}} \frac{v^t (D^{-1/2} L D^{-1/2}) v}{\|v\|^2}$$

Substitute $w := D^{1/2} v$ to get

$$\lambda_2 = \inf_{w \perp D \underline{1}} \frac{w^t L w}{\|D^{1/2} w\|^2} = \inf_{w \perp D \underline{1}} \frac{\sum_{i \sim j} (\omega_i - \omega_j)^2}{\sum_{i,j} d_j w_j^2}$$

Relation between Cheeger constant and λ_2 (4)

To construct an upper bound on λ_2 , we now consider a particular vector w^* :

$$\text{Let } S^* := \underset{S \subset V}{\operatorname{argmin}} \frac{\operatorname{cut}(S, \bar{S})}{\min\{\operatorname{vol}(S), \operatorname{vol}(\bar{S})\}} \quad \text{the}$$

set that minimizes the Cheeger constant. Define the vector u by

$$u_i := \begin{cases} 1/\operatorname{vol}(S^*) & \text{if } i \in S^* \\ -1/\operatorname{vol}(\bar{S}^*) & \text{if } i \in \bar{S}^* \end{cases}.$$

Relation between Cheeger constant and λ_2 (5)

Note that $u \perp \mathbb{1}$ because

$$\langle u, d \rangle = \underbrace{\sum_{i \in S^*} \frac{1}{\text{vol}(S^*)} d_i}_{=1} - \underbrace{\sum_{i \notin S^*} \frac{1}{\text{vol}(\bar{S}^*)} d_i}_{=1} = 0$$

Relation between Cheeger constant and λ_2 (6)

Now we get

$$\lambda_2 \leq \frac{\sum_{i,j} (u_i - u_j)^2}{\sum_i d_i u_i^2} \quad \leftarrow \text{by def. of } u$$

edges from S^* to \bar{S}^*

$$= \frac{\sum_{i \in S^*, j \in \bar{S}^*, i \sim j} \left(\frac{1}{\text{vol}(S^*)} - \frac{1}{\text{vol}(\bar{S}^*)} \right)^2}{\underbrace{\sum_{i \in S^*} \left(\frac{1}{\text{vol}(S^*)} \right)^2 d_i}_{\frac{1}{\text{vol}(S^*)}} + \underbrace{\sum_{i \in \bar{S}^*} \left(\frac{1}{\text{vol}(\bar{S}^*)} \right)^2 d_i}_{\frac{1}{\text{vol}(\bar{S}^*)}}}$$

$$= \text{cut}(S^*, \bar{S}^*) \left(\frac{1}{\text{vol}(S^*)} + \frac{1}{\text{vol}(\bar{S}^*)} \right)$$

$$\leq \text{cut}(S^*, \bar{S}^*) \cdot 2 \max \left\{ \frac{1}{\text{vol}(S^*)}, \frac{1}{\text{vol}(\bar{S}^*)} \right\}$$

$$= 2 \frac{\text{cut}(S^*, \bar{S}^*)}{\min \{ \text{vol}(S^*), \text{vol}(\bar{S}^*) \}} = 2 h_n.$$

Relation between Cheeger constant and λ_2 (7)

Proof of the lower bound:

Follows similar principles, but is a bit longer, see the Chung book if you are interested.



Spectral clustering

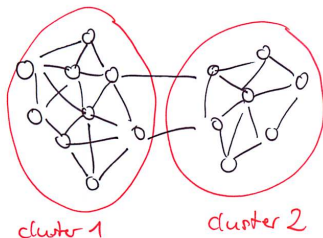
Literature:

- ▶ U. Luxburg. Tutorial on Spectral Clustering, Statistics and Computing, 2007.
- ▶ The more recent editions of Tibshirani/Hastie/Friedman also contain a chapter on it.

Clustering in graphs

General problem:

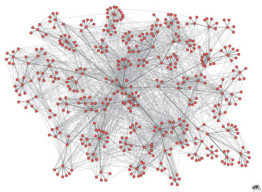
- ▶ Given a graph
- ▶ Want to find “clusters” in the graph:
 - ▶ many connections inside the cluster
 - ▶ few connections between different clusters



Clustering in graphs (2)

Examples:

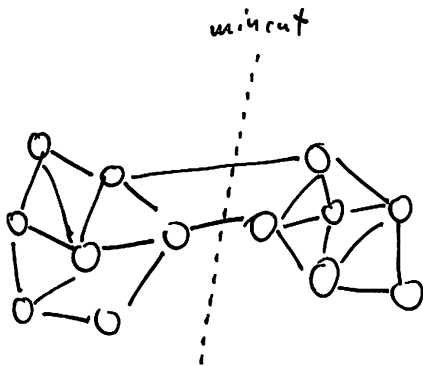
- ▶ Find “communities” in a social network (e.g., to analyze communication patterns in a company; to place targeted ads in facebook)
- ▶ Find groups of jointly acting proteins in a protein-interaction network
- ▶ Find groups of similar films (\leadsto “genres”)
- ▶ Find subgroups of diseases (for more specific medical treatment)



Clustering in graphs (3)

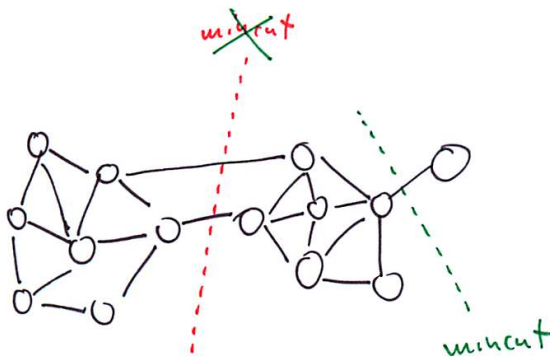
First idea:

- ▶ “Few connections between clusters” \approx “small cut”.
- ▶ Thus clustering \approx find a mincut in the graph.



Clustering in graphs (4)

Problem: outliers



Clustering in graphs (5)

Better idea: find two sets such that

- ▶ the cut between the sets is small
- ▶ each of the clusters is “reasonably large”

Some background on complexity of cut problems:

- ▶ Finding any mincut (without extra constraints) is easy and can be done in polynomial time.
- ▶ Finding the balanced mincut is NP hard
- ▶ Can we do something in between?

RatioCut criterion

Idea:

- ▶ want to define an objective function that measures the quality of a “nearly balanced cut”:
 - ▶ the smaller the cut value, the smaller the objective function
 - ▶ the more balanced the cut, the smaller the objective function

Measuring the balancedness of a cut:

- ▶ Consider a partition $V = A \cup B$.
- ▶ Define $|A|$:= number of vertices in A
- ▶ Introduce the **balancing term** $1/|A| + 1/|B|$.
- ▶ Observe: The balancing term is small when A and B have (approximately) the same number of vertices:

RatioCut criterion (2)

- ▶ Example: n vertices in total.
 - ▶ Case $|A| = n/2$, $|B| = n/2$. Then $1/|A| + 1/|B| = 4/n = O(1/n)$.
 - ▶ Case $|A| = 1$, $|B| = n - 1$. Then $1/|A| + 1/|B| = 1 + 1/(n - 1) = O(1)$.
- ▶ In general: Under the constraint that $|A| + |B| = n$, the term $\frac{1}{|A|} + \frac{1}{|B|}$ is minimal if $|A| = |B|$.

Formally, this can be seen by taking the derivative of the function $f(a) = 1/a + 1/(n - a)$ with respect to a and setting it to 0.

RatioCut criterion (3)

Combining cut and balancing: Define:

$$\text{cut}(A, B) := \sum_{i \in A, j \in B} w_{ij}$$

$$\text{RatioCut}(A, B) = \text{cut}(A, B) \left(\frac{1}{|A|} + \frac{1}{|B|} \right)$$

RatioCut gets smaller if

- ▶ the cut is smaller
- ▶ the clusters are more balanced

This is what we wanted to achieve.

RatioCut criterion (4)

Target is now: find the cut with the minimal RatioCut value in a graph.

Bad news:

finding the global minimum of RatioCut is NP hard ☹

Good news:

But there exists an algorithm that finds very good solutions in most of the cases: spectral clustering ☺

Unnormalized spectral clustering

Goal: minimize RatioCut

Consider the following problem:

Given an undirected graph G with non-negative edge weights. What is the minimal RatioCut in the graph?

On the following slides we want to show how we can use spectral graph theory to achieve what we want.

Relaxing balanced cut

Consider a graph with an even number n of vertices. For $A \subset V$, denote $\bar{A} = V \setminus A$. We want to solve the following problem:

$$\min_{A \subset V} \text{cut}(A, \bar{A}) \quad \text{subject to} \quad |A| = |\bar{A}| \quad (*)$$

We want to rewrite the problem in a more convenient way.

Introduce $f = (f_1, \dots, f_n)^t \in \mathbb{R}^n$ with

$$f_i = \begin{cases} +1 & \text{if } i \in A \\ -1 & \text{if } i \notin A. \end{cases}$$

Now observe:

$$\text{cut}(A, \bar{A}) = \sum_{i \in A, j \in \bar{A}} w_{ij} = \frac{1}{4} \sum_{i,j=1}^n w_{ij} (f_i - f_j)^2 = \frac{1}{2} f^t L f$$

Relaxing balanced cut (2)

So we can rewrite problem (*) equivalently as follows:

$$\min_f f^t L f \quad \text{subject to} \quad \sum_{i=1}^n f_i = 0 \quad \text{and} \quad f_i = \pm 1 \quad (**)$$

So far, we did not change the problem at all, we just wrote it in a different way.

It still looks difficult because it is a **discrete optimization problem**.

Relaxing balanced cut (3)

Now we are going to **relax** the problem: we simply replace the difficult condition $f_i = \pm 1$ by the two conditions $f_i \in \mathbb{R}$ and $\|f\| = 1$:

$$\min_f f^t L f \quad \text{subject to} \quad \sum_{i=1}^n f_i = 0 \quad \text{and} \quad f_i \in \mathbb{R} \quad \text{and} \quad \|f\| = 1 \quad (\#)$$

Finally, observe that $\sum_i f_i = 0 \iff f \perp \mathbb{1}$ where $\mathbb{1}$ is the constant-one vector $(1, 1, \dots, 1)$. We obtain:

$$\min_f f^t L f \quad \text{subject to} \quad f \perp \mathbb{1} \quad \text{and} \quad f_i \in \mathbb{R} \quad \text{and} \quad \|f\| = 1 \quad (\#\#)$$

Relaxing balanced cut (4)

The final observation is now:

- ▶ $\mathbb{1}$ is the smallest eigenvector of the matrix L
- ▶ So Rayleigh's principle tells us that the solution to Problem (##) is f^* being the second-smallest eigenvector of L .

To transform the solution of the relaxed problem into a partition we simply consider the sign:

$$i \in A : \iff f_i^* \geq 0$$

Relaxing balanced cut (5)

So we end up with the following algorithm:

HardBalancedCutRelaxation(G)

- 1 Input: Weight matrix (or adjacency matrix) W of the graph
- 2 $D :=$ the corresponding degree matrix
- 3 $L := D - W$ (the corresponding graph Laplacian)
- 4 Compute the second-smallest eigenvector f of L
- 5 Define the partition $A = \{i \mid f_i \geq 0\}$, $\bar{A} = V \setminus A$
- 6 Return A, \bar{A}

Relaxing balanced cut (6)

Remarks about the relaxation approach:

- ▶ Our original problem was NP hard.
- ▶ We now solve a **relaxed problem** (in polynomial time, see below).
- ▶ In general, relaxing a problem does not lead to any guarantees about whether the solution of the relaxed problem is close to the solution of the original problem.
- ▶ This is also the case for spectral clustering. We can construct example graphs for which the relaxation is arbitrarily bad. However, such examples are very artificial.
- ▶ However, in practice the spectral relaxation works very well!!!

Relaxing RatioCut

Now we want to solve the soft balanced mincut problem of optimizing ratio cut:

$$\min_{A \subset V} \text{RatioCut}(A, \bar{A}) \quad (*)$$

This goes along the same lines as the hard balanced mincut problem:

- Define particular values of f_i , namely

$$f_i = \begin{cases} +(|\bar{A}|/|A|)^{1/2} & \text{if } i \in A \\ -(|A|/|\bar{A}|)^{1/2} & \text{if } i \in \bar{A} \end{cases}$$

- Observe that we can write $\text{RatioCut}(A, \bar{A}) = \dots = f^t L f$.

Relaxing RatioCut (2)

- So the RatioCut problem is equivalent to

$$\min_f f^t L f \text{ subject to } f_i \text{ "of the form given above"} \quad (**)$$

- Also observe that any f of the form given above satisfies $\sum_{i \in V} f_i = \dots = 0$. So any such f satisfies $f \perp \mathbb{1}$.
- So the RatioCut problem is also equivalent to

$$\min_f f^t L f \text{ subject to } f \perp \mathbb{1} \text{ and } f_i \text{ "of the form given above"} \quad (**)$$

- Now we relax the condition f_i "of the form given above" to $f_i \in \mathbb{R}$, and apply Rayleigh to see that we need to compute the second eigenvector.

Relaxing RatioCut (3)

- As before, we assign points to A and \bar{A} according to the sign of the resulting f^* .

In pseudo-code, this algorithm is exactly the one we have already seen above. It is called (unnormalized) spectral clustering.

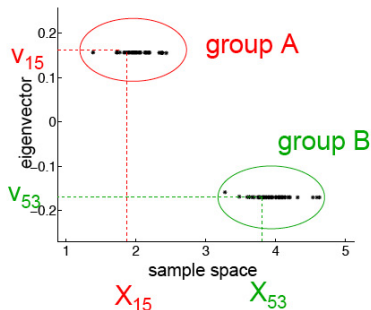
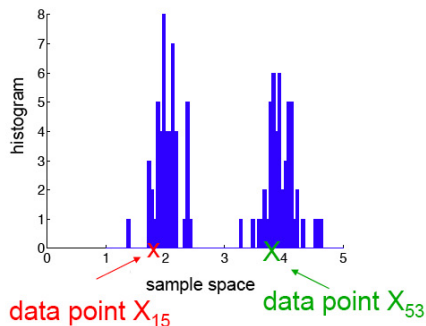
(Unnormalized) Spectral Clustering, case two clusters

UnnormalizedSpectralClustering(G)

- 1 Input: Weight matrix (or adjacency matrix) W of the graph
- 2 $D :=$ the corresponding degree matrix
- 3 $L := D - W$ (the corresponding graph Laplacian)
- 4 Compute the second-smallest eigenvector f of L
- 5 Define the partition $A = \{i \mid f_i \geq 0\}$, $\bar{A} = V \setminus A$
- 6 Return A, \bar{A}

Examples

A couple of data points drawn from a mixture of Gaussians on \mathbb{R} .

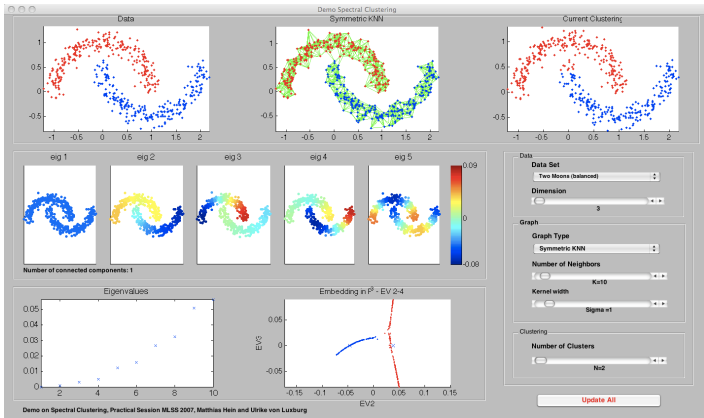


eigenvector $v = (-0.18, -0.18, 0.17, 0.18, \dots, \mathbf{0.17}, \dots \mathbf{-0.18}, \dots)$

Examples (2)

/Users/ule/matlab_ule/not_in_path/demos_for_teaching/practical_session_summerschool_tuebingen07/

GraphDemos



Unnormalized spectral clustering for k clusters

One can extend the algorithm to the case of k clusters.

General idea:

- ▶ The first k eigenvectors encode the cluster structure of k disjoint clusters.
(To see this, consider the case of k perfectly disconnected clusters)
- ▶ To extract the cluster information from the first k eigenvectors, we construct the so-called **spectral embedding**:
 - ▶ Let V be the matrix that contains the first k eigenvectors as columns.
 - ▶ Now define new points $Y_i \in \mathbb{R}^k$ as the i -th row of matrix V .

Unnormalized spectral clustering for k clusters

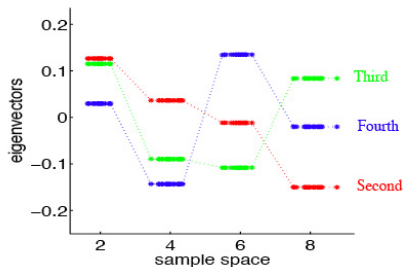
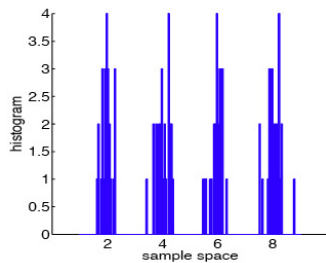
(2)

- Note: in the “ideal case” (disconnected clusters) the Y_i are the same for all points in the same cluster.

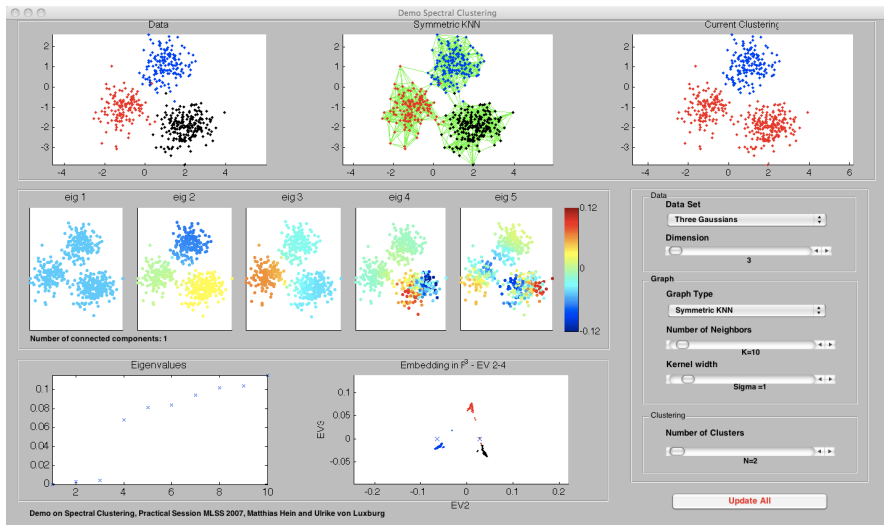
$$V = \begin{matrix} & \overbrace{\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{pmatrix}}^k \\ \left. \begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \end{matrix} \right\} n & \left. \begin{matrix} \} \\ \} \\ \} \end{matrix} \right\} \begin{matrix} (1 \ 0 \ 0) \\ (0 \ 1 \ 0) \\ (0 \ 0 \ 1) \end{matrix} \end{matrix}$$

- Idea: they are “nearly the same” if we still have nice (but not perfect) clusters.
- In particular, any simple algorithm can recover the cluster membership based on the embedded points Y_i . We use k -means to do so.

Unnormalized spectral clustering for k clusters (3)



Unnormalized spectral clustering for k clusters (4)



Unnormalized spectral clustering for k clusters (5)

UnnormalizedSpectralClustering(G)

- 1 Input: Weight matrix (or adjacency matrix) W of the graph
- 2 $D :=$ the corresponding degree matrix
- 3 $L := D - W$ (the corresponding graph Laplacian)
- 4 Compute the $n \times k$ matrix V that contains the first k eigenvectors as columns.
- 5 Define the new data points $Y_i \in \mathbb{R}^k$ to be the rows of the matrix V . This is sometimes called the spectral embedding.
- 6 Now cluster the points $(Y_i)_{i=1,\dots,n}$ by the k -means algorithm.

Unnormalized spectral clustering for k clusters (6)

Some more intuition:

- ▶ Seems funny: we first say we want to use spectral clustering, and in the end we run k -means.
- ▶ The point is that the spectral embedding is such a clever transformation of the original data that after this transformation the cluster structure is “obvious”, we just have to extract it by a simple algorithm.

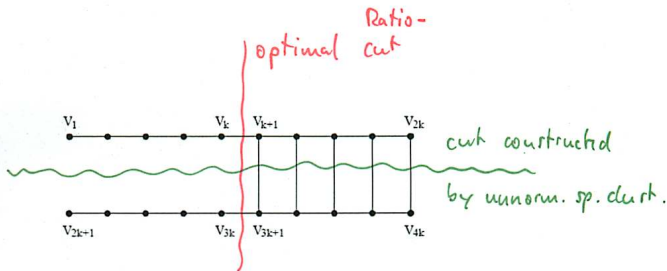
Analysis: Running time

The bottleneck of the algorithm is the computation of the eigenvector:

- ▶ In general, the first eigenvectors of a symmetric matrix can be computed in time $O(n^3)$
- ▶ However, one can do much better on sparse matrices (running time then depends on the sparsity and on other conditions such as the “spectral gap”).
See also the slides on the power method (page 1385) in the maths appendix.

Analysis: No approximation guarantees

- ▶ As hinted above: we cannot guarantee that the solution we find by unnormalized spectral clustering is close to the minimizer of RatioCut
- ▶ In the following example, the cut constructed by spectral clustering is $c \cdot n$ times larger than the best RatioCut:



Note that this example relies heavily on symmetry.

Guattery, S., Miller, G. (1998). On the quality of spectral separators. SIAM Journal of Matrix Anal. Appl., 1998.

Analysis: No approximation guarantees (2)

- ▶ However, despite the lack of approximation guarantees, it performs extremely well in practice. In terms of clustering performance, it is the state of the art and one of the most widely used “modern” algorithms for clustering.

Normalized spectral clustering

Normalized cut criterion

We have seen that the unnormalized spectral clustering algorithm solves the (relaxed) problem of minimizing RatioCut.

For various reasons (see later), it turns out to be better to consider the following objective function called **normalized cut**:

$$\text{Ncut}(A, B) = \text{cut}(A, B) \left(\frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)} \right)$$

This looks very similar to RatioCut, but we measure the size of the sets A and B not by their number of vertices, but by the weight of their edges:

$$\text{vol}(A) = \sum_{i \in A} d_i$$

Normalized cut criterion (2)

By a derivation that is very similar to what we have seen for Ratocut:

- ▶ relaxing the problem to minimizing N_{cut} leads to clustering the eigenvectors of the random walk Laplacian L_{rw} .
- ▶ For computational reasons, one replaces the eigendecomposition of L_{rw} by the one of L_{sym} (WHY?)

Minimizing normalized cut

Relaxation approach, very similar to the one for Ratocut, leads to the following algorithm:

NormalizedSpectralClustering(G)

- 1 Input: Weight matrix (or adjacency matrix) W of the graph
- 2 $D :=$ the corresponding degree matrix
- 3 $L_{sym} := D^{-1/2}(D - W)D^{-1/2}$ (the normalized graph Laplacian)
- 4 Compute the second-smallest eigenvector f of L_{sym}
- 5 Compute the vector $g = D^{-1/2}f$
- 6 Define the partition $A = \{i \mid g_i \geq 0\}$, $\bar{A} = V \setminus A$
- 7 Return A, \bar{A}

Normalized vs. unnormalized spectral clustering

You should always prefer the normalized spectral clustering algorithm. There are several theoretical (and practical) results that show this. Here is one of them:

The unnormalized spectral algorithm is not statistically consistent: if the sample size increases, the second eigenvector of the unnormalized Laplacian can converge to a trivial Dirac function that just separates one point from the rest of the space. This never happens for normalized spectral clustering. Details are beyond the scope of this lecture and require heavy functional analysis.

Literature: Luxburg, Bousquet, Belkin: Consistency of spectral clustering, 2008

Regularized spectral clustering

Literature:

- ▶ Zhang, Rohe: Understanding Regularized Spectral Clustering via Graph Conductance, NeurIPS, 2018
- ▶ Chaudhuri, Chung, Tsias: Spectral clustering of graphs with general degrees in the planted partition model, COLT 2012
- ▶ Amini, Cheng, Bickel, Levina: Pseudo-Likelihood methods for community detection, Annals of Statistics, 2013

Regularization for spectral clustering

- ▶ In practice it still happens regularly that normalized spectral clustering identifies outliers rather than the true clusters. The reason is that the balancing is not strong enough.
- ▶ There is a very simple cure for it that helps very often: add a little regularization term to the adjacency matrix before applying normalized spectral clustering. Concretely, replace the weight matrix W by the matrix

$$\tilde{W} := W + \frac{\tau}{n} J$$

where J is the all-ones-matrix and τ a small parameter, and then compute the normalized Laplacian and use spectral clustering as usual:

$$\tilde{D} = \text{degrees of } \tilde{W}$$

$$\tilde{L} = \tilde{D} - \tilde{W}$$

Regularization for spectral clustering (2)

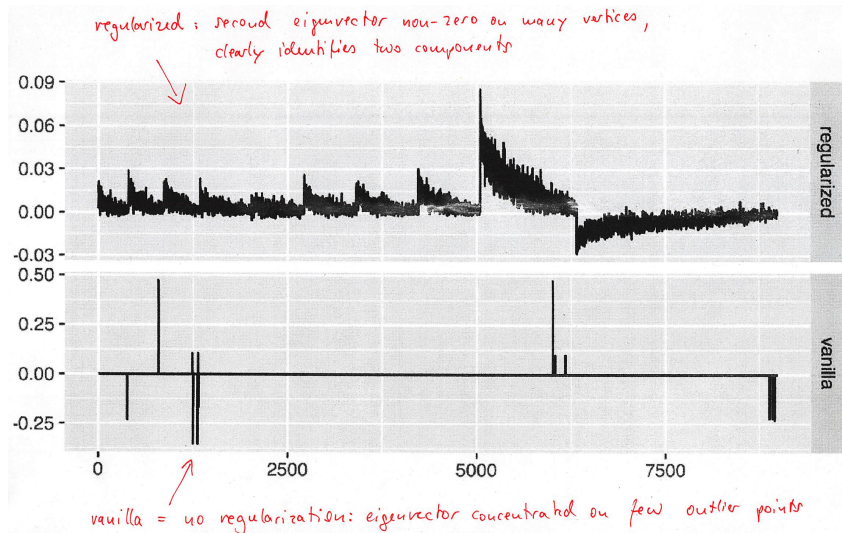
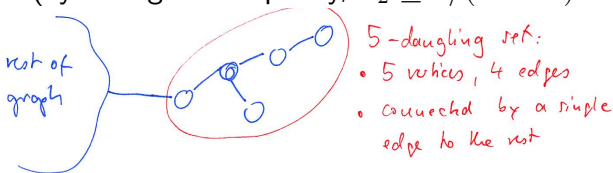


Figure: Karl Rohe

Regularization for spectral clustering (3)

Why does regularization help? Here is the line of argument:

- ▶ Many sparse random graphs provably have many “ k -dangling sets”, and those sets create small eigenvalues of the normalized Laplacian (by Cheeger's inequality, $\lambda_2 \leq 2/(2k - 1) \approx 1/k$)



- ▶ If we regularize, one can prove that these eigenvalues “disappear”: if the graph has a good cluster structure, then one can prove that the correct cut has a smaller value in the regularized graph than the cut of dangling sets.

Regularization for spectral clustering (4)

Hang on, the regularized matrix is dense! So do we run into computational issues?

No, we can still use the power method and exploit sparsity of the graph:

$$\begin{aligned}(\tilde{D}^{-1/2} \tilde{W} \tilde{D}^{-1/2})v &= \tilde{D}^{-1/2} \left(W + \frac{\tau}{n} J \right) \tilde{D}^{-1/2} v \\ &= \underbrace{\tilde{D}^{-1/2} W \tilde{D}^{-1/2} v}_{\text{sparse}} + \underbrace{\frac{\tau}{n} \mathbb{1}(\mathbb{1}v)}_{O(n)}\end{aligned}$$

History of spectral clustering

- ▶ Has been discovered and rediscovered several times since the 1970ies, but went pretty much unnoticed.
- ▶ Breakthrough papers:
 - ▶ *Shi, J. and Malik, J. Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000.*
 - ▶ *Meila, M. and Shi, J. A random walks view of spectral segmentation. AISTATS, 2001.*
 - ▶ *Ng, A., Jordan, M., and Weiss, Y. On spectral clustering: analysis and an algorithm. NIPS, 2002.*
- ▶ By now, it has been established as the most popular “modern” clustering algorithm, with many theoretical results underpinning its usefulness.

Still active field of research, e.g. see regularized spectral clustering.

Spectral clustering summary

- ▶ Spectral clustering tries to solve a balanced cut problem:
 - ▶ minimize Ratocut (\leadsto unnormalized spectral clustering)
 - ▶ minimize Ncut (\leadsto normalized spectral clustering)
- ▶ Both these problems are discrete optimization problems and NP hard to solve.
- ▶ Spectral clustering solves a relaxed version of these problems.
- ▶ In theory, there are no approximation guarantees — the relaxed solution can be miles away from the one we want. In practice, it works very well and is state of the art in many applications.
- ▶ Running time complexity can be as bad as $O(n^3)$, but for sparse graphs it is very fast.

Normalized spectral clustering is THE modern state of the art clustering algorithm.

Introduction to learning theory

The classic theory for supervised learning

Classic Learning Theory: setup and main questions

Literature on classic learning theory:

- ▶ High-level: U. von Luxburg and B. Schölkopf. Statistical Learning Theory: Models, Concepts, and Results. 2011.
- ▶ More technical: Bousquet, Boucheron, Lugosi: Introduction to statistical learning theory, 2003
- ▶ The “classic” book (technical): Devroye, Györfi, Lugosi: A probabilistic theory of pattern recognition. Springer, 1996
- ▶ Some of the general text books cover different aspects, for example the books by Shalev-Shwartz, Ben-David and Mohri, Rostamizadeh, Talwalkar.

Statistical learning theory

On an abstract level, SLT tries to answer questions such as:

- ▶ Which **learning tasks** can be performed by computers in general (positive and negative results)?
- ▶ What kind of **assumptions** do we have to make such that machine learning can be successful?
- ▶ What are the key properties a **learning algorithm** needs to satisfy in order to be successful?
- ▶ Which **performance guarantees** can we give on the results of certain learning algorithms?

In the following we focus on the case of binary classification.

The framework

The data. Data points (X_i, Y_i) are an i.i.d. sample from some underlying (unknown) probability distribution P on the space $\mathcal{X} \times \{\pm 1\}$.

Goal. Our goal is to learn a deterministic function $f : \mathcal{X} \rightarrow \{\pm 1\}$ such that the expected loss (risk) according to some given loss function ℓ is as small as possible. In classification, the natural loss function is the 0-1-loss.

The framework (2)

Assumptions we (do not) make:

- ▶ We do not make any assumption on the **underlying distribution** P that generates our data, it can be anything.
- ▶ True **labels** do not have to be a deterministic function of the input (consider the example of predicting male/female based on body height).
- ▶ Data points have been sampled **i.i.d.**
- ▶ Data does not change over time (the ordering of the training points does not matter, and the distribution P does not change),
- ▶ The distribution P is unknown at the time of learning.

Recap: Bayes classifier

The Bayes classifier

The Bayes classifier for a particular learning problem is the classifier that achieves the minimal expected risk.

Have already seen: if we knew the underlying distribution P , then we also know the Bayes classifier (just look at the regression function).

The challenge is that we do not know P . The goal is now to construct a classifier that is “as close to the Bayes classifier” as possible. Now let's become more formal.

Convergence and consistency

Assume we have a set of n points (X_i, Y_i) drawn from P . Consider a given function class \mathcal{F} from which we are allowed to pick our classifier. Denote:

- ▶ f^* the Bayes classifier corresponding to P .
- ▶ $f_{\mathcal{F}}$ the best classifier in \mathcal{F} , that is

$$f_{\mathcal{F}} = \operatorname{argmin}_{f \in \mathcal{F}} R(f)$$

- ▶ f_n the classifier chosen from \mathcal{F} by some training algorithm on the given sample of n points.

Now consider the following definitions:

Convergence and consistency (2)

1. A learning algorithm is called **consistent with respect to \mathcal{F} and P** if the risk $R(f_n)$ converges in probability to the risk $R(f_{\mathcal{F}})$ of the best classifier in \mathcal{F} , that is for all $\varepsilon > 0$,

$$P(R(f_n) - R(f_{\mathcal{F}}) > \varepsilon) \rightarrow 0 \text{ as } n \rightarrow \infty.$$

2. A learning algorithm is called **Bayes-consistent with respect to P** if the risk $R(f_n)$ converges to the risk $R(f^*)$ of the Bayes classifier, that is for all $\varepsilon > 0$,

$$P(R(f_n) - R(f^*) > \varepsilon) \rightarrow 0 \text{ as } n \rightarrow \infty.$$

3. A learning algorithm is called **universally consistent with respect to \mathcal{F} (resp. universally Bayes-consistent)** if it is consistent with respect to \mathcal{F} (resp. Bayes-consistent) for all probability distributions P .

Convergence and consistency (3)

Note that consistency with respect to a fixed function class \mathcal{F} only concerns the estimation error, not the approximation error:

- ▶ Here consistency means that our decisions are not affected systematically from the fact that we only get to see a finite sample, rather than the full space. In other words, all “finite sample effects” cancel out once we get to see enough data.
- ▶ If a learning algorithm is consistent, it means that it does not overfit when it gets to see enough data (low estimation error, low variance).
- ▶ Consistency with respect to \mathcal{F} does not tell us anything about underfitting (approximation error; this depends on the choice of \mathcal{F}).

Empirical risk minimization (ERM)

True risk of a function: $R(f) = E(\ell(X, f(X), Y))$

Empirical risk: $R_n(f) = \frac{1}{n} \sum_{i=1}^n \ell(X_i, f(X_i), Y_i)$

Empirical risk minimization: given n training points $(X_i, Y_i)_{i=1, \dots, n}$ and a fixed function class \mathcal{F} , select the function f_n that minimizes the training error on the data:

$$f_n = \operatorname{argmin}_{f \in \mathcal{F}} R_n(f)$$

Earlier we informally discussed that ERM won't work if the function class \mathcal{F} is "too large". We are now going to make this formal.

Controlling the estimation error: generalization bounds

Law of large numbers and concentration

Recall from probability theory:

Proposition 32 (Law of large numbers, simplest version)

Let $(Z_i)_{i \in \mathbb{N}}$ be a sequence of independent random variables that have been drawn according to some probability distribution P , denote its expectation as $E(Z)$. Then (under mild assumptions)

$$\frac{1}{n} \sum_{i=1}^n Z_i \rightarrow E(Z) \quad (\text{almost surely}).$$

Law of large numbers and concentration (2)

Note that independence is a crucial assumption for the LLN:

- ▶ Let X_1 be the toss of a fair coin. Let X_2, X_3, \dots be identical to X_1 . Then:
- ▶ Individually, each of the rv's follows the same Bernoulli $Ber(0.5)$ distribution, so they are identically distributed and have expectation $E(Z_i) = 0.5$.
- ▶ But they are not independent of course.
- ▶ And obviously, depending on the outcome of X_1 , the empirical average is either 0 or 1, so it does not converge to 0.5.

Law of large numbers and concentration (3)

Even more than the LLN, there exist very strong guarantees on how fast this convergence takes place:

Proposition 33 (Concentration inequality, Chernoff 1952, Hoeffding 1963)

Assume that the random variables Z_1, \dots, Z_n are independent and take values in $[0, 1]$. Define their sum $S_n := (1/n) \cdot \sum_{i=1}^n Z_i$. Then for any $\varepsilon > 0$

$$P(|S_n - E(S_n)| \geq \varepsilon) \leq 2 \exp(-2n\varepsilon^2).$$

Law of large numbers and concentration (4)

Observe:

- ▶ The rv's don't need to have the same distribution, as long as we can control their range (or more generally, their variance).
- ▶ Independence is crucial for this theorem to hold, see examples below.

LLN/Hoeffding applied to classification loss

Now consider our scenario of binary classification.

Proposition 34 (Risks converge for fixed function)

Fix a function $f_0 \in \mathcal{F}$. Then, for this **fixed function** f_0 ,

$$R_n(f_0) \rightarrow R(f_0) \quad (\text{almost surely}).$$

DO YOU SEE WHY?

LLN/Hoeffding applied to classification loss (2)

Proof:

- ▶ Apply the Hoeffding bound to the variables $Z_i := \ell(f_0(X_i), Y_i)$. This leads to convergence in probability.
- ▶ (For those who know about probability theory: To get almost sure convergence, you need to do one extra step, namely apply the Borel-Cantelli lemma. Key is that $\sum_{n=1}^{\infty} \exp(-2n\varepsilon^2)$ is finite. Exercise.)

LLN/Hoeffding applied to classification loss (3)

Question: Let f_n be the function selected by empirical risk minimization based on the first n sample points. Does the LLN imply that

$$R_n(f_n) - R(f_n) \rightarrow 0 \quad \text{?????????????}.$$

LLN/Hoeffding applied to classification loss (4)

NO!!! Here is the formal argument:

- ▶ The key property in the LLN or Hoeffding inequality is the independence.
- ▶ However, the function f_n depends on all data points $(X_i, Y_i)_{i=1, \dots, n}$.
- ▶ So even if the data points $(X_i, Y_i)_{i=1, \dots, n}$ are independent, the random variables $(Z_i)_{i=1, \dots, n}$ with $Z_i := \ell(f_n(X_i), Y_i)$ typically are not independent any more.

LLN/Hoeffding applied to classification loss (5)

Not convinced? consider the following counter-example:

- ▶ $\mathcal{X} = [0, 1]$ with uniform distribution; labels deterministic with $x < 0.5 \implies y = -1$ and $x \geq 0.5 \implies y = +1$
- ▶ Draw n training points
- ▶ Define f_n as follows: for all points in the training sample, predict the training label; for all other points predict -1.
- ▶ Here we have $R_n(f_n) = 0$ but $R(f_n) = 0.5$, for all n .
- ▶ Or more formally, in the Hoeffding notation: if we set $Z_i := \ell(f_n(X_i), Y_i)$, then we always have $S_n = \sum_i Z_i/n = 0$, but $E(S_n) = 0.5$.

Uniform convergence

Want to have a condition that is sufficient for the convergence of the empirical risk of the data-dependent function f_n :

- ▶ We require that **for all functions in \mathcal{F}** , the empirical risk (as measured on the data) has to be close to the true risk.
- ▶ Intuitively, for any $\varepsilon > 0$, we want that with high probability,

$$\sup_{f \in \mathcal{F}} |R_n(f) - R(f)| < \varepsilon$$

- ▶ Formally, we have uniform convergence (in probability) if $\forall \varepsilon > 0 : \lim_{n \rightarrow \infty} P(\sup_{f \in \mathcal{F}} |R_n(f) - R(f)| > \varepsilon) = 0$
- ▶ Note the logic behind this condition: if $R_n(f)$ and $R(f)$ are close for all functions $f \in \mathcal{F}$, they particularly will be close for the function f_n that has been chosen by the classification algorithm.

Uniform convergence (2)

Note that in the counter-example above, this requirement is clearly not satisfied.

WHY EXACTLY?

Uniform convergence (3)

Definition:

We say that the law of large number holds uniformly over a function class \mathcal{F} if for all $\varepsilon > 0$,

$$P(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| \geq \varepsilon) \rightarrow 0 \text{ as } n \rightarrow \infty.$$

Uniform convergence: sufficient for consistency

Relatively easy to see:

Proposition 35 (Uniform convergence is sufficient for consistency)

Let f_n be the function that minimizes the empirical risk in \mathcal{F} . Then:

$$P(|R(f_n) - R(f_{\mathcal{F}})| \geq \varepsilon) \leq P(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| \geq \varepsilon/2).$$

Uniform convergence: sufficient for consistency (2)

Proof.

$$\begin{aligned} & |R(f_n) - R(f_{\mathcal{F}})| \\ & \quad (\text{by definition of } f_{\mathcal{F}} \text{ we know that } R(f_n) - R(f_{\mathcal{F}}) \geq 0) \\ & = R(f_n) - R(f_{\mathcal{F}}) \\ & = R(f_n) - R_n(f_n) + R_n(f_n) - R_n(f_{\mathcal{F}}) + R_n(f_{\mathcal{F}}) - R(f_{\mathcal{F}}) \\ & \quad (\text{note that } R_n(f_n) - R_n(f_{\mathcal{F}}) \leq 0 \text{ by def. of } f_n) \\ & \leq R(f_n) - R_n(f_n) + R_n(f_{\mathcal{F}}) - R(f_{\mathcal{F}}) \\ & \leq 2 \sup_{f \in \mathcal{F}} |R(f) - R_n(f)| \end{aligned}$$



Uniform convergence: necessary for consistency

What is much less obvious is that uniform convergence is also necessary, this is in fact a very deep result:

Theorem 36 (Vapnik & Chervonenkis, 1971)

Let \mathcal{F} be any function class. Then empirical risk minimization is uniformly consistent with respect to \mathcal{F} if and only if uniform convergence holds:

$$P(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| > \varepsilon) \rightarrow 0 \text{ as } n \rightarrow \infty, \quad (1)$$

The proof is beyond the scope of this lecture.

Uniform convergence: necessary for consistency (2)

But the big question is now:

How do we know whether we have uniform consistency for some function class \mathcal{F} ???

Capacity measures for function classes

Finite classes

Capacity measures: intuition

Have seen:

- ▶ If a function class is too large (as in the counter-example), then we don't have uniform convergence.
- ▶ If a function class is small (say, it only consists of a single function), then we have uniform convergence.

We now want to come up with ways to measure the size of a function class — in such a way that we can bound the term

$$P(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| > \varepsilon)$$

Generalization bound for finite classes

Recall the Hoeffding bound for a fixed function f_0 :

$$P(|R(f_0) - R_n(f_0)| \geq \varepsilon) \leq 2 \exp(-2n\varepsilon^2).$$

Now consider a function class with finitely many functions:

$\mathcal{F} = \{f_1, \dots, f_m\}$. We get:

$$\begin{aligned} & P(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| \geq \varepsilon) \\ &= P(\sup_{i=1, \dots, m} |R(f_i) - R_n(f_i)| \geq \varepsilon) \\ &= P\left(|R(f_1) - R_n(f_1)| \geq \varepsilon \text{ or } |R(f_2) - R_n(f_2)| \geq \varepsilon \text{ or } \dots\right) \\ &\leq \sum_{i=1}^m P(|R(f_i) - R_n(f_i)| \geq \varepsilon) \\ &\leq 2m \exp(-2n\varepsilon^2) \end{aligned}$$

Generalization bound for finite classes (2)

Leads to the first result:

Proposition 37 (Generalization of finite classes)

Assume \mathcal{F} is finite and contains m functions. Chose any $\varepsilon, 0 < \varepsilon < 1$. Then, with probability at least $1 - 2m \exp(-2n\varepsilon^2)$, we have for all $f \in \mathcal{F}$ that

$$|R(f) - R_n(f)| < \varepsilon.$$

Note that this statement is somewhat inconvenient, it is “the wrong way round”: we choose the size of the error, and get the probability that this error holds; but we would like to say that with a chosen confidence (error probability), the error is only as large as BLA.

Generalization bound for finite classes (3)

So we now try to reverse the statement: set the probability to some value δ , and then solve for ε :

$$\delta = 2m \exp(-2n\varepsilon^2) \implies \varepsilon = \sqrt{\frac{\log(2m) + \log(1/\delta)}{2n}}$$

With this, the proposition becomes the following **generalization bound**:

Generalization bound for finite classes (4)

Theorem 38 (Generalization bound for finite classes)

Assume \mathcal{F} is finite and contains m functions. Choose some failure probability $0 < \delta < 1$. Then, with probability at least $1 - \delta$, for all $f \in \mathcal{F}$ we have

$$R(f) \leq R_n(f) + \sqrt{\frac{\log(2m) + \log(1/\delta)}{2n}}$$

Note that the generalization bound holds uniformly (with the same error guarantee) for all functions in \mathcal{F} , so in particular for the function that a classifier might pick based on the sample points it has seen.

Generalization bound for finite classes (5)

Let's digest this bound:

- ▶ It bounds the true risk by the empirical risk plus a “capacity term”.
- ▶ If the function class gets larger (m increases), then the bound gets worse.
- ▶ If m is “small enough” compared to n (in the sense that $\log m/n$ is small, then we get a tight bound.
- ▶ The whole bound only holds with probability $1 - \delta$. When we decrease δ (higher confidence), the bound gets worse.
- ▶ If m is fixed, and the confidence value δ is fixed, and $n \rightarrow \infty$, then the empirical risk converges to the true risk. The speed of convergence is of the order $1/\sqrt{n}$.

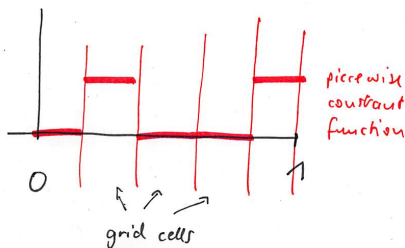
Generalization bound for finite classes (6)

- If you want to grow your function space with n in order to be able to fit more accurately if you have more data, you need to make sure that $(\log m)/n \rightarrow 0$ if you want to get consistency.

Generalization bound for finite classes (7)

EXERCISE:

Consider $\mathcal{X} = [0, 1]$, split it into a grid of k cells of the same size. As function class, consider all functions that are piecewise constant (0 or 1) on all cells. Denote this function class by \mathcal{F}_k .



Case 1: k is fixed.

- Prove that ERM is uniformly consistent with respect to \mathcal{F}_k if k is fixed.

Generalization bound for finite classes (8)

- Is the classifier also Bayes consistent? Exercise: prove or give a counter-example.

Generalization bound for finite classes (9)

Case 2: k grows with n , so formally k is a function of n , denoted by $k(n)$.

- ▶ How fast can $k(n)$ grow such that we still have consistency with respect to $\mathcal{F}_{k(n)}$?
- ▶ What about the approximation error / Bayes consistency?

Generalization bound for finite classes (10)

Bottom line:

- ▶ For finite function classes, we can measure the size of \mathcal{F} by its number m of functions.
- ▶ This leads to a generalization bound with plausible behavior.

However, what should we do if \mathcal{F} is infinite (say, space of all linear functions)? Then the approach above does not work ... WHY?

Shattering coefficient

Shattering coefficient: definition

We now want to measure the capacity of an infinite class of functions. The most basic such capacity measure is the following:

Definition: For a given sample $X_1, \dots, X_n \in \mathcal{X}$ and a function class \mathcal{F} define $\mathcal{F}_{X_1, \dots, X_n}$ as the set of those functions that we get by restricting \mathcal{F} to the sample:

$$\mathcal{F}_{X_1, \dots, X_n} := \{f|_{X_1, \dots, X_n}; f \in \mathcal{F}\}$$

The **shattering coefficient** $\mathcal{N}(\mathcal{F}, n)$ of a function class \mathcal{F} is defined as the maximal number of functions in $\mathcal{F}_{X_1, \dots, X_n}$:

$$\mathcal{N}(\mathcal{F}, n) := \max\{|\mathcal{F}_{X_1, \dots, X_n}|; X_1, \dots, X_n \in \mathcal{X}\}$$

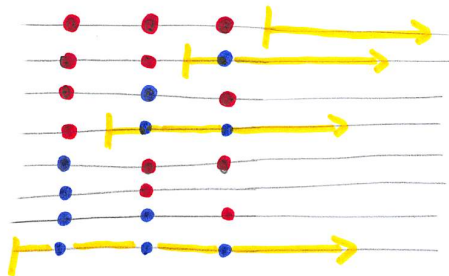
Shattering coefficient: definition (2)

Example 1: $\mathcal{X} = \mathbb{R}$, \mathcal{F} as below (positive class = right half-space)

$$\mathcal{X} = \mathbb{R}$$

$$\mathcal{F} = \{ \mathbb{1}_{[a, \infty[} \mid a \in \mathbb{R} \}$$

(all half-spaces with the
"infinite end" at the
right side)



With \mathcal{F} we can only realize
4 out of the 8 possible labelings.

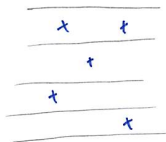
$$\text{Thus } \mathcal{N}(\mathcal{F}, 3) = 4.$$

(because this argument holds for
all possible sets of three points)

Shattering coefficient: definition (3)

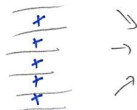
Example 2: $\mathcal{X} = \mathbb{R}^2$, \mathcal{F} such that positive class = space above a horizontal line

Data 1 :



5 different ways to separate them $\Rightarrow \mathcal{N}(\mathcal{F}, 5) \geq 5$

Data 2 :



6 different ways $\Rightarrow \mathcal{N}(\mathcal{F}, 5) \geq 6$

Data 3 :

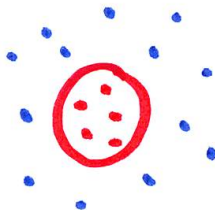
x x x x x

two different ways

$\Rightarrow \mathcal{N}(\mathcal{F}, 5) \geq 2$

Shattering coefficient: definition (4)

Example 3: $\mathcal{X} = \mathbb{R}^2$, $\mathcal{F} =$ interior of circles.



CAN YOU COME UP WITH A BOUND ON THE SHATTERING COEFFICIENT FOR A SMALL n ?

Shattering coefficient: generalization bound

Theorem 39 (Generalization bound with shattering coefficient)

Let \mathcal{F} be any arbitrary function class. Then for all $0 < \varepsilon < 1$,

$$\mathbb{P}(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| > \varepsilon) \leq 2\mathcal{N}(\mathcal{F}, 2n) \exp(-n\varepsilon^2/4).$$

The other way round: With probability at least $1 - \delta$, all functions $f \in \mathcal{F}$ satisfy

$$R(f) \leq R_n(f) + 2\sqrt{\frac{\log(\mathcal{N}(\mathcal{F}, 2n)) - \log(\delta)}{n}}.$$

Proof of Theorem 39 by symmetrization

- ▶ By R_n we denote the risk on our given sample of n points.
- ▶ By R'_n we denote the risk that we get on a second, independent sample of n points, called the “ghost sample”.

Proposition 40 (Symmetrization lemma)

$$\begin{aligned} & \mathbb{P}(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| > \varepsilon) \\ & \leq 2 \mathbb{P}(\sup_{f \in \mathcal{F}} |R_n(f) - R'_n(f)| > \varepsilon/2). \end{aligned}$$

(Proof elementary, omitted)

Proof of Theorem 39 by symmetrization (2)

What is the point of symmetrization?

- ▶ The right hand side only depends on the values of the functions f on the two samples:
If two functions f and g coincide on all points of the original sample and the ghost sample, that is $f(x) = g(x)$ for all x in the samples, then $R_n(f) = R_n(g)$ and $R'_n(f) = R'_n(g)$.
- ▶ So the supremum over $f \in \mathcal{F}$ in fact only runs over finitely many functions: all possible binary functions on the two samples.
- ▶ The number of such functions is bounded by the shattering coefficient $\mathcal{N}(\mathcal{F}, 2n)$.
- ▶ Now Theorem 39 is a consequence of Theorem 38.

Discussion of the generalization bound with shattering coefficient

- ▶ The bound is analogous to the one for finite function classes, just the number m of functions has been replaced by the shattering coefficient.
- ▶ Intuitively, the shattering coefficient measures “how powerful” a function class is, how many different labelings of a data set it can possibly realize.
- ▶ Overfitting happens if a function class is very powerful and can in principle fit everything. Then we don't get consistency, the shattering coefficient is large.
The smaller the shattering coefficient, the less prone we are to overfitting (in the extreme case of one function, we don't overfit).

Discussion of the generalization bound with shattering coefficient (2)

- To prove consistency of a classifier, we need to establish that $\log \mathcal{N}(\mathcal{F}, 2n)/n \rightarrow 0$ as $n \rightarrow \infty$.

Intuitively: the number of possibilities in which a data set can be labeled has to grow at most polynomially in n .

- Shattering coefficients are complicated to compute and to deal with. To prove consistency, we would need to know how fast the shattering coefficients grow with n (exponentially or less).

We now study a tool that can help us with this.

VC dimension

VC dimension: Definition

Definition: We say that a function class \mathcal{F} **shatters** a set of points X_1, \dots, X_n if \mathcal{F} can realize all possible labelings of the points, that is $|\mathcal{F}_{X_1, \dots, X_n}| = 2^n$.

The **VC dimension of \mathcal{F}** is defined as the largest number n such that there exists a sample of size n which is shattered by \mathcal{F} .

Formally,

$$\text{VC}(\mathcal{F}) = \max\{n \in \mathbb{N} \mid \exists X_1, \dots, X_n \in \mathcal{X} \text{ s.t. } |\mathcal{F}_{X_1, \dots, X_n}| = 2^n\}.$$

If the maximum does not exist, the VC dimension is defined to be infinity.

(VC stands for Vapnik-Chervonenkis, the people who invented it)

VC dimension: Definition (2)

Example: positive class = closed interval

$$\mathcal{X} = \mathbb{R}, \quad \mathcal{F} = \left\{ \mathbb{1}_{[a,b]} \mid a, b \in \mathbb{R} \right\} \quad \text{"closed intervals"}$$

- Can find a set of two points that can be shattered:



- Can we find a set of three points that can be shattered?

No!



Can never realize their labeling

$$\Rightarrow VC(\mathcal{F}) = 2$$

VC dimension: Definition (3)

Example: positive class = interior of a axis aligned rectangle

$\mathcal{X} = \mathbb{R}^2$, \mathcal{F} = axis-parallel rectangles

- There exists a point configuration of 4 points that can be shattered:



(verify it yourself).

$$\Rightarrow VC \geq 4$$

[Also note: we cannot shatter all sets of 4 points, here is an example that cannot be shattered:



- No set of 5 points can be shattered:
 - There has to exist one point that is not an extreme point in both axis-parallel directions



$$\Rightarrow VC < 5$$

VC dimension: Definition (4)

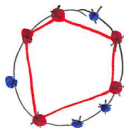
Example: positive class = interior of a convex polygon

$\mathcal{X} = \mathbb{R}^2$, $\mathcal{F}_d =$ convex polygons with d corners

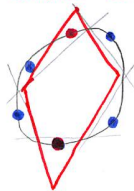
Then: $VC = 2d + 1$.



Lower bound: $2d+1$ points on a circle can be shattered:



If we have less than d red points



If we have more than d red points

Upper bound: more technical, prove that max. Number of shattered points is achieved for points on a circle.

VC dimension: Definition (5)

Example: sine waves

$$\mathcal{X} = \mathbb{R}, \quad \mathcal{F} = \{ \sin(tx) \mid t \in \mathbb{R} \}$$

Then $VC = \infty$.



VC dimension: Definition (6)

Finally, examples that are relevant for practice (SVMs!):

- ▶ $\mathcal{X} = \mathbb{R}^d$, \mathcal{F} = linear hyperplanes. Then $VC(\mathcal{F}) = d + 1$.
Proof see exercises.
- ▶ $\mathcal{X} = \mathbb{R}^d$, $\rho > 0$, $\mathcal{F}_\rho :=$ linear hyperplanes with margin at least ρ . Then one can prove: if the data points are restricted to a ball of radius R , then

$$VC(\mathcal{F}) = \min \left\{ d, \frac{2R^2}{\rho^2} \right\} + 1$$

VC dimension: Sauer-Shelah Lemma

Why are we interested in the VC dimension? Here is the reason:

Proposition 41 (Vapnik, Chervonenkis, Sauer, Shelah)

Let \mathcal{F} be a function class with finite VC dimension d . Then

$$\mathcal{N}(\mathcal{F}, n) \leq \sum_{i=0}^d \binom{n}{i}$$

for all $n \in \mathbb{N}$. In particular, for all $n \geq d$ we have

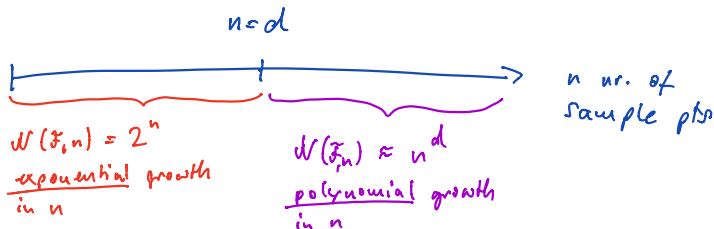
$$\mathcal{N}(\mathcal{F}, n) \leq \left(\frac{en}{d}\right)^d.$$

Proof: nice combinatorial argument, see the exercises.

VC dimension: Sauer-Shelah Lemma (2)

This is a really cool statement:

- ▶ If a function class has a finite VC dimension, then the shattering coefficient only grows polynomially!
- ▶ If a function class has infinite VC dimension, then the shattering coefficient grows exponentially.
- ▶ It is impossible that the growth rate of the function class is “slightly smaller” than 2^n . Either it is 2^n , or much smaller, polynomial.



Generalization bound with VC dimension

Plugging the Sauer-Shelah-Lemma in Theorem 39 immediately gives the following generalization bound in terms of the VC dimension:

Theorem 42 (Generalization bound with VC dimension)

Let \mathcal{F} be a function class with VC dimension d . Then with probability at least $1 - \delta$, all functions $f \in \mathcal{F}$ satisfy

$$R(f) \leq R_n(f) + 2\sqrt{\frac{d \log(2en/d) - \log(\delta)}{n}}.$$

Consequence: VC-dim finite \implies consistency

Generalization bound with VC dimension (2)

More generally, the statement also holds the other way round:

Theorem 43

Empirical risk minimization is consistent with respect to \mathcal{F} if and only if $VC(\mathcal{F})$ is finite.

Proof skipped.

Generalization bound with VC dimension (3)

Yet another interpretation of the generalization bound: how many samples do we need to draw to achieve error at most ε ?

- ▶ Set $\varepsilon := 2\sqrt{\frac{d \log(2en/d) - \log(\delta)}{n}}$, solve for n and ignore all constants.
- ▶ Result: We need of the order $n = d/\varepsilon^2$ many sample points.

Rademacher complexity

Rademacher complexity

The shattering coefficient is a purely combinatorial object, it does not take into account what the actual probability distribution is. This seems suboptimal.

Definition: Fix a number n of points. Let $\sigma_1, \dots, \sigma_n$ be i.i.d. tosses of a fair coin (result is -1 or 1 with probability 0.5 each). The **Rademacher complexity** of a function class \mathcal{F} with respect to n is defined as

$$\text{Rad}_n(\mathcal{F}) := E \sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \sigma_i f(X_i)$$

The expectation is both over the draw of the random points X_i and the random labels σ_i .

It measures how well a function class can fit random labels.

Rademacher complexity (2)

There exist a number of generalization bounds for Rademacher complexities, and they tend to be sharper than the ones by combinatorial concepts like shattering coefficients. They typically look like this:

Theorem 44 (Rademacher generalization bound)

With probability at least $1 - \delta$, for all $f \in \mathcal{F}$,

$$R(f) \leq R_n(f) + 2 \operatorname{Rad}_n(\mathcal{F}) + \sqrt{\frac{\log(1/\delta)}{2n}}$$

Proofs are beyond the scope of this lecture.

Rademacher complexity (3)

Computing Rademacher complexities for function classes is in many cases much simpler than computing shattering coefficients or VC dimensions.

Generalization bounds: conclusions

- ▶ Generalization bounds are a tool to answer the question whether a learning algorithm is consistent.
- ▶ Consistency refers to the estimation error, not the approximation error.
- ▶ Typically, generalization bounds have the following form:
With probability at least $1 - \delta$, for all $f \in \mathcal{F}$

$$R(f) \leq R_n(f) + \text{capacity term} + \text{confidence term}$$

The capacity term measures the size of the function class. The confidence term deals with how certain we are about our statement.

- ▶ There are many different ways to measure the capacity of function classes, we just scratched the surface.

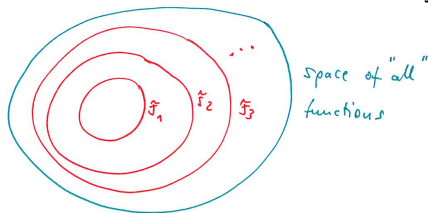
Generalization bounds: conclusions (2)

- Generalizations are worst case bounds: worst case over all possible probability distributions, and worst case over all learning algorithms that pick a function from \mathcal{F} .

Controlling the approximation error

Nested function classes

- ▶ So far, we always fixed a function class \mathcal{F} and investigated whether the estimation error in this class vanishes as we get to see more data.
- ▶ However, we need to take into account the approximation error as well.
- ▶ Idea is now: consider function classes that slowly grow with n :



Nested function classes (2)

- ▶ If we have few data, the class is supposed to be small to avoid overfitting (generalization bound!)
- ▶ Eventually, when we see enough data, we can afford a larger function class without overfitting. The larger the class, the smaller our approximation error.

There are two major approaches to this:

- ▶ Structural risk minimization: explicit approach
- ▶ Regularization: implicit approach

Structural risk minimization (SRM)

- ▶ Consider a nested sequence of function spaces: $\mathcal{F}_1 \subset \mathcal{F}_2 \subset \dots$
- ▶ We now select an appropriate function class and a good function in this class simultaneously:

$$f_n := \operatorname{argmin}_{m \in \mathbb{N}, f \in \mathcal{F}_m} R_n(f) + \text{capacity term}(\mathcal{F}_m)$$

- ▶ The capacity term is the one that comes from a generalization bound.
- ▶ If the nested function classes approximate the space of “all” functions, one can prove that such an approach can lead to universal consistency.

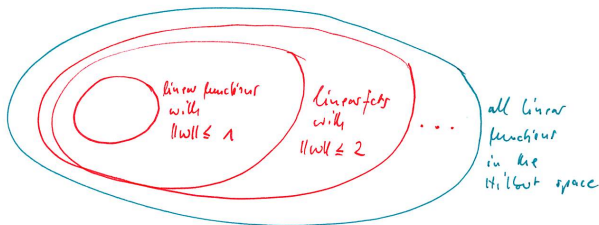
Regularization

Recap: regularized risk minimization:

$$\text{minimize } R_n(f) + \lambda \cdot \Omega(f)$$

where Ω punishes “complex” functions.

The trick is now: Regularization is an implicit way of performing structural risk minimization.



Regularization (2)

Proving consistency for regularization is technical but very elegant:

- ▶ Make sure that your overall space of functions \mathcal{F} is dense in the space of continuous functions

Example: linear combinations of a universal kernel.

- ▶ Consider a sequence of regularization constants λ_n with $\lambda_n \rightarrow 0$ as $n \rightarrow \infty$.
- ▶ Define function class $\mathcal{F}_n := \{f \in \mathcal{F} ; \lambda_n \cdot \Omega(f) \leq \text{const}\}$
- ▶ Choose $\lambda_n \rightarrow 0$ so slow that $\log \mathcal{N}(\mathcal{F}_n, n)/n \rightarrow 0$.
 - ▶ On the one hand, this ensures that in the limit we won't overfit, the estimation error goes to 0.
 - ▶ On the other hand, if $\lambda_n \rightarrow 0$, then $\mathcal{F}_n \rightarrow \mathcal{F}$ because $\lambda \Omega(f) < c \implies \Omega(f) \leq c/\lambda_n \rightarrow \infty$.
Hence, the approximation error goes to 0, so we won't underfit.

Regularization (3)

If you want to see the mathematical details, I recommend the following paper:

Steinwart: Support Vector Machines Are Universally Consistent.
Journal of Complexity, 2002.

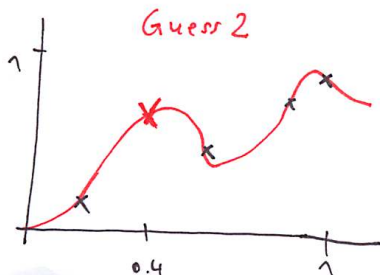
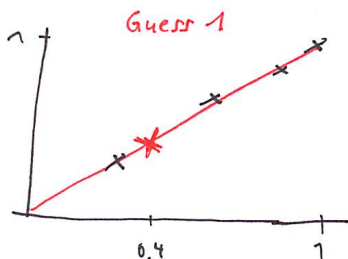
Brief history

- ▶ The first proof that there exists a learning algorithm that is universally Bayes consistent was the Theorem of Stone 1977, about the kNN classifier.
- ▶ The combinatorial tools and generalization bounds have essentially been developed in the early 1970ies already (Vapnik, Chervonenkis, 1971, 1972, etc) and refined in the years around 2000.
- ▶ The statistics community also proved many results, in particular rates of convergence. There the focus is more on regression rather than classification.
- ▶ By and large, the theory is well understood by now, the focus of attention moved to different areas of machine learning theory (for example, online learning, unsupervised learning, etc).

Getting back to Occam's razor

Examples revisited

Remember the examples we discussed in the first lecture?



The question was which of the two functions should be preferred.

Many of you had argued that unless we have a strong belief that the right curve is correct, we should prefer the left one due to “simplicity”.

Examples revisited (2)

This principle is often called “Occam’s razor” or “principle of parsimony”:

When we choose from a set of otherwise equivalent models, the simpler model should be preferred.

Intuitive argument:

“Occam’s razor helps us to shave off those concepts, variables or constructs that are not really needed to explain the phenomenon. By doing that, developing the model will become easier, and there is less chance of introducing inconsistencies, ambiguities and redundancies. “

These formulations can be found in many papers and text books, I don’t know the original source ...

Occam's razor vs. learning theory

However:

- ▶ The main message of learning theory was that we need to control the size of the function class \mathcal{F} .
- ▶ We had not at all talked about “simplicity” of functions!

Is this a contradiction? Is Occam's razor wrong???

Occam's razor vs. learning theory (2)

First point of view: we don't need "simplicity":

- ▶ Consider an example of a function class that just contains 10 functions, all of which are very "complicated" (not "simple").
- ▶ For the estimation error, this would be great, we would soon be able to detect which of the function minimizes the ERM, with high probability.
- ▶ If the function class also happens to be able to describe the underlying phenomenon (low approximation error), this would be perfect.
- ▶ In this case, we do not need simple functions!!!

Occam's razor vs. learning theory (3)

Second point of view: **Spaces of simple functions tend to be small.**

Example: Polynomials in one variable, with a discrete set of coefficients:

$$f(x) = \sum_{k=1}^d a_k x^k \text{ with } a_k \in \{-1, -0.99, -0.98, \dots, 0.98, 0.99, 1\}$$

There are about 200 polynomials of degree 1,
 200^2 polynomials of degree 2,
 200^d polynomials of degree d .

Here, the spaces get larger the more “parameters” we have.

Occam's razor vs. learning theory (4)

Both points of view come together if we talk about data compression.

- ▶ A space with few functions can be represented with few bits (say, by a small lookup table).
- ▶ A space with “simple” functions can be represented with few bits as well (encode all the parameters).
- ▶ A space of “complex” function cannot be compressed.

Intuitive conclusion:

- ▶ Spaces of simple functions are small, spaces of complex functions tend to be large.
- ▶ Learning theory tells us that we should prefer small function spaces.
- ▶ This often leads to spaces of simple functions.

Occam's razor vs. learning theory (5)

This intuition can be made rigorous and formal:

- ▶ Sample compression bounds in statistical learning theory
- ▶ The whole branch of learning based on the “Minimum description length principle” (comprehensive book in this area by Peter Grünwald)

Occam's razor vs. learning theory (6)

Bottom line:

- ▶ The quantity that is important is not so much the simplicity of the functions but rather the size of the function space.
- ▶ But spaces of simple functions tend to be small and are good candidates for learning.
- ▶ Occam's razor slightly misses the point, but is a good first proxy. It is not always correct, but often...

(*) The No-Free-Lunch Theorem

Literature:

- ▶ The way I present it is taken from the paper: Ho and Pepyne. Simple explanation of the no-free lunch theorem and its implications. Journal of Optimization Theory and Applications, 2002.
- ▶ The book by Shalev-Shwartz and Ben-David discusses the NFL as well in Sec. 5.1, albeit with a different formulation.
- ▶ More general version can be found in Chapter 7 in Devroye / Györfi / Lugosi.

Intuition

- ▶ Intuitively the no free lunch theorem (NFL) says that there does not exist a single best classifier that outperforms any other classifier on all learning problems.
- ▶ There exist many different versions to state this formally, below we describe the easiest one.

NFL, simple version

- ▶ Assume that the space of all input points just consists of a finite set $\mathcal{X} = \{x_1, \dots, x_m\}$. Assume that the marginal distribution over these points is the uniform one (that is, each value x_i is equally likely).
- ▶ Assume that we consider binary classification, that is $\mathcal{Y} = \{\pm 1\}$, and that the labels are deterministic functions of the input.
- ▶ Particularly, there exists some function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that does not make any error.

NFL, simple version (2)

Now consider the following table:

- ▶ Rows correspond to all possible true functions (there are 2^m such functions)
- ▶ Columns correspond to all possible estimated functions
- ▶ The entries r_{ij} give the true error of function f_i when the true function is f_j .

estimated \ true					
	f_1	f_2	f_3	\dots	f_{2^m}
f_1					
f_2					
f_3					
\vdots					
f_{2^m}					

NFL, simple version (3)

Proposition 45 (Risk in each row is the same)

In each row of the table, each risk value occurs the same number of times.

Proof.

- ▶ $r_{ij} = 0$ exactly once (if $f_i = f_j$)
- ▶ $r_{ij} = 1/m$ exactly m times
- ▶ $r_{ij} = 2/m$ exactly $\binom{m}{2}$ times
- ▶ ...



NFL, simple version (4)

Proposition 46 (Simple NFL)

In the model introduced above: On average over all true functions f , the performance of all classifiers \hat{f} is the same.

Proof. Obvious consequence of the previous proposition.



NFL, simple version (5)

Proposition 47 (Simple NFL with training data)

In the model introduced above: Assume we are given a training set $(X_i, Y_i)_{i=1, \dots, n}$. Then, on average over all test distributions, all classifiers that are consistent with the training set perform the same.

Proof.

- ▶ In the table above, eliminate all columns that are not consistent with the training data.
- ▶ Among the remaining ones, all distributions over test labels are possible.
- ▶ Then the result follows by a similar argument as above. 😊

NFL, simple version (6)

Note that much more general theorems exist, for example for the standard machine learning scenario where we draw data from joint distribution P on $\mathcal{X} \times \mathcal{Y}$ and \mathcal{X} is any space you want ...

NFL, simple version (7)

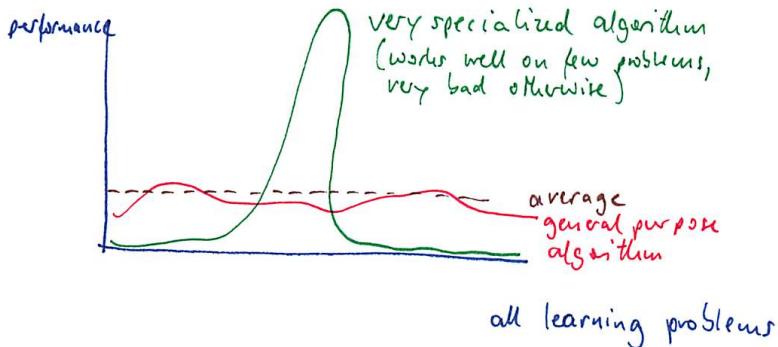
Discussion:

- ▶ Have seen: “The best possible classifier for all data sets” does not exist.
- ▶ SHOULD WE GIVE UP? IS MACHINE LEARNING MEANINGLESS?

NFL, simple version (8)

- ▶ No: the key is that in practice we do not see “all possible data sets”. As soon as we make assumptions on the data sets, the NFL breaks down (“Making assumptions” means to delete some columns from the above matrix, and then the proof breaks down).
- ▶ This shows once more how important it is to incorporate these assumptions to the machine learning algorithm \leadsto inductive bias!

NFL, simple version (9)



The integral over all these curves is exactly
the same by NFL.

History / Literature

- ▶ Wolpert, David. *The Lack of A Priori Distinctions between Learning Algorithms*. Neural Computation, 1996.
- ▶ Many generalizations since then (both to the field of machine learning and optimization in general).
- ▶ Ho and Pepyne. Simple explanation of the no-free lunch theorem and its implications. *Journal of Optimization Theory and Applications*, 2002.
- ▶ Chapter 7 in Devroye/ Györfi / Lugosi

A glimpse on modern results in learning theory

What is wrong with classic learning theory?

Why we need a new kind of learning theory

We have just scratched the surface and considered the simplest results in the “classic regime” of supervised learning.

However, this type of learning theory can NOT explain why deep networks work:

Function classes seem too large

Huge function classes: The function classes used in modern ML are huge: The language model GPT3 has of the order 10^{10} many parameters! One would expect that the capacity is huge.

Empirical observation: Deep neural networks can fit random labels. This means that their shattering coefficient is always maximal (Bengio et al, Understanding deep learning requires rethinking generalization, 2017 and 2021).

Classic SLT breaks down: Huge function classes with huge shattering coefficients leads to useless generalization bounds. This does not mean that the bounds are not correct; but the generalization bounds cannot explain why the test error becomes small (uniform convergence does not hold over the huge function class).

Common practice of overfitting seems wrong

DNNs are trained to overfit!

DNNs are trained to highly overfit the data — but still they generalize! This is all the more surprising, given that the large function classes contain many, many functions that have close to 0 training error. Many of them would not generalize to new, unseen data.

Why don't we get stuck in local optima?

Computational surprise:

DNNs induce highly non-convex optimization problems in very high-dimensional spaces. Yet it is very surprising to see that the optimisation algorithms manage to find good local (global?) optima.

Need new tools

All this does not mean that the standard learning theory is wrong. Its results simply do not shed any insights on deep learning, or we need to apply it more carefully.

We need other tools to explain DNNs.

Two sides of the story

In the following we look at two sides of the story:

(1) Why do DNNs work at all?

(2) Why we really need DNNs (or related models) to solve today's machine learning problems.

Why DNNs might work: Overfitting and the double descent curve

Literature:

Mikhail Belkin, Daniel Hsu, Siyuan Ma, Soumik Mandal: Reconciling modern machine-learning practice and the classical bias-variance trade-off. PNAS, 2019.

Mikhail Belkin: Fit without fear, remarkable mathematical phenomena of deep learning through the prism of interpolation. Acta Numerica, 2021.

Peter Bartlett, Andrea Montanari, Sasha Rakhlin: Deep learning, a statistical viewpoint. Arxiv, 2021.

Double descent curve

Radically new insight:

Need to distinguish between **underparameterized regime** (less parameters than data points, cannot interpolate, “traditional” behavior) and **overparameterized / interpolation regime**. Here the test risk sometimes decreases dramatically again and sometimes even gets lower than in the traditional regime.

Double descent curve (2)

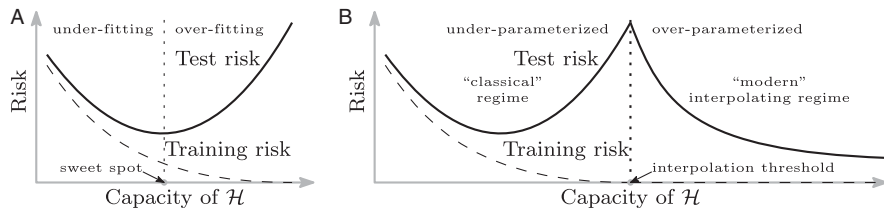


Fig. 1. Curves for training risk (dashed line) and test risk (solid line). (A) The classical U-shaped risk curve arising from the bias-variance trade-off. (B) The double-descent risk curve, which incorporates the U-shaped risk curve (i.e., the “classical” regime) together with the observed behavior from using high-capacity function classes (i.e., the “modern” interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Figure from Belkin et al, 2019, PNAS

In their original paper, such behavior has been established for a range of algorithms including neural networks and random forests:

Double descent curve (3)

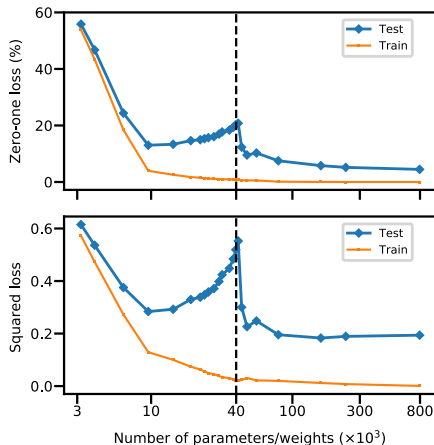


Figure 4: **Double descent risk curve for fully connected neural network on MNIST.** Training and test risks of network with a single layer of H hidden units, learned on a subset of MNIST ($n = 4 \cdot 10^3$, $d = 784$, $K = 10$ classes). The number of parameters is $(d+1) \cdot H + (H+1) \cdot K$. The interpolation threshold (black dotted line) is observed at $n \cdot K$.

Double descent curve (4)

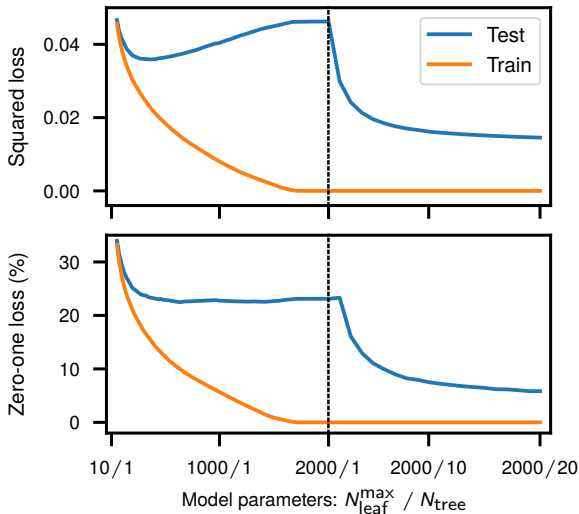


Figure 5: **Double descent risk curve for random forests on MNIST.** The double descent risk curve is observed for random forests with increasing model complexity trained on a subset of MNIST ($n = 10^4$, 10 classes). Its complexity is controlled by the number of trees N_{tree} and the maximum number of leaves allowed for each tree $N_{\text{leaf}}^{\text{max}}$.

Explaining double descent: Implicit regularization

Consider the overparameterized regime. There might exist many functions that interpolate the data perfectly. Some of them would generalize well, some of them would be awful.

Why is it the case that in practice the functions generalize so well?

Explaining double descent: Implicit regularization (2)

Theorem (Bartlett, Montanari, Rakhlin, informal) Consider a set of n data points in \mathbb{R}^d with real-valued labels. Assume the overparameterized regime: $d > n$. Choose the L_2 loss function and consider the linear regression problem. (Then there are many solutions of the interpolation problem, they fill a space of dimension $d - n$). Then the solution that is achieved by gradient descent converges to one specific solution in this space of all solutions, namely the minimum norm solution.

A similar result holds for classification with the logistic loss, where gradient descent converges to the maximum ℓ_2 -margin separator.

Similar results for specific, simple neural networks (eg fully connected, linear activation).

Explaining double descent: Implicit regularization (3)

Such behavior is called implicit regularization: Even though we do not actively model the fact that we would like to find a “simple” (nice, smooth, sparse, ...) solution, this happens implicitly through the choice of the optimization algorithm.

Bartlett et al: “In overparameterized problems that admit multiple minimizers of the empirical objective, the choice of the optimization method and the choice of the parameterization both play crucial roles in selecting a minimizer with certain properties.”

Explaining double descent: Implicit regularization (4)

One can then prove that **under certain conditions, the minimum norm interpolators are consistent** (they converge to the optimal solution). For example, the following decomposition has been used:

We write the prediction rule in the form

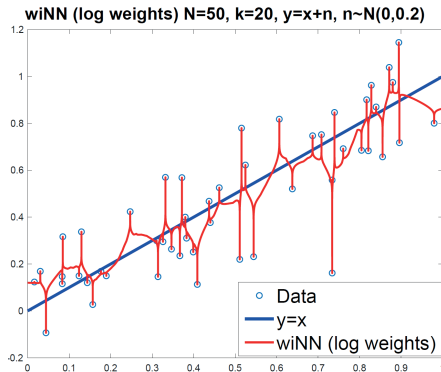
$$f = f_0 + \Delta$$

where f_0 is benign and takes care of generalization, and Δ is a very spiky function that takes care of the interpolation, but otherwise “does not hurt”.

Bartlett, Montanari, Rakhlin, 2021

Explaining double descent: Implicit regularization (5)

Here is a 1-dim illustration of this decomposition for a weighted nearest neighbor rule (taken from Belkin 2021):



Explaining double descent: Implicit regularization (6)

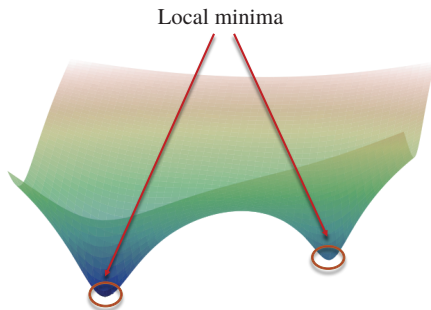
This is particularly helpful in high-dimensional settings where the volume of the “spikes” is so low that it does not hurt generalization.

Explaining double descent: Implicit regularization (7)

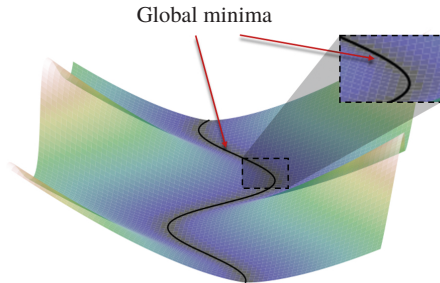
Many more results about benign overfitting out there in the literature ...

Why does SGD often find a global optimum?

We used to think that non-convex problems are hard to solve, and GD / SGD would get stuck in local optima. But:



(a) under-parametrized models



(b) over-parametrized models

Why does SGD often find a global optimum? (2)

One would expect non-convex problem to look as the ones on the left side: lots of local optima, so that depending on our starting point we end up in different local minima.

However, the landscape in the overparameterized regime seems to be rather as on the right side: No matter where we start, ultimately we always end in a global minimum (and there are many of them).

This can be characterized and proved, see Belkin 2021 and references therein.

Why is there no contradiction to classic SLT?

Literature: On Uniform Convergence and Low-Norm Interpolation Learning, Lijia Zhou, Danica J. Sutherland, Nathan Srebro, NeurIPS 2020

Hang on ...

The current story goes as follows:

- ▶ Function classes in deep learning are so large that classic generalization bounds are not informative.
- ▶ Fine, because generalization bounds themselves provide sufficient conditions for uniform convergence. So violating the sufficient conditions might not yet mean that convergence cannot take place.
- ▶ However, there was Vapnik's theorem: uniform convergence is necessary and sufficient for consistency.
- ▶ But the function classes in deep learning are still so large that uniform convergence seems difficult to obtain ...

So, yet again, where is the catch?

The catch

- ▶ Even though the function class that in principle can be used by a DNN is huge, in the end we select our solution from a tiny subset of this class. For example, we know that the selected functions have a small norm. And the set of small norm functions might again be small enough such that uniform convergence might hold.
- ▶ As it turns out, this still does not easily work out: Srebro et al (2020) prove that the set of functions of small norm is still too large for uniform convergence: “uniformly bounding the difference between empirical and population errors cannot show any learning in the norm ball, and cannot show consistency for any set, even one depending on the exact algorithm and distribution.”

The catch (2)

- ▶ However, one can get uniform convergence if one further restricts the function space to those functions that have a small norm AND are interpolating functions ...

See On Uniform Convergence and Low-Norm Interpolation Learning, Lijia Zhou, Danica J. Sutherland, Nathan Srebro, NeurIPS 2020.

Why we might really need large models: Smooth interpolation

Literature: A universal law of robustness via isoperimetry. Sebastien Bubeck, Mark Selke, NeurIPS 2021.

Smooth interpolation: intuition

Robustness: To achieve a robust solution, we would like to have a classifier that is “not too wiggly”. This is measured in terms of the Lipschitz constant:

$$L(f) := \max_{x,y} \frac{|f(x) - f(y)|}{\|x - y\|}$$

The higher $L(f)$, the more wiggly is the function, the less robust. Ideally, we would like to achieve a constant Lipschitz constant (not depending on n or d for example).

Smooth interpolation: intuition (2)

Interpolation: To solve a system of n equations, you typically need only n unknowns. However, the solution can have very high Lipschitz constants, so it is not robust.

Is there a way to achieve robust interpolation? (an interpolating classifier that has low Lipschitz constant)? If we use a larger function class, we might have more solutions to our interpolation problem. Is there one that has a low Lipschitz constant?

Empirically it has been observed that larger networks help tremendously for robustness.

Result by Bubeck and Selke

Theorem in words: Consider any function class \mathcal{F} that is smoothly parameterized by p parameters, and a d -dim data set that satisfies mild regularity assumptions (comes from a somewhat nice distribution). Then any (!) function in \mathcal{F} that fits the training data below the noise model (e.g. it interpolates) must (!) have Lipschitz constant larger than $\sqrt{nd/p}$.

The other way round, if we want to achieve Lipschitz constant $L = 1$, then we need to choose the number of parameters p at least as large as nd .

Result by Bubeck and Selke (2)

Intuition: constructing an example

- ▶ Consider the uniform distribution on the unit sphere in \mathbb{R}^d .
- ▶ Randomly sample n points (n moderate with respect to d)
- ▶ Then with probability at least $1 - \exp(-\Omega(d))$, the distance between any two sample points is at least 1.
(concentration argument due to high dimension).
- ▶ Now choose any labels for the points that you like.
- ▶ Then we can construct a function f on the whole sphere that takes the values of the labels: let g be something like a “bump” (some RBF function), and set
$$f(x) = \sum_{i=1}^n g(\|x - x_i\|)y_i.$$
- ▶ This function interpolates, needs $d \cdot n$ many parameters (the centers of the n bumps) and has Lipschitz constant 1.

Result by Bubeck and Selke (3)

The proof strategy for the main theorem in case of finite \mathcal{F}

- ▶ Consider \mathcal{F} with N functions with Lip constant L , fix some $f \in \mathcal{F}$.
- ▶ Draw a random data set of n points from the underlying distribution (with random labels).
- ▶ due to concentration (the “niceness assumption” on the high-dim distribution): The probability that the fixed f can fit the labels is bounded by

$$\exp(-nd/L^2)$$

Result by Bubeck and Selke (4)

- By a union bound: the prob that there exists some $f \in \mathcal{F}$ fits the sample is at most

$$N \exp(-nd/L^2) = \exp(\log N - nd/L^2)$$

This bound only gets large if L is on the order $\sqrt{nd/\log(N)}$. (The other way round: any function that interpolates the training data must have L of the order $\sqrt{nd/\log(N)}$ or larger.)

- A standard argument involving ε -nets extends the argument to infinite function spaces. As it turns out, the $\log N$ is then replaced by p .

The results can also be extended towards generalization bounds, see the paper for details.

Result by Bubeck and Selke (5)

Applied to ImageNet, the authors estimate that for a robust estimator, one would need of the order 10^{11} many parameters (current models use about 10^9 parameters).

If this conjecture is correct, it is only a matter of time until we achieve robust classification on ImageNet. We would not need new tools, just larger models.

Summary of “modern” learning theory

To explain the generalization capability of DNNs, we use the following chain of arguments:

- ▶ Consider the highly overparameterized regime, high-dimensional setting.
- ▶ By the geometry of the loss landscape, we can find global optima (interpolating solutions) easily through SGD.
- ▶ Under certain conditions, through implicit regularization, this interpolating solution has nice properties (e.g. it is the minimum norm solution)
- ▶ For such nice solutions, one can prove for certain models that we achieve generalization (e.g. through the “simple-plus-spiky” decomposition).

Summary of “modern” learning theory (2)

This explains why DNNs might work at all. It does not yet explain whether we actually need DNNs (or other huge models) to solve modern ML problems. Could it be the case that we just haven't found a simpler approach?

Perhaps, the answer is no. The arguments on smooth interpolations show that if we want to achieve robust solutions, then we need large models (not necessarily DNNs, but models with many parameters).

Summary of “modern” learning theory (3)

Many of the results above (and many others as well) have only been established in special cases, under special assumptions, etc. But they start painting a picture. Much more work required to pin it down ...

Machine Learning in the context of society

The general debate

Fairness

Use of energy

Explainability

The issues with ML

Literature:

see the literature lists of the two seminars on AI and ethics (summer 2019 and winter 2018/19, links on my webpage under “past lectures”.)

Foreword: AI vs ML

In the public debate, everybody uses the term Artificial Intelligence (AI) rather than Machine Learning (ML)...

Personally, I find it important to use the term “Machine Learning”: it does not implicitly suggest that there is any intelligence involved.

ML: potential “good” and “bad” uses

There is a lot of discussion in which social contexts ML might be acceptable and in which contexts ML is undesired. Here are some keywords:

Problematic uses of ML (most people in Germany would find this undesirable):

- ▶ Fake videos and fake news
- ▶ Profiling and filter bubbles
- ▶ Social Scoring
- ▶ Predictive Policing
- ▶ Compas system for bail decisions
- ▶ Automatic weapons

WHAT ARE THE ISSUES?

ML: potential “good” and “bad” uses (2)

Too extreme? Consider the following systems. Used in many places, but many people have reservations against it (WHY?)

- ▶ College admission
- ▶ Automatic screening of job applications
- ▶ Credit scoring

ML: potential “good” and “bad” uses (3)

A more technical product:

Self-driving cars (WHAT MIGHT BE ISSUES?)

ML: potential “good” and “bad” uses (4)

Good uses of ML:

- ▶ Medical applications (e.g., app for skin cancer detection that can be used in remote areas)
- ▶ Speech synthesis, machine translation: consider you are in a foreign country; you talk in your mother tongue to your phone, which translates it and pronounces it in the target language of the country.
- ▶ Applications in science
- ▶ Optimization of processes (eg forecasting of energy production and consumption to achieve better distribution)

WHAT MAKES THESE APPLICATIONS “BETTER” THAN THE PREVIOUS ONES?

ML: potential “good” and “bad” uses (5)

Another consideration: AI uses a lot of energy! Can we afford this, given the climate crisis?

ML: potential “good” and “bad” uses (6)

And last but not least:

super-intelligence... a topic that raises many emotions and speculations as well.

ML reasearch: a critical debate

There is an ongoing debate about the responsibility of researchers in AI. Here are some typical questions you might get asked:

- ▶ How can you still justify to work on AI, given that it can be used for XXX (where XXX might be one of the things mentioned above).
- ▶ How can you still justify to work on AI, given the climate crisis?
- ▶ Can we still control the AI research, or does/did/will it get out of control? (\leadsto super-intelligence)

ML reasearch: a critical debate (2)

In particular, in Tübingen you might additionally get asked or encounter statements such as:

- ▶ Do you think it is legitimate to have a position in public research and at the same time work for an industry company (e.g. Amazon)?
- ▶ ML research is governed by the interetests of industry (Cyber Valley), free reseearch does not exist any more.

ML reasearch: a critical debate (3)

It is REALLY REALLY IMPORTANT that all of you get familiar with that debate, and also that you form YOUR OWN opinion.

- ▶ In the following slides, I give you some **keywords** that are important in this debate.
- ▶ These slides are meant to highlight some of the important questions, issues, topics. Covering the debate on each of these topics would take a few lectures, so take it as a teaser. If you want to know more, please check out the literature that we had in the past ethics and society seminars (on my webpage under “past teaching”).
- ▶ **My slides don't give any answers!!! There often are no simple answers!!! And your answers might be different from my answers.**

Algorithm ethics

- ▶ For which applications is it ok to use algorithms, for which ones it it not ok?
Consider the example of information filtering such as detecting the sexual orientation of a person from images.
- ▶ Can we counteract biases in algorithms (e.g., “Man is to Computer Programmer as Woman is to Homemaker”)?
- ▶ How can we realize the transparency of decisions taken by algorithms? (right for explanations)
- ▶ Who is responsible for decisions taken by algorithms? Who is liable?

Moral machines

If machines (potentially robots) actively take decisions that influence our lives, is it possible to embed some “moral values” in these machines? If yes, how?

- ▶ Extreme example for self driving cars: if “the car” encounters a critical traffic situation where an accident is unavoidable, should it “decide” to kill three old ladies or one kid?

This is a special instance of all kind of “trolley problems” in philosophy and ethics.

ML: potential consequences for economy and public welfare

- ▶ ML might make many people lose their jobs: truck drivers, workers in call centers, ... but potentially also medical assistants or lawyers
- ▶ How many, how fast?
- ▶ Will their jobs be replaced by “other jobs”?
- ▶ In the long run, if nobody is going to work, who is going to pay taxes?

Philosophy of science and technology

determinism

The question is:

- ▶ Can we “control” AI research?
- ▶ Could we (in principle?) stop AI research if we realize it gets “dangerous” (eg building super-human robots)?

AI in the future: Utopia or dystopia?

Utopia:

- ▶ In the (far?) future, intelligent robots will do all the work for us. We all get paid unconditional basic income and can spend our time with the things we really like to do.

Dystopia:

- ▶ Intelligent robots are going to take over the world. Humans might even end up being their slaves or wiped out.
- ▶ Even if robots don't take over the world, we lose our meaning of life because they are better in everything (even in arts or music)

Regulations

Can we regulate the use of AI? If yes, how would a good regulation look like, on which level would it be realizable (Germany? EU? world wide?)

Examples:

- ▶ The new European General Data Protection Regulation (GDPR), and explainability
- ▶ San Francisco bans use of face recognition systems for its police and other agencies (2019)

Why are YOU concerned

AI is going to change the world.

You will get asked what your part in this is, and what your responsibility is, and how you take on this responsibility.

This is the case in Tübingen, but also in the rest of the world.

This public debate is REALLY IMPORTANT. And it is also important that you form your own opinion in this debate.

Fairness

Literature:

- ▶ Book draft by Solon Barocas, Moritz Hardt, Arvind Narayanan: Fairness and machine learning. Available online: <http://www.fairmlbook.org>

This book also contains many other references to literature.

Why can ML be unfair?

Example: Credit scoring

Machine learning systems are used to predict whether applicants are going to pay back or default a credit.

Say we just build a linear model and explore which features of a person are important / highly correlated with the decision of that system.

It turns out that that the ZIP code of your current address is a very strong predictor for the "yes/no" decision. So just because you live in a low-income neighborhood, you have a harder time getting the credit.

In particular, you probably need to "compensate" your bad ZIP code by, say, a higher income.

Example: College admission

Harvard Admissions Lawsuit: 2014/15, students complained that Harvard admission rules are unfair:

Race plays a significant role in admissions decisions. Consider the example of an Asian-American applicant who is male, is not disadvantaged, and has other characteristics that result in a 25% chance of admission. Simply changing the race of the applicant to white — and leaving all his other characteristics the same — would increase his chance of admission to 36%. Changing his race to Hispanic (and leaving all other characteristics the same) would increase his chance of admission to 77%. Changing his race to African-American (again, leaving all other characteristics the same) would increase his chance of admission to 95%.

Example: College admission (2)

More formally, the argument is:

$$\begin{aligned} &P(\text{accepted} \mid \text{features} = f, \text{race} = \text{asian-american}) \\ \ll &P(\text{accepted} \mid \text{features} = f, \text{race} = \text{white}) \\ \ll &P(\text{accepted} \mid \text{features} = f, \text{race} = \text{hispanic}) \\ \ll &P(\text{accepted} \mid \text{features} = f, \text{race} = \text{african-american}) \end{aligned}$$

Is it obviously unfair? Can it be fair?

What could be the reason for the behavior of the system?

Example: College admission (3)

Affirmative action!

According to Harvard: "If Harvard stopped taking race into consideration as one factor in its admissions process and adopted the race-neutral alternatives that SFFA suggested, the result would be a class that fails to achieve the diversity and excellence that Harvard seeks. "

(SFFA = Students for Fair Admission)

Example: College admission (4)

Oct 2019: judges concluded: what Harvard does is lawful. Here are some quotes (taken from the Harvard webpage, potentially biased):

- ▶ Consistent with what is required by Supreme Court precedent, Harvard has demonstrated that it uses race as a factor that can act as a “plus” or a “tip” in making admissions decisions.
- ▶ Harvard has demonstrated that there are no workable and available race-neutral alternatives that would allow it to achieve an adequately diverse student body while still perpetuating its standards for academic and other forms of excellence.
- ▶ Based on the Court’s preferred model there is not a statistically significant difference between the chances of admission for similarly situated Asian American and white applicants.

Sources:

<https://admissionscase.harvard.edu/>

Fair ML book, Sec. on “Counterfactual discrimination analysis”

Example: Deliveries

Amazon deliveries:

Amazon uses a data-driven system to determine the neighborhoods in which to offer free same-day delivery. A 2016 study found stark disparities in the demographic makeup of these neighborhoods: in many U.S. cities, white residents were more than twice as likely as black residents to live in one of the qualifying neighborhoods.

Source: Fairness book.

Example: Street bump

This is a project by the city of Boston to crowdsource data on potholes. The smartphone app automatically detects pot holes using data from the smartphone's sensors and sends the data to the city, which then fixes the streets.

The problem:

In areas with a large elderly population, people tend to have less smartphones than in other areas.

The same obviously holds for low-income neighborhoods as compared to wealthy neighborhoods.

Source: fairness book.

Example: Recrutement tools (1)

Amazon, where men hold 74 percent of the company's managerial positions, recently discontinued use of a recruiting algorithm after discovering gender bias. The data that engineers used to create the algorithm were derived from the resumes submitted to Amazon over a 10-year period, which were predominantly from white males. The algorithm was taught to recognize word patterns in the resumes, and these data were benchmarked against the company's predominantly male engineering department. As a result, the AI software penalized any resume that contained the word "women" in the text and downgraded the resumes of women who attended women's colleges.

Source: Hamilton, Isobel Asher. Why It's Totally Unsurprising That Amazon's Recruitment AI Was Biased against Women" Business Insider, October 13, 2018.

<https://www.brookings.edu/research/>

algorithmic-bias-detection-and-mitigation-best-practices-and-policies-to-reduce-consumer-harms/

Example: Recrutement tools (2)

- ▶ The employment agency in Austria is planning to use automatic tools to classify unemployed persons. The three classes specify what (according to the algorithm) are the chances of finding a new job: “low”, “middle”, “high”.
- ▶ Data: personal features such as age, education, prior jobs, ...
- ▶ The explicit idea is then to spend more effort on the top group than on the bottom group.
- ▶ The procedure is under criticism because it could transform existing discrimination into a technical solution.
- ▶ For example, it has been proven that female gender and older age lead to a worse evaluation by the algorithm.

Source: Süddeutsche Zeitung, Oktober 2019:

<https://www.sueddeutsche.de/digital/digitalisierung-arbeitslosigkeit-jobcenter-1.4178635>

Example: Word embeddings

Word embeddings are a popular tool to generate vector space representations of words (e.g., word2vec algorithm). Input consists of word co-occurrences in text corpora, output generates an embedding r of words into some vector space, say \mathbb{R}^{300} .

It has been observed that these word embeddings capture semantic structures, so you can do “word arithmetics”:

Example:

- ▶ Compute the difference $v := r(\text{“France”}) - r(\text{“Paris”})$
- ▶ Compute $r(\text{“England”}) + v$, then you obtain something close to “London”
- ▶ We say: “London is to England as Paris is to France”.

Example: Word embeddings (2)

Now you can start playing:

- ▶ Fix a word X
- ▶ Ask “man is to X as woman is to Y ” and observe Y .
- ▶ Here are some word pairs that you will find:

man = computer programmer	\implies	woman = home maker
man = surgeon	\implies	woman = nurse
man = brilliant	\implies	woman = lovely

You can see how the word embedding perfectly reproduces all the gender stereotypes that you might find in large text corpora!

Source: Bolukbasi, T., Chang, K. W., Zou, J. Y., Saligrama, V., Kalai, A. T. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. NeurIPS, 2016.

Example: Compas

- ▶ The algorithm COMPAS is used in the US nationwide to decide whether defendants awaiting trial are too dangerous to be released on bail
(German: on bail = “auf Kaution freilassen”)
- ▶ The COMPAS tool assigns defendants scores from 1 to 10 that indicate how likely they are to re-offend. The score is based on more than 100 factors, including age, sex and criminal history. Notably, race is not used.
- ▶ The higher this recidivism score, the more likely a person is considered risky and is being detained.
(German: recidivism=Rückfall)

There is a heated debate about whether this score is biased against black people:

Example: Compas (2)

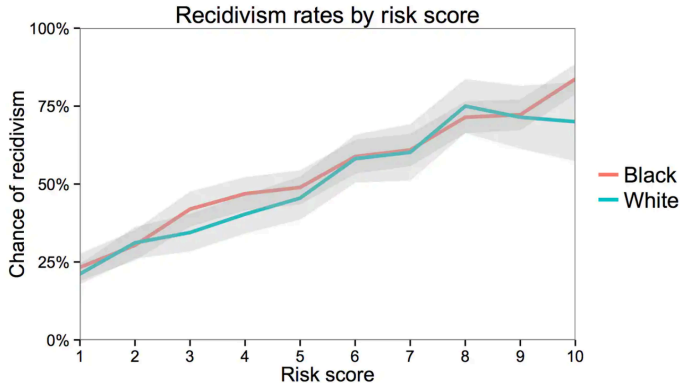
First point of view: score is not biased:

If you have a certain score s , the probability to reoffend is about the same for white and black persons:

$$\forall s = 1, \dots, 10 :$$

$$\begin{aligned} &P(\text{reoffend} | \text{score} = s; \text{race} = \text{white}) \\ &\approx P(\text{reoffend} | \text{score} = s; \text{race} = \text{black}) \end{aligned}$$

Example: Compas (3)



Recidivism rate by risk score and race. White and black defendants with the same risk score are roughly equally likely to reoffend. The gray bands show 95 percent confidence intervals.

Consequence: when judges see a defendant's risk score, they need not consider the defendant's race when interpreting it.

Example: Compas (4)

Second point of view: the scores are biased:

Among those defendants who ultimately did not reoffend, blacks were more than twice as likely as whites to be classified as medium or high risk (42 percent vs. 22 percent):

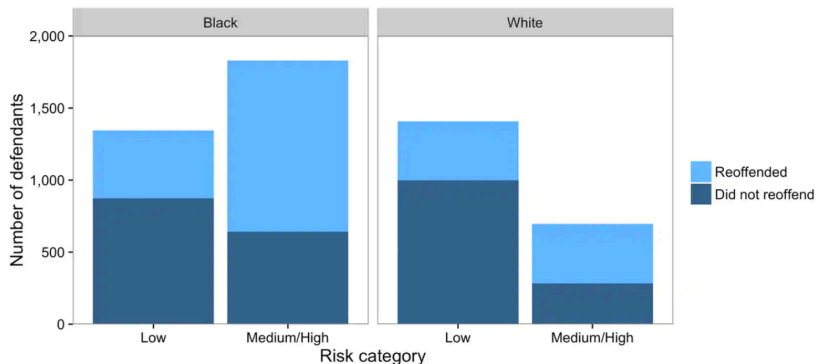
$P(\text{high risk score} | \text{not reoffend, race=black}) \gg$

$P(\text{high risk score} | \text{not reoffend; race=white})$

Even though these defendants did not go on to commit a crime, the black ones are subjected to harsher treatment by the courts than the white ones.

Example: Compas (5)

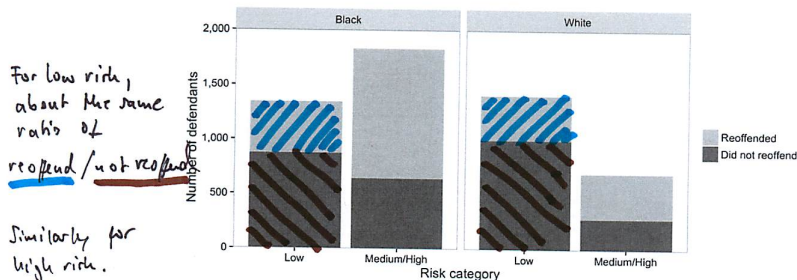
Bringing it together (see text on next slide):



Distribution of defendants across risk categories by race. Black defendants reoffended at a higher rate than whites, and accordingly, a higher proportion of black defendants are deemed medium or high risk. As a result, blacks who do not reoffend are also more likely to be classified higher risk than whites who do not reoffend.

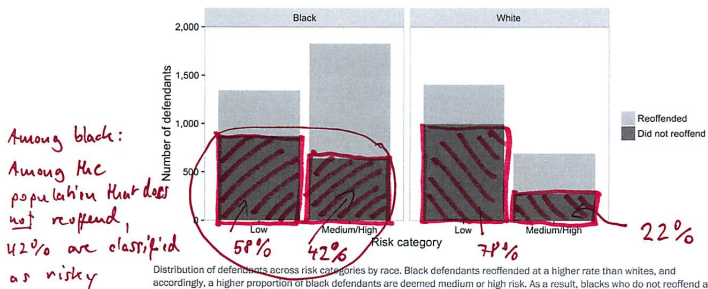
Example: Compas (6)

First point of view: the algorithm is fair: Within each risk category, the proportion of defendants who reoffend is approximately the same regardless of race (In the figure: take the "low" bar of both populations; both for white and black, this "low" bar has roughly the same fraction of light and dark blue; same goes for "medium/high" bar).



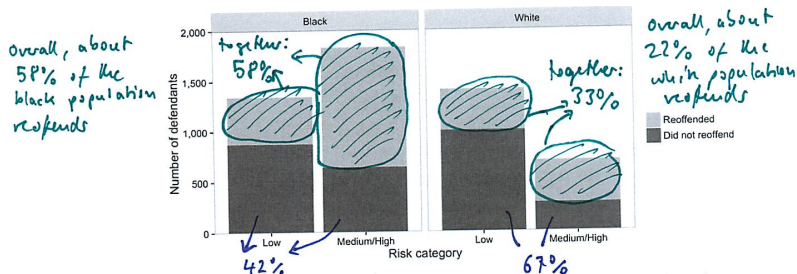
Example: Compas (7)

Second point of view: The algorithm is unfair: Black defendants who don't reoffend are predicted to be riskier than white defendants who don't reoffend. In the figure: for the black population, compare the two dark blue areas between "low" and "medium/high". They are nearly the same. This is not true for the white population.



Example: Compas (8)

And why can this happen? In the raw data, black defendants reoffend at a higher rate than whites (in the figure: the overall area of light blue vs dark blue is higher for black (58%) than for white (33%). A classifier that is perfect in terms of accuracy will be more likely to classify black defendants as medium or high risk than white defendants (58 percent vs. 33 percent).



Example: Compas (9)

So who is right? Is it fair or not?

Depends on the way we measure fairness, and also on the way we measure the “success” of such a system.

In particular, for this data it is impossible to construct non-trivial classifier that is “fair” with respect to both points of view.

Literature:

Machine bias: There is software across the country to predict future criminals. And it is biased against black. Julia Angwin, Jeff Larson, Surya Mattu and Lauren Kirchner, ProPublica, 2016

Washington Post: A computer program used for bail and sentencing decisions was labeled biased against blacks. It's actually not that clear. <https://www.washingtonpost.com/news/monkey-cage/wp/2016/10/17/can-an-algorithm-be-racist-our-analysis-is-more-cautious-than-propublicas/>

Example: Compas (10)

As a side remark, there are more obviously problematic things about the data. In particular, there might be people who have been released and also have re-offended, but were never caught in doing so. Likelihood that this might happen might be different, depending on where you live, for example. This might introduce yet more bias to the data.

Examples: First take away

- ▶ Implicitly, all kinds of biases can happen in automatic classification systems.
- ▶ In most of the systems outlined above, the ones who had designed the system did not intend to be unfair or discriminatory (e.g., street bump; word embeddings)
- ▶ In some systems, the ones who had designed it intentionally tried to promote certain minorities (affirmative action in Harvard admission).

Examples: First take away (2)

- ▶ There is no unique definition of fairness! Many definitions exist, typically they are exclusive (you cannot be fair with respect to all of them).
- ▶ Which definition of fairness is appropriate can be very different for different applications, and always needs to be discussed in the context of society.
- ▶ When we want to have a “fair” system, we might need to give up performance on other aspects, for example overall classification accuracy (e.g., compass) or utility/profit (e.g., credit scoring).
- ▶ Having “fairness” as a criterion is a request that comes from society.

Examples: First take away (3)

It is hard to come up with a real application for machine learning where we cannot spot any potential discriminatory behavior!!!

So this is an issue. Always.

And typically there is no simple solution.

But there are at least some things that we can do, see below.

Data and measurement

Sources of unfairness: minorities

- ▶ If we test for overall accuracy, a 5 % error statistical model might perform terribly for a minority group:

Minorities simply get “drowned” in systems that maximize accuracy: training and test error do not change much if we change the classification on minority groups.

- ▶ Worse, in many settings **minority groups might be underrepresented** relative to population statistics (due to sampling bias, see next slide)

Sources of unfairness: Biases in data

- **Sampling bias:** Data collection might have demographic, geographic, behavioral, temporal biases. Certain groups will be over / under-represented in the sample.

Example: The record of crimes only come from those crimes observed by police. The police department tends to dispatch more officers to the place that was found to have higher crime rate initially and is thus more likely to record crimes in such regions. Even if people in other regions have higher crime rate later, it is possible that due to less police attention, the police department still record that these regions have lower crime rate.

Sources of unfairness: Biases in data (2)

- ▶ **Pre-existing biases** in data, e.g. gender roles in text and images, racial stereotypes, ...

Example:

- ▶ past hiring data;
- ▶ word embeddings

Sources of unfairness: measurement

Problem of feature measurement: Which features do we measure, and how?

Examples:

- ▶ In the US it is common to collect statistics according to race (“African-American, Hispanic, Asian, ... ”) and even gender (“self-identified male”, “self-identified female”). The problem:
 - ▶ People might not want to answer truthfully
 - ▶ People might not even be able to answer truthfully
 - ▶ By forming these categories, they become normative: How we measure a concept changes how we think about it.

By selecting features we influence the way we model the problem.
The selection of measurements might systematically favor/ disfavor certain groups!

Sources of unfairness: measurement (2)

Problem of target measurement:

In the very end, ML algorithms try to learn a function that fits a target variable. In social contexts, the choice of the target variable is often difficult, and often only a coarse proxy of what we would like to measure. By making a choice we already encode biases:

Examples:

- Job promotion: If our target variable is the idea of a “good employee”, we might use performance review scores to quantify it. This means that our data inherits any biases present in managers' evaluations.

Sources of unfairness: measurement (3)

- ▶ Hiring: Instead of relying on performance reviews for a sales job, we might rely on the number of sales closed. But is that an objective measurement or is it subject to the biases of the potential customers (who might respond more positively to certain salespeople than others) and workplace conditions (which might be a hostile environment for some, but not others)?
- ▶ Computer vision: there are systems out there that are supposed to rank people's physical attractiveness. The training data consists of human evaluation of attractiveness, and all these classifiers showed a preference for lighter skin — simply because most of the images show white persons and most of the evaluators were white, male, young men.

Sources of unfairness: measurement (4)

- Health systems rely on commercial prediction algorithms to identify and help patients with complex health needs. We show that a widely used algorithm, affecting millions of patients, exhibits significant racial bias: At a given risk score, Black patients are considerably sicker than White patients, as evidenced by signs of uncontrolled illnesses. Remedying this disparity would increase the percentage of Black patients receiving additional help from 17.7 to 46.5%. The bias arises because the algorithm predicts health care costs rather than illness, but unequal access to care means that we spend less money caring for Black patients than for White patients. Thus, despite health care cost appearing to be an effective proxy for health by some measures of predictive accuracy, large racial biases arise. We suggest that the choice of convenient,

Sources of unfairness: measurement (5)

seemingly effective proxies for ground truth can be an important source of algorithmic bias in many contexts.

Ziad Obermeyer, Brian Powers, Christine Vogeli, Sendhil Mullainathan: Dissecting racial bias in an algorithm used to manage the health of populations. Science, Okt 2019.

Discussion

Fairness is not a technical requirement, but one that comes from society!

Nothing about fairness is obvious and simple.

Some basic notions of fairness

Very simplified setup

Data consists of

- ▶ Features $X \in \mathcal{X}$
- ▶ Protected/sensitive attributes $A \in \mathcal{A}$ (e.g., gender, race,...). Depending on the application, the protected attributes are explicitly known or not.
- ▶ True target variable $Y \in \mathcal{Y}$.

The high-level goal is to learn a classifier $C : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{Y}$ which for each individual predicts a target variable \hat{Y} (which can be an estimate of \mathcal{Y} , but also something more abstract).

In the following, for simplicity we assume the protected attribute to be binary (e.g., black/white), the true outcome is binary (“reoffends or not”), and the predicted outcome is binary (“kept in jail / released on bail”).

Very simplified setup (2)

Example 1: compas

- ▶ X = features of the person
- ▶ A = white or black
- ▶ Y = whether the person would reoffend
- ▶ \hat{Y} = whether the system says that the person should be released on bail

Very simplified setup (3)

Again, keep in mind: all this is a very poor proxy!

- ▶ the actual data X incorporates all sorts of measurement biases
- ▶ the sensitive attributes A are often not even known, might be ill-defined (gender), misreported, inferred, ...
- ▶ The classifier C is often hard to understand (e.g., a deep neural net)
- ▶ Y is often a poor proxy of the actual variable of interest

Measuring fairness

As we have seen in the examples above: there is no unique definition of fairness.

Below are some popular definitions, many more definitions exist.

Demographic parity

(called independence in fairml book)

Demographic parity:

$$P(\hat{Y} = 1 \mid A = 1) = P(\hat{Y} = 1 \mid A = 0)$$

Independently of all other features, both groups should have the same rate of success.

Examples could be:

- ▶ The same proportion of males ($A = 0$) and females ($A = 1$) should be promoted ($\hat{Y} = 1$).
- ▶ The same proportion of white ($A = 0$) and black ($A = 1$) people should be released on bail ($\hat{Y} = 1$).

Demographic parity (2)

Alternative (more general) definition that can be used beyond the discrete cases:

The two random variables \hat{Y} and A are independent:

$$\hat{Y} \perp A$$

Demographic parity (3)

The assumption that there is no dependence / correlation between target variable Y and sensitive variable A is very strong. As soon as such a correlation exists, demographic parity might be highly problematic.

Can you come up with an example?

Equalized odds

(called separation in fairml book)

All groups experience the same false positive rate and the same false negative rate:

$$P(\hat{Y} = 0|Y = 1, A = 0) = P(\hat{Y} = 0|Y = 1, A = 1)$$

and

$$P(\hat{Y} = 1|Y = 0, A = 0) = P(\hat{Y} = 1|Y = 0, A = 1)$$

(Sometimes, one cares more about one direction than the other.)

Examples:

- Among the students who have the potential to achieve a MSc degree ($Y = 1$), the likelihood to be accepted to the masters program ($\hat{Y} = 1$) should be the same for male ($A = 0$) and females ($A = 1$).

Equalized odds (2)

- ▶ Among the people who do not reoffend ($Y = 1$), the likelihood to be released on bail ($\hat{Y} = 1$) should be the same for both white ($A = 0$) and black ($A = 1$).

Equalized odds (3)

More general definition:

Conditionally on the random variable Y , the two random variables \hat{Y} and A are independent:

$$\hat{Y} \perp A \mid Y$$

Equalized odds (4)

Sounds good.

But note: the perfect classifier (with false negative and false positive rate being 0) is always perfectly fair according to this definition.

Hm...

Predictive parity

(called sufficiency in fairml book)

Predictive parity criterion:

$$P(Y = 1 | \hat{Y} = y, A = 0) = P(Y = 1 | \hat{Y} = y, A = 1).$$

If the prediction for a person is $\hat{Y} = y$, then the probability that the true value $Y = 1$ should be the same for all sensitive groups.

More generally, we can define it as $Y \perp A \mid \hat{Y}$.

Example:

Compas, the first point of view: if the compas system predicts score $\hat{y} \in \{1, \dots, 10\}$, the likelihood to actually reoffend should be the same for black and white. Then the judge, when he sees a particular score \hat{y} , can treat white and black persons the same.

Predictive parity (2)

More general notion: calibration.

Assume the score is supposed to predict a certain probability (say, the probability that a person reoffends). We say that the score \hat{Y} satisfies calibration by group if for all score values y and all groups a we have

$$P(Y = 1 | \hat{Y} = y, A = a) = y$$

In words: the probabilities are “correct”.

In simple cases, calibration and predictive parity are more or less the same, but this does not always need to be the case.

See fairml book, and the paper: Fair prediction with disparate impact: A study of bias in recidivism prediction instruments.

Alexandra Chouldechova, 2017.

Extreme cases

Consider the situation of binary classification and a binary sensitive attribute.

Extreme case 1: Constant classifier:

- ▶ $Y = 1$ for all inputs.
- ▶ The output Y is independent of anything else, in particular the sensitive attribute.
- ▶ This classifier is maximally fair. In particular, it satisfies demographic parity and equalized opportunity.

Extreme cases (2)

Extreme case 2: predicting the sensitive attribute.

- ▶ $Y = 1 \iff A = 1$.
- ▶ The output Y is identical with the sensitive attribute.
- ▶ This classifier is maximally unfair with respect to demographic parity and equalized opportunity

Extreme cases (3)

Whenever you evaluate fairness/accuracy of classifiers, use the two extreme case classifiers as baselines to see how much you can improve...

Impossibility results

There exist all kinds of statements that in non-trivial circumstances, the different criteria typically cannot hold at the same time:

- ▶ Assume that A and Y are not independent. Then demographic parity and equalized odds cannot both hold.
- ▶ Assume Y is binary, A is not independent of Y , and \hat{Y} is not independent of Y . Then, demographic parity and predictive parity cannot both hold.
- ▶ Assume that all events in the joint distribution of (A, \hat{Y}, Y) have positive probability, and assume A is not independent of Y . Then, equalized odds and predictive parity cannot both hold.

Impossibility results (2)

- Assume Y is not independent of A and assume \hat{Y} is a binary classifier with nonzero false positive rate. Then, predictive parity and equalized odds cannot both hold.

For literature and proofs, see for example the fairml book, or the following paper: Kleinberg, Jon, Sendhil Mullainathan, and Manish Raghavan. "Inherent trade-offs in the fair determination of risk scores." arXiv preprint arXiv:1609.05807 (2016).

Individual fairness

- ▶ Here we don't talk about different groups and whether they are treated in a similar way, but about individuals.
- ▶ The notion of fairness simply says that two individuals who are "similar" (according to some pre-specified metric) should receive similar treatment.
- ▶ However, in practice this is hard to use. In particular: what is the right notion of similarity, both in the input space and the target space?

Literature: Dwork, Cynthia, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. 2012.

Counterfactual fairness

- ▶ computes what (the distribution of) any of the variables would have been had certain other variables been different, other things being equal.
- ▶ For instance, given the causal model we can ask “Would individual i have graduated ($Y = 1$) if they hadn't had a job?”

Literature: Russell, C., Kusner, M. J., Loftus, J., Silva, R. When Worlds Collide: Integrating Different Counterfactual Assumptions in Fairness. NeurIPS 2017.

Lots of criteria

Many criteria for fairness exist and have been invented several times (and many more are not in this table):

Name	Closest relative	Note	Reference
Statistical parity	Independence	Equivalent	Dwork et al. (2011)
Group fairness	Independence	Equivalent	
Demographic parity	Independence	Equivalent	
Conditional statistical parity	Independence	Relaxation	Corbett-Davies et al. (2017)
Darlington criterion (4)	Independence	Equivalent	Darlington (1971)
Equal opportunity	Separation	Relaxation	Hardt, Price, Srebro (2016)
Equalized odds	Separation	Equivalent	Hardt, Price, Srebro (2016)
Conditional procedure accuracy	Separation	Equivalent	Berk et al. (2017)
Avoiding disparate mistreatment	Separation	Equivalent	Zafar et al. (2017)
Balance for the negative class	Separation	Relaxation	Kleinberg, Mullainathan, Raghavan (2016)
Balance for the positive class	Separation	Relaxation	
Predictive equality	Separation	Relaxation	Chouldechova (2016)
Equalized correlations	Separation	Relaxation	Woodworth (2017)
Darlington criterion (3)	Separation	Relaxation	Darlington (1971)
Cleary model	Sufficiency	Equivalent	Cleary (1966)
Conditional use accuracy	Sufficiency	Equivalent	Berk et al. (2017)
Predictive parity	Sufficiency	Relaxation	Chouldechova (2016)
Calibration within groups	Sufficiency	Equivalent	Chouldechova (2016)
Darlington criterion (1), (2)	Sufficiency	Relaxation	Darlington (1971)

Source: fairml book

Lots of criteria (2)

- ▶ **The one, unique fairness criterion does not exist.**
- ▶ Fairness is a concept that comes from society, and cannot always be captured in a satisfactory manner by statistical definitions.
- ▶ While being plausible in some applications, all of the existing criteria have serious drawbacks and fail to capture important aspects of the problem. See the fairml book for a discussion.
- ▶ But also note: the baseline is decisions made by humans, and they are definitely biased as well.

Lots of criteria (3)

One aspect that I find interesting:

- ▶ The different definitions of fairness set the scene for a discussion that needs to be conducted in society.
- ▶ This discussion is obviously necessary regarding the use of algorithms.
- ▶ But one can also extend this discussion to examine / quantify the bias of human decisions (e.g., courts).

Technical approaches to improve fairness

Three main approaches

There are three high-level ideas to improve unfair classifiers:

- ▶ Preprocessing: Try to fix the bias in the data
- ▶ Training: Train algorithms to learn decisions that are accurate and fair at the same time
- ▶ Postprocessing: Try to fix an unfair blackbox model in hindsight

Fix unfairness in data

First naive idea to fix some of the unfairness: remove sensitive features from data.

However, this is pretty much impossible!

The obvious problem: many other variables are highly correlated with the sensitive attribute. So the ML algorithm can focus on those as a proxy:

Fix unfairness in data (2)

Standard example:

- ▶ In many cities, there exist quarters that are predominantly inhabited by white resp. black people. And often, black people have a low income. Hence, then the zip code of their neighborhood is both highly correlated with income and with race. Consequently, it might be harder to get a credit if you live in one of those quarters.
- ▶ So even if the race is not used explicitly as a feature in a classifier, it is implicitly present whenever the ZIP code is being used. The same is true for many other variables.

Fix unfairness in data (3)

In many cases, the discriminatory features are not even well-defined:

Some patterns in the training data (smoking is associated with cancer) represent knowledge that we wish to mine using machine learning, while other patterns (girls like pink and boys like blue) represent stereotypes that we might wish to avoid learning.

It is hard (impossible) to tell the algorithms which patterns it is supposed to find. In fact, we might not even have a unique opinion among humans...

Fix unfairness in data (4)

Of course you could **try to remove all variables that are correlated with the sensitive attribute**. But:

- ▶ In many cases, this will remove all the relevant variables.
- ▶ This is in particular true if you have more than one sensitive variable (e.g., gender and race)
- ▶ And other cases, you might not have access to the sensitive attribute, so you cannot even compute the correlation in advance.

Fix unfairness in data (5)

Another approach: **remove feature representation altogether** and use prototypes. Represent each data point x as a weighted linear combination of K prototypes to satisfy demographic parity, and keep original information and accuracy as much as possible.

Won't work in most cases ...

Literature: Richard Zemel, Yu Wu, Kevin Swersky, Toniann Pitassi: Learning Fair Representations. ICML 2013

Train for accuracy and fairness

Consider a standard supervised machine learning setup: you want to learn a function f that minimizes the empirical risk $\hat{R}(f)$ (with respect to some loss function ℓ , over some function space \mathcal{F}). You might use a regularizer $\Omega(f)$ to prevent overfitting.

Additionally, we now consider the fairness of the classifier. Assume your data has two sensitive groups, $A = 1$ and $A = 0$, and assume we are after demographic parity.

Define the true unfairness as the difference in demographic parity:

$$\text{unf}(f) = P(f(x) > 0 \mid A = 1) - P(f(x) > 0 \mid A = 0)$$

Train for accuracy and fairness (2)

and its empirical version as

$$\widehat{\text{unf}}(f) = \frac{1}{n} \sum_{i|a_i=1} \mathbb{1}_{f(x_i)>0} - \sum_{i|a_i=0} \mathbb{1}_{f(x_i)>0}$$

Now we pose the following optimization problem:

$$\begin{aligned} \min_{f \in \mathcal{F}} R_n(f) + \lambda \Omega(f) \\ \text{subject to } \widehat{\text{unf}}(f) < \tau \end{aligned}$$

WHAT DO YOU THINK, IS IT EASY? DIFFICULT?

Train for accuracy and fairness (3)

- ▶ Obvious problem: unf is discrete, so it is hard to optimize.
- ▶ Standard solution: use convex relaxations. However, we found: Most existing relaxations are too loose! Even if the “relaxed fairness” is perfectly satisfied, the true fairness can be very bad.
- ▶ We consider some first approaches that come with guarantees for fairness, see our paper cited below if you are interested. Definitely not the end of the story yet.

Bottom line: I believe that this approach is the most useful one, but currently there is nothing out there that really works well.

Literature:

- M. B. Zafar, I. Valera, M. G. Rodriguez, and K. P. Gummadi. Fairness Constraints: Mechanisms for Fair Classification. WWW 2017, ICML 2017.
- M. Lohaus, M. Perrot, U. von Luxburg. Too Relaxed to Be Fair. 2020.

Fix models in hindsight

- ▶ Consider a scenario where an agency / company trains a classifier on data, but the way it trains the classifier is kept private (e.g., credit assessment; compas score).
- ▶ Now assume somebody evaluates the results of the classifier and finds it to be unfair.
- ▶ Could we “fix it” in hindsight, without getting access to the internal workings of the classifier?

What are our options?

- ▶ We only get access to the prediction \hat{Y} of the classifier, and the sensitive attribute A (which we need to evaluate the classifier).
- ▶ All we can do is now to construct a “derived classifier” that takes \hat{Y} and A as input and produces a new output \tilde{Y} (potentially, randomized output).

Fix models in hindsight (2)

If A and \hat{Y} are binary, our derived classifier can set exactly four parameters:

$$p_{ya} = P(\tilde{Y} = 1 \mid \hat{Y} = y, A = a) \quad , \text{ for } y \in \{0, 1\}, a \in \{0, 1\}$$

We can now construct a post-processed solution that satisfies equalized opportunity, see the assignment this week. This leads to a randomized classifier (randomization here helps to get a better accuracy-fairness tradeoff).

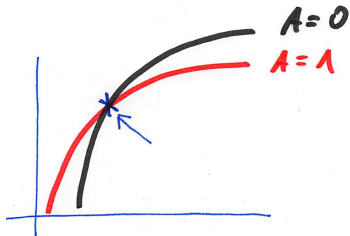
But obviously, the accuracy of the classifier might decrease when we apply this linear program. The original paper gives some guarantees on this

Fix models in hindsight (3)

Assume we not only get access to the binary label \hat{Y} , but to a real valued score $f(X)$ that we would threshold at some value t :

$$\hat{Y} = 1 \iff f(X) \geq t$$

Then we could plot the the ROC-curves of both groups separately and choose the classifier where both curves intersect. Then for both groups, false positive and false negative rates are equal:



Fix models in hindsight (4)

If the curves do not intersect, there are still ways to solve the problem using a randomized predictor. See the paper below for more details.

Literature:

M. Hardt, E. Price, and N. Srebro. Equality of opportunity in supervised learning. NeurIPS, 2016.

Tradeoffs

Tradeoff accuracy / fairness

- ▶ “Fair” classifiers try to optimize two objectives: fairness and accuracy.
- ▶ Obviously, the accuracy of the fair classifier can only be \leq then one of the one that is optimized for accuracy alone.
- ▶ A choice has to be made (how much accuracy loss do we tolerate for how much fairness?)

Fair classifiers can require randomization

In particular, if classifiers are modified in hindsight, it is often impossible to achieve classifiers without resorting to randomization.

Depending on the application, randomization is highly questionable (a randomized decision who is going to stay in jail?!?)

Agarwal, A., Beygelzimer, A., Dudík, M., Langford, J., Wallach, H. (2018). A reductions approach to fair classification. M. Hardt, E. Price, and N. Srebro. Equality of opportunity in supervised learning. NeurIPS, 2016.

Feedback loops

We need to be careful of feedback loops when employing ML tools:

Self-fulfilling predictions:

- ▶ Suppose a predictive policing system determines certain areas of a city to be at high risk for crime.
- ▶ More police officers might be deployed to such areas, hence more crimes might be detected in these areas.
- ▶ The prediction that the areas are risky will appear to be validated (even if the area does not have an increased crime risk).

Feedback loops (2)

Predictions that affect the training set:

- ▶ predictive policing activity will leads to arrests, records of which might be added to the algorithm's training set.
- ▶ These areas might then continue to appear to be at high risk of crime.

Literature: Ensign, D., Friedler, S. A., Neville, S., Scheidegger, C., Venkatasubramanian, S. (2017). Runaway feedback loops in predictive policing. [arXiv:1706.09847](https://arxiv.org/abs/1706.09847).

Discussion

- ▶ Fairness is a topic that has a long history of debate, in ethics, sociology, and many other fields.
- ▶ Bottom line, both from looking at data and from considering the theoretical negative results: there often is no obvious “solution”.
- ▶ In practice, there are many different decisions to take (which notion of fairness, which tradeoff is acceptable, is randomization acceptable, etc). Different decisions lead to different solutions.
- ▶ Most of these issues cannot be fixed by technical solutions. Society has to decide! (But we need to help them and explain the issues).

See also: Sam Corbett-Davies, Sharad Goel: The Measure and Mismeasure of Fairness: A Critical Review of Fair Machine Learning. Arxiv, 2018

Explainability

Literature: cited on the slides directly.

Explanations of algorithmic decisions in general

... are important in many contexts:

- ▶ Example: Medical context
- ▶ Example: Legal context (in the EU Legislation):
 - ▶ General Data Protection Regulation (2018):
"In cases of automated decision making, the controller [=the one who runs the algorithm] shall provide the subject [the person on whom a decision is taken] with meaningful information about the logic involved"
 - ▶ Draft of the AI Act (2021):
High-risk AI systems have to be "designed and developed in such a way to ensure that their operation is sufficiently transparent to enable users to interpret the system's output and use it appropriately".

Explanations of algorithmic decisions in general

(2)

Goal of explanations might be: establish understanding, trust, sanity checks, possibility to object, ...

Two types of explanations

Interpretable ML algorithm / ML model:

The model itself is so simple and transparent that a human can understand what the model does.

Examples:

- ▶ Simple decision trees
- ▶ Additive models
- ▶ Linear models (to a certain extent)

Often, explanations then highlight the influence of particular features on the decision. For example, for a credit decision the explanation might be: “the most important feature might be the income, followed by the age”.

Two types of explanations (2)

Local post-hoc explanations:

- ▶ The model itself is so complex that it cannot be understood as a whole (e.g. a DNN).
- ▶ Instead we would like to receive an explanation why the model made certain individual decisions.
- ▶ To this end, we invoke yet another algorithm, the explanation algorithm.
- ▶ The explanation algorithm gets the model function and a query point as input. It tries to justify “post hoc” the decision of the model at this particular, “local” query point.

Two types of explanations (3)

- ▶ Examples:
 - “Mr. Schmidt did not receive the loan because his income was too low. ”
 - “Ms. Baker gets a credit because she had paid back her previous credit.”
- ▶ In particular, explanations can differ between different points.

Local post-hoc explanations: LIME on tabular data

LIME = Local Interpretable Model-agnostic Explanations.

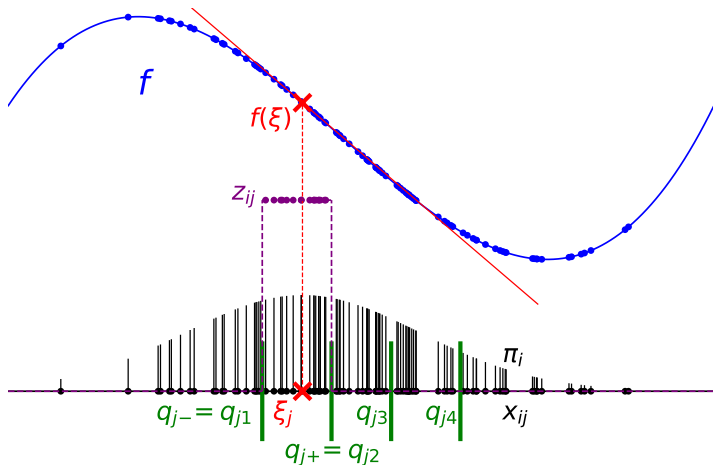
- ▶ Assume we are in a high-dim Euclidean feature space. We have learned a complex function (e.g, SVM with Gaussian kernel; neural network).
- ▶ We obviously cannot globally replace the complicated function by a simple function that is based on very few “explainable” features and use this interpretable model instead (WHY?)
- ▶ But we might be able to locally approximate the complicated function by a simple linear function, to understand the decision at a particular point x .
- ▶ The hope is that the explanation that is provided by such a local approximation might be something meaningful.

Local post-hoc explanations: LIME on tabular data (2)

How LIME works for tabular ($=\mathbb{R}^d$) data, very rough sketch:

- ▶ Fix a point x for which you want to explain the decision of a complicated function f
- ▶ Sample points x_1, \dots, x_m locally around x and evaluate the function $f(x_1), \dots, f(x_m)$
- ▶ Locally approximate the complicated decision function f by a simple linear function.
- ▶ Identify the few most prominent coordinates in this linear model and use those as explanations.
- ▶ Important: we don't need to know anything about the learning algorithm that generated f . LIME can be applied to any black box algorithm as soon as we can evaluate the model f for arbitrary input points.

Local post-hoc explanations: LIME on tabular data (3)



Given a specific datapoint x (in red), we want to build a local model for f (in blue), given new samples x_1, \dots, x_n (in black).

Local post-hoc explanations: LIME on tabular data (4)

Conceptually, it is a bit doubtful what the intuitive meaning of the highest coefficients of the local linear fit really is.

In an attempt to improve understanding, we could prove:

LIME identifies those coordinates that contribute most to the gradient of f at x .

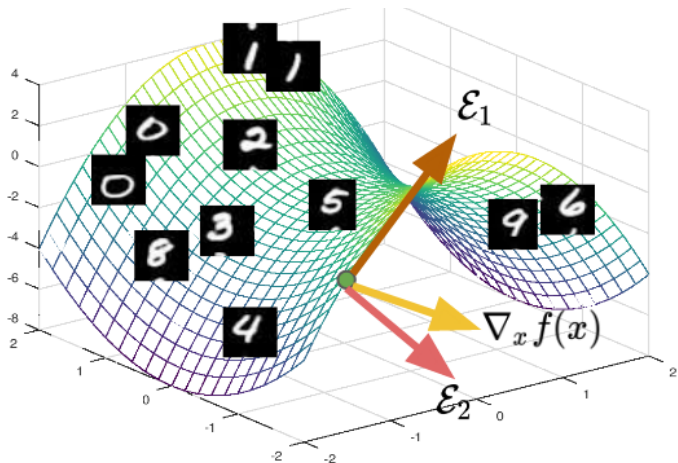
Intuitively: LIME identifies those features such that if we would change those features, we would most easily get to a different output of f (Garreau, Luxburg 2020).

Local post-hoc explanations: LIME on tabular data (5)

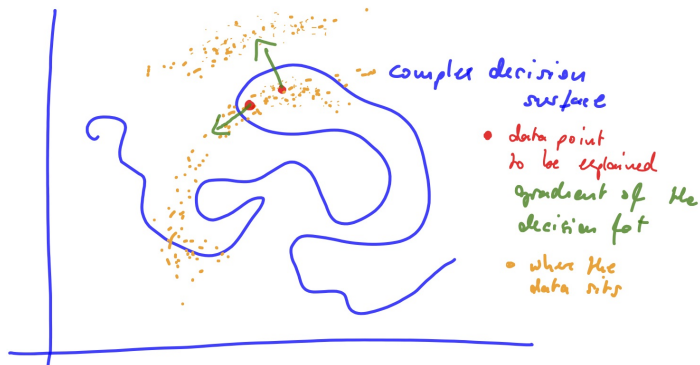
However, depending on the geometry of the data, this gradient might not always be useful as an explanation (Bordt, Luxburg, 2022):

- ▶ The gradient identifies the direction in which we would need to move in order to quickly change the class label.
- ▶ If this direction is along the data manifold, it includes meaningful changes, and might give interesting explanations.
- ▶ However, if this direction encodes changes that do not result in meaningful data, Then the explanations might be useless.

Local post-hoc explanations: LIME on tabular data (6)



Local post-hoc explanations: LIME on tabular data (7)

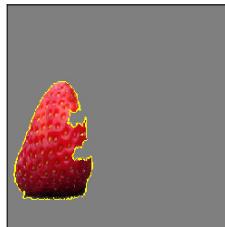
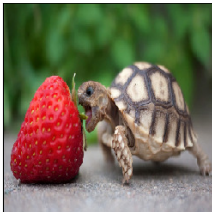


Local post-hoc explanations: LIME on tabular data (8)

Some references on LIME and its properties:

- M. T. Ribeiro, S. Singh, and C. Guestrin. Why should I trust you? Explaining the predictions of any classifier. In SIGKDD, 2016.
- Garreau, Luxburg: Explaining the Explainer: A First Theoretical Analysis of LIME. AISTATS, 2020.
- Bordt, Upadhyay, Akata, von Luxburg: The Manifold Hypothesis for Gradient-Based Explanations. Preprint, 2022.

Local post-hoc explanations: LIME on images



(b) explaining 'terrapiin' (c) explaining 'strawberry'

Local post-hoc explanations: LIME on images (2)

- ▶ Feature bins are being replaced by “superpixels”:
 - ▶ We split the image into superpixels (pre-processing step).
 - ▶ We sample images in the vicinity of the given image, by randomly switching on and off some of the superpixels.
 - ▶ The presence/absence of a superpixel is encoded in a binary vector. This vector plays the same role as the feature vector for tabular data.
- ▶ The LIME algorithm then identifies those “features/superpixels” that contribute most to the classification of the image.

Local post-hoc explanations: SHAP

Takes intuition from cooperative game theory: want to evaluate how much influence each player has on the payoff that a coalition of players receives.

In the context of ML: Want to evaluate how much each feature contributes to the final model prediction.

Local post-hoc explanations: SHAP (2)

Vanilla idea:

- ▶ Consider the model prediction f at a given input point x as our payoff.
- ▶ Interpret features as players. Let F be the set of all features.
- ▶ To evaluate the importance of feature i , we consider all subsets $S \subset F$ of features.
- ▶ For each such subset S , we would like to compare the “prediction just based on the features in S ” with the “prediction based on features in $S \cup \{i\}$ ”:
$$f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)$$
- ▶ Then we would sum this over all possible subsets, which then gave us a score for the importance of feature i (the higher the more important).

But...

Local post-hoc explanations: SHAP (3)

But:

- (1) How could $f_S(x_S)$ be defined, what should it mean, and how could we compute it?
- (2) Do we really want to sum over all possible subsets?

Local post-hoc explanations: SHAP (4)

Solution for (1): Definition of $f_S(x_S)$

- ▶ One idea could be to always retrain the algorithm with subsets of features included/excluded. But this would be prohibitive from a computational point of view.
- ▶ Instead we use conditional expectations: To evaluate a model f at input point x based on a subset S of features, we consider $E(f(x)|x_S)$. That is, we fix those features of x that are present in the set S , and try to average over the missing features.

Local post-hoc explanations: SHAP (5)

- Example: Consider $x = (17, -4, 0.5)$. It has three features. For $S = \{1, 2\}$, we then integrate over the third coordinate:

$$\begin{aligned} f_S(x_S) &= E(f(x)|x_S) := E(f(x)|x_1 = 17, x_2 = -4) \\ &= \int_{-\infty}^{\infty} f((17, -4, \mathbf{x}_3)) dp(\mathbf{x}_3) \end{aligned}$$

- Easy in theory. But in practice?
Typically, this integral is approximated over the training set, but many other choices are possible.

Local post-hoc explanations: SHAP (6)

Solution for (2):

- ▶ Naive solution: just compute the sum over some randomly sampled subsets.
- ▶ More complicated solutions exist, under particular assumptions.

Local post-hoc explanations: SHAP (7)

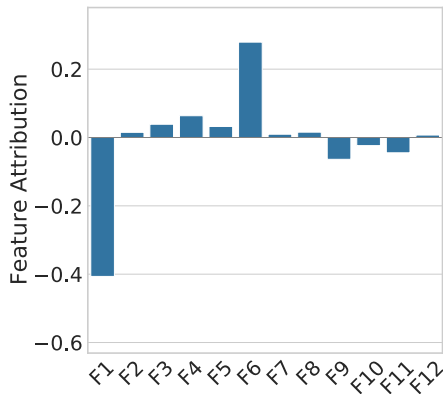
SHAP values have some nice theoretical properties. The (somewhat simplified story) goes like this:

SHAP tries to approximate the function f locally at point x by a model that is additive in the features:

$$f(x) \approx \Phi_0 + \sum_{i=1}^d \Phi_i x_i$$

Φ_i are the scores we are looking for. The higher the absolute value of the score Φ_i , the more important the feature i for the decision $f(x)$.

Local post-hoc explanations: SHAP (8)



(a) SHAP

Local post-hoc explanations: SHAP (9)

See original paper:

Lundberg and Lee. A unified approach to interpreting model predictions. NeurIPS 2017.

Local post-hoc explanations: Counterfactual explanations

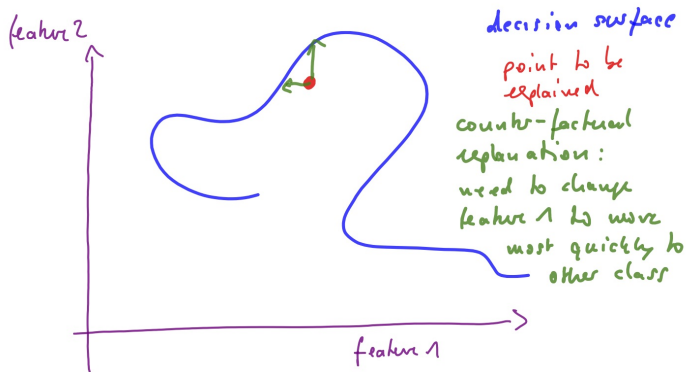
Main idea: what would be the smallest change to the input such that the decision function would return a different result?

Examples:

- ▶ If your income would be 500 Euro more, you would get the credit.
- ▶ If your gender would be male, you would be admitted to the program. (?!?)

To construct such explanation, one tries to identify one (few) features such that increasing/decreasing/altering the feature brings the local data point to the other side of the decision surface.

Local post-hoc explanations: Counterfactual explanations (2)



Local post-hoc explanations: Counterfactual explanations (3)

Literature: Wachter, Mittelstadt, and Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. 2017.

Criticism on post-hoc explanations

- Explanation algorithms can easily be tricked or manipulated or cheated upon
- Different explanation algorithms give rise to completely different explanations:

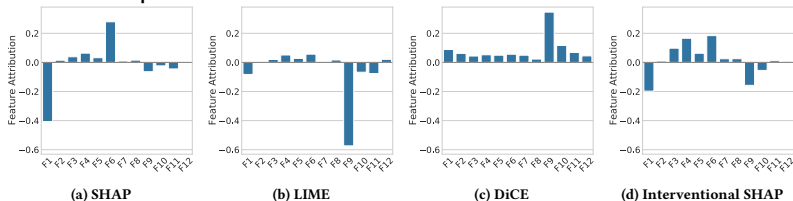


Figure 1: Different explanation algorithms lead to different explanations. Depicted are the feature attri-

Hence, in an adversarial setting, the “opponent” can cherry-pick an explanation that he likes.

Criticism on post-hoc explanations (2)

- ▶ For an examiner, such cherry-picking is pretty much impossible to detect. So we cannot trust these explanations if we don't trust the one who invokes the algorithm.
- ▶ My personal take: if it is important to get reliable explanations (for example, in a societal context), only use interpretable algorithms, do not allow for complicated models with post-hoc explanations. Other people have different opinions here...

Literature:

Anders et al: Fairwashing explanations with off-manifold detergent. ICML 2020.

D. Slack et al: Fooling lime and shap: Adversarial attacks on post hoc explanation methods. IAES 2020

Slack et al: Counterfactual Explanations Can Be Manipulated. 2021

Rudin: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead.

Nature Machine Intelligence 2019

Bordt, Finck, Raidl, von Luxburg: Post-hoc explanations fail to achieve their purpose in adversarial contexts. FAccT, 2022

Summary

- ▶ Explanations in ML are an active and important field of research
- ▶ This is also related to Ethics and Philosophy of Science (what is an explanation, after all?)
- ▶ Some approaches exist, but there is definitely room for more ...

Energy footprint of ML

Literature:

David MacKay: Sustainable Energy, without the hot air. 2008

This book is highly recommended, a must read for everybody interested in climate debates!

AI and climate

AI does use a lot of energy and produces a lot of carbon dioxide!
To understand its impact, let's first dive into some numbers.

In the following I closely follow the book of MacKay (and this is the source of the numbers; even if numbers might be a bit outdated, it is all about the orders of magnitude, which are still correct).

Units of energy

Let's first convert all sources of energy to the same unit, so we can compare different sources of energy (electricity, gas, fuel, ...).

The basic units are:

- ▶ W (Watt) for power (German: "Leistung")
- ▶ Wh or kWh for energy (German: "Energie", "Arbeit"), This is the critical number.
- ▶ The connection between the two: Power is the rate at which something uses energy:
 $\text{power} = \text{energy} / \text{time}.$
- ▶ Analogy with water running through pipes: the absolute consumption in liters would be the equivalent of energy, the flow capacity of your pipes would be the equivalent of power (flow = volume/time). In the end, what is important is the absolute amount of water you use.

Units of energy (2)

To get a feeling, we often use a somewhat **unusual unit**, **kWh/day**. For example, we might say that an old-fashioned light bulb of 40 W, switched on round the clock, would consume 1 kWh / day:

$$1 \text{ kWh/day} = 1000 \text{ Wh}/24\text{h} = 1000/24 \text{ W} \approx 40 \text{ W}$$

The advantage of this unit is that it relates to something we know (the light bulb), hence is easier to interpret.

We now use this unit to get a feeling for the orders of magnitude of energy consumption:

Energy consumption of the average European

A typical energy footprint of a person in Europe might be:

		Power necessary round the clock
20%	Car (say 50 km per day)	40 kWh / day = 1600 W
20 %	Plane (say, one intercontinental trip per year)	30 kWh / day = 1200 W
10%	Food	15 kWh / day = 600 W
20 %	Heating/cooling	37 kWh / day \approx 1500 W
30 %	Stuff (production of all the things that we consume)	60 kWh / day \approx 2400 W

(source: David McKay book)

Your household and devices

Fridge/Freezer (new/old)	20 - 150 W
heat pump (new/old)	50 - 150 W
light bulbs old	40 - 100 W
ceiling floodlights (old)	up to 500 W!!!!
LED lamps	3 - 15 W
30" LED screen	50 W
65" LED TV	150 W
Mac Mini	85 W
a wifi router	10 W
apple phone charger, no load	0,012 W
apple phone charger, loading	< 1 W
google / youtube use	1 W
Total energy consumption of all of Germany for google/youtube, averaged over the whole population	

Your household and devices (2)

Take away:

- ▶ Unplugging your phone won't make much of a difference, really.
- ▶ **The important devices are the ones that are constantly running, for many hours or even round the clock.**
- ▶ Running a webserver (60W) day and night adds up to something considerable: $60W = 1.5kW/h$. This is 10% of the energy of your food production!
- ▶ Try to switch off all devices if you don't need them (desktops, screens, wifi, ...)
- ▶ One more flight, and your energy footprint rises considerably.

YOUR energy use for AI

How expensive is computing on GPUs / CPUs?

one GPU (eg Nvidia Tesla V100)	300 W
one CPU (eg intel Core i7)	100 W

If you train a neural network on 10 GPUs for 1 week, this gives

$$(10 \times 300W) \times (7 \cdot 24h) \approx 500kWh$$

If we convert this to our standard unit of measurement (distributing it constantly over the year), this means

$$500kWh/365 \text{ days} \approx 1.4kWh/day$$

Again, this is about 10% of the energy used to produce your food.

Multiply it by the number of weeks you train...

YOUR energy use for AI (2)

Take away: if you work in the field of ML / AI, **your energy consumption for training and testing might be significant**, so please be responsible:

- ▶ make sure you test your code sufficiently on a small scale before you deploy it large scale.
- ▶ don't just fiddle around with lots of architectures, parameters, datasets on a cluster; have a plan first
- ▶ think about your setup to avoid running meaningless experiments altogether

But, also keep the relations sane: unless you are a super-user, the power consumption of the experiments that you conduct to write a NeurIPS paper is an order of magnitude smaller than the power consumption of the **flight that actually brings you to the conference.**

The large scale

On the large scale, energy use of information technology is not negligible. Here are some examples:

All numbers from 2017:

Google world wide 2017	6 TWh
Bitcoin world wide 2017	20 TWh
German yearly electricity production	250 TWh

Google Germany 2017, averaged over all population	1 W per person, constantly running
---	------------------------------------

Saving energy with the help of AI?

People often advocate that AI might also help us to save energy:

- ▶ Intelligent power management
- ▶ More efficient farming
- ▶ More efficient production
- ▶ ...

My personal take on this:

- ▶ It is definitely worth to work on such questions, and small contributions might add to something considerable.
- ▶ But we are definitely not there yet!
- ▶ Personally, I doubt that the savings through AI will be on the same scale as the consumption.

Takeaway

First takeaway:

Energy consumption of AI research is not negligible:

- ▶ **Compute power** (say, for some of us as large as the our food consumption, but for most of us it is far less). Be responsible and don't waste computation power on compute clusters!
- ▶ **Traveling!!!** Avoid flying.
 - ▶ Focus on conferences that are close-by.
 - ▶ Resist the temptation to fly to an invited talk at MIT for two days.

Note of caution: not all energy that is being used by “digital services” can be attributed to AI. Much of it goes into streaming videos, using cloud services, etc ...

Takeaway (2)

Second takeaway:

To save energy on a scale that matters for the climate, there are many other important things that you personally can take care of:

- ▶ Avoid flying. Avoid the car.
- ▶ For devices that are running round the clock: replace old ones by new ones (freezer; heat pump; light bulbs).
- ▶ For devices that are running round the clock: consider switching them off if you don't need them.
- ▶ Insulate your house, use solar energy. It is not about saving money, but about saving energy.

Low rank matrix methods

Introduction: recommender systems, collaborative filtering

Recommender systems

Goal: give recommendations to users, based on their past behavior:

- ▶ Recommend movies (e.g., netflix)
- ▶ recommend music (e.g., lastfm)
- ▶ recommend products to buy (e.g., amazon)

ANY IDEAS HOW WE COULD DO THIS?

Recommender systems (2)

Content-based approach:

- ▶ Model products based on **explicit features**. Use these features to define a similarity function between products
- ▶ If a user likes product A, then recommend products similar to A.

Prominent example: Pandora Radio. You start with a song you like, and then Pandora plays similar songs.

Recommender systems (3)

Collaborative approach:

- ▶ Forget about explicitly modeling users or features.
- ▶ Instead, **implicitly** model similarity of users and products based on past shopping behavior.
- ▶ Consider user/product matrix with ratings. Defines an implicit similarity between users (or products).
- ▶ Then recommend similar items to similar users.

Prominent example: lastfm

ADVANTAGES / DISADVANTAGES OF THE TWO?

Matrix factorization basics

Hastie, Tibshirani, Wainwright: Statistical learning with sparsity.
2015. Chapter 7.2

Recap: singular value decomposition (SVD)

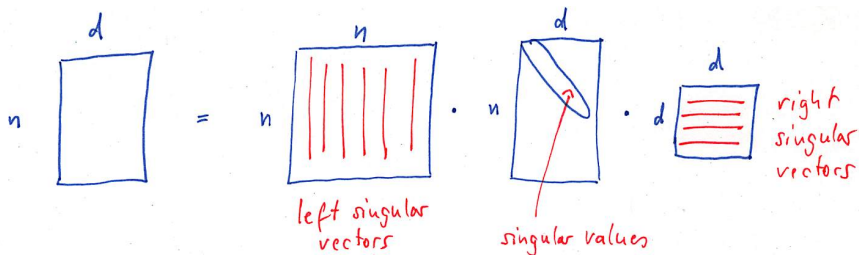
Recall PCA:

- ▶ Eigenvalue decomposition for a symmetric matrix
- ▶ Best rank- k approximation of the matrix: based on highest k eigenvalues

Now want to do something more general for arbitrary (non-square) matrices.

Recap: singular value decomposition (SVD) (2)

Every (!) matrix can be decomposed as follows:



$$\Phi = U \cdot \Sigma \cdot V^t$$

U is the matrix of left singular vectors, V the right singular vectors, and the diagonal of Σ contains the singular values.

Recap: singular value decomposition (SVD) (3)

There is a simple relationship between SVD and PCA:

For any matrix A ,

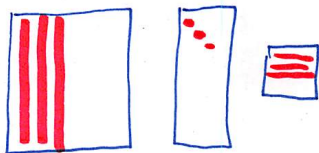
- ▶ the left singular vectors are the eigenvectors of AA^t
- ▶ the right singular vectors are the eigenvectors of A^tA
- ▶ the non-zero singular values are the square roots of the eigenvalues of both AA^t and A^tA .

PROOFS: EXERCISE!

SVD for rank-k approximation

Consider the following “Top-k-SVD” procedure:

- ▶ Given a $n \times d$ matrix A .
- ▶ Compute the SVD such that the singular values are sorted in decreasing order.
- ▶ Keep the first k columns of U and V . Call the resulting matrices $U_k \in \mathbb{R}^{n \times k}$ and $V_k \in \mathbb{R}^{d \times k}$ (such that $V_k^t \in \mathbb{R}^{k \times d}$).
- ▶ Keep the singular values $\sigma_1, \dots, \sigma_k$ and write them in a diagonal matrix $\Sigma_k \in \mathbb{R}^{k \times k}$.
- ▶ Now define $A_k := U_k \Sigma_k V_k^t$.



SVD for rank-k approximation (2)

Intuitive interpretation:

- ▶ Assume that A is a matrix recording ratings of n users about d products.
- ▶ Then the top- k right singular vectors can be interpreted as basic “customer types”. Each customer is a weighted mixture of the basic customers.
- ▶ Similarly, the top- k left singular vectors can be interpreted as basic “product types”.

SVD for rank-k approximation (3)

The Frobenius norm of matrix is defined as $\|B\|_F := \sqrt{\sum_{ij} b_{ij}^2}$.

Theorem 48 (SVD as rank-k approximation)

The matrix A_k defined by the first k singular values/vectors solves the following rank- k -approximation problem:

Given A and k , find the matrix A_k with rank at most k such that $\|A - A_k\|_F$ is minimized.

Proof: EXERCISE (consider the hints in Exercise 7.2. p. 196 in “statistical learning with sparsity”).

EXERCISE: COMPARE THIS RESULT WITH THE CORRESPONDING RESULT FOR PCA!

SVD for rank- k approximation (4)

Digest again what the intuitive interpretation is:

- ▶ We know the full product / user matrix.
- ▶ The k top singular vectors define k types of users / products.
- ▶ Based on these few types, we can “explain” the behavior of everybody (up to a small approximation error).

Low rank matrix completion

Literature

- Hastie, Tibshirani, Wainwright: Statistical learning with sparsity. 2015. Chapter 7

Some important original papers:

- Candes, Recht: Exact matrix completion via convex optimization. Foundations of computational mathematics, 2009.
- R. Keshavan, Andrea Montanari, and Sewoong Oh: Matrix completion from noisy entries. JMLR, 2010
- Mazumder, Hastie, Tibshirani: Spectral regularization algorithms for learning large incomplete matrices. JMLR, 2010.

Netflix problem

General problem:

- ▶ Consider a huge matrix of user ratings of movies. Rows correspond to movies, columns correspond to users, entries are ratings on a scale from 1 to 5.
- ▶ We only know few entries in this matrix.
- ▶ The matrix completion problem is to estimate the missing entries in order to recommend new movies to a user.

Netflix problem (2)

History of the Netflix challenge:

- ▶ Launched in 2006
- ▶ Data: About 20.000 movies, 500.000 users, 10^9 ratings (that is, about 1% of the entries are known)
- ▶ Goal: predict the missing entries, error measure RMSE (root mean squared error $\sqrt{\sum_{i=1}^n (z_i - \hat{z}_i)^2 / n}$)
- ▶ First team that beats Netflixes own algorithm by an improvement of at least 10% wins a prize of 1 million dollars.
- ▶ Was finally achieved in 2009.

Matrix completion problem

General setup:

- ▶ Consider an $m \times n$ matrix which is unknown.
- ▶ We get to see some entries in the matrix.
- ▶ Assume that the position of the revealed entries is random (no adversarial setting).
- ▶ Goal is to estimate the unknown entries as well as possible.

CAN YOU THINK OF EASY / DIFFICULT CASES? IS IT ALWAYS POSSIBLE?

Matrix completion problem (2)

We need to make assumptions to be able to solve this problem (inductive bias!). If the entries are not related to each other (say, independent random numbers), there is no way in which we could predict missing entries.

Matrix completion problem (3)

High-level idea from learning theory: A useful inductive bias is one that leads to a “small” set of possible matrices.

Here is what everybody uses:

We are going to look for a matrix that has low rank.

(Just as a sanity check: a matrix with independent random entries typically has high rank, the eigenvalues follow the semi-circle law.)

Matrix completion, first formulations

Denote by Ω the set of entries of a matrix Z that have been observed: we know the values z_{ij} for all $(i, j) \in \Omega$. We would like to solve the following problem:

$$\text{minimize } \text{rank}(M) \quad \text{subject to } m_{ij} = z_{ij} \text{ for } (i, j) \in \Omega.$$

or a slightly weaker version

$$\text{minimize } \text{rank}(M) \quad \text{subject to } \sum_{(i,j) \in \Omega} (m_{ij} - z_{ij})^2 \leq \delta$$

or the regularization version

$$\text{minimize } \sum_{(i,j) \in \Omega} (m_{ij} - z_{ij})^2 + \lambda \text{rank}(M)$$

Is NP hard ☹

CAN YOU SEE WHAT MAKES IT SO DIFFICULT?

Matrix completion, first approach using SVD

Here is a straight forward heuristic by which we can try to solve the optimization problem:

Hard-Impute

- ▶ Have an initial guess for the missing entries \leadsto matrix Z_1
- ▶ Compute the SVD of Z_1 , keep the first r singular components $\leadsto Z_2$
- ▶ Fill in the missing entries with the ones of Z_2 , and start over again ...

Sometimes this works reasonably.

But let's try to think about alternatives ... one option for non-convex optimization problems is always to construct a convex relaxation (have seen this before, at least twice, where?)

Trace as convex relaxation of rank

Let us try to find a convex relaxation of the rank function:

- If $\sigma := \sigma(A)$ denotes the vector of singular values of matrix A , then

$$\text{rank}(A) = \|\sigma\|_0$$

- Recall the standard approach in sparse regression (Lasso). We relaxed the 0-norm to the 1-norm, which is convex:

$$\|\sigma\|_1 = \sum_i |\sigma_i|.$$

We now use this as a norm for matrices, it is called the nuclear norm or the trace norm:

$$\|A\|_{tr} := \|\sigma(A)\|_1.$$

One can prove that the nuclear norm is the tightest convex relaxation of the rank of a matrix.

Trace norm regularization

Now consider the following optimization problems:

$$\text{minimize } \|M\|_{tr} \text{ subject to } \sum_{(i,j) \in \Omega} (m_{ij} - z_{ij})^2 \leq \delta \quad (*)$$

$$\text{minimize } \frac{1}{2} \sum_{(i,j) \in \Omega} (m_{ij} - z_{ij})^2 + \lambda \|M\|_{tr} \quad (**)$$

These two problems are essentially the same, once in the natural formulation (*) and once in the regularization / Lagrangian formulation (**).

Trace norm regularization (2)

Two big questions:

- ▶ Can we give an efficient algorithm that can find the global optimum, either in formulation (*) or in (**)?
- ▶ If yes, what can we say about the theoretical properties of the global optimum, how close is it going to be to the matrix we are looking for? In particular, **how many entries do we need to observe to find a good reconstruction?**

Solving $(*)$, naive algorithm: semi-definite program

The first formulation of the problem is a **semi-definite program**. In principle, SDPs can be solved in polynomial time, but “polynomial” can still be very long... There are general-purpose solvers for such problems, but they are so slow that they only work for small instances.

We skip the details.

Solving $(**)$ efficiently: soft-impute

Here is a strategy to solve $(**)$:

- ▶ Start with initial guesses for the missing values.
- ▶ Compute the SVD, “soft-threshold” the singular values by some threshold λ .
- ▶ Repeat until convergence.

Soft-thresholding:

- ▶ Given the SVD of a matrix $Z = UDV^t$, denote the singular values by d_i .
- ▶ We define $S_\lambda(Z) := UD_\lambda V^t$ where D_λ is the diagonal matrix with diagonal entries $(d_i - \lambda)_+ := \max(d_i - \lambda, 0)$
- ▶ Soft thresholding decreases the trace norm and also often decreases the rank of a matrix.

Solving $(**)$ efficiently: soft-impute (2)

Let's first consider one step of soft-thresholding on a completely known matrix Z (no missing entries):

Proposition 49

Consider a matrix Z that is completely known, and choose some $\lambda > 0$. Then solution of the optimization problem

$$\min_M \|Z - M\|_F^2 + \lambda \|M\|_{tr}$$

is given by the result $S_\lambda(Z)$ of one round of soft-thresholding.

Proof: see Mazumder, Hastie, Tibshirani: Spectral regularization algorithms for learning large incomplete matrices. JMLR, 2010.

Solving $(**)$ efficiently: soft-impute (3)

Now we want to use a similar approach to complete the matrix Z in case it is just partially observed (problem $(**)$).

Introduce notation:

- ▶ Denote by Ω the set of matrix entries that are known.
- ▶ Define the “projection” $P_{\Omega}(Z)$ as the matrix that has the original values z_{ij} at all the observed positions of Z , and 0 otherwise (that is, fill the unobserved entries with zeros).
- ▶ With this definition,
$$\sum_{(i,j) \in \Omega} (z_{ij} - m_{ij})^2 = \|P_{\Omega}(Z) - P_{\Omega}(M)\|_F.$$
- ▶ Define $P_{\Omega}^{\perp}(Z)$ as the “projection” of the matrix Z on the entries that are NOT in Ω (so that $Z = P_{\Omega}(Z) + P_{\Omega}^{\perp}(Z)$).

Solving $(**)$ efficiently: soft-impute (4)

Input: Z the partially observed matrix; a sequence of parameters $\lambda_1, \dots, \lambda_K$

Algorithm 1 SOFT-IMPUTE

1. Initialize $Z^{\text{old}} = 0$.
2. Do for $\lambda_1 > \lambda_2 > \dots > \lambda_K$:
 - (a) Repeat:
 - i. Compute $Z^{\text{new}} \leftarrow \mathbf{S}_{\lambda_k}(P_{\Omega}(Z) + P_{\Omega}^{\perp}(Z^{\text{old}}))$.
 - ii. If $\frac{\|Z^{\text{new}} - Z^{\text{old}}\|_F^2}{\|Z^{\text{old}}\|_F^2} < \varepsilon$ exit.
 - iii. Assign $Z^{\text{old}} \leftarrow Z^{\text{new}}$.
 - (b) Assign $\hat{Z}_{\lambda_k} \leftarrow Z^{\text{new}}$.
3. Output the sequence of solutions $\hat{Z}_{\lambda_1}, \dots, \hat{Z}_{\lambda_K}$.

Intuition:

Solving $(**)$ efficiently: soft-impute (5)

- ▶ Inner loop (a) for fixed λ : clamp the observed values of the matrix, fill the rest by a low-rank approximation, until convergence
- ▶ Outer loop (2): start with a case that is easy (λ_i large, matrix low rank) and work our way towards the more difficult situation of smaller λ_i

Solving $(**)$ efficiently: soft-impute (6)

Properties:

- ▶ It can be proved that this algorithm always converges to the global solution of $(**)$ (for a suitable choice of the sequence $(\lambda_k)_{k=1,\dots,K}$).
- ▶ It can be implemented efficiently even for huge matrices (just a couple of hours for the whole Netflix dataset). Main trick: can decompose the dense matrix Z in a sum of a sparse matrix and a low rank matrix. Can exploit this cleverly in the algorithm.

$$\mathcal{P}_\Omega(\mathbf{Z}) + \mathcal{P}_\Omega^\perp(\mathbf{Z}^{\text{old}}) = \underbrace{\mathcal{P}_\Omega(\mathbf{Z}) - \mathcal{P}_\Omega(\mathbf{Z}^{\text{old}})}_{\text{sparse}} + \underbrace{\mathbf{Z}^{\text{old}}}_{\text{low rank}}$$

- ▶ R-package softImpute

Details: Mazumder, Hastie, Tibshirani

Solving $(**)$ efficiently: soft-impute (7)

Soft-impute on the Netflix data:

Netflix Competition Data

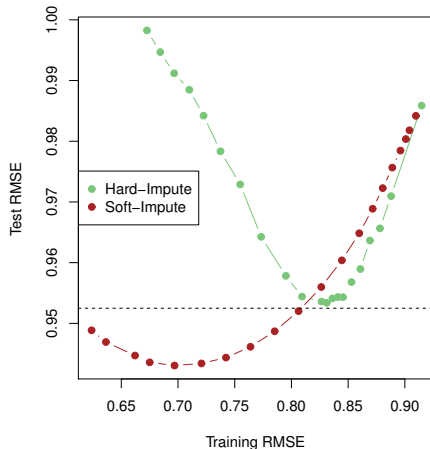
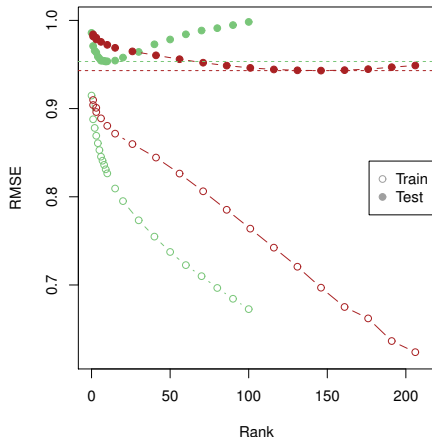


Figure from "Statistical learning with sparsity". Dotted line on rhs = Netflixes own algorithm, baseline. Sanity check: random guessing would have $RMSE \approx 2$

Theoretical results on matrix completion

Setting:

- ▶ Consider an arbitrary matrix Z . For simplicity, assume that Z is square of size $p \times p$.
- ▶ Assume that we observed n entries of the matrix, drawn uniformly at random.
- ▶ Question: how large does n need to be such that we can successfully reconstruct the matrix Z (exactly or approximately)?

Condition: no empty columns or rows

Problem of empty columns:

If there is a column (or a row) that does not have any observed entry, it will be impossible to reconstruct the matrix.

QUESTION TO YOU: We sample n elements. Each element belongs to one of p columns. How large does n need to be such that, with reasonably high probability, we have at least one element per column?

Condition: no empty columns or rows (2)

ANSWER: this is the coupon collector problem, we need at least $p \log p$ samples.

Condition: enough parameters to express rank r

Number of “parameters” for a rank- r matrix:

- ▶ A rank- r matrix of dimension $p \times p$ can be described by r vectors of length $p \leadsto rp$ parameters
- ▶ In general, we cannot assume that we can “compress” these rp entries any further
- ▶ So it is plausible that we won't be able to perfectly reconstruct such a matrix if we observe less than rp entries of the matrix.

(of course, this argument is really hand-waiving, but often such hand-waiving intuition helps to get a first feeling for a problem; the next step is then to make it precise)

Condition: coherency

Consider the following matrix

$$\mathbf{Z} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Assume we want to solve EXACT recovery:

- ▶ It is of rank one (so the problem should be easy, few parameters to learn).
- ▶ However, there is only one important entry.
- ▶ If we sample of order less than of the order p^2 entries, likelihood is high that we never get to see this particular entry. So if we are after exact recovery, we have a problem.

Condition: coherency (2)

The problem in this example is:

- ▶ Some entries are much more important than others.
- ▶ In mathematical terms: the eigenvectors of this matrix are too much aligned with the standard basis of \mathbb{R}^p .

Condition: coherency (3)

To deal with this problem:

We want to “measure” up to which extent the eigenvectors of the matrix are aligned with the standard basis: this leads to the notion of **coherence**.

Definition: Let U be a subspace of \mathbb{R}^d of dimension r and P_U the orthogonal projection on U . Then the **coherence** of U wrt to the standard basis $(e_i)_i$ is defined as

$$\mu(U) := \frac{d}{r} \max_{i=1,\dots,d} \|P_U e_i\|^2$$

FOR MATRIX COMPLETION, IS IT BETTER TO HAVE SMALL OR LARGE COHERENCE?

Condition: coherency (4)

To get intuition, consider the case where U is spanned by one vector:

- ▶ The smaller $\mu(U)$, the “easier” the matrix will be to recover.
- ▶ Maximum value of coherence occurs if $U = \text{span}(e_i)$ (more generally, if $e_i \in U$). Then we have $\|P_u e_i\| = 1$.
- ▶ Minimum value of coherence occurs if U is spanned by the vector $(1/\sqrt{n}, \dots, 1/\sqrt{n})^t$.
- ▶ Intuition: More generally, coherence is low (good) if all entries of the vector have about the same order of magnitude. **Then each entry contains about the same amount of information, so sampling few entries should be fine.**

Condition: coherency (5)

Examples:

The matrix we started with:

$$M = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & 0 & \\ 0 & & & \end{pmatrix}, \quad \sigma_1 = 1, \quad \sigma_2 = \dots = \sigma_n = 0, \quad v_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \Rightarrow \|P_{v_1} e_1\| = 1$$

\Rightarrow maximal coherence

Condition: coherency (6)

Same matrix, just flipped entries:

$$M = \begin{pmatrix} 0 & 1 & \dots & 1 \\ 1 & & & \\ \vdots & & & \\ 1 & & & \end{pmatrix}, \quad \sigma_1, \sigma_2 > 0, \sigma_3 \dots \sigma_n = 0$$

$$v_1 = \begin{pmatrix} 0.3 \\ 0.4 \\ 0.4 \\ 0.4 \\ 0.4 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 0.9 \\ -0.1 \\ -0.1 \\ -0.1 \\ -0.1 \end{pmatrix}, \quad v := (v_1, v_2)$$

$$\|P_v e_1\| = 1$$

(because $e_1 \in \text{span}\{v_1, v_2\}$)

\Rightarrow maximal coherence

As a sanity check, the all ones matrix:

$$M = \begin{pmatrix} 1 & \dots & 1 \\ \vdots & & \\ 1 & & \end{pmatrix}, \quad \sigma_1 = n, \sigma_2 = \dots \sigma_n = 0$$

$$v_1 = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \quad \|P_{v_1} e_i\| = \frac{1}{\sqrt{n}}$$

\Rightarrow low coherence
(as small as it can be)

Condition: coherency (7)

A random rank- r matrix:

$$M = \text{rand}(n, r), \quad M = M \cdot M' \quad (\text{random rank-}r \text{ matrix})$$

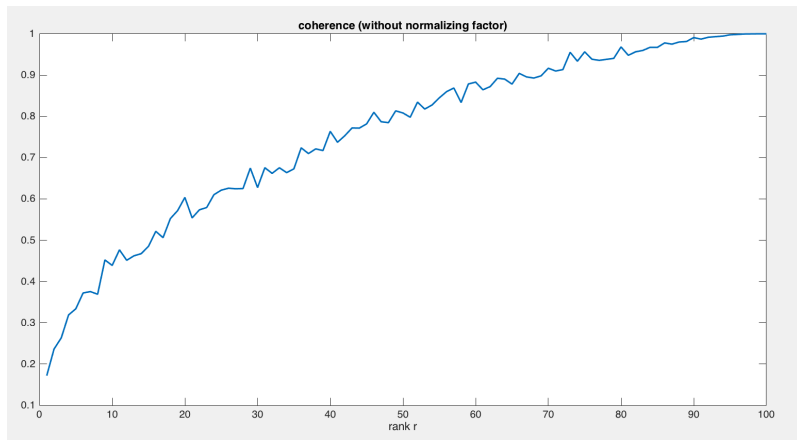
$$\sigma_1, \dots, \sigma_r > 0, \quad \sigma_{r+1} \dots \sigma_n = 0$$

- If r low, coherence low as well
- If r high, coherence gets higher (extreme case
 $r = n$: then we project on the full space, hence
 coherence = 1)

See plot on next page:

Condition: coherency (8)

Plot of $\max_{i=1\dots n} \|P_V e_i\|$, random rank- r matrix, $n=100$:



Guarantee for exact recovery

One can prove guarantees of the following flavor (p size of matrix, r rank):

With high probability, exact recovery is possible if the number of observed entries is at least

$$N \geq Crp \log p$$

Here, C is a constant that depends on the coherence of the matrix:

- ▶ Coherence low: $N \approx rp \log p$
- ▶ Coherence high: $C \cdot r \approx p$, such that we need to sample about $p^2 \log p$ entries.

Proofs are beyond the scope of this lecture.

GIVEN OUR PREVIOUS OBSERVATIONS, DO YOU THINK THIS IS A GOOD OR A BAD RESULT?

Guarantee for exact recovery (2)

Concretely, the coherency-based theorem for exact recovery looks as follows (Candes, 2009):

Theorem 1.1 *Let M be an $n_1 \times n_2$ matrix of rank r with singular value decomposition $U\Sigma V^*$. Without loss of generality, impose the conventions $n_1 \leq n_2$, Σ is $r \times r$, U is $n_1 \times r$ and V is $n_2 \times r$. Assume that*

A0 *The row and column spaces have coherences bounded above by some positive μ_0 .*

A1 *The matrix UV^* has a maximum entry bounded by $\mu_1 \sqrt{r/(n_1 n_2)}$ in absolute value for some positive μ_1 .*

Suppose m entries of M are observed with locations sampled uniformly at random. Then if

$$m \geq 32 \max\{\mu_1^2, \mu_0\} r(n_1 + n_2) \beta \log^2(2n_2) \quad (1.2)$$

for some $\beta > 1$, the minimizer to the problem

$$\begin{array}{ll} \text{minimize} & \|X\|_* \\ \text{subject to} & X_{ij} = M_{ij} \quad (i, j) \in \Omega. \end{array} \quad (1.3)$$

is unique and equal to M with probability at least $1 - 6 \log(n_2)(n_1 + n_2)^{2-2\beta} - n_2^{2-2\beta^{1/2}}$.

(here $\|\cdot\|_*$ denotes the trace norm).

Guarantee for approximate recovery

Guarantee from Keshavan et al, 2010:

Theorem 1.2 Let $N = M + Z$, where M is a (μ_0, μ_1) -incoherent matrix of rank r , and assume that the subset of revealed entries $E \subseteq [m] \times [n]$ is uniformly random with size $|E|$. Further, let $\Sigma_{\min} = \Sigma_r \leq \dots \leq \Sigma_1 = \Sigma_{\max}$ with $\Sigma_{\max}/\Sigma_{\min} \equiv \kappa$. Let \hat{M} be the output of OPTSPACE on input N^E . Then there exists numerical constants C and C' such that if

$$|E| \geq Cn\sqrt{\alpha}\kappa^2 \max\{\mu_0 r \sqrt{\alpha} \log n; \mu_0^2 r^2 \alpha \kappa^4; \mu_1^2 r^2 \alpha \kappa^4\}, \approx n r^2 \log n \cdot \mu$$

then, with probability at least $1 - 1/n^3$,

$$\underbrace{\frac{1}{\sqrt{mn}} \|\hat{M} - M\|_F}_{\text{error of recovery}} \leq C' \kappa^2 \frac{n\sqrt{r\alpha}}{|E|} \underbrace{\|Z^E\|_2}_{\text{amount of noise}}. \quad (3)$$

provided that the right-hand side is smaller than Σ_{\min} .

Σ_i : singular values of M

Simulations: noise-free setting

Assume you want to run simulations for low rank matrix completion, under controlled conditions.

HOW COULD YOU GENERATE A “RANDOM” LOW RANK MATRIX TO PLAY WITH?

Simulations: noise-free setting (2)

- ▶ Model the ground truth by a simple low rank model:
 - ▶ Generate the $p \times r$ matrices U and V with independent random entries, normally distributed according to $N(0, 1)$ (in the figures below: $p = 20$ or 40 , and $r = 1$ or 5 .)
 - ▶ Define $Z = U \cdot V^t$.
(WHAT IS THE RANK OF THESE MATRICES? WHAT CAN YOU SAY ABOUT THE SINGULAR VALUES?)
- ▶ Generate the toy data: Sample n random entries from this matrix.
- ▶ Try to recover the ground truth:
 - ▶ Use soft-impute to complete the matrices, results in \hat{Z} .
 - ▶ Check whether $\|\hat{Z} - Z\| \approx 0$
 - ▶ Repeat this experiment many times and report fraction of correctly recovered matrices.

Simulations: noise-free setting (3)

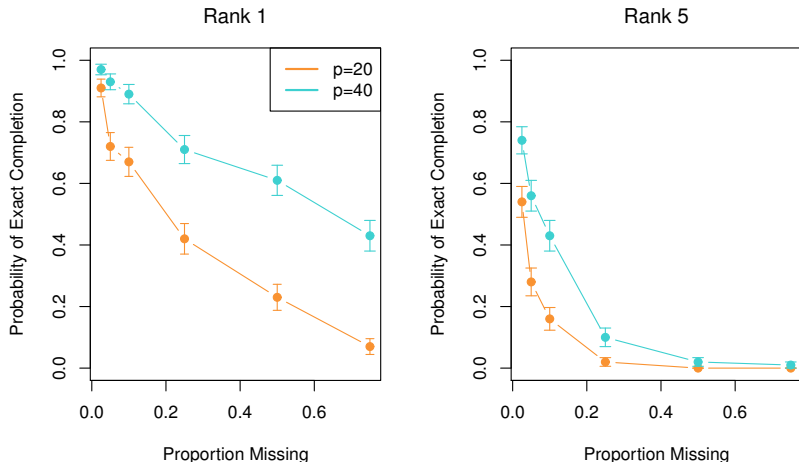


Figure from "Statistical learning with sparsity"; r =rank of true matrix, p dimension of matrix

WHAT CAN YOU SEE?

Simulations: noise-free setting (4)

- ▶ Problem gets **harder the less entries we observe** (curves decrease from left to right). Makes sense.
- ▶ Problem gets **more difficult if original matrix has a higher rank** (compare left and right figure). Makes sense.
- ▶ For a fixed r , likelihood of exact recovery increases with the dimension p (that is, the **problem seems simpler if the original matrix is larger!!!**). Not entirely sure here, let's discuss:

The last point seems surprising, here are possible explanations:

Simulations: noise-free setting (5)

- Possible explanation 1: to recover a rank r -matrix of size p means to recover rp parameters. The matrix has $\text{const} \cdot p^2$ many entries that can serve as our source of information. So the ratio between what we want and what we have is $rp/p^2 = r/p$, which for fixed r gets better when p increases.
- Possible explanation 2: Consider the recovery theorem of Keshavan: everything else being fixed, the necessary number of samples m increases as $m_p \approx p \log p$, but probability of exact recovery increases as $1 - 1/p^3$. So if we would compare the probability of recovering all entries from m_p measurements, then this probability would increase with p .
(Note that strictly speaking, the theorem does not make a statement that says that if the proportion of missing samples is fixed to a particular constant, that then the probability of

Simulations: noise-free setting (6)

recovery grows with p . But I guess one could extract such a statement from the proof.)

- All these explanations are a bit ad hoc, and if I really had to find out, I would need to dig in, rewrite the theorems, and run simulations on my own.

Simulations: noisy setting

Noisy setting:

- ▶ Generate the matrix Z as before.
- ▶ Now add Gaussian noise with standard deviation 0.5 to each of the entries of Z , this results in Z_{noisy} .
- ▶ Now try to reconstruct Z , when you observe entries from Z_{noisy} (using Soft-impute).
- ▶ Plot

$$\text{average relative error} := \frac{\text{average Frobenius norm error}}{\text{noise standard deviation}}$$

(WHAT IS THE BEST AVERAGE RELATIVE ERROR YOU COULD HOPE FOR?)

Results:

Simulations: noisy setting (2)

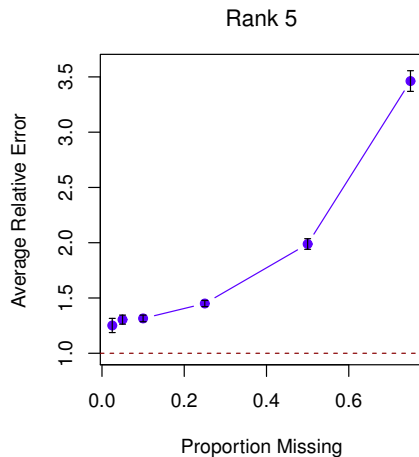
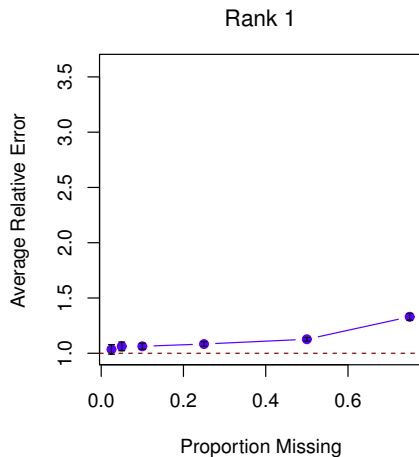


Figure from "Statistical learning with sparsity"

Outlook / literature

- ▶ We just scratched the surface, there are many more variants of the problem, and also many more algorithms.
- ▶ If you are interested, the book “Statistical learning with sparsity” is a good starting point.

History:

- ▶ PhD thesis Fazel 2002: nuclear norm as surrogate for rank
- ▶ Nati Srebro et al, 2005, nuclear norm relaxations, with first generalization bounds.
- ▶ Candes, Recht: Exact matrix completion via convex optimization. Foundations of computational mathematics (FOCM), 2009. Bounds in exact case.
- ▶ Netflix challenge: 2006 - 2009.

Compressed sensing

Book chapters:

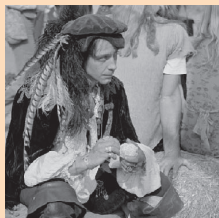
- Hastie, Tibshirani, Wainwright: Statistical learning with sparsity. 2015. Chapter 10.
- Shalev-Shwartz, Ben-David: Understanding Machine Learning, Section 23.3.

Motivation

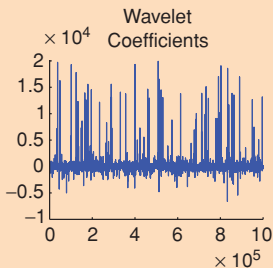
Consider the camera in your phone:

- ▶ If you take a picture, it first generates a raw image that is stored by a pixel-based representation (e.g., rgb values for each pixel).
- ▶ Then it compresses the picture by representing it in a suitable basis (say, a wavelet basis) and generates a compressed version of the image (say, a jpg file).

Motivation (2)



(a)



(b)



(c)

Figure 1.2 (a) Original megapixel image with pixel values in the range $[0, 255]$ and (b) its wavelet transform coefficients (arranged in random order for enhanced visibility). Relatively few wavelet coefficients capture most of the signal energy; many such images are highly compressible. (c) The reconstruction obtained by zeroing out all the coefficients in the wavelet expansion but the 25,000 largest (pixel values are thresholded to the range $[0, 255]$). The differences from the original picture are hardly noticeable.

Motivation (3)

Idea: it would be great if we could skip the first step and directly capture the data in the better representation.

This is called compressed (compressive) sensing.

Applications:

- ▶ Cameras with little power / storage. Take a picture with less pixels, but achieve the same quality in the end.
- ▶ MRI / tomography: scans parts of the body, scanning time increases for larger images. Want to speed up scanning (take less pictures) but still have the same quality.

Setup

Assume we observe a vector $x \in \mathbb{R}^d$.

- ▶ Typically it is not be sparse in the standard basis, that is $\|x\|_0$ is close to d
- ▶ But it might be sparse in a different basis: There exists an orthonormal matrix U such that $x = U\alpha$ and α is a sparse vector: $\|\alpha\|_0 =: s$ is small
- ▶ If we would know the basis U *and* would have a technical way to measure the signal in this basis directly, this would be great.
- ▶ Goal is now: construct a basis that does the job.

Setup (2)

Notation in the following:

- ▶ d dimension of the original space (high)
- ▶ s true sparsity of the signal in the basis U (low)
- ▶ k the sparsity we actually achieve (hopefully low as well)

We always have $s \leq k \leq d$.

Example: Single pixel camera

- ▶ Standard camera: record millions of pixels and then apply compression (e.g. jpeg compression) after the picture has been taken.
- ▶ New approach: we use only a single pixel detector to create images and we gather only a small fraction of the information, **effectively compressing the image while taking it.**

Example: Single pixel camera (2)

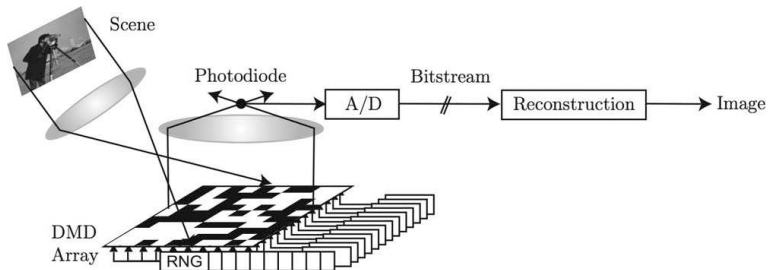


Figure 6. A schematic diagram of the “one-pixel camera.” The “DMD” is the grid of micro-mirrors that reflect some parts of the incoming light beam toward the sensor, which is a single photodiode. Other parts of the image (the black squares) are diverted away. Each measurement made by the photodiode is a random combination of many pixels. In *“One is Enough”* (p.114), 1600 random measurements suffice to create an image comparable to a 4096-pixel camera. (Figure courtesy of Richard Baraniuk.)

Original resolution: $d = 64 \times 64 = 4096$.

Compressed: $k = 1600$ measurements.

Example: Single pixel camera (3)

How does it work? (see figure):

- ▶ uses an array of bacteria-sized mirrors to acquire a random sample of the incoming light and project it on a single photodiode.
- ▶ Each mirror can be tilted in one of two ways, either to reflect the light toward the single sensor or away from it. **The photodiode thus receives a linear combination of the images on all the mirrors that are “on”.**
- ▶ Thus the light that the sensor receives is a weighted average of many different pixels, all combined into one pixel.
- ▶ One configuration w_i of the mirrors gives rise to one linear measurement of our signal.
- ▶ **Repeat this procedure for k different mirror configurations w_1, \dots, w_k and store the k measurements the photodiode receives.**

Example: Single pixel camera (4)

Cool result: By taking of the order $\Theta(s \log(d/s))$ snapshots, with a different random selection of pixels each time, the single-pixel camera is able to acquire a recognizable picture with a resolution comparable to d pixels.

Example: Single pixel camera (5)



One is Enough. A photograph taken by the “single-pixel camera” built by Richard Baraniuk and Kevin Kelly of Rice University. (a) A photograph of a soccer ball, taken by a conventional digital camera at 64×64 resolution. (b) The same soccer ball, photographed by a single-pixel camera. The image is derived mathematically from 1600 separate, randomly selected measurements, using a method called compressed sensing. (Photos courtesy of R. G. Baraniuk, *Compressive Sensing* [Lecture Notes], Signal Processing Magazine, July 2007. © 2007 IEEE.)

Example: Single pixel camera (6)

Nice blog discussion by Terrence Tao on this topic:

[https://terrytao.wordpress.com/2007/04/13/
compressed-sensing-and-single-pixel-cameras/](https://terrytao.wordpress.com/2007/04/13/compressed-sensing-and-single-pixel-cameras/)

Key steps in compressed sensing

1. We design k “linear measurements” $w_1, \dots, w_k \in \mathbb{R}^d$ (in applications, this is done by specific hardware, see later).
2. Nature picks an unknown signal $x \in \mathbb{R}^d$ (d large)
3. We directly receive the measurement results
 $\tilde{x}_1 = \langle w_1, x \rangle, \tilde{x}_2 = \langle w_2, x \rangle, \dots$, resulting in the measurement vector $\tilde{x} \in \mathbb{R}^k$ (k reasonably small).
4. We are now supposed to reconstruct $x \in \mathbb{R}^d$ from $\tilde{x} \in \mathbb{R}^k$.

Note: The goal is to design a single (!) set of measurement vectors w_1, \dots, w_k that works well for all (!) signals x in the sense that we are able to reconstruct with little error.

Compressed sensing

Another way to describe it:

- ▶ We don't measure the signal x directly, but just a “compressed” version of it, namely we measure

$$\tilde{x} = Wx \in \mathbb{R}^k$$

where W is a $k \times d$ -matrix with $k \ll d$. The matrix is known to us, we choose it before we see any data.

- ▶ Now we want to reconstruct the high-dimensional signal x from the low-dimensional representation \tilde{x} .

WHAT WOULD BE THE NAIVE WAY OF RECONSTRUCTION?
WHY DOESN'T IT WORK?

Compressed sensing (2)

- ▶ To reconstruct, we would need to solve the linear system $\tilde{x} = Wx$ for x .
- ▶ However, the latter is heavily underdetermined (we have k equalities but d unknowns, with $k \ll d$). There are infinitely many solutions to this linear system.

The trick is going to be:

- ▶ We need to make assumptions on the x we are looking for.
- ▶ In particular, we assume that x is sparse in some basis. We will see below that this does the job.

Compressed sensing main result

Theorem 50 (Compressed sensing with random measurements)

Fix a signal length d and a sparsity level s . Let W be a $k \times d$ matrix with $k = \Theta(s \log(d/s))$, with each of its entries chosen independently from a standard normal distribution $N(0, 1)$. Then, with high probability over the choice of W , every s -sparse signal can be efficiently recovered from $\tilde{x} = Wx$ by the following optimization problem:

$$\text{minimize } \|x\|_1 \text{ subject to } \tilde{x} = Wx$$

Compressed sensing main result (2)

DO YOU THINK THAT $k = \Theta(s \log(d/s))$ IS GOOD OR BAD?

Compressed sensing main result (3)

- ▶ To measure a signal with sparsity s , we will definitely need at least s measurements.
- ▶ $s \log(d/s)$ is definitely much smaller than d , it is close to s .

So it looks really great!

Some hand-waivy intuition why the result is exactly

$k = \Theta(s \log(d/s))$:

- ▶ We know that we are looking for a vector of length d which has only s non-zero components (in some appropriate basis). But we don't know which are the non-zero components.
- ▶ There are $\binom{d}{s}$ subsets of size s among the d components.
- ▶ To recover the vector we could try out all different such subsets, and then reconstruct based on these subsets.

Compressed sensing main result (4)

- Any efficient algorithm to do so would need to be able to distinguish at least $\log \binom{d}{s} \approx \Theta(s \log(d/s))$ many situations.

(This argument is similar to the proof for the lower bound in comparison-based sorting ...)

Compressed sensing main result (5)

Key steps in the proof of the theorem:

1. If we choose the matrix W such that it has the “restricted isometry property” (RIP, see below), then any k -sparse vector x can be reconstructed from its compressed image \tilde{x} with only little distortion, by an inefficient algorithm using ℓ_0 -norm optimization.
2. The reconstruction of x from \tilde{x} can be calculated equally well using ℓ_1 -norm optimization (rather than ℓ_0 -norm). This is very surprising!
3. It is easy to find matrices W that have the RIP property: we can use a random matrix with $k = \Omega(s \log d)$ where s is the sparsity of the signal and d the original dimension of the space.

Proof step 1: define RIP matrices

Definition (Restricted Isometry Property):

A matrix $W \in \mathbb{R}^{k \times d}$ is $(\varepsilon, 2s)$ -RIP if for all $x \neq 0$ with $\|x\|_0 \leq s$ we have

$$\left| \frac{\|Wx\|_2^2}{\|x\|_2^2} - 1 \right| \leq \varepsilon$$

Intuitively:

Multiplying an RIP-matrix to a sparse vector does not considerably change the norm of the vector, no matter which vector we choose.

Proof step 1: Perfect reconstruction from RIP using ℓ_0

The following theorem shows that RIP matrices yield a lossless compression for sparse vectors:

Theorem 51 (Reconstruction based on 0-norm)

Let W be $(\varepsilon, 2s)$ -RIP for some $\varepsilon < 1$, x with $\|x\|_0 \leq s$ (that is, x is sparse in the standard basis of \mathbb{R}^d), $y = Wx$ the compression of x by matrix W . Then the reconstruction

$$\tilde{x} := \operatorname{argmin}\{\|v\|_0 ; v \in \mathbb{R}^d, Wv = y\}$$

coincides exactly with x .

Proof step 1: Perfect reconstruction from RIP using ℓ_0 (2)

Proof of the theorem, by contradiction:

- ▶ Assume that $\tilde{x} \neq x$.
- ▶ By definition of \tilde{x} we have

$$\|\tilde{x}\|_0 \leq \|x\|_0 \leq s$$

In particular, $\|x - \tilde{x}\|_0 \leq 2s$.

- ▶ Now apply the RIP property to the vector $z := x - \tilde{x}$. Recall that $Wx = W\tilde{x}$, hence $Wz = 0$.
- ▶ Then the RIP property gives

$$\left| \frac{\|Wz\|_2^2}{\|z\|_2^2} - 1 \right| = |0 - 1| = 1 > \varepsilon$$

Proof step 1: Perfect reconstruction from RIP using ℓ_0 (3)

Note that this theorem immediately gives a first algorithm for reconstructing x from its sparse representation \tilde{x} .

WHICH ONE? IS IT A GOOD ONE?

Proof step 1: Perfect reconstruction from RIP using ℓ_0 (4)

We need to solve an ℓ_0 -optimization problem. This is combinatorial, hard, undesirably ...

Proof step 2: Perfect reconstruction from RIP using ℓ_1

Now comes the very surprising result: we get exact (!) recovery even if we replace the ℓ_0 norm by the ℓ_1 norm:

Theorem 52 (Reconstruction based on 1-norm)

Under the same assumptions as before:

$$\operatorname{argmin}\{\|v\|_0 ; v \in \mathbb{R}^d, Wv = y\} = \operatorname{argmin}\{\|v\|_1 ; v \in \mathbb{R}^d, Wv = y\}$$

Proof: omitted, a nice writeup can be found in Chapter 23.3 of Shalev-Shwartz and Ben-David.

WHY IS THIS SURPRISING? WHY IS IT INTERESTING?

Proof step 2: Perfect reconstruction from RIP using ℓ_1 (2)

The theorem gives us an pretty efficient (=polynomial) way to exactly (!) reconstruct the original signal from the sparse one, by solving a linear program!

Remarks:

- ▶ There exists an even stronger version of the theorem which does not assume that the original vector is s -sparse. Essentially, the statement says that we can perfectly recover the s largest components.
- ▶ There also exists a version of the theorem which only assumes that the matrix is sparse in some unknown basis (not the original one).

Proof step 3: Constructing RIP matrices

We still haven't clarified how we actually construct the compression matrix W :

Theorem 53 (Random matrices are RIP)

- (i) Let $s \leq d$ an integer, $\varepsilon, \delta \in]0, 1[$. Choose $k \geq \text{const} \cdot \frac{s \log(d/(\delta\varepsilon))}{\varepsilon^2}$. Now choose $W \in \mathbb{R}^{k \times d}$ such that each entry is drawn randomly from a normal distribution $N(0, 1/s)$. Then, with probability $1 - \delta$ (over the choice of the matrix), the matrix W is (ε, s) -RIP.
- (ii) More generally, if U is any $d \times d$ orthonormal matrix, then with probability $1 - \delta$, the matrix WU is (ε, s) -RIP.

Proof: omitted, a nice writeup can be found in Chapter 23.3 of Shalev-Shwartz and Ben-David.

Remarks:

Proof step 3: Constructing RIP matrices (2)

- ▶ This result is closely related to the theorem of Johnson-Lindenstrauss, which is widely used in randomized algorithms.
- ▶ The second part of the theorem takes care of the situation that the signal is not sparse in the original basis, but a different basis, by additionally applying a basis transformation U to the signal.

More intuition: a different way to tell the same story

Compressed sensing is advantageous whenever

- ▶ signals are sparse in a known basis
- ▶ measurements (or computation at the sensor end) are expensive
- ▶ but computations at the receiver end are cheap.

More intuition: a different way to tell the same story (2)

- ▶ One measures a relatively small number of **random linear combinations of the signal values** — much smaller than the number of signal samples nominally defining it.
- ▶ However, because the underlying signal is compressible, the nominal number of signal samples is still an overestimate of the effective number of degrees of freedom of the signal.
- ▶ As a result, the signal can be reconstructed with good accuracy from relatively few measurements by a clever nonlinear procedure.

More intuition: a different way to tell the same story (3)

When does it work?

Transform sparsity: The desired image should have a sparse representation in a known transform domain (i.e., it must be compressible by transform coding).

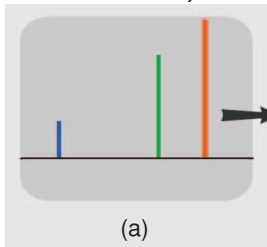
Incoherence of undersampling artifacts: The artifacts in linear reconstruction caused by undersampling should be incoherent (noise like) in the sparsifying transform domain.

Nonlinear reconstruction: The image should be reconstructed by a nonlinear method that enforces both sparsity of the image representation and consistency of the reconstruction with the acquired samples.

Example: time series

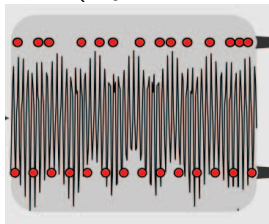
The following example is taken from Lustig, M., Donoho, D. L., Santos, J. M., Pauly, J. M. (2008). Compressed sensing MRI. Signal Processing Magazine, IEEE, 25(2), 72-82.

Sparse signal, as it would be in the appropriate basis (say, a vector of Fourier coefficients of a time series).



Example: time series (2)

Signal in the “default basis” (say, the time series itself, not sparse):



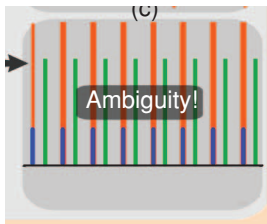
Assume it is too costly to sample the whole time series completely.

Example: time series (3)

Obvious first idea: equispaced undersampling.

- ▶ Just measure (“sense”) the signal at equispaced positions (in the image on the previous slide, at the positions indicated by the red dots at the bottom).
- ▶ Replace the remaining entries with 0.
- ▶ Go over to the sparse basis and represent the signal there.

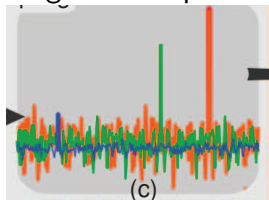
Result: artifacts called “aliasing”. It does not work at all!



Example: time series (4)

The compressed sensing approach: Random undersampling

- ▶ Instead of sampling at equispaced positions, randomly pick some entries (in the image before, this is indicated by the red dots at the top).
- ▶ Try to represent the image in the sparse basis:



Works! If we threshold the small Fourier coefficients, we are left with the sparse representation of the signal.

Example: time series (5)

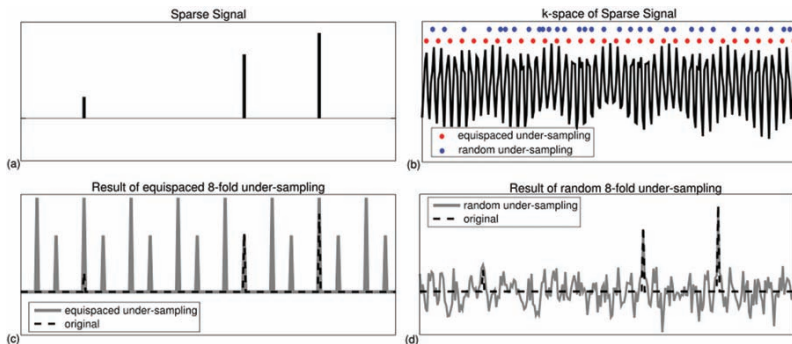


Figure 2. Reconstructing a sparse wave train. (a) The frequency spectrum of a 3-sparse signal. (b) The signal itself, with two sampling strategies: regular sampling (red dots) and random sampling (blue dots). (c) When the spectrum is reconstructed from the regular samples, severe “aliasing” results because the number of samples is 8 times less than the Shannon-Nyquist limit. It is impossible to tell which frequencies are genuine and which are impostors. (d) With random samples, the two highest spikes can easily be picked out from the background. (Figure courtesy of M. Lustig, D. Donoho, J. Santos and J. Pauly, *Compressed Sensing MRI*, Signal Processing Magazine, March 2008. © 2008 IEEE.)

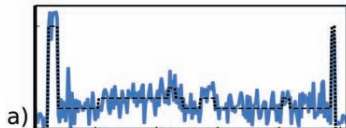
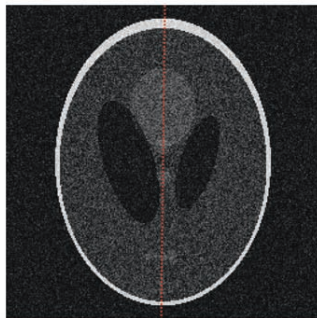
Example: Images

Example taken from: MacKenzie, Dana. Compressed sensing makes every pixel count. What is happening in the mathematical sciences 7 (2009): 114-127.

Original noisy image. Shown is the image itself and (I guess) the coefficients in Fourier (Wavelet?) basis. Signal is sparse in this basis (but of course, it was not recorded in this basis, here the transform to the sparse basis happened afterwards):

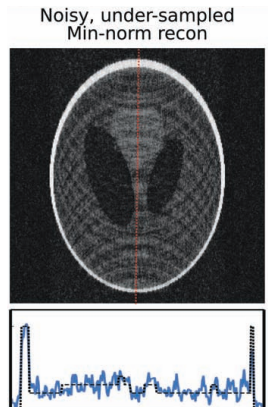
Example: Images (2)

Noisy, Fully sampled



Example: Images (3)

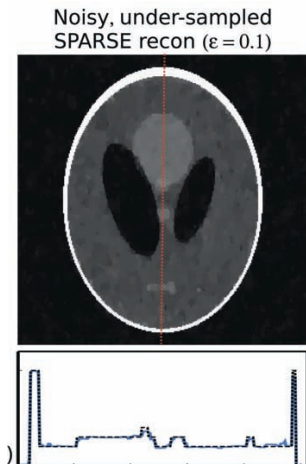
Now use random undersampling to record the picture, and reconstruct based on l_2 -minimization:



Many artifacts.

Example: Images (4)

Random undersampling, reconstruction based on ℓ_1 -minimization:



Nice :-)

Relation to standard information theory

Shannon sampling theorem (1949):

- ▶ A time-varying signal with no frequencies higher than d hertz can be perfectly reconstructed by sampling the signal at regular intervals of $1/2d$ seconds (that is, we sample at $2d$ different time points).
- ▶ A signal with frequencies higher than d hertz cannot be reconstructed uniquely if we sample with this rate; there is always a possibility of aliasing (two different signals that have the same samples).

Compressed sensing: makes stronger assumptions than Shannon:

- ▶ The achievable resolution is controlled not only by the maximal number of frequencies (the dimension d of the space), but by the “information content” (the sparsity s of the signal).

Relation to standard information theory (2)

- If we know that among the d different frequencies only s of them really occur, then we can reconstruct the signal from a small number of measurements.

Outlook

- ▶ Active area of research
- ▶ Lots of actual applications!!! Cameras, MRI scanning, etc

(*) Ranking from pairwise comparisons

Introduction

Text books (but I don't like both chapters so much):

- ▶ Mohri et al. chapter 9
- ▶ Shalev-Shwartz/Ben-David, Chapter 17.4

Papers: see the individual sections.

Introduction, informal

- ▶ Ranking candidates for a job offering
- ▶ Ranking of the world's best tennis players
- ▶ Ranking of search results in google
- ▶ Ranking of molecules according to whether they could serve as a drug for a certain disease

IN WHICH SENSE ARE THESE PROBLEMS DIFFERENT, IN WHICH SENSE SIMILAR?

Introduction, informal (2)

- ▶ top-k ranking vs full ranking
- ▶ sampling with or without replacement
- ▶ active vs. passive selection of comparisons
- ▶ distributed or not
- ▶ ground truth exists or not

Problems run under many different names: rank aggregation, ranking, tournaments, voting, ... and are tackled in many different communities (machine learning, computational social choice, theoretical computer science, etc).

Introduction, more formal

- ▶ Given n objects x_1, \dots, x_n .
- ▶ In the simplest case, we assume that there exists a “true” total order \prec on the objects, that is there exists a permutation π such that $x_{\pi(1)} \prec x_{\pi(2)} \prec \dots \prec x_{\pi(n)}$.
- ▶ Goal is to learn this permutation from partial observations of the ranking. In the simplest case, observations are of the form $x_k \prec x_l$ for certain pairs (k, l) .

Introduction, more formal (2)

Distance functions between permutations:

Given two permutations π and $\hat{\pi}$ of the same set of objects. Want to compute how different these rankings are.

- **Kendall- τ distance:** Count the number of pairs (i, j) that are in different order in the two permutations:

$$d_{\tau}(\pi, \hat{\pi}) := \frac{2}{n(n-1)} \sum_{i=1}^n \sum_{j=i+1}^n \mathbb{1}\{\text{sign}(\pi(i) - \pi(j)) \neq \text{sign}(\hat{\pi}(i) - \hat{\pi}(j))\}$$

- **Spearman- ρ distance:** Count for each object by how much it is “displaced” in one permutation with respect to the other:

$$d_{\rho}(\pi, \hat{\pi}) = \sum_{i=1}^n |\pi(i) - \hat{\pi}(i)|$$

Introduction, more formal (3)

- **Top-k differences.** Assume we are just interested in whether the top k objects in the two rankings coincide. Denote by S_k the set of first k objects in π , and by \hat{S}_k the corresponding set in $\hat{\pi}$. We define the distance

$$d_k(\pi, \hat{\pi}) := |S_k \triangle \hat{S}_k| := |(S_k \cup \hat{S}_k) \setminus (S_k \cap \hat{S}_k)|$$

Note that it only looks at the unordered sets, not at the order within the sets.

- **Normalized discounted cumulative gain (NDCG):** We take the ranking π as “reference ranking”. Then we compare it to the second ranking $\hat{\pi}$, but we weight errors among the top items of π more severely than errors for items at the bottom of the ranking π . Many different ways in which this can be done ...

Introduction, more formal (4)

There exists a large variety of probabilistic model assumptions in the literature. Here are some typical examples:

- ▶ We assume there exists a true ranking. When asking a user to provide an answer to the question $x_i ? x_j$, he gives an incorrect answer with probability p (where p is independent of x_i, x_j).
- ▶ We assume that the objects x_i can be represented by a real number $u(x_i)$, for example a utility score. Then we define $x_i \prec x_j := u(x_i) < u(x_j)$. The likelihood to observe an incorrect answer depends on the distance $u(x_i) - u(x_j)$. Many different versions, for example the BTL model below.

Introduction, more formal (5)

- **Model for paired comparisons: Bradley-Terry-Luce (BTL) model.** Each object has a score $u(x_i)$ (utility value, skill, ...). Probability of answers to comparisons are modeled by a logistic model:

$$P(x_i \succ x_j) = \frac{1}{1 + \exp(-(u(x_i) - u(x_j)))}$$

- **Mallows model (probability distribution over all permutations):** Assume that π is the true ranking. Then the probability to observe a ranking $\hat{\pi}$ is chosen proportional to $\alpha^{d_\tau(\pi, \hat{\pi})}$ where $\alpha \in]0, 1]$ is a parameter and d_τ is the Kendall- τ distance. Choosing $\alpha = 1$ implies the uniform distribution over all permutations, the closer α is to 0, the more the mass concentrates around π .

Introduction, more formal (6)

Default statistical approach. Given a probabilistic model, a straight forward idea is to use a **maximum likelihood estimator**: Find the permutation that maximizes the likelihood of the observed data. However, it is often infeasible due to computational complexity (need to have a clever way to try out all permutations).

Default algorithmic approach: Given the observations, find the permutation that is as consistent as possible with your observations (minimizes a loss function). For example, assume in a sports tournament that everybody played against everybody. Now find a ranking that violates as few outcomes as possible. This problem is NP hard, there exists a PTAS for it (Kenyon-Mathieu and Schudy: How to rank with few errors. STOC 2007).

Simple but effective counting algorithm

Based on the paper:

Shah, Wainwright: Simple, robust and optimal ranking from pairwise comparisons. Arxiv, 2015.

The model

Ground truth model is very general:

- ▶ n objects
- ▶ For each pair of objects, assume a parameter $p_{ij} := P(i \succ j)$. Assume that $P(i \succ j) + P(i \prec j) = 1$ (no ties).
- ▶ Define the score that measures the probability that object i beats a randomly chosen object j :

$$\tau_i := \frac{1}{n} \sum_{j=1}^n P(i \succ j)$$

This score τ_i can be interpreted as the probability that object i wins against a randomly chosen object j (under the uniform probability distribution of objects). We consider the ranking induced by these scores as the true ranking. Note: high score = top of the list.

The model (2)

Observation model:

- ▶ Assume that the number of times that a pair (i, j) is observed is distributed according to a binomial distribution $\text{Bin}(r, p_{\text{obs}})$ (where $r \in \mathbb{N}$ and $p_{\text{obs}} \in [0, 1]$ are global parameters independent of i and j).
- ▶ To generate the observations we proceed as follows:
 - ▶ For each pair (i, j) , we draw a random variable $n_{ij} \sim \text{Bin}(r, p_{\text{obs}})$. This is the number of times that we are going to observe comparisons between i and j .
 - ▶ Now we ask n_{ij} times independently whether $i \prec j$ or $i \succ j$. We get the answers with probabilities according to p_{ij} .

The model (3)

This model is very general, it encompasses most of the more specialized models that exist in the literature.

Goal: Given a set of comparisons, find either the top-k ranking or the full ranking.

The counting algorithm

Simple counting algorithm:

- ▶ Define $\hat{\tau}_i$ as the number of times that object i has won over another object j , based on the observed comparisons.
- ▶ Define the estimated ranking (or the top-k set) as the order induced by the estimated scores $\hat{\tau}_i$.

This algorithm is about the simplest thing you can come up with, it is sometimes called **Borda count** or **Copeland method** in the literature.

Bounds for exact recovery of top-k items

Define the following parameter:

$$\Psi_k(n, r, p_{\text{obs}}) := \underbrace{(\tau_k - \tau_{k+1})}_{=: \text{separation para.}} \cdot \underbrace{\sqrt{\frac{n \cdot p_{\text{obs}} \cdot r}{\log n}}}_{\text{sampling para.}}$$

- ▶ The separation parameter measures how well-separated the first k items are from the remaining ones.
- ▶ The sampling parameter is a complexity term that depends on the number n of objects and the expected number $n \cdot p_{\text{obs}} \cdot r$ of observations per object.
- ▶ We will see below that that the larger Ψ_k , the easier it is to discover the true ranking. (DOES IT MAKE SENSE?)

Bounds for exact recovery of top-k items (2)

Theorem 54 (Exact top k recovery)

- (a) Upper bound: Denote by S_k the set of true top-k items, and by \hat{S}_k the estimated set of top-k items according to the counting algorithm. If $\Psi_k(n, r, p_{\text{obs}}) \geq 8$, then $\hat{S}_k = S_k$ with probability at least $1 - 1/n^{14}$.
- (b) Lower bound: If $\Psi_k(n, r, p_{\text{obs}}) \leq 1/7$, $n \geq 7$ and $r \cdot p_{\text{obs}} > \log n / (2n)$. Then there exist instances such that any algorithm that attempts to recover the top-k items will err with probability at least $1/7$.

Digesting the theorem

Let's digest the upper bound by constructing a simple example:

Example 1:

- ▶ Assume the true ordering is $o_1 \succ o_2 \succ \dots \succ o_n$, that is the best player is o_1 . Our goal is to find the best player (that is, $k = 1$).
- ▶ Assume a noise-free setting: a better player always wins against a worse player, that is $p_{ij} = 1$ if $o_i \succ o_j$ and 0 otherwise.
- ▶ Then $\tau_i = (n - i)/n$, and in particular $\tau_1 - \tau_2 = 1/n$.
- ▶ Consider the case where we observe each pair exactly r times (we set $p_{\text{obs}} = 1$, so the number of observations is deterministic as well).

Digesting the theorem (2)

- Upper bound in the theorem: perfect recovery works if $\Psi := (\tau_1 - \tau_2) \cdot \sqrt{\frac{nr}{\log n}} \geq \text{const.}$. In our case, $\tau_1 - \tau_2 = 1/n$, and solving the equation leads to $r \geq n \log n$. That is, the upper bound guarantees perfect recovery if we observe each pair (!) at least $n \log n$ times. So overall we have to make $n^3 \log n$ comparisons.

On the other side, the lower bound says the following:

- Assume that we have separation $\tau_1 - \tau_2 = 1/n$ as in our example. Then, if we observe less than $r = n^3 \log n$ we cannot guarantee recovery.

Digesting the theorem (3)

- ▶ The point is that the lower bound is a worst case statement: for the worst of all examples, we **cannot guarantee recovery if we observe less than $n^3 \log n$** examples.
- ▶ For example, we can construct an example that has separation $1/n$ as well, but has lots of noise:
Example 2: Assume that for $i > 2$ we have
 $P(1 \succ i) = 1/2 + 2/(n-2)$, $P(2 \succ i) = 1/2 + 2/(n-2)$,
and all other pairwise probabilities are $1/2$. This clearly is a very difficult case.

Taken jointly, upper and lower bound say:

- ▶ The counting algorithm gives perfect recovery if we get to see $n^3 \log n$ comparisons (this is for the case of separation $1/n$).

Digesting the theorem (4)

- ▶ There are instances where it fails if we get to see less than this amount of samples.
- ▶ In this sense, the counting algorithm is optimal (up to constants in the bounds).
- ▶ But of course, the query complexity (number of comparisons we need) is huge. The problem is not that we have a bad algorithm (this is what the lower bound tells). The problem is that we make too little assumptions, so there is no structure we can exploit.

(As a side remark: the lower bound also holds if we make the Bradely-Terry-Luce assumptions, one can construct an example that satisfies their assumptions and still needs many comparisons).

Digesting the theorem (5)

Just as comparison: In our example 1, we are in a completely noiseless case.

- ▶ WHAT IS THE NUMBER OF COMPARISONS WE NEED TO SORT A SEQUENCE?
- ▶ WHAT IS THE NUMBER OF COMPARISONS WE NEED TO FIND THE TOP ITEM IN A LIST OF ITEMS?

So we are miles away from this good performance. HOW CAN THIS BE, WHAT IS THE DIFFERENCE?

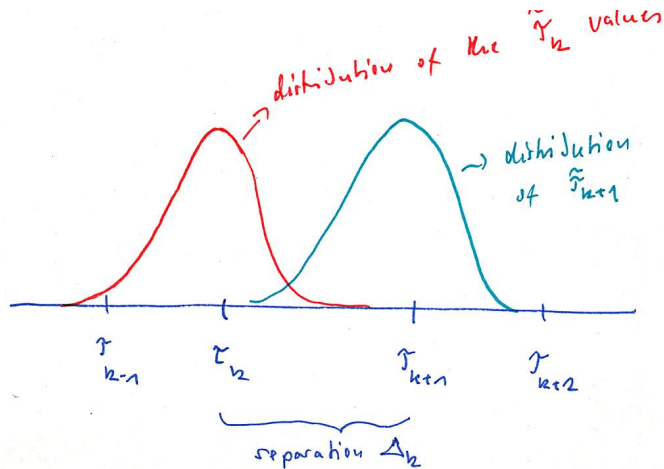
Proof sketch, upper bound

Let's briefly look at the proof:

- ▶ For each pair (i, j) , we have a certain number of independent observations.
- ▶ The parameter $\hat{\tau}_i$ is an average over these observations.
- ▶ This average is highly concentrated around its expectation. Applying standard concentration inequalities (Bernstein), one can show that the deviations of the random variables are small.
- ▶ In particular, we can then bound the probability that one of the top- k items “is beaten” (in terms of $\hat{\tau}_i$) by one of the not-top- k items.

See the following figure:

Proof sketch, upper bound (2)



Proof sketch, lower bound

We construct one particular example in a clever way:

- ▶ For each a in $\{k-1, k, k+1, \dots, n\}$ let $S^*(a) := \{1, 2, \dots, k-1\} \cup \{a\}$. This is supposed to be the true top- k set.
- ▶ Define the probabilities

$$P_a(i \succ j) \begin{cases} 1/2 & \text{if } i, j \in S^*(a) \text{ or } i, j \notin S^*(a) \\ 1/2 + \delta & \text{if } i \in S^*(a) \text{ and } j \notin S^*(a) \\ 1/2 - \delta & \text{if } i \notin S^*(a) \text{ and } j \in S^*(a) \end{cases}$$

- ▶ Note that the true τ -values give the correct top- k set.
- ▶ Our goal is to identify the true permutation based on observations, that is we want to find the correct parameter a that has been used.

Proof sketch, lower bound (2)

To construct the lower bound, we now want to show that no matter which algorithm we use to estimate the correct top- k set in our example, it always errs with a constant probability.

To this end we use a tool from information theory: **Fano's inequality**. Essentially it says that if we want to recover a certain parameter, we need to receive a certain amount of “signal” or “information”.

- Assume that a is chosen uniformly from k, \dots, n . Then we sample observations according to the model P_a .

Proof sketch, lower bound (3)

- Fano's inequality now states that any algorithm that estimates a by some \hat{a} has to make an error of at least

$$P(a \neq \hat{a}) \geq 1 - \frac{I(a, \text{observation}) + \log 2}{\log(n - k + 1)}$$

So we need to bound the mutual information $I(a, \text{observation})$, which boils down to a sum of Kullback-Leibler divergences $D(P_a || P_b)$. They can be computed by standard methods.

Details skipped.

Exact recovery of full ranking

The bound for top-k ranking can immediately be turned into a bound of exact full ranking. The main observation is that a ranking is correct if the top-k rankings for all $k = 1, \dots, n - 1$ are correct. This immediately leads to:

Theorem 55 (Upper bound, full permutation)

Let $\hat{\pi}$ be the permutation induced by the estimated scores $\hat{\tau}$, and π the one by the true scores τ . If $\Psi_k(n, r, p_{\text{obs}}) \geq 8$ for all $k = 1, \dots, n - 1$, then $P(\hat{\pi} = \pi) \geq 1 - 1/n^{13}$

Proof: union bound with the previous theorem (union bound leads to power 13 instead of 14).

Approximate recovery

Result looks surprisingly similar. Just the separation term now not depends just on $\tau_k - \tau_{k+1}$, but on all τ -values in a certain neighborhood of k (where the size of the neighborhood depends on the error we are allowed to make).

We still get the same kind of worst case query complexity.

Details skipped.

Discussion

- ▶ On a high level, the theorem shows two things:
 - ▶ Ranking from noisy data is difficult if we don't make any assumptions.
 - ▶ You cannot improve on the counting algorithm — unless you do make more assumptions.
- ▶ In practice, the query complexity of $n^3 \log n$ is completely out of bounds, there is no way you can collect that many comparisons in a realistic setting. So what is obviously needed are algorithms that work well with less queries in realistic settings (assumptions).

Learning to rank

Learning to rank

- ▶ Objects x_1, \dots, x_n .
- ▶ Observations of the form $x_i \prec x_j$. Encode this as follows:
 - ▶ Consider the space S of all unordered pairs of objects.
 - ▶ Output variable $y_{ij} = \begin{cases} +1 & \text{if } x_i \prec x_j \\ -1 & \text{otherwise} \end{cases}$
- ▶ Goal: learn a classifier that makes as few mistakes as possible.

Naive idea: ERM

The first naive algorithm we can think of is to perform empirical risk minimization on the set of permutations: that is we pick the permutation that agrees most with our observations.

We have mentioned already that this is NP hard to do (**computational complexity**), but let's look at how many queries we would need (**query complexity**).

Generalization bounds for learning to rank

Proposition 56 (VC dim of permutations)

Consider a set V of n objects, and the set S of all unordered pairs of objects. Denote by Π the set of permutations of V . Each permutation π induces a classifier $f_\pi : S \rightarrow \{-1, 1\}$ on the set S (as described above). Then the space $\mathcal{F} := \{f_\pi \mid \pi \in \Pi\}$ has VC-dimension $n - 1$.

Proof: Step 1: Prove that $VC < n$. To this end, consider any subset $S' \subset S$ with $|S'| = n$. Want to show that it cannot be shattered by the function class \mathcal{F} . We construct a proof by contradiction.

- Assume we have a set $S' \subset S$ of n pairs of objects that can be shattered by \mathcal{F} .

Generalization bounds for learning to rank (2)

- ▶ Consider the comparison graph of S' : Vertices = all n objects; undirected edge from object i to j if $\{i, j\} \in S'$.
- ▶ The graph has n vertices and n edges by construction, so it needs to contain an undirected cycle. Now observe that **we cannot shatter the pairs in S' that correspond to the edges in the cycle**: we cannot realize the function that corresponds to $x_i \prec x_j \prec \dots \prec x_l \prec x_i$, because the latter implies $x_i \prec x_i$. ⚡

Generalization bounds for learning to rank (3)

Step 2: Prove that $VC \geq n - 1$. To this end, need to find at least one subset $S' \subset S$ with $|S'| = n - 1$ that can be shattered. Using the same construction as above, we simply choose S' such that the graph is a tree. Can always be done. This does it.



Remark: naively, the set Π consists of $n!$ many permutations, so the shattering coefficient is $n!$. The log-shattering coefficient is then $\log(n!) = n \log n$. So the first natural guess is that the VC dim might be $n \log n$. We now see that it is even $n - 1$.

Generalization bounds for learning to rank (4)

Our standard VC-generalization bound for a class with VC-dimension d over a sample of m comparisons is that with probability at least $1 - \delta$, any permutation π satisfies

$$R(f) \leq R_n(f) + 2\sqrt{\frac{d \log(2em/d) - \log(\delta)}{m}}.$$

As a rule of thumb: how many sample points do you need to achieve an error of about ε at most? Here is the argument:

- ▶ Ignore all log terms and constants.
- ▶ Then the error ε is of the order $\varepsilon := \sqrt{d/m}$. Solving for m tells us that we need to observe of the order d/ε^2 comparisons. In our case with $d = n$ this means that we need to observe about $m := n/\varepsilon^2$ comparisons to achieve an error of at most ε , with high probability.

Generalization bounds for learning to rank (5)

Seems pretty good!

Generalization bounds for learning to rank (6)

Comparison to the bound in the Shah/Wainwright approach (simple counting algorithm):

- ▶ Note: the bound in the Shah/Wainwright approach was: recovery works if we see about $n^3 \log n^3$ comparisons (with similar results for approximate recovery and recovery of the full ranking).
- ▶ Now we have a VC bound that says that of the order n examples are enough for good classification performance.
- ▶ Both approaches make only minimalistic / no assumptions whatsoever on the structure of the numbers we want to sort.

WHERE IS THE CATCH?

Generalization bounds for learning to rank (7)

- ▶ Note that the Shah/Wainwright bound talks about **identifying** the correct ranking, while the VC bound just talks about **predicting** the outcome of comparisons.
- ▶ Consider an example that is difficult in the Shah / Wainwright framework: all π_{ij} -values close to $1/2$, so the τ -values are very similar to each other.
 - ▶ Shah/Wainwright: need many samples to find the actual ranking.
 - ▶ Learning to rank: the bound only considers the estimation error of the classifier, when applied to predict unobserved comparisons. [As a side remark: in the given example even the Bayes classifier would have a poor performance close to random guessing. The generalization bound just tells us that we need not so many comparisons to come close to the performance of the actual Bayes classifier.]

Generalization bounds for learning to rank (8)

- So the bounds are difficult to compare, neither the estimation error of the predictor nor its approximation error are directly related to the difficulty of the ranking problem.

SVM ranking

As ERM is infeasible computationally, we could use a linear SVM instead:

- ▶ Encode an ordered pair of objects by a feature vector $x_{ij} := e_i - e_j \in \mathbb{R}^n$ and the outcome y_{ij} as described above.
- ▶ Get training points of the form (x_{ij}, y_{ij}) .
- ▶ Classify using a linear hyperplane (that is, find a vector $w \in \mathbb{R}^n$) such that $\text{sign}(\langle w, z_{ij} \rangle)$ makes as few errors as possible. Use an SVM to find this hyperplane.
- ▶ In particular, the predicted ordering can then be recovered by the ordering of the coordinates of w .

Can also prove margin-type generalization bounds for SVM ranking, skipped.

Application: distance completion problem

This is a topic we are actually working on right now in my research group.

Setup: Triplet comparisons

A scenario beyond simple ranking:

- ▶ Points X_1, \dots, X_n from \mathbb{R}^d
- ▶ We don't know any numeric information such as vector representations or distance values
- ▶ We just get to see binary variables that compare distances:

$$d(X_i, X_j) < d(X_k, X_l) = \text{true or false}$$

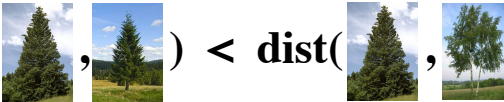
So in the ranking language, we get a partial ranking between the distances of the objects.

Setup: Triplet comparisons (2)

Why is this interesting?

It is often easy to say that things “are pretty similar” or “not similar at all”, but it is hard to come up with good ways to quantify this.

Example user ratings: easier to compare

$$\text{dist}(\text{img1}, \text{img2}) < \text{dist}(\text{img1}, \text{img3})$$


... than to give numeric distance values:

Setup: Triplet comparisons (3)

$$\text{dist}(\text{img1}, \text{img2}) = 0.1$$

$$\text{dist}(\text{img1}, \text{img3}) = 0.7$$

Setup: Triplet comparisons (4)

In the following we consider:

- ▶ Triple questions: $d(X_i, X_j) \stackrel{?}{\leq} d(X_i, X_k)$
- ▶ Quadruple questions: $d(X_i, X_j) \stackrel{?}{\leq} d(X_k, X_l)$

Distance completion problem

Distance comparison problem:

- ▶ n objects from \mathbb{R}^d
- ▶ All we observe are a subset of all triple comparisons of the form $d(X_i, X_j) < d(X_i, X_k)$.
- ▶ Want to estimate the full ranking between all distances d_{ij} .

The full distance ranking can then for example be used to find the nearest neighbors of each data points, and then we can apply classification algorithms, regression algorithms, clustering algorithms, etc.

Query complexity of the distance completion problem

Given n objects, how many randomly chosen triple comparisons do I need in order to estimate the true distance ranking reliably?

Query complexity, first observations

First observations about ranking m objects.

- ▶ There are of the order $m := n^2$ distances. A comparison-based sorting algorithm would need $\Theta(m \log m) \Theta(n^2 \log n^2)$ many (actively chosen !) comparisons. In the noiseless case, it would produce the perfect ranking.
- ▶ If we use ERM to recover an approximate ranking, we would just need $m/\varepsilon^2 = n^2/\varepsilon^2$ many queries to learn the ranking up to error ε (ignoring that the computational complexity is much too high).
- ▶ If we apply the simple counting algorithm by Shah/Wainwright, we also get a query complexity of $m^3 \log m = n^6 \log n$ (of randomly chosen comparisons). Would also work in a noisy case.

Query complexity, first observations (2)

In any application in real-world, query complexities of order n^2 are prohibitive ...

Query complexity, first observations (3)

However:

- ▶ Observe that the three approaches I mentioned above do not make any assumption on the objects that need to be ordered, it can be any arbitrary collection of numbers.
- ▶ We know more about our data: the things we want to order are Euclidean distances. Can we exploit this in some way?

Query complexity, exploit structure

The answer is yes: we can exploit the structure of the problem.

- ▶ Consider the set of points $X_1, \dots, X_n \in \mathbb{R}^d$, and a certain subset of triple questions.
- ▶ Observe that a triple comparison gives a relationship in form of a hyperplane: $d_{ij} < d_{ik}$ is equivalent to saying: if we consider the hyperplane between point X_j and X_k , then point X_i is on the same side as X_j .
- ▶ We can now build equivalence classes of point sets: the ones that satisfy the same hyperplane conditions.
- ▶ It is now possible to “count” the number of equivalence classes (non-trivial!). The result is: there are of the order $2^{dn \log n}$ such equivalence classes (where d is the dimension of the space).
- ▶ This means that the log-shattering coefficient of the set of all Euclidean (!) distance completions is just $dn \log n$.

Query complexity, exploit structure (2)

- ▶ So we the standard shattering-coefficient generalization bound says that with high probability, if we want to approximate the correct distance ranking up to error ε , we need of the order $dn \log n / \varepsilon^2$ many triple questions, close to linear!!!
- ▶ Note that this is much better than the $n^2 \log n$ requirements we had without making any assumption.

This is a very nice example to demonstrate that exploiting the structure in the problem helps (at least in theory).

And this is also a nice example for the type of questions we work in in my group - we just proved this result a couple of weeks ago...

Query complexity, exploit structure (3)

Outlook:

- ▶ Note that while this shows that in theory we only need few triples to recover the full distance ranking for Euclidean points, we don't know how to do it in practice.
- ▶ We would need to have an algorithm that does ERM on the set of equivalence classes ...

Spectral ranking

Based on the following paper:

Fogel, d'Aspremont, Vojnovic: SerialRank: Spectral ranking using seriation. NIPS 2014.

Spectral ranking

Setting as before: we observe pairwise comparison, want to output a ranking.

Define the comparison matrix:

$$C_{ij} = \begin{cases} 1 & \text{if } i \succ j \\ -1 & \text{if } i \prec j \\ 0 & \text{if no data exists} \end{cases}$$

Define a similarity matrix as follows:

$$S_{ij} := \sum_{k=1}^n \frac{1 + C_{i,k}C_{j,k}}{2}$$

(counts the number of matching comparisons of i and j with other items k)

Spectral ranking (2)

SpectralRanking algorithm:

- ▶ Compute the similarity matrix S based on the observed data
- ▶ Construct the unnormalized Laplacian L and compute its second eigenvector.
- ▶ Rank all items according to the corresponding entries in this eigenvector.

Spectral ranking (3)

There are lots of theoretical results on this algorithm:

- ▶ Assume we get to see all pairwise comparisons, answered truthfully, and there are no ties. Then SpectralRanking recovers the correct ranking perfectly.
(not interesting from an algorithmic point of view, we could just do topological sort in this case).
- ▶ Given a comparison matrix for n objects, with at most m corrupted entries (selected uniformly at random). Then if $m = O(\sqrt{\delta n})$, then the SerialRank algorithm will produce the ground truth ranking with probability at least $1 - \delta$.
This is the interesting statement.

Proofs are based on some old work by Atkinson 1998, we skip them.

Google page rank

The setting here is not a pairwise-comparison setting. But no student should leave this university without knowing google page rank, so let's discuss it anyway.

The setting

Want to build a search engine:

- ▶ Query comes in
- ▶ First need to find all documents that match the query
- ▶ Then need to decide which to display on the top of the list. So we need to rank the search results according to their “relevance” .

Early attempts looked at the content of the documents (count how often the keyword occurs, etc).

The new idea by the google founders was to instead look at the link structure of the webpages.

Page Rank

Published by Brin, Page, 1998.

Main idea:

- ▶ A webpage is important if many important links point **to** that page.
- ▶ A link is important if it comes **from** an page that is important.

Results of a search query should then be ranked according to importance.

Page Rank (2)

Given a directed graph $G = (V, E)$, potentially with edge weights s_{ij} , define:

- ▶ Out-degree: $d_{\text{out}}(i) = \sum_{\{k|i \rightarrow k\}} s_{ik}$
- ▶ In-degree: $d_{\text{in}}(j) = \sum_{\{k|k \rightarrow j\}} s_{kj}$

Define the ranking function r for all vertices:

$$r(j) = \sum_{i \in \text{parents}(j)} \frac{r(i)}{d_{\text{out}}(i)} \quad (*)$$

This is an implicit definition. We need to find a way to solve this for $r(j)$, for all j .

Page Rank (3)

Define the matrix A with entries

$$a_{ij} = \begin{cases} 1/d_{\text{out}}(i) & \text{if } i \rightarrow j \\ 0 & \text{otherwise} \end{cases}$$

and the vector r with the relevance scores as entries.

Observe that $(r^t \cdot A)_j = \sum_i r_i a_{ij}$, so we can rewrite $(*)$ as

$$r^t = r^t \cdot A$$

So r is a left eigenvector of A with eigenvalue 1.

The page rank idea consists of ranking vertices according to this eigenvector.

In the following we will consider two things:

Page Rank (4)

- ▶ Interpretation as a random surfer model
- ▶ How to compute the eigenvector.

Random walks on a graph

To give the random surfer interpretation to pagerank, we first need to learn about random walks on a graph:

- ▶ Consider a directed graph $G = (V, E)$ with n vertices.
- ▶ A **random walk** on the graph is a time-homogenous, discrete-time Markov chain. At each point in time, we randomly jump from one vertex to a neighboring vertex. The probability to end in one of the neighbors only depends on the current vertex, not on the past beyond this.
- ▶ It is fully described by **transition matrix** P with entries

$$p_{ij} = P(X_{t+1} = v_j | X_t = v_i).$$

For a weighted graph with similarity edge weights s_{ij} , have

$$p_{ij} = s_{ij}/d_i \text{ and in particular } P = D^{-1}S.$$

Random walks on a graph (2)

Initial distribution:

At time point 0, we start the random walk at a random vertex according to probability row vector $\mu = (\mu_1, \dots, \mu_n)$ with $\mu_i \geq 0$, $\sum \mu_i = 1$. The special case where we start at a deterministic vector corresponds to the case where $\mu = (0, \dots, 0, 1, 0, \dots, 0)$.

Random walks on a graph (3)

k -step distribution:

- ▶ At time $t = 0$, the state distribution is μ , our initial distribution.
- ▶ At time $t = 1$, the distribution is μP .
- ▶ At time $t = 2$, the distribution is $(\mu P)P = \mu P^2$. Note that entry ij of the matrix P^2 describes all possible ways to get with exactly 2 steps from vertex i to j .
- ▶ In general, the matrix P^k describes the k -step transition probabilities.

Random walks on a graph (4)

Stationary distribution:

Intuition: if the random walk runs for a long time, it will converge to an equilibrium distribution. It is called the stationary distribution or the invariant distribution.

Definition of a stationary distribution: If we start in a stationary distribution π and perform one step of the random walk, we have again the stationary distribution. In formulas: π is a stationary distribution of P if

$$\pi P = \pi$$

Random walks on a graph (5)

Convergence to the stationary distribution:

Consider a graph G that is weighted, undirected, connected, and not bipartite. Denote by P the transition matrix. Then:

- ▶ The stationary distribution of P is given as the (normalized) degree vector: $\pi_i = d_i / (\sum_j d_j)$ (CAN YOU SEE WHY?)
- ▶ $\lim_{t \rightarrow \infty} P(X_t = v_i) = \pi_i$
- ▶ The matrix P^t converges to the matrix $\mathbb{1}\pi$ with constant columns π . The speed of convergence depends on the eigengap between the first and second eigenvalue of P :
 - ▶ Largest eigenvalue is always 1, second largest eigenvalue λ_2 satisfies $\lambda_2 < 1$ (note: it might not be real-valued!). Note that right and left eigenvalues are the same, just the eigenvectors differ.
 - ▶ Perron-Frobenius theorem: $P^t = \mathbb{1}\pi + O(n^c |\lambda_2|^t)$.

The random surfer model

Recall the definition of the matrix A for pagerank. Observe:

- ▶ the matrix A is the transition matrix of a random walk on the graph of the internet.
- ▶ the ranking vector r is its stationary distribution.

If done naively, two big problems:

- ▶ dangling nodes (e.g., pdf pages).
- ▶ disconnected components

The random surfer model (2)

Solution to both problems:

we introduce a random restart (“teleportation”):

- ▶ with probability α close to 1, we walk along edges of the graph.
- ▶ With probability $1 - \alpha$, we teleport: we jump to any other random webpage.

Transition matrix is then given as

$$\alpha P + (1 - \alpha) \frac{1}{n} \mathbb{1}$$

where n is the number of vertices and $\mathbb{1}$ the constant one matrix.

The ranking is then the stationary distribution of this matrix.

How to compute it: the power method

Need to compute an eigenvector of a matrix of size $n \times n$ where n is the number of webpages in the internet (2014: one billion webpages).

Computing an eigenvector of a symmetric matrix has worst case complexity of about $O(n^3)$ (and btw, our current matrix is not symmetric).

IS THERE ANY REASON TO BELIEVE THAT THIS MIGHT WORK?

How to compute it: the power method (2)

The simplest way to compute eigenvectors: the power method

- ▶ Let A be any diagonalizable matrix.
- ▶ Goal: want to compute eigenvector corresponding to the largest eigenvalue.
- ▶ Observe: Denote by v_1, \dots, v_n a basis of eigenvectors of matrix A . Consider any vector $v = \sum_i a_i v_i$. Then

$$Av = A\left(\sum_i a_i v_i\right) = \sum_i a_i (Av_i) = \sum_i a_i \lambda_i v_i$$

If we apply A k times, then:

$$A^k v = \sum_i a_i \lambda_i^k v_i = \underbrace{a_1 \lambda_1^k}_{\text{dominates}} \left(v_1 + \underbrace{\sum_{i=2}^n \frac{a_i}{a_1} \frac{\lambda_i^k}{\lambda_1^k} v_i}_{\text{vanishes}} \right)$$

How to compute it: the power method (3)

The Power Method, vanilla version:

- 1 Initialize q_0 by any random vector with $\|q_0\| = 1$
- 2 **while** not converged
- 3 $z^{(k)} := Aq^{(k)}$
- 4 $q^{(k)} := z^{(k)} / \|z^{(k)}\|$

Caveat:

- ▶ Won't work if $q_0 \perp$ first eigenvector
- ▶ Does not necessarily converge if the multiplicity of the largest eigenvalue is larger than 1.
- ▶ Speed of convergence depends on the gap between the first and second eigenvalue, namely λ_2/λ_1 .

How to compute it: the power method (4)

Implementation of page rank is a simple power iteration:

- ▶ We Initialize with constant vector $r = e = (1, \dots, 1)^t$.
- ▶ We iterate until convergence:

$$\begin{aligned}
 r_{k+1}^t &= r_k^t (\alpha A + (1 - \alpha) e v^t) \\
 &= \alpha r_k^t A + (1 - \alpha) \underbrace{r_k^t e}_{=1} v^t \\
 &= \alpha \underbrace{r_k^t A}_{\text{sparse}} + (1 - \alpha) v^t
 \end{aligned}$$

Comments:

- ▶ v is the “personalization vector” (\approx probability over all webpages of whether the surfer would like to see that page)

How to compute it: the power method (5)

- ▶ $1 - \alpha$ is the teleportation parameter.
- ▶ in the last line, we essentially have to perform one sparse matrix-vector multiplication, this can be done in parallel.
- ▶ Speed of convergence depends on the gap between first and second eigenvalue. Personalization adds speed because if the spectrum of P is $\{1, \lambda_2, \lambda_3, \dots\}$, then the spectrum of the personalize matrix is $\{1, \alpha\lambda_2, \alpha\lambda_3, \dots\}$.
Thus we have a tradeoff: α large \leadsto small gap, slow convergence, but structure of the web graph well represented.

Meta ML: How does research work? In general, and in Tübingen

How does research funding work? In general, and in Tübingen

Research funding for university groups

- ▶ Professor's salary is payed by the university
- ▶ Typically, the university provides funds to employ between 1 and 3 PhD students per group.
- ▶ Whenever a group is larger than this, the funding has to be aquired by the professor.
- ▶ It is considered an important part of the work of a professor to aquire such funding, and the ranking of a university depends to a significant amount on the amount of additional funding that is being aquired.

Research funding for university groups (2)

Research funding can come from various sources:

- ▶ **National research funding agencies by the state of Germany.** This is the German Research Foundation, DFG, and the Bundesministerium BMBF. There exist individual grants you can apply for, and larger consortia (Forschergruppe, Sonderforschungsbereich, Exzellenzcluster). It is expected that you publish all your results.
- ▶ **Foundations** such as Volkswagen-Stiftung, Robert-Bosch-Stiftung, Zeiss-Stiftung, Friedrich-Ebert-Stiftung. There exist a large number of such foundations, and many of them act as a “charity” and provide money for (sometimes unusual) research projects. It is expected that you publish all your results.

Research funding for university groups (3)

- ▶ **Industrial funding**, which comes in many different variants. The typical construction is that a company and a professor agree on a joint research project, and then the professor gets the money to employ PhD students. There exist various types of contracts for such projects. In some of them there are restrictions on publications in the sense that any publication would need to be approved by the company. In others, researchers can publish freely.

In all cases, the money is being routed through the university. It never ends on a professor's bank account, nor in other secret accounts.

ML Funding in Tübingen

Institutions:

- ▶ University of Tübingen
- ▶ Max Planck Institute for Intelligent Systems
- ▶ Max Planck Institute for Biological Cybernetics

On the next few slides I introduce the largest funding initiatives in Tuebingen:

ML Funding in Tübingen (2)

Cluster of Excellence: “Machine Learning: New Perspectives for Science”

- ▶ The aim of this cluster is to enable machine learning to take a central role in all aspects of scientific discovery and to understand how such a transformation will impact the scientific approach as a whole.
- ▶ Funded by the DFG (German Research Foundation) out of the excellence initiative
- ▶ Budget: about 35 Mio for 7 years.

ML Funding in Tübingen (3)

Cyber Valley Initiative, <https://cyber-valley.de>

- ▶ Goal: Partners from science and industry are building bridges between curiosity-driven basic research and applied research.
- ▶ Funding to a large part by the public sector, to a smaller part by the industrial partners

ML Funding in Tübingen (4)

Tue.AI Center

- ▶ Mission: We shape the next generation of machine intelligence to develop more robust, efficient and accountable learning systems.
- ▶ Funded by the BMBF (German Ministry for Research)
- ▶ Budget: ramping up, in the steady state about 20 Mio Euros per year, constantly, for funding new professorships and research groups.
- ▶ <https://tuebingen.ai/>

ML Funding in Tübingen (5)

Ellis initiative (European Laboratory for Learning and Intelligent Systems):

- ▶ <https://ellis.eu/>
- ▶ ELLIS is a pan-European AI network of excellence which focuses on fundamental science, technical innovation and societal impact.
- ▶ Funding for the PhD network: locally (no big funding around), this is more a networking initiative.
- ▶ Ellis institute in Tübingen: funded by the Hector foundation, 100 Mio Euros for the next 10 years, to establish professorships.

ML Funding in Tübingen (6)

Max Planck Graduate School for Intelligent Systems

- ▶ <https://imprs.is.mpg.de/>
- ▶ The graduate school in which most of our PhD students are.

How to find a good PhD position?

How to apply in Tuebingen

- ▶ IMPRS (International Max Planck School for Intelligent Systems), covers both MPI and University, Application deadline typically in November
- ▶ Ellis network, a European PhD network, application deadline at some point in fall
- ▶ You can also apply to professors individually

Finding a PhD position

Look at the potential supervisor and find out whether he / she is an active researcher:

(but attention, all these things differ a lot between junior and senior persons, and both of that might be fine!):

- ▶ Your supervisor should have published regularly during the last couple of years, in good conferences / journals.
- ▶ For more senior supervisors, you could look at citation numbers, h-index (not: impact factors, they are bogus!), prizes, whether is he/she in editorial boards, program committee/area chair for conferences,

Finding a PhD position (2)

Note of caution:

If you apply in a junior research group (group leader is not yet a professor), these numbers might not yet tell a lot — but these people can be great supervisors because they still have enough time for a tight supervision, and your fate is also really important to them.

Even if you apply to a senior researcher, these numbers are not always helpful: different persons approach research differently. Some of them write one great paper per year, without few collaborators, some others have millions of collaborations and partners but barely know the contents of all their papers. Also, depending on the field of research these numbers / indicators differ a lot. My main point: find out whether the supervisor is an active researcher.

Finding a PhD position (3)

Look at the group:

- ▶ How large is the group? Only PhD students or also some postdocs?
- ▶ If the group is large, find out who would supervise you (not formally, but on a regular basis), because it won't be the head of the group.
- ▶ Check how much and where all the other PhD students publish.
- ▶ To which conferences the people in the group go regularly.
- ▶ Are there any regular activities going on? Reading groups, seminars, invited guests, ...?

Finding a PhD position (4)

Ask during the interview (questions regarding the general situation):

- ▶ How is the supervision organized in the group?
- ▶ Find out how much freedom the people have in selecting what they want to work on.
- ▶ How much time do people have for research, what other obligations are there (teaching, project work, ...)?
- ▶ How long does a PhD in that group take, usually.
- ▶ Opportunities to travel to conferences, travel money?
- ▶ Ask before the job interview whether you will get the opportunity to talk to another PhD student in the group. Listen to what they say “between the lines”. Ask all the questions above to the head of the group, and once more to the PhD student.

Finding a PhD position (5)

Ask during the interview, regarding your personal position:

- ▶ Where does the funding for your position would come from? If not university / public sector, then ask about constraints regarding publications.
- ▶ Any kind of duties attached to the contract, and if yes, how much? (Teaching, project work,)
- ▶ Who would be your supervisor? How often would you meet?
- ▶ Ultimately, you also need to have the impression that you like the place and get along with your potential supervisor...

Publications and reviewing in ML

Publication culture in Computer Science

Keywords to discuss:

- ▶ Reviewed or not reviewed
- ▶ Typically reviewed: journals, good conferences,
- ▶ Not seriously reviewed: workshop papers, lecture notes, arxiv, technical reports

Publication culture in Computer Science (2)

Top conferences in general Machine Learning: NeurIPS, ICML, ICLR, AISTATS, ...

Conferences dedicated to smaller subfields, for example COLT (learning theory)

Top journal in Machine Learning: JMLR

Top journal in Statistics: Annals of Statistics

As opposed to other scientific fields, Nature or Science are not important in ML.

Reviewing: the process itself

... in journals:

Keywords to discuss:

- ▶ blind, double-blind,
- ▶ editor (in chief, associate)
- ▶ how the whole process works: submission, editor, associate editor, 3 reviewers, associate editor, decision (accept, minor/major revision, reject), notification

Reviewing: the process itself (2)

... in conferences:

Keywords are:

- ▶ program chair = editor in chief
- ▶ area chair = associate editor (mainly in large conferences)
- ▶ programm committee: sometimes this means reviewer, sometimes area chair.
- ▶ Biggest issue: scaling! (NeurIPS 2020: 10.000 submissions, 400 area chairs, need $10.000 * 3 / 6 = 5.000$ Reviewers who reach review 6 papers)
- ▶ How the process works: submission, program chairs, bidding + paper assignment to reviewers, reviews come in, discussion among reviewers, discussion among area chairs/program chairs, decision (accept/reject).

Points to address in a review

Two different parties are addressed in a review:

- ▶ Authors: want constructive and fair feedback about their paper
- ▶ Editors: need arguments for their decision

Points typically addressed in a review:

- ▶ **Quality.** Is the paper technically sound? Are claims well-supported? Is this a complete piece of work, or merely a position paper? Are the authors careful (and honest) about evaluating both the strengths and weaknesses of the work?
- ▶ **Clarity.** Is the paper clearly written? Is it well-organized? Does it adequately inform the reader? (A superbly written paper provides enough information for the expert reader to reproduce its results.)

Points to address in a review (2)

- ▶ **Originality.** Are the problems or approaches new? Is this a novel combination of familiar techniques? Is it clear how this work differs from previous contributions? Is related work adequately referenced?
- ▶ **Significance.** Are the results important? Does the paper address a difficult problem in a better way than previous research? Does it advance the state of the art? Does it provide unique data, unique conclusions on existing data, or a unique theoretical or pragmatic approach?

Points to address in a review (3)

Typical structure of a review:

- ▶ **Summarize** what you believe are the main contributions of the paper, in own words (about 1 paragraph), and summarize your opinion about the paper (few sentences)
- ▶ **Give detailed evaluation:** address the four points (quality, clarity, originality, significance), and summarize pros / cons.
- ▶ **Give minor comments** to the authors (typos, unclear formulations, parts that cannot be understood, wrong formulas, etc)
- ▶ **Private comments to the editor** (not seen by the authors). Here one can declare conflicts of interests, whether one knows the authors, how thoroughly one has done the review (eg, checked proof details).

Mathematical Appendix

Recap: Probability theory

Literature:

- ▶ In general, any book on probability theory
- ▶ On the homepage you can also find the link to a probability recap writeup for a CS course at Stanford University (written by Arian Maleki and Tom Do).

Discrete probability theory

Discrete probability measure

- ▶ Ω = space of “elementary events”, “sample space”.
This space is called “discrete” if it has finitely many elements.
- ▶ “Space of events”: In the discrete case this is simply the power set $\mathcal{P}(\Omega)$ of Ω , that is all possible subsets of Ω .
(In general it is more complicated, the space of events has to be a “ σ -algebra”).
- ▶ **Probability measure:** $P : \mathcal{P}(\Omega) \rightarrow [0, 1]$ such that the following three rules are satisfied (“Axioms of Kolomogorov”)
 - ▶ $P(A) \geq 0$ for all events $A \subset \mathcal{P}(\Omega)$
 - ▶ $P(\Omega) = 1$
 - ▶ “sigma-additivity”: Let $S_1, S_2, \dots \subset \Omega$ be at most countably many disjoint sets. Then $P(S_1 \cup S_2 \cup \dots) = \sum_i P(S_i)$

Note: in the discrete case, the probability measure is uniquely defined on all of $\mathcal{P}(\Omega)$ if we know $P(\omega)$ for all elementary events $\omega \in \Omega$.

Discrete probability measure (2)

Example: throwing a die

- ▶ Elementary events: $\{1, 2, \dots, 6\}$
- ▶ Probability of the elementary events:
 $P(1) = P(2) = \dots = P(6) = 1/6.$
- ▶ Probabilities of all other subsets of Ω can be computed based on the elementary events due to the sigma-additivity.
Example: $P(1, 2, 5) = P(1) + P(2) + P(5) = 3 \cdot 1/6 = 1/2.$

Conditional probabilities

Define the probability of event A under the condition that event B has taken place:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Example with a die: compute the probability $P(\{3\} \mid \text{"uneven"})$.

Solution:

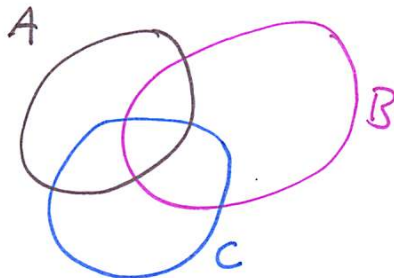
$A = \{3\}$, $B = \{1, 3, 5\}$, $P(A \cap B) = P(\{3\}) = 1/6$, $P(B) = 1/2$,
this implies $P(\{3\} \mid \text{"uneven"}) = (1/6)/(1/2) = 1/3$.

Important formulas

- **Union bound.** Let A_1, \dots, A_k be any events. Then

$$P(A_1 \cup A_2 \cup \dots \cup A_k) \leq \sum_{i=1}^k P(A_i)$$

Intuitive reason:



Important formulas (2)

- **Formula of total probability.** Let B_1, \dots, B_k be a disjoint decomposition of the probability space, that is all B_i are disjoint and $B_1 \cup \dots \cup B_k = \Omega$. Then:

$$P(A) = \sum_{i=1}^k P(A \cap B_i) = \sum_{i=1}^k P(A \mid B_i)P(B_i)$$

Important formulas (3)

- Bayes' formula:

$$P(B \mid A) = \frac{P(B \cap A)}{P(A)} = \frac{P(A \mid B) \cdot P(B)}{P(A)}$$

Example:

The probability that a woman has breast cancer is 1%. The probability that the disease is detected by a mammography is 80 % (true positive rate). The probability that the test detects the disease although the patient does not have it is 9.6% (false positive rate). If a woman at age 40 is tested as positive, what is the probability that she indeed has breast cancer?

Important formulas (4)

Define the following events:

$A :=$ mammography is positive

$B :=$ woman has breast cancer

Given:

- ▶ $P(B) = 0.01$
- ▶ $P(A | B) = 0.80$
- ▶ $P(A | \neg B) = 0.096$
- ▶ Need to compute $P(A)$. Here we use the total probability:

$$\begin{aligned} P(A) &= P(A|B)P(B) + P(A|\neg B)P(\neg B) \\ &= 0.8 \cdot 0.01 + 0.096 \cdot 0.99 = 0.103 \end{aligned}$$

Now we plug this into Bayes theorem and obtain

$$P(B|A) = \frac{0.80 \cdot 0.01}{0.103} = 0.078$$

Random variables

A **random variable** is a function $X : \Omega \rightarrow \mathbb{R}$.

Example:

- ▶ We have 5 red and 5 black balls in an urn
- ▶ We draw 3 balls randomly without replacement
- ▶ Random variable X = number of red balls we got

A **random variable is called discrete** if its image is discrete (it can take at most finitely many values).

Random variables (2)

A random variable $X : \Omega \rightarrow \mathbb{R}$ induces a probability distribution P_X on its image: for any (measurable) set $A \subset \mathbb{R}$ we define

$$P_X(A) = P(X \in A)$$

The measure P_X is called the **distribution of the random variable**.

Important discrete probability distributions

- ▶ **Bernoulli distribution**: we throw a biased coin once. It takes value 1 with probability p and value 0 with probability $(1 - p)$.
- ▶ **Binomial distribution** $B(n, p)$. We throw a biased coin n times independently from each other. The binomial random variable counts how often we got 1. It is defined as

$$P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

It has expected value np and variance $np(1 - p)$.

Important discrete probability distributions (2)

- Poisson distribution $Pois(\lambda)$.

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

The Poisson distribution counts the occurrence of “rare events” in a fixed time interval (like radioactive decay), λ is the intensity parameter.

It has expected value λ and variance λ .

Independence

Two events A, B are called independent if $P(A \cap B) = P(A) \cdot P(B)$.

Note that this implies that $P(A \mid B) = P(A)$.

Two random variables $X, Y : \Omega \rightarrow \mathbb{R}$ are called independent if for all events A, B we have that $P(X \in A, Y \in B) = P(X \in A) \cdot P(Y \in B)$.

Example:

- Throw a coin twice. X = result of the first toss, Y = result of the second toss. These two random variables are independent.

Independence (2)

- Throw a coin twice. X = result of the first toss, Y = sum of the two results. These two random variables are not independent.

Expectation

For a discrete random variable $X : \Omega \rightarrow \{r_1, \dots, r_k\}$ its expectation (mean value) is defined as

$$E(X) := \sum_{i=1}^k r_i \cdot P(\{X = r_i\})$$

Intuition: the expectation is the “average result”, where the results are weighted according to their probabilities.

Examples:

- ▶ We throw a die, X is the result. Then
$$E(X) = \sum_{i=1}^6 i \cdot \frac{1}{6} = 3.5.$$
- ▶ We throw a biased coin, heads occurs with probability p , tails with probability $1 - p$. We assign the random variable $X = 1$ for heads and $X = 0$ for tails. Then
$$E(X) = 0 \cdot (1 - p) + 1 \cdot p = p.$$

Expectation (2)

Important formulas and properties:

- The expectation is linear: for random variables X_1, \dots, X_n and real numbers $a_1, \dots, a_n \in \mathbb{R}$,

$$E\left(\sum_{i=1}^n a_i X_i\right) = \sum_{i=1}^n a_i E(X_i)$$

- Expectation and independence: If X, Y are independent, then

$$E(X \cdot Y) = E(X) \cdot E(Y).$$

Variance

The variance of a random variable is defined as

$$\text{Var}(X) = E((X - E(X))^2) = E(X^2) - (E(X))^2$$

For a discrete random variable with possible values r_1, \dots, r_n , it is given as

$$\text{Var}(X) = \sum_{i=1}^n (r_i - E(X))^2 \cdot P(X = r_i)$$

The variance measures how much the random variable “varies” about its mean.

Variance (2)

Example:

- ▶ We throw a biased coin, heads occurs with probability p , tails comes with probability $1 - p$. We assign the random variable $X = 1$ for heads and $X = 0$ for tails.
- ▶ We have already seen: $E(X) = p$.
- ▶ Now let's compute the variance:

$$\text{Var}(X) = (1 - p)^2 p + (0 - p)^2 (1 - p) = (1 - p)p$$

Important properties of the variance:

- ▶ $\text{Var}(X) \geq 0$.
- ▶ For random variables X and scalars $a, b \in \mathbb{R}$ we have $\text{Var}(aX + b) = a^2 \text{Var}(X)$

Variance (3)

- If X, Y are independent random variables, then

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y).$$

Standard deviation

The standard deviation of a random variable is just the square root of the variance:

$$\text{std}(X) = \sqrt{\text{Var}(X)}$$

Covariance and correlation

The covariance of two real-valued random variables X and Y is defined as

$$\begin{aligned}\text{Cov}(X, Y) &= E((X - E(X))(Y - E(Y))) \\ &= E(XY) - E(X)E(Y)\end{aligned}$$

It provides (one particular) measure of how related the two random variables are: whether we can use a linear (!) function to predict one of them from the other one.

Properties:

- ▶ $\text{Cov}(X, Y) = \text{Cov}(Y, X)$
- ▶ $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2 \text{Cov}(X, Y)$.
- ▶ If $\text{Cov}(X, Y) = 0$, the random variables are called uncorrelated.
- ▶ X, Y independent $\implies X, Y$ uncorrelated (but not vice versa)

Covariance and correlation (2)

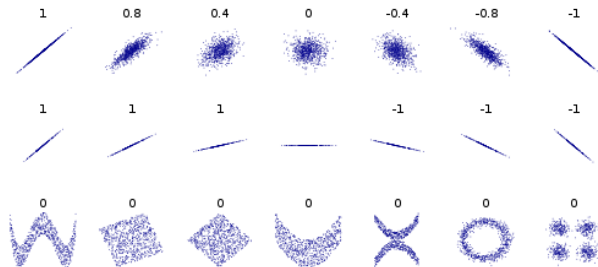
The **correlation coefficient** is defined as

$$\text{Cor}(X, Y) := \rho(X, Y) := \text{Cov}(X, Y) / (\text{std}(X)\text{std}(Y))$$

- ▶ rescales the covariance to a number between -1 and 1
- ▶ $\rho = 1$ iff $Y = aX + b$ for $a > 0, b \in \mathbb{R}$
- ▶ $\rho = -1$ iff $Y = aX + b$ for $a < 0, b \in \mathbb{R}$

Covariance and correlation (3)

Examples (point sets and their correlation coefficient, taken from wikipedia):



Covariance and correlation (4)

(*) Covariance and correlation cares about linear relationships:

- Random variables X, Y
- Find $a, b \in \mathbb{R}$ to linearly estimate Y from X by
 $Y \approx aX + b$
- Measure mean squared error: $E((Y - aX + b)^2)$
- Is minimized if we choose $a = \text{Cov}(X, Y) / \text{Var}(X)$,
 $b = E(Y) - \frac{\text{Cov}(X, Y)}{\text{Var}(X)} \cdot E(X)$

- Their best linear predictor:

$$\hat{Y} = E(Y) + \frac{\text{Cov}(X, Y)}{\text{Var}(X)} (X - E(X))$$

\hat{Y} as linear fct of X , coefficient is given by covariance (correlation)

If variables are standardized to mean $\rightarrow 0$, $\text{Var} = 1$, the formula is:

$$\hat{Y} = \text{Cov}(X, Y) \cdot X$$

(*) Even if Y is a deterministic function of X , the covariance can be 0

Covariance and correlation (5)

Exercise: consider a symmetric random variable X (such that the distribution of X and $-X$ are the same), and define $Y = X^2$.

Then $Cov(X, Y) = 0$

(*) Important inequalities

- **Markov's inequality:** Let X be a non-negative random variable and $t > 0$. Then

$$P(X \geq t) \leq \frac{E(X)}{t}$$

- **Chebyshev's inequality:**

$$P(|X - E(X)| \geq t) \leq \frac{\text{Var}(X)}{t^2}$$

Joint, marginal and product distribution

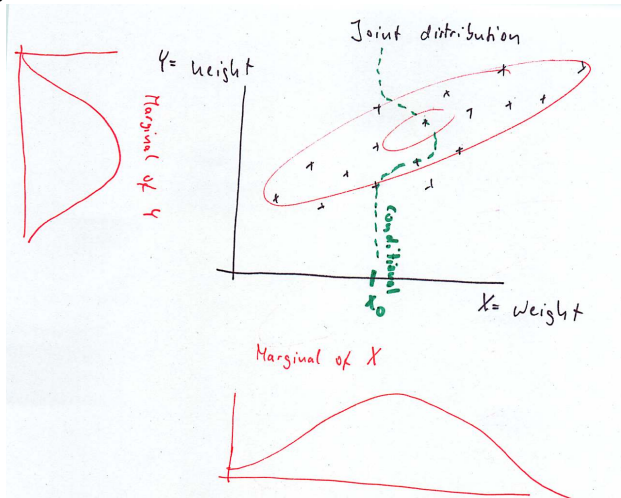
We want to look at the “joint distribution” of two random variables.

Example:

- ▶ We “sample” people: Ω = set of all people
- ▶ X = their weight (in kg), Y = their height (in cm).
- ▶ The **joint distribution** measures how the pair of random variables $(X, Y) : \Omega \rightarrow \mathbb{R}^2$ is distributed.

Joint, marginal and product distribution (2)

- The distribution of X is called the **marginal distribution** of X , similarly for Y .



Joint, marginal and product distribution (3)

- Note that for given marginal distributions, there exist many joint distributions that respect the marginals!

Joint, marginal and product distribution (4)

A particular joint distribution is the **product distribution**: it gives the joint distribution of X and Y if they are independent of each other:

- ▶ Consider two discrete random variables $X, Y : \Omega \rightarrow \mathbb{R}$.
- ▶ Define the product distribution
$$P((X, Y) = (x, y)) = P(X = x) \cdot P(Y = y).$$

The construction works analogously for a product of finitely many spaces.

(*) Conditional independence

Consider three discrete random variables $X, Y, Z : \Omega \rightarrow \mathbb{R}$. We say that X and Y are conditionally independent given Z if

$$\begin{aligned} &P(X \in A, Y \in B \mid Z \in C) \\ &= P(X \in A \mid Z \in C) \cdot P(Y \in B \mid Z \in C) \end{aligned}$$

for all sets $A, B, C \subset \Omega$ with $P(Z \in C) > 0$.

Variance and covariance of multivariate random variables

Variance and covariance for **1-dim random variables** $X \in \mathbb{R}$:

$$\text{Var}(X) = E((X - E(X))^2)$$

$$\text{Cov}(X, Y) = E((X - E(X))(Y - E(Y)))$$

They can be estimated from sample points x_1, \dots, x_n and y_1, \dots, y_n as follows:

$$\bar{x} := 1/n \sum_{i=1}^n x_i$$

$$\hat{\text{Var}}(X) = 1/n \sum_{i=1}^n (x_i - \bar{x})^2$$

Variance and covariance of multivariate random variables (2)

$$\hat{\text{Cov}}(X, Y) = 1/n \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

Note that for variance and covariance, one sometimes normalizes the estimator $\hat{V}\text{ar}$ resp. $\hat{\text{Cov}}$ with the factor of $1/(n-1)$ instead of $1/n$ to achieve an unbiased estimate (we skip this issue here).

Variance and covariance of multivariate random variables (3)

Now consider d -dim random variables: $X = (X^{(1)}, \dots, X^{(d)})'$.

The **expectation** $E(X)$ of a d -dim random variable is the vector that contains the coordinate-wise expectations.

The **overall variance** over all d dimensions is the sum of the variances of the individual dimensions:

$$\text{Var}_d(X) = \sum_{i=1}^d E(\|X^{(i)} - E(X^{(i)})\|^2)$$

The **covariance matrix** of X is a $d \times d$ -matrix C which encodes the covariances between the individual dimensions of the distribution:

Variance and covariance of multivariate random variables (4)

$$C_{kl} = \text{Cov}(X^{(k)}, X^{(l)})$$

EXAMPLE: SHOE SIZE / HEIGHT / AGE OF A PERSON

Variance and covariance of multivariate random variables (5)

These quantities can be estimated from the data:

$$\hat{\text{Var}}_d(X) = 1/n \sum_{k=1}^d \sum_{i=1}^n (x_i^{(k)} - \bar{x}^{(k)})^2$$

$$(\hat{C})_{kl} = \frac{1}{n} \sum_{i=1}^n (x_i^{(k)} - \bar{x}^{(k)})(x_i^{(l)} - \bar{x}^{(l)})$$

- \hat{C} is called the empirical covariance matrix or the sample covariance matrix.

Variance and covariance of multivariate random variables (6)

- ▶ If the data points are centered, and we define matrix \mathbf{X} containing the points as rows, then the empirical covariance matrix \hat{C} coincides with $\mathbf{X}'\mathbf{X}$ because

$$C_{kl} = \sum_{i=1}^n X_i^{(k)} X_i^{(l)} = (\mathbf{X}'\mathbf{X})_{kl}$$

- ▶ In the following, we often drop the “hat” and the word “empirical” ...

(*) Conditional expectation

Example:

- ▶ X, Y two independent throws of a die, $Z = X + Y$.
- ▶ Want to compute the expectation of Z under the condition that X was 3.
- ▶ We write $E(Z \mid X = 3)$

If we don't fix the outcome value of X , then we write $E(Z \mid X)$, this is a random variable (because we don't know the random outcome of X).

Formally, this is a pretty complicated mathematical object. For those who have not seen it before, we just treat it in an intuitive manner.

Continuous probability theory

Probability theory gets more complicated once we go beyond the discrete regime. In this class, we try to keep it on a somewhat intuitive level.

Density and cumulative distribution

Consider a random variable $X : \Omega \rightarrow \mathbb{R}$. We say that X has **density function** $p : \mathbb{R} \rightarrow \mathbb{R}$ if for all (measurable) subsets $A \subset \mathbb{R}$ we have

$$P(X \in A) = \int_A p(x) dx$$



Density and cumulative distribution (2)

- ▶ Intuitively, a density is something like a “continuous histogram”.
- ▶ Sometimes the density is abbreviated as “pdf” (“probability density function”) in the literature.
- ▶ Density functions are always non-negative and integrate to 1. They don’t have to be continuous.
- ▶ Not every random variable can be described by a density, but in this course we won’t discuss this.

(*) Cumulative distribution function

A real-valued random variable can always be described by its **cumulative distribution function** (sometimes abbreviated as “cdf” in the literature).

For a random variable $X : \Omega \rightarrow \mathbb{R}$ it is defined as

$$g : \mathbb{R} \rightarrow \mathbb{R}, \quad g(x) = P(X \leq x)$$



Uniform distribution

The uniform distribution on $[0, 1]$: for $0 \leq a < b \leq 1$ we define

$$P(X \in [a, b]) = b - a$$

Its density is constant.

Normal distribution (univariate)

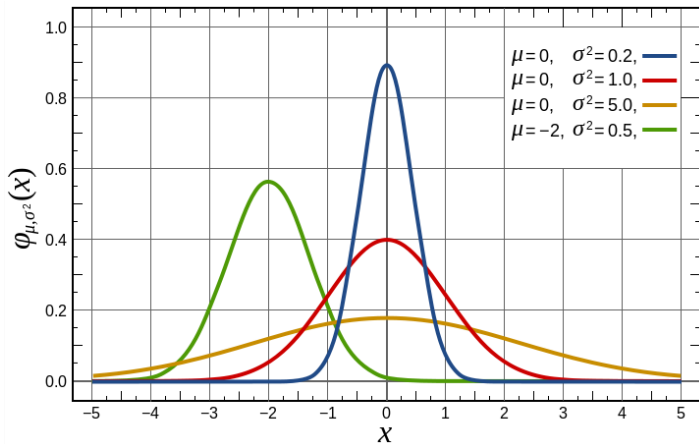
The most important continuous distribution on \mathbb{R} is the normal distribution, abbreviated $N(\mu, \sigma^2)$.

- ▶ It has two parameters: its expectation μ and its variance σ^2 .
 - ▶ μ controls the location of the distribution
 - ▶ σ controls the “width” of the distribution
- ▶ The density function of $N(\mu, \sigma^2)$ is given as

$$f_{\mu, \sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

- ▶ The special case of mean 0 and variance 1 is called the “standard normal distribution”. Sometimes the normal distribution is also called a Gaussian distribution.

Normal distribution (univariate) (2)



Multivariate normal distribution

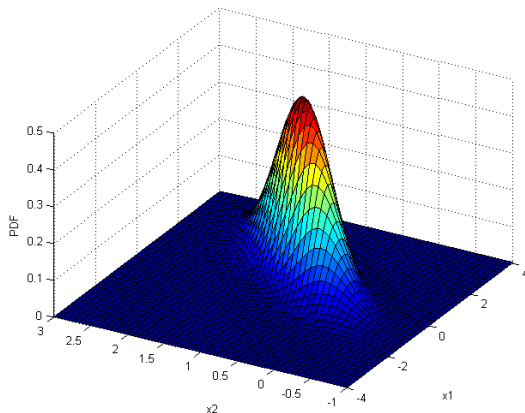
The multivariate normal is defined for the d -dimensional space \mathbb{R}^d , it is abbreviated by $N(\mu, \Sigma)$.

- ▶ It has two parameters: the expectation vector $\mu \in \mathbb{R}^d$, and the covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$. The covariance matrix is always positive semi-definite.
- ▶ The density function is defined as follows:

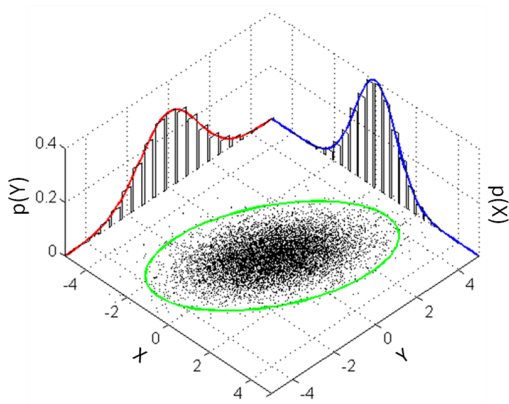
$$f_{\mu, \Sigma}(x) = \frac{1}{\sqrt{2\pi \det(\Sigma)}} \exp\left(-\frac{1}{2}(x - \mu)' \Sigma^{-1}(x - \mu)\right)$$

- ▶ The eigenvectors and eigenvalues of the covariance matrix control the shape of the Gaussian.
- ▶ Each of the marginal distributions is a univariate normal distribution.

Multivariate normal distribution (2)



Multivariate normal distribution (3)



Mixture of Gaussians

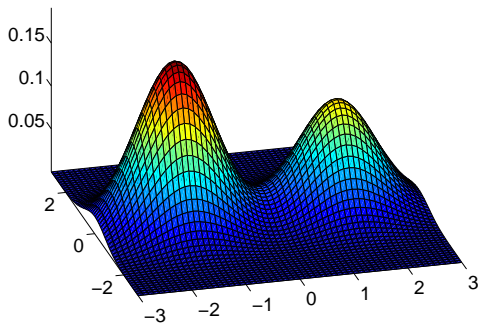
When generating toy data for machine learning applications, one often uses a mixture of Gaussian distributions:

Given mean vectors $\mu_1, \dots, \mu_k \in \mathbb{R}^d$, positive definite covariance matrices $\Sigma_1, \dots, \Sigma_k \in \mathbb{R}^{d \times d}$, and mixing coefficients $\alpha_1, \dots, \alpha_k > 0$ with $\sum \alpha_i = 1$, the density function of the mixture of Gaussians as follows:

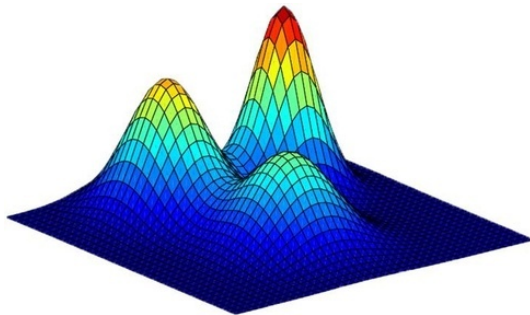
$$f(x) = \sum_{i=1}^k \alpha_i f_{\mu_i, \Sigma_i}$$

Mixture of Gaussians (2)

density



Mixture of Gaussians (3)



Expectation

In the continuous domain, sums are going to be replaced by integrals. For example, the expectation of a random variable X with density function $p(x)$ is defined as

$$E(X) = \int_{\mathbb{R}} x \cdot p(x) dx$$

Recap: Linear algebra

Literature:

- ▶ In general, any introductory book on linear algebra
- ▶ On the homepage you can also find the link to a short linear algebra recap writeup (by Zico Kolter and Chuong Do).

The maths

Vector space

A **vector space** V is a set of “vectors” that supports the following operations:

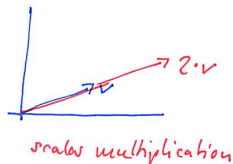
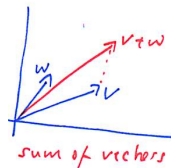
- ▶ We can add and subtract vectors: For $v, w \in V$ we can build $v + w, v - w$
- ▶ We can multiply vectors with scalars: For $v \in V, a \in \mathbb{R}$ we can build av .
- ▶ These operations satisfy all kinds of formal requirements (associativity, commutativity, identity element, inverse element, and so on).

Vector space (2)

Most prominent example: $V = \mathbb{R}^d$.

$$\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{pmatrix} + \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{pmatrix} = \begin{pmatrix} v_1 + w_1 \\ \vdots \\ v_d + w_d \end{pmatrix}$$

$$a \cdot \begin{pmatrix} v_1 \\ \vdots \\ v_d \end{pmatrix} = \begin{pmatrix} a \cdot v_1 \\ \vdots \\ a \cdot v_d \end{pmatrix}$$



Basis

A **basis** of a vector space is a set of vectors $b_1, \dots, b_d \in V$ that satisfies two properties:

- Any vector in V can be written as a linear combination of basis vectors:

For any $v \in V$ there exist $a_1, \dots, a_d \in \mathbb{R}$ such that

$$v = \sum_{i=1}^d a_i b_i$$

- The vectors in the basis cannot be expressed in terms of each other, they are linearly independent:

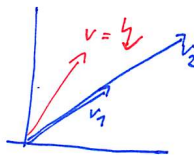
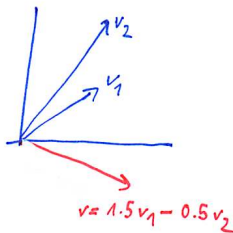
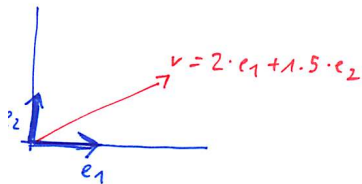
$$\sum_{i=1}^d a_i b_i = 0 \implies a_i = 0 \text{ for all } i = 1, \dots, d.$$

The number of vectors in a basis is called the **dimension** of the vector space.

Basis (2)

Example:

- ▶ $e_1 := (1, 0)$ and $e_2 := (0, 1)$ form a basis of \mathbb{R}^2
- ▶ $v_1 := (1, 1)$ and $v_2 := (1, 2)$ form a basis of \mathbb{R}^2
- ▶ $v_1 := (1, 1)$ and $v_2 := (2, 2)$ do not form a basis of \mathbb{R}^2 .



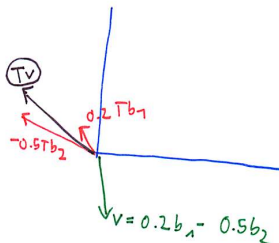
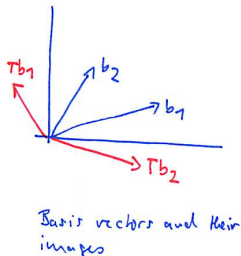
Linear mappings

A **linear mapping** $T : V \rightarrow V$ satisfies

$T(av_1 + bv_2) = aT(v_1) + bT(v_2)$ for all $a, b \in \mathbb{R}$, $v_1, v_2 \in V$.

Typical linear mappings are: stretching, rotation, projections, etc., and combinations thereof.

Note: to figure out what a linear mapping does, it is enough to know what it does on the basis vectors: for $v = \sum_i a_i b_i$ we know by linearity that $T(v) = T(\sum_i a_i b_i) = \sum_i a_i T(b_i)$



Matrices

$m \times n$ -matrix A :

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & & a_{mn} \end{pmatrix}$$

Matrices (2)

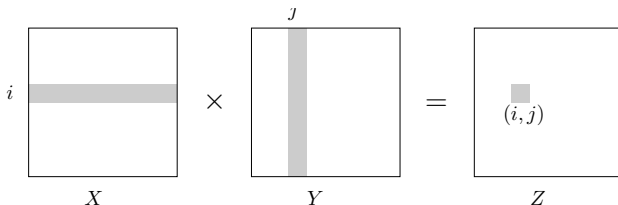
Transpose of a matrix , written as A^t or A' is the matrix where we exchange rows with columns (that is, instead of a_{ij} we have a_{ji}).

Matrices (3)

We can **multiply to matrices** if their “dimensions” fit:

$X = m \times n$ -matrix, $Y = n \times k$ matrix. Then $Z = X \cdot Y$ is a $m \times k$ -matrix with entries

$$z_{ij} = \sum_{s=1}^n x_{is} y_{sj}$$



Matrices (4)

Special case where Y is a vector of length $n \times 1$ is called **matrix-vector-multiplication**:

$$z = Xy \text{ with } z_i = \sum_j x_{ij}y_j$$

Linear mappings correspond to matrices

Linear mappings correspond to **matrices**:

Intuition: the columns of the matrix contain the images of the basis vectors:

Basis e_1, \dots, e_d , linear mapping T

\leadsto corresponding matrix: $\begin{pmatrix} | & | & & | \\ T e_1 & T e_2 & \dots & T e_d \\ | & | & & | \end{pmatrix} =: A$

- ▶ Matrix-vector multiplication is then the same as applying the mapping to the vector.
- ▶ Multiplication of two matrices is the same as applying the mappings one after the other.

The rank of a matrix

Many equivalent definition: The **rank** of a matrix is ...

- ▶ ... the largest number of independent columns in the matrix
- ▶ ... the largest number of independent rows in the matrix
- ▶ ... the dimension of the image space of the linear mapping that corresponds to the matrix
- ▶ ... in case the matrix is symmetric: the rank is the number of non-zero eigenvalues of the matrix (see below).

A $n \times n$ -matrix is said to have **full rank** if it has rank n .

A $n \times n$ -matrix is said to have **low rank** if its rank is “small” compared to n (this is not a formal definition, it is often used informally).

Inverse of a matrix

- ▶ For some matrices A we can compute the **inverse matrix** A^{-1} . It is the unique matrix that satisfies

$$A \cdot A^{-1} = A^{-1} \cdot A = Id$$

where Id is the identity matrix (1 on the diagonal, 0 everywhere else).

- ▶ A matrix is called invertible if it has an inverse matrix.
- ▶ A square matrix is invertible if and only if it has full rank.

Norms and scalar products

Some vector spaces have additional structure: norms or even scalar products. In particular, this is true for \mathbb{R}^d .

Given two vectors $v = (v_1, \dots, v_n)^t$ and $w = (w_1, \dots, w_n)^t \in \mathbb{R}^n$, their **scalar product** is defined as $\langle v, w \rangle = \sum_{i=1}^n v_i w_i$.

The **norm** $\|v\|$ of a vector $v \in \mathbb{R}^d$ is defined as $\|v\|^2 = \langle v, v \rangle$.

Intuition:

- ▶ The scalar product is related to the angle between the two vectors:
 - ▶ $\langle v, w \rangle = 0 \iff v \perp w$ (vectors are orthogonal)
 - ▶ If v and w have norm 1, then $\langle v, w \rangle$ is the cosine of the angle between the two vectors.
- ▶ The norm is the length of a vector.

Norms and scalar products (2)

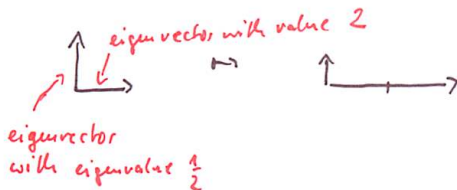
A matrix A is called **orthogonal** if all its columns are orthogonal to each other. It is called **orthonormal** if additionally, all its columns have norm 1.

For orthogonal matrices, we always have $A^t = A^{-1}$.

Eigenvalues and eigenvectors

A vector $v \in \mathbb{R}^n, v \neq 0$ is called an **eigenvector** of $A \in \mathbb{R}^{n \times n}$ with **eigenvalue** λ if $Av = \lambda v$.

Intuition: in the direction of v , the linear mapping corresponding to A is stretching by factor λ .



Taken together, all eigenvectors with eigenvalue λ form a subspace called the **eigenspace** associated to eigenvalue λ . The dimension of this subspace is called the **geometric multiplicity** of λ .

Eigenvalues and eigenvectors (2)

Just for completeness:

Eigenvectors can also be defined as the roots of the **characteristic polynomial** $f(\lambda) := \det(A - \lambda I) \stackrel{!}{=} 0$. The degree of this polynomial is d (the dimension of the space). The multiplicity of this root is called the **algebraic multiplicity of λ** .

The algebraic multiplicity is always larger or equal to the geometric one. In case of strict inequality, the matrix cannot be diagonalized.

Simple example where the two multiplicities do not agree: the nilpotent matrix $\begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ has eigenvalue 0 with geometric multiplicity 1, but algebraic multiplicity 2. It cannot even be diagonalized over \mathbb{C} .

Eigenvalue decomposition of a symmetric matrix

Diagonalization:

- ▶ A matrix A is called **diagonalizable** if there exists a basis of eigenvectors.
- ▶ In this case, we can write the matrix in the form

$$A = VDV^t$$

where V is an orthonormal matrix that contains the eigenvectors as columns, and D is a diagonal matrix containing the eigenvalues.

- ▶ One can also write the matrix in the form

$$A = \sum_{i=1}^d \lambda_i v_i v_i^t$$

where λ_i are the eigenvalues and v_i the eigenvectors.

Eigenvalue decomposition of a symmetric matrix

(2)

- Intuitively, a matrix is diagonalizable if it performs “Strecken und Spiegeln”, but no rotation.

Symmetric matrices are always diagonalizable and have real-valued eigenvalues. Their eigenvectors (of different eigenvalues) are always perpendicular to each other

(*) Singular Value Decomposition (SVD)

If a matrix is not square, we cannot compute eigenvalues. But there exists a closely related concept, the singular values:

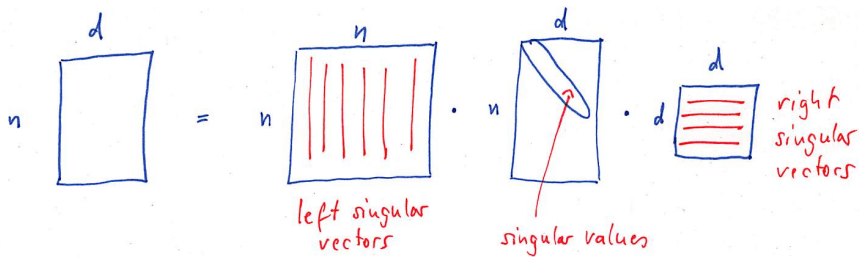
Any matrix $\Phi \in \mathbb{R}^{n \times d}$ can be decomposed as follows:

$$\Phi = U \Sigma V^t$$

where

- ▶ $U \in \mathbb{R}^{n \times n}$ is orthogonal. Its columns are called **left singular vectors**.
- ▶ $\Sigma \in \mathbb{R}^{n \times d}$ is a diagonal matrix containing the **singular values** $\sigma_1, \dots, \sigma_d$ on the diagonal
- ▶ $V \in \mathbb{R}^{d \times d}$ is an orthogonal matrix. Its columns are called **right singular vectors**.

(*) Singular Value Decomposition (SVD) (2)



$$\Phi = U \cdot \Sigma \cdot V^t$$

(*) Singular Value Decomposition (SVD) (3)

There is a close relation between the singular values of Φ and the eigenvalues of the (symmetric!) matrices $\Phi\Phi^t$ and $\Phi^t\Phi$:

- ▶ The left singular vectors of Φ are the eigenvectors of $\Phi\Phi^t$.

CAN YOU SEE WHY?

$$\Phi\Phi^t = (U\Sigma V^t)(U\Sigma V^t)^t = U\Sigma VV^t\Sigma U^t = U\Sigma^2 U^t.$$

- ▶ The right singular vectors of Φ are the eigenvectors of $\Phi^t\Phi$.
- ▶ The non-zero singular values of Φ are the square roots of the non-zero eigenvalues of both $\Phi^t\Phi$ and $\Phi\Phi^t$.

(*) Singular Value Decomposition (SVD) (4)

Note in particular:

- ▶ An SVD exists for any matrix!
- ▶ The singular values are unique.
- ▶ The singular vectors are “as unique” as in an eigenvector decomposition (that is, up to scalar multiplication, and in case of higher multiplicity the singular vectors span a whole space).

Positive Definite Matrices

A symmetric matrix A is called **positive semi-definite** if all its eigenvalues are ≥ 0 . In case of strict inequality it is called **positive definite**.

Equivalent formulations:

- ▶ Positive definite $\iff v^t A v > 0$ for all $v \in \mathbb{R}^n \setminus \{0\}$.
- ▶ Positive semi-definite $\iff v^t A v \geq 0$ for all $v \in \mathbb{R}^n \setminus \{0\}$.
- ▶ Positive semi-definite \iff we can decompose the matrix in the form $A = X X^t$.

(*) Generalized inverse

Consider a symmetric matrix $A \in \mathbb{R}^{d \times d}$.

- Let $\lambda_1, \dots, \lambda_d$ the eigenvalues and v_1, \dots, v_d a corresponding set of eigenvectors of A . We can write A in the spectral decomposition as

$$A = \sum_{i=1}^d \lambda_i v_i v_i^t$$

- In case the matrix has rank d , all its eigenvalues are non-zero. Then we can write the inverse of A as

$$A^{-1} = \sum_{i=1}^d \frac{1}{\lambda_i} v_i v_i^t$$

(*) Generalized inverse (2)

- In case the matrix is not of full rank, it is not invertible. However, we can define the **Moore-Penrose generalized inverse** as

$$A^+ := \sum_{i: \lambda_i \neq 0} \frac{1}{\lambda_i} v_i v_i^t$$

(intuitively, this is the inverse of the matrix A restricted to the subspace orthogonal to its nullspace).

(*) Generalized inverse (3)

Properties of the generalized inverse:

In general we don't have that $AA^+ = I$ or $A^+A = I$.

But we have the following slightly weaker properties:

- ▶ $AA^+A = A$ and $A^+AA^+ = A^+$
- ▶ $(A^+)^+ = A$
- ▶ A^+A and AA^+ are both symmetric.
- ▶ If A is invertible, then $A^{-1} = A^+$.
- ▶ AA^+ is an orthogonal projection on the $\text{ran}(A)$ (the image of the matrix A), and A^+A is an orthogonal projection on $\text{ran}(A^t)$.

(*) Generalized inverse (4)

Some intuition:

Consider a linear operator A that is a projection on some lower-dimensional subspace. As example, consider the projection of the three-dim space to the two-dim plane:

$$A(x_1, x_2, x_3)^t := (x_1, x_2)^t \in \mathbb{R}^2$$

Call the projection A and consider a “reconstruction” operator A^{rec} .

- Note that from the result of the projection, it is impossible to reconstruct the original point exactly (this is why the matrix A is not invertible).
- However, I can reconstruct another point that would give the same projection result: for example, I can simply define

$$A^{rec}(x_1, x_2) := (x_1, x_2, 17) \in \mathbb{R}^3$$

(*) Generalized inverse (5)

- Note that if I apply the projection again after reconstruction, I get the same result as after the first projection: I have

$$AA^{rec}A = A$$

The Moore-Penrose pseudoinverse is one particular such reconstruction operator.

(*) Rayleigh principle

Proposition 57 (Rayleigh principle)

Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix with eigenvalues $\lambda_1 \geq \dots \geq \lambda_n$ and eigenvectors v_1, \dots, v_n . Then

$$\lambda_1 = \max_{v \in \mathbb{R}^n} \frac{v^t A v}{\|v\|^2} = \max_{v \in \mathbb{R}^n: \|v\|=1} v^t A v.$$

The eigenvector v_1 is the vector for which this maximum is attained. Moreover,

$$\lambda_{k+1} = \max_{v \perp \{v_1, \dots, v_k\}: \|v\|=1} v^t A v.$$

This theorem holds analogously for minimization problems. In this case, the solution is given by the smallest eigenvalue / vector.

(*) Rayleigh principle (2)

Proof intuition.

- ▶ Let λ be any eigenvalue with eigenvector v . Then $v^t A v = v^t (\lambda v) = \lambda$ (because $v^t v = 1$).
- ▶ So among all eigenvectors v_1, \dots, v_n , the eigenvector v_1 leads to the largest value λ_1 .
- ▶ Now consider an arbitrary unit vector $w \in \mathbb{R}^n$. Because A is symmetric, there exists a basis of eigenvectors v_1, \dots, v_n . In particular, there exist coefficients c_i such that $w = \sum_i c_i v_i$ and $\|c\| = 1$.
- ▶ Then $w^t A w = \dots = \sum_{i,j=1}^n c_i c_j v_i^t A v_j$
- ▶ But for $i \neq j$ we get $v_i^t A v_j = v_i^t \lambda v_j = 0$ (because different eigenvectors are perpendicular to each other).
- ▶ so $w^t A w = \sum_i c_i^2 v_i^t A v_i = \sum_i c_i^2 \lambda_i$.

(*) Rayleigh principle (3)

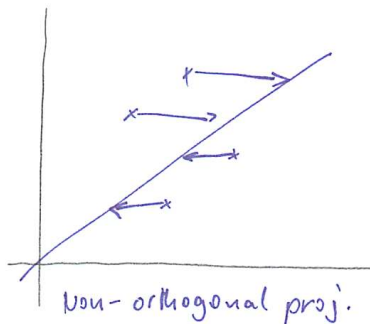
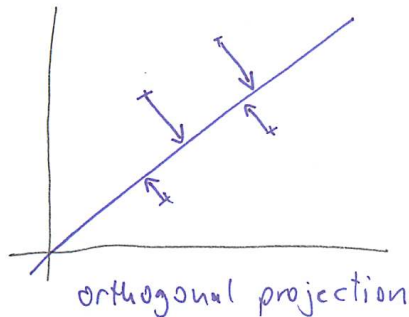
- ▶ Among all c with $\|c\| = 1$, the maximum of this expression is attained for $c_1 = 1, c_2 = \dots = c_n = 0$.



Projections

A linear mapping $P : E \rightarrow E$ between vector spaces is a **projection** if and only if $P^2 = P$.

It is an **orthogonal projection** if and only if it is a projection and $\text{nullspace}(P) \perp \text{image}(P)$.



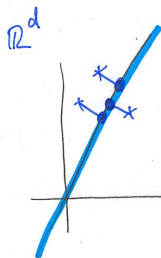
Projections (2)

We always have two points of view of a projection:

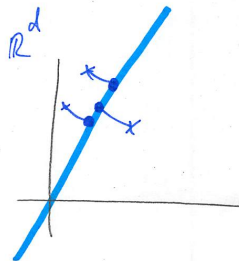
View 1: represent the projected points still as elements of the original space, that is $P : \mathbb{R}^d \rightarrow \mathbb{R}^d$.

View 2: Represent the projected points just as elements of the low-dim space, that is $\pi : \mathbb{R}^d \rightarrow \mathbb{R}^\ell$

Projections (3)



view 1:
 $P: \mathbb{R}^d \rightarrow \mathbb{R}^d$



view 2:
 $\pi: \mathbb{R}^d \rightarrow \mathbb{R}^L$



Projections (4)

View 2, Projection on a one-dimensional subspace:

The orthogonal projection on a one-dimensional space spanned by vector a can be expressed as

$$\pi : \mathbb{R}^d \rightarrow \mathbb{R}, \pi(x) = a^t x$$

View 2, Projection on an ℓ -dimensional subspace:

Want to project on an ℓ -dim subspace S with ONB v_1, \dots, v_ℓ .

Define the matrix V with the vectors v_1, \dots, v_ℓ as columns. Then compute the low-dim representation as

$$\pi : \mathbb{R}^d \rightarrow \mathbb{R}^\ell, x \mapsto V^t x$$

View 1:

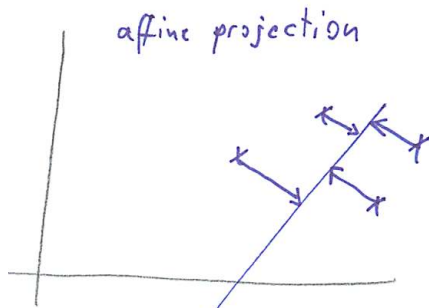
Define $P := VV^t$ (with V as above) and set

$$\pi : \mathbb{R}^d \rightarrow \mathbb{R}^d, x \mapsto Px$$

Projections (5)

Affine projections:

Linear projections always map 0 to 0. If we want to perform an orthogonal projection on an affine (= shifted) space $\tilde{S} = S + \mu$, we need to express the mapping as $Tx = P(x - \mu) + \mu$.



(*) Matrix norms

There are many different ways in which one can define a norm of a matrix:

- **Operator norm, spectral norm:** Consider a matrix as linear operator on a normed vector space V with norm $\|\cdot\|$. Then define

$$\|A\| = \sup_{\{x \in V; \|x\|=1\}} \|Ax\|$$

If A is normal (that is, $AA^* = A^*A$), then the operator coincides with

$$\max\{|\lambda|; \lambda \text{ eigenvalue of } A\}.$$

This is sometimes called the **spectral norm**.

(*) Matrix norms (2)

- Frobenius norm, Hilbert-Schmidt norm, nuclear norm:

$$\|A\|_F := \sqrt{\sum_{ij} a_{ij}^2} = \sqrt{\text{trace}(A * A)} = \sqrt{\sum_i \sigma_i^2}$$

where σ_i are the singular values of the matrix.

Some numerical procedures you should know

The power method

- ▶ Let A be any diagonalizable matrix.
- ▶ Goal: want to compute eigenvector corresponding to the largest eigenvalue.
- ▶ Observe: Denote by v_1, \dots, v_n a basis of eigenvectors of matrix A . Consider any vector $v = \sum_i a_i v_i$. Then

$$Av = A\left(\sum_i a_i v_i\right) = \sum_i a_i (Av_i) = \sum_i a_i \lambda_i v_i$$

If we apply A k times, then:

$$A^k v = \sum_i a_i \lambda_i^k v_i = \underbrace{a_1 \lambda_1^k}_{\text{dominates}} \left(v_1 + \underbrace{\sum_{i=2}^n \frac{a_i}{a_1} \frac{\lambda_i^k}{\lambda_1^k} v_i}_{\text{vanishes}} \right)$$

The power method (2)

The Power Method, vanilla version:

- 1 Initialize q_0 by any random vector with $\|q_0\| = 1$
- 2 **while** not converged
- 3 $z^{(k)} := Aq^{(k)}$
- 4 $q^{(k)} := z^{(k)} / \|z^{(k)}\|$

Caveat:

- ▶ Won't work if $q_0 \perp$ first eigenvector
- ▶ Does not necessarily converge if the multiplicity of the largest eigenvalue is larger than 1.
- ▶ Speed of convergence depends on the gap between the first and second eigenvalue, namely λ_2/λ_1 .
- ▶ Can be implemented efficiently if matrix is sparse.

Excursion to convex optimization: primal, dual, Lagrangian

Literature:

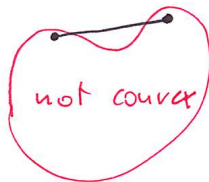
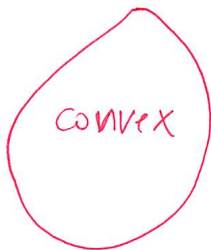
- ▶ Appendix E in the book by Bishop
- ▶ Section 6.3 in the book by Schölkopf / Smola
- ▶ Your favorite book on convex optimization, for example:
 - ▶ Boyd, S. and L. Vandenberghe. Convex Optimization. Cambridge University Press, 2004. Comprehensive, yet easy to read.

Convex optimization problems: intuition

Convex optimization problems

Convex sets:

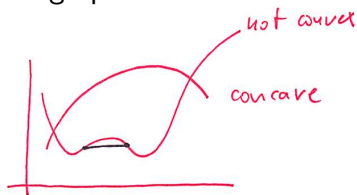
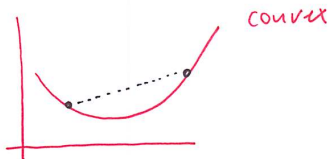
- ▶ A subset S of a vector space is called convex if for all $x, y \in S$ and for all $t \in [0, 1]$ it holds that $tx + (1 - t)y \in S$.
- ▶ In words: for any two points $x, y \in S$, the straight line connecting these two points is contained in the set S .



Convex optimization problems (2)

Convex functions:

- ▶ A function $f : S \rightarrow \mathbb{R}$ that is defined on a convex domain S is called convex if for all $x, y \in S$ and $t \in [0, 1]$ we have $f(tx + (1 - t)y) \leq tf(x) + (1 - t)f(y)$.
- ▶ Intuitively, this means that if we look at the graph of the function and we connect two points of this graph by a straight line, then this line is always above the graph.



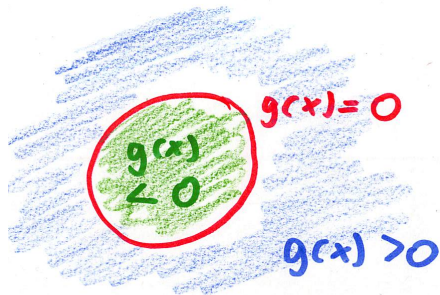
Convex optimization problems (3)

Examples:

- ▶ functions of one variable that are twice differentiable are convex iff their second derivative is non-negative.
- ▶ Functions of several variables that are twice differentiable are convex if their Hessian matrix is positive (semi)-definite.

Convex optimization problems (4)

Observe: For convex functions g , the sublevel sets of the form $\{x | g(x) \leq 0\}$ are convex.



(Funnily, this is not true the other way round: you can have all sublevel sets convex, but yet the function is not convex.)

Convex optimization problems (5)

Convex optimization problem:

- An optimization problem of the the form

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0 \quad (i = 1, \dots, k)\end{array}$$

is called convex if the functions f , g_i are convex.

- Sometimes one also considers equality constraints of the form $h_j = 0$. They can always be replaced by two inequality constraints $h_j \leq 0$ and $-h_j \leq 0$. However, the only situation in which h_j and $-h_j$ are both convex occurs if h is a linear function.
- Convex optimization problems have the desirable property that any local minimum is already a global minimum.

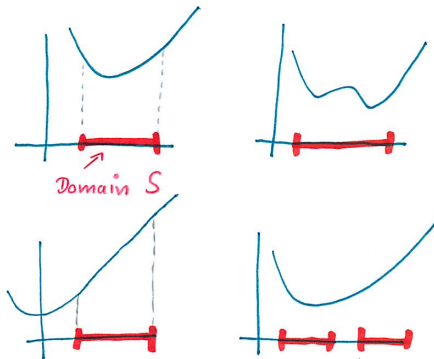
Convex optimization problems (6)

Important terms:

- ▶ The function f over which we optimize is called the **objective function**
- ▶ The functions g_i are called the **constraints**.
- ▶ The set of points where all constraints are satisfied is called the **feasible set**.

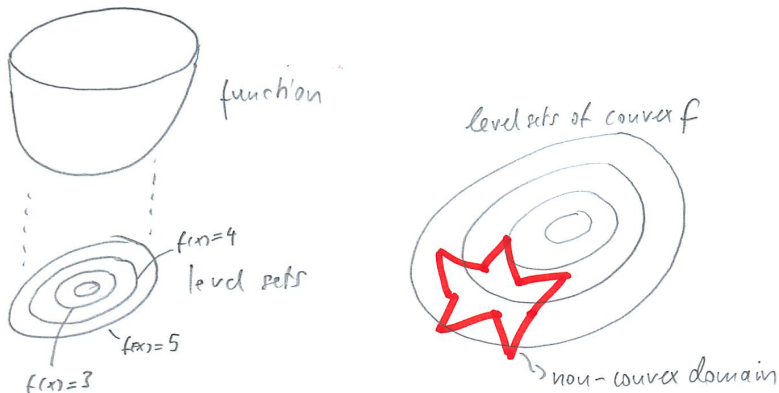
Convex optimization problems (7)

Convex optimization without constraints: which problem is convex?



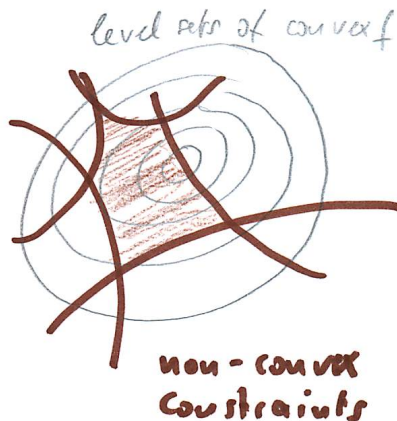
Convex optimization problems (8)

Convex optimization without constraints: the problem with non-convex domains:



Convex optimization problems (9)

Convex optimization with non-convex constraints:



Lagrangian: intuitive point of view

Convex optimization problem

We now want to derive a “recipe” by which many convex optimization problems can be analyzed / rewritten / solved. We don’t consider formal proofs, but just derive the concepts in an intuitive way.

In particular, for the ease of presentation assume that all functions are continuously differentiable (all statements hold in more general settings as well, but one would need convex analysis for this).

Recap: gradient of a function

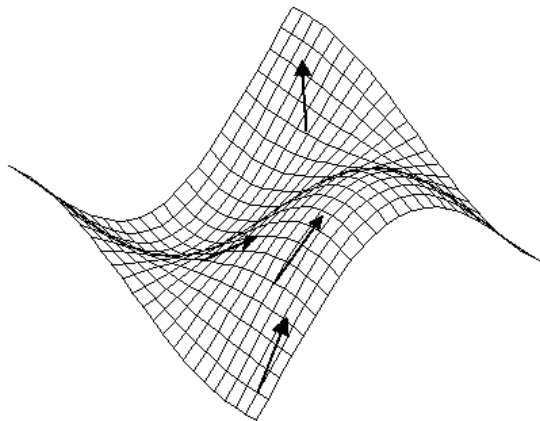
Consider a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$.

- ▶ The **gradient** of f is the vector of partial derivatives:

$$\nabla f(x) = (\partial/\partial x_1, \dots, \partial/\partial x_d)'(x)$$

- ▶ For each x , the gradient $\nabla f(x)$ points in the direction where the function increases most:

Recap: gradient of a function (2)



Gradient Vectors Shown at Several Points on the
Surface of $\cos(x) \sin(y)$

Lagrange multiplier for equality constraints

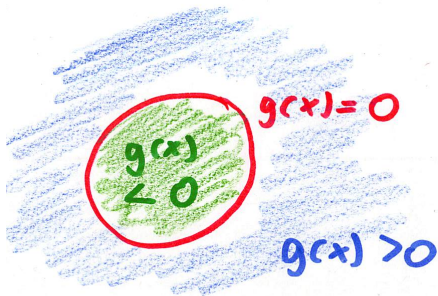
Consider the following convex optimization problem:

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{subject to} & g(x) = 0\end{array}$$

where f and g are convex.

Lagrange multiplier for equality constraints (2)

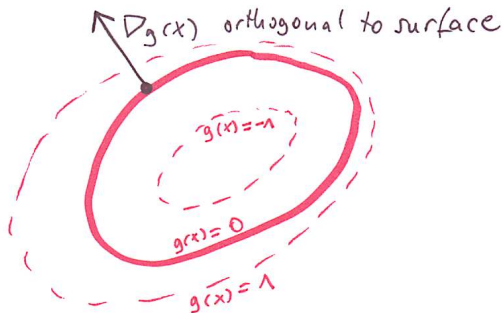
Recall: if g is convex, then its sublevel-sets are convex:



Sublevel set: $\{x | g(x) \leq c\}$ (the green set in the figure)

Lagrange multiplier for equality constraints (3)

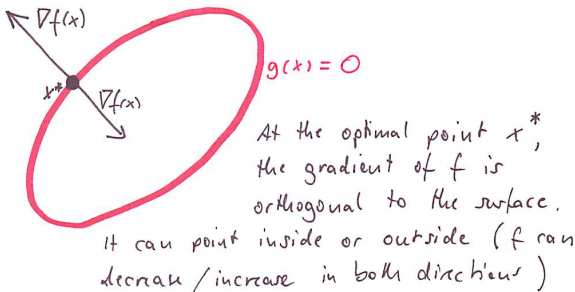
Gradient (equality constraint): For any point x on the “surface” $\{g(x) = 0\}$ the gradient $\nabla g(x)$ is orthogonal to the surface itself.



Intuition: to increase / decrease $g(x)$, you need to move away from the surface, not walk along the surface.

Lagrange multiplier for equality constraints (4)

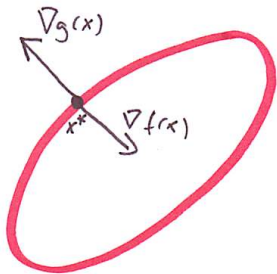
Gradient (objective function): Consider the point x^* on the surface $\{g(x) = 0\}$ for which $f(x)$ is minimized. This point must have the property that $\nabla f(x)$ is orthogonal to the surface.



Intuition: otherwise we could move a little along the surface to decrease $f(x)$.

Lagrange multiplier for equality constraints (5)

Consequence: at the optimal point, $\nabla g(x)$ and $\nabla f(x)$ are parallel, that is there exists some $\nu \in \mathbb{R}$ such that $\nabla f(x) + \nu \nabla g(x) = 0$.



At the optimal point x^* , ∇g and ∇f are parallel to each other (and can point either in the same or the opposite direction).

Lagrange multiplier for equality constraints (6)

We now define the **Lagrangian** function

$$L(x, \nu) = f(x) + \nu g(x)$$

where $\nu \in \mathbb{R}$ is a new variable called **Lagrange multiplier**. Now observe:

- ▶ The condition $\nabla f(x) + \nu \nabla g(x) = 0$ is equivalent to $\nabla_x L(x, \nu) = 0$
- ▶ The condition $g(x) = 0$ is equivalent to $\nabla_\nu L(x, \nu) = 0$.

To find an optimal point x^* we need to find a **saddle point** of $L(x, \nu)$, that is a point such that both $\nabla_x L(x, \nu)$ and $\nabla_\nu L(x, \nu)$ vanish.

Simple example

Consider the problem to minimize $f(x)$ subject to $g(x) = 0$, where $f, g : \mathbb{R}^2 \rightarrow \mathbb{R}$ are defined as

$$f(x_1, x_2) = x_1^2 + x_2^2 - 1$$

$$g(x_1, x_2) = x_1 + x_2 - 1$$

Observe: it is hard to solve this problem by naive methods because it is unclear how to take care of the constraints!

Solution by the Lagrange approach:

Write it in the standard form:

$$\text{minimize } x_1^2 + x_2^2 - 1$$

$$\text{subject to } x_1 + x_2 - 1 = 0$$

Simple example (2)

The Lagrangian is

$$L(x, \nu) = \underbrace{x_1^2 + x_2^2 - 1}_{f(x_1, x_2)} + \nu \underbrace{(x_1 + x_2 - 1)}_{g(x_1, x_2)}$$

Now compute the derivatives and set them to 0:

$$\nabla_{x_1} L = 2x_1 + \nu \stackrel{!}{=} 0$$

$$\nabla_{x_2} L = 2x_2 + \nu \stackrel{!}{=} 0$$

$$\nabla_{\nu} L = x_1 + x_2 - 1 \stackrel{!}{=} 0$$

If we solve this linear system of equations we obtain $(x_1^*, x_2^*) = (0.5, 0.5)$.

Lagrange multiplier for inequality constraints

Consider the following convex optimization problem:

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{subject to} & g(x) \leq 0\end{array}$$

where f and g are convex.

We now distinguish two cases: constraint is “active” or “inactive”:

Lagrange multiplier for inequality constraints (2)

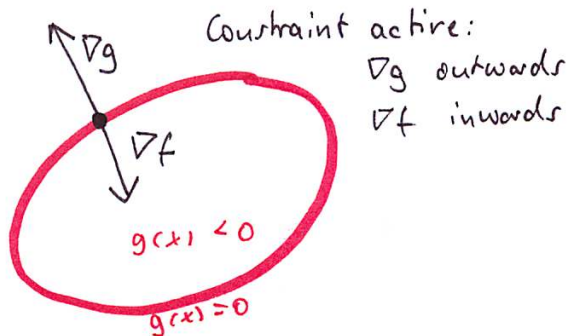
Case 1: Constraint is “active”, that is the optimal point is *on* the surface $g(x) = 0$.

Again ∇f and ∇g are parallel in the optimal point.

But furthermore, the direction of derivatives matters:

- ▶ The derivative of g points outwards (at any point on the surface $g = 0$). This is always the case if g is convex.
- ▶ Then the derivative of f is directed inwards (otherwise we could decrease the objective by walking inside).

Lagrange multiplier for inequality constraints (3)



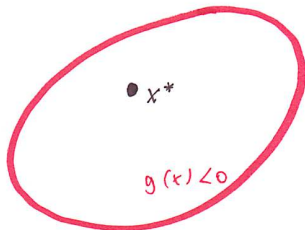
So we have $\nabla f(x) = -\lambda \nabla g(x)$ for some value $\lambda > 0$.

Lagrange multiplier for inequality constraints (4)

Case 2: Constraint is “inactive”, that is the optimal point is not on the surface $g(x) = 0$ but somewhere in the interior.

- ▶ Then we have $\nabla f = 0$ at the solution (otherwise we could decrease the objective value).
- ▶ We do not have any condition on ∇g (it is as if we would not have this constraint).

Constraint inactive. No condition on ∇g



Lagrange multiplier for inequality constraints (5)

We can summarize both cases using the Lagrangian again. We now define the Lagrangian

$$L(x, \lambda) = f(x) + \lambda g(x)$$

where the Lagrange multiplier has to be positive: $\lambda \geq 0$.

- ▶ Case 1: constraint active, $\lambda > 0$.
 - ▶ Need to find a saddle point: $\nabla_x L(x, \lambda) = \nabla_\lambda L(x, \lambda) = 0$.
- ▶ Case 2: constraint inactive, $\lambda = 0$.
 - ▶ Then $L(x, \lambda) = f(x)$. Hence $\nabla_x L(x, \lambda) = \nabla_x f(x) \stackrel{!}{=} 0$,
 $\nabla_\lambda L(x, \lambda) \equiv 0$.
- ▶ So in both cases we have again a saddle point of the Lagrangian.

Lagrange multiplier for inequality constraints (6)

Also in both cases we have $\lambda g(x^*) = 0$.

- ▶ Constraint active: $\lambda > 0$, $g(x^*) = 0$.
- ▶ Constraint inactive: $\lambda = 0$, $g(x^*) \neq 0$.

This is called the **Karush-Kuhn-Tucker (KKT) condition**.

Simple example

What are the side lengths of a rectangle that maximize its area, under the assumption that its perimeter is at most 1?

We need to solve the following optimization problem:

$$\text{maximize } x \cdot y \text{ subject to } 2x + 2y \leq 1$$

Bring the problem in standard form:

$$\text{minimize } (-x \cdot y) \text{ subject to } 2x + 2y - 1 \leq 0$$

Form the Lagrangian:

$$L(x, y, \lambda) = -xy + \lambda(2x + 2y - 1)$$

Simple example (2)

Saddle point conditions / derivatives:

$$\partial L / \partial x = -y + 2\lambda \stackrel{!}{=} 0$$

$$\partial L / \partial y = -x + 2\lambda \stackrel{!}{=} 0$$

$$\partial L / \partial \lambda = 2x + 2y - 1 \stackrel{!}{=} 0$$

Solving this system of three equations gives $x = y = 0.25$.

Simple example (3)

Now need to see: when does this approach work, when does it not work, what can we prove about it?

Lagrangian: formal point of view

Lagranigan and dual: formal definition

Consider the primal optimization problem

$$\begin{aligned} &\text{minimize } f_0(x) \\ &\text{subject to } f_i(x) \leq 0 \quad (i = 1, \dots, m) \\ &\quad \quad \quad h_j(x) = 0 \quad (j = 1, \dots, k) \end{aligned}$$

Denote by x^* a solution of the problem and by $p^* := f_0(x^*)$ the objective value at the solution.

Lagranigan and dual: formal definition (2)

Define the corresponding **Lagrangian** as follows:

- ▶ For each equality constraint j introduce a new variable $\nu_j \in \mathbb{R}$, and for each inequality constraint i introduce a new variable $\lambda_i \geq 0$. These variables are called **Lagrange multipliers**.
- ▶ Then define

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \sum_{j=1}^k \nu_j h_j(x)$$


Define the **dual function** $g : \mathbb{R}^m \times \mathbb{R}^k \rightarrow \mathbb{R}$ by

$$g(\lambda, \nu) = \inf_x L(x, \lambda, \nu)$$

Dual function as lower bound on primal

Proposition 58 (Dual function is concave)

No matter whether the primal problem is convex or not, the dual function is always concave in (λ, ν) .

Proof. For fixed x , $L(x, \lambda, \nu)$ is linear in λ and ν and thus concave. The dual function as a pointwise infimum over concave functions is concave as well. 

Note that concave is good, because we are going to maximize this function later on.

Dual function as lower bound on primal (2)

Proposition 59 (Dual function as lower bound on primal)

For all $\lambda_i \geq 0$ and $\nu_j \in \mathbb{R}$ we have $g(\lambda, \nu) \leq p^*$.

Proof.

- ▶ Let x_0 be a feasible point of the primal problem (that is, a point that satisfies all constraints).
- ▶ For such a point we have

$$\sum_{i=1}^m \underbrace{\lambda_i}_{\geq 0} \underbrace{f_i(x_0)}_{\leq 0} + \sum_{j=1}^k \nu_j \underbrace{h_j(x_0)}_{=0} \leq 0$$

Dual function as lower bound on primal (3)

- This implies

$$L(x_0, \lambda, \nu) = f_0(x_0) + \sum_{i=1}^m \lambda_i f_i(x_0) + \sum_{j=1}^k \nu_j h_j(x_0) \leq f_0(x_0)$$

Note that this property holds in particular when x_0 is x^* .

- Moreover, for any x_0 (and in particular for $x_0 := x^*$) we have

$$\inf_x L(x, \lambda, \nu) \leq L(x_0, \lambda, \nu)$$

- Combining the last two properties gives

$$g(\lambda, \nu) = \inf_x L(x, \lambda, \nu) \leq L(x^*, \lambda, \nu) \leq f_0(x^*)$$



Dual optimization problem

Have seen: the dual function provides a lower bound on the primal value. Finding the highest such lower bound is the task of the dual problem:

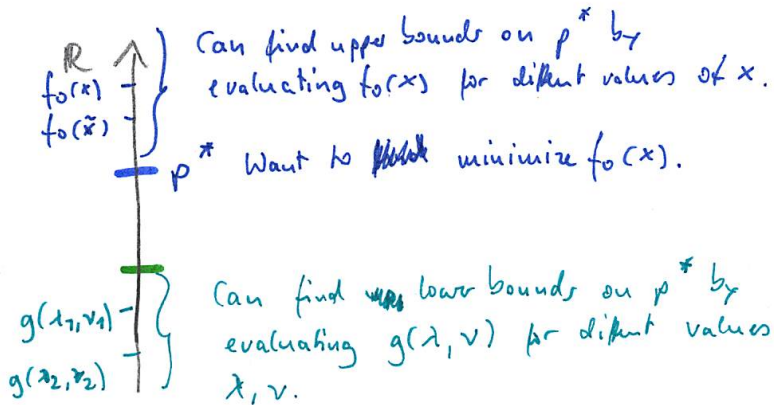
We define the **dual optimization problem** as

$$\max_{\lambda, \nu} g(\lambda, \nu) \text{ subject to } \lambda_i \geq 0, \nu_j \in \mathbb{R}$$

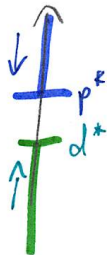
Denote the solution of this problem by λ^*, ν^* and the corresponding objective value $d^* := g(\lambda^*, \nu^*)$.

Dual optimization problem (2)

Dual vs Primal, some intuition:



Dual optimization problem (3)



Primal problem: find best upper bound on p^*
(= find p^*)

Dual problem: find best lower bound on p^*
(= find d^*)

Weak duality

Proposition 60 (Weak duality)

The solution d^* of the dual problem is always a lower bound for the solution of the primal problem, that is $d^* \leq p^*$.

Proof. Follows directly from Proposition 59 above.



We call the difference $p^* - d^*$ the **duality gap**.

Strong duality

- ▶ We say that **strong duality** holds if $p^* = d^*$.
- ▶ This is not always the case, just under particular conditions. Such conditions are called **constraint qualifications** in the optimization literature.
- ▶ Convex optimization problems often satisfy strong duality, but not always.

Strong duality (2)

Examples:

- ▶ Linear problems have strong duality
- ▶ Quadratic problems have strong duality (\leadsto support vector machines)
- ▶ There exist many convex problems that do not satisfy strong duality. Here is an example:

$$\begin{array}{ll}\text{minimize}_{x,y} & \exp(-x) \\ \text{subject to} & x/y \leq 0 \\ & y \geq 0\end{array}$$

One can check that this is a convex problem, yet $p^* = 1$ and $d^* = 0$.

Strong duality: how to convert the solution of the dual to the one of the primal

By strong duality: $p^* = d^*$, that is we get the same objective values. But how can we recover the primal variables x^* that lead to this solution, if we just know the dual variables λ^*, ν^* of the optimal dual solution?

EXERCISE!

Strong duality implies saddle point

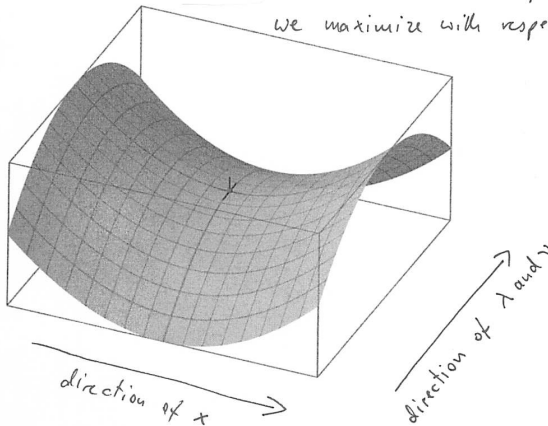
Proposition 61 (Strong duality implies saddle point)

Assume strong duality holds, let x^* be the solution of the primal and (λ^*, ν^*) the solution of the dual optimization problem. Then (x^*, λ^*, ν^*) is a saddle point of the Lagrangian.

Strong duality implies saddle point (2)

Lagrangian saddlepoint

we minimize with respect to x ,
we maximize with respect to λ, ν



Strong duality implies saddle point (3)

Proof.

- ▶ We first show that x^* is a minimizer of $L(x, \lambda^*, \nu^*)$:
 - ▶ By the strong duality assumption we have $f_0(x^*) = g(\lambda^*, \nu^*)$.
 - ▶ With this we get

$$f_0(x^*) = g(\lambda^*, \nu^*) = \inf_x L(x, \lambda^*, \nu^*) \leq L(x^*, \lambda^*, \nu^*) \leq f_0(x^*)$$

(last inequality follows from Proposition 60).

- ▶ Because we have the same term on the left and side, we have equality everywhere.
 - ▶ So in particular, $\inf_x L(x, \lambda^*, \nu^*) = L(x^*, \lambda^*, \nu^*)$.
- ▶ Then we show that (λ^*, ν^*) are maximizers of $L(x^*, \lambda, \nu)$.
 - ▶ This follows from the definition of (λ^*, ν^*) as solutions of $\max_{\lambda, \nu} \min_x L(x, \lambda, \nu)$.

Strong duality implies saddle point (4)

- Taken together we get

$$L(x^*, \lambda, \nu) \leq L(x^*, \lambda^*, \nu^*) \leq L(x, \lambda^*, \nu^*)$$

That is, (x^*, λ^*, ν^*) is a **saddle point** of the Lagrangian:

- It is a minimum for x (with fixed λ^*, ν^*).
- It is a maximum for (λ, ν) (with fixed x^*).



Saddle point always implies primal solution

Proposition 62 (Saddlepoint implies primal solution)

If (x^*, λ^*, ν^*) is a **saddle point** of the Lagrangian, then x^* is always a solution of the primal problem.

Proof. Not very difficult, but we skip it.



Remarks:

- ▶ This proposition always holds (not only under strong duality).
- ▶ This proposition gives sufficient conditions for optimality. Under additional assumptions (constraint qualifications) it is also a necessary condition.

Why is this whole approach useful?

- ▶ Whenever we have a saddle point of the Lagrangian, we have a solution of our constraint optimization problem. This is great, because otherwise we would not know how to solve it.
- ▶ If strong duality holds, we even know that any solution must be a saddle point. So if we don't find a saddle point, then we know that no solution exists.
- ▶ If your original minimization problem is not convex, at least its dual is a concave maximization problem (or, by changing the sign, a convex minimization problem). If the duality gap is small, then it might make sense to solve the dual instead of the primal (you will not find the optimal solution, but maybe a solution that is close).
- ▶ As we will see for support vector machines, the Lagrangian framework sometimes gives important insights into properties of the solution.