

Statistical Machine Learning

Ulrike von Luxburg

Summer term 2026

Department of Computer Science, University of Tübingen

(Version of September 4, 2026)

Table of contents

Introduction	9
Prelude	10
A very short history of AI/ML	11
Logistics of the lecture	29
Machine learning as inductive inference	40
Predicting based on perfect knowledge	53
Probability distributions, loss functions, risk	54
Optimal prediction functions in closed form	62
... for classification under 0-1 loss	63
... for other classification loss functions	73
... for regression under L_2 loss	84
Statistical and Bayesian decision theory	90
Predicting in the finite sample setting	110
Training data, learning, and consistency	111
Estimating the true risk by the empirical risk	120
Framework of empirical risk minimization	130
Estimation and approximation error	139
Bias-Variance Decomposition	147
The No-Free-Lunch Theorem	153

Table of contents (2)

Fundamental principles of machine learning	166
Motivation	167
Warmup: Perceptron mistake bound	170
The perceptron algorithm	171
Perceptron performs linear loss minimization	174
Mistake bound for the perceptron	182
Generalization error of the perceptron	192
Bounding the complexity of the function class	204
Statistical Learning Theory	205
Important tool: Concentration inequalities	209
Controlling the estimation error: generalization bounds	216
Estimating the error of a fixed function	217
Uniform convergence is necessary and sufficient for consistency	222
Generalization bound for finite classes	229
Example: partitioning-based estimators	235
Infinite classes, symmetrization and bounds with the shattering coefficient	244
VC dimension	263
Definition, Sauer-Shelah lemma and generalization bound	264
VC dimension of linear classifiers and margin-based classifiers	276
VC dimension of neural networks	286

Table of contents (3)

Rademacher complexity	297
Limitations and outlook	302
Stability	309
Motivation and key definitions	310
Algorithmic stability implies generalization	317
ERM with strongly convex loss function is stable	330
(Stochastic) gradient descent on smooth loss function is stable	342
Explicit regularization	360
Principle of regularization	361
Recap: Linear least squares regression	370
Ridge regression / Tikhonov regularization: least squares with L_2 -regularization	387
Tikhonov regularization leads to stability	402
Lasso: least squares with L_1 -regularization	406
Theoretical results regarding the different types of regularization	420
Aggregation, bagging, boosting	441
Motivation from statistics: Ensembles, bagging, bootstrap	442
Random Forests	451
Boosting	465
AdaBoost algorithm	475
Bounding the training error	478

Table of contents (4)

Bounding the test error of AdaBoost	484
Gradient Boosting	498
Beyond algorithms: data, measurement and validity	507
Data	508
The problem of measuring the target concept	509
Harms that can come with data	519
Bias and discrimination in data sets	520
Violation of ethical standards	524
Privacy violation and copyright	526
Garbage in, garbage out	529
Good data practices	531
Features and representation	538
Explicitly constructing features	539
Going non-linear by choosing basis functions	544
Implicit feature representation by kernel methods	551
Random feature models	557
Representation learning	570
Validation and benchmarking	574
Benchmarking in machine learning	575

Table of contents (5)

Four notions of validity	580
Validity of machine learning benchmarks	583
Validity threats in foundation model research	595
Learning in the over-parameterized interpolation regime	605
Puzzling empirical results that seem at odds with classical theory	606
Principles that enable deep learning	613
The double descent curve	614
Implicit regularization by the choice of the optimization algorithm	634
GD and SGD for linear least squares regression converge to the minimum norm solution	635
GD for logistic loss classification converges to maximum margin solution	644
Implicit bias of neural networks	647
The statistics of deep learning: When does the minimum norm solution generalize? Benign overfitting.	650
Benign overfitting and spiky-smooth structure in a toy setup: Learning the constant 0 function	652
Not so benign: Linear model with isotropic Gaussian inputs	665
Benign overfitting: Linear model with aligned covariance structure	687
Does benign overfitting contradict the classical learning theory?	711
Why (?) do (?) we need large models?	717
Representation power of neural networks	718
Large models are necessary for robust interpolation	731

Table of contents (6)

The loss landscape in our favor	739
Infinite width theory and the neural tangent kernel	748
Theory regarding depth?	756
Take home	758
Machine Learning in the context of society	766
Fairness	767
Is it fair?	768
Machine learning systems are often unfair	782
Formal notions of fairness, and why we cannot satisfy all of them at the same time	798
Technical approaches to improve fairness	818
Tradeoffs	830
Explainability for tabular data	834
Introduction	835
Local posthoc explanations (feature attributions)	855
Local posthoc explanations (feature attributions)	856
First issues with feature importance scores: correlations and interactions	857
Counterfactual explanations	862
SHAP	867
LIME	891

Table of contents (7)

Criticism, discussion about local posthoc explanations	897
Legal issues	908
European AI Act	911
European GDPR	918
European laws around intellectual properties and copyright	922
Regulation outside the EU	925

Introduction

Prelude

A very short history of AI/ML

Literature: I didn't really dig out a book, but you could read the intro of the Hardt book on "Patterns, predictions, actions"

1950ies: AI gets “invented”

Dartmouth Summer Conference, 1956:

“Every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it.”



John McCarthy



Marvin Minsky



Claude Shannon



Ray Solomonoff



Alan Newell



Herbert Simon



Arthur Samuel



Oliver Selfridge



Nathaniel Rochester

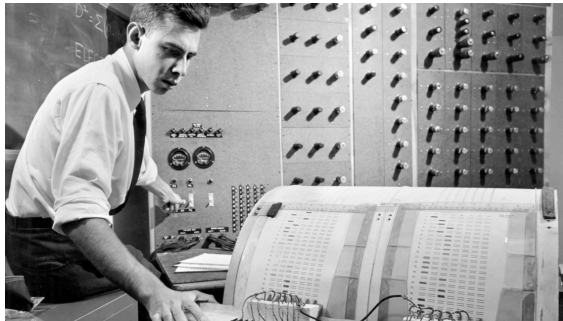


Trenchard More

Main paradigm: Symbolic AI

1950ies: AI gets “invented” (2)

1957 Frank Rosenblatt invents the Perceptron, the first machine that learns. It had 400 photosensors, could read an image of 20×20 pixels and sort it in two classes. Rosenblatt discovered that it could learn to do so!



1950ies: AI gets “invented” (3)

Huge optimism:

”The Navy revealed the embryo of an electronic computer that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence” - New York Times, 1958

“Within ten years a digital computer will be the world’s chess champion.” — Herbert A. Simon, 1958

“Within a generation . . . the problem of creating ‘artificial intelligence’ will substantially be solved.” — Marvin Minsky (early 1960ies)

1960ies: Symbolic AI and the limits of perceptrons

- ▶ ELIZA (1966): early chatbot that mimicked conversation using simple pattern matching, creating the illusion of understanding (psychotherapist!)
 - ▶ **User:** I feel stressed about school.
 - ▶ **ELIZA:** Why do you feel stressed about school?
 - ▶ **User:** I have too much homework.
 - ▶ **ELIZA:** How long have you had too much homework?
 - ▶ **User:** For a few weeks now.
 - ▶ **ELIZA:** Do you think it is normal to have too much homework?
 - ▶ **User:** No, it's overwhelming.
 - ▶ **ELIZA:** Why do you say it is overwhelming?

1960ies: Symbolic AI and the limits of perceptrons (2)

- ▶ 1969: Minsky/Papert: Single-layer perceptron can only learn linearly separable functions, for example cannot learn XOR. No good training methods for multi-layer perceptrons known.

1970ies: AI winter, but first steps in statistical pattern recognition

AI Winter: big hopes do not materialize:

“In no part of the field have the discoveries made so far produced the major impact that was then promised.”

(Lighthill Report to the UK Science Research Council, 1973)

“The difficulties of achieving artificial intelligence have been underestimated.”

(US funding reviews, 1970s)

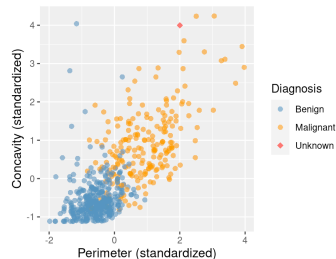
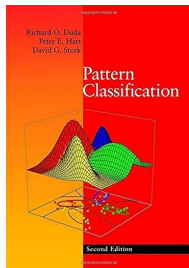
“The question of whether a computer can think is no more interesting than the question of whether a submarine can swim.”

(Edsger W. Dijkstra, 1980ies)

1970ies: AI winter, but first steps in statistical pattern recognition (2)

First steps in statistical pattern recognition:

- ▶ Pattern recognition (often classification) with a statistical flavor develops: kNN, naive Bayes, Bayesian decision theory, ...
- ▶ Learning from data with (few) hand-designed features



- Backpropagation gets invented, sparks new interest in neural networks: multi-layer networks get possible; progress e.g. hand-written digit recognition.

Fachbereich Informatik

Saal 14

72076 Tübingen

- Decision trees (ID3 algorithm)

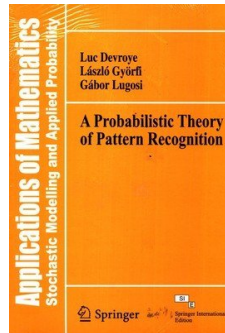
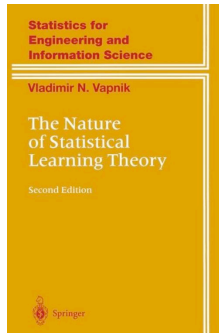
1990ies

- Deep Blue beats Garry Kasparov in chess (by brute force searching, no machine learning involved)



1990ies (2)

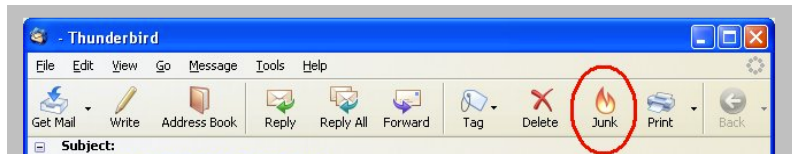
- Modern ML rises: Support vector machines, first generalization bounds (VC dimension)



2000s: ML starts to become “practical”

This was when I started my PhD ...

- ▶ Kernel methods everywhere
- ▶ Boosting
- ▶ Random forests, bagging (statistics community!)
- ▶ Strong theoretical developments (concentration inequalities, generalization bounds, ...)
- ▶ Many real-world problems get solved with lots of data: spam filtering, intrusion detection, recommender systems, bio-informatics with gene expression data, ...



2000s: ML starts to become “practical” (2)

Towards self-driving cars:

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



2000s: ML starts to become “practical” (3)

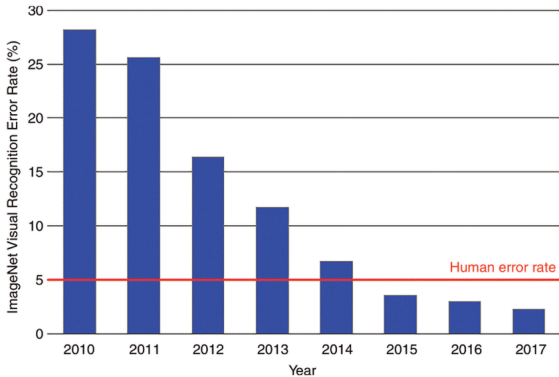
First success stories in language:

2011: Computer “Watson” wins the american quiz show “jeopardy” (lots of word games).



2010s: rise of deep neural networks

2012 Imagenet breakthrough using CNNs and GPUs, no hand-coded features.



2010s: rise of deep neural networks (2)

2016 Alpha Go defeated the world champion Lee Se-dol.



2010s: rise of deep neural networks (3)

2017 Transformers / Attention

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* [†]
University of Toronto
aidan@cs.toronto.edu

Lukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin* [‡]
illia.polosukhin@gmail.com

2020ies: Generative AI

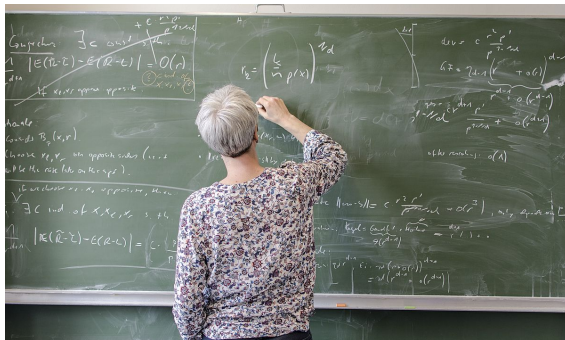
- ▶ Generative AI for images, audio, text, video, ...
- ▶ Language models, Foundation models
- ▶ Claude Coding, Agentic AI



Blueberry spaghetti

Logistics of the lecture

Introducing myself



The team of TAs



Gunnar König

Kai Luedemann

Shubham Kashyapi

Anupam Sourav Patra

Who are you?



What this lecture is about

The lecture is rather theoretical. It tries to make the point why and under which assumptions machine learning works or doesn't work.

- ▶ Statistical setup of machine learning: Bayesian decision theory, convergence, consistency, estimation / approximation error, bias-variance tradeoff, ...
- ▶ Fundamental principle that enable machine learning (learning theory, stability, regularization, aggregation), both in the classic and in the overparameterized regime
- ▶ Learning in the interpolation regime and why / when it works
- ▶ General issues in practice regarding data, training, validation, benchmarking
- ▶ Machine learning and society: fairness, discrimination, explainable machine learning, legal frameworks

What this lecture is about (2)

Table of contents: see webpage

What this lecture does NOT cover

- ▶ Basic machine learning algorithms (visit the BSc Intro to ML instead)
- ▶ Unsupervised learning (used to be included, but now dropped; you can still find it in the old lecture notes and videos)
- ▶ Reinforcement learning, probabilistic learning (visit these lectures separately)
- ▶ Online learning, time series
- ▶ Images, Language models, ...

Required prior knowledge

Relies heavily on maths for ML: linear algebra (matrices, eigenvectors, ...), multi-dimensional calculus (gradients, Hessians, ...), probability theory. If in doubt, look at the videos of the corresponding lecture (link on the webpage).

Assumes that you are familiar with machine learning in general and have seen some first introductory algorithms.

Literature and text books

There is not a single book that covers all I want to say, but I will repeatedly draw from the following books:

- ▶ Hardt, Recht: Patterns, predictions and actions, 2022.
- ▶ Shalev-Shwartz and Ben-David: Understanding machine learning, 2014.

Less often I might also use:

- ▶ Francis Bach: Learning theory from first principles, 2024.
- ▶ Hastie, Tibshirani, Friedman: Elements of Statistical Learning
- ▶ Mohri, Rostamizadeh, Talwalkar: Foundations of machine learning.
- ▶ Hardt: The Emerging Science of Machine Learning Benchmarks. Book not yet printed, PDF online.
- ▶ Devroye, Györfi, Lugosi: Probabilistic theory of pattern recognition, 1996

I try to put references at the beginning of each section.

Looking for a latex student assistant

- ▶ I am in the process of creating many new slides, handwritten
- ▶ Looking for a student who would like to convert them in proper latex, in parallel to the lecture.
- ▶ You get a contract as a student assistant, may-july, 20 h / month (and we keep track of your time; if in the end it takes longer, we will extend the contract).
- ▶ If you are interested, please send me an email with your name, your quick CV, your transcripts, and a statement how well you know latex and what your biggest latex project has been so far.

Logistics of lecture, tutorials, exam

... we discuss this along the information file on the webpage ...

Machine learning as inductive inference

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

One way to “define” machine learning is the following:

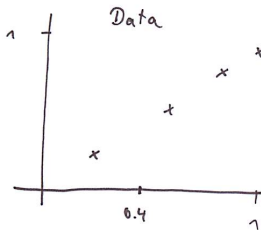
Machine learning tries to automate the process of inductive inference.

Why should machine learning work at all?

Consider the following simple regression example:

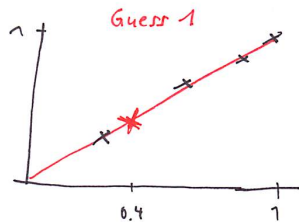
- ▶ Given: input-output pairs (X_i, Y_i) , $X_i \in \mathcal{X}$, $Y_i \in \mathcal{Y}$.
- ▶ Goal: learn to predict the Y -values from the X -values, that is we want to “learn” a suitable function $f : \mathcal{X} \rightarrow \mathcal{Y}$.

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)

Consequence 1:

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

Examples:

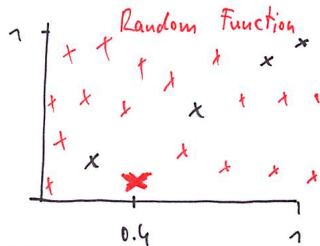
- ▶ The function f is periodic (for example, because we look at daily temperatures)
- ▶ The function f is “simple” (but caution, this is not the end of the story, see later in the section on learning theory)
- ▶ “Similar inputs” lead to “similar outputs”

Side remark: even animal learning has inductive biases. Check out the “Garcia effect” if you want to know more.

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.

Why should machine learning work at all? (4)

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? (5)

Consequence 2:

We will only be able to learn if “there is something we can learn”, if the input data contains some signal regarding the output values.

Examples:

- ▶ Output Y “has something to do” with input X
- ▶ There is a “simple relationship” or “simple rule” to generate the output for a given input

Excursion: Inductive bias in animal learning

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.

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 (2)

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.

Bottom line for now

Machine learning can only work on reasonable data.

Any successful learning algorithm has an inherent inductive bias.

For some algorithms it is obvious what the inductive bias is; for others it is hard to understand what exactly the bias is, and how to manipulate it. 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.

Predicting based on perfect knowledge

Probability distributions, loss functions, risk

Literature:

Bach Sec. 2

Devroye et al Sec 2

Hastie, Section 2.4 - 2.9 (parts only)

Making predictions based on perfect knowledge

In this section we investigate the case where we have complete knowledge, and ask ourselves whether and how we can derive the “best” predictions from our knowledge.

We first introduce the standard statistical setup of machine learning:

The underlying data spaces

- ▶ 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 $f : \mathcal{X} \rightarrow \mathcal{Y}$.

In this lecture we consider the following two cases:

- ▶ If $\mathcal{Y}$ discrete: classification
- ▶ If $\mathcal{Y} = \mathbb{R}$: regression

(Other output spaces are possible as well, for example "structured prediction")

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?

Later we often drop the dependence of the loss on x and simply write $\ell(y, y')$.

True Risk of a classifier / prediction function

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**.

- ▶ Note that the Bayes classifier is not necessarily unique.

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 binary 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 for 0-1-loss

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 for 0-1-loss (2)

Theorem 2 (f° is a 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 for 0-1-loss (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 for 0-1-loss (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 for 0-1-loss (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

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



... for other classification loss functions

Loss functions for binary classification based on a scoring function f

Because the 0-1-loss is hard to work with (WHY?), one uses many alternative loss functions instead (in the section on practical aspects of ML we will discuss more about this, their properties, and whether/when this will lead to the same result or not). To account for this, we slightly change the setup:

- Consider the setting of binary classification where we consider a real valued “scoring function” $f : \mathcal{X} \rightarrow \mathbb{R}$ and then use thresholding at 0 to assign the binary label $+1$ or -1 :

$$c_f(x) = \text{sign}(f(x)).$$

- Observe: A point (X, Y) is correctly classified by function f (resp. c_f) if $f(X) \cdot Y > 0$.

Loss functions for binary classification based on a scoring function f (2)

- The Bayes predictor f^* is the scoring function that minimizes the loss:

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

The corresponding classifier is then

$$c^*(x) := c_{f^*}(x) = \operatorname{sign}(f^*(x)).$$

- Because we now use labels -1 and +1 (rather than 0 and 1 as before) our regression function is:

$$\eta_1(x) := \mathbb{E}(Y \mid X = x) = P(Y = 1 \mid X = x) - P(Y = -1 \mid X = x) \in [-1, 1]$$

(I use the notation η_1 to distinguish it from the η that we derived in the last subsection).

Loss functions for binary classification based on a scoring function f (3)

In particular,

$$\eta(x) = P(Y = 1 \mid X = x) = \frac{1 + \eta_1(x)}{2}$$

$$1 - \eta(x) = P(Y = -1 \mid X = x) = \frac{1 - \eta_1(x)}{2}$$

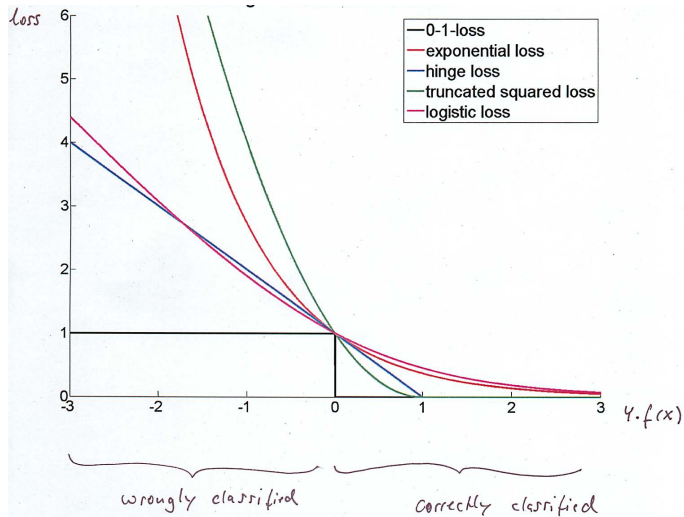
- The Bayes classifier under the 0-1-loss, expressed in the old η and the new η_1 is given as

$$\begin{cases} 1 & \text{if } \eta(x) > 0.5 \\ -1 & \text{if } \eta(x) < 0.5 \end{cases} = \begin{cases} 1 & \text{if } \eta_1(x) > 0 \\ -1 & \text{if } \eta_1(x) < 0 \end{cases}$$

Loss functions for binary classification based on a scoring function f (4)

- There are many different loss functions that can be used in this regime: hinge loss, squared loss, exponential loss, logistic loss, etc. Note that in case of binary classification, the cross-entropy loss coincides with the logistic loss.

Loss functions for binary classification based on a scoring function f (5)



Bayes predictors for various loss functions

One can derive the optimal Bayes predictor f^* and the corresponding classifier c_{f^*} for all these loss functions:

name	loss $\ell(y, f(x))$	Bayes predictor f^*	Classifier c_{f^*}
hinge loss	$\max(0, 1 - yf(x))$	$\begin{cases} 1 & \text{if } \eta_1(x) > 0 \\ -1 & \text{if } \eta_1(x) < 0 \end{cases}$	$\begin{cases} 1 & \text{if } \eta_1(x) > 0 \\ -1 & \text{if } \eta_1(x) < 0 \end{cases}$
squared loss	$\max(0, 1 - yf(x))^2$	$\eta_1(x)$	$\begin{cases} 1 & \text{if } \eta_1(x) > 0 \\ -1 & \text{if } \eta_1(x) < 0 \end{cases}$
exponential loss	$\exp(-yf(x))$	$\frac{1}{2} \log\left(\frac{1 + \eta_1(x)}{1 - \eta_1(x)}\right)$	$\begin{cases} 1 & \text{if } \eta_1(x) > 0 \\ -1 & \text{if } \eta_1(x) < 0 \end{cases}$
logistic loss	$\log_2(1 + \exp(-yf(x)))$	$\log\left(\frac{1 + \eta_1(x)}{1 - \eta_1(x)}\right)$	$\begin{cases} 1 & \text{if } \eta_1(x) > 0 \\ -1 & \text{if } \eta_1(x) < 0 \end{cases}$

Bayes predictors for various loss functions (2)

Some first comments:

- ▶ We see that different loss functions ℓ give rise to different Bayes predictors. In particular, they induce a different inductive bias because they prefer slightly different scoring functions.
- ▶ Perhaps surprisingly (?) we see that the thresholded classifiers that are derived from the different scoring functions are always the same and coincide with the Bayes classifier under the 0-1-loss. This is a great property, and it has a name: the loss is classification-calibrated. See later.

Proof for the exponential loss function

As one more example, here is the derivation of the Bayes predictor for the **exponential loss function**:

Computing the risk of f at point x : We consider the risk of f , conditioned on $X = x$. Observe that while the values of the scoring function f are $\in \mathbb{R}$, our labels Y are still binary in $\{-1, +1\}$:

$$\begin{aligned} R_x(f) &= P(Y = 1 \mid X = x) \ell(f(x), 1) + P(Y = -1 \mid X = x) \ell(f(x), -1) \\ &= \frac{1 + \eta_1(x)}{2} \ell(f(x), 1) + \frac{1 - \eta_1(x)}{2} \ell(f(x), -1) \\ &= \frac{1 + \eta_1(x)}{2} e^{-f(x)} + \frac{1 - \eta_1(x)}{2} e^{f(x)} \end{aligned}$$

Proof for the exponential loss function (2)

Finding the minimum. We now want to find the value $f(x)$ that minimizes this expression. For clarity we now write $t := f(x)$:

$$R_x(t) := \frac{1 + \eta_1(x)}{2} e^{-t} + \frac{1 - \eta_1(x)}{2} e^t$$

We need to minimize this over t , so we compute the derivative wrt t and set it to 0 (observe that as we take the derivative wrt t , the expression $\eta(x)$ is just a constant):

$$\frac{\partial R_x(t)}{\partial t} = -\frac{1 + \eta_1(x)}{2} e^{-t} + \frac{1 - \eta_1(x)}{2} e^t \stackrel{!}{=} 0.$$

Solving this for t is simple and results in

$$t^* := f^*(x) := \frac{1}{2} \log \frac{1 + \eta_1(x)}{1 - \eta_1(x)},$$

which is what we wanted to find.

Proof for the exponential loss function (3)

To verify that this is indeed the global minimizer, we compute the second derivative at t^* , see that it is > 0 , so we found a minimum. Because the function that we minimize is convex, this is even a global minimum.

Classifier: It is easy to see that thresholding this function at 0 results in the expression for c_{f^*} on the slide (just set the function $f^*(x)$ to 0 and solve for $\eta_1(x)$).



... for regression under L_2 loss

Least squares loss for regression

The case of regression seems similar to the case of classification with a scoring function, but with one important difference: the true value Y can now be any real number, so in the proof we cannot simply distinguish between the two cases $Y = \pm 1$. In particular, conditional expectations in the next lines are really integrals, not sums, so the maths often gets more complicated.

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

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

As before we define the regression function:

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

Least squares loss for regression (2)

We now want to show an explicit formula for the Bayes learner. We need one more intermediate result:

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 above, 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.

Least squares loss for regression (3)

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

$$\begin{aligned}
 \# &= E([f(x) - \eta(x)][\eta(x) - Y]) \quad (\text{use } E(Z) = E(E(Z|\zeta))) \\
 &= E(E([f(x) - \eta(x)][\eta(x) - Y]|X)) \quad (\text{use } E(g(X)g(Y, X)|X) = g(X) \cdot E(g(Y, X)|X)) \\
 &= E([f(x) - \eta(x)] \cdot \underbrace{E([\eta(x) - Y]|X)}_{\substack{= \eta(X) - E(Y|X) \\ \underbrace{\hspace{1.5cm}}_{\eta(X)}}}) = 0
 \end{aligned}$$

Least squares loss for regression (4)



Explicit form of optimal solution under L_2 loss (regression)

Again we consider a particular function f° :

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

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 $f(X) = \eta(X)$.



Statistical and Bayesian decision theory

Literature:

- ▶ Hastie, Section 2.4 - 2.9 (parts only)
- ▶ Devroye, Section 2

More than just Bayes classifiers

In the last section we had derived the explicit form of Bayes classifiers for particular loss functions. Now we want to look at various possibilities to come up with “perfect classifiers” in a slightly more abstract setting.

We consider one running example throughout this section: male or female.

Running example: Male or female?

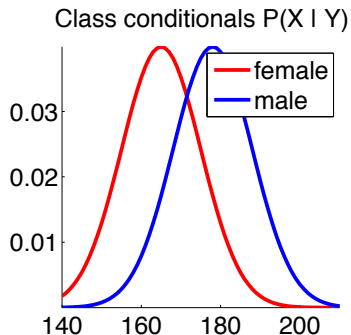
Predict gender of a person from body height:

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)

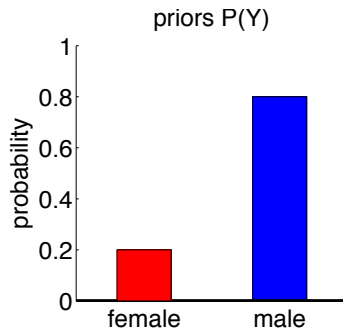
Decide based on class prior probabilities $P(Y)$.

- 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)

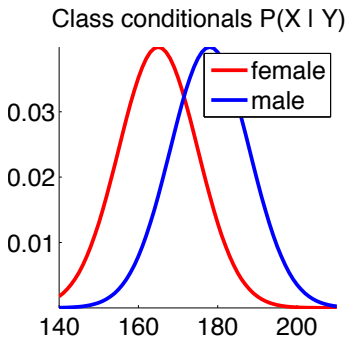
Visually: select the higher bar



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

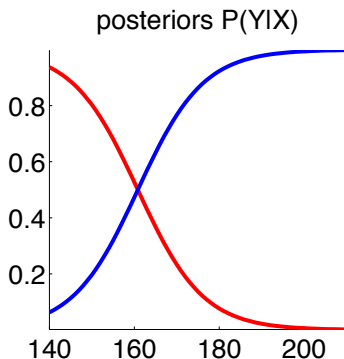
$$P(Y = m|X = x) = \frac{P(X = x|Y = m) \cdot P(Y = m)}{P(X = x)}$$

- 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

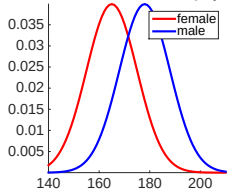
Run `python_u1e/teaching/demo_bayesian_decision_theory.py`

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.

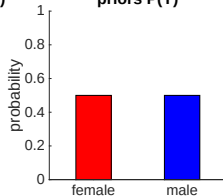
For your reference, the formulas to produce all plots follow at the end.

Example: male vs female (2)

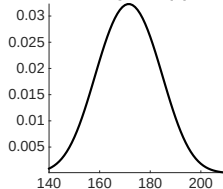
Class conditionals $P(X | Y)$



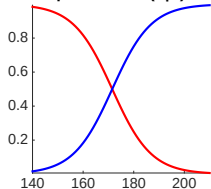
priors $P(Y)$



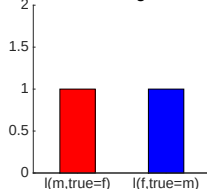
marginal $P(X)$



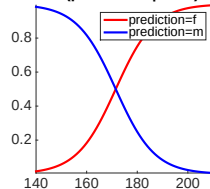
posteriors $P(Y|X)$



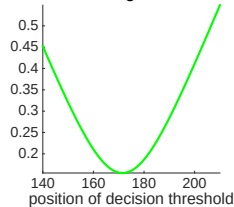
loss weights



pointwise risk
 $R(\text{prediction} | X=x)$

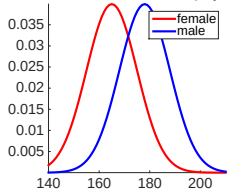


overall risk at given threshold

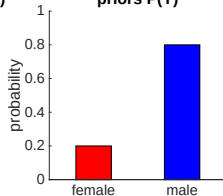


Example: male vs female (3)

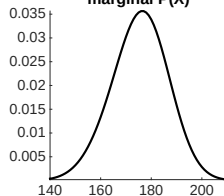
Class conditionals $P(X | Y)$



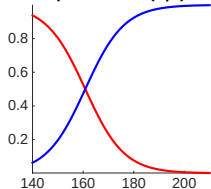
priors $P(Y)$



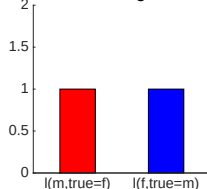
marginal $P(X)$



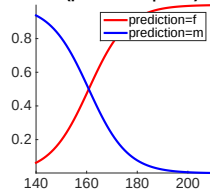
posteriors $P(Y|X)$



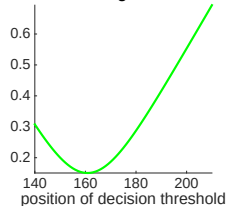
loss weights



pointwise risk
 $R(\text{prediction} | X=x)$

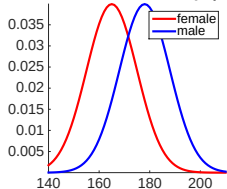


overall risk at given threshold

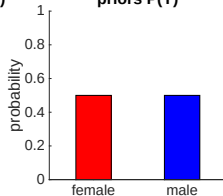


Example: male vs female (4)

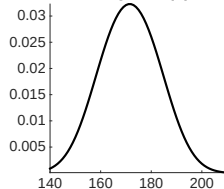
Class conditionals $P(X | Y)$



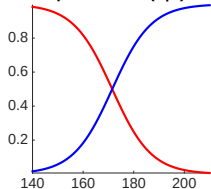
priors $P(Y)$



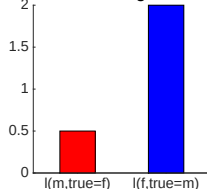
marginal $P(X)$



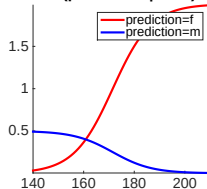
posteriors $P(Y|X)$



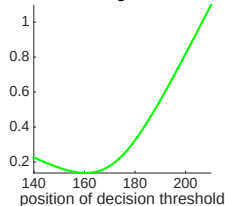
loss weights



pointwise risk
 $R(\text{prediction} | X=x)$

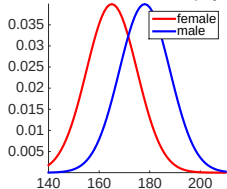


overall risk at given threshold

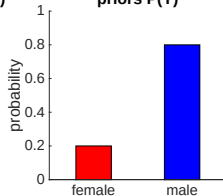


Example: male vs female (5)

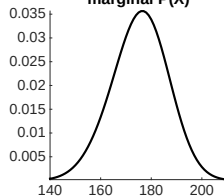
Class conditionals $P(X | Y)$



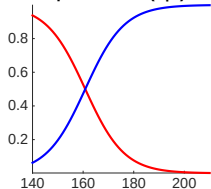
priors $P(Y)$



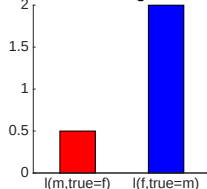
marginal $P(X)$



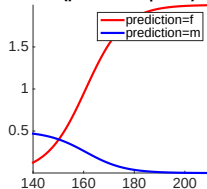
posteriors $P(Y|X)$



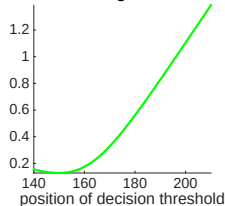
loss weights



pointwise risk
 $R(\text{prediction} | X=x)$

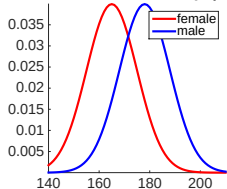


overall risk at given threshold

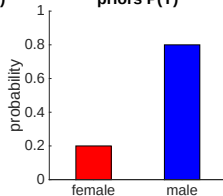


Example: male vs female (6)

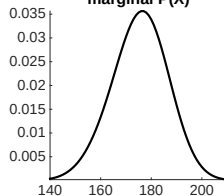
Class conditionals $P(X | Y)$



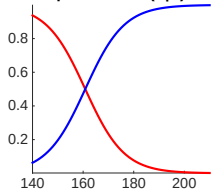
priors $P(Y)$



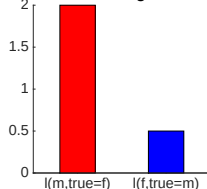
marginal $P(X)$



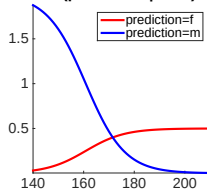
posteriors $P(Y|X)$



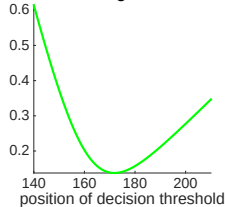
loss weights



pointwise risk
 $R(\text{prediction} | X=x)$



overall risk at given threshold



Example: male vs female (7)

In case somebody wants to re-implement it, these are the exact 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 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

$$R_t = P(Y = m | X \in]-\infty, t]) \cdot \ell_{fm} + P(Y = f | X \in]t, \infty]) \cdot \ell_{mf}$$

Example: male vs female (9)

$$= \ell_{fm} \int_{-\infty}^t P(Y = m \mid X = x) dx + \ell_{mf} \int_t^{\infty} P(Y = f \mid X = x) dx$$

Predicting in the finite sample setting

Training data, learning, and consistency

Literature:

Bach Sec. 2

Devroye et al Sec 2

Hastie, Section 2.4 - 2.9 (parts only)

Finite sample setting

- ▶ So far we discussed the “optimistic” regime in which we already know the underlying probability distributions. In this case, there is nothing to learn.
- ▶ Now we turn to the standard machine learning setting: we do not know the underlying distributions, but we get to see samples from it.
- ▶ Everything we do then has an attached uncertainty that depends on how well the sample we see represents the true underlying distribution.

The training data

In this lecture we always assume that we get training data by the following process:

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 .

The i.i.d. assumption:

- ▶ On the one hand it makes a lot of sense, to a certain degree even in language models (blocks of text during pre-training).
- ▶ The assumption allows us to use many tools from statistics which otherwise would not be valid.
- ▶ However there also exist many applications where “i.i.d.” is a strong assumption! (Can you come up with an example?)
- ▶ Another issue that is not covered by this framework is the one of different train and test distributions.

The goal of machine learning

Given n training points as described before, **the goal of learning is to construct a function f_n that has true risk “close” to the Bayes risk, that is $R(f_n) \approx R^*$.**

We now want to make this a bit more precise.

Recap: Convergence of Random Variables

Let $(U_i)_{i \in \mathbb{N}}$ be a sequence of random variables and U a random variable. We say that U_i converges to U ...

... **in probability** if $\forall \varepsilon > 0: \lim_{i \rightarrow \infty} P(|U_i - U| > \varepsilon) = 0$

... **almost surely (a.s.)** if $P(\lim_{i \rightarrow \infty} U_i = U) = 1$

... **in L_1 / in expectation** if $\lim_{i \rightarrow \infty} \mathbb{E}(|U_i - U|) = 0$

Relationships:

almost sure convergence implies convergence in probability

L_1 -convergence implies convergence in probability

no other implications in general.

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”; if we have convergence almost surely, the algorithm is called strongly consistent.)

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

Consistency of a learning algorithm (2)

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.

Observe that we do not argue about the convergence of the function f_n itself to the function f^* . We only are about the convergence of the true 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 such algorithms is an important part of this course, and we will take a glimpse on how to get consistency statements.

Why not use the plug-in classifier and be done?

If we don't know the underlying distribution, but are given some training data, we simply estimate the underlying distributions, for example 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. 😞

Estimating the true risk by the empirical risk

Estimating the risk of a classifier

- ▶ Have defined the “best” classifier as the one that minimizes the true risk.
- ▶ But if we do not know the underlying probability distribution, we cannot compute the risk of a prediction function.
- ▶ So the obvious idea is: try to estimate the risk of prediction functions based on the finite sample, and then pick the one that has the smallest estimated risk.
- ▶ The big question is whether this approach is (always? sometimes?) successful.
- ▶ We now first investigate how we can estimate the risk of a fixed, given prediction function f .

Standard setup

- ▶ We assume to operate in the standard setup: data spaces $\mathcal{X}$, $\mathcal{Y}$, underlying probability distribution P (exists but is now unknown), loss function ℓ .
- ▶ We are given a set of training points $(X_i, Y_i)_{i=1, \dots, n}$ that have been sampled i.i.d. from P .
- ▶ Ultimately, we want to “learn” a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that has as small true $R(f)$ loss as possible.

Empirical risk as an estimator of the true risk

As we don't know P we cannot compute the true risk of a function f . 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.

The first question is whether this is a “reasonable” / “good” estimate. OPINIONS?

What would be a formal way to express whether the estimate is “reasonable” or “good”?

Recap: law of large numbers

Recall from probability theory:

Proposition 5 (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) we have:

$$\frac{1}{n} \sum_{i=1}^n Z_i \rightarrow E(Z) \quad (\text{almost surely}).$$

Recap: law of large numbers (2)

Note that independence is a crucial assumption for the LLN. Here is an example that shows why:

- ▶ 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.

Recap: law of large numbers (3)

“Mild assumptions”:

- ▶ There exist many different versions of the LLN, with different assumptions.
- ▶ Often, the LLN is stated for random variables with finite variance, which makes the proof much simpler. But the LLN also holds without this assumption, as long as the expectation exists.
- ▶ The Z_i do not need to be identically distributed, as long as they have the same expectation.
- ▶ Many more variants ... see text books on probability theory if you are interested in details.

Convergence of the empirical risk of a fixed function

Proposition 6 (Empirical risk converges to true risk for fixed function)

In the standard setup, fix a function $f_0 : \mathcal{X} \rightarrow \mathcal{Y}$. Then, for this **fixed function** f_0 ,

$$R_n(f_0) \rightarrow R(f_0) \quad (\text{almost surely}).$$

DO YOU SEE WHY?

Convergence of the empirical risk of a fixed function (2)

Proof:

Simply apply the law of large numbers to the variables $Z_i := \ell(f_0(X_i), Y_i)$.

Discussion

- ▶ What does the convergence statement mean? What does it not mean?
- ▶ Limitations?

Framework of empirical risk minimization

Empirical risk minimization (ERM)

- ▶ Have seen: for a fixed function f , the empirical risk measured on a finite sample converges to the true risk as the sample size increases.
- ▶ So to find the Bayes classifier f^* (which minimizes the true risk), it makes a lot of sense to select the function f_n that minimizes the empirical risk and hope that it is close to the desired function.

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:

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

- ▶ Note that it might not be unique
- ▶ For simplicity, let's assume the minimizer exists (otherwise one has to argue about a sequence of functions that approaches the infimum)

The function class $\mathcal{F}$

Above we minimized the Bayes risk over a set of functions $\mathcal{F}$:

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

This set is an important “parameter” of ERM. It could be the set of all linear functions, the set of all decision trees of depth 5, the set of all random forests, the set of all two-layer MLP, etc.

“Classical regime” vs “modern” regime

In the following we sometimes distinguish between two different scenarios:

- ▶ the “classical” regime: the function class $\mathcal{F}$ does not allow for interpolation.
- ▶ the “modern” regime / interpolation regime: $\mathcal{F}$ is so large that training data can be interpolated (fitted with training error 0).

Is ERM always successful?

In our framework, a “successful classifier” is a consistent one. So: Does ERM always converge?

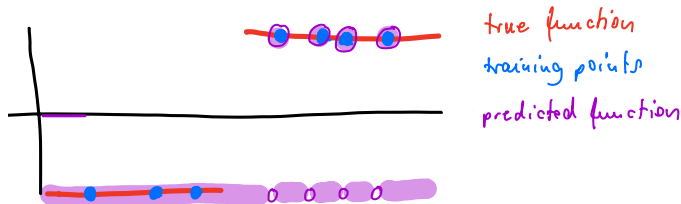
More formally: Let f_n be the function selected by empirical risk minimization based on the first n sample points. Is it always true that the risk of the classifier selected by ERM converges to the Bayes risk, that is $R_n(f_n) - R(f_n) \rightarrow 0$?

DISCUSSION ...

Is ERM always successful? (2)

No!!! 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.
- ▶ Then for all n we have $R_n(f_n) = 0$ but $R(f_n) = 0.5$.



Is ERM always successful? (3)

So why can't we use the LLN to prove convergence?

Here is the formal argument:

- ▶ The key property in the LLN is the independence of the variables Z_i .
- ▶ In our case, we consider $Z_i := \ell(f_n(X_i), Y_i)$
- ▶ 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 $(Z_i)_{i=1, \dots, n}$ are not independent any more (they all depend on f_n , which depends on all data points)
- ▶ Because independence is a key assumption in the LLN, we cannot use it.

Preview: Statistical learning theory

As we have seen, it is NOT true that the minimizer of the empirical risk always converges to the Bayes classifier!

So is machine learning doomed?!?

We will see soon that statistical learning theory can tell us conditions under which machine learning works after all.

ERM in practice?

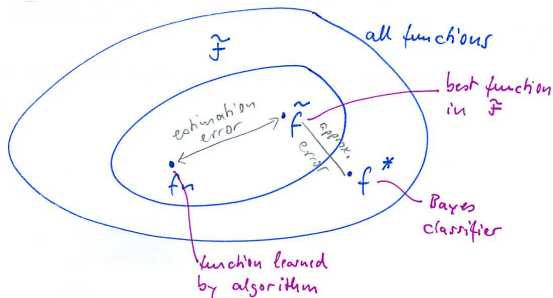
- ▶ From a conceptual/theoretical side, ERM is a straight forward learning principle.
- ▶ We will see that the key to the success / failure of ERM is to choose a “good” function class $\mathcal{F}$.
- ▶ We will see that this choice can be made explicitly (as in the “classical” regime) or happens implicitly (in the interpolation regime, sometimes).
- ▶ 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 often use convex relaxations of the 0-1-loss function, see later.

Estimation and approximation error

Estimation and approximation error: Definitions

When we perform empirical risk minimization, we can make two types of error:

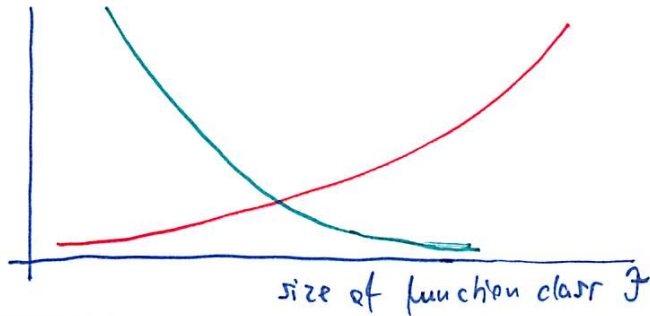
- ▶ Denote by $\tilde{f}$ the true best function in the set $\mathcal{F}$, that is $\tilde{f} = \operatorname{argmin}_{f \in \mathcal{F}} R(f)$.
- ▶ The quantity $R(f_n) - R(\tilde{f})$ is called the **estimation error**. It is a random variable that depends on the random sample.
- ▶ The quantity $R(\tilde{f}) - R(f^*)$ 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 and approximation error (classical regime)

Classical regime: the function class $\mathcal{F}$ is reasonably small (does not allow interpolation).

In this regime, this is how approximation and estimator error behave (which is which?).



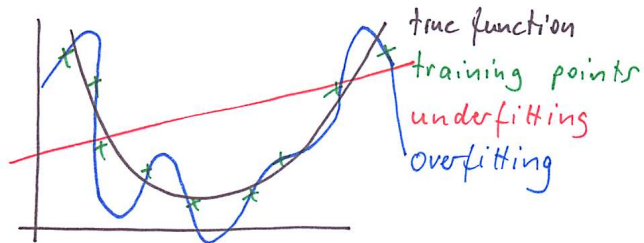
WHICH ONE IS WHICH? HOW WOULD THE CURVE OF THE TRUE RISK OF THE CLASSIFIER LOOK LIKE?

Overfitting and underfitting (classical regime)

Estimation and approximation error are also related to overfitting and underfitting:

Consider the following example:

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



Overfitting and underfitting (classical regime) (2)

Overfitting:

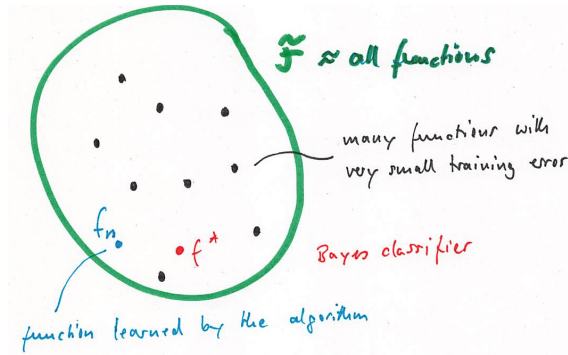
- ▶ Overfitting occurs if we can find a function $f \in \mathcal{F}$ that explains all training points very well or even exactly: $\mathcal{F}$ is very large.
- ▶ But such a function tends to be very complicated and models the noise as well
- ▶ Predictions for unseen data points are poor (“large test 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 and underfitting (modern regime)

Modern regime: the function class $\mathcal{F}$ is huge and pretty much covers the space of all possible functions.



There exist many functions 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 and underfitting (modern regime) (2)

In the modern regime:

- ▶ function classes are so large that they empirical risk minimizer interpolates the training points (the function f_n has 0 training error).
- ▶ The function class is so large that the approximation error is often close to 0 (the functions can represent pretty much anything we want).
- ▶ Regarding the estimation error, traditional approaches would predict that it is huge due to overfitting.
- ▶ But it turned out that some algorithms always find solutions that interpolate, yet the classifier generalizes well. These solutions then have a small estimation error.
- ▶ Practice shows that overfitting can even be beneficial!!!

How can this be?

Overfitting and underfitting (modern regime) (3)

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

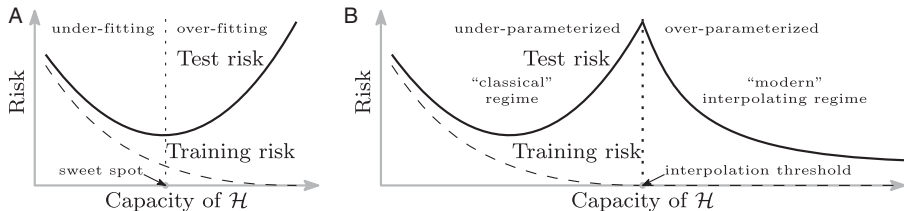


Fig. 1. Curves for training risk (dashed line) and test risk (solid line). (A) The classical U-shaped risk curve arising from the bias-variance trade-off. (B) The double-descent risk curve, which incorporates the U-shaped risk curve (i.e., the “classical” regime) together with the observed behavior from 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 for an in-depth discussion.

Bias-Variance Decomposition

Bias-Variance tradeoff in L_2 -regression

Sometimes another decomposition of the errors is used that. It is most simple for the case of regression with squared loss function:

Let f_n be the function constructed by an algorithm on n points sampled randomly from the underlying distribution, and $f^* : \mathbb{R}^d \rightarrow \mathbb{R}$ the true best function (the regression function). Then, for each fixed point $x \in \mathbb{R}^d$, we can decompose the pointwise expected L_2 risk in two terms (where the expectation is over the draw of the random training sample, and the test point x is considered fixed):

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

- 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.

$$\begin{aligned}
 E((f_n(x) - f^*(x))^2) &= E((f_n(x) - \eta(x))^2) \quad (\text{add/subtract red term}) \\
 &= E((\underbrace{f_n(x) - Ef_n(x)}_{\text{variance}} + \underbrace{Ef_n(x) - \eta(x)}_{\text{bias}}))^2 \quad (\text{expand the square}) \\
 &= E((f_n(x) - E(f_n(x))))^2 \quad \text{variance} \\
 &\quad + E(Ef_n(x) - \eta(x))^2 \quad \text{bias} \\
 &\quad + \underbrace{2E((f_n(x) - Ef_n(x))(Ef_n(x) - \eta(x)))}_{=0 \text{ see below}} \quad \text{term disappears}
 \end{aligned}$$

Bias-Variance tradeoff in L_2 -regression (3)

To see that the blue term disappears:

$$\begin{aligned}\text{blue term} &= E((f_n(x) - E(f_n(x)))(Ef_n(x) - \eta(x))) && \text{deterministic, pull out} \\ &= E((f_n(x) - Ef_n(x))) \cdot (Ef_n(x) - \eta(x)) \\ &= (Ef_n(x) - Ef_n(x)) \cdot (Ef_n(x) - \eta(x)) \\ &= 0\end{aligned}$$



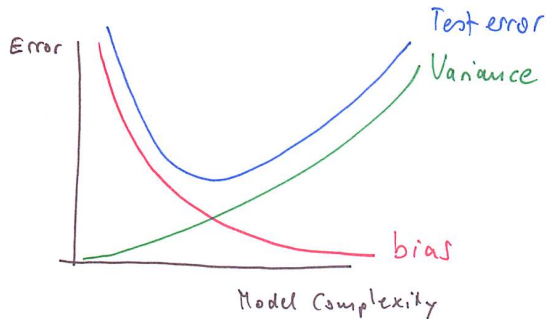
Bias-Variance tradeoff in L_2 -regression (4)

Intuition of the bias-variance decomposition:

- ▶ 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 (5)

The classical version of this figure looks like this:



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 all other classifiers when averaged on all possible learning problems.
- ▶ There exist many different versions to state this formally, below we describe the easiest one.

NFL, setup

- ▶ Assume that the space of all input points just consists of a finite set $\mathcal{X} = \{x_1, \dots, x_m\}$. Fix a probability distribution on these points, say the uniform distribution (the proofs below work for all distributions).
- ▶ Assume that we consider binary classification, that is $\mathcal{Y} = \{\pm 1\}$, and that the labels are deterministic functions of the input.
- ▶ We can now enumerate all possible classification functions that map $\mathcal{X}$ to $\mathcal{Y}$, there are 2^m many of them. Call the set of all these functions $\mathcal{F}$.
- ▶ Assume that one of these functions, call it f^* , is the true underlying function, and we observe (noise-free) training points of the form $(x_i, f^*(x_i))$.
- ▶ Assume that we have an algorithm that estimates a function f_n based on the training points.

NFL, setup (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 , under the 0-1 loss.

	true functions				
	f_1	f_2	f_3	$\dots$	f_{2^m}
f_1					
f_2					
$\vdots$					
r_{ij}					
$\vdots$					
f_{2^m}					

Each row is equally good

Proposition 7 (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 (in the col. of f^*)
- ▶ $r_{ij} = 1/m$ exactly m times (in the cols of all functions that disagree with f^* on one point).
- ▶ $r_{ij} = 2/m$ exactly $\binom{m}{2}$ times
- ▶ ...



Simple NFL without training data

Proposition 8 (Simple NFL)

In the model introduced above: On average over all true functions $f \in \mathcal{F}$, the performance of all classifiers $\hat{f}$ is the same.

Proof. Obvious consequence of the previous proposition.



Simple NFL (with training data)

Proposition 9 (Simple NFL with training data)

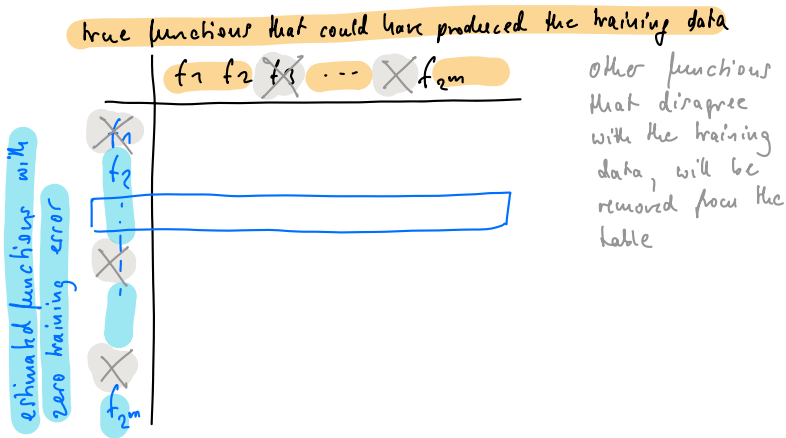
In the model introduced above: Assume we are given a training set $(X_i, Y_i)_{i=1, \dots, n}$. Then, on average over all true functions functions that could have generated the training data, all estimated classifiers that have 0 training error perform the same.

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.



Simple NFL (with training data) (2)



More NFLs

There also exist no free lunch theorems for infinite function classes, and the theorems also exist in different flavors. So it is NOT a phenomenon that comes from a finite function class.

We won't discuss the more general theorems.

Discussion

- ▶ Have seen: “The best possible classifier for all data sets” does not exist.
- ▶ SHOULD WE GIVE UP? IS MACHINE LEARNING MEANINGLESS?

Discussion (2)

- ▶ No: the key is that in practice we do not consider “all possible functions”. As soon as we make assumptions on the function space (or even, just exclude pathological functions) the NFL breaks down: Assumptions restrict the set of possible true functions, which means to delete some columns from the above matrix, and then the proof breaks down.
- ▶ Example: we would have a notion of similarity among our input points x_i , and then we would only consider functions that give similar labels to similar data points. This would exclude functions that look as if they distribute the labels randomly.
- ▶ This shows once more how important it is to incorporate these assumptions to the machine learning algorithm $\leadsto$ inductive bias!

Machine learning without an inductive bias is impossible!

Discussion (3)

Question to think about: Why doesn't the NFL contradict the existence of universally consistent learning algorithms?

History / Literature

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.

- ▶ *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*

Fundamental principles of machine learning

Motivation

Fundamental principles of machine learning

- ▶ The machine learning history went through many different phases, yet fundamental principles are in place when machine learning works.
- ▶ Some of the principles have already been discovered and discussed decades ago.
- ▶ When deep learning was on the rise, it seemed as if many of these “old” principles might not be “correct” or “in place” any more.
- ▶ But by now the community has re-discovered many of them also in the regime of deep learning — but often are not so plainly visible or in disguise
- ▶ My goal in this lecture is to show you some of the fundamental principles that enable machine learning.
- ▶ The main point that should become clear by the end of the lecture: if machine learning works, some of these principles has to be in place. And it might be important to know which one, because different principles induce different biases. The other way round: if machine learning does not seem to work, the reason might be that no inductive principle is in place that can solve the current problem at hand.

Fundamental principles of machine learning (2)

In our lecture, we will discuss in depth the following principles:

- ▶ Bounding the complexity of function classes: the approach justified by classic learning theory
- ▶ Regularization: explicit and implicit
- ▶ Stability: often implicit
- ▶ Aggregation: often helps

We will see how these principles are in place both in the classical domain of underparameterized algorithms and in the modern domain of overparameterized algorithms.

And we will prove under which conditions these principles help.

Warmup: Perceptron mistake bound

Literature:

- ▶ Sec 3 in the Hardt book (Patterns, predictions, actions)
- ▶ Sec 7.1. in the Mohri book (Foundations of ML)
- ▶ Sec 21.4 in the Ben-David book (Understanding ML)

The perceptron algorithm

Linear Classification in $\mathbb{R}^d$

Training points $x_1, x_2, \dots \in \mathbb{R}^d$, $y_1, y_2, \dots \in \{\pm 1\}$

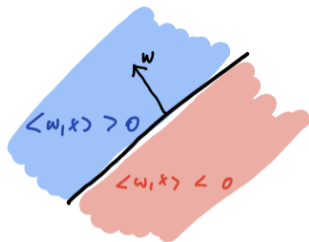
We will study linear classifiers:

$$f(x_i) = \text{sign}(\langle w, x_i \rangle)$$

with coefficients $w \in \mathbb{R}^d$

(and no offset parameter b)

Observe:



A point (x_i, y_i) is correctly classified

$$\iff y_i \langle w, x_i \rangle > 0$$

The perceptron Algorithm

Require: $(x_1, y_1), \dots, (x_n, y_n)$

- 1: $w_0 \leftarrow 0$
 - 2: $t \leftarrow 1$
 - 3: $N \leftarrow n$
 - 4: **while** $t \leq N$ **do**
 - 5: **if** $y_t \langle w_{t-1}, x_t \rangle \leq 0$ **then**
 - 6: $w_t \leftarrow w_{t-1} + y_t x_t$
 - 7: **else**
 - 8: $w_t \leftarrow w_{t-1}$
 - 9: **end if**
 - 10: $t \leftarrow t + 1$
 - 11: **end while**
- ▷ x_t is wrongly classified
 - ▷ update w
 - ▷ x_t is correctly classified
 - ▷ keep current w

Perceptron performs linear loss minimization

The linear loss function

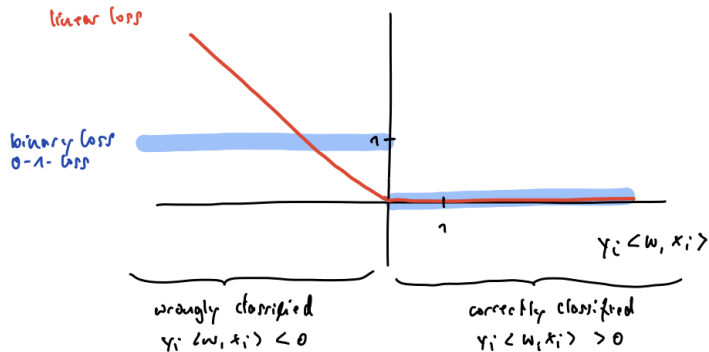
“Natural” loss function for classification is the **0-1 loss**:

$$\ell(\langle w, x_i \rangle, y_i) = \begin{cases} 0 & \text{if } f(x_i) = y_i, \\ 1 & \text{otherwise} \end{cases}$$

If we want to quantify “how wrong” the function is, we could use the **linear loss**:

$$\ell(\langle w, x_i \rangle, y_i) = \max\{0, -y_i \langle w, x_i \rangle\}$$

The linear loss function (2)



Linear loss minimization

Given: training points $(x_1, y_1), \dots, (x_n, y_n)$

Goal: find the linear classifier that minimizes the empirical risk:

$$\min_w \frac{1}{n} \sum_{i=1}^n \ell(y_i \langle w, x_i \rangle)$$

where ℓ is the linear loss function.

- ▶ How would we attempt to solve this?
- ▶ How is it related to the perceptron?

Recap: stochastic gradient descent

Consider a classifier $f_w(x)$ with parameter vector $w \in \mathbb{R}^d$.

Assume that $\ell(y_i, f_w(x))$ is differentiable w.r.t. w .

We want to minimize:

$$\min_{w \in \mathbb{R}^d} R_n(w) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f_w(x_i)).$$

Gradient descent:

- ▶ Start with some initial w_0
- ▶ Compute gradient:

$$\nabla_w R_n(w) = \frac{1}{n} \sum_{i=1}^n \nabla_w \ell(y_i, f_w(x_i))$$

- ▶ Update:

$$w_{t+1} = w_t - \eta \nabla_w R_n(w)$$

where η is the step size (learning rate)

Recap: stochastic gradient descent (2)

Stochastic gradient descent:

- ▶ Start with some initial w_0
- ▶ Randomly pick i and compute gradient:

$$\nabla_w \ell(y_i, f_w(x_i))$$

- ▶ Update:

$$w_{t+1} = w_t - \eta \nabla_w \ell(y_i, f_w(x_i))$$

Gradient of the linear loss

Consider the linear loss of a linear classifier:

$$\ell(\langle w, x_i \rangle, y_i) = \max\{0, -y_i \langle w, x_i \rangle\} = \begin{cases} 0 & \text{if } -y_i \langle w, x_i \rangle < 0, \\ -y_i \langle w, x_i \rangle & \text{otherwise} \end{cases}$$

The gradient with respect to w is given as:

$$\nabla \ell(\langle w, x_i \rangle, y_i) = \begin{cases} 0 & \text{if } -y_i \langle w, x_i \rangle < 0, \\ -y_i x_i & \text{otherwise} \end{cases}$$

where $-y_i x_i$ is update form of the perceptron

Perceptron runs SGD with linear loss

The perceptron update rule performs a simplified version of SGD (with $\eta = 1$ and one pass through all training points).

But by itself, it doesn't mean a lot...

- (1) Does it really converge to an ERM classifier that minimizes the training loss?
- (2) And what about the test error and generalization?

Mistake bound for the perceptron

Towards bounding the numbers of mistakes

Want to prove:

If the training points are "well-separable" then we can upper bound the number of update steps that are necessary to find a linear classifier that makes 0 error on the training points.

First, we need to clarify what it means that data is well-separable.

Distance of a point to the hyperplane

Consider a point x and a hyperplane H with normal vector w , assume $y = \langle w, x \rangle > 0$.

Let $h \in H$ be the point on H that is closest to x , and denote the distance between h and x by r :

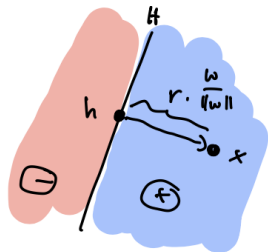
Then:

$$x = h + r \cdot \frac{w}{\|w\|}$$

$$\Rightarrow \langle x, w \rangle = \langle h + r \cdot \frac{w}{\|w\|}, w \rangle = \langle h, w \rangle + r \frac{\langle w, w \rangle}{\|w\|}$$

Note: $\langle h, w \rangle = 0$ and $\frac{\langle w, w \rangle}{\|w\|} = \|w\|$

$$\left. \begin{array}{l} \langle x, w \rangle = r \cdot \|w\| \Rightarrow r = \frac{\langle x, w \rangle}{\|w\|} \\ \text{Similarly if } \langle x, w \rangle < 0 \quad r = -\frac{\langle x, w \rangle}{\|w\|} \end{array} \right\} \Rightarrow r = y \frac{\langle x, w \rangle}{\|w\|}$$



The margin of separating hyperplane

Consider training points $(x_1, y_1), \dots, (x_n, y_n)$ and a linear classifier $\langle w, x \rangle$ that separates the training points. The margin is then defined as the smallest distance of all training points to the separating hyperplane H_w :

$$\begin{aligned} g &= \min \text{distance}(x_i, H_w) \\ &= \min_{i=1, \dots, n} \frac{y_i \langle w, x_i \rangle}{\|w\|} \end{aligned}$$

Perceptron mistake bound

Theorem 10

Let $\{(x_i, y_i)\}_{i=1}^N$ be such that

$$\|x_i\| \leq R \quad \text{for all } i.$$

Assume that there exists a separating hyperplane induced by $w^* \in \mathbb{R}^d$ that has margin $\rho > 0$ on the training points.

Then the number of mistakes made by the perceptron algorithm is bounded by

$$\frac{R^2}{\rho^2}.$$

In particular, after at most $\frac{R^2}{\rho^2}$ many rounds, the perceptron has found a linear classifier that makes 0 training errors. First proved by Novikoff (1962).

Proof

Step 1: Upper bound on w_t :

- Suppose at step t , the algorithm misclassifies point x_t . Then:

$$\begin{aligned} \|w_{t+1}\|^2 &= \|w_t + y_t x_t\|^2 \\ &\quad \text{(by using update rule from perceptron algorithm)} \\ &= \|w_t\|^2 + 2y_t \langle w_t, x_t \rangle + \|x_t\|^2 \\ &\quad \text{(use the fact } \langle w_t, x_t \rangle \leq 0 \text{ and } \|x_t\|^2 \leq R^2) \\ &\leq \|w_t\|^2 + R^2 \quad (\star) \end{aligned}$$

- Let $I \subset \{1, \dots, n\}$ be the indices where a mistake happened, denote $M = |I|$ as the total number of mistakes

Proof (2)

- We now take a sum over $(\star)$ for all $i \in I$:

$$\sum_{i \in I} \|w_{i+1}\|^2 \leq \sum_{i \in I} \|w_i\|^2 + M \cdot R^2$$

$$\sum_{i \in I} (\|w_{i+1}\|^2 - \|w_i\|^2) \leq M \cdot R^2$$

- Note that if no mistake happens, we have $\|w_{j+1} - w_j\| = 0$ (no update). Thus we can replace $\sum_{i \in I}$ by $\sum_{i=1}^n$. Thus we can apply a telescoping sum and get

$$\sum_{i \in I} (\|w_{i+1}\|^2 - \|w_i\|^2) = \sum_{i=0}^{n-1} \dots = \|w_n\|^2 - \|w_0\|^2 = \|w_n\|^2$$

since $\|w_0\|^2 = 0$.

- Thus,

$$\|w_n\|^2 \leq M \cdot R^2 \quad (**)$$

Proof (3)

Step 2: Lower bound on w_t :

- By the margin assumption, we know that for all x_i we have:

$$\rho \leq \frac{y_i \langle w_i^*, x_i \rangle}{\|w^*\|}$$

- We now sum this inequality over all $i \in I$ where a mistake happens:

$$M \cdot \rho \leq \sum_{i \in I} \frac{y_i \langle w^*, x_i \rangle}{\|w^*\|} = \frac{1}{\|w^*\|} \langle w^*, \sum_i y_i x_i \rangle$$

(now use Cauchy-Schwartz)

$$\leq \frac{1}{\|w^*\|} \|w^*\| \left\| \sum_i y_i x_i \right\| = \left\| \sum_i y_i x_i \right\|$$

Proof (4)

- Note that for mistakes at i we have the update $w_{t+1} = w_t + y_i x_i$. Plugging this into the previous inequality we get:

$$M \cdot \rho \leq \left\| \sum_i y_i x_i \right\| = \left\| \sum_i w_{i+1} - w_i \right\| = \|w_n\| \quad (\#)$$

Note: used above the telescoping sum again

Step 3:Combine. Plugging together $(\star\star)$ and $(\#)$ gives:

$$M^2 \cdot \rho^2 \leq \|w_n\|^2 \leq M \cdot R^2$$

$$M \leq \frac{R^2}{\rho^2}$$



Discussing the bound

- ▶ The bound does not depend on the dimension, cool!
- ▶ Somewhere implicit in the proof: SGD does not pick just any odd ERM minimizer. It picks a special one, and we can prove nice guarantees
→ see implicit regularization of SGD later
- ▶ The bound only holds under the assumptions on the data distribution (margin).
- ▶ Margins pop up at many places in ML (SVMs, boosting, ...)
- ▶ But... the theorem only consider the separable case. In the non-separable case (or the 0-margin case), no bound exist.
- ▶ Interesting follow-up: how does the bound behave if we introduce a learning rate η and define the update rule:

$$w_{t+1} = w_t + \eta y_t x_t \quad ?$$

→ see assignment.

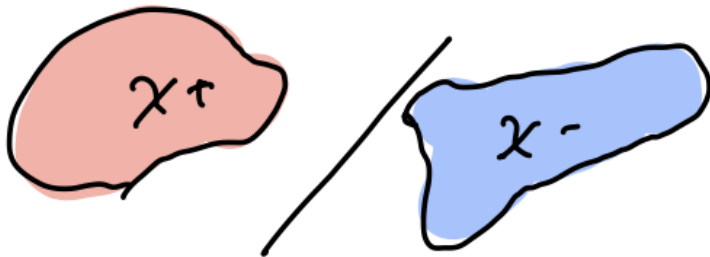
- ▶ And obviously, the bound only considers the training errors.
What about generalization?

Generalization error of the perceptron

Assumption: distribution is linearly separable

To prove our generalization bound, we will make strong assumption:

1. There exists a deterministic function $f^* : X \rightarrow Y$ that produces the true labels, that is $y = f^*(x)$
2. We assume that the set $\mathcal{X}^+ = \{x \in \mathcal{X} | f(x) = +1\}$ and $\mathcal{X}^- = \{x \in \mathcal{X} | f(x) = -1\}$ are linearly separable: there exists a hyperplane with parameters w^* such that $\mathcal{X}^+$ and $\mathcal{X}^-$ are on different sides:



Theorem 11

Consider a distribution P on $X \times Y$ that can be perfectly separated by a hyperplane. Let $S_{n+1} = \{(x_1, y_1), \dots, (x_{n+1}, y_{n+1})\}$ be sampled i.i.d. from this distribution.

Let $w(S_n)$ be the perceptron classifier trained on $S_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$, where we trained the perceptron until it makes no mistakes on S_n .

Denote by $R(S_{n+1})$ the radius of the ball that contains all samples in S_{n+1} , and $\rho(S_{n+1})$ the largest margin that can be achieved by any hyperplane on S_{n+1} .

Then the probability that the classifier trained on S_n makes an error on S_{n+1} is bounded by

$$P(y_{n+1} \langle w(S_n), x_{n+1} \rangle < 0) \leq \frac{1}{n+1} E \left(\frac{R^2(S_{n+1})}{\rho^2(S_{n+1})} \right)$$

Discussion: Scaling with d

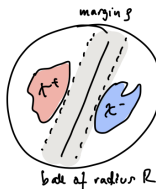
- ▶ The generalization error does not depend on the dimension of the underlying space!
- ▶ That is cool!

Discussion: Scaling with n

- ▶ Consider a scenario where the underlying distribution lives inside a ball of radius R^* , and there exists a separating hyperplane that separates all points of X by margin ρ^* .
- ▶ Then for all sets S_{n+1} we have $R(S_{n+1}) \leq R^*$, $\rho(S_{n+1}) \geq \rho^*$ and thus

$$P(\text{error}) \leq \frac{1}{n+1} E \left(\frac{R^2(S_{n+1})}{\rho^2(S_{n+1})} \right) \leq \frac{1}{n+1} \frac{(R^*)^2}{(\rho^*)^2}$$

- ▶ So the generalization error (in the 0-1-loss) decreases with the rate $\frac{1}{n}$.
- ▶ This is cool!
- ▶ In learning theory, this is called a "fast rate".



Proof

Idea of the proof: Leave-one-out scenario

- Consider the augmented sample:

$$S_{n+1} = Z_1, Z_2, \dots, Z_n, Z_{n+1}$$

with the Z_i $i \in 1, \dots, n$ as training samples and Z_{n+1} as test sample.

Also the leave-one-out set where sample k is removed from S_{n+1} :

$$S^{-k} = Z_1, \dots, Z_{k-1}, Z_{k+1}, \dots, Z_n, Z_{n+1}$$

- Since the data is i.i.d., in probability or in expectation, it does not make a difference whether we train on S_n and test on Z_{n+1} or train on S^{-k} and test on Z_k .
- Denote by $w(S^{-k})$ the weight vector after training on S^{-k} with error 0.

Proof (2)

► Then:

$$P(y_{n+1}\langle w(S_n), x_{n+1} \rangle < 0) = E(\mathbb{1}\{y_{n+1}\langle w(S_n), x_{n+1} \rangle < 0\})$$

(same expectations on all S^{-k})

$$= \frac{1}{n+1} \sum_{k=1}^{n+1} E(\mathbb{1}\{y_k \langle w(S^{-k}), x_k \rangle < 0\})$$

$$= \frac{1}{n+1} E\left(\sum_{k=1}^{n+1} \mathbb{1}\{y_k \langle w(S^{-k}), x_k \rangle < 0\}\right) \quad (\star)$$

(now consider the summation part)

Proof (3)

- ▶ Now consider running the perceptron on the whole sequence $Z_1, Z_2, \dots, Z_n, Z_{n+1}$ (from scratch). By mistake bound we know that the number M of errors it makes is bounded by $\frac{R^2(S_{n+1})}{\rho^2(S_{n+1})}$.
- ▶ Denote by $i_1, \dots, i_M$ the indices on which the algorithm makes a mistake in any of its cycles over the data. If $k \notin \{i_1, \dots, i_M\} = J_{n+1}$, the output of the algorithm is the same even if we remove Z_k from the training sample S_{n+1} and instead train on S^{-k} .
- ▶ Thus:

$$\begin{aligned}
 \sum_{k=1}^{n+1} \mathbb{1}\{y_k \langle w(S^{-k}), x_k \rangle < 0\} &= \sum_{k \in J_{n+1}} \mathbb{1}\{y_k \langle w(S_n), x_k \rangle < 0\} \\
 &\quad (\text{as } \mathbb{1}\{y_k \langle w(S_n), x_k \rangle < 0\} \leq 1) \\
 &\leq |J_{n+1}| \\
 &= M(S_{n+1})
 \end{aligned}$$

Proof (4)

► Finally, we get:

$$(*) \leq \frac{1}{n+1} E(M(S_{n+1})) \leq \frac{1}{n+1} E\left(\frac{R^2(S_{n+1})}{\rho^2(S_{n+1})}\right)$$

where the right side is a mistake bound of the previous theorem.



Discussion proof technique: leave-one-out principle

- ▶ Implicitly our proof exploits a general insight:
- ▶ The average leave-one-out error of sample of size $n + 1$ is an unbiased estimate of the average generalization error for samples of size n .
- ▶ (Proof: Holin p.69/70, Lemma 4.1)

More discussion: stability

- ▶ The proof technique also hints at the important principle that we will discuss in more detail later: centred () stability
- ▶ Stability investigates how much the result of a learning algorithm changes if we remove one training point or replace it by different one.
- ▶ One can prove very strong generalization bounds with this technique.

Outlook: What if data is not linearly separable?

Then one can prove more general versions of the theorem: Mistake bound:

Given margin ρ , define the Hinge loss of point (x, y) as

$$\delta(x, \rho) := \max\{0, \rho - y\langle w, x \rangle\} \quad \text{and}$$

$$D := \left(\sum_{i=1}^n \delta(x_i, \rho)^2 \right)^{1/2}.$$

Then the number of updates by the perceptron is bounded by

$$M \leq \frac{(R + D)^2}{\rho^2}.$$

(first proved by Freund/Schapire, 1999; we skip the proof, but it is not difficult; see the Mohri book if you are interested)

Generalization bound: more complicated, skipped. only scales as $1/\sqrt{n}$...

Bounding the complexity of the function class

Statistical Learning Theory

Literature on (classical) statistical 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 “classical” 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 (less technical) and Francis Bach (pretty technical).

We consider the standard statistical setup as introduced earlier:

- ▶ there exists an (unknown) probability distribution P that generates the data.
- ▶ In this section, we only consider the principle of binary classification, typically with the 0-1-loss (this makes the maths simpler, many results also hold more generally).
- ▶ The perfect prediction function would be the one that minimizes the risk (expected loss) over the distribution P .
- ▶ We receive training data that is sampled i.i.d. from distribution P
- ▶ Because we do not know P , we perform empirical risk minimization (ERM) over a fixed function class $\mathcal{F}$ that we chose in advance, before even seeing the data.
- ▶ We do not consider the question of how to solve this optimization problem; we simply assume that we always find an empirical risk minimizer in $\mathcal{F}$
- ▶ We now ask under which conditions this approach is “successful”.
- ▶ We will see that one approach that leads to successful machine learning then consists in bounding the complexity of the function class $\mathcal{F}$.

Recap, convergence and consistency

Let us revisit and refine the notions of convergence in the ERM setting. 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.

Recap, convergence and consistency (2)

Now consider the following definitions:

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 .

Note that consistency with respect to a fixed function class $\mathcal{F}$ only concerns the estimation error, not the approximation error:

Important tool: Concentration inequalities

Literature:

- ▶ Bach book, Sec. 1.2
- ▶ Hoeffding inequality is treated in pretty much any text book on learning theory.
- ▶ McDiarmid's inequality has been proved in this paper: Colin McDiarmid "On the Method of Bounded Differences" Surveys in Combinatorics, 1989
- ▶ There are many, many concentration inequalities, and if you want to dive into learning theory you need to learn much more about them. If you want to know details, consider the book "Concentration inequalities" by Boucheron, Lugosi, Massart.

Concentration inequalities

We have seen the law of large numbers before:

Proposition 12 (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 $E(Z)$. Define their sum $S_n := (1/n) \cdot \sum_{i=1}^n Z_i$. Then (under mild assumptions)

$$S_n \rightarrow E(Z) \quad (\text{almost surely}).$$

In learning theory, we often want quantitative versions that tell us “how far” the empirical average is from the true average, and with which probability.

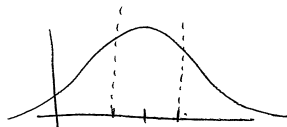
Concentration inequalities (2)

Strong such statements are called concentration inequalities. They take the form:

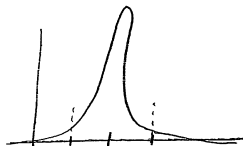
With probability at most $\delta(\varepsilon)$, the absolute difference between the empirical mean S_n and the true mean $E(Z)$ is larger than ε :

$$P(|S_n - E(S_n)| \geq \varepsilon) \leq \delta(\varepsilon)$$

where $\delta(\varepsilon)$ goes very fast to 0 as ε grows:



$E(X)$
large deviations occur often
not concentrated



$E(X)$
large deviations are extremely
unlikely $\leadsto$ concentrated

Hoeffding inequality

One of the simplest concentration inequalities is this one:

Proposition 13 (Concentration inequality, Chernoff 1952, Hoeffding 1963)

Assume that the random variables $Z_1, \dots, Z_n$ are independent and take values in $[0, 1]$. Then for any $\varepsilon > 0$,

$$P(|S_n - E(S_n)| \geq \varepsilon) \leq 2 \exp(-2n\varepsilon^2).$$

Proof skipped.

Hoeffding inequality (2)

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.
- ▶ Whenever you want to prove concentration, Hoeffding is always the first thing you try!!!
- ▶ The Hoeffding inequality is tight in some situations (eg fair coin tosses), and not so tight in others. If you know more about the random variables, you might consider other inequalities.

McDiarmid inequality

The following theorem is called McDiarmid's inequality or Bounded-differences inequality:

Theorem 14 (McDiarmid)

Consider a function $f : \mathcal{X}^n \rightarrow \mathbb{R}$. Assume there exist constants $c_1, \dots, c_n > 0$ such that for all $x_1, \dots, x_n \in \mathcal{X}$ and $x'_i \in \mathcal{X}$ it holds that

$$|f(x_1, \dots, x_i, \dots, x_n) - f(x_1, \dots, x'_i, \dots, x_n)| \leq c_i$$

Let $X_1, \dots, X_n \in \mathcal{X}$ be a set of independent random variables. Then

$$P(|f(X_1, \dots, X_n) - E(f(X_1, \dots, X_n))| \geq \varepsilon) \leq 2 \exp\left(\frac{-2\varepsilon^2}{\sum_{i=1}^n c_i^2}\right)$$

McDiarmid inequality (2)

- ▶ Observe that the Hoeffding inequality is a special case of McDiarmid's inequality (HOW EXACTLY)?
- ▶ McDiarmid is a super-useful inequality, not only in learning theory but for example also in the field of randomized algorithms.
- ▶ We will later revisit it in the section on stability.

Controlling the estimation error: generalization bounds

Estimating the error of a fixed function

Estimating the error of a fixed function

Now consider our scenario of binary classification. We have already seen that we can use the LLN to prove strong consistency of binary classification with a fixed function f_0 . We can get a sharper result by the Hoeffding inequality:

Proposition 15 (Error in measuring the risk for a fixed function)

Fix a function $f_0 \in \mathcal{F}$, and consider a bounded loss function. Then,

$$P(|R_n(f_0) - R(f_0)| \geq \varepsilon) \leq \exp(-2n\varepsilon^2)$$

Estimating the error of a fixed function (2)

Proof:

- ▶ Apply the Hoeffding bound to the variables
- ▶ Observe that this leads to convergence in probability (weak consistency).
- ▶ To get almost sure convergence, one needs to do one extra step, namely apply the Borel-Cantelli lemma from probability. Key is that $\sum_{n=1}^{\infty} \exp(-2n\varepsilon^2)$ is finite. This then leads to strong consistency. We will not dive into this technique.



Estimating the error of a fixed function (3)

Turning the statement into a generalization bound:

- ▶ In machine learning, we often fix an error probability $\delta > 0$, say 0.05, and want to know how many sample points we would need to achieve this error level.
- ▶ To this end, we take the error term in the Hoeffding bound, set it equal to δ and then solve for ε :

$$\delta = \exp(-2n\varepsilon^2) \implies \varepsilon = \sqrt{\frac{\log(1/\delta)}{2n}}$$

- ▶ Now the statement reads:

with probability at least $1 - \delta$,

$$|R(f_0) - R_n(f_0)| \leq \sqrt{\frac{\log(1/\delta)}{2n}}$$

Sample complexity of error estimation

From the two versions of stating the generalization bound, we can deduce how many sample points we roughly need to get the error small. We treat the error probability as a small fixed constant to derive:

- ▶ From the first version of the generalization statement we see that if we want to get error at most ε with small fixed probability, we need that $n\varepsilon^2 = O(1)$, that is $n = O(\frac{1}{\varepsilon^2})$
- ▶ From the second version we immediately see that the error we get from n sample points scales as $1/\sqrt{n}$.

For example, if you already fixed a function f_0 by some complicated training procedure, and you now want to use a new sample to estimate the test error up to ε accuracy, then you need of the order $1/\varepsilon^2$ many test points. If $\varepsilon = 1\%$, you need $1/\varepsilon^2 = 10.000$ sample points!

And it is impossible to beat or trick these numbers.

Uniform convergence is necessary and sufficient for consistency

Uniform convergence

The bound we have just derived is only true for a fixed function f_0 , and we have already discussed that this does NOT imply that our ERM classifier is consistent.

Can we define a condition that is sufficient for the convergence of the empirical risk of the data-dependent function f_n ?

Idea:

- ▶ 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.
- ▶ 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.
- ▶ So what we want is that for any $\varepsilon > 0$, with high probability it holds that

$$\sup_{f \in \mathcal{F}} |R_n(f) - R(f)| < \varepsilon$$

Uniform convergence (2)

Definition 16

We say that $R_n(f)$ converges to $R(f)$ **uniformly** over $\mathcal{F}$ or that the **law of large number holds uniformly** over a function class $\mathcal{F}$ if

$$\forall \varepsilon > 0 : \lim_{n \rightarrow \infty} P(\sup_{f \in \mathcal{F}} |R_n(f) - R(f)| > \varepsilon) = 0$$

Uniform convergence: sufficient for consistency

Relatively easy to see:

Proposition 17 (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.

$$|R(f_n) - R(f_{\mathcal{F}})|$$

(by definition of $f_{\mathcal{F}}$ we know that $R(f_n) - R(f_{\mathcal{F}}) \geq 0$)

$$= R(f_n) - R(f_{\mathcal{F}})$$

$$= R(f_n) - R_n(f_n) + R_n(f_n) - R_n(f_{\mathcal{F}}) + R_n(f_{\mathcal{F}}) - R(f_{\mathcal{F}})$$

(note that $R_n(f_n) - R_n(f_{\mathcal{F}}) \leq 0$ by def. of f_n)

$$\leq R(f_n) - R_n(f_n) + R_n(f_{\mathcal{F}}) - R(f_{\mathcal{F}})$$

$$\leq 2 \sup_{f \in \mathcal{F}} |R(f) - R_n(f)|$$



Uniform convergence: necessary for consistency

What is much less obvious, and a deep result, is that uniform convergence is also necessary to achieve a (slightly different notion) of consistency:

Theorem 18 (Vapnik & Chervonenkis, 1971)

Let $\mathcal{F}$ be any function class. Then empirical risk minimization is (strictly) 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 notion of strict uniform consistency has been defined to prevent to artificially blow up the function class $\mathcal{F}$ by stupid functions that will not be picked by ERM, but that destroy uniform convergence; essentially, we want consistency even if we remove the good functions from $\mathcal{F}$.
- ▶ The proof of this result 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 $\mathcal{F}$???

Generalization bound for 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

$$P\left(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| > \varepsilon\right)$$

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:

$$\begin{aligned} & P(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| \geq \varepsilon) \\ &= P(\sup_{i=1, \dots, m} |R(f_i) - R_n(f_i)| \geq \varepsilon) \\ &= P(|R(f_1) - R_n(f_1)| \geq \varepsilon \text{ or } |R(f_2) - R_n(f_2)| \geq \varepsilon \text{ or } \dots) \\ &\leq \sum_{i=1}^m P(|R(f_i) - R_n(f_i)| \geq \varepsilon) \\ &\leq 2m \exp(-2n\varepsilon^2) \end{aligned}$$

Generalization bound for finite classes (2)

Leads to the first result:

Proposition 19 (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

$$|R(f) - R_n(f)| < \varepsilon.$$

Again converting the statement in its other form (setting the probability to δ and solving for ε) results in

$$\delta = m \cdot 2 \exp(-2n\varepsilon^2) \implies \varepsilon = \sqrt{\frac{\log(2m) + \log(1/\delta)}{2n}}$$

With this, the proposition becomes the following **generalization bound**:

Generalization bound for finite classes (3)

Theorem 20 (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 (4)

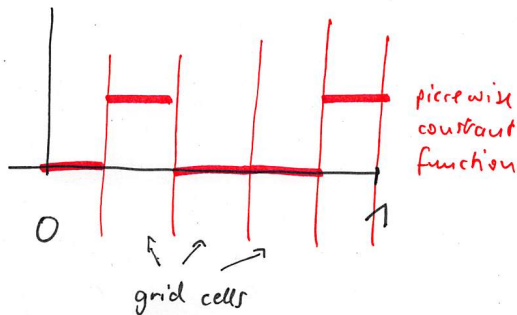
Let's digest this bound:

- ▶ 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 sample complexity of learning from a finite class scales as $\sqrt{(\log m)/n}$. As before, we have the $1/\sqrt{n}$ behavior, and for the size of the function class we pay logarithmically in m .
- ▶ 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.

Example: partitioning-based estimators

Partitioning-based estimators

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 $\mathcal{F}_k$.



Partitioning-based estimators (2)

We want to study the question how to choose the number k of partitions depending on the sample size n .

Extreme case: $k = 1$

- ▶ Then the estimation error will converge to 0 really quickly (why?)
- ▶ The approximation error will remain large (why?)

Other extreme case: $k \approx n$

- ▶ Then the estimation error is going to be large (why?)
- ▶ The approximation error might (?) be small (why?)

Partitioning-based estimators (3)

Intuition:

- ▶ To get the estimation error to 0, we need to see “infinitely many points” in each of the intervals, to consistently estimate the mean.
- ▶ To get the approximation error to 0, we need the number of partitions k to go to ∞ , because we need to approximate a possibly complicated Bayes function with a piecewise constant function.

Partitioning-based estimators in 1d, k fixed

Case 1: k is fixed.

- ▶ Then ERM is uniformly consistent with respect to $\mathcal{F}_k$.

Proof: The function class has 2^k many functions. Now just apply the above bound: The key error term is of the order $\log(m)/n = \log(2^k)/n \approx k/n$, and this goes to 0 if $n \rightarrow \infty$ and k is fixed.

- ▶ Note: this means that the estimation error goes to 0.
- ▶ Question: is the classifier also Bayes consistent? (Does the approximation error go to 0?)

Partitioning-based estimators in 1d, k growing with n

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 $\mathcal{F}_{k(n)}$?
- ▶ By the same argument as above, the error term is of the order $k(n)/n$. So the bound only goes to 0 if $k(n)/n \rightarrow 0$. For example, we could choose $k(n) = \log n$.
- ▶ Intuition: In each of the k cells, we have of the order n/k many points if they are uniformly distributed. If $k/n \rightarrow 0$, we thus have $n/k \rightarrow \infty$. So even if the cells get smaller in diameter (because k grows), the number of data points per cell grows. So eventually, we average over infinitely many points in each cell, which leads to a consistent estimate of the mean value in this cell. Hence minimizing the empirical risk will eventually also minimize the true risk.

Partitioning-based estimators in 1d, k growing with n (2)

- What about the approximation error / Bayes consistency?

If $k(n)/n \rightarrow 0$, the estimation error goes to 0.

If we also want that the approximation error goes to 0 we additionally need to require $k \rightarrow \infty$. One can now invoke maths / measure theory to prove that $k \rightarrow \infty$ really allows us to approximate the Bayes classifier by finite histograms, for all probability measures P (skipped here).

So if we have both $k \rightarrow \infty$ and $k(n)/n \rightarrow 0$, and if our distribution satisfies some regularity assumptions (so that it can be well-approximated by finite intervals), then we will get Bayes consistency.

Partitioning-based estimators in higher dimensions

- ▶ Consider a grid which, in each dimension, splits into k cells. Then we have k^d cells in total, and the number of functions in our class is 2^{k^d} . By the same argument as above, the error term that is of order $\log(2^{k^d})/n \approx k^d/n$. We get convergence if it goes to 0 — but note that we need a huge amount of samples to achieve this (exponentially many in d).
- ▶ This already hints that this is not a very practical idea...

Partitioning-based estimators in higher dimensions (2)

Final note of caution:

The above results for partitioning-based estimators only hold if the partitions are formed independently of the data (before we see the data).

The results do not a priori hold if the partitions do depend on the data, as it would be the case for k -nearest neighbor estimators, decision trees, or random forests. For these type of classifiers, one needs to apply different techniques.

Infinite classes, symmetrization and bounds with the shattering coefficient

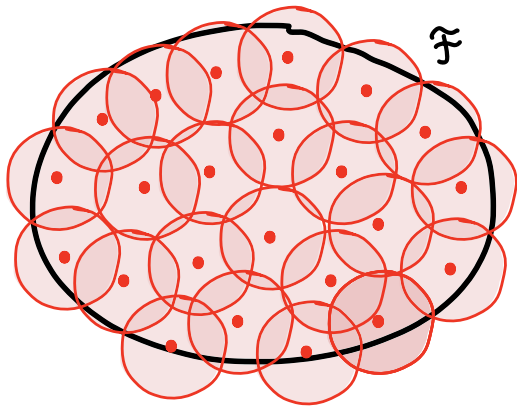
What about infinite function classes?

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?

Approach 1: geometric argument with covering numbers



Cover $\mathcal{F}$ with m small balls
(finitely many!)

Control deviations within a ball

Overall error = finite class error with m functions + deviations

Approach 1: geometric argument with covering numbers (2)

Proposition 21 (Simple bound with covering numbers)

Fix an error probability $0 < \delta < 1$. Fix a radius $r > 0$ and a norm $\|\cdot\|$ on $\mathcal{F}$. Assume

- ▶ that $\mathcal{F}$ can be covered by $C(\mathcal{F}, \|\cdot\|, r)$ many balls of radius r
- ▶ that there exists $\eta > 0$ such that for all centers $f \in \mathcal{F}$ and for all $f_1, f_2 \in B_r(f)$,
 $R(f_1) - R(f_2) \leq \eta$ and $R_n(f_1) - R_n(f_2) \leq \eta$

Then with probability at least $1 - \delta$, for all $f \in \mathcal{F}$,

$$R(f) \leq R_n(f) + \sqrt{\frac{\log(2C(\mathcal{F}, \|\cdot\|, r)) + \log(1/\delta)}{2n}} + 2\eta$$

Approach 1: geometric argument with covering numbers (3)

Proof idea: Assume that $\mathcal{F}$ is covered by m balls with centers $f_1, \dots, f_m \in \mathcal{F}$.

- For an arbitrary $f \in \mathcal{F}$, let f_i the closest center of a ball to f . Then:

$$\begin{aligned} |R(f) - R_n(f)| &= |R(f) - R(f_i) + R(f_i) - R_n(f_i) + R_n(f_i) - R_n(f)| \\ &\leq |R(f) - R(f_i)| + |R(f_i) - R_n(f_i)| + |R_n(f_i) - R_n(f)| \end{aligned}$$

- Then bound overall worst case error over $\mathcal{F}$:

$$\sup_{f \in \mathcal{F}} |R_n(f) - R(f)| \leq \sup_{f \in \{f_1, \dots, f_m\}} |R_n(f) - R(f)| + 2\eta$$

- To the red term, apply our previous bound for a finite function class.



Example: linear regression with absolute loss

- ▶ Consider regression on $[0, 1]$ with linear functions with bounded slope:
 $\mathcal{F} = \{h_a(x) = ax : a \in [0, 1]\}$.
- ▶ For an unknown target function $g : [0, 1] \rightarrow [0, 1]$, we consider the L_1 -loss function:
 $\ell(h(x), g(x)) = |h(x) - g(x)|$
- ▶ We consider η -balls in $\mathcal{F}$ with respect to the infinity-norm:

$$B_\eta(h) = \{\tilde{h} \in \mathcal{F} : \|h - \tilde{h}\|_\infty \leq \eta\}.$$

Observe that for two functions with slopes a and b we have

$$\|h_a - h_b\|_\infty = \sup_{x \in [0, 1]} |ax - bx| = |a - b|$$

So if $h_a \in B_\eta(h_b)$ then $|a - b| \leq \eta$.

Example: linear regression with absolute loss (2)

- ▶ For any $h_a \in B_\eta(h_b)$, we can bound their difference in loss:
For every x ,

$$|\ell(h_a(x), g(x)) - \ell(h_b(x), g(x))| \leq |h_a(x) - h_b(x)| \leq \eta.$$

- ▶ To cover the space, we can use the balls with slopes $a_0 = 0, a_1 = \eta, a_2 = 2\eta, \dots$ (observe that by definition, the maximal slope of our functions is 1). Overall, we need of the order $1/\eta$ many balls to cover the space, so $C(\mathcal{F}, \eta) \approx 1/\eta$.
- ▶ Altogether, with probability at least $1 - \delta$, we have that for all $f \in \mathcal{F}$,

$$R(f) \leq R_n(f) \leq 2\eta + \sqrt{\frac{\log(2/\eta) + \log(1/\delta)}{2n}}.$$

- ▶ Note that we can choose any η we want. For example, if we choose $\eta = 1/n$ we obtain an overall bound of the order $\sqrt{\log(n)/n}$.

Example: linear regression with absolute loss (3)

Extending this example to a d -dimensional space, one will find that the covering numbers are of the order of $C(\mathcal{F}, \eta) = (1/\eta)^d$. So the crucial term in the bound gets an extra factor of d :

$$\log(C(\mathcal{F}, \eta)) = d \log(1/\eta)$$

Approach 2: Shattering coefficient

In case of binary classification, we can consider an approach with a combinatorics flavor that takes into account the training data. We observe that even if we have infinitely many functions in $\mathcal{F}$, on the training data they can only do finitely many things.

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 $\mathcal{F}_{X_1, \dots, X_n}$:

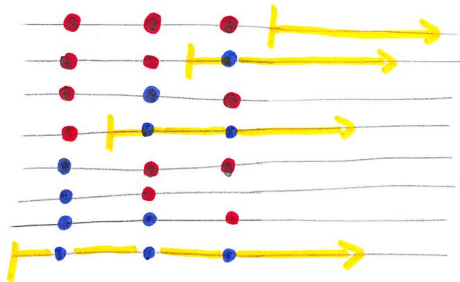
$$\mathcal{N}(\mathcal{F}, n) := \max\{|\mathcal{F}_{X_1, \dots, X_n}|; X_1, \dots, X_n \in \mathcal{X}\}$$

Approach 2: Shattering coefficient (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.

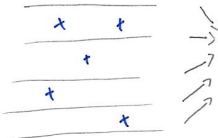
$$\text{Thus } \mathcal{N}(\mathcal{F}, 3) = 4.$$

(because this argument holds for all possible sets of three points)

Approach 2: Shattering coefficient (3)

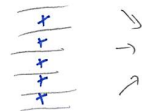
Example 2: $\mathcal{X} = \mathbb{R}^2$, $\mathcal{F}$ such that positive class = space above a horizontal line

Data 1:



5 different ways to separate them $\Rightarrow \mathcal{N}(\mathcal{F}, 5) \geq 5$

Data 2:



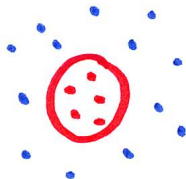
6 different ways $\Rightarrow \mathcal{N}(\mathcal{F}, 5) \geq 6$

Data 3:

$x \ x \ x \ x \ x$ two different ways $\Rightarrow \mathcal{N}(\mathcal{F}, 5) \geq 2$

Approach 2: Shattering coefficient (4)

Example 3: $\mathcal{X} = \mathbb{R}^2$, $\mathcal{F} = \text{interior of circles}$.



CAN YOU COME UP WITH A BOUND ON THE SHATTERING COEFFICIENT FOR A SMALL n ?

Shattering coefficient: generalization bound

One can then prove the following bound, but some tricks are required (see below!):

Proposition 22 (Generalization bound with shattering coefficient)

Let $\mathcal{F}$ be any arbitrary function class. Then for all $0 < \varepsilon < 1$,

$$\mathbb{P}\left(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| > \varepsilon\right) \leq 2\mathcal{N}(\mathcal{F}, 2n) \exp(-n\varepsilon^2/4).$$

The other way round: With probability at least $1 - \delta$, all functions $f \in \mathcal{F}$ satisfy

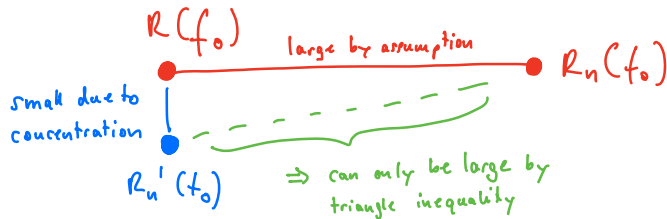
$$R(f) \leq R_n(f) + 2\sqrt{\frac{\log(\mathcal{N}(\mathcal{F}, 2n)) - \log(\delta)}{n}}.$$

Proof of the shattering bound 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 23 (Symmetrization lemma)

$$P\left(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| > \varepsilon\right) \leq 2 P\left(\sup_{f \in \mathcal{F}} |R_n(f) - R'_n(f)| > \varepsilon/2\right).$$



Proof of the shattering bound by symmetrization (2)

Proof intuition:

- ▶ Draw the original sample.
- ▶ If $\sup_{f \in \mathcal{F}} |R(f) - R_n(f)|$ is large, this can only be the case if there exists some $f_0 \in \mathcal{F}$ such that $|R(f_0) - R_n(f_0)|$ is large. Fix this function f_0 .
- ▶ Now draw the independent ghost sample. Because f_0 has been fixed in advance, with high probability we have concentration: $|R(f_0) - R'_n(f_0)|$ is small.
- ▶ But together, this implies that $|R'_n(f_0) - R_n(f_0)|$ is large (see figure).
- ▶ If it is already large for f_0 , the sup can only make it larger.

Proof of the shattering bound by symmetrization (3)

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 $\mathcal{N}(\mathcal{F}, 2n)$ (we have $2n$ in the shattering coefficient because of n points both in original and in ghost sample)
- ▶ Now Prop. 22 is a consequence of the bound for finitely many functions.

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.

The growth function

Sometimes, if one wants to stress how the shattering coefficient behaves with n , one also defines the **growth function**:

$$S_{\mathcal{F}}(n) : \mathbb{N} \rightarrow \mathbb{N}, n \mapsto \mathcal{N}(\mathcal{F}, n)$$

Another common notations for this function is $\Pi_{\mathcal{F}}(n)$.

VC dimension

Definition, Sauer-Shelah lemma and generalization bound

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

$$X = \mathbb{R}, \mathcal{F} = \{\mathbb{1}_{[a,b]} \mid a, b \in \mathbb{R}\} \quad \text{"closed intervals"}$$

- 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 an axis-aligned rectangle

$$X = \mathbb{R}^2, \mathcal{F} = \text{axis parallel rectangles}$$

- 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)

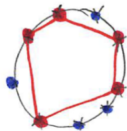
Example: positive class = interior of a convex polygon

$$X = \mathbb{R}^2, \mathcal{F}_d = \text{convex polygons with } d \text{ corners}$$

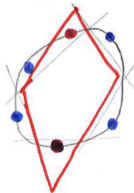
Then: $VC = 2d + 1$



- 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

$$X = \mathbb{R}, \mathcal{F} = \{\operatorname{sgn}(\sin(tx)) \mid t \in \mathbb{R}\}$$

Then $VC = \infty$.



VC dimension: Definition (6)

Finally, examples that are relevant for practice:

- ▶ $\mathcal{X} = \mathbb{R}^d$, $\mathcal{F} =$ linear hyperplanes. Then $VC(\mathcal{F}) = d + 1$.
- ▶ $\mathcal{X} = \mathbb{R}^d$, $\rho > 0$, $\mathcal{F}_\rho :=$ linear hyperplanes with margin at least ρ . Then one can prove:
if the data points are restricted to a ball of radius R , then

$$VC(\mathcal{F}) = \min \left\{ d, \frac{2R^2}{\rho^2} \right\} + 1$$

Proofs see below.

VC dimension: Sauer-Shelah Lemma

Why are we interested in the VC dimension? Here is the reason:

Proposition 24 (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

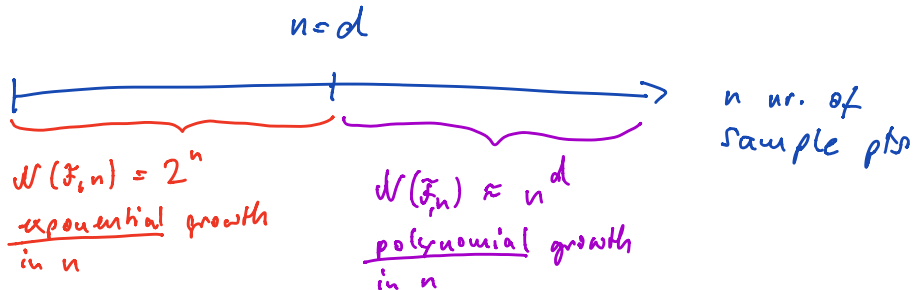
$$\mathcal{N}(\mathcal{F}, n) \leq \left(\frac{en}{d}\right)^d.$$

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 22 immediately gives the following generalization bound in terms of the VC dimension:

Theorem 25 (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 26

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.

VC dimension of linear classifiers and margin-based classifiers

Literature:

- ▶ The VC dim of linear classifiers: many text books.
- ▶ The large margin proof is nicely written up in Hush, Scovel: On the VC Dimension of Bounded Margin Classifiers. Machine learning, 2001.
- ▶ Neural networks:

VC dimension of linear classifiers

Consider the set $\mathcal{F}$ of linear classifiers on $\mathbb{R}^d$:

$$f_{w,b} : \mathbb{R}^d \rightarrow \{-1, 1\}, f_{w,b}(x) = \text{sign}(\langle w, x \rangle + b)$$

$$\mathcal{F} := \{f_{w,b} \mid w \in \mathbb{R}^d, b \in \mathbb{R}\}$$

Proposition 27 (VC dim of linear classifiers)

The VC dimension of the set $\mathcal{F}$ of linear classifiers on $\mathbb{R}^d$ is $d + 1$

VC dimension of linear classifiers (2)

Proof sketch:

Step 1: $\mathcal{F}$ can shatter a set of $d + 1$ points.

- ▶ Take the points on the simplex, $x_0 = 0, x_1 = e_1, \dots, x_d = e_d, 0$ where e_i is the i -th basis vector.
- ▶ Observe that $f_{w,b}(x_0) = b$ and for $i \geq 1$: $f_{w,b}(x_i) = w_i + b$.
- ▶ Now choose any labels $y_0, \dots, y_d \in \{\pm 1\}$.
- ▶ Set $b = y_0$, $w_i = y_i - b$
- ▶ Works: we then have $f_{w,b}(x_i) = y_i$ for all i .

VC dimension of linear classifiers (3)

Step 2: $\mathcal{F}$ cannot shatter any set of $d + 2$ points.

- ▶ It is well known that any set of $d + 2$ points in $\mathbb{R}^d$ can be partitioned into two subsets P and N such that their convex hulls intersect (this is called Radon's theorem; it holds for all $d \geq 1$; the proof is elementary).
- ▶ Now assign the points in P the label $+1$, and the points in N the label -1 .
- ▶ If there would be a hyperplane that separates P and N , then it would also separate the convex hulls of P and N . But the Radon theorem tells us that this is impossible, because the two intersect.



VC dimension of linear classifiers (4)

Again, what have we proved?

Consider linear classification in a fixed dimension d . Then with high probability over the choice of the sample the following holds:

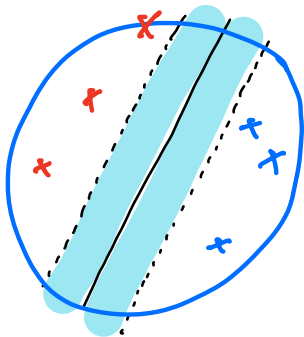
- ▶ We pick a sample of size n .
- ▶ For **all** functions $f \in \mathcal{F}$ it will be the case that $R(f) - R_n(f)$ is small.
- ▶ So in particular, it makes sense to select a function that has small $R_n(f)$ because this implicitly will also mean that $R(f)$ is small.

Consider linear classification in a setting where d grows linearly with n :

- ▶ In this scenario, the VC-bound would NOT converge to 0.
- ▶ This also makes sense intuitively: if we have as many dimensions as data points, it seems plausible that a linear classifier can memorize the labels of the existing data points, without generalization.

Linear classifiers with a margin

Assume we know that the data can not only be separated by a linear hyperplane, but even by a hyperplane with margin ρ (definition as in the perceptron lecture). Does this improve the VC dimension?



Linear classifiers with a margin (2)

Theorem 28 (VC dim of large margin hyperplanes)

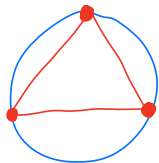
Let $X_1, \dots, X_n \in \mathbb{R}^d$ be a set of points that is contained in a sphere of radius $R > 0$. Let $\mathcal{H}_\rho$ be the set of all linear classifiers with margin ρ , restricted to the sphere. Then the VC-dimension of $\mathcal{H}_\rho$ satisfies

$$VC(\mathcal{H}_\rho) = \min \left\{ \frac{R^2}{\rho^2}, d + 1 \right\}$$

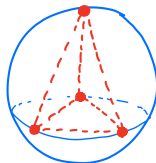
Linear classifiers with a margin (3)

Proof outline:

- ▶ We consider n points in $\mathbb{R}^d$ and want to find how large n can be, if we still want to be able to shatter the set of n points with margin ρ . Observe that this won't work at all if $n > d + 1$ (WHY?)
- ▶ We try to find a placement of n points in a ball of radius R in $\mathbb{R}^{n-1}$ such that all subsets can be separated with a margin as large as possible. The point configuration to consider is when the points form a regular n -simplex with vertices on a sphere of radius R in $\mathbb{R}^{n-1}$ (the simplex is the most "spread-out" set of points on the sphere). It satisfies $\|x_i\| = R$ and $\langle x_i, x_j \rangle = R^2/(n-1)$.



3-simplex in $\mathbb{R}^2$



4-simplex in $\mathbb{R}^3$

Linear classifiers with a margin (4)

- ▶ As next step, separate the points in a set P labeled $+1$ and a set N labeled -1 . The distance between the convex hulls of these sets corresponds to the margin.
- ▶ By elementary (but somewhat lengthy) computations one can find that the distance is minimal when P and N have the same number of points, and that then it is given by $R/\sqrt{n}$.
- ▶ Setting $\rho = R/\sqrt{n}$ and solving for n leads to $n = R^2/\rho^2$



Discussion

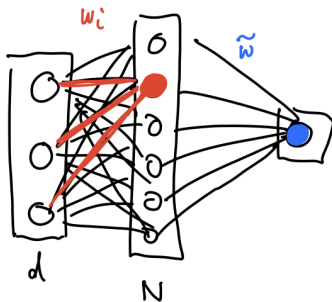
- ▶ The VC bound for linear hyperplanes depends on the dimension of the underlying space. It is natural to expect that this happens.
- ▶ But if the margin is large, the margin bound is independent of the dimension. This is cool!!! (Somehow, the large margin "eats" more space than the many dimensions might "gain", so the margin can overrule the dependence of the dimension.)
- ▶ But in order for it to work, we need extra knowledge on the underlying data distribution.
- ▶ We have seen the margin already, for the perceptron!

VC dimension of neural networks

VC dimension of a simple 2-layer neural network

Consider a simple neural network with one hidden layer and N hidden units:

- ▶ Input $x \in \mathbb{R}^d$
- ▶ N hidden units with activation function $f_i(x) = \text{sign}(\langle w_i, x \rangle + b_i)$ for $w_i, b_i \in \mathbb{R}^d$
- ▶ output $h(x) = \text{sign}(\langle \tilde{w}, x \rangle + b)$ for $\tilde{w}, b \in \mathbb{R}^d$



VC dimension of a simple 2-layer neural network (2)

Theorem 29

The VC-dimension of such a network is of order $O(N \cdot d \log(N \cdot d))$.

Proof sketch

Step 1: Hypothesis classes. Let $\mathcal{H}_i$ be the hypothesis class of neuron i :

$$\mathcal{H}_i = \{f | f(x) = \text{sign}(\langle w_i, x \rangle) + b_i\}$$

and let $\mathcal{H}'$ the hypothesis class of the function output unit:

$$\mathcal{H}' = \{f | f(x) = \text{sign}(\langle \tilde{w}, (f_1(x), \dots, f_N(x)) \rangle) + b\}$$

The final hypothesis class is a combination of the form:

$$\mathcal{H} = \mathcal{H}' \circ (\mathcal{H}_1 + \dots + \mathcal{H}_N)$$

Proof sketch (2)

Step 2: VC-dim and shattering coefficients of individual classes.

Because we consider the linear activation function, we have

$$VC(\mathcal{H}_i) = d + 1 \quad (\text{each neuron: } d \text{ input parameters})$$

$$\text{shattering coefficient } (\mathcal{H}_i) : \mathcal{N}(\mathcal{H}_i, n) \leq n^{d+1}$$

$$VC(\mathcal{H}') = N + 1 \quad (\text{last layer has } N \text{ parameters})$$

$$\text{shattering coefficient } (\mathcal{H}') : \mathcal{N}(\mathcal{H}', n) \leq n^{N+1}$$

Proof sketch (3)

Step 3: Combining them.

The final hypothesis class is a combination of the form

$$\mathcal{H} = \mathcal{H}' \circ (\mathcal{H}_1 \times \cdots \times \mathcal{H}_N)$$

and one can show (by independent lemma, skipped) that the shattering coefficient multiply:

$$\begin{aligned}\mathcal{N}(\mathcal{H}, n) &= \mathcal{N}\left(\mathcal{H}' \circ (\mathcal{H}_1 \times \cdots \times \mathcal{H}_N)\right) \\ &\leq \mathcal{N}(\mathcal{H}') \cdot \mathcal{N}(\mathcal{H}_1) \cdot \mathcal{N}(\mathcal{H}_2) \cdots \mathcal{N}(\mathcal{H}_N) \\ &\leq n^{N(d+1)+N+1} \quad \approx \quad n^{Nd}\end{aligned}$$

Proof sketch (4)

Step 4: From growth function to VC-dim.

Know: The VC-dim is the place where the growth function stops to grow exponentially.
Thus try to solve for n that satisfies

$$\begin{aligned}\mathcal{N}(\mathcal{H}, n) &< 2^n \\ n^{Nd} &\stackrel{!}{<} 2^n\end{aligned}$$

With a bit of math, this results in

$$n \approx Nd \log(Nd)$$



VC-dim of 2-layer neural networks

Typical results prove that a 2-layer NN (with thresholding, sigmoid, ReLU... activation function) has a VC dim of the order

$$\mathcal{O}(W \log W)$$

where W is the overall number of parameters in the NN.

Observe that this is consistent with the previous result, where the network had of the order $W = N \cdot d$ many parameters and we proved a VC-dim of $\mathcal{O}(d \cdot N \log(d N))$.

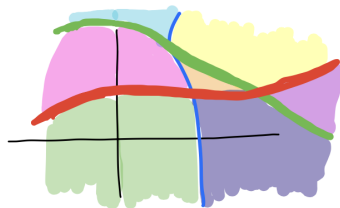
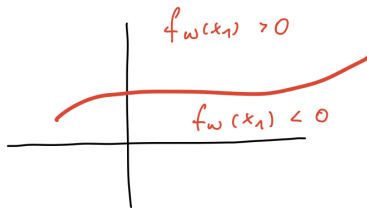
More general: VC-dim of neural networks

For multi-linear ReLU networks with L layers, the VC-dim is

$$O(L \cdot W \log W)$$

Key technique in the proofs: counting regions

- ▶ Figure denotes the set of all w parameters (not the data space!)
- ▶ Fix **one data point** x_1 . For all parameter vectors w , investigate whether they would classify x_1 as positive or negative $\leadsto$ one separating curve
- ▶ Now add **a second data point** x_2 , **a third point** x_3 , ...
- ▶ Each **region** corresponds to one particular labeling of the data points.
- ▶ The number of regions tells you how many different labelings are possible.
- ▶ So we need to be clever to count regions.



Discussion of the VC bound for NN

- ▶ Depends heavily on the dimension.
 - ▶ Modern networks are very large, in such scenarios the generalization bounds on the VC dimension are vacuous and cannot explain why NNs can learn.
- ~→ See section on learning in the over-parametrized regime later.

Rademacher complexity

Rademacher complexity

The shattering coefficient or the VC dim are purely combinatorial objects, they do 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

$$\text{Rad}_n(\mathcal{F}) := E \sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \sigma_i f(X_i)$$

where the expectation is both over the draw of the random points $X_1, \dots, X_n$ and the random labels $\sigma_1, \dots, \sigma_n$.

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. For example this one:

Theorem 30 (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.

Computing Rademacher complexities for function classes is in many cases much simpler than computing shattering coefficients or VC dimensions.

Generalization bounds: summary

- ▶ Generalization bounds are a tool to answer the question whether a learning algorithm is consistent within $\mathcal{F}$.
- ▶ They give interesting insights, but are not practical in concrete numeric examples.
- ▶ Concerns 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.

- ▶ Generalization bounds are worst case bounds: worst case over all possible probability distributions, and worst case over all learning algorithms that pick a function from $\mathcal{F}$.

Generalization bounds for regression

- ▶ We only considered generalization bounds for the case of classification. The technique can be extended to regression as well.
- ▶ For example, instead of the shattering dimension we consider the fat-shattering dimension.
- ▶ Or we adapt the Rademacher complexity to deal with variables σ_i that are drawn from a normal distribution.
- ▶

However, for regression these techniques have never been super-popular. I guess the reason is that for regression, there are many other techniques from calculus (continuity, differentiability, Lipschitz,) that help to analyze what is going on, and that achieve better results.

Limitations and outlook

Limitations of the SLT and ERM principle

DISCUSSION: WHAT IS GOOD ABOUT THE RESULTS THAT WE HAVE PROVED SO FAR? WHAT IS BAD?

Limitations of the SLT and ERM principle (2)

SLT:

- ▶ The standard approach of SLT is to prove generalization by bounding the capacity of the underlying function class. This approach has revealed many important principles that can help algorithms to learn.
- ▶ In particular, applies primarily to machine learning in the under-parameterized regime, where we can control the uniform convergence.

Disadvantages:

- ▶ This approach is often technically challenging (how to compute the VC dim or the Rademacher complexity of the function class for a particular learning algorithm?)
- ▶ It sometimes leads to unsatisfactory results because it gives worst case guarantees over all functions in the class, rather than looking at the actual algorithms.
- ▶ In particular, in the SLT approach it does not matter which ERM minimizer an algorithm selects (in case there are several ones): the generalization bounds will apply to all of them or none.

SLT in the overparameterized regime?

- In the over-parameterized regime, the bounds are still valid, but vacuous. As an example consider the case of linear classification (details on the next slides). If $d \gg n$, then the generalization bound simply says that

$$R(f) - R_n(f) < \text{something large}$$

This is correct, but useless.

Different minima with different properties

We now consider an example where there exist many different empirical risk minimizers, some of which generalize well, and some of which don't.

- ▶ Data space $\mathbb{R}^d$, input points drawn according to $N(0, I_d)$ normal distribution
- ▶ True output: $f(x) = f((x^{(1)}, \dots, x^{(d)})) = x^{(1)}$ (first coordinate)
- ▶ Written as a linear model, the true model is $y = \langle w^*, x \rangle$ with $w^* = (1, 0, \dots, 0)$.
- ▶ Now we draw n training points (X_i, Y_i) and try to minimize the squared loss.
- ▶ **Consider the $d \gg n$ scenario.**
- ▶ Then the linear system $Y = Xw$ is under-determined, and there are many solutions:
 - ▶ The true solution $w^* = e_1$; it also has 0 population error.
 - ▶ Any vector of the form $w = e_1 + v$ where $v \in \ker(X)$ has also 0 training error. The test error of such a vector, say in the least squares loss, is

$$E(\|\langle w^*, x \rangle - \langle w, x \rangle\|^2) = E(\langle v, x \rangle^2) = E(v^T x x^T v) = v^T E(x x^T) v = v^T I_d v = \|v\|^2$$

Because v can be any vector of any length, this error can be as large as we want ...

Different minima with different properties (2)

What this example shows:

- ▶ There are many empirical risk minimizers.
- ▶ Looking at the solutions, we find that some of them are good (the true solution has 0 test error) and some of them are bad (large test error).
- ▶ For later: observe that the good solution in this case has the minimum norm among all solutions.
- ▶ Note that VC theory does not yield useful results in this scenario: if $d \gg n$, the generalization bound is large. So we cannot use VC theory to conclude that all of these risk minimizers have good generalization (which also is not true, as we have seen).
- ▶ The problem is that the VC bounds try to make a worst case statement over all possible solutions...

Outlook

In the over-parameterized regime, we will need to find a mechanism that helps us selecting a good solution among the many possible minimizers! Some fundamental principles are going to help us:

- ▶ Stability: stable optima tend to generalize better.
- ▶ Explicit regularization: we can reduce the capacity of the solution space by regularization. In some cases, we can then apply standard generalization bounds again.
- ▶ Implicit regularization: We will see that some optimization algorithms such as SGD perform implicit regularization, for example because their solution converges to the minimum norm solution. Under favorable assumptions (benign overfitting) one can then prove that such solutions generalize.

Stability

Literature:

- ▶ Section 6.4 in the Hardt/Recht book
- ▶ Section 11.2. in the Mohri book
- ▶ Section 13.2. in the Ben-David/Shalev-Shwartz book

Some key original references:

- ▶ The standard generalization bound in terms of uniform stability: Bousquet / Elisseeff: Stability and generalization, JMLR, 2002
- ▶ Stability is sufficient and necessary for learning: Shalev-Shwartz, Shamir, Srebro, Sridharan, Learnability, Stability and Uniform Convergence, JMLR 2010.
- ▶ Stability of SGD: Hardt, Recht, Singer. Train faster, generalize better: Stability of stochastic gradient descent, ICML 2016

Motivation and key definitions

Motivation 1

The standard approach of SLT is to prove generalization by bounding the capacity of the underlying function class. This approach is often technically challenging (how to compute the VC dim or the Rademacher complexity of the function class for a particular learning algorithm?) and often leads to unsatisfactory results (because it gives worst case guarantees over all functions in the class, rather than looking at the actual optimizers).

In the perceptron bound we had already seen a first glimpse of the **stability principle**: It says that if the result of a learning algorithm only changes by a small amount if we fiddle with the data by a small amount, then the algorithm tends to generalize well.

This section explores this principle in more depth.

Motivation 1 (2)

Second motivation:

Consider the LLN or the CLT in standard probability. It tends to hold as long as we can guarantee by assumptions that there is not a single data point that has much more influence than the others. For example in the LLN, the “influence” of one particular point X_i on the empirical average

$$\frac{1}{n} \sum_{i=1}^n X_i$$

should be “on the order of $1/n$ ”. This is typically ensured by the conditions, for example bounding the variance of all random variables.

Stability is a tool that in a similar spirit measures the amount of influence of single data points on the empirical risk.

Definitions

In the following we consider two samples drawn i.i.d. from the underlying distribution:

$$S_n := \{(X_1, Y_1), \dots, (X_n, Y_n)\}$$

$$S'_n := \{(X'_1, Y'_1), \dots, (X'_n, Y'_n)\}$$

We then consider an adapted sample S_n^i where we replaced the i -th sample in S_n by the corresponding point (X'_i, Y'_i) from S'_n , that is:

$$S_n^i := \{(X_1, Y_1), \dots, (X_{i-1}, Y_{i-1}), (X'_i, Y'_i), (X_{i+1}, Y_{i+1}), \dots, (X_n, Y_n)\}$$

We introduce the shorthand notation $Z_i = (X_i, Y_i)$ and slightly abuse notation to write $\ell(f, Z)$ instead of $\ell(f(X_i), Y_i)$.

Denote by $A(S_n)$ (and also by $f(S_n)$) the function that is learned by an algorithm A on training sample S_n .

Definitions (2)

In the following we will denote the empirical risk of a function f as measured on sample S_n as $R_{S_n}(f)$ (rather than just $R_n(f)$ as before; we want to be explicit on which sample the empirical error is computed):

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

Analogously for $R_{S'_n}$ and $R_{S_n^j}$.

Definitions (3)

We denote the **average stability** of learning algorithm A by

$$\Delta_{av}(A, n) := E_{S_n, S'_n} \left(\frac{1}{n} \sum_{i=1}^n \ell(f(S_n), Z'_i) - \ell(f(S_n^i), Z'_i) \right)$$

Observe what happens in the sum:

- For a particular index i we compare two loss terms. The functions that we compare are $f(S_n)$ trained on the original sample, and $f(S_n^i)$ trained on the sample where we replaced point Z_i by Z'_i . In both terms, we evaluate the loss on the same point Z'_i .
- Observe that Z'_i is a test point from the perspective of S_n , but a training point from the perspective of S_n^i .

Definitions (4)

For the next definition, we consider again the two samples S_n and S_n^i that differ in exactly one sample point, and a completely new test point Z . We define the **uniform stability** of learning algorithm A by

$$\Delta_{\text{sup}}(A, n) := \sup_{i=1, \dots, n} \sup_{S_n, S_n^i} \sup_Z |\ell(f(S_n), Z) - \ell(f(S_n^i), Z)|$$

In this definition, the point Z is independent of both the samples S_n and S_n^i .

Obviously we always have $\Delta_{\text{av}}(A, n) \leq \Delta_{\text{sup}}(A, n)$

Algorithmic stability implies generalization

Literature: Mohri textbook

Expected generalization gap

Proposition 31 (Expected generalization gap equals average stability)

$$\mathbb{E}(\underbrace{R(f(S_n))}_{\text{true error}} - \underbrace{R_{S_n}(f(S_n))}_{\text{empirical error}}) = \Delta_{\text{av}}(A, n)$$

expectation wrt training sample S_n

Proof:

$$\begin{aligned} & \mathbb{E}(R(f(S_n)) - R_{S_n}(f(S_n))) \\ &= \mathbb{E}(R(f(S_n))) - \mathbb{E}(R_{S_n}(f(S_n))) = \quad (\text{see next page}) \\ &= \mathbb{E}\left(\frac{1}{n} \sum_{i=1}^n \ell(f(S_n), z'_i) - \ell(f(S_n^i), z'_i)\right) \\ &= \Delta_{\text{av}}(A, n) \end{aligned}$$

Expected generalization gap (2)

$$\mathbb{E}(R(f(S_n))) = \rightsquigarrow \text{we just blow up by a sum of equal terms}$$

$$= \mathbb{E} \left(\frac{1}{n} \sum_{i=1}^n R(f(S_n)) \right) = \rightsquigarrow \text{write out } R, \text{ using independent test points}$$

$$= \mathbb{E} \left(\frac{1}{n} \sum_{i=1}^n \ell(f(S_n), z'_i) \right)$$

$$\mathbb{E}(R_{S_n}(f(S_n))) = \rightsquigarrow \text{use definition of empirical risk, observe that } z_i \text{ is training point}$$

$$= \mathbb{E} \left(\frac{1}{n} \sum_{i=1}^n \ell(f(S_n), z_i) \right) =$$

(in the expectation replace S_n and z_i by S_n^i and z'_i)

$$= \mathbb{E} \left(\frac{1}{n} \sum_{i=1}^n \ell(f(S_n^i), z'_i) \right) \quad \text{☺}$$

Discussion

- ▶ The above proposition already points in an interesting direction (bounding generalization error by stability), but has limited practical use because it is difficult to evaluate average stability.
- ▶ It turns out that the stronger notion of uniform stability is both easier to bound in practice and leads to a nice generalization bound.
- ▶ To be able to prove it, we will make use of McDiarmid's concentration inequality.

Uniformly stable algorithms generalize

Theorem 32 (Generalization bound for stable algorithm)

Assume that the loss function l is bounded by a constant $M > 0$. Let $\beta_n := \Delta_{\text{sup}}(A, n)$ the uniform stability of algorithm A on n samples. Then with probability at least $1 - \delta$ over samples S_n we have:

$$R(f(S_n)) \leq R_{S_n}(f(S_n)) + \beta_n + \frac{(2n\beta_n + M)\sqrt{\log(1/\delta)}}{\sqrt{2n}}$$

Proof

We will just consider a proof sketch. A full, clean proof can be found in the Mobri-Textbook section 11.2(p.268). We will apply McDiarmid's concentration inequality to the function:

$$\varphi(S_n) = R(f(S_n)) - R_{S_n}(f(S_n))$$

► **Step 1:** To apply the McDiarmid theorem we need to bound $|\varphi(S_n) - \varphi(S_n^i)|$

$$\begin{aligned} |\varphi(S_n^i) - \varphi(S_n)| &= \quad (\leadsto \text{definition}) \\ &= \left| R(f(S_n^i)) - R_{S_n^i}(f(S_n^i)) - R(f(S_n)) + R_{S_n}(f(S_n)) \right| \\ &\quad (\leadsto \text{triangle inequality}) \\ &\leq \underbrace{|R(f(S_n^i)) - R(f(S_n))|}_{(1) \leq \beta_n} + \underbrace{|R_{S_n^i}(f(S_n^i)) - R_{S_n}(f(S_n))|}_{(2) \leq \beta_n + \frac{M}{n}} \end{aligned}$$

Proof (2)

$$\begin{aligned} |R(f(S_n^i)) - R(f(S_n))| &= (\leadsto \text{definition}) \\ &= \left| \mathbb{E}(\ell(f(S_n^i), z)) - \mathbb{E}(\ell(f(S_n), z)) \right| \\ &\quad (\leadsto \text{pull absolute value in the expectation}) \\ &\leq \mathbb{E} \left(\underbrace{\left| \ell(f(S_n^i), z) - \ell(f(S_n), z) \right|}_{\leq \beta_n \text{ (by definition of stability)}} \right) \end{aligned}$$

Proof (3)

$$|R_{S_n^i}(f(S_n^i)) - R_{S_n}(f(S_n))| \leq$$

($\leadsto$ write as sum, split in parts $j \neq i$ and $j = i$
pull absolute values inside)

$$\begin{aligned} &\leq \frac{1}{n} \sum_{\substack{j=1 \\ j \neq i}}^n \underbrace{\left| \ell(f(S_n^i), z_j) - \ell(f(S_n), z_j) \right|}_{\leq \beta_n} \\ &\quad + \frac{1}{n} \underbrace{\left| \ell(f(S_n^i), z'_i) - \ell(f(S_n^i), z_i) \right|}_{\leq M} \\ &\leq \frac{n-1}{n} \beta_n + \frac{M}{n} \\ &\leq \beta_n + \frac{M}{n} \end{aligned}$$

Proof (4)

So overall we get:

$$|\varphi(S_n^i) - \varphi(S_n)| \leq 2\beta_n + \frac{M}{n}$$

► **Step 2:** Plug it into McDiarmid, applied to function φ :

$$P(\varphi(S) - \mathbb{E}(\varphi(S)) > \varepsilon) \leq \underbrace{\exp\left(-\frac{2\varepsilon^2}{n(2\beta_n + \frac{M}{n})^2}\right)}_{=\delta}$$

Now we rearrange the bound in the usual way: set the right-hand-side to δ and solve for ε with probability at least $1 - \delta$,

$$\underbrace{\varphi(S)}_{R(f(S_n)) - R_{S_n}(f(S_n))} \leq \mathbb{E}(\varphi(S)) + (2n\beta_n + M) \sqrt{\frac{\log(1/\delta)}{2n}}$$

Proof (5)

this term is the expected stability gap of the previous proposition. So we know

$$E(\varphi(S)) = \Delta_{sup}(A, s) \leq \Delta_{sup}(A, n)$$

Discussion

- ▶ This theorem gives a powerful tool. As soon as we can bound the uniform stability $\Delta_{sup}(A, n)$ of a learning algorithm, we can plug it into the theorem.
- ▶ This gives generalization guarantees when $\Delta_{sup}(A, s) = \beta_n$ is small enough such that the bound in the previous theorem goes to 0.
- ▶ The key term in the theorem is

$$\beta_n + \frac{(2n\beta_n + M)}{\sqrt{n}} \sqrt{\log(1/\delta)}$$

it goes to 0 if

$$\beta_n \leadsto 0 \text{ as } n \rightarrow \infty$$

$$\frac{(n\beta_n)}{\sqrt{n}} \approx \sqrt{n}\beta_n \rightarrow 0 \text{ as } n \rightarrow \infty$$

So the uniform-stability β_n needs to be of order of $o\left(\frac{1}{\sqrt{n}}\right)$ to be useful.

Attention: Stability does not immediately imply consistency

Observe: the previous theorem only says that for the function $f(S_n)$ picked by the algorithm we know that $R(f(S_n)) \approx R_n(f(S_n))$.

We do not (yet) know whether this function is anywhere close to the best in $\mathcal{F}$, nor to the Bayes predictor!

Simple counterexample:

- ▶ consider a naive algorithm that always predicts 0
- ▶ it is perfectly stable
- ▶ it satisfies $R(f(S_n)) = R_n(f(S_n))$
- ▶ but it is not the best in $\mathcal{F}$...

So what we need for consistency is a separate condition about the (empirical) risk of the classifier.

Stability+approx. ERM $\Rightarrow$ consistency

Assume we run ERM on a huge function class $\mathcal{F}$, we consider f_n and $f_{\tilde{\mathcal{F}}}$ as before.

Then: If f_{ERM} is uniformly stable, we get consistency.

Main argument:

$$\begin{aligned}
 R(f_n) - R(f_{\tilde{\mathcal{F}}}) &= \underbrace{R(f_n) - R_n(f_n)}_{\text{(1) small if ERM is stable, by the previous theorem}} \\
 &\quad + \underbrace{R_n(f_n) - R_n(f_{\tilde{\mathcal{F}}})}_{\text{(2) } \leq O \text{ by def. of } f_n \text{ as minimizer}} \\
 &\quad + \underbrace{R_n(f_{\tilde{\mathcal{F}}}) - R(f_{\tilde{\mathcal{F}}})}_{\text{(3) small by the standard LLN application to deterministic } f_{\tilde{\mathcal{F}}}}
 \end{aligned}$$

ERM with strongly convex loss function is stable

Literature: Hardt/Recht: Patterns, predictions and actions, Chapter 6 (p.111), Theorem 4

Motivation

Have seen:

- ▶ Machine learning often learns by minimizing a loss function on training data $\rightarrow$ ERM
- ▶ If the function class is small, we can (perhaps) prove uniform convergence (classical SLT) and obtain consistency.
- ▶ If the function class is so large that SLT does not work, we might be able to use the principle of uniform stability.

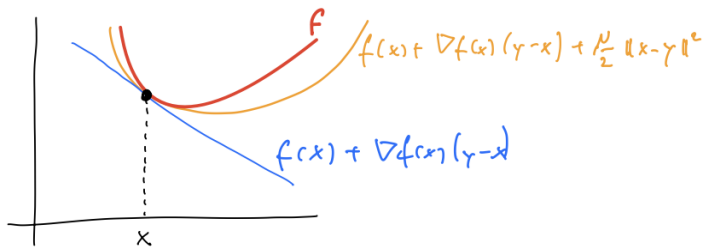
Question: **is ERM uniformly stable?** Always? Sometimes?

Answer will be: Only under very strong assumption on the loss function!

Recap: strongly convex functions

Consider a function $f : \mathcal{X} \rightarrow \mathbb{R}$, defined on a convex domain $\mathcal{X} \subset \mathbb{R}^d$. Assume that f is differentiable. Then the function is called **μ -strongly convex** (for $\mu > 0$) if for all $x, y \in \mathcal{X}$

$$f(y) \geq f(x) + \nabla f(x)(y - x) + \frac{\mu}{2} \|y - x\|^2$$



Recap: strongly convex functions (2)

To gain intuition, let us consider the 1-dim case, and assume that $f : \mathbb{R} \rightarrow \mathbb{R}$ is twice differentiable. Then:

- ▶ f is convex if $f''(x) \geq 0$
- ▶ and strongly convex if $f''(x) > \mu$ everywhere
intuition: f has a lower bound μ on the curvature

In $\mathbb{R}^d$, a convex loss is μ -strongly convex if the smallest eigenvalue λ of the Hessian satisfies $\lambda > \mu$.

Recap: strongly convex functions (3)

In particular, if we deal with loss functions that are strongly convex, then the empirical risk is also strongly convex (with the same parameter μ). For such loss functions, there always exists a global optimum and it is not “hidden” in flat parts:



GD and SGD would easily find it.

ERM is stable for strongly convex loss functions

Theorem 33 (ERM for strongly convex loss fcts is stable)

Consider a set of functions $\mathcal{F}$ in which each function is parameterized by a vector $w \in \mathbb{R}^d$. Assume the loss function $\ell(f_w(x), y)$ is μ -strongly convex in w , for every $(x, y) \in \mathcal{X} \times \mathcal{Y}$. Further assume that the loss function is L -Lipschitz in w , for every (x, y) . Then the empirical risk minimizer f_{ERM} is uniformly stable with

$$\Delta_{sup}(f_{ERM,n}) \leq \frac{4L^2}{\mu \cdot n}$$

First discussion

- ▶ This theorem makes a statement on the empirical risk minimizer itself (not about any algorithm that might attempt to find it).
- ▶ The complexity of the function class $\mathcal{F}$ does not show up anywhere, this is great.
- ▶ The stability Δ_{sup} scales as $1/n$, so the generalization bound above leads to consistency.
- ▶ The assumptions are really strong!
- ▶ The theorem is promising, but not yet a solution to situations with more complex loss functions.

Proof of the theorem

Fix sample

$$S = (x_1, y_1), \dots, (x_i, y_i), \dots, (x_n, y_n),$$

$$S' = (x_1, y_1), \dots, (x'_i, y'_i), \dots, (x_n, y_n)$$

and a test point (x, y) .

Let w_S be the ERM-minimizer on S and $w_{S'}$ be the one on S' .

We need to prove:

$$|\ell(w_S, z) - \ell(w_{S'}, z)| \leq \frac{4L^2}{\mu n}$$

Because ℓ is Lipschitz, we have:

$$|\ell(w_S, z) - \ell(w_{S'}, z)| \leq L \|w_S - w_{S'}\|$$

So it is sufficient to prove that:

$$\|w_S - w_{S'}\| \leq \frac{4L}{\mu n}$$

Proof of the theorem (2)

Step 1:

- ▶ Any function g that is strongly convex with minimizer w^* satisfies

$$\frac{\mu}{2} \|w - w^*\|^2 \leq g(w) - g(w^*)$$

(follows directly from the definition of μ -strongly convex and the fact that the gradient at w^* is 0.)

- ▶ If the loss function is μ -strongly convex, so is the empirical risk.
- ▶ Thus if we consider g as the empirical risk wrt S we get:

$$\frac{\mu}{2} \|w_S - w_{S'}\|^2 \leq \underbrace{R_S(w_S) - R_S(w_{S'})}$$

we now bound this term in Step 2:

Proof of the theorem (3)

Step 2:

$R_S(w_{S'}) - R_S(w_S)$ (write as finite sum over S , split off term i with argument z_i)

$$= \frac{1}{n} (\ell(w_{S'}, z_i) - \ell(w_S, z_i)) + \frac{1}{n} \sum_{i \neq j} (\ell(w_{S'}, z_j) - \ell(w_S, z_j)) =$$

(add and subtract blue terms with argument z'_i , then consider sum over S')

$$= \frac{1}{n} \left(\underbrace{\ell(w_{S'}, z_i) - \ell(w_S, z_i)}_{\text{apply Lipschitz: } \leq L \|w_{S'} - w_S\|} \right) + \frac{1}{n} \left(\underbrace{\ell(w_S, z'_i) - \ell(w_{S'}, z'_i)}_{\text{apply Lipschitz: } \leq L \|w_{S'} - w_S\|} \right)$$

$$+ \left(\underbrace{R_{S'}(w_{S'}) - R_{S'}(w_S)}_{\leq 0 \text{ because } w_{S'} \text{ minimizes the emp. risk } R_{S'}} \right)$$

$$\leq \frac{2L}{n} \|w_{S'} - w_S\|$$

Proof of the theorem (4)

Step 3: Put it together:

$$\frac{\mu}{2} \|w_S - w_{S'}\|^2 \leq R_S(w_S) - R_S(w_{S'}) \leq \frac{2L}{n} \|w_{S'} - w_S\|$$

$$\Rightarrow \|w_S - w_{S'}\| \leq \frac{4L}{\mu n}$$

$$\Rightarrow \Delta_{sup}(f_{ERM}, n) \leq \frac{4L}{\mu n}$$



Discussing the assumption

Consider a linear model with the squared loss function:

$$\ell(w, (x, y)) = (\langle w, x \rangle - y)^2$$

The gradient is $\nabla \ell(w) = 2w^t x$ and the Hessian is $\nabla^2 \ell = 2xx^t$. Because the Hessian just has rank 1, it is not strongly convex !!!

We can make it strongly convex if we add regularization, and this will also be the main use case for the current theorem.

(Stochastic) gradient descent on smooth loss function is stable

Literature:

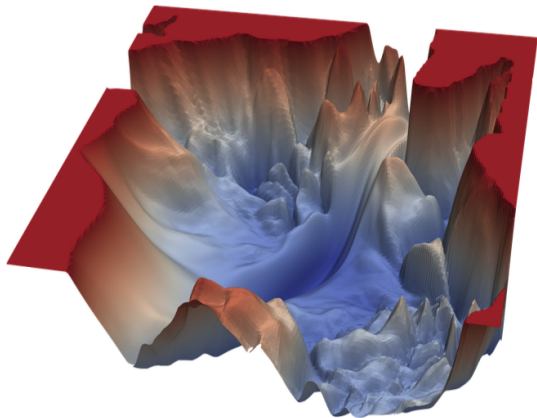
Text book: Hardt/Recht, Patterns, predictions and actions, Chapter 6, Theorem 6, p.120

Original paper: Hardt, Recht, Singer. Train faster, generalize better: Stability of stochastic gradient descent, ICML 2016

Motivation

- ▶ We have already proved that ERM is stable, but only under super-strong assumptions (strongly convex loss, a lower bound on the curvature of the loss).
- ▶ And the result was about the perfect ERM minimizer (but we ignored the question how to find it).
- ▶ Now we look at a specific algorithm and the stability of its solutions: SGD.

Intuition: is SGD always stable?



Depicted: the loss landscape of a neural network.

(Figure by Li et al, NeurIPS 2018, "Visualizing the loss landscape of neural nets")

Intuition: is SGD always stable? (2)

- ▶ Want uniform stability of order $o\left(\frac{1}{\sqrt{n}}\right)$.
- ▶ SGD won't be stable if the algorithm makes super-large steps that cannot be controlled.
- ▶ SGD won't be stable if the loss landscape is super-fragmented, so we cannot really control where we jump...
- ▶ So we will need assumptions!

Recap: β -smooth functions

A differentiable function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is β -smooth if its gradient is β -Lipschitz:

$$\|\nabla f(x) - \nabla f(y)\| \leq \beta \|x - y\|$$

Intuitively, smoothness is good for optimization with gradient methods because the gradient changes only slowly. So if we make a small step in one direction, the gradient at the new point will not point into a completely different direction.

While μ -strongly convexity provides a lower bound on the curvature, β -smoothness is more like an upper bound.

Example for β -smooth loss function with linear regression.

- The squared loss for linear regression:

$$\ell(w, (x, y)) = \frac{1}{2} (\langle x, w \rangle - y)^2$$

is convex and smooth in w with constant $\beta = \|x\|^2$. Hence, if $\|x\|$ is bounded by R , we have R -smoothness.

- For the logistic loss

$$\ell(w, (x, y)) = \log(1 + \exp(-yw^t x))$$

one can prove that the Hessian satisfies

$$\nabla^2 \ell \leq \text{const.} \cdot xx^t$$

so we again get smoothness if $\|x\|$ is bounded.

Recap: GD and SGD

Assume we want to minimize the empirical risk $g(w) := \frac{1}{n} \sum_{i=1}^n \ell(f_w(x_i), y_i)$

In gradient descent, we compute the full gradient and update the parameters:

$$\nabla g(w) = \frac{1}{n} \sum_{i=1}^n \nabla \ell(f_w(x_i), y_i)$$

$$w_{t+1} = w_t - \eta \nabla g(w_t)$$

In stochastic gradient descent, at each point we sample one training point i , just compute the gradient wrt this sample and update:

$$w_{t+1} = w_t - \eta \nabla \ell(f_{w_t}(x_i), y_i)$$

Theorem 34 (Stability of GD)

Consider a loss function that is continuously differentiable, convex, β -smooth and L -Lipschitz. Suppose that we run GD with step size $\eta_t < 2/\beta$ for T steps from a fixed starting configuration w_0 . The the resulting solution is uniformly stable with

$$\Delta_{sup}(GD) \leq \frac{2L^2}{n} \sum_{t=1}^T \eta_t$$

Proof sketch

Step1: Want to compare the two trajectories:

$$w_{t+1} = w_t - \eta_t \nabla R_S(w_t) \quad (\text{original sample})$$

$$w'_{t+1} = w'_t - \eta_t \nabla R_{S'}(w'_t) \quad (\text{adapted sample } S')$$

$$\|w_{t+1} - w'_{t+1}\| = w_t - \eta_t \nabla R_{\text{different datasets } S, S'}(w_t) - w'_t + \eta_t \nabla R_{S'}(w'_t)$$

(add and subtract $\nabla R_S(w'_t)$ and rearrange)

$$\leq \underbrace{\|w_t - w'_t - \eta_t \nabla R_{\text{same dataset } S}(w_t) - \eta_t \nabla R_S(w'_t)\|}_{\leq \|w_t - w'_t\| \text{ see step 3 below}} + \underbrace{\eta_t \|\nabla R_S(w'_t) - \nabla R_{S'}(w'_t)\|}_{\leq \eta_t \cdot \frac{2L}{n} \text{ see step 2 below}}$$

Proof sketch (2)

Step 2 Bound the blue term: If S and S' differ by one sample, then

$$\|\nabla R_S(w) - \nabla R_{S'}(w)\| \leq \frac{2L}{n}$$

To see this observe that the difference in gradient at one point is bounded by $2L$, and that only one of the summands changes (hence the $1/n$).

Step 3 Bound the green term: A gradient step of size $2/\beta$ is “non-expansive”: for any w, v we have:

$$\|(w - \eta \nabla R_S(w)) - (v - \eta \nabla R_S(v))\| \leq \|w - v\|$$

(follows from smoothness definition; this is where we need the assumption)

So in particular, the statement holds for $w = w_t$ and $v = w'_t$ (green term).

Proof sketch (3)

Step 4 Combine: from (1)-(3) we get:

$$\|w_{t+1} - w'_{t+1}\| \leq \|w_t - w'_t\| + \eta_t \frac{2L}{n}$$

Now we sum these equations for $t = 0$ to T , and observe that we start with the same w_0 :

$$\sum_{t=0}^{T-1} \|w_{t+1} - w'_{t+1}\| \leq \sum_{t=0}^{T-1} \left(\|w_t - w'_t\| + \eta_t \frac{2L}{n} \right)$$

Now nearly all terms occur on both sides and can be deleted, which leaves us with:

$$\|w_T - w'_T\| \leq \frac{2L}{n} \sum_{t=1}^T \eta_t$$

Proof sketch (4)

Step 5 Finally, to evaluate the stability we need to compare the losses of the functions w_T and w'_T , and we can use the Lipschitz condition:

$$\begin{aligned} |\ell(w_T, (x, y)) - \ell(w'_T, (x, y))| &\leq L \cdot \|w_T - w'_T\| \\ &\leq \frac{2L^2}{n} \sum_{t=1}^T \eta_t \end{aligned}$$

This is the stability result we have been looking for.



Stability of SGD, convex case

We get pretty much the same theorem for SGD:

Theorem 35 (Stability of SGD)

Consider a loss function that is continuously differentiable, that is convex, β -smooth, and L -Lipschitz. Suppose that we run **SGD** with step size $\eta_t < 2/\beta$ for T steps. Then the resulting solution is uniformly stable with

$$\Delta_{sup}(SGD) \leq \frac{2L^2}{n} \sum_{t=1}^T \eta_t$$

Proof sketch

The proof goes along similar lines, but due to randomness it is more complicated:

- ▶ In GD, we introduce a small bias at every step.
- ▶ In SGD, we only seldom have a bias (if we update the one changed sample), but then a larger change happens.

We skip the details, refer to the Hardt, Singer, Recht paper for details (and the assignments).

General remarks:

- ▶ While the above result bounds the stability, we cannot yet guarantee consistency (how many steps T ? Convergence to ERM-minimizer?)
- ▶ Hardt/Recht/Singer do provide some guarantees for a convex setting.
- ▶ We skip it here... look in the paper if you are interested.

Stability of SGD, non-convex case

The stability result can even be extended to non-convex loss functions, as long as the loss is still Lipschitz and β -smooth:

Theorem 36 (Stability of SGD, non-convex case)

Assume the loss is L -Lipschitz, β -smooth. Suppose we run SGD for T steps with decreasing step sizes $q_t \leq c/t$. Then SGD is uniformly stable with

$$\Delta_{sup} \leq \frac{1 + 1/(\beta c)}{n - 1} (2cL^2)^{1/(\beta c + 1)} T^{\beta c/(\beta c + 1)}$$

Ignoring constants, we get

$$\Delta_{sup} \approx \frac{T^{1-1/(\beta c + 1)}}{n}$$

Stability of SGD, non-convex case (2)

Depending on T , we can combine this result with the standard stability bound to get generalization (but not necessarily consistency).

Discussion of the theorem

- ▶ The theorems about stability of GD / SGD are the first results about an actual learning algorithm (and not just an abstract result about some empirical risk minimizer).
- ▶ It does bound the stability, but depending on the number T of steps, we might or might not get stability in the order $\mathcal{O}(1/\sqrt{n})$ (as it would be needed in the generalization bound).
- ▶ From this point of view of stability, fewer steps are better, leading to a smaller generalization gap — but this ignores whether this brings us close enough to the actual minimizer!
- ▶ We will revisit this issue when we talk about regularization.

More results

Stability has been proved for a wide range of algorithms:

- ▶ Regularization, see later
- ▶ Many kernel algorithms
- ▶ GD / SGD in various scenarios (also non-convex ones):
 - ▶ Uniform stability of SGD for deep networks: Moritz Hardt, Benjamin Recht, and Yoram Singer, “Train faster, generalize better: Stability of stochastic gradient descent,” ICML 2016.
 - ▶ Stability under noisy SGD: Ankit Pensia, Varun Jog, and Po-Ling Loh, “Stability of stochastic gradient descent on non-convex losses,” arXiv 2018.
 - ▶ Stability and early stopping: Moritz Hardt, Tengyu Ma, and Benjamin Recht, “Identity matters in deep learning,” ICLR 2020.
 - ▶ Stability of sharp vs. flat minima: Lei Wu, Zhanxing Zhu, et al., “Understanding SGD generalization in deep learning via stability,” NeurIPS 2020.

Explicit regularization

Literature:

- Understanding machine learning, sec 13

Principle of regularization

Principle of regularization

- ▶ Consider a training set of n points in our standard setup.
- ▶ Let $\mathcal{F}$ be a very large space of functions, say all real-valued differentiable functions.
- ▶ Fix a loss function ℓ and denote by R_n and R the corresponding empirical and true risks.
- ▶ 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**

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

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

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

Why might it make sense?

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.

SLT suggests to control the size of the function class. But in practice, it is difficult to do this explicitly. Regularization now does it implicitly. Note that typically, spaces of simple functions have smaller capacity than spaces of complex functions.

Why might it make sense? (2)

We will see later that regularization also might have other nice side effects:

- ▶ It might make the solution unique
- ▶ It might improve the optimization problem (easier to solve)
- ▶ It might turn a learning algorithm into a stable one that has better generalization guarantees

What can we choose as regularizer?

Regularizers are often a good tool to enforce properties that we like and that are not so easy to control otherwise, for example sparsity or interpretability.

To achieve this, we need a function Ω that takes the parameters of our function as input and can act as a regularizer:

- ▶ It returns a real number that measures what we are interested in (sparsity, interpretability, ...)
- ▶ The smaller the better (because we minimize).
- ▶ To make our life simpler, the function should be differentiable, so we can solve the optimization problem by gradient methods ...

What are the tradeoffs?

What happens if λ is very small? Very larger?

Relation to overfitting / underfitting, approximation / estimation error?

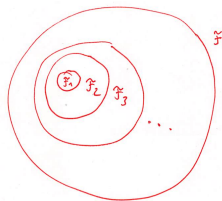
Relationship to ERM?

What are the tradeoffs? (2)

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



But we changed the objective!?!

Yes, regularization does change the objective that we optimize during learning!!!

SO WHAT ABOUT CONSISTENCY?

But we changed the objective!?! (2)

We will see later that consistency can still be achieved, here is the idea how:

- ▶ We choose a huge function class $\mathcal{F}$ that definitely contains the Bayes classifier.
- ▶ While the number of sample points is small, we use a large regularization constant λ_n (we don't want to overfit with a complex function).
- ▶ As the sample size increases, we slowly decrease the regularization constant λ_n because we might need (and can afford) more complex functions to fit the data. We still prevent overfitting.
- ▶ As $n \rightarrow \infty$, we let $\lambda_n \rightarrow 0$ (with a rate that needs to be determined). This means that in the infinite sample limit, we essentially do not regularize any more and can pick the best function in $\mathcal{F}$. This is then the Bayes classifier.
- ▶ To prove consistency of the learning rule, we need to prove that the framework outlined above really works, so that f_n converges to f^* .

Recap: Linear least squares regression

Literature:

- ▶ Understanding machine learning sec 9
- ▶ Bach book, 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* of the form

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

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

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

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

Linear setup (2)

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:

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

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:

$$X = \left(\begin{array}{ccc} x_{11} & \dots & x_{1d} \\ \vdots & & \\ x_{n1} & \dots & x_{nd} \end{array} \right) \Bigg\}^n \quad Y = \left(\begin{array}{c} y_1 \\ \vdots \\ y_n \end{array} \right) \Bigg\}^n \quad w = \left(\begin{array}{c} w_1 \\ \vdots \\ w_d \end{array} \right) \Bigg\}^d$$

- 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)

To get rid of the parameter b :

- Define the matrices

$$\tilde{X} = \begin{pmatrix} x_{11} & \dots & x_{1d} & 1 \\ \vdots & & \vdots & \vdots \\ x_{n1} & \dots & x_{nd} & 1 \end{pmatrix} \quad \left. \begin{array}{c} \text{d+1} \\ \end{array} \right\} n$$

$$\tilde{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_d \\ b \end{pmatrix} \quad \left. \begin{array}{c} \end{array} \right\} d+1$$

- Then we have

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

Concise notation (3)

- 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

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

Least squares regression

Without loss of generality, from now on we consider the simplified problem that does not involve the intercept b (and we remove the twiddles on the letters $\tilde{X}$ and $\tilde{w}$ to keep notation simple):

$$\min_{w \in \mathbb{R}^d} \|Y - Xw\|^2$$

n denotes the number of data points, d the dimension of the space, the matrix X has the sizes $n \times d$, Y and w are vectors of length n and d , respectively.

Proposition 37 (Least squares is convex)

The least squares optimization problem is a convex optimization problem.

Proof. Exercise

Solution, case of full rank

Theorem 38 (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: We write all equations in matrix form:

- ▶ Objective function: $\text{Obj} : \mathbb{R}^d \rightarrow \mathbb{R}$, $\text{Obj}(w) := \frac{1}{2} \|Y - Xw\|^2$
- ▶ Derivative: $\nabla(\text{Obj})(w) = -X^t(Y - Xw)$.
- ▶ Setting the gradient to zero gives the necessary condition: $X^t Y = (X^t X)w$. Ideally, we would like to solve this for w .
- ▶ We always have $\text{rank}(X) = \text{rank}(X^t X) = \text{rank}(X X^t)$. In particular, under the assumption that X has rank d , the matrix $X^t X$ is of full rank, hence invertible.
- ▶ So we can solve for w by $w = (X^t X)^{-1} X^t Y$.

Solution, case of full rank (2)

- ▶ 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.



Recap: Generalized inverse of a matrix

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

$$A^{-1} = \sum_{i=1}^d \frac{1}{\lambda_i} v_i v_i^t$$

Recap: Generalized inverse of a matrix (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).

- 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 $A^+AA^+ = A^+$. In particular, if A is invertible, then $A^{-1} = A^+$.

Solution, general case

Theorem 39 (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!)
- ▶ So $w = (X^t X)^+ X^t Y$ is one solution to the problem, which proves part 1 of the theorem.

Solution, general case (2)

- ▶ 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.
 - ▶ So our problem has many solution vectors. Note that all of them lead to the same objective value.
- ▶ Observe: all solutions give the same results on the training points:
 - ▶ 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.
- 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”.)

Ridge regression / Tikhonov regularization: least squares with L_2 -regularization

Literature:

- ▶ Bach book, sec 3.6
- ▶ Understanding machine learning sec 13.1
- ▶ Hastie/Tibshirani/Friedman Section 3.4.3;

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:

- ▶ Linear regression setup as in the last section (data $X_i \in \mathbb{R}^d$, $Y_i \in \mathbb{R}$, linear regression, squared loss)
- ▶ As regularizer use $\Omega(f) := \|w\|^2 = \sum_{i=1}^d w_i^2$. Choose a regularization constant $\lambda > 0$.
- ▶ Then solve the problem

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

Solution of ridge regression

Theorem 40 (Solution of Ridge Regression)

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

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

where I_d is the $d \times d$ identity matrix.

Solution of ridge regression (2)

Proof.

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

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

- ▶ It is straight forward to see that the matrix $(X^t X + n\lambda I_d)$ 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 of ridge regression (3)

Proof that $(X^tX + n\lambda I_d)$ is invertible:

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

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

- ▶ All eigenvalues $\sigma + \lambda$ of $A + \lambda I$ are strictly larger than 0 because $\sigma \geq 0$ and $\lambda > 0$.
- ▶ So $A + \lambda I$ has full rank and is invertible.



Choice of the parameter λ

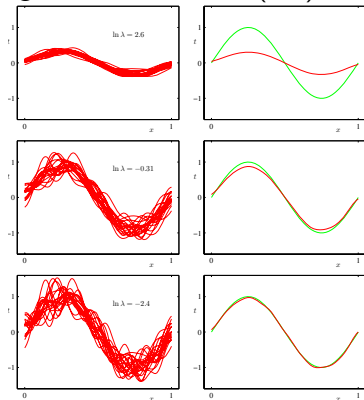
QUESTION: WHAT IS THE ROLE OF λ ? WHAT HAPPENS TO ESTIMATION AND APPROXIMATION ERROR IF IT IS HIGH / LOW?

Choice of the parameter λ (2)

Example (from Bishop's book):

Left: results for decreasing amount of regularization

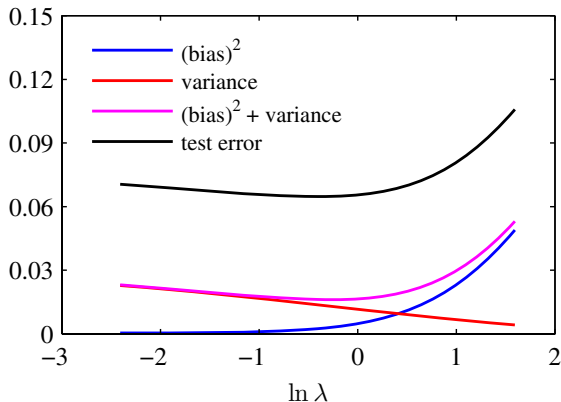
Right: True curve (green), average estimated curve (red)



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 $X \in \mathbb{R}^{n \times d}$:

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

- 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

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

- Regularized case:

(*) Ridge regression as shrinkage method (2)

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

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

- Case σ_j small: here it makes a lot of difference whether we have σ_j^2 or $\sigma_j^2 + \lambda$ 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: $\hat{\theta}_{LS} = X$.
- ▶ Now consider the following “shrinkage estimator” (it is called the James-Stein estimator): $\hat{\theta}_{JS} = \left(1 - \frac{d-2}{\|X\|^2}\right)X$.

(*) Ridge regression as shrinkage method (4)

- One can prove that it outperforms the standard least squares error in terms of expected least squares error:

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

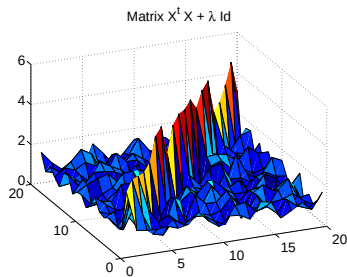
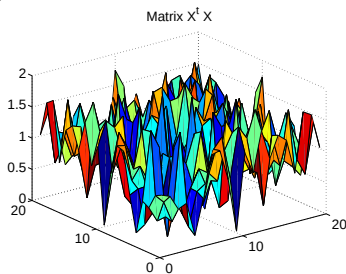
Read it on wikipedia if you are interested ☺

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 $X^t X$ in the least squares solution by the matrix $X^t X + \lambda Id$.
 - 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.

Tikhonov regularization leads to stability

Literature:

- ▶ Understanding machine learning, Sec 13.3
- ▶ Hardt book 113

Tikhonov regularization leads to strong convexity

- ▶ Consider any convex loss function ℓ (for example, the least squares loss when we restrict the parameter space to a bounded domain).
- ▶ While convexity is already nice, it does not yet guarantee any particular rate of convergence for the optimizer, nor does it suffice to guarantee stability.
- ▶ To achieve these kind of properties, we need strongly convex loss function!
- ▶ Now: adding the Tikhonov regularizer $\lambda\|w\|^2$ turns standard least squares into a strongly convex problem:

Proposition 41

- ▶ The function $f(w) = \lambda\|w\|^2$ is 2λ -strongly convex.
- ▶ If g is convex and f is λ -strongly convex, then $f + g$ is λ -strongly convex.

Proof: exercise.

Tikhonov regularization leads to stability

We can now invoke the previous theorems on stable learning algorithm.

- ▶ We have proved in the last chapter that ERM with a μ -strongly convex Lipschitz loss is uniformly stable with $\Delta_{\text{sup}} = \frac{4L^2}{\mu n}$
- ▶ The previous proposition says that if we combine a convex loss function with the Tikhonov regularizer, we get a μ -strongly convex function with $\mu = 2\lambda$
- ▶ This leads to stability $\Delta_{\text{sup}} = \frac{2L^2}{\lambda n}$
- ▶ We can for example plug this into the bound on the expected generalization gap to obtain that

$$E\left(R_n(f_n) - R(f_n)\right) \leq \Delta_{\text{sup}} = \frac{2L^2}{\lambda n}$$

Choice of λ

While the previous theorem holds for any fixed value of λ , there is a tradeoff:

- ▶ The stability term Δ_{sup} and thus the generalization gap get smaller if λ increases.
- ▶ But the empirical risk $R_n(f_n)$ gets larger if λ increases because the objective is not so much focused on achieving a small empirical risk.

So which value should we chose for λ ? Not obvious! In practice, we often use cross validation. In theory, we can get some rules of thumb as well, see later.

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 again.
- ▶ It is very desirable to obtain a solution for which many of the coefficients w_i are zero. Such a solution is called “sparse”.
- ▶ Reasons:
 - ▶ Stability of the solution
 - ▶ Computational reasons: even if we have many basis functions, we just need to evaluate few of them.
 - ▶ Interpretability of the solution

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

A naive regularizer for sparsity

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 (2)

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

Excursion: p -norms

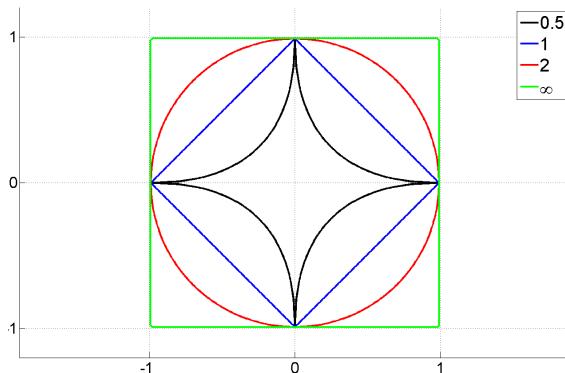
- ▶ For $p > 0$, define for a vector $w \in \mathbb{R}^d$

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

- ▶ 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 $\|w\|_2 = 1$).



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

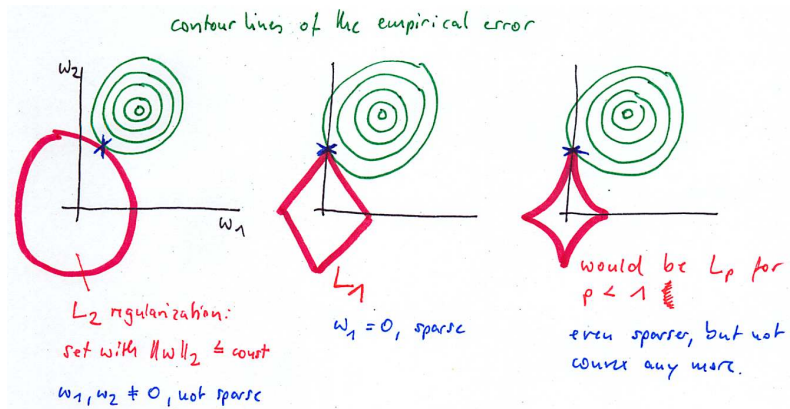
We now want to settle for $\|w\|_1$ as a regularizer: It is “as close” to the non-convex regularizer $\|w\|_0$ as possible while still being convex.

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.



Sparsity and the L_1 -norm (3)

Here is an intuitive argument why solutions with L_2 -regularization tend not to be sparse:

- ▶ 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 . So L_2 -regularization might rather prevent sparsity.
- ▶ 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. So this gives at least some hope that L_1 -regularization is more sparse than L_2 .

The Lasso

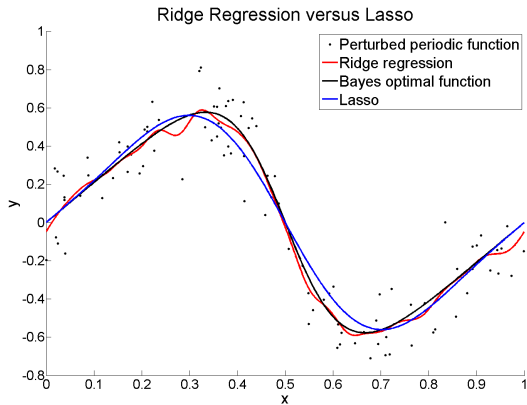
Consider the following regularization problem:

- ▶ Input space $\mathcal{X} \subset \mathbb{R}^d$, output space $\mathcal{Y} = \mathbb{R}$ (regression)
- ▶ Training data $X \in \mathbb{R}^{n \times d}$, $Y \in \mathbb{R}^n$
- ▶ Linear function space $\mathcal{F}$, least squares loss
- ▶ As regularizer use $\Omega(f) := \|w\|_1 = \sum_{i=1}^d |w_i|$. Choose a regularization constant $\lambda > 0$.
- ▶ Then solve the problem

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

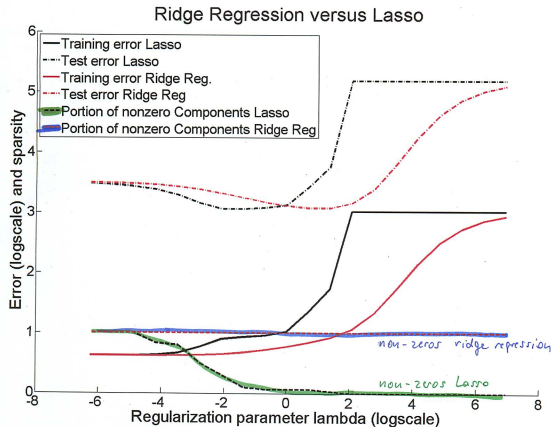
- ▶ The Lasso objective function is convex (it is a sum of two convex functions).
- ▶ However, there does not exist a closed form solution.
- ▶ Hence it has to be solved by a standard algorithm for convex optimization.

Example



(Figure by Matthias Hein)

Example (2)



(Figure by Matthias Hein)

- ▶ 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)

Theoretical results regarding the different types of regularization

Literature: Bach book Sec 3 (least squares without regularization, ridge regression), Sec 8 (Lasso)

Model assumptions: linear model + noise

In the following we are going to present a number of generalization bounds for linear regression (without proofs; see the Bach book for proofs). To make the analysis simpler, we make strong assumptions:

- **Linear model:** we assume that the data has been generated by a linear model: The data points are $X_1, \dots, X_n \in \mathbb{R}^d$. There exists a vector $\theta^* \in \mathbb{R}^d$ such that the relationship between input X_i and output Y_i is linear plus noise:

$$Y_i = \langle X_i, \theta^* \rangle + \varepsilon_i$$

The noise variables $\varepsilon_i \in \mathbb{R}$ are independent with expectation $E(\varepsilon_i) = 0$ and variance $E(\varepsilon_i^2) = \sigma^2$.

By $X \in \mathbb{R}^{n \times d}$ we denote the matrix containing the training points X_i as row; in statistics, this matrix is called the design matrix. By $Y \in \mathbb{R}^d$ we denote the vector with the training outputs. By Σ and $\hat{\Sigma} = X^T X / n$ we denote the true and the sample covariance matrices, respectively.

Model assumptions: linear model + noise (2)

- We consider the **least squares loss function**, different regularizers $\Omega(\theta)$, and minimize the following problem:

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \|Y - X\theta\|_2^2 + \lambda \Omega(\|\theta\|)$$

- **Fixed vs random design setting:** In statistics, the **fixed design setting** assumes that the data points $X_1, \dots, X_n$ are fixed, and we are interested in the loss on these data points. This setting is closely related to studying the training error in machine learning.

This is in contrast to the **random design setting**, where we assume our data are random and we are interested in the test error.

Fixed design risk

Consider a **fixed parameter vector** θ . For the fixed design risk of this model we consider the data points $X_1, \dots, X_n$ as fixed, and $Y_1, \dots, Y_n$ as random quantities drawn from the underlying distribution. The fixed design risk is then denoted as

$$\mathcal{R}_n(\theta) = E_{Y_1, \dots, Y_n} \left(\frac{1}{n} \sum_{i=1}^n (Y_i - \langle X_i, \theta \rangle)^2 \right) = E_Y \left(\frac{1}{n} \|Y - X\theta\|^2 \right)$$

It is similar in spirit to the training error, but we additionally take the expectation over the labels Y_i . If θ is a deterministic vector, then $R_n(\theta)$ is deterministic as well.

Fixed design risk (2)

If we consider a **model** $\hat{\theta} := \hat{\theta}_{(X_i, Y_i)_{i=1, \dots, n}}$ that has been estimated from training data $(X_i, Y_i)_{i=1, \dots, n}$, then $\mathcal{R}_n(\hat{\theta})$ is a random quantity, and we are interested in $E(\mathcal{R}_n(\hat{\theta}))$.

In the fixed design setting, this expression involves two expectations: the outer expectation, which takes care of the randomness in $\hat{\theta}$ (due to the random training outputs Y_i ; the training inputs X_i are treated fixed in random design); and the inner expectation that evaluates against independently drawn labels $\tilde{Y}_i$:

$$E(\mathcal{R}_n(\hat{\theta})) = E_{Y_1, \dots, Y_n} E_{\tilde{Y}_1, \dots, \tilde{Y}_n} (\tilde{Y}_i - X \hat{\theta}_{(X_i, Y_i)_{i=1, \dots, n}})^2$$

The **fixed design Bayes risk** $\mathcal{R}_n^*$ is defined as the minimizer of $\mathcal{R}_n(\theta)$ (so it also depends on the design matrix).

Random design risk

The true risk in the **random design setting** is denoted as $\mathcal{R}(\theta)$. Here we additionally take the expectation over the input points X_i :

$$\mathcal{R}(\theta) = E_{(X,Y)}(Y - \langle X, \theta \rangle)^2$$

The Bayes risk is denoted as $\mathcal{R}^*$, it denotes the true risk of the perfect classifier θ^* .

Bounds for ordinary least square regression (fixed design)

Theorem 42 (Excess risk, fixed design, no regularization)

Under the linear model and fixed design assumptions, for any $\theta \in \mathbb{R}^d$, we have $\mathcal{R}^* = \sigma^2$ and

$$\mathcal{R}_n(\theta) - \mathcal{R}_n^* = \|\theta - \theta^*\|_{\hat{\Sigma}}^2,$$

where $\hat{\Sigma} = \frac{1}{n}X^\top X$ is the input covariance matrix and $\|\theta\|_{\hat{\Sigma}}^2 = \theta^\top \hat{\Sigma} \theta$.

If $\hat{\theta}$ is the estimate constructed by OLS, then the expected empirical risk satisfies

$$\mathbb{E}(\mathcal{R}_n(\hat{\theta})) - \mathcal{R}_n^* = \frac{\sigma^2 d}{n}$$

where the expectation is over the randomness in the construction of $\hat{\theta}$ due to the labels Y_i (the points X_i stay fixed, no expectation over them!)

Bounds for ordinary least square regression (fixed design) (2)

Observe that the bound grows linearly in d , so it is only going to be small in the setting $d \ll n$.

In a bias-variance decomposition, we see that the **bias** in this case is 0, and the **variance** is the term given in the theorem.

Bounds for ordinary least square regression (random design)

Surprisingly, the bounds for the random design are similar in spirit to the fixed design:

Proposition 43 (Excess risk, random design, no regularization)

For any fixed θ , the true excess risk is given as follows:

$$\mathcal{R}(\theta) - \mathcal{R}_* = \|\theta - \theta_*\|_{\Sigma}^2$$

(compared to the fixed design, we just replace $\hat{\Sigma}$ with Σ).

If $\hat{\theta}$ is the estimate constructed by OLS, and if the empirical covariance matrix $\hat{\Sigma}$ is invertible, then the expected true excess risk is given by

$$E(\mathcal{R}(\hat{\theta})) - \mathcal{R}_* = \frac{\sigma^2}{n} \cdot E(\text{tr}(\Sigma \hat{\Sigma}^{-1}))$$

where tr denotes the trace of a matrix.

Bounds for ordinary least square regression (random design) (2)

Just as a gut feeling, if $\hat{\Sigma}$ were identical to Σ , then this term would result in $\text{tr}(I_d) = d$, so we recover the fixed design bound.

Bound for least squares with L_2 regularization (fixed design)

Consider a solution of the ridge regression problem (least squares loss, L_2 -regularization).

Proposition 44 (Excess risk, fixed design, L_2 -regularization)

Under the linear model assumption (and for the fixed design setting), the ridge least-squares estimator

$$\hat{\theta} = \frac{1}{n}(\hat{\Sigma} + \lambda I)^{-1} X^\top y \quad \text{with } \hat{\Sigma} = \frac{1}{n} X^\top X$$

has the following excess risk:

$$\mathbb{E}(\mathcal{R}_n(\hat{\theta})) - \mathcal{R}_n^* = \lambda^2 \theta^{*\top} (\hat{\Sigma} + \lambda I)^{-2} \hat{\Sigma} \theta^* + \frac{\sigma^2}{n} \text{tr}(\hat{\Sigma}^2 (\hat{\Sigma} + \lambda I)^{-2}).$$

- The first term on the rhs is the bias term (it is not 0 in this case!), the second one is the variance term.

Bound for least squares with L_2 regularization (fixed design) (2)

- Just to get a feeling for the relevant quantities, let's assume $\hat{\Sigma}$ is of the form I_d/n . Then the **bias** and **variance** terms are of the order

$$\frac{\lambda^2}{n(\frac{1}{n} + \lambda)^2} \cdot \|\theta_*\|^2 \quad \text{and} \quad \frac{\sigma^2 d}{n^3(\frac{1}{n} + \lambda)^2}$$

(To see this, exploit that $\text{tr}(I_d) = d$).

- Observe that for $\lambda = 0$ we recover bias 0 and variance $\sigma^2 d/n$ as in the OLS bound. If λ tends to increase, the bias gets large, the variance gets small.

Bound for least squares with L_2 regularization (fixed design) (3)

We now ask ourselves how to choose λ in dependency of n to make the sum of bias and variance as small as possible. As a rule of thumb, we set the bias term equal to the variance term and solve for λ . We then find the following result:

Proposition 45 (Excess risk with optimally chosen λ , fixed design, L_2 -regularization)

With the choice of regularization parameter

$$\lambda^* = \frac{\sigma \sqrt{\text{tr}(\hat{\Sigma})}}{\|\theta^*\|_2 \sqrt{n}},$$

we have

$$\mathbb{E}[\mathcal{R}_n(\hat{\theta}_{\lambda^*})] - \mathcal{R}_n^* \leq \frac{\sigma \sqrt{\text{tr}(\hat{\Sigma})} \|\theta^*\|_2}{\sqrt{n}}.$$

Bound for least squares with L_2 regularization (fixed design) (4)

- ▶ Again gaining intuition with $\hat{\Sigma} = I_d/n$ we find that this reduces to

$$\frac{\sigma\sqrt{d}}{\sqrt{n}} \cdot \|\theta_*\|$$

- ▶ This is pretty much the square root of the bound for OLS. So this bound converges slower to 0 as a function of n (the rate is now $1/\sqrt{n}$ rather than $1/n$), but it also has a milder dependence on the dimension ($\sqrt{d}$ instead of d).

Bound for least squares with L_1 regularization (fixed design)

Proposition 46 (Excess risk, fixed design, L_1 -regularization, optimal λ)

Assume that $y = x\theta^* + \varepsilon$, with $\varepsilon \in \mathbb{R}^n$ being a vector with independent Gaussian components of zero mean and variance σ^2 . Let $\hat{\theta}$ be the minimizer of the Lasso problem

$$\frac{1}{n} \|Y - X\theta\|_2^2 + \lambda \|\theta\|_1.$$

Then, for

$$\lambda = \frac{2\sigma}{\sqrt{n}} \sqrt{2\|\hat{\Sigma}\|_\infty (\log d + \log \frac{1}{\delta})},$$

we have, with probability at least $1 - \delta$,

$$\frac{1}{n} \|X(\hat{\theta} - \theta^*)\|_2^2 \leq 3\|\theta^*\|_1 \cdot \frac{2\sigma}{\sqrt{n}} \sqrt{2\|\hat{\Sigma}\|_\infty (\log d + \log \frac{1}{\delta})}$$

Bound for least squares with L_1 regularization (fixed design) (2)

Here $\|\theta^*\|_1$ describes the sparsity of the true vector. For example, for a vector with k non-zero components bounded by one the term $\|\theta^*\|_1$ is bounded by k .

This bound essentially scales as

$$\frac{k\sigma\sqrt{\log(d)}}{\sqrt{n}}$$

Bound for least squares with L_0 regularization (fixed design)

Theorem 47 (Excess risk, fixed design, L_0 -regularization, optimal λ)

Assume that $y = x\theta^* + \varepsilon$, with $\varepsilon \in \mathbb{R}^n$ being a vector with independent Gaussian components of zero mean and variance σ^2 , with $\|\theta^*\|_0 \leq k$. Let $\hat{\theta}$ be a minimizer of

$$\frac{1}{n} \|Y - X\theta\|_2^2 + \lambda \|\theta\|_0.$$

Then, for $\lambda = 8\sigma^2 \log(\sqrt{2}d)/n$, we have

$$\mathbb{E} \left(\frac{1}{n} \|X(\hat{\theta} - \theta^*)\|_2^2 \right) \leq \frac{16k\sigma^2}{n} (1 + \log d) + \frac{16\sigma^2}{n}.$$

Bound for least squares with L_0 regularization (fixed design) (2)

This bound essentially scales as

$$\frac{k\sigma^2 \log(d)}{n}$$

- ▶ Compared to OLS or ridge regression, the dependency moved from d to $\log(d)$, which is good if we work in high-dimensional spaces.
- ▶ The bound has an extra factor k that corresponds to the number of non-zero elements. If we consider k as the “intrinsic dimension” of the problem, this makes sense: it now scales with the intrinsic dimension k rather than the full dimension d . Intuitively, we get an extra factor $\log d$ compared to OLS with $d = k$ because we don't know which are the k out of d coordinates are the relevant ones.

Summary: rates for fixed design

We now summarize the rates from the previous slides for the fixed-design excess prediction error $E(\mathcal{R}_n(\hat{\theta})) - \mathcal{R}_n^*$.

Regularization / assumption	Fixed design error rate
No regularization	$\frac{\sigma^2 d}{n}$
L_2 if $\hat{\Sigma} \approx I_d$	$\frac{\sigma \ \theta^*\ _2 \sqrt{d}}{\sqrt{n}}$
L_1 if θ^* is k -sparse with bounded entries	$\frac{\sigma k \sqrt{\log d}}{\sqrt{n}}$
L_0 if $\ \theta^*\ _0 \leq k$	$\frac{\sigma^2 k \log d}{n}$

Summary: rates for fixed design (2)

Rates that scale as n are called **fast rates**, rates that scale as $\sqrt{n}$ are often called **slow rates**. We see natural fast rates appear in OLS (but with bad dependence on d). For the regularized versions, fast rates typically require strong assumptions, as we made them in the L_0 case. One can also derive fast rates for L_2 and L_1 under stronger assumptions. For example in Lasso, we need to make assumptions that ensure low covariance between variables, and assumptions how the eigenvalues of the covariance matrix $\hat{\Sigma}$ and the design matrix $\theta_*^T \theta$ behave.

Summary: rates for fixed design (3)

- ▶ OLS gives a fast rate $1/n$, but it scales with the full dimension d , so it is only useful when $d \ll n$.
- ▶ The basic L_2 and L_1 bounds above are slow rates $1/\sqrt{n}$. Fast rates are also possible, but they require stronger assumptions on the design matrix (for example, regarding its eigenvalues).
- ▶ For L_1 and L_0 regularization, the rate depends only logarithmically on d !
- ▶ The L_0 bound gives the ideal sparse fast rate $\sigma^2 k \log d/n$ (but this is not suitable for practice).

Aggregation, bagging, boosting

Motivation from statistics: Ensembles, bagging, bootstrap

Literature:

- ▶ The statistics point of view on these methods is extensively discussed in the Hastie/Tibshirani/Friedman book

Combining classifiers in ensembles

- ▶ No free lunch theorem: there does not exist a single best classifier. So we need to select classifiers / prediction functions, ideally based on their inductive bias.
- ▶ Empirically, we observe that there might be many classifiers that are specialized towards particular cases: they are really good (or really confident, if we track uncertainty) in some part of the space, but very poor / in-confident in other parts.
- ▶ Idea: is it possible to combine many classifiers to one big, overarching “committee” classifier that is confident and good in all parts of the space?
- ▶ The answer is yes, and there exist many different approaches. The big keyword is “ensemble methods”.

Before dive into two particular algorithms and their theoretical guarantees, we look at related methods in statistics (they are the origin of many ensemble methods).

Bootstrap to estimate parameters

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.

Bootstrap to estimate parameters (2)

The Bootstrap method:

- ▶ Assume you want to estimate a parameter of some unknown distribution, for example the test error of a classifier.
- ▶ Assume you have a sample $X_1, \dots, X_n$ of points and an estimate $\hat{\Theta}$ of a true parameter Θ of this population. To be able to “judge” your estimate, you would ideally like to know the distribution of the estimate $\hat{\Theta}$, for example you want to know its variance, or you want to construct confidence sets. However, the underlying distribution is unknown.
- ▶ **The idea of bootstrap is to try to construct an empirical distribution of your parameter:** 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 empirical distribution $\hat{\Theta}_1, \dots, \hat{\Theta}_B$ serves as an approximation of the true distribution of $\hat{\Theta}$ that you can use to compute its mean, its variance, confidence sets, etc.

Bootstrap to estimate parameters (3)

Why / when does it make sense?

- ▶ Assume that the estimates $\hat{\Theta}_1, \dots, \hat{\Theta}_B$ were independent. Then, by the Glivenko-Cantelli theorem, their empirical distribution would approximate the true underlying distribution, and we could readily prove guarantees on the quantities derived from the empirical distribution.
- ▶ However, in bootstrap the different estimates $\hat{\Theta}_1, \dots, \hat{\Theta}_B$ are NOT independent, because they are based on largely overlapping samples!!! So the big question underlying bootstrap theory is whether / under which conditions the empirical distribution still serves as a good approximation of the true one.
- ▶ Intuitively, this might be the case if random reweighting of our observed data mimics real sampling variability, even though the resamples are dependent. This tends to be the case for statistics based on averaging (like the mean or the variance), but for example it tends not to be the case for extreme value statistics like the maximum or minimum of a distribution.

Bootstrap to estimate parameters (4)

There exists a large theory formal theory of bootstrap, proving under which conditions the procedure is statistically sound (consistent). The standard textbook on the bootstrap method is the book by Efron / Tibshirani. A more mathematical treatment can be found in the one by Shao / Tu:

- ▶ An Introduction to the Bootstrap, by Efron and Tibshirani.
- ▶ The Jackknife and Bootstrap, by Shao and Tu

Bagging to reduce the variance

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 hope related to this procedure is that the estimate $\hat{\Theta}_{bag}$ can have a much smaller variance than each of the individual estimates $\hat{\Theta}_b$:

Bagging to reduce the variance (2)

When does it work?

- ▶ 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 . This is great, as it reduces by a factor of B !
- ▶ If the estimates are identically distributed but have a (hopefully small) positive pairwise correlation ρ , then the variance of $\hat{\Theta}_{bag}$ can be computed to be $\rho\sigma^2 + (1 - \rho)\frac{\sigma^2}{B}$. If ρ is small and B is large, this is still good
- ▶ So whether bagging will or will not be successful will depend on “how independent” we manage the $\hat{\Theta}_i$ to be.

Two opposing principles: bagging vs boosting

We will now discuss two machine learning approaches, both based on bagging/boosting, but with very different flavors:

- ▶ Random forests: We start with an ensemble that contains many deep decision trees that completely overfit. Then we build a naive average over many such trees and obtain a prediction function that generalizes well.
- ▶ Boosting: We use an ensemble of super-simple classifiers (e.g., decision stumps: trees of depth 1 or 2) that completely underfit. Then we build a specifically-weighted average over many simple classifiers and obtain a prediction function that generalizes well.

Currently, random forests and boosting are about the most successful algorithms for machine learning on tabular data.

Random Forests

Literature:

- ▶ Original paper: Breiman: Random forests. Machine Learning 2001.
- ▶ Nice 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

Bagging decision trees

We would now like to apply the bagging 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: $y_{bag} = \text{mean}(y_1, \dots, y_B)$.

Bagging decision trees (2)

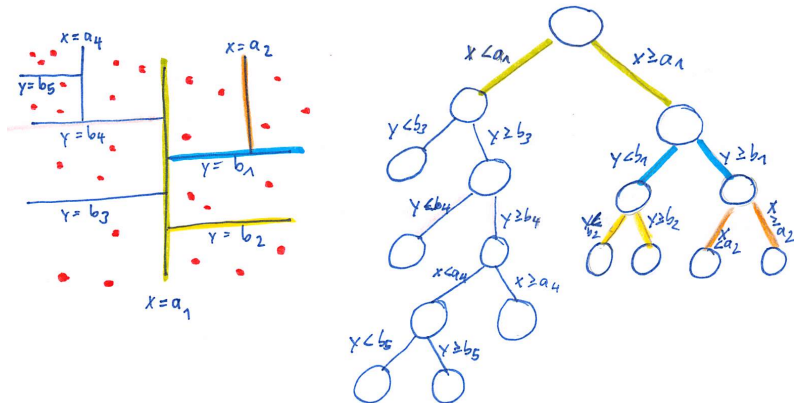
First considerations for choosing the baseline classifier:

- ▶ 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”.

Spatial decision trees on $\mathbb{R}^d$ (high-level description)

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 $\mathbb{R}^d$ (high-level description) (2)

To construct the tree, we proceed **recursively**. For each cell, we consider various splits in different dimensions, and then select one of the splits for the current cell. We keep on splitting cells until they contain only a pre-defined number of data points.

To be able to select the best split among a set of candidate splits, we need a **criterion to evaluate the quality of a proposed split**. There are many such criteria, here is one example: 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 training error we would achieve on both sides:

$$\text{error}_{\text{split}} = \sum_{i \in A} (\hat{Y}_A - Y_i)^2 + \sum_{i \in A^c} (\hat{Y}_{A^c} - Y_i)^2$$

Spatial decision trees on $\mathbb{R}^d$ (high-level description) (3)

Typically, spatial decision trees **select axis-parallel splits**:

- ▶ 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.

To use the tree 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.

Spatial decision trees on $\mathbb{R}^d$ (high-level description) (4)

Depth of the tree / 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.

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 pseudocode

Figure from "Elements of statistical learning"

Input: data points in $\mathbb{R}^p$ (p = dimension of the space)

Parameters: 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

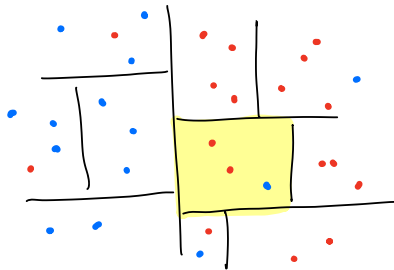
Parameters:

- ▶ 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, $n \approx \log n$.

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

- ▶ 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 (CAN YOU PROVE IT?)



Decision tree consistent:

- diam (cells) $\rightarrow 0$
- # points per cell $\rightarrow \infty$

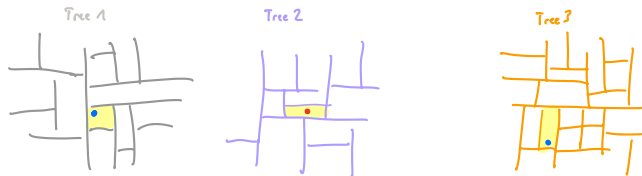
- ▶ If all individual trees are consistent, so is the random forest (see G. Biau. Analysis of a random forests model. JMLR 2012)

Consistency of random forests (2)

- ▶ 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).
- ▶ This is not obvious and very much depends on the exact setup. See also our paper “When do random forests fail” (Tang, Garreau, Luxburg 2018)
- ▶ For some intuition, see the figure on the next slide:

Consistency of random forests (3)

Each individual deep tree just has one point in each cell.



But in the forest, we still average over many points that come from a larger region

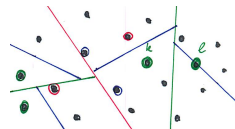
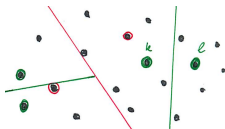
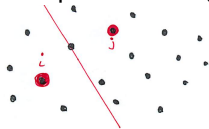


Outlook: Comparison-based random forests

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”, Haghiri, Garreau, Luxburg 2018)
- ▶ Empirically, it works really well.

Boosting

Literature:

- ▶ Shalev-Shwartz/Ben-David: Understanding Machine Learning, Section 10
- ▶ Hastie/Tibshirani/Friedman: Elements of Statistical Learning, has a long chapter on boosting and gradient boosting
- ▶ A whole book, very readable: Boosting - Foundations and algorithms by Schapire and Freund
- ▶ Original paper: Robert Schapire, Yoav Freund, 1995.

Strong and weak learners, intuitively

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.

Weak and strong PAC learners, formally

In the 1990ies, researchers studied learning in the setting of "PAC learning" (PAC = probably approximately correct), somewhat inspired from complexity theory:

Consider a data space $\mathcal{X}$ with a probability distribution $\mathcal{D}$. Consider a hypothesis class $\mathcal{H}$ of classifiers, and assume that the true classifier lives in this class, that is $y = h(x)$ for some deterministic $h \in \mathcal{H}$. Then the class $\mathcal{H}$ is called **strongly PAC-learnable** if there exists a learning algorithm such that the following holds: For every $\varepsilon, \delta > 0$, for every probability distribution $\mathcal{D}$ over $\mathcal{X}$, and every $h \in \mathcal{H}$, if we run the algorithm on $n(\delta, \varepsilon)$ many training points, the algorithm returns a function $\hat{h}$ such that with probability at least $1 - \delta$, the zero-one-loss ℓ_{01} of $\hat{h}$ is at most ε .

The class $\mathcal{H}$ is **weakly PAC-learnable** if for some $\gamma > 0$, there exists an algorithm that takes $n(\delta)$ many examples and learns a function $\hat{h}$ that with probability at least $1 - \delta$ has an error that is at least γ better than random guessing: the zero-one-loss of $\hat{h}$ satisfies $\ell_{01}(\hat{h}) \leq 1/2 - \gamma$.

Weak and strong PAC learners, formally (2)

The class $\mathcal{H}$ is **efficiently weak / strong PAC-learnable** if the learning algorithm runs in polynomial time (and in particular, only sees polynomially many samples).

For quite some time, researchers wondered whether weak learnability is “easier” than strong learnability. We will answer this at the end of this chapter.

Boosting, the outline

- ▶ Given a training set of n points $(x_i, y_i)_{i=1, \dots, n}$, a loss function ℓ , and a weak learner WL , the boosting algorithm proceeds in T rounds.
- ▶ The training points x_i have weights $w_t(x_i)$ that change from round to round. The weights for all the training points always add up to 1.
- ▶ In each round t :
 - ▶ We select the weak learner WL such that it minimizes the training loss on the points with the current weights:

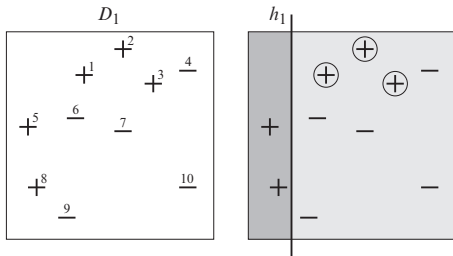
$$\sum_{i=1}^n w_t(x_i) \ell(WL(x_i), y_i)$$

- ▶ We then update the weights:
 - ▶ If point x_i was mis-classified, we increase its weight $w_t(x_i)$.
 - ▶ If point x_i was correctly classified, we decrease its weight $w_t(x_i)$.
- ▶ At the very end, the final classifier is a weighted sum of the weak classifiers of each round.

Boosting: toy example (all figures from Schapire/Freund book)

Left: data set, initialized with uniform weights

Right: a weak classifier. Its accuracy is larger than 0.5, but it gets a significant amount of points wrong (the ones indicated by a circle).

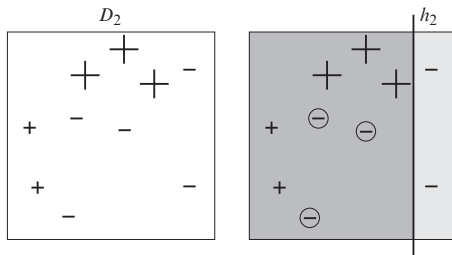


We now adapt the weights:

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.

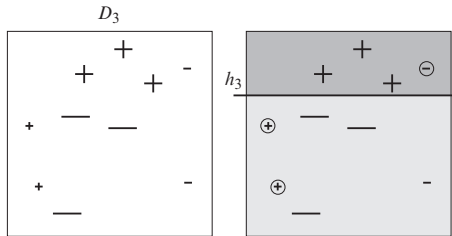


Again we reweight:

Boosting: toy example (all figures from Schapire/Freund book) (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 (all figures from Schapire/Freund book) (4)

We combine them by a linear combination (weighted majority vote):

$$H = \text{sign} \left(0.42 \begin{array}{|c|c|} \hline \text{shaded} & \text{light} \\ \hline \end{array} + 0.65 \begin{array}{|c|c|} \hline \text{shaded} & \text{light} \\ \hline \end{array} + 0.92 \begin{array}{|c|c|} \hline \text{shaded} & \text{light} \\ \hline \end{array} \right)$$

$$= \begin{array}{|c|c|c|c|} \hline \text{shaded} & \text{+} & \text{+} & \text{+} \\ \hline \text{+} & \text{shaded} & \text{shaded} & \text{shaded} \\ \hline \text{+} & \text{shaded} & \text{shaded} & \text{shaded} \\ \hline \end{array}$$

Boosting: toy example (all figures from Schapire/Freund book) (5)

First remarks:

- ▶ As opposed to random forests, which generate randomness through subsampling points and dimensions, the training set in boosting is always the same. But we change the weights of the points.
- ▶ 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

AdaBoost algorithm

Figure from "Understanding Machine Learning"

input:

training set $S = (\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$

weak learner WL

number of rounds T

sample weights

initialize $\mathbf{D}^{(1)} = (\frac{1}{m}, \dots, \frac{1}{m})$.

for $t = 1, \dots, T$:

invoke weak learner $h_t = \text{WL}(\mathbf{D}^{(t)}, S)$

compute $\epsilon_t = \sum_{i=1}^m D_i^{(t)} \mathbb{1}_{[y_i \neq h_t(\mathbf{x}_i)]}$

let $w_t = \frac{1}{2} \log \left(\frac{1}{\epsilon_t} - 1 \right)$

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 $i = 1, \dots, m$

train WL to minimize weighted loss on training points

final function

output the hypothesis $h_s(\mathbf{x}) = \text{sign} \left(\sum_{t=1}^T w_t h_t(\mathbf{x}) \right)$.

error at step t

update factor

AdaBoost algorithm

Figure from "Understanding Machine Learning"

(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.

Bounding the training error

Bounding the training error

There are many theoretical results for boosting. For example, the following theorem shows how the training error improves by boosting:

Theorem 48 (Training error of AdaBoost)

Consider the AdaBoost algorithm on a training set of size n , run for T rounds. Assume that in each round t , the weak learner always returns a function with training 0-1-error $\varepsilon_t \leq 1/2 - \gamma$ (for some $\gamma > 0$). Then the training 0-1-error of the final output function of AdaBoost is at most $\exp(-2\gamma^2 T)$.

The proof is surprisingly simple, let's look at it:

Bounding the training error (proof)

Step 1: We look at the global function f_t that has been learned up to time t :

$$f_t = \sum_{p \leq t} w_p h_p$$

(and use the convention $f_0 \equiv 0$). We define a convenient random variable, Z_t , as the empirical exponential loss of function f_t :

$$Z_t = \frac{1}{n} \sum_{i=1}^n \exp(-y_i f_t(x_i))$$

Note that for any classifier h , we can bound the 0-1-loss by the exponential loss:

$$\mathbb{1}(h(x) \neq y) = \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y \end{cases} \leq \begin{cases} 1/e \approx 0.3 \\ e \approx 2.7 \end{cases} \leq \exp(-yh(x))$$

So if we can bound Z_T by the term in the theorem, we are done.

Bounding the training error (proof) (2)

Step 2: Telescope sum to bound Z_T :

Now we want to bound Z_t and use the following trick:

$$Z_T = \frac{Z_T}{Z_{T-1}} \cdot \frac{Z_{T-1}}{Z_{T-2}} \cdot \dots \cdot \frac{Z_1}{Z_0}$$

Observe that with $f_0 = 0$ we get $Z_0 = 1$.

Now we want to show that each of these factors satisfies:

$$\frac{Z_{t+1}}{Z_t} \leq \exp(-2\gamma^2)$$

Bounding the training error (proof) (3)

Step 3: Bound each of the terms

By a simple induction one can prove:

$$\mathcal{D}_i^{(t+1)} = \frac{\exp(-y_i f_t(x_i))}{\sum_{j=1}^m \exp(-y_j f_t(x_j))}$$

Exploiting this, one can show in a couple of elementary computation steps that

$$\frac{Z_{t+1}}{Z_t} = 2\sqrt{\varepsilon_{t+1}(1 - \varepsilon_{t+1})}.$$

where ε_t denotes the error of base classifier h_t . Exploiting that $\varepsilon_{t+1} \leq \frac{1}{2} - \gamma$ (weak learning assumption!) and the fact that $1 - \alpha \leq \exp(-\alpha)$ we obtain

$$\dots \leq \sqrt{1 - 4\gamma^2} \leq \exp(-2\gamma^2)$$

(For the omitted proof details, see the Shalev-Schwartz/Ben-David book (Sec.10) and the Schapire/Freund book (Sec. 3.1)).

Bounding the training error (proof) (4)

Step 4: Combining everything:

Finally, combining everything we get:

$$\begin{aligned} \text{0-1-Training error } (f_T) ; \leq Z_T &\leq \prod_{t=0}^{T-1} \left(\frac{Z_{t+1}}{Z_t} \right) \\ &\leq \left(\exp(-2\gamma^2) \right)^T = \exp(-2\gamma^2 T) \end{aligned}$$



Bounding the test error of AdaBoost

Literature: Understanding machine learning

Theoretical results on the test error

The key result that we are interested in: does the test error also improve during the boosting procedure?

The answer is yes, and there are many different approaches to prove it. In this lecture, we will cover two of them.

Generalization bound based on VC theory

Function class related to AdaBoost:

- ▶ Assume that AdaBoost uses weak classifiers that come from the base function class $\mathcal{B}$ (for example, $\mathcal{B}$ might be the set of decision stumps).
- ▶ Then the final classifier lives in this class:

$$\mathcal{H}_T = \left\{ \text{sign} \left(\sum_{t=1}^T w_t h_t(x) \mid w \in \mathbb{R}^T, h_t \in \mathcal{H} \right) \right\}$$

- ▶ One can prove that the VC-dimension of $\mathcal{L}(T)$ can be bounded by (essentially) T times the VC dimension of the base class $\mathcal{B}$:

Generalization bound based on VC theory (2)

Theorem 49

Consider a base class $\mathcal{B}$ and the corresponding "boosting class" $\mathcal{H}_T$, for some $T \in \mathbb{N}$.
Then

$$VC(\mathcal{H}_T) \leq T \cdot VC(\mathcal{B}) \log(T \cdot VC(\mathcal{B}))$$

Proof sketch

Consider a set $C := \{x_1, \dots, x_n\}$ that is shattered by $\mathcal{H}_T$.

- Observe that a labeling of the set C is produced by a function of the form

$$f(x) = \sum_{t=1}^T w_t h_t(x)$$

Such a function can be interpreted as first choosing $h_1, \dots, h_T \in \mathcal{B}$ and then applying a linear classifier with weight w to the vector

$$(h_1(x), \dots, h_T(x)) \in \mathbb{R}^T.$$

We now count how many different labeling we can achieve in this way:

Proof sketch (2)

- Denote by d the VC-dim of $\mathcal{B}$. We know that the set $h_1, \dots, h_T$ can represent at most $\left(\frac{en}{d}\right)^d$ many different labelings of C (Sauer-Shelah-lemma).

So to select $h_1, \dots, h_T$ we have to choose T out of $\left(\frac{en}{d}\right)^d$ possible classifiers in $\mathcal{B}$:

$$\leadsto \left(\frac{en}{d}\right)^{d \cdot T} \text{ possibilities}$$

- Because linear classifiers have VC dim $T + 1$ if T is the dim of the space, the blue base functions can now be combined (using different weights) to

$$\left(\frac{en}{T}\right)^T$$

many linear classifiers

Proof sketch (3)

- So all in all there are about

$$\left(\frac{en}{d}\right)^{dT} \cdot \left(\frac{en}{T}\right)^T \leq n^{(d+1)T}$$

many possible classifiers.

- To shatter the set C , we need 2^n many functions. Shattering is possible as long as

$$2^n \stackrel{!}{\leq} n^{(d+1)T}$$

The parameter n for which this inequality stops to hold is the VC dimension

- To determine this n , we use equality and solve for n . This leads to

$$n \leq 2^{\frac{(d+1)T}{\log(2)}} \log \frac{(d+1)T}{\log(2)}$$

Ignoring all constants, the result we want to prove.

Final generalization bound

- To obtain a generalization bound, we now apply the standard VC-bound. Ignoring constants, it is roughly like

$$R(f_T) \leq R_n(f_T) + \sqrt{\frac{d \log(n/d) - \log(\delta)}{n}}$$

So to determine whether we get generalization or not, we need to look at the term d/n . In our case, $d_{\mathcal{H}} = T \cdot d_{\mathcal{B}}$, hence the key quantity that we need to look at becomes $Td_{\mathcal{B}}/n$.

- Because $\mathcal{B}$ is supposed to be a class of simple base classifiers, it would have a small constant VC dimension $d_{\mathcal{B}}$. So we get generalization from training to testing error if $T/n \rightarrow 0$.

Final generalization bound (2)

- ▶ Now we can use this bound in two ways:
 - ▶ We can either just train few rounds T , live with the training error that we have until then, and conclude by VC-arguments that the test error will be on a similar order of magnitude.
 - ▶ Or we train until our training error is really small, and check how many rounds T we needed to achieve. If T is still small compared to n , we will also have a small test error. If T is large compared to n , then we have likely overfitted and do not get generalization.

Generalization bound based on margins

The success of boosting can be even better described by a margin-based theory:
Consider the classifier constructed by AdaBoost:

$$f(x) = \sum_{t=1}^T w_t h_t(x)$$

Assume that the non-negative weights are normalized such that $\sum_t w_t = 1$. Then we say that the value $yf(x)$ is the “margin” of sample (x, y) : it specifies how close / how far the point (x, y) is from being misclassified.

One can now prove generalization bounds in terms of the margin. On the next slide we show one of the formal results (without proof, figure taken from the Schapire/Freund book):

Generalization bound based on margins (2)

θ : threshold that describes the margin.

in particular the function by AdaBoost

Theorem 5.5 Let $\mathcal{D}$ be a distribution over $\mathcal{X} \times \{-1, +1\}$, and let S be a sample of m examples chosen independently at random according to $\mathcal{D}$. Suppose the base-classifier space $\mathcal{H}$ has VC-dimension d , and let $\delta > 0$. Assume that $m \geq d \geq 1$. Then, with probability at least $1 - \delta$ over the random choice of the training set S , every weighted average function $f \in \text{co}(\mathcal{H})$ satisfies the following bound:

$$\Pr_{\mathcal{D}}[yf(x) \leq 0] \leq \Pr_S[yf(x) \leq \theta] + O\left(\sqrt{\frac{d \log(m/d) \log(m\theta^2/d)}{m\theta^2}} + \frac{\log(1/\delta)}{m}\right)$$

for all $\theta > \sqrt{8d \ln(em/d)/m}$.

true prob.

test error

empirical prob.

not the training error,

but the fraction of training points with margin $\leq \theta$

Cool: only the VC-dim of the base classifier enters the bound

$\text{co}(\mathcal{H})$: convex hull of $\mathcal{H}$. AdaBoost fcts are contained in it.

Generalization bound based on margins (3)

Somewhat simplified, ignoring constants, the bound says that for all $\theta \geq \sqrt{d/m}$,

$$P_{true}(yf(x) \leq 0) \leq P_{empirical}(yif(x_i) \leq \theta) + O\left(\frac{1}{\theta} \sqrt{\frac{d \log n}{n}}\right)$$

- ▶ So if the margin is large, we can choose θ to be a large constant. There will be few points in the margin, so the **left term** is small, and the **right term** will be small as well because θ is not too close to 0.
- ▶ What is cool is that the **complexity term** essentially contains the VC dim of the base classifier, but not of the final classifier. In particular, the number T of rounds that we used to create the AdaBoost classifier does NOT enter the bound!!!
- ▶ This is consistent with empirical observations: the performance of boosting often improves if we keep training even when the training error is already 0: the margin θ might still increase, while increasing T does not hurt.

Equivalence of weak and strong PAC learnability

For quite some time, researchers wondered whether weak learnability is “easier” than strong learnability. With AdaBoost, this question could be settled:

Theorem 50

A hypothesis class $\mathcal{H}$ is (efficiently) weakly PAC-learnable if and only if it is (efficiently) strongly PAC-learnable.

Do you find it surprising?

Equivalence of weak and strong PAC learnability (2)

Proof: Sec 4.3. in the Schapire/Freund book on boosting (the proof is not very difficult, but we will skip it here, as it requires to introduce a slightly different boosting algorithm based on resampling rather than reweighting).

Gradient Boosting

Literature:

- ▶ Hastie/Tibshirani/Friedman book, Section 10, contains an extensive treatment of boosting and gradient boosting.
- ▶ Original paper: Friedman: Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics, 2001.
- ▶ The implementation that made it very popular: Chen, Guestrin: XGBoost - A scalable Tree Boosting Algorithm. KDD 2016.

Gradient Boosting

- ▶ ... is a combination of ideas of random forests with variants of boosting.
- ▶ Optimizing for the next model in the boosting framework can be interpreted as a gradient descent on the residuals (the errors we still make). This framework can then also be generalized to other models, loss functions, ...
- ▶ is extremely successful in practice, e.g. using the XGBoost implementation.

AdaBoost, rewritten as a stagewise additive modeling approach

Let us re-consider AdaBoost from a more general perspective:

- ▶ The algorithm learns an additive model of the form

$$f(x) = \sum_{t=1}^T w_t h_t(x)$$

where h_t comes from a set of simple classifiers.

- ▶ While the AdaBoost algorithm has a very particular way of picking the next weights w_t and base classifier h_t , a more general procedure would be “forward modeling”:
- ▶ Fix a loss function ℓ . We start with one base function h_1 , fit it to the data, and from now on keep its parameters fixed. Then we add a second base function h_2 , optimize its parameters and the parameter w_2 and fix them as well. In general, at each iteration t we find w_t and h_t such that the function f_t that represents the partial sum of all weighted classifiers up to time t makes as small loss ℓ as possible.

AdaBoost, rewritten as a stagewise additive modeling approach (2)

- Let us assume that the functions h are parameterized by some vector θ , we write them as h_θ . Then in each step t we need to solve the optimization problem:

$$\operatorname{argmin}_{w_t, \theta_t} \sum_{i=1}^n \ell(y_i, f_{t-1}(x_i) + w_t h_{\theta_t}(x_i))$$

- One can prove that the AdaBoost algorithm is equivalent to running this stagewise additive modeling approach with the exponential loss function $\ell(y, f(x)) = \exp(-yf(x))$. This is pretty cool.
- However, for general loss functions and complicated space $\mathcal{H}$ of base classifiers, solving this optimization problem analytically can be quite a challenge. This is where gradient boosting comes in.

Gradient boosting

In each iteration t of the boosting algorithm, we want to solve the following optimization problem:

$$\operatorname{argmin}_{w_t, \theta_t} \sum_{i=1}^n \ell(y_i, f_{t-1}(x_i) + w_t h_{\theta_t}(x_i))$$

Ideally, one would run gradient descent to solve this problem. But sometimes our function classes are not even differentiable with respect to their parameters θ_t . So we consider an approximation: we compute the gradient of the loss function with respect to the individual function values on all data points and fit the function $w_t h_{\theta_t}(x_i)$ such that it performs an approximate gradient step. To this end, we find the function $w_t h_{\theta_t}(x_i)$ that is “as close as possible” to the negative gradient.

Gradient boosting (2)

So what we do to perform step t is as follows:

- ▶ Assume we already found f_{t-1}
- ▶ For $i = 1, \dots, n$ (for all training points) we compute the gradient

$$r_i := \frac{\partial \ell(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)}$$

Be clear to understand what this means: $f_{t-1}(x_i)$ is just a real number, maybe let's call it $\hat{y}$. We take the derivative of the loss with respect to $\hat{y}$ (and not with respect to any parameters of f_{t-1} or so). The gradient tells us in which direction $\hat{y}$ needs to move to improve the loss.

Gradient boosting (3)

- We now find the function h_{θ_t} that approximates this gradient on all training points as well as possible:

$$\theta_t = \operatorname{argmin}_{\theta} \sum_{i=1}^n (r_i - h_{\theta}(x_i))$$

Depending on the base classifiers, this step can be solved differently; essentially we solve a little regression problem in which we approximate the gradient values by a function h_{θ} .

- We then run a second quick search to find the optimal step size w_t :

$$w_t = \operatorname{argmin}_w \sum_{i=1}^n \ell(y_i, f_{t-1}(x_i) + w h_{\theta_t}(x_i))$$

- Finally, we set

$$f_t = f_{t-1} + w_t h_{\theta_t}$$

XGBoost: Gradient boosted trees

One of the most popular algorithms / implementations of gradient boosting is XGBoost. It applies gradient boosting to decision trees with shallow depth, say 4, as base classifiers.

Why using shallow decision trees as base classifiers?

- ▶ Easy to learn
- ▶ Decision trees can work with mixed data types
- ▶ They can handle variable interactions (as long as the tree depth is larger than 1, which is why one doesn't use decision stumps in practice). This is a big plus!
- ▶ The resulting final function can approximate arbitrary functions by piecewise-constant regions.
- ▶ No funny extrapolation properties (as opposed to splines, say)

Boosted trees vs. random forests

- ▶ Random forests are trained in parallel, as independent as possible from each other. We form a simple average in the end and hope that this reduces the variance of our estimator.
- ▶ Boosted trees are built sequentially, where each new tree tries to correct the errors that have been made by the previous solutions. By this approach, it targets the bias of our estimator (and perhaps the variance as well). We directly try to minimize the loss by gradient descent.

Beyond algorithms: data, measurement and validity

Data

The problem of measuring the target concept

Literature: This is briefly discussed in the Hardt Pattern book section 4, but the examples are from else where (references given explicitly on the slides).

It all starts with measurement

The main ingredient to machine learning is data, with its two facets:

- ▶ input points x_i
- ▶ output variable y_i

Obviously, the outcome of machine learning heavily depends on how we “generate” the data in the first place.

Measurement procedure vs. target construct

Whenever we “record” or “collect” data, we need to decide what to collect and how. It is important to distinguish two aspects:

- ▶ The **target construct** is the abstract concept that we are interested in and that we want to measure. Often it is not directly observable.
- ▶ The **measurement procedure** describes the “device” that is supposed to “measure” the construct. In the end, we typically come up with “numbers” that we store to (approximately) describe the target construct.

Here are some examples:

Measurement procedure vs. target construct (2)

Target construct	Measurement process
Temperature	Thermometer reading
Air quality	Sensor readings: amount of dust particles; concentration of SO ₂ , CO ₂ , CO, O ₃ , ...)
Kidney function	Blood creatinine level, glomerular filtration rate (GFR)
Pain	Patient rating scale (e.g. 1–10), facial expression
Intelligence	IQ test scores
Motivation	Self-report surveys, task persistence
Depression	Questionnaire scores, clinical assessment
Socio-economic status	Income, education level, occupation
Customer satisfaction	Survey ratings, repeat purchases

It already become obvious from this table that the measurement process might be a poor approximation of the target construct.

Categories become normative

Even worse, by forming specific categories, they become normative: How we measure a concept changes how we think about it!

Example:

- ▶ Binary categories for sex: male or female
- ▶ More gender categories: male, female, non-binary, transgender, agender, genderfluid, ...

If a public entity or a company puts out a classification system that is widely used, it has the potential to change the norms of a society.

Concrete examples: US Census

In the U.S., the standard format for census surveys follows a two-question format:

1. Ethnicity:

- ▶ Hispanic or Latino
- ▶ Not Hispanic or Latino

2. Race (select one or more):

- ▶ White
- ▶ Black or African American
- ▶ American Indian or Alaska Native
- ▶ Native Hawaiian or Other Pacific Islander
- ▶ Some other race

This is super-weird, but this is what is used in all official statistics and diversity monitoring reports. And of course this categorization has a lot of influence.

Concrete examples: US Health insurance

Ziad Obermeyer, Brian Powers, Christine Vogeli, Sendhil Mullainathan: Dissecting racial bias in an algorithm used to manage the health of populations. Science, Okt 2019:

- ▶ US Health systems rely on commercial prediction algorithms to identify and select patients that get access to “high-risk care management” programs.
- ▶ 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. 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 (“the measurement”)** rather than **illness (“the concept of interest”)**: the more money is spent on a person, the sicker he/she is supposed to be. 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, closer inspection reveals that it is not a good choice; in particular, large racial biases arise.

Concrete examples: UK liver transplant matching

- ▶ UK uses an algorithm to match donated livers to patients waiting for a liver transplant. A lot of work went into designing this algorithm.
- ▶ However, after putting it in practice it has been observed that the algorithm systematically excludes younger patients from receiving a liver transplant.
- ▶ This is of course not intended, and it took a long time until people noticed this behavior and found out the reason:
- ▶ The target objective of the matching system is to identify the recipient whose life expectancy would be increased the most through the transplant (and it would seem that young patients have an advantage here).
- ▶ The obvious way to do this would be to predict each patient's expected survival time with and without the transplant.

Concrete examples: UK liver transplant matching (2)

- ▶ **However, because of lack of data, the algorithm used a slightly different target variable: the algorithm predicts each patient's likelihood of surviving 5 years with and without the transplant.** (The choice of a 5-year period seems to be because of data availability: "This length of follow-up was selected as data were readily available ... while longer follow-up was not.")
- ▶ What now happens: Younger patients are (correctly!) predicted to be more likely to survive 5 years without a transplant than elderly patients. So younger patients' predicted net benefit (over a 5-year period) is much less than older patients'. Hence, elder patients are preferred for transplants.

Read the following report if you are interested, highly recommended!!!

<https://www.aisnakeoil.com/p/does-the-uks-liver-transplant-matching>

Summary

Machine learning does not learn the target construct — it learns the measurement!!!

- ▶ There often is a big mismatch between target construct and measurement.
- ▶ Note that this mismatch affects both input and output variables, but it is particularly harmful for the output variable.
- ▶ Different measurement procedures for the same construct can lead to systematically different conclusions.
- ▶ Different measurement procedures for the same construct can also introduce different kinds of harm, e.g. biases or discrimination.

So whenever you have to design an ML system from scratch, be aware of this problem and explicitly try to foresee what could go wrong.

Harms that can come with data

Bias and discrimination in data sets

Minorities in data sets

Minority groups might be underrepresented relative to population statistics, both in training data and in systematic test runs. Prominent example (2015): Google's image recognition system performed poorly on Black people and even misclassified some of them as "gorillas"!

Google Photos, y'all [redacted] up. My friend's not a gorilla.

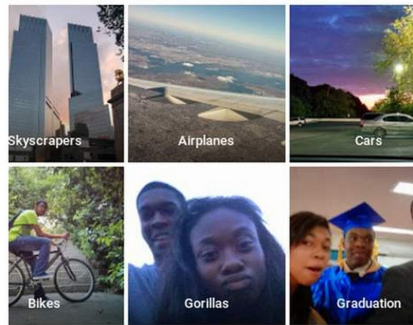


Image reference: Jacky Alcine/Twitter

Sampling bias

- ▶ The way data is collected can systematically distort the sample.
- ▶ Data collection might have demographic, geographic, behavioral, temporal biases. Certain groups will be over / under-represented in the sample.
- ▶ Example: Crime data is confounded with police activity: if the police is more active in some neighborhood, it will show a higher crime rate.

Biased patterns in data

Pre-existing biases in data, e.g. gender roles in text and images, racial stereotypes, etc:

- ▶ Historical discrimination encoded in data
- ▶ Stereotypes in text, images, labels
- ▶ Past decisions used as "ground truth"

Examples:

- ▶ past hiring data of a company (for example, over-representation of graduates from a small set of "elite" universities)
- ▶ past role distributions and employment patterns in historic employment data (eg, women working part time; in germany: "foreign" workers in less-qualified jobs)
- ▶ word embeddings: "Man is to computer programmer as womanto house wife"

Violation of ethical standards

Offensive and illegal data

Many image and text models contain offensive or even illegal data in their training set:

- ▶ ImageNet contains porn images
- ▶ A popular data set to train generative image models in 2022 was LAION-5B, a web-scraped image–text dataset. On purpose, it included links to adult sexual material, and later investigations found thousands of links to illegal child sexual abuse material (which was not on purpose, but not taken really seriously either).

Privacy violation and copyright

Public data sets often violate privacy

- ▶ Many data sets that are widely used in machine learning were used and made public without the consent of the persons represented in the data set.
- ▶ Making data sets anonymous by removing names does not protect against privacy violations!!! In many cases, people were successful in de-anonymizing data sets.
- ▶ Example: the 2006 netflix data set, consisting of movie ratings of millions of users. Even though the names of the users were removed, researchers could identify many of the users by comparing the netflix data to publically available movie ratings. (Narayanan, Shmatikov: Robust de-anonymization of large sparse data sets. 2008)
- ▶ For a more detailed discussion, see this paper: Dwork et al: Exposed! A survey of attacks on private data, 2017

Public data sets often violate copyright

- ▶ Many of the data sets used in machine learning have been copy/pasted from various sources, often scraped from the internet.
- ▶ But information being publicly available in the internet does not mean that everybody is allowed to use them for their own purposes!
- ▶ For example, there is an ongoing legal battle between the New York Times and Open AI regarding the use of NY times articles to train ChatGPT. The lawsuit is currently (2026) still ongoing.
- ▶ Similarly, many artists are concerned that generative models are trained on their data, without the artists ever giving consent or getting any revenues from it.

Garbage in, garbage out

Obvious, but ...

- ▶ Machine learning can only be as good as your data set. If the data is flawed, your machine learning model will be flawed as well.
- ▶ Having many data points can correct for unbiased measurement noise.
- ▶ But if your data is biased or discriminating, so will be your machine learning model.

The machine learning model cannot fix problems in the data in hindsight!!!

Good data practices

Target concept vs measurement

Define the target concept:

- ▶ What is the actual concept we care about?
- ▶ Is it directly observable or not?
- ▶ Are there multiple reasonable definitions?

Question its measurement:

- ▶ Does the measurement match the construct?
- ▶ Are we using a proxy? If yes: what are the risks?
- ▶ Could different measurements lead to different conclusions?

Try to identify errors and mitigate bias

Sources of errors:

- ▶ Measurement noise (sensor error, missing data)
- ▶ Systematic bias (sampling, survey design)
- ▶ Human factors (misreporting, low effort)

Is the data set representative?

- ▶ Who is included in the data?
- ▶ Who is missing?
- ▶ Does the data reflect the target population?

Data sheets for data sets!

Gebru et al: "Datasheets for datasets", 2018

- ▶ The outcome of machine learning algorithms often depends on seemingly small details and choices that have been made during the data creation and preprocessing phase.
- ▶ **Every dataset should be accompanied with a datasheet that documents its motivation, composition, collection process, recommended uses, and so on.**
- ▶ Datasheets have the potential to increase transparency and accountability within the machine learning community, mitigate unwanted societal biases in machine learning models, facilitate greater reproducibility of machine learning results, and help researchers and practitioners to select more appropriate datasets for their chosen tasks.

Data sheets for data sets! (2)

To break it down: Document everything you do!

- ▶ How was the data collected?
- ▶ What decisions were made during collection, and during pre-processing?
- ▶ What are known limitations?

Respect privacy, copyright, and ethical standards

- ▶ Was data collected with consent?
- ▶ Is it conceivable that individuals could be re-identified by a privacy attack (as in the Dwork paper)?
- ▶ Are there legal or ethical concerns?

Be aware of limitations

Many problems with machine learning are not based on algorithms, they come from data and measurement!!!!

There might be situations where we might simply want to refrain from using machine learning.

Features and representation

Literature: Chapter 4 in the Hardt book on Patterns, predictions and actions

Explicitly constructing features

Data representation and Feature representation of data

Many ML algorithms assume that the data points are in $\mathbb{R}^d$. This 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”.

Data representation and Feature representation of data (2)

Example: items and users on an online platform:

- ▶ 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 user1 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	
⋮				

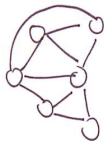
how often was book 3 bought together with book 2

Data representation and Feature representation of data (3)

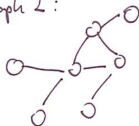
Example: motif representation of graphs (such as chemical molecules)

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

...

Data representation and Feature representation of data (4)

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.

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 .

Going non-linear by choosing basis functions

Going non-linear in the data representation

- ▶ Assume we have data that lives in $\mathbb{R}^d$ already.
- ▶ We could, for example, fit a linear model by OLS:

$$f(x) = w^\top x, \quad \hat{w} = \arg \min_w \|Xw - y\|^2$$

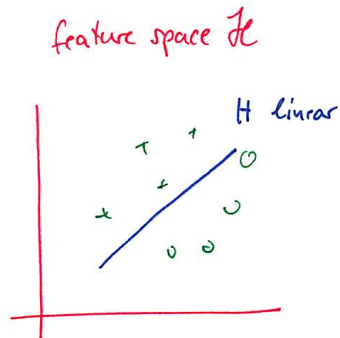
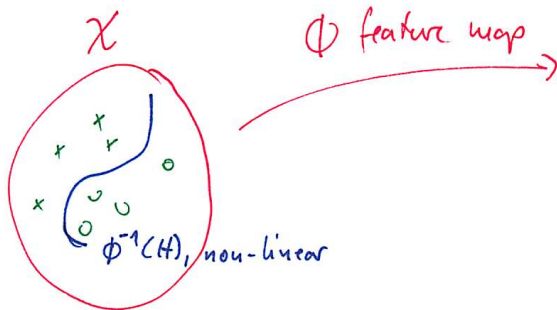
But linear relationships between inputs and outputs are often not powerful enough.

- ▶ There is one simple idea by which we can still keep the simplicity of linear methods, yet get non-linear: Map input points x_i to a “feature space” $\mathbb{R}^D$ first, then apply OLS there.

$$x_i \mapsto \Phi(x_i) \in \mathbb{R}^D, \quad \hat{f}(x_i) = \hat{w}^\top \Phi(x_i)$$

- ▶ If the mapping Φ is non-linear, the resulting final model is also non-linear in x . But it is linear in $\varphi(x)$, so we can apply OLS in the feature space $\mathbb{R}^D$ to learn a model.

The feature map



Examples

Polynomial basis. If your data lives in $\mathbb{R}$, you could transform it into a polynomial basis:

$$\Phi : \mathbb{R} \rightarrow \mathbb{R}^{D+1}$$

$$\Phi(x) = (1, x, x^2, \dots, x^D)$$

A similar principle can be applied to data in $\mathbb{R}^d$, where you take products of powers of coordinates as features, such as $(x^{(2)})^3(x^{(5)})^3$.

Observe that if one uses all such combinations, then the number of features grows exponential in the degree of the polynomial, though.

Examples (2)

Fourier basis: Assume your data lives in $\mathbb{R}$, then we can transform it as follows:

$$\Phi(x) = (1, \cos(x), \sin(x), \cos(2x), \sin(2x), \dots)$$

This is a natural approach for periodic signals (due to Fourier decompositions). In general, any square-integrable function can be approximated arbitrarily well by taking enough terms.

Examples (3)

Radial basis functions (RBFs). Fix centers $c_1, \dots, c_D \in \mathbb{R}^d$ and a scaling parameter $\sigma \in \mathbb{R}$ and map $x \in \mathbb{R}^d$ to $\mathbb{R}^D$ via the coordinates

$$\varphi_j(x) = \exp\left(-\frac{\|x - c_j\|^2}{2\sigma^2}\right)$$

The overall feature mapping is then

$$\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^D, \quad \Phi(x) = (\varphi_1(x), \dots, \varphi_D(x))^T$$

Each basis function looks like a Gaussian, centered at c_j .

The problem with explicit basis construction

In practice, it is not really clear how exactly to choose the basis, how to set its parameters, etc.

Much of machine learning since around 2000 tries to avoid constructing a basis explicitly.

Implicit feature representation by kernel methods

Literature:

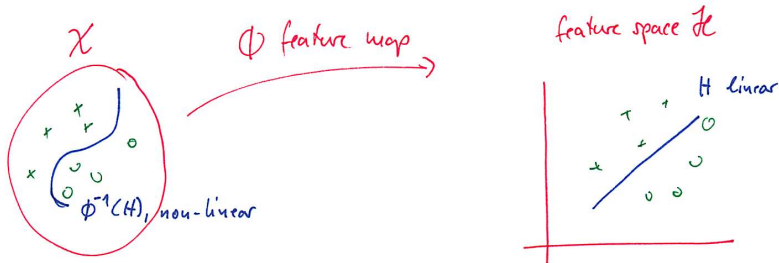
Kernels are treated as chapters in many ML textbooks, for example in

- ▶ Understanding machine learning
- ▶ Tibshirani / Hastie
- ▶ Bach
- ▶ ...

There are also complete books just dedicated to kernels, for example the one by Schölkopf/Smola (focus on algorithms and some theory) or Steinwart (pure theory).

Non-linear feature maps again

Have already seen the idea of a feature mapping:



The idea of kernel methods is to achieve a non-linear feature mapping φ without ever computing it explicitly. This works whenever we have linear machine learning algorithms that can be expressed purely in terms of scalar products of input points of the form $\langle x_i, x_j \rangle$

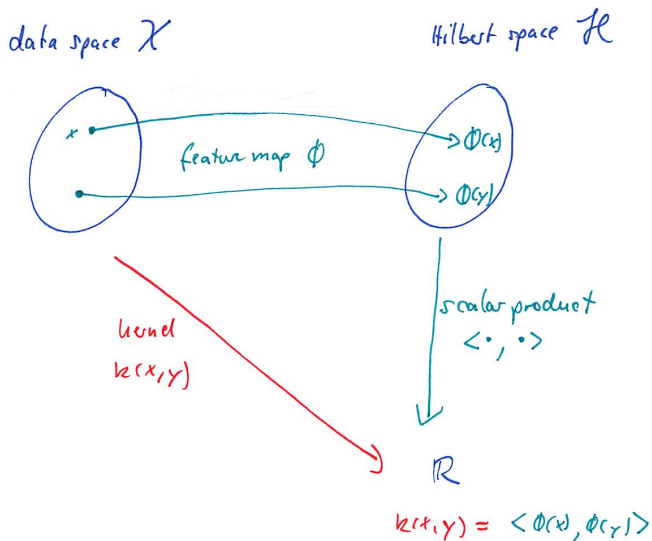
Kernel methods — the overall picture

- ▶ Given points in some space $\mathcal{X}$
- ▶ Would like to (implicitly) embed the points into space $\mathbb{R}^d$ via a (non-linear) feature map Φ
- ▶ In that space, we use a linear machine learning method
- ▶ 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)



Examples for kernels

Example: Gaussian kernel

- ▶ For two points x_i, x_j in our original space $\mathbb{R}^d$ and a fixed parameter $\sigma \in \mathbb{R}$, we define

$$k(x_i, x_j) := \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

- ▶ It encodes a notion of similarity based on the Euclidean distance between points.
- ▶ It is the “default kernel” whenever your data lives in $\mathbb{R}^d$

Many specific kernels have been devised for specific applications, just take the example of graph kernels. Even in this field, there are many variants, for example to graphs are considered similar if random walk trajectories behave similar in both graphs.

Some history

- ▶ A big part of the machine learning research in the decade between 2000 and 2010 went into designing and understanding kernel methods.
- ▶ The most prominent algorithm is the support vector machine: it essentially learns a linear classifier with a large margin, but in the feature space.
- ▶ There is a beautiful theory related to kernel methods, but we won't touch it (it is still part of the old slides and videos if you are interested).

Random feature models

Literature:

- ▶ Original paper: Rahimi, Recht: "Random features for large-scale kernel machines." NeurIPS (2007).
- ▶ Shortly mentioned in the Hardt pattern book, section 4
- ▶ Also mentioned in the Bach book section 7.4.3

Random Fourier features

- Assume we want to learn a function f , and we believe that representing this function through a Fourier decomposition would be helpful. It is well known that any real-valued, square-integrable function $f \in L^2(\mathbb{R}^d)$ admits a representation

$$f(x) = \int_{\mathbb{R}^d} c(\omega) \cos(\omega^\top x + b(\omega)) d\omega,$$

where ω are the frequencies and $c(\omega)$ and $b(\omega)$ describe the corresponding amplitudes and phases.

- Instead of representing f in this continuous integral, we could try to approximate it by a finite sum over selected frequencies ω . But maybe we don't really know which frequencies ω would be important.
- Funny idea: we could try to sample frequencies randomly. This is what a random Fourier feature model does.

Random Fourier features (2)

Given data $(x_i, y_i)_{i=1, \dots, n}$, here is how it works:

Step 1: we bring our input data $x_1, \dots, x_n \in \mathbb{R}^d$ in a new, random (!) representation:

- ▶ We draw some random parameters (all independently):
 - ▶ D random projection vectors $\in \mathbb{R}^d$:

$$\omega_1, \dots, \omega_D \stackrel{\text{iid}}{\sim} \mathcal{N}(0, \sigma^{-2} I_d)$$

- ▶ D phase shift scalars $\in \mathbb{R}$:

$$b_1, \dots, b_D \stackrel{\text{iid}}{\sim} \text{Uniform}[0, 2\pi]$$

- ▶ For data point $x \in \mathbb{R}^d$, the j -th random feature is given by

$$\varphi_j = \cos(\omega_j^\top x + b_j)$$

so new representation of point x is given as $(\varphi_1, \dots, \varphi_D) \in \mathbb{R}^D$.

Random Fourier features (3)

- Taken together, these random features map a data point from $\mathbb{R}^d$ to a representation in $\mathbb{R}^D$. Typically D is large.

Step 2: we learn a linear model on the new data representation:

- Once the random features w_j , b_j are fixed, we want to learn a function of the form

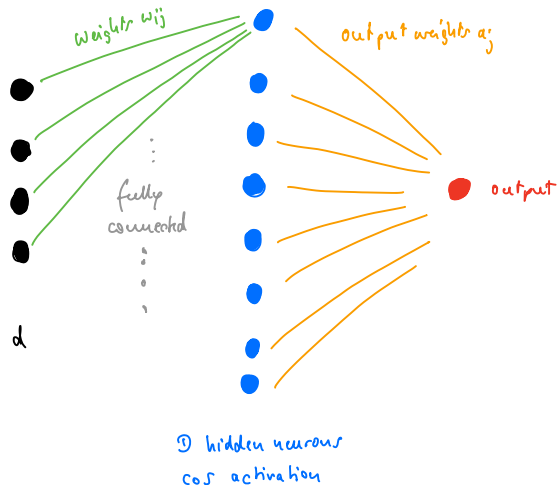
$$f(x) = \sum_{j=1}^D a_j \cos(\omega_j^\top x + b_j).$$

We now use least squares regression (or related methods) to learn parameters a_j such that f fits the training labels y_i .

- The choice of the sampling distribution of the ω_j determines which functions are well-approximated by the finite sum (e.g., Gaussian weights work well for smooth functions, more heavy-tailed distributions lead to less smooth functions).

Random Fourier features (4)

Note that this model can be interpreted as a neural network with one hidden layer with D neurons, random weights w_{ij} , cosine activation with random offset b_j , and linear output layer. Step 1 consists in drawing the random weights and offsets. Then they remain fixed, and in Step 2 we train the output weights a_j .



Random ReLU features

- ▶ We can also come up with a different random feature model by using different activation functions. For example, we could use the ReLU function $\sigma(x) = x_+ = \max\{0, x\}$.
- ▶ In general, this network represents functions of the form

$$f(x) = \sum_{j=1}^m a_j (w_j^T x + b_j)_+$$

- ▶ Assume the weights w_{ij} and the offsets b_j are initialized randomly and then kept fixed, and only the output weights a_j are being trained.
- ▶ The representation of the original data in the hidden layer is called a random feature model with ReLU activation.

Many more random feature models

Even more general:

- ▶ Assume we are given a parametric family of basis functions, where each basis function $b_\theta : \mathbb{R}^d \rightarrow \mathbb{R}$ takes a vector $x \in \mathbb{R}^d$ as input and depends on a parameter $\theta \in \Theta$.
- ▶ We can construct a random feature map by choosing $\theta_1, \dots, \theta_D$ randomly from some probability distribution μ on Θ and constructing the feature map

$$\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^D, \quad x \mapsto (b_{\theta_1}(x), \dots, b_{\theta_D}(x))^T$$

- ▶ We then consider prediction functions of the form

$$f(x) = \sum_{j=1}^D a_j b_{\theta_j}(x)$$

Many more random feature models (2)

When constructing random feature maps, it is good if the basis functions are linearly independent, and ideally even nearly orthogonal (in some appropriate space). (WHY?)

But why would a representation in random features be useful?

The answer is: Random features are closely connected to kernels!

From random features to kernels

- ▶ Consider a random feature mapping $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$, $x \mapsto \Phi(x)$.
- ▶ For this map Φ , the empirical kernel can be written as

$$k_{emp}(x, x') = \left\langle \Phi(x), \Phi(x') \right\rangle = \frac{1}{D} \sum_{j=1}^D b_{\theta_j}(x) b_{\theta_j}(x')$$

- ▶ We can now try to understand this empirical kernel by studying its expected version:

$$\begin{aligned} k_{expected}(x, x') &= E_{\theta_1, \dots, \theta_D} \left(\frac{1}{D} \sum_{j=1}^D b_{\theta_j}(x) b_{\theta_j}(x') \right) \\ &= E_{\theta} \left(b_{\theta}(x) b_{\theta}(x') \right) \\ &= \int b_{\theta}(x) b_{\theta}(x') d\mu(\theta) \end{aligned}$$

From random features to kernels (2)

- ▶ Under suitable assumptions this expression represents a kernel. In any case, it encodes a particular similarity function between the data points.
- ▶ In some cases, one can even find an analytic expression for this kernel:
 - ▶ Consider original points $x, x' \in \mathbb{R}^d$ and $\Phi(x), \Phi(x') \in \mathbb{R}^D$ their representation under the **random Fourier feature model**. Then one can show that their inner product approximates the **Gaussian kernel** k_{Gauss} (the larger D , the better the approximation):

$$\langle \Phi(x), \Phi(x') \rangle_{\mathbb{R}^D} \approx k_{Gauss}(x, x')$$

(Reference: Rahimi/Recht paper).

- ▶ Similarly, one can analytically derive which kernel corresponds to the random ReLU representation (somewhat complicated; skipped from the slides). It has several names, ReLU kernel, Neural Network Gaussian Process kernel, and others. See the Bach book.
- ▶ The neural network related to the random features can then be interpreted as approximating a kernel classifier!

From kernels to random features

The other way round, one can also start with a kernel and try to derive a random feature approximation. Here is just a sketch of the idea:

- Under some conditions, a kernel function with inputs from $x, x' \in \mathbb{R}^d$ admits a representation as follows:

$$k(x, x') = \int_{\Theta} \varphi(x, \theta) \varphi(x', \theta) d\mu(\theta)$$

where μ is a probability distribution over the space $\Theta \subset \mathbb{R}$ (cf. Bochner's theorem for shift-invariant kernels; Mercer theorem).

- We can now approximate this integral by an empirical average where we sample $\theta_1, \dots, \theta_D$ iid from μ :

$$\hat{k}(x, x') = \sum_{j=1}^D \varphi(x, \theta_j) \varphi(x', \theta_j)$$

From kernels to random features (2)

- The values $\varphi(x, \theta_j)$ for $j = 1, \dots, D$ are then a random feature representation of x .

Note that while this construction works in theory, it is difficult to analytically derive the distribution μ , the functions φ , and to implement the corresponding sampling mechanism that would lead to the random feature representation for a particular kernel k .

Summary random feature model

- ▶ Random feature models are an elegant way to make connections between neural networks and kernel methods.
- ▶ They sometimes help us to study properties of neural networks, see later.

Representation learning

From features to representations

So far, we have seen different ways to represent data:

- ▶ Explicit feature maps $\Phi(x)$, hand-designed
- ▶ Implicit feature maps via kernels
- ▶ Random feature models

All of these approaches assume that the features / representation is fixed in advance.

Representation learning: we do not design the features ourselves (explicitly or implicitly), but we directly learn a representation from our data.

Representation learning

- ▶ We still need to start with our data in some representation, typically “raw data”: pixels of an image; tokens in text; a time series
- ▶ We then train a neural network on this raw representation. If we apply the trained network to a particular data point, each layer of the network can be interpreted as a particular representation of our data.

$$x \longmapsto h_1(x) \longmapsto \cdots \longmapsto h_L(x)$$

We consider the **final layer** as the learned representation; this is the one used to finally classify the result.

Advantages / Disadvantages

Hand-designed features:

- ▶ Hand-designed features require domain knowledge; great if we have it, then by all means do make use of it.
- ▶ However, be aware that they also might miss some important aspects.

Learning features:

- ▶ When we learn the representation from raw data, we hope that the neural network identifies the important structure itself.
- ▶ This is particularly helpful for cases where we do not have good ideas how to describe data in terms of feature. Prime examples: images and text data.
- ▶ In practice, this only works if we have a huge amount of data.
- ▶ The resulting interpretations might work well for learning, but are hard /impossible to interpret. This makes it difficult to run sanity checks, debug your code, really understand what is going on.

Validation and benchmarking

Literature:

Parts of the material and many figures in this section are taken from the book by Moritz Hardt: The Emerging Science of Machine Learning Benchmarks. In print / available online.

Some papers: Liao, Taori, Raji, Schmidt: Are we learning yet? A meta review of evaluation failures across machine learning. NeurIPS Datasets and Benchmarks Track, 2021.

Validity Threats for Foundation Model Research. Gunnar König, Martin Pawelczyk, Ulrike von Luxburg, Sebastian Bordt, Arxiv 2026.

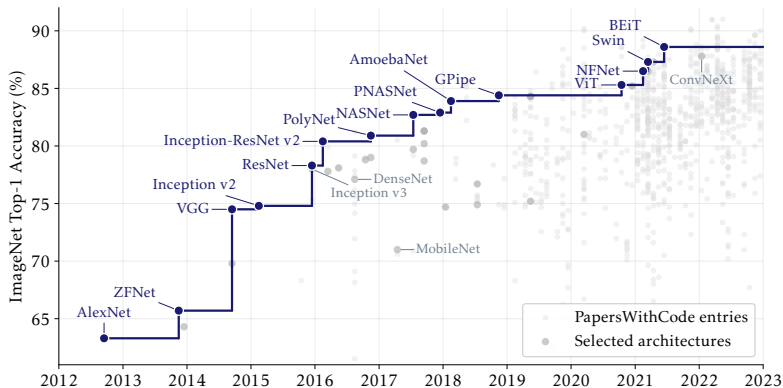
More references on the slides.

Benchmarking in machine learning

Benchmarking in machine learning

For a long time, machine learning researchers happily compared their algorithms on benchmarks, in competitions, etc, in order to develop better and better learning algorithms and to advance the field. The same benchmarks were/are used over and over again.

ImageNet Classification Progress (2012–2022)



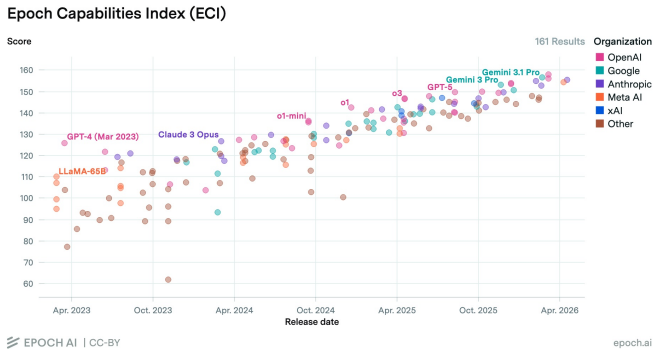
Benchmarking in machine learning (2)

This raises many challenges:

- ▶ Does the community overfit on particular benchmarks?
- ▶ How to construct good benchmarks that actually report something worthwhile?
- ▶ In howfar do results obtained on benchmarks carry over to the real world?
- ▶ What if the benchmark is only a very rough proxy of what we are interested in?
- ▶ Which tasks can actually be reasonably evaluated on benchmarks?
- ▶ How to measure success of machine learning methods? Are benchmarks a good choice in the first place?

Benchmarking in machine learning (3)

By now, benchmark scores even became a strategic target for big companies (image source: Epoch AI, downloaded April 2026)



Benchmarking in machine learning (4)

This even raises more challenges:

- ▶ How can we measure performance in a fair way?
- ▶ Which benchmarks to use (and who “decides”)?
- ▶ How to combine the results of many benchmarks to come up with a final ranking?
(impossibility results in voting/ranking)
- ▶ How to set the incentives to prevent strategic behavior?

Four notions of validity

Four notions of validity, borrowed from the social sciences

Already in the 1960ies, the field of social science had asked similar questions, and developed a framework of different types of validity:

- ▶ **Statistical validity:** If we estimate some phenomenon on a finite sample, in howfar is it representative for the underlying distribution?
- ▶ **Internal validity:** Does a study really identify some kind of causal effect within the studied sample? Is it free from confounding or bias?
- ▶ **External validity:** Do the findings generalize to other populations or setups?
- ▶ **Construct validity:** Does the study actually capture the intended abstract concept that we are interested in?

Four notions of validity, illustrated on ImageNet

Illustrated on ImageNet, these read as follows:

- ▶ **Statistical validity:** Are the observed performance differences statistically reliable, or could they be due to randomness? If one method performs 0.2% better than another one, is this just due to sampling issues or “true”?
- ▶ **Internal validity:** Are the reported performance differences caused by the models themselves, rather than artifacts of the dataset, pre-processing, or evaluation protocol?
- ▶ **External validity:** Do the benchmark results generalize to other datasets, tasks, or real-world settings beyond ImageNet?
- ▶ **Construct validity:** Does the benchmark metric “ImageNet accuracy” actually measure the intended concept “object recognition”?

Validity of machine learning benchmarks

Are results that are obtained on benchmarks reliable?

- ▶ Consider the situation where the ML community focuses on a particular benchmark, say ImageNet.
- ▶ We run many machine learning architectures and compute the test scores on ImageNet, and create rankings based on these scores.
- ▶ How “meaningful” are these scores? Do they at least have internal and perhaps also external validity?

Case studies about internal validity

Main question: if we were to repeat ImageNet experiments with a "new sample if ImageNet", do we get similar results?

To study this question, people carefully (re-)created test sets for widely used benchmarks (eg, following the procedure by which ImageNet was created and adding more images). Then they train various models on the original ImageNet training set and compare the test scores on the original test set with the ones on the new test set.

On the plots we see original test accuracy (x-axis) vs new test accuracy (y-axis). The dashed line depicts the diagonal with equal accuracy. Each point is one trained model.

Case studies about internal validity (2)

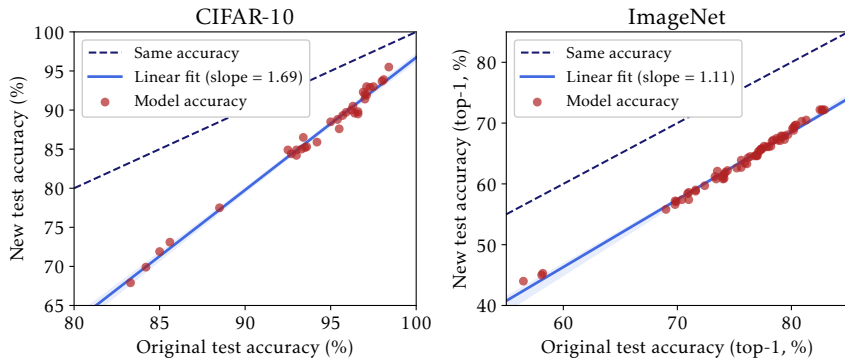
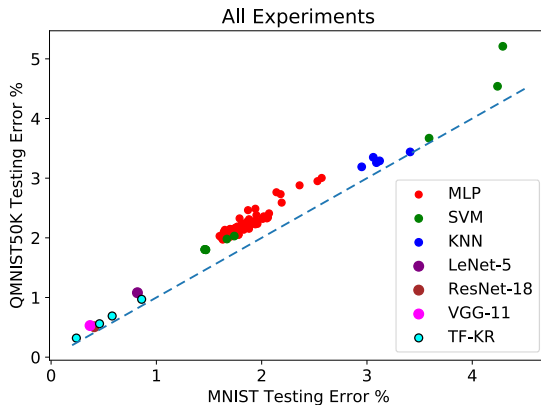


Figure 7.2: Model accuracies on the original versus new test set for CIFAR-10 (left) and ILSVRC 2012 (right). Each data point corresponds to one model from a set of representative models.

Recht, Roelofs, Schmidt, Shankar: Do ImageNet classifiers generalize to ImageNet? ICML, 2019

Case studies about internal validity (3)

Similar results on recreated test images for MNIST (attention, here we plot error, not accuracy, so higher is worse):



Yadav, Bottou: Cold Case: The Lost MNIST Digits, NeurIPS 2019.

Case studies about internal validity (4)

The two primary findings were:

- ▶ **Accuracy numbers drop significantly** between the old and the new test set (which is what people expected; limited internal validity of the test scores themselves).
- ▶ **Model rankings are largely preserved** (which is surprising; internal validity for rankings holds)

Case studies about external validity

The obvious next question is about external validity: do model rankings achieved on one benchmark carry over to other benchmarks?

First setup:

- ▶ Create two similar data sets (e.g., ImageNet and ImageNot).
- ▶ For many architectures, evaluate two cases: train and test on ImageNet; train and test on ImageNot
- ▶ Then look at the rankings induced by ImageNet resp. ImageNot.

The following slides report the results of: Salaudeen, Hardt: "ImageNot: A contrast with ImageNet preserves model rankings", Arxiv, 2024.

Case studies about external validity (2)

Surprisingly, rankings still carried over.

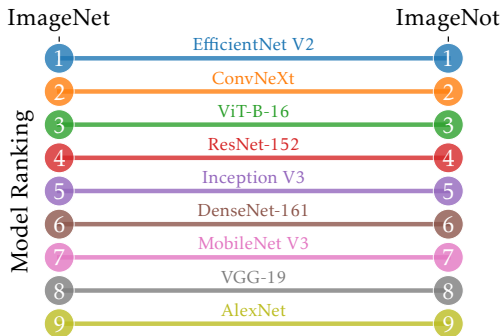


Figure 7.3: Model rankings are preserved between ImageNet and ImageNet.

Case studies about external validity (3)

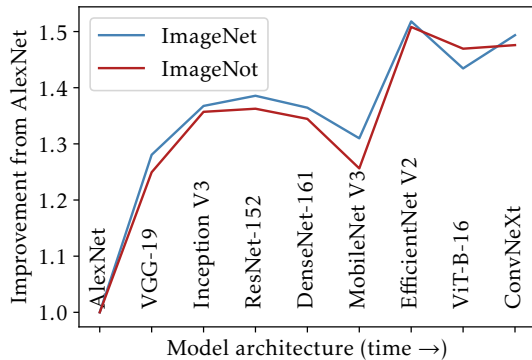
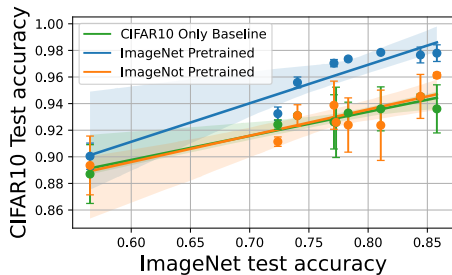


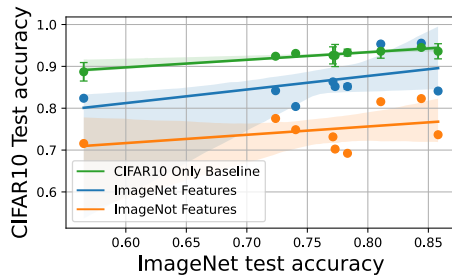
Figure 7.4: Relative accuracy improvements from one model to the next are approximately the same on ImageNet and ImageNot.

Case studies about external validity (4)

Second setup: What if we train on one data set, and then fine-tune to another one? Similar results:



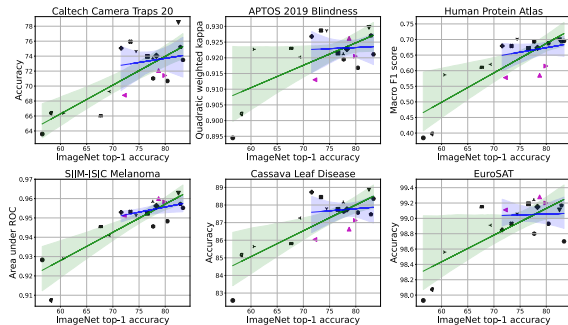
(a) **Fine-tuned** to CIFAR10—test accuracy compared to standard ImageNet test accuracy for each model architecture



(b) **Transfer Learning** to CIFAR10—test accuracy compared to standard ImageNet test accuracy for each model architecture.

Case studies about external validity (5)

Third setup, with more negative results: Pre-train models on ImageNet, evaluate on real-world tasks, e.g., classifying images from camera traps or satellites



Although there is a strong linear trend between ImageNet accuracy and the target metrics (green), these trends become less certain when we restrict the models to those above 70% ImageNet accuracy (blue). Fang, Kornblith, Schmidt, “Does progress on ImageNet transfer to real-world datasets?” NeurIPS 2024

Summary

- ▶ *Absolute* benchmark numbers, such as accuracy numbers, may satisfy the minimal bar of code re-execution. But even seemingly benign forms of distribution shift change absolute numbers significantly. In particular, absolute numbers do not have external validity.
- ▶ *Relative* model comparisons and model rankings in the ImageNet era robustly satisfy internal validity. Model *rankings* replicate under reasonable attempts to recreate original testing conditions. Model rankings are sometimes stable even under major dataset variation. In particular, they show signs of external validity.
- ▶ However, if we really change data sets and tasks, rankings of the good models do not really reproduce! No external validity to more different data sets or tasks.
- ▶ So in general, the scope of external validity is not fully known.

Validity threats in foundation model research

Validity Threats for Foundation Model Research. Gunnar König, Martin Pawelczyk, Ulrike von Luxburg, Sebastian Bordt, Arxiv 2026.

Validity threats of foundation model research

- ▶ Foundation model research / large language model research: many big questions, but to answer them we would need to train many different foundation models in different setups and compare the outcomes.
- ▶ But this is impossible because of limited compute (in particular, in an academic environment)
- ▶ To deal with this, researchers run “simplified” experiments that try to answer the given question as well as possible.
- ▶ However, savings in compute come at the cost of hidden and sometimes untestable assumptions, which introduce validity threats one should be aware of.
- ▶ Let us discuss a number of different approaches

Proxy approach: replace large experiments with small ones

Examples for the proxy approach:

- ▶ Instead of training a big model, train a small model to perform the task.
- ▶ Instead of training on a task, we just fine-tune a pre-trained model.
- ▶ We measure validation loss rather than task performance.

Proxy approach: replace large experiments with small ones (2)

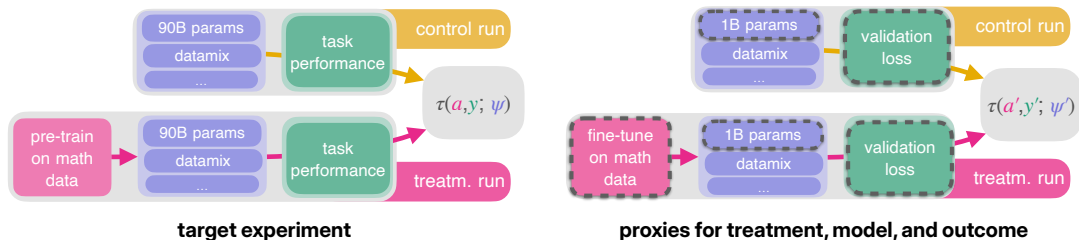


Figure 2: **The proxy approach.** We conduct an experiment but replace one or several parts with proxies (---) to make it computationally feasible. For example, we conduct the experiment with a **proxy model** (1B instead of 90B parameters), a **proxy treatment** (fine-tuning instead of pre-training on math data), or a **proxy outcome** (validation loss instead of task performance).

Key threat: External validity: do the results still hold in the target setting?

Observational approach

- ▶ Asks how particular training choices might influence the outcome.
- ▶ Instead, of running experiments oneself, researchers analyze publicly available meta-data of existing trained models, like open source or open weight models.

Observational approach (2)



Figure 3: **The observational approach.** Instead of running new experiments, we can leverage the publicly available meta-data of existing training runs. This data is observational, meaning that we cannot perform treatments and observe their effects. Instead we have to use the statistical relationships between *recipe components* and *outcomes* in combination with (untestable) causal assumptions to *identify* treatment effects.

Observational approach (3)

Key threats:

- ▶ Statistical validity: Do we have enough independent data to be able to estimate the statistical patterns?
- ▶ Internal validity! As in all research approaches that use observational data to make claims about causal effects ...

Single run approach

Single run approach:

- ▶ For some research questions, we can frame one run as containing a population of qualitatively similar, independently treatable units with separate outcomes (example: data memorization).
- ▶ Thus, within a reasonably large, single training run we can have many small experiments in parallel (eg, including a document a number of times in the training data to measure memorization)

Single run approach (2)

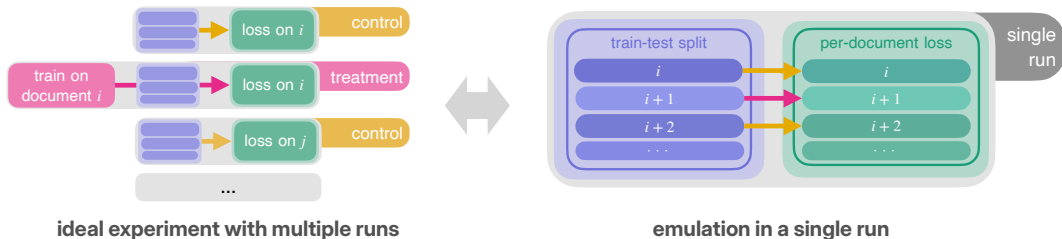


Figure 4: **Analyzing a single training run.** Single-run analysis emulates multiple training runs (left) from one (right). Here, documents are regarded as independent “units”, their assignment to the training set as the “treatment”, and their respective loss as the “outcome”.

Key threats: Internal validity (are the tasks really independent? Do we identify causal effects with this approach?); external validity (would the result generalize to other setups)?

Validity threats, summary

- ▶ Machine learning is so complex that we cannot use the trial-and-error and naive train/test setup to make firm conclusions.
- ▶ **We need to develop scientific approaches to validation (they largely do not exist yet).**
- ▶ We need to be critical about our own setups and discuss validity threats. Ideally, this should happen in every paper that gets published.

Learning in the over-parameterized interpolation regime

Puzzling empirical results that seem
at odds with classical theory

Learning in the over-parameterized regime

In the over-parameterized regime we deal with models that have much more parameters than we have data points. Intuitively, in this regime the model could simply memorize the training data without any generalization.

Hence it is super-surprising that machine learning works in this regime!

Learning in the over-parameterized regime (2)

- ▶ Images: Training a neural net (say, 20 million parameters) on ImageNet (1 million images)
- ▶ Language: GPT4 has about 10^{10} many parameters. Its training data has of the order of 10^{12} many tokens. Sounds like we have more tokens than parameters — but maybe the simple number of tokens is not the right way to count: we believe that the “effective signal” contained in this data is much smaller than the number of tokens. Similarly, the number of parameters might also not be the best way to specify the size of the model. But still ...
- ▶ But also in much simpler applications, people train DNNs with many parameters to fit relatively few sample points, and it often works.

Overfitting might not hurt!

The first big surprise with learning in the over-parameterized regime is that the models are typically trained to perfectly fit the training data (have training error 0), and often one even continues training far beyond the point where we already have 0 training error!

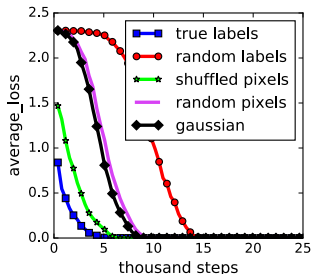
But still the models generalize!

Giving it a bit more thought, it becomes all the more surprising, given that the large function classes contain many, many functions that have close to 0 training error. And many of them would not generalize to new, unseen data (recall the example of overparameterized linear models we discussed earlier).

Function classes seem too large: DNNs can perfectly fit random labels!

Empirical experiment to find out how large the DNN function classes are:

train an MLP on images with randomly shuffled labels (CIFAR10, 60.000 images, 10 classes). We see that the MLP always achieves training error 0. This indicates that the shattering coefficient of the function class is maximal. On image net, the result looks very similar (1 million images, 1000 random labels).



Reference: Bengio et al, Understanding deep learning requires rethinking generalization, 2017

Why don't we get stuck in local optima?

DNNs induce highly non-convex optimization problems in very high-dimensional spaces. Yet it is very surprising to see that the optimization algorithms manage to find good local (global?) optima.

Need new tools

All this does not mean that the traditional way of looking at machine learning 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.

Principles that enable deep learning

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: Acta Numerica, 2021.

The statistics puzzle: why interpolation not always hurts

In the overparameterized regime we train to complete overfitting. The traditional bias-variance description would predict that the test error would explode. Why is it not the case????

Primer: random feature example

Let's study a toy example where we have few points in very high-dim space. We use a random feature model to illustrate it:

Original data: Fix a low dimension d , say $d = 5$, and a low sample size, say $n = 200$. We sample $x_1, \dots, x_n$ uniformly from the sphere in $\mathbb{R}^d$. Fix one vector $v^* \in \mathbb{R}^d$. As true output we define the non-linear function

$$f(x) = \left(\frac{1}{4} + \langle v^*, x \rangle^2 \right)^{-1}$$

As training data, we observe the true function plus noise $\varepsilon_i \sim N(0, \sigma^2)$:

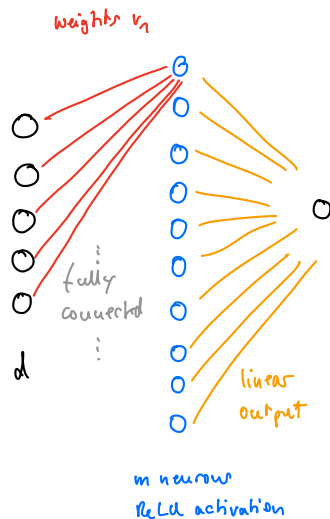
$$y_i = f(x_i) + \varepsilon_i$$

We now consider the sample $(x_i, y_i)_{i=1, \dots, n}$ and want to solve the regression problem of learning f . We use least squares loss.

Primer: random feature example (2)

Transformed data: We now transform the inputs by a random feature model: we sample m random vectors $v_1, \dots, v_m$ from the sphere, and for $x \in \mathbb{R}^d$ consider the m -dimensional random feature representation $(v_1^\top x, v_2^\top x, \dots, v_m^\top x)$. We apply a ReLU activation and a linear classifier with parameter vector θ :

$$f(x) = \sum_{j=1}^m \theta_j (v_j^\top x)_+$$



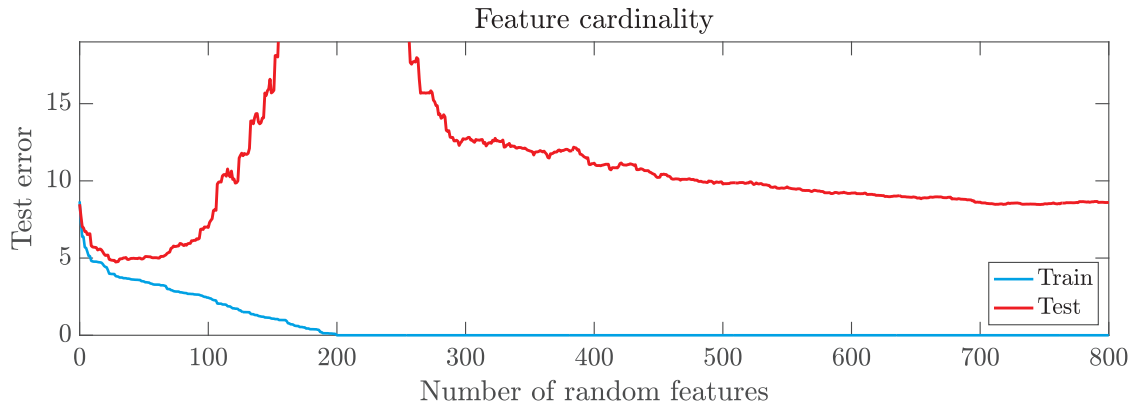
Primer: random feature example (3)

Now we use GD to minimize the squared loss of f by training (only!) the last layer of the network:

$$\min_{\theta \in \mathbb{R}^m} \sum_{i=1}^n \left(y_i - \sum_{j=1}^m \theta_j (v_j^\top x_i)_+ \right)^2$$

We find the following empirical plots of training and test errors:

Primer: random feature example (4)



Primer: random feature example (5)

Observe that the sample size was $n = 200$, so with number m of random features being $m = 200$ we have as many parameters as data points. This is called the interpolation threshold: we can achieve training error 0 with this many parameters (as we can see in the plot). The surprise happens with the test error: while it increases (as expected) when m approaches this threshold from the left, it decreases afterwards!

We will go through the maths of this example in the assignments!

Similar observations can be made with many other models, see below for more examples.

Double descent curve

Radically new insight:

Need to distinguish between **under-parameterized regime** (less parameters than data points, cannot interpolate, “traditional” behavior) and **over-parameterized / interpolation regime**. Here the test risk sometimes decreases dramatically again and sometimes even gets lower than in the traditional regime.

Double descent curve (2)

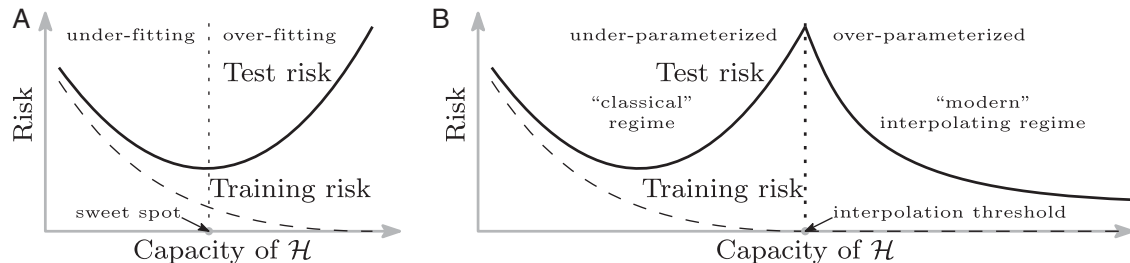


Fig. 1. Curves for training risk (dashed line) and test risk (solid line). (A) The classical U-shaped risk curve arising from the bias–variance trade-off. (B) The double-descent risk curve, which incorporates the U-shaped risk curve (i.e., the “classical” regime) together with the observed behavior from 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

Double descent curve (3)

In their original paper, such behavior has been established empirically for a range of algorithms including neural networks and random forests:

Double descent curve (4)

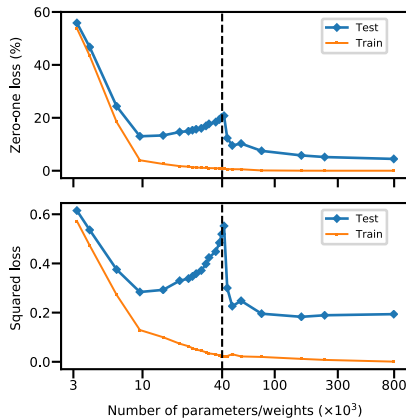


Figure 4: **Double descent risk curve for fully connected neural network on MNIST.** Training and test risks of network with a single layer of H hidden units, learned on a subset of MNIST ($n = 4 \cdot 10^3$, $d = 784$, $K = 10$ classes). The number of parameters is $(d+1) \cdot H + (H+1) \cdot K$. The interpolation threshold (black dotted line) is observed at $n \cdot K$.

Double descent curve (5)

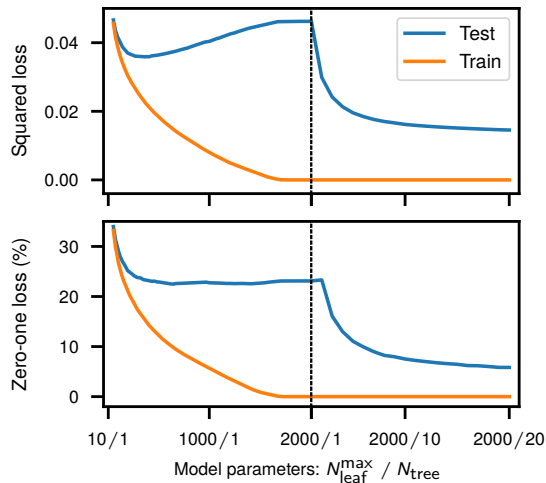


Figure 5: **Double descent risk curve for random forests on MNIST.** The double descent risk curve is observed for random forests with increasing model complexity trained on a subset of MNIST ($n = 10^4$, 10 classes). Its complexity is controlled by the number of trees N_{tree} and the maximum number of leaves allowed for each tree $N_{\text{leaf}}^{\text{max}}$.

Other descent curves

It is not the case that the training/test error curves always show a double descent, many more shapes are possible:

- A single descent:

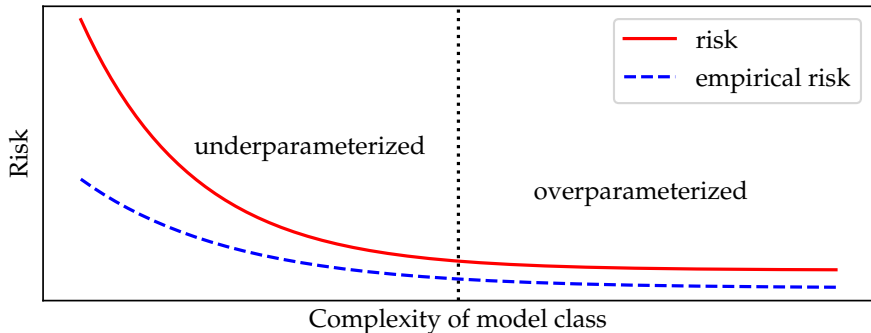
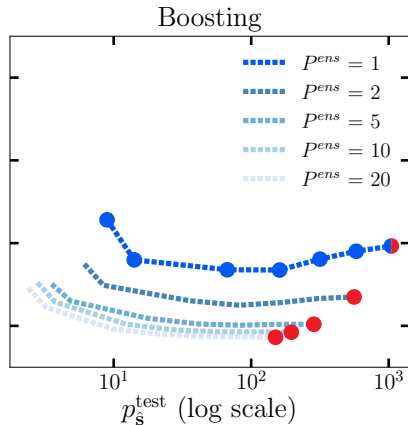
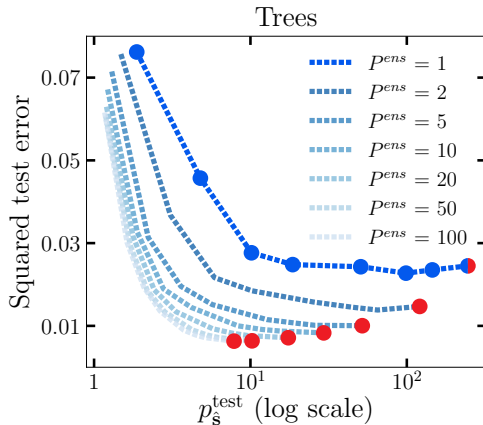


Figure from Hardt book

Other descent curves (2)



Curth et al, U-turn on double descent, NeurIPS 2023

Other descent curves (3)

► Triple descent:

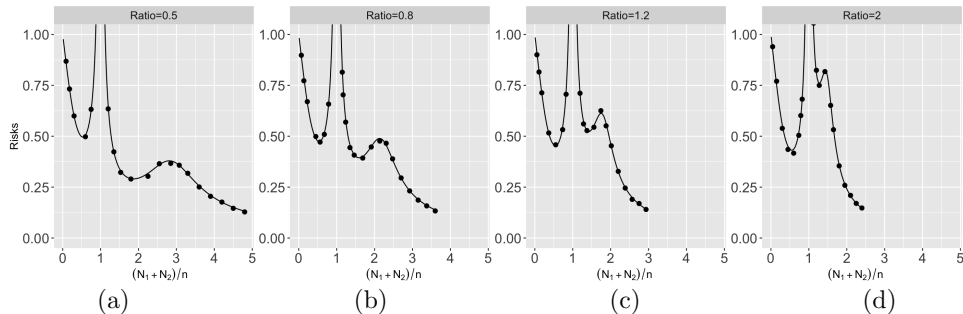
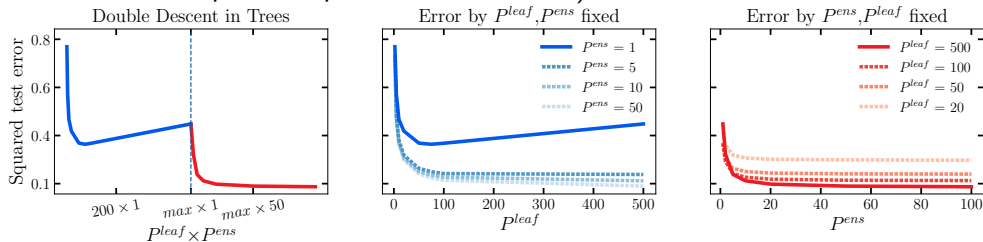


Figure 6: Risk curves of DRFMs with different ratios between random feature dimensions. The plots show both the asymptotic excess risks (curves) and empirical excess risks (dots). From (a) to (d), the ratios N_1/N_2 are 0.5, 0.8, 1.2 and 2, respectively. The activation functions are chosen as $\sigma_1(x) = \text{ELU}(3x)$ and $\sigma_2(x) = \text{ReLU}(x/4)$.

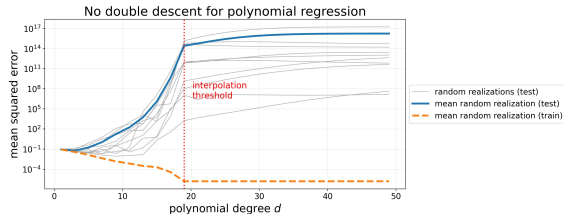
Other descent curves (4)

- It has also been argued repeatedly that the shape of the curve strongly depends on how we count the parameters of the model (Curth et al, NeurIPS 2023). For example, for random forests, the complexity measure in the orig paper is the total number of leafs in the forest (left plot below, x-axis). However, the behavior of the forest changes when we go from one tree (blue part) to several trees (red part), introducing the “kink” in the left plot. Changing how we count parameters changes the plot (middle, right: p^{leaf} = number of leaves per tree; p^{ens} number of trees).



A case where double descent does NOT happen

Just to make sure this is clear: double descent does not automatically happen!!! Here is an example where overfitting beyond the interpolation threshold goes wrong:



The setup:

A case where double descent does NOT happen (2)

The setup:

- ▶ We consider $n = 20$ training points x_i sampled uniformly from $[-1, 1]$, with noisy labels

$$y_i = \frac{1}{1 + 25x_i^2} + \varepsilon_i.$$

- ▶ We fit polynomial regression models of increasing degree d , using the monomial features $1, x, x^2, \dots, x^d$ with least squares loss:

$$\min_{a \in \mathbb{R}^{d+1}} \frac{1}{n} \sum_{i=1}^n \left(y_i - \sum_{k=0}^d a_k x_i^k \right)^2.$$

The interpolation threshold occurs at $d = 19$.

- ▶ The test error is evaluated on a dense grid in $[-1, 1]$. Beyond the interpolation threshold, the high-degree polynomial becomes unstable and the test error grows, rather than exhibiting a second descent.

Take-home

The key take-home message: learning in the over-parameterized regime can work successfully! The exact shape of the curve is not important. What is important that the curve might (but not always does) exhibit smaller test error beyond the interpolation threshold.

But what is the intuition why it can work well?

- ▶ Have seen: empirically, double descent / multiple descent can happen in many over-parameterized models.
- ▶ In some models we can even derive the behavior of the curves analytically.
- ▶ But what is the intuitive reason why it happens? To see this, we will first need to cover two more topics: implicit regularization and benign overfitting.
- ▶ We will then come back to this question.

Implicit regularization by the choice of the optimization algorithm

GD and SGD for linear least squares regression converge to the minimum norm solution

Literature:

Bach book, Sec 12.1.1. (p. 344)

Setup: GD, linear least squares, over-parameterized

We consider linear least squares regression in the over-parameterized regime where $n < d$. Our training data consists of $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$. To make our lives simpler we assume that X has full row rank n (e.g., this is true with probability 1 if our points come from a Gaussian distribution). Under this assumption, $XX^\top \in \mathbb{R}^{n \times n}$ is invertible and hence perfect interpolation is possible: there exists a vector w^* such that $Xw^* = y$.

As learning algorithm we consider the least-squares objective $L(w) = \frac{1}{2} \|Xw - y\|^2$ and run gradient descent initialized at $w_0 = 0$, with the update steps

$$w_{t+1} = w_t - \eta X^\top (Xw_t - y).$$

We choose a constant step size satisfying $0 < \eta < 2/\lambda_{\max}(XX^\top)$ where $\lambda_{\max}$ denotes the largest eigenvalue of the matrix.

Convergence to minimum norm solution

Theorem 51 (Convergence to minimum norm solution)

In the setup of the previous page, the following statements are true:

- (1) The GD iterates w_t converge to $w^* = X^\top (XX^\top)^{-1}y$.
- (2) w^* is the minimum-norm solution: w^* is a solution, and among all the solutions of $Xw = y$ it is the one with minimal Euclidean norm $\|w\|$. It is the unique solution with this property.

A similar result also holds for SGD.

Proof

Step 1: All iterates w_t lie in $S_n := \text{span}\{x_1, \dots, x_n\} = \text{range}(X^\top)$: Proof by induction.

- Obviously true for $w_0 = 0$.
- To treat the general case $t > 0$, observe that the update term $\eta X^\top (Xw_t - y)$ is a linear combination of training points (because no matter what v is, it is true that $X^\top v = \sum_{i=1}^n v_i x_i$). Thus, if $w_t \in S_n$, also $w_{t+1} \in S_n$, so we can express them in the form $w_t = X^\top \alpha_t$ for some $\alpha_t \in \mathbb{R}^n$.

Proof (2)

Step 2: The residuals satisfy $r_{t+1} = (I - \eta XX^\top) r_t$.

Define the residual $r_t := Xw_t - y$. Then we have:

$$\begin{aligned} r_{t+1} &= Xw_{t+1} - y \text{ (plug in GD update rule)} \\ &= X(w_t - \eta X^\top (Xw_t - y)) - y \\ &= Xw_t - \eta XX^\top (Xw_t - y) - y \\ &= (Xw_t - y) - \eta XX^\top (Xw_t - y) \text{ (use def of } r_t) \\ &= r_t - \eta XX^\top r_t \\ &= (I - \eta XX^\top) r_t \end{aligned}$$

Proof (3)

Step 3: Gradient descent is contractive and converges to an interpolating solution.

Since $XX^\top$ is symmetric positive semi-definite and $0 < \eta < 2/\lambda_{\max}(XX^\top)$, all eigenvalues of $I - \eta XX^\top$ have modulus strictly smaller than 1. Hence $I - \eta XX^\top$ is a strict contraction.

Consequently, $r_t \rightarrow 0$ and hence $Xw_t \rightarrow y$.

Step 4: Solution is given by $w^* = X^\top(XX^\top)^{-1}y$:

- ▶ We already know that $Xw_t \rightarrow y$ (by Step 3) and $Xw_t = XX^\top \alpha_t$ for some α_t (by Step 1), and $XX^\top$ is invertible (by assumption).
- ▶ Thus $\alpha_t \rightarrow (XX^\top)^{-1}y$, and thus $w_t = X^\top \alpha_t \rightarrow X^\top(XX^\top)^{-1}y$.

Proof (4)

Step 5: Inside S_n there only exists one unique solution. S_n is an n -dim subspace, and by assumption $\text{rank } X$ is n , so the linear system can have at most one solution in S_n :

if $w = X^\top \alpha \in S_n$ and $Xw = y$, then $XX^\top \alpha = y$. Since $XX^\top$ is invertible, $\alpha = (XX^\top)^{-1}y$ is unique.

Step 6: Minimum norm solution.

We claim that among all interpolating solutions, the unique one in $\text{range}(X^\top)$ is the minimum-norm solution.

- ▶ Let w be any solution: $Xw = y$.
- ▶ Decompose it orthogonally as $w = w_{\parallel} + w_{\perp}$, where $w_{\parallel} \in \text{range}(X^\top)$ and $w_{\perp} \in \ker(X)$.
- ▶ Observe that $Xw = Xw_{\parallel}$ (because by definition $w_{\perp} \in \ker(X)$, thus $Xw_{\perp} = 0$), so $w_{\parallel} \in S_n$ is also a solution to our problem.

Proof (5)

- ▶ In particular, because we already know by Step 5 that the solution in S_n is unique, the part $w_{\parallel}$ is exactly the same for all solutions w , namely $w_{\parallel} = w^*$.
- ▶ Finally, $\|w\|^2 = \|w_{\parallel}\|^2 + \|w_{\perp}\|^2$. Hence the norm is minimized iff $w_{\perp} = 0$, i.e. iff $w = w_{\parallel} = w^*$.



Note: Neither the proof nor the result itself generalizes to the case where we use an arbitrary starting vector different from $w_0 = 0$. The general result is weaker, it states that the solution converges to the interpolating solution that is closest to the initialization.

Discussion

The result shows that GD performs "implicit regularization": among the many solutions that interpolate the data, it always selects the one that has the smallest norm.

In many other scenarios, we would try to achieve the same by explicit regularization. Here it happens automatically, implicitly.

In particular, the choice of the optimization algorithm (GD or SGD) ensures that we not only find a solution that has a small objective value, but also that has other nice properties.

This is not obvious at all, and other procedures for solving the optimization problem might not have this property! (and thus, potentially, might not generalize).

GD for logistic loss classification converges to maximum margin solution

Literature: Bach book, Sec. 12.1.2.

Convergence to max margin solution

Theorem 52 (Implicit bias of GD for logistic regression)

Consider data $(x_i, y_i)_{i=1, \dots, n}$ for binary classification, that is $x_i \in \mathbb{R}^d$, $y_i \in \{\pm 1\}$. Assume the data is linearly separable (so we can "interpolate"). We consider the logistic loss function $L(w) = \sum_{i=1}^n \log(1 + e^{-y_i \langle w, x_i \rangle})$ and run gradient descent from any initialization with sufficiently small, constant step size. Then the GD iterates diverge ($\|w_t\| \rightarrow \infty$), but the direction converges:

$$\frac{w_t}{\|w_t\|} \rightarrow \frac{w_{\text{mm}}}{\|w_{\text{mm}}\|},$$

where w_{mm} is the maximum-margin linear separator:

$$w_{\text{mm}} = \arg \min_w \|w\|^2 \quad \text{s.t.} \quad \forall i : y_i \langle w, x_i \rangle \geq 1$$

Reference: Soudry, Hoffer, Nacson, Gunasekar, Srebro: "The Implicit Bias of Gradient Descent on Separable Data," JMLR 2018.

Convergence to max margin solution (2)

Proof: skipped, see the Bach book if you are interested.

Note that a related result for SGD exists, but one needs stronger assumptions.

Implicit bias of neural networks

Implicit bias on neural networks

There also exist some results describing the implicit bias of neural networks, for example:

- ▶ Wide, shallow networks converge to the minimum RKHS norm interpolant in the corresponding kernel space (Jacot et al., 2018; Arora et al., 2019).
- ▶ Deep linear networks trained with gradient descent converge towards maximum-margin predictors with respect to an architecture-dependent norm (Gunasekar et al., 2018).
- ▶ Homogeneous neural networks trained on separable data often maximize margin under an implicit path-type norm (Lyu, Li, 2019).
- ▶ In matrix factorization and low-rank models, gradient descent implicitly favors low-rank / minimum nuclear norm solutions (Gunasekar et al., 2017).

Implicit bias on neural networks (2)

However:

- ▶ For nonlinear deep networks trained with SGD, the implicit bias is not very well understood and may even depend on optimization noise and initialization (Blanc et al., 2020; Damian et al., 2021).
- ▶ So to a certain degree, the idea of implicit regularization can formally explain many simple cases, but no strong results (yet?) exist for complex architectures as they are often used in practice...

The statistics of deep learning: When does the minimum norm solution generalize? Benign overfitting.

Literature:

Not really treated in text books (not even the Bach one). Here are some of the key original papers. Many of them present intuition as well, but the maths is difficult:

- ▶ Benign overfitting in linear regression Peter Bartlett, Philip M Long Gabor Lugosi, Alexander Tsigler, PNAS 2020
- ▶ Tsigler, A., Bartlett, P. L. (2023). Benign overfitting in ridge regression. Journal of Machine Learning Research, 24(123), 1-76.
- ▶ Hastie, Montanari, Rosset, Tibshirani: Surprises in high-dimensional ridgeless least squares interpolation. Annals of Statistics, 2022
- ▶ Deep learning: a statistical viewpoint (Peter Bartlett, Andrea Montanari, Sasha Rakhlin, 2021).

What is benign overfitting?

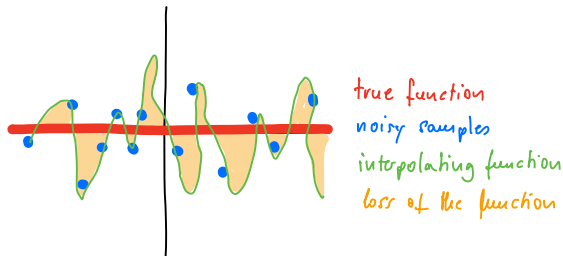
- ▶ Classical statistical wisdom would suggest: interpolating many data points leads to bad overfitting: we have a low training error but a high test error.
- ▶ However, in modern high-dimensional models, this intuition can fail: in DNNs we often fit to training error 0, yet still have a good test performance.
- ▶ Situations in which this happens have been called benign overfitting. This term was coined in the paper by Bartlett, Long, Lugosi, Tsigler: “Benign overfitting in linear regression.”
- ▶ How can a function interpolate noise and still predict well?
- ▶ We will study this question for different setups of linear regression.

Benign overfitting and spiky-smooth structure in a toy setup: Learning the constant 0 function

Toy setup: isotropic Gaussian input, constant 0 output with noise

We first consider a toy setup where the function that we want to learn is constant 0, and all we observe are noisy observations of this function.

“Classical intuition” would lead us to think that if we interpolate the noise, it cannot generalize, and the test error would be large.



The trick is now: while this intuition might be correct for many interpolating solutions, it is wrong if we use a very specific solution: the minimum norm interpolator.

Toy setup: isotropic Gaussian input, constant 0 output with noise (2)

Formal setup:

- ▶ We consider a regression problem with input points $x_1, \dots, x_n \in \mathbb{R}^d$, drawn iid from $N(0, I_d)$. Denote by $X \in \mathbb{R}^{n \times d}$ the data matrix.
- ▶ We assume the true output is equal to 0 everywhere (the Bayes predictor is the constant 0 function).
- ▶ We assume that we observe the true label 0 with iid additive noise, that is $y_i \sim N(0, \sigma^2)$.
- ▶ We consider linear prediction in the over-parameterized regime where $n + 1 < d$.

Benign overfitting in toy setup

Theorem 53 (Benign overfitting in simple setup)

Consider the minimum-norm interpolator $\hat{w} = X^\top (XX^\top)^{-1}y$ for the setup of the previous slide. Its expected true risk with respect to the least squares loss function is given by

$$\mathbb{E}_{X,y,x}((x^\top \hat{w} - 0)^2) = \sigma^2 \frac{n}{d - n - 1}.$$

Here $\mathbb{E}_{X,y,x}$ denotes the expectation with respect to the training points X , their labels y and the test point x .

In particular, if the dimension d is very large compared to n , then the excess prediction risk tends to 0, although $\hat{w}$ interpolates pure noise!!!

Let's first look at the proof, and then at some geometric intuition of what is going on.

Proof sketch (Benign overfitting in toy setup)

Proof sketch.

- ▶ The matrix X has size $n \times d$, with $d > n$. One can prove that for Gaussian random variables, the $n \times n$ -matrix $XX^\top$ is invertible almost surely. Therefore the least squares solution $\hat{w} = X^\top(XX^\top)^{-1}y$ perfectly interpolates the training data:

$$X\hat{w} = XX^\top(XX^\top)^{-1}y = y.$$

- ▶ For a test point $x \sim N(0, I_d)$, the true expected squared error of the prediction $\hat{y} = x^t \hat{w}$ is given as

$$E_x((\hat{y} - 0)^2) = E_x((x^t \hat{w})^2) = E_x(\hat{w}^t x x^t \hat{w}) = \hat{w}^t E_x(x x^t) \hat{w} = \hat{w}^t I_d \hat{w} = \|\hat{w}\|^2$$

- ▶ We can compute this norm by plugging in the expression of $\hat{w}$:

$$\|\hat{w}\|^2 = \langle \hat{w}, \hat{w} \rangle = y^\top (XX^\top)^{-1} XX^\top (XX^\top)^{-1} y = y^\top (XX^\top)^{-1} y.$$

Proof sketch (Benign overfitting in toy setup) (2)

- We now want to compute the expected value of this quantity: $E_{X,y} \|\hat{w}\|^2$. We do it in two steps: first we treat X as fixed and compute the expectation wrt y conditioned on the fixed X , and then we compute the expectation wrt X .

On the way we exploit two facts:

- For any fixed matrix A and the random vector y we have $E_y(y^\top A y) = \text{tr}(A E_y(y y^\top))$:

$$\begin{aligned} E_y(y^\top A y) &= (\text{because for a real number } a, \text{ we have } a = \text{tr}(a)) \\ &= E_y(\text{tr}(y^\top A y)) \\ &= E_y(\text{tr}(A y y^\top)) \\ &= (\text{linearity of } E \text{ and } \text{tr}) \\ &= \text{tr}(E_y(A y y^\top)) \\ &= \text{tr}(A E_y(y y^\top)) \end{aligned}$$

Proof sketch (Benign overfitting in toy setup) (3)

So in our case,

$$E_y(y^\top (XX^\top)^{-1} y) = \text{tr} \left((XX^\top)^{-1} E(yy^\top) \right)$$

- ▶ $E_y(yy^\top) = \sigma^2 I_n$ because y are the Gaussian noise variables
- ▶ Now we need to take the expectation with respect to X . We exploit that $XX^\top$ is a Wishart matrix with d degrees of freedom, and for such a matrix and $d > n$ one can prove that $E((XX^\top)^{-1}) = I_n / (d - n - 1)$, where I_n denotes the identity matrix.
- ▶ Taken together, one can conclude that

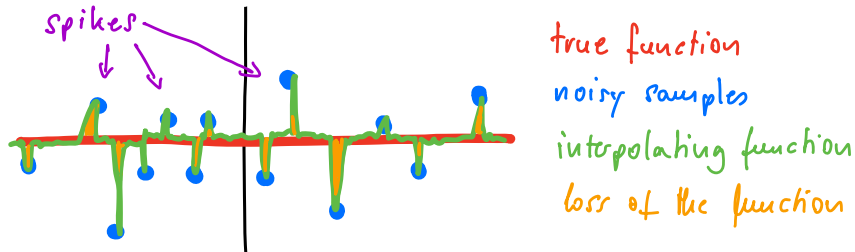
$$E_{X,y} \|\hat{w}\|^2 = \text{tr} \left(\frac{1}{d - n - 1} I_n \sigma^2 I_n \right) = \sigma^2 \frac{1}{d - n - 1} \text{tr}(I_n) = \sigma^2 \frac{n}{d - n - 1}.$$



Original sources: Hastie, Montanari, Rosset, Tibshirani (2019); Bartlett, Long, Lugosi, Tsigler (2020).

What happens geometrically? Spiky-smooth structure of the solution

While the theorem is mathematically correct, it is still difficult to grasp what happens. Here is an illustration:



Large “spikes” interpolate the noisy labels y_i ; in high dimensions, these regions **have vanishing volume**. Outside the spikes, the function approximates the true function (the 0-function in this case).

The maths of the spiky-smooth behavior

Step 1: Form of the prediction function

We have already seen above that the solution vector lies in the span of our data points, and thus our prediction function has the form $f(x) = \sum_{i=1}^n \alpha_i \langle x, x_i \rangle$

Step 2: Function interpolates

By assumption, the function interpolates the training points. Here it takes potentially large values.

The maths of the spiky-smooth behavior (2)

Step 3: The Spike We want to look at the region around a training point where the function still has large value. We call this a “spike”:

Define the “spike” $S_i(t)$ of training point x_i with parameter t as **the set**

$$S_i(t) = \{x \in \mathbb{R}^d \mid \left\langle \frac{x}{\|x\|}, \frac{x_i}{\|x_i\|} \right\rangle > t\}$$



Observe that the **red term** describes a projection of the vector x on the vector x_i (or vice versa).

The maths of the spiky-smooth behavior (3)

Step 4: Concentration of random projections:

We want to find out how likely it is that a random test point x is inside the spike of a random training point x_i . In other words: what is the likelihood that **the projection of one random point on another random point** is large?

In high-dim spaces, such random projections are highly concentrated: when we draw the two points randomly, the likelihood that the random projection takes a large value (larger than t) is very small. Let's look at this:

Consider u, v drawn from the unit sphere in $\mathbb{R}^d$ according to the uniform distribution. Then one can prove the following concentration inequality about their scalar product:

$$P(\langle u, v \rangle > t) \leq 2 \exp(-cdt^2)$$

where c is a constant, d is the dimension.

The maths of the spiky-smooth behavior (4)

Step 5: Volume of spike is low:

The probability that a test point x ends up in any of the spikes is then, according to the union bound:

$$P(x \in \cup_{i=1}^n S_i(t)) \leq 2n \exp(-c d t^2)$$

This explains why the spikes ultimately do not hurt consistency: if $n/d \rightarrow 0$, the error bound in our theorem goes to 0.

The maths of the spiky-smooth behavior (5)

Step 6: Outside the spikes the function approximates the 0 function.

- ▶ In high dimensions, the data points are nearly orthogonal. The learned function must take large values at the training points in order to interpolate. This is possible because each data point essentially “opens up” a new dimension in our data space, and we have enough parameters to memorize the point in this dimension.
- ▶ As we have seen above, a random test point, with high probability, is not aligned with any of the training points, so it is outside of the spikes.
- ▶ Now we exploit that we consider the minimum norm solution: The overall function has norm as small as possible. In particular, away from the training points (in the directions where we have no training points), the solution will have small norm, so the prediction is close to zero. Again this is possible because in high dimensions, there are enough parameters to make the function behave “close to zero” in directions that are nearly orthogonal to the training points.

Not so benign: Linear model with isotropic Gaussian inputs

Literature: Bach, Sec.12.2.3;

Literature:

Bartlett, Long, Lugosi, Tsigler: "Benign overfitting in linear regression". PNAS, 2020.

See also

Bartlett, Montanari, Rakhlin: "Deep learning, a statistical viewpoint". Acta Numerica, 2021.

Setup: linear model, isotropic Gaussian sampling

We are now going to investigate whether similar results as above hold in a slightly more general setting: linear regression with Gaussian input variables. We will see: partially yes, but there are some caveats!

- ▶ Input distribution: $x_1, \dots, x_n \sim N(0, I_d)$ in $\mathbb{R}^d$, iid
- ▶ Response variables:

$$y_i = x_i^T \theta_* + \varepsilon_i$$

where $\theta_* \in \mathbb{R}^d$ "true signal", $\varepsilon_i \sim N(0, \sigma^2)$, iid (so the overall noise is $\varepsilon \sim N(0, \sigma^2 I_n)$)

- ▶ As usual we denote by $X \in \mathbb{R}^{n \times d}$ the design matrix with data points as rows, and by $\hat{\Sigma} = \frac{1}{n} X^T X \in \mathbb{R}^{d \times d}$ the empirical covariance matrix.
- ▶ Recall the definition of the excess risk at test point x , defined as

$$R(\hat{\theta}) = E_x((\hat{\theta}^T x - \theta_*^T x)^2)$$

The expectation is over the test point x . We will then consider the expectation with respect to the training data of the excess risk: $E_{X,y} R(\hat{\theta})$.

Main result: Expected excess risk in the over-parametrized regime

Theorem 54 (Expected excess risk, linear model, Gaussian data)

In the setup outlined in the previous slide, the expected excess risk is given as follows:

- In the under-parameterized regime $n > d + 1$, we have

$$E_{X,y}(R(\hat{\theta})) = \underbrace{\frac{\sigma^2 d}{n - d - 1}}_{\text{variance}}$$

- In the over-parameterized regime $d > n + 1$ we have:

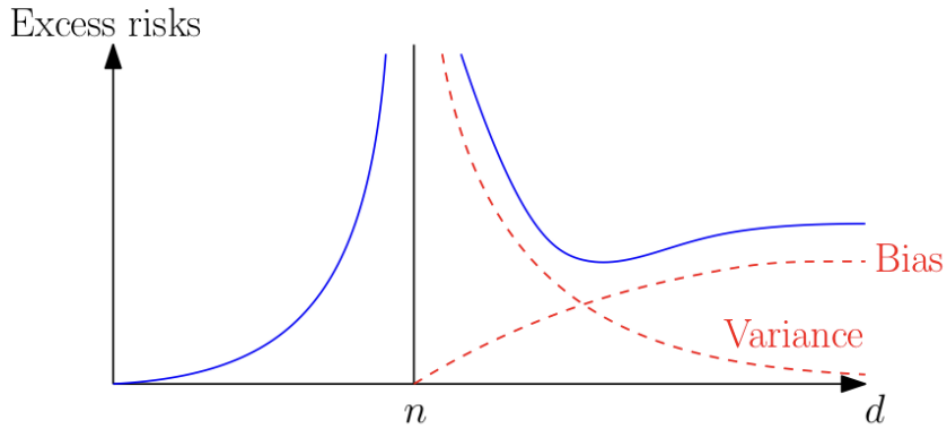
$$E_{X,y}(R(\hat{\theta})) = \underbrace{\frac{\sigma^2 n}{d - n - 1}}_{\text{variance}} + \underbrace{\|\theta_*\|_2^2 \frac{d - n}{d}}_{\text{bias}}$$

Main result: Expected excess risk in the over-parametrized regime (2)

In particular:

- ▶ The variance terms in both regimes are nice (and look quite similar, just the roles of d and n are switched), and they are small in both regimes.
- ▶ In the under-parameterized setup, the estimate is unbiased.
- ▶ The behavior of the bias in the over-parameterized regime does not “explode” (which is good), but it is not satisfactory either: there is a bias that remains large (unless we make strong assumptions, see later).
- ▶ Here we would NOT say that benign overfitting takes place!!!
- ▶ In particular, learning works better in the under-parameterized regime.

Main result: Expected excess risk in the over-parametrized regime (3)



Proof step 1: excess risk

We first derive a simpler expression for the excess risk:

$$\begin{aligned} R(\hat{\theta}) &= E_x((\hat{\theta}^\top x - \theta_*^\top x)^2) \\ &= E_x((\hat{\theta}^\top x - \theta_*^\top x)^\top (\hat{\theta}^\top x - \theta_*^\top x)) \\ &= (\hat{\theta} - \theta_*)^\top \underbrace{E(xx^\top)}_{=:\Sigma} (\hat{\theta} - \theta_*) \\ &= (\hat{\theta} - \theta_*)^\top \Sigma (\hat{\theta} - \theta_*) \\ &=: \|\hat{\theta} - \theta_*\|_\Sigma^2 \end{aligned}$$

Proof step 2: Bias-Variance- decomposition

Now additionally consider the expectation over the training points. In this setting, a version of the bias-variance-decomposition is given as

$$\begin{aligned} E_{X,y} R(\hat{\theta}) &= E_{X,y} \|\hat{\theta} - \theta_*\|_{\Sigma}^2 \\ &= E_{X,\varepsilon} \|\hat{\theta} - \theta_*\|_{\Sigma}^2 \\ &= \underbrace{E_X(\|E_{\varepsilon}(\hat{\theta}|X) - \theta_*\|_{\Sigma}^2)}_{\text{bias term}} + \underbrace{E_{X,\varepsilon}(\|\hat{\theta} - E_{\varepsilon}(\hat{\theta}|X)\|_{\Sigma}^2)}_{\text{variance term}} \end{aligned}$$

(We can replace the expectation wrt y by the one wrt to the noise variable ε because the response variable y is given as deterministic function plus noise;).

Proof step 3: Analysis in the under-parameterized regime $n \geq d + 2$

The minimum-norm interpolator is simply the solution of ordinary least squares:

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

- We have already seen the results regarding the expected excess risk:
In the random design setting, the expected (wrt X, y) excess risk is given by

$$E_{X,y}(R(\hat{\theta})) = \frac{\sigma^2}{n} E_{X,y} \left(\text{tr} \left(\underbrace{\Sigma}_{I_d} \underbrace{\hat{\Sigma}^{-1}}_{\left(\frac{1}{n}(X^T X)\right)^{-1}} \right) \right) = \sigma^2 E_X(\text{tr}((X^T X)^{-1}))$$

Proof step 3: Analysis in the under-parameterized regime $n \geq d + 2$ (2)

- The matrix $X^T X$ follows a Wishart distribution with n degrees of freedom. It is known that for $n \geq d + 2$ we have

$$E((X^T X)^{-1}) = \frac{1}{n - d - 1} I_d,$$

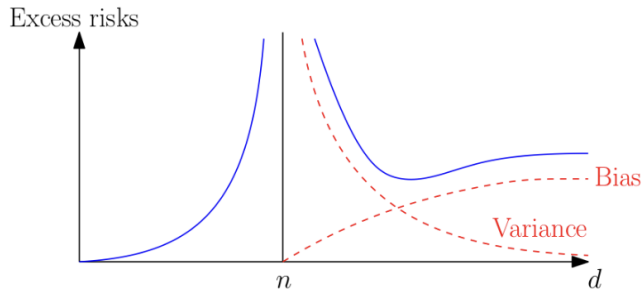
hence its expected trace is $d/(n - d - 1)$.

- So overall the excess risk in the under-parameterized regime is given as

$$E(R(\hat{\theta})) = \sigma^2 \frac{d}{n - d - 1}$$

Proof step 3: Analysis in the under-parameterized regime $n \geq d + 2$ (3)

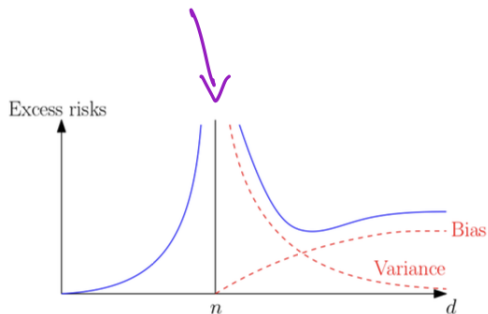
Observe that for $d \ll n$, the excess risk is small:



under-parameterized
regime

Proof step 4: What happens close to the interpolation threshold, $n \approx d$

- ▶ For $n = d$ and $n = d + 1$ it holds that $E(\text{tr}((X^T X)^{-1})) = \infty$ (it is a property of Wishart matrices when the degree of freedom is d or $d + 1$).
- ▶ Hence, the variance explodes.
- ▶ This is why the double-descent-plot diverges close to the interpolation threshold $d = n$.



Proof step 5: Analysis in the over-parametrized regime $n < d$

Before we get started with the maths in the over-parameterized settings, let us build some intuition:

- ▶ We have a signal θ_* in a d -dimensional space. But in the setting $n \ll d$ we have much less data points than dimensions.
- ▶ So even if each data point would serve to perfectly identify one of the d parameters of θ_* , we will in general NOT have enough data to observe all the d different directions of θ_* .
- ▶ So in general, we will NOT converge to a small risk!
- ▶ To achieve a small risk, we need additional assumptions. These assumptions would need to capture that the true signal θ_* can in fact be described by at most of the order n many parameters (not d many).

Proof step 5: Analysis in the over-parametrized regime $n < d$ (2)

Let us analyze the situation formally:

- We consider the bias-variance-decomposition:

$$E_{X,\varepsilon}(R(\hat{\theta})) = \underbrace{E_{X,\varepsilon}(\|E_{\varepsilon}(\hat{\theta}|X) - \theta_*\|_{\Sigma}^2)}_{\text{bias}} + \underbrace{E_{X,\varepsilon}(\|\hat{\theta} - E_{\varepsilon}(\hat{\theta}|X)\|_{\Sigma}^2)}_{\text{variance}}$$

- We have already seen that in this scenario, GD/SGD converges to the minimum norm interpolator, which is given by

$$\hat{\theta} = X^T(XX^T)^{-1}y$$

Proof step 6: Variance term in the over-parameterized regime

To evaluate the green part in the variance term, first observe that

$$\begin{aligned}\hat{\theta} &= X^T (XX^T)^{-1} y \\ &= X^T (XX^T)^{-1} (X\theta_* + \varepsilon) \\ &= X^T (XX^T)^{-1} X\theta_* + X^T (XX^T)^{-1} \varepsilon.\end{aligned}$$

And because of the independence of the noise, $E_\varepsilon(\varepsilon \mid X) = 0$. Thus

$$E_\varepsilon(\hat{\theta} \mid X) = X^T (XX^T)^{-1} X\theta_*.$$

So the green term is

$$\hat{\theta} - E_\varepsilon(\hat{\theta} \mid X) = X^T (XX^T)^{-1} \varepsilon.$$

Proof step 6: Variance term in the over-parameterized regime (2)

Now we compute the full variance term:

$$\begin{aligned}
 E_{X,\varepsilon} \left(\|\hat{\theta} - E_{\varepsilon}(\hat{\theta} \mid X)\|_{\Sigma}^2 \right) &= E_{X,\varepsilon} (\|X^T (XX^T)^{-1} \varepsilon\|_{\Sigma}^2) \\
 &= E_{X,\varepsilon} (\varepsilon^T (XX^T)^{-1} X \Sigma X^T (XX^T)^{-1} \varepsilon) \\
 &\quad \text{(exploit } \Sigma = I) \\
 &= E_{X,\varepsilon} (\varepsilon^T (XX^T)^{-1} XX^T (XX^T)^{-1} \varepsilon) \\
 &\quad \text{(green terms cancel)} \\
 &= E_{X,\varepsilon} (\varepsilon^T (XX^T)^{-1} \varepsilon) \\
 &\quad \text{(trace argument as before; exploit that conditional on } X \\
 &\quad \text{we have that } E_{\varepsilon}(\varepsilon \varepsilon^T) = \sigma^2 I_n, \text{ which is independent of } X) \\
 &= \sigma^2 E_X (\text{tr} (XX^T)^{-1}) \\
 &= \sigma^2 \frac{n}{d - n - 1}
 \end{aligned}$$

Proof step 6: Variance term in the over-parameterized regime (3)

The variance term behaves nice in the over-parameterized regime: it goes to 0 as $d \rightarrow \infty$ with $n/d \rightarrow 0$.

(Observe that it is the same variance term as we obtained in the toy setup with the constant-0 function above).

Proof step 7: Bias term in the over-parameterized regime

$$\begin{aligned}
 \underbrace{E_{X,\varepsilon}(\|E_\varepsilon(\hat{\theta} \mid X) - \theta_*\|_\Sigma^2)}_{\text{bias term}} &= E_X(\|E_\varepsilon(\hat{\theta} \mid X) - \theta_*\|_\Sigma^2) \\
 &= E_X\left(\left\|E_\varepsilon\left(X^T(XX^T)^{-1}(X\theta_* + \varepsilon) \mid X\right) - \theta_*\right\|_\Sigma^2\right) \\
 &= E_X\left(\left\|X^T(XX^T)^{-1}X\theta_* + X^T(XX^T)^{-1}E_\varepsilon(\varepsilon \mid X) - \theta_*\right\|_\Sigma^2\right) \\
 &\quad \text{the green term is 0} \\
 &= E_X(\|X^T(XX^T)^{-1}X\theta_* - \theta_*\|_\Sigma^2) \\
 &\quad \text{exploit that } \Sigma = I_d \\
 &= E_X(\theta_*^T(I - X^T(XX^T)^{-1}X)\theta_*)
 \end{aligned}$$

Proof step 7: Bias term in the over-parameterized regime (2)

For the blue term: The matrix $X^T(XX^T)^{-1}X \in \mathbb{R}^{d \times d}$ is a projection matrix on a random subspace of $\dim n < d$:

- ▶ A projection is a linear mapping that satisfies $P^2 = P$. It is easy to see that this holds here.
- ▶ One can also see that it maps every data point (row of X) to itself, and every vector of $\text{range}(X)$ is being mapped to itself.
- ▶ For every vector $x \in (\text{range}(X))^\perp = \ker(X)$ one can see that it is mapped to 0.

Proof step 7: Bias term in the over-parameterized regime (3)

Because our data comes from $N(0, I)$, the random subspace is uniformly distributed. So in the computation of the expected value we can replace θ_* with $\|\theta_*\| \cdot e_j$ where e_j is the j -th unit vector (formally, we rotate the data to achieve this). Thus we get:

$$\begin{aligned} E(\theta_*^\top X^\top (XX^\top)^{-1} X \theta_*) &\leadsto \text{replace } \theta_* \text{ with } \|\theta_*\| e_j \text{ for some } j \\ &= \|\theta_*\|_2^2 \cdot E(e_j^\top X^\top (XX^\top)^{-1} X e_j) \\ &\leadsto \text{even do it for all } j \text{ and build the rescaled sum} \\ &= \|\theta_*\|_2^2 \cdot E\left(\frac{1}{d} \sum_{j=1}^d e_j^\top X^\top (XX^\top)^{-1} X e_j\right) \\ &= \|\theta_*\|_2^2 \cdot \frac{1}{d} \sum_{j=1}^d E(e_j^\top X^\top (XX^\top)^{-1} X e_j) \end{aligned}$$

Proof step 7: Bias term in the over-parameterized regime (4)

$$\begin{aligned} &\leadsto \text{exploit } \sum_{j=1}^d e_j^\top A e_j = \text{tr}(A) \\ &= \|\theta_*\|_2^2 \cdot \frac{1}{d} E(\text{tr}(X^\top (XX^\top)^{-1} X)) = \\ &= \|\theta_*\|_2^2 \cdot \frac{n}{d} \end{aligned}$$

For the last equality, we exploit again that the matrix $X^\top (XX^\top)^{-1} X$ is the projection that takes any vector in $\mathbb{R}^d$ and projects it on the n -dim subspace $\text{span } x_1, \dots, x_n = \text{range}(X^\top)$. Its trace is n .

Or alternatively, you can use the cyclicity of the trace directly: $\text{tr}(ABC) = \text{tr}(CAB)$.

Proof step 7: Bias term in the over-parameterized regime (5)

So all in all, the bias term gives:

$$\begin{aligned} E_X(\theta_*^T (I - X^T (XX^T)^{-1} X) \theta_*) &= E_X(\theta_*^T \theta_*) - E(\theta_*^T (X^T (XX^T)^{-1} X) \theta_*) \\ &= \|\theta_*\|^2 - \|\theta_*\|^2 \frac{n}{d} \\ &= \|\theta_*\|^2 \cdot \frac{d - n}{d} \end{aligned}$$

Proof step 7: Bias term in the over-parameterized regime (6)

Let us discuss what this bias term means:

If $\|\theta_*\| = 0$, the bias is 0. This is great, and it corresponds to our toy example that we have seen before. We get benign overfitting.

If $\|\theta_*\| \neq 0$, the situation is bad:

- ▶ because we are in the over-parameterized regime $n \ll d$, the term $(d - n)/d \approx \theta(1)$. It does not vanish, there is some bias that remains. We cannot recover the true underlying model!
- ▶ This is NOT a regime of benign overfitting in general! We interpolate, but there is a large bias that does not disappear!
- ▶ Intuitively, the reason is that with n points we can only capture n/d -fraction of the signal, the rest "gets lost"! (unless we make strong assumptions, see below).



Benign overfitting: Linear model with aligned covariance structure

Literature: Deep learning, a statistical viewpoint (Bartlett, Montanari, Rakhlin, Acta Numerica 2021)

More general setup: linear model, arbitrary sampling

- ▶ Assume input points x_i come from some distribution on $\mathbb{R}^d$ with $E(x_i) = 0$.
- ▶ Denote the true covariance matrix of the input distribution by $\Sigma = E(xx^T)$.
- ▶ Consider the eigenvalues of the covariance matrix,

$$\Sigma = \text{diag}(\lambda_1, \dots, \lambda_d), \quad \lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$$

- ▶ W.l.o.g. we now assume that these eigenvectors are the unit vectors $e_1, \dots, e_d$, in the ordering that corresponds to the eigenvalues (e_1 is the eigenvector corresponding to the largest eigenvalue, etc).
- ▶ Assume the true regression function is linear:

$$f^*(x) = \langle \theta_*, x \rangle \quad \text{for some } \theta_* \in \mathbb{R}^d.$$

- ▶ We consider the over-parameterized regime $d > n$.

Need assumptions

- ▶ Observe that in this setting, there might not enough data to learn all the model parameters:
 - ▶ We get n scalar observations $y_1, \dots, y_n$ from the n input points
 - ▶ We need to learn all the d parameters of θ_*
 - ▶ We assume $n \ll d$.
 - ▶ Writing it up as a linear system we have d unknowns but only n equations.
- ▶ Thus, learning will only be possible under additional assumptions.
- ▶ First, we need **assumptions on the “signal”**: we must be able to “express it” with much less than d “numbers”, so that if we see n points, we have enough information to estimate the “numbers”.
- ▶ Second, we need **assumptions on the sampling** mechanism: we need to make sure that the variance of the sampling distribution is aligned with the signal. See next pages for intuition.

Need assumptions (2)

Assumptions on the signal:

Want to estimate the vector $\theta_* = (\theta_{*1}, \dots, \theta_{*d})^t$.

Possible assumptions that might work:

- ▶ The vector is (approximately) sparse and only contains few (not more than n) non-zero components.
- ▶ The vector might be approximately sparse if expressed in a different basis (for example, the basis given by the principal components of the data).
- ▶ The data lives on some low-dimensional manifold
- ▶ etc

Need assumptions (3)

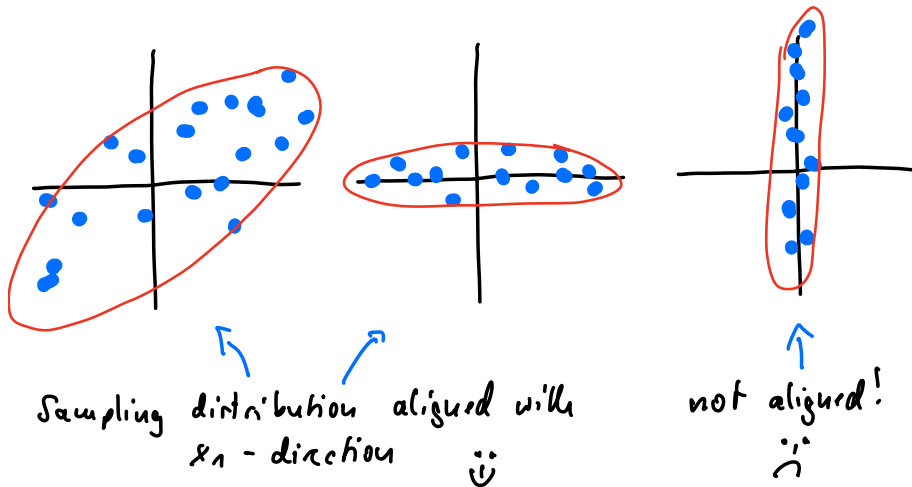
Assumptions on alignment:

Even if the signal is inherently low-dimensional, we need to make sure that we are able to observe the signal in all important directions.

Example:

- ▶ Two-dimensional setup, true linear parameter vector $\theta_* = (1, 0)^t$, so for $x = (x_1, x_2)^t \in \mathbb{R}^2$ the true function is $f(x) = \theta_*^t x = x_1$.
- ▶ Observing this signal will only be possible if we see enough data points with different x_1 -components.
- ▶ We express this by saying that the covariance of the sampling distribution in direction of e_1 must be reasonably large.
- ▶ See illustration on next page.

Need assumptions (4)



Need assumptions (5)

How this will show up in the technical part:

Main results, informal

- ▶ We work in the basis given by the covariance matrix (see setup slide), and wlog we assume that the eigenvectors are $e_1, \dots, e_d$ with eigenvalues are ordered such that $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_d$.
- ▶ **We now split the space in two subspaces**, the first one spanned by eigenvectors $e_1, \dots, e_k$ and the second one spanned by $e_{k+1}, \dots, e_d$ (for some choice of k).
- ▶ The idea of the analysis is to use the first group for the prediction part (the function we want to learn), while the second group takes care of interpolation (the spikes that are necessary to interpolate).
- ▶ Specifically, we choose this k by

$$k = \min \left\{ \ell \mid \frac{\sum_{i \geq \ell} \lambda_i}{\lambda_{\ell+1}} \geq \text{const} \cdot n \right\}$$

Main results, informal (2)

- ▶ The fraction compares the total variance in the tail starting at ℓ with largest individual tail eigenvalue λ_ℓ . If this term is large (at least of order n), then the tail variance is spread over many directions, rather than dominated by one or a few eigenvectors.
- ▶ We choose the smallest such k that is possible: we figure out how many leading eigenvalues we need to remove before the tail becomes “well-conditioned”.
- ▶ In particular, the tail is then also “heavy enough” to allow interpolation.
- ▶ The exact form of this term is chosen such that the analysis goes through ...
- ▶ With the help of this construction one can prove the following theorem:

Main results, informal (3)

Theorem 55 (Expected excess risk of minimum norm interpolator, general case)

In the setup outlined above, let $\hat{\theta} = X^T(XX^T)^{-1}y$ be the minimum-norm interpolator. Choose $k < d$ as on the previous slide, such that the tail is not dominated by only a few eigenvalues. Then, with high probability over the choice of the design matrix X , the expected excess risk can be bounded by

$$E_{\varepsilon} R(\hat{\theta}) \lesssim \underbrace{\sum_{i=1}^d \lambda_i \theta_{*i}^2 \left(\frac{\frac{1}{n} \sum_{j>k} \lambda_j}{\lambda_i + \frac{1}{n} \sum_{j>k} \lambda_j} \right)^2}_{\text{bias: loss from directions not learned}} + \underbrace{\sigma^2 \left(\frac{k}{n} + n \frac{\sum_{i>k} \lambda_i^2}{(\sum_{i>k} \lambda_i)^2} \right)}_{\text{variance: cost of fitting the noise}}.$$

The proof of this theorem is quite difficult, we skip it. You cannot find it in a text book, but in the papers by Bartlett et al.

Let us try to understand what the theorem means.

Variance term, intuition

The variance term in the theorem is this one:

$$\sigma^2 \left(\underbrace{\frac{k}{n}}_{\text{variance in the signal itself}} + \underbrace{n \frac{\sum_{i>k} \lambda_i^2}{(\sum_{i>k} \lambda_i)^2}}_{\text{tail weight from interpolation part}} \right)$$

This term depends on the input covariance and the noise σ , but not on the true direction θ_* .

This term is small if we have many small tail eigenvalues, and large if we have few dominant tail eigenvalues. (See next slide):

Variance term, intuition (2)

To get a feeling for the behavior of the variance term:

- As one extreme case, assume $\lambda_{k+1} = \dots = \lambda_d = a$. Then

$$n \frac{\sum_{i>k} \lambda_i^2}{(\sum_{i>k} \lambda_i)^2} = n \frac{(d-k)a^2}{((d-k)a)^2} = \frac{n}{d-k}$$

For small k , and small n and large d , this is small. **Good.**

- The other way round, assume $\lambda_{k+1} = a$, and all others are 0. Then

$$n \frac{\sum_{i>k} \lambda_i^2}{(\sum_{i>k} \lambda_i)^2} = n \frac{a^2}{a^2} = n$$

which does not vanish. **Bad.**

Variance term, intuition (3)

My personal intuition for this behavior:

- ▶ If we have many small terms: we use the terms to memorize the noise, but we also use the terms to achieve a small norm of our solution (minimum norm interpolator). Because we have so many possibilities to interpolate, both goals can be achieved in a satisfactory way. In particular, the memorization part only contains small spikes and otherwise fits the 0 function and doesn't hurt generalization. The resulting norm of the minimum norm interpolator is small.
- ▶ If we have only few large terms in the tail, then these terms will be used to memorize the noise, but nothing else can be achieved. In particular, while we always find a minimum norm interpolator, this norm would be much higher than in the other regime.

Bias term, intuition

The bias term in the theorem is given as

$$\sum_{i=1}^d \lambda_i \cdot (\theta_{*i})^2 \left(\frac{\frac{1}{n} \sum_{j>k} \lambda_j}{\lambda_i + \frac{1}{n} \sum_{j>k} \lambda_j} \right)^2$$

Bias term, intuition (2)

- ▶ Case where the signal is well aligned with the top eigenvectors:
 - ▶ Assume that the covariance mainly lives in the first dimension: $\lambda_1 \gg \sum_{j>k} \lambda_j$ and $\sum_{j>k} \lambda_j$ is small.
 - ▶ Assume the signal is aligned perfectly with the first eigenvector, that is $\theta_{*1} = 1$ and for all $i > 1$, $\theta_{*i} = 0$.
 - ▶ Then in the sum, all terms except the first one disappear, and the first one is “small”.
So the overall bias is small. Good.
- ▶ Case where the signal is well aligned with the small tail eigenvalues (that is, $\theta_{1*} = \theta_{2*} = \dots = \theta_{k*} = 0$ and the remaining θ_j are large).
 - ▶ Then the first k terms in the sum disappear, and the remaining ones are large (because λ_j is small).
 - ▶ So the bias gets worse. Bad!

Combining bias and variance terms

Good situation:

- ▶ Covariance spectrum contains few large eigenvalues $\lambda_1 \dots \lambda_k$ and then many tiny remaining eigenvalues.
- ▶ The signal θ_* is aligned with the top eigenvalues.

Then:

- ▶ Then the true signal and the data essentially live in (the same) k -dimensional subspace.
- ▶ Then both bias and variance are small.
- ▶ The top k directions can be used to represent the signal, similarly to the under-parameterized regime.
- ▶ The remaining $d - k$ directions serve to interpolate, but don't accumulate bias.

Benign overfitting!

Combining bias and variance terms (2)

Several possibilities for bad situations:

- ▶ If the tail spectrum has too much weight, bias accumulates. The interpolator only catches a fraction of n/d signal, and $1 - n/d$ noise. The noise part (from interpolation) dominates, and the variance term does not go to 0. (This was the situation that we had already discussed earlier where we sampled the data from $N(0, I)$).
- ▶ If the tail spectrum contains few dominant eigenvalues, the variance does not go to 0.
- ▶ If the signal θ_* is not well-aligned with the dominating eigenvalues, then the bias term does not go to 0. Intuitively, we don't have enough data to learn the signal.

Different regimes

On the following slides we illustrate the behavior in a toy example:

- ▶ $n = 60$ points sampled uniformly from $[0, 2\pi]$,
- ▶ We represent these points in a Fourier basis with the basis functions $\varphi_j(x) = s_j \cos(jx)$, in dimension $d = 2000$ (we will choose different values for s_j later).
- ▶ One can see that in this setup, the eigenvectors of the true covariance matrix are already the basis vectors, hence the true covariance matrix is a diagonal matrix. Its diagonal entries / eigenvalues are $\lambda_i = s_i^2/2$.
- ▶ As the true underlying function we choose $\cos(x)$ (the first basis function). This corresponds to a linear model in the Fourier space with true parameters $\theta_* = (1, 0, \dots, 0)$.
- ▶ Observe that in this setup, the signal is always perfectly aligned with the first eigenvector.
- ▶ We sample the noisy output values $y_i = \cos(x_i) + \varepsilon_i$ with $\varepsilon_i \sim \mathcal{N}(0, 0.01)$
- ▶ Then we look at the minimum-norm interpolating solution (minimal least squares loss).

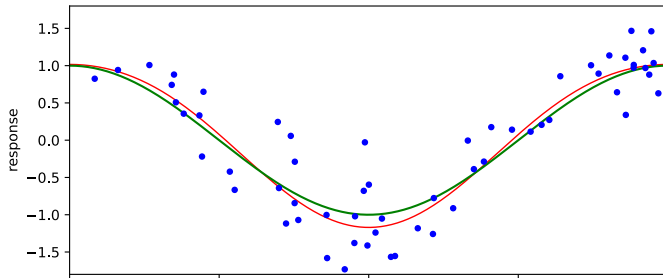
Different regimes (2)

- ▶ On the next slides we vary the eigenvalues of the covariance matrix. We always plot the true underlying function in green, the sampled data points in blue, and estimated function / minimum norm interpolator in red.

Different regimes (3)

1. **Few large eigenvalues, then many 0 eigenvalues:** similar to the under-parameterized regime. No overfitting possible, but learning works if signal is well-aligned with covariance structure.

Regime 1: few large eigenvalues,
then zeros



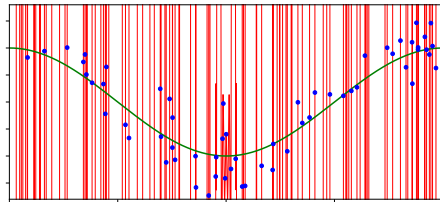
$$\lambda_1 = \lambda_2 = \lambda_3 \approx 1/2, \\ \lambda_4 = \dots = \lambda_d = 0.$$

Different regimes (4)

2. Few large eigenvalues, then few small ones:

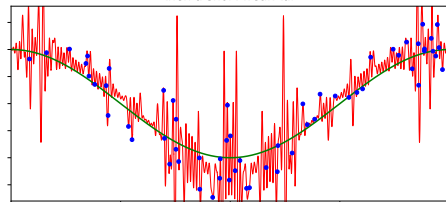
Here interpolation is possible if we have enough small eigenvalues in the tail (essentially, we need n non-zero eigenvalues in the tail). But the variance remains large. We depict two cases, one where we just have enough non-zero eigenvalues to interpolate (left), and another case where we have some more non-zero eigenvalues (right). We can see that the variance of the red predicted function decreases with with the number of small non-zero eigenvalues.

Regime 2: few large eigenvalues,
then a short weak tail



$$\begin{aligned}\lambda_1 &= \lambda_2 = \lambda_3 = 1/2, \\ \lambda_4 &= \dots = \lambda_{60} = 0.005, \\ \lambda_{61} &= \dots = \lambda_d = 0.\end{aligned}$$

Regime 2: few large eigenvalues,
then a short weak tail

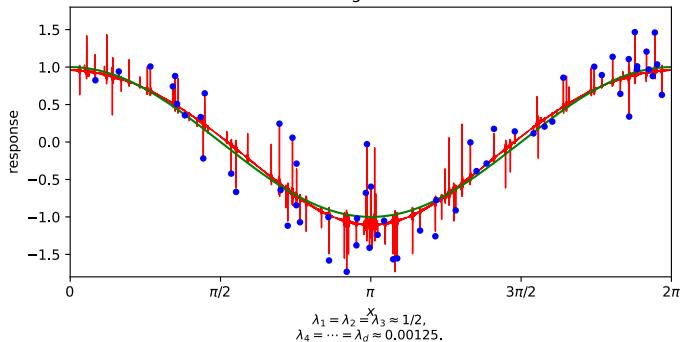


$$\begin{aligned}\lambda_1 &= \lambda_2 = \lambda_3 = 1/2, \\ \lambda_4 &= \dots = \lambda_{150} = 0.005, \\ \lambda_{151} &= \dots = \lambda_d = 0.\end{aligned}$$

Different regimes (5)

3. Few large eigenvalues, many small ones: benign overfitting

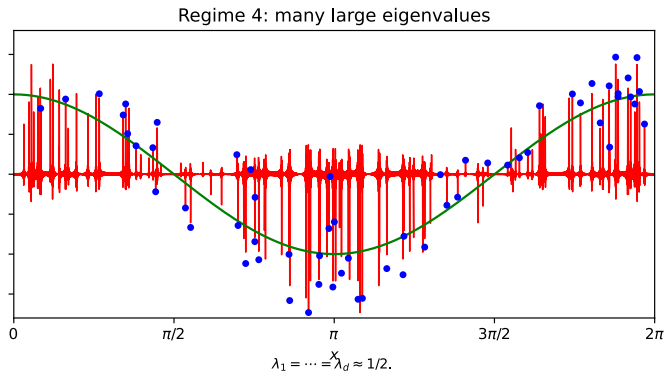
Regime 3: few large eigenvalues,
then a long flat weak tail



Different regimes (6)

4. Many large eigenvalues:

impossible to learn, bias remains high (this is essentially the situation as in the isotropic Gaussian sampling that we have seen already before)



Linear regression: summary

Whether benign overfitting can take place depends on the covariance matrix of the data and whether it is aligned with the signal θ_* that we try to recover.

It can take place if the "effective dimension" of the signal k is small compared to n and if the remaining dimensions contain isotropic small noise to allow for the interpolation.

Analysis is difficult, and little is known beyond linear setting.

Does benign overfitting contradict the classical learning theory?

Literature: On Uniform Convergence and Low-Norm Interpolation Learning, Lijia Zhou, Danica J. Sutherland, Nathan Srebro, NeurIPS 2020

Hang on ...

- ▶ Function classes in deep learning are so large that classic generalization bounds are not informative.
- ▶ However, there is also Vapnik's theorem: uniform convergence is not only sufficient, but also necessary for consistency.

So 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 (implicit regularization!)
- ▶ Consider the case of the minimum norm interpolator. Srebro et al (2020) prove for linear regression 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.” However, one can prove 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.

See also [“The Implicit Bias of Benign Overfitting”](#) by Ohad Shamir, 2022.

Summary of “modern” learning theory

To explain the generalization capability of DNNs, we currently use the following chain of arguments:

- ▶ Consider the highly over-parameterized regime, high-dimensional setting.
- ▶ By the geometry of the loss landscape, we can find global optima (interpolating solutions) easily through SGD (or some other algorithms).
- ▶ 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 that under certain assumptions 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.

So far, none of the results could be extended to the really interesting scenarios.

Much more work required to understand what is really going on...

Why (?) do (?) we need large models?

Representation power of neural networks

How “powerful” are neural networks in general?

Representation power: which functions can be represented in general by a specific neural network? How can we describe the space of functions mathematically?

Trainability: given that a network can represent a specific function, and we just observe training data from this function, can we actually find a parameter vector that represents this function?

Note that these are two very different questions!

Two-layer networks are dense in continuous functions: statement

Here is a classical result, see Pinkus: Approximation theory of the MLP model in neural networks. Acta Numerica, 1999:

Consider the family of fully connected neural networks with one hidden layer with an arbitrary number m of hidden units, with continuous activation function σ in the hidden layer, and with one linear output neuron. The functions represented by this family are

$$\mathcal{F} := \text{span}\{x \mapsto \sigma(w^\top x + b) \mid w \in \mathbb{R}^d, b \in \mathbb{R}\}$$

The following theorem says that we can approximate pretty much any continuous function by such a network.

Two-layer networks are dense in continuous functions: statement (2)

Theorem 56 (Universal approximation)

Consider any compact subset $K \subset \mathbb{R}^d$. If the activation function σ is continuous and is not a polynomial, then the family $\mathcal{F}$ is dense in the space $\mathcal{C}(K)$ (continuous functions, with respect to the sup-norm):

For every every $f_* \in \mathcal{C}(K)$, and every $\varepsilon > 0$, there exist a number $m \in \mathbb{N}$ of hidden units and corresponding neural network parameters θ such that

$$\sup_{x \in K} |f_*(x) - f_\theta(x)| < \varepsilon.$$

Two-layer networks are dense in continuous functions: proof

Proof sketch.

Case 1: For simplicity, assume first that $\sigma \in C^\infty(\mathbb{R})$, the space of functions that are infinitely often continuously differentiable. .

- Consider a single neuron of the hidden layer: $\sigma(w^\top x + b)$. We first show that arbitrary derivatives of this neuron with respect to w are contained in $\bar{\mathcal{F}}$ (the topological closure of $\mathcal{F}$). To see this, fix $w \in \mathbb{R}^d$, $b \in \mathbb{R}$ and $i \in \{1, \dots, d\}$. For $h \neq 0$ consider the difference quotient

$$Q_h(x) = \frac{\sigma((w + he_i)^\top x + b) - \sigma(w^\top x + b)}{h} = \frac{1}{h} \sigma((w + he_i)^\top x + b) - \frac{1}{h} \sigma(w^\top x + b).$$

For each fixed $h \neq 0$, Q_h is a linear combination of two neurons, so $Q_h \in \mathcal{F}$, so $\lim_{h \rightarrow 0} Q_h \in \bar{\mathcal{F}}$. Moreover, as $\sigma \in C^\infty$, we have $\lim_{h \rightarrow 0} Q_h = \partial_{w_i} \sigma(w^\top x + b)$.

Two-layer networks are dense in continuous functions: proof (2)

- We can easily extend this argument to prove that for every multi-index $\alpha = (\alpha_1, \dots, \alpha_d)$,

$$\frac{\partial^{|\alpha|}}{\partial w^\alpha} \sigma(w^\top x + b) = x^\alpha \sigma^{(|\alpha|)}(w^\top x + b) \in \overline{\mathcal{F}}, \quad x^\alpha := x_1^{\alpha_1} \cdots x_d^{\alpha_d}.$$

- We now set $w = 0$, which gives, for every $b \in \mathbb{R}$,

$$x^\alpha \sigma^{(|\alpha|)}(b) \in \overline{\mathcal{F}}.$$

- Dividing by the nonzero scalar $\sigma^{(|\alpha|)}(b_{|\alpha|})$ yields

$$x^\alpha \in \overline{\mathcal{F}} \quad \text{for every multi-index } \alpha.$$

(here we need to be careful that we do not divide by 0: Because σ is not a polynomial by assumption, for each $k \in \mathbb{N}$ there exists $b_k \in \mathbb{R}$ with $\sigma^{(k)}(b_k) \neq 0$)

Two-layer networks are dense in continuous functions: proof (3)

- ▶ Thus $\overline{\mathcal{F}}$ contains all monomials, hence all polynomials.
- ▶ By the Weierstrass theorem the polynomials are dense in $C(K)$ for every compact $K \subset \mathbb{R}^d$.

Case 2: If $\sigma \neq C^\infty$, we “smooth” it first, for example by a convolution with a C^∞ -function, and adapt our argument to this case.



Two-layer networks are dense in continuous functions: discussion

So two-layer networks seem already sufficient for pretty much everything?!?

Yes ... but:

- ▶ The theorem does not say anything about the size of m in dependence of ε and the complexity of f
- ▶ It does not say anything about the complexity of the solution function f_θ (e.g., its Lipschitz constant).
- ▶ It does not say whether one can actually find such a function from training data, and how much data one might need.
- ▶ And it says nothing about generalization outside the training set.

Depth can be more expressive than width

In terms of representation, there are a number of results that say that depth can be much more expressive than width:

Telegarsky 2016 constructs functions as follows:

- ▶ Fix a depth L of a network.
- ▶ He constructs a specific function f_L that is exactly represented by a ReLU network of depth L and width $O(1)$
- ▶ But at the same time, if we try to approximate the same function by a network with depth at most $L - 1$ up to constant accuracy (in the L_1 -norm), then we need width of the order $2^{\Omega(L)}$.

M. Telgarsky. Benefits of depth in neural networks. In *Proceedings of COLT*, 2016.

Gradient descent might not be able to learn a small ReLU network

Regarding the question whether it is always easy to find a specific function represented by a neural network, there are a number of negative results. Here is one of them:

- ▶ Let $\mathcal{D}$ be a “nice” input distribution on $\mathbb{R}^d$ (e.g. uniform on $\{-1, +1\}^d$, the standard Gaussian, or the uniform distribution on the unit sphere).
- ▶ We consider any neural network architecture f_θ with $\text{poly}(d)$ parameters. (the network is “small”)
- ▶ We consider an idealized learner who can learn the function by querying the gradient of the true population loss:

$$\nabla_\theta \mathbb{E}_{x \sim \mathcal{D}}[\ell(f_\theta(x), f^*(x))] \Big|_{\theta=\theta_t}$$

So the learner can perform variants of gradient descent such as GD, SGD, Adam, starting at some parameter vector θ_0 and then working its way to convergence.

Gradient descent might not be able to learn a small ReLU network (2)

- In particular, we do not consider any statistical issues stemming from finite sample size, and we also ignore all computational issues.

Theorem 57 (Hardness of learning a simple network)

In this setup, there exists a family $\mathcal{F}$ of target functions $f^* : \mathbb{R}^d \rightarrow \mathbb{R}$ with the following two properties.

1. Every $f^* \in \mathcal{F}$ is exactly computed by f_θ for some parameter setting θ^* (the function can be computed by a small network)
2. Any (possibly randomised, computationally unbounded) algorithm that interacts with the data only through T (possibly noisy) gradient queries and outputs a hypothesis $\hat{f}$ satisfying $P_x(\hat{f}(x) \neq f^*(x)) \leq \frac{1}{3}$ for a uniformly random $f^* \in \mathcal{F}$, must use $T \geq 2^{\Omega(d)}$ gradient queries (gradient methods such as GD, SGD, Adam, etc need exponentially many steps to achieve a non-trivial result)

Gradient descent might not be able to learn a small ReLU network (3)

O. Shamir. Distribution-specific hardness of learning neural networks. *Journal of Machine Learning Research*, 19(32):1–29, 2018

Gradient descent might not be able to learn a small ReLU network (4)

Let us try to understand this:

- We can construct a family of functions for which a small network exists but *no* gradient-based algorithm can find it in polynomial time.

So definitely, representation issues are not the whole story. We really need to look into specific algorithms to understand what they can and cannot learn.

Large models are necessary for robust interpolation

Literature: A universal law of robustness via isoperimetry. Sebastien Bubeck, Mark Selke, NeurIPS 2021.

Extended version of this paper:

A Universal Law of Robustness via Isoperimetry, Journal of the ACM, 2023.

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

$$\exp(-nd/L^2)$$

- ▶ 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.)

Result by Bubeck and Selke (4)

- ▶ 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 at the time when the paper was published, 2021, used 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.

The loss landscape in our favor

Loss landscapes

Minimizing a loss over a complex network architecture is typically not a convex optimization problem. Yet, one can observe that standard optimization algorithm do a surprisingly good job at finding good local (?) / even global optima.

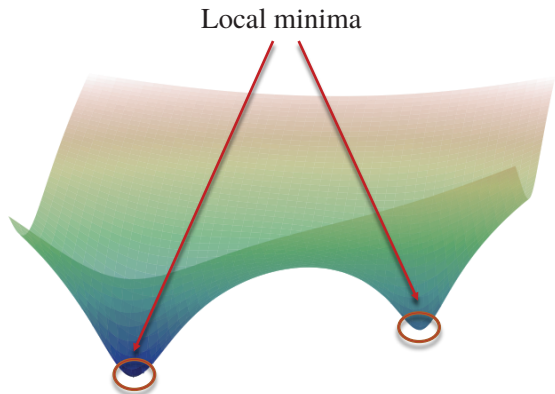
Why???

On the following few slides, I will just point out some of the hypotheses why this might be the case. We are not going to explore the topic in depth.

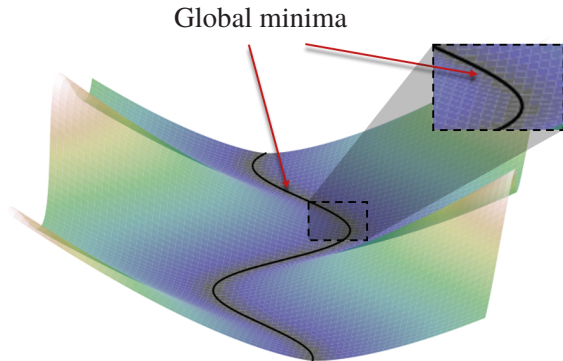
Be aware that many of the results cited below do not hold in general, but for very specific architectures, with specific loss functions.

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 (left side). But they often look as on the right side:



(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 over-parameterized regime seems to be rather as on the right side of the figure: 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.

Over-parameterization creates many connected global minima

If a neural net has m parameters and is trained on n data points, with $m > n$, the locus of global minima is usually not discrete, but rather an $m - n$ - dimensional submanifold of $\mathbb{R}^n$ (Cooper: The loss landscape of over-parameterized neural networks. Arxiv 2018)

Making a network wider might an connect previously discrete minima into a single manifold. (Simsek, Berfin, et al.: Geometry of the loss landscape in overparameterized neural networks: Symmetries and invariances. ICML, 2021.)

Minima tend to be connected by low-loss paths

The minima found by two networks are connected by a path of non-increasing error.

Frankle, Jonathan, et al. : Linear mode connectivity and the lottery ticket hypothesis. ICML, 2020.

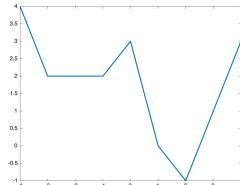
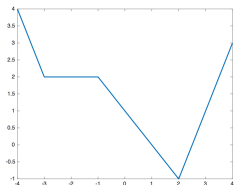
Width removes bad basins

From narrow to wide networks, there is a phase transition from having suboptimal basins to no suboptimal basins

(Li, Tang, and Sun, On the Benefit of Width for Neural Networks. SIAM J Optimization, 2022)

Li, Ding, and Sun, Over-Parameterized Deep Neural Networks Have No Strict Local Minima For Any Continuous Activations:

The loss landscape looks as in the left plot, not as in the right plot:



Training dynamics

Beyond the loss landscape itself, there are also many papers that study the dynamics of neural network training with specific algorithms (Vanilla GD, SGD, Adam, ... for specific initializations).

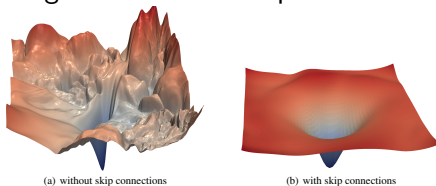
Example: edge of stability:

- ▶ One might expect large learning rates to bias training toward flatter solutions because sharp regions are unstable.
- ▶ However, GD may not simply avoid sharpness; instead it often sits right at the maximum sharpness compatible with the step size.
- ▶ There is a nice blog about it: <https://centralflows.github.io/part1/>

Jeremy M. Cohen, Alex Damian, Ameet Talwalkar, J. Zico Kolter, Jason D. Lee:
Understanding Optimization in Deep Learning with Central Flows. ICLR 2025.

Summary

- ▶ There are many, many papers out there that show that for certain architectures under certain assumptions for certain loss functions, with certain optimization algorithms, something nice happens to optimization.
- ▶ General tenor: large number of parameters helps!
- ▶ There are also lots of papers that consider specific tricks of the trade and explain them in terms of the loss landscape. For example, skip connection make the loss landscape smoother, Li et al: Visualizing the Loss Landscape of Neural Nets. NeurIPS, 2018:



- ▶ In the end, there is not a single reason / story / result that explains the success of neural network optimization. But together, many results do paint a picture.

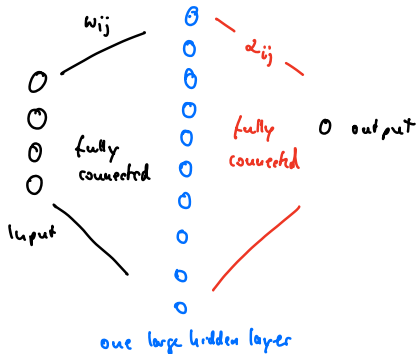
Infinite width theory and the neural tangent kernel

Literature:

- ▶ Original paper: Neural Tangent Kernel: Convergence and Generalization in Neural Networks, Arthur Jacot, Franck Gabriel, Clément Hongler, NeurIPS 2018.
- ▶ Some of the derivations below are nicely written in this paper: On Exact Computation with an Infinitely Wide Neural Net. Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, NeurIPS, 2019
- ▶ The lazy learning regime is discussed here:
Chizat, Oyallon, Bach (2019). On Lazy Training in Differentiable Programming. NeurIPS 2019.

Why could it help to have wide networks?

A whole branch of theory papers considers wide shallow neural networks, for example a network that has only one very wide hidden layer:



The idea was to explore why it might / might not be good to have very wide networks.

Gradient flow

- ▶ An important tool in this literature is to analyze gradient descent with infinitesimal step size, called gradient flow (Bach book, section 12.3.)
- ▶ Denote by $u_t := u_t = (f_{w_t}(x_1), f_{w_t}(x_2), \dots, f_{w_t}(x_n))^T \in \mathbb{R}^n$. the vector of predictions on the the training data at time t . One can show (Arora, Lemma 3.1) that while using gradient flow to train a fully connected, differentiable network with squared loss, the vector of predictions on the training points evolves according to a linear differential equation:

$$\frac{du_t}{dt} = -H_t(u_t - y),$$

where $H_t \in \mathbb{R}^{n \times n}$ is the matrix

$$H_{ij,t} = \left\langle \frac{\partial f_{w_t}(x_i)}{\partial w}, \frac{\partial f_{w_t}(x_j)}{\partial w} \right\rangle.$$

- ▶ The entries of H_t are inner products between gradients of the network's outputs. As such, H_t is an inner product matrix (a kernel matrix).

Neural tangent kernel (NTK) regime

- ▶ Consider a two-layer fully connected network with ReLU activation.
- ▶ One can prove (Arora, appendix D) that if the network is very wide, H_t essentially doesn't change with time t but remains close to constant: $H_t \approx H_0$.
- ▶ At the same time, when we drive the width of the hidden layer to infinity, the kernel matrix H_0 converges to a kernel function, called the neural tangent kernel K_{NTK} .
- ▶ From this one can deduce that in the infinite-width regime, training a neural network behaves like training a kernel method with kernel K_{NTK} . One can exactly compute the limiting solutions for a range of different loss functions.

What is the neural tangent kernel?

- For a neural network $f(w, x)$, the neural tangent kernel is defined by

$$K_{\text{NTK}}(x, x') = \langle \nabla_w f(w_0, x), \nabla_w f(w_0, x') \rangle,$$

where w_0 denotes the random initialization.

- Intuitively, $K_{\text{NTK}}(x, x')$ measures how similarly the predictions at x and x' change under an infinitesimal parameter update.

No feature learning in the NTK regime

- ▶ In very wide networks, all the network parameters only change by minuscule amounts during training. Intuitively, we have so many hidden neurons that the gradient signal is distributed to all of them. The many tiny changes can still add to something substantial in the final layer, thus allowing the network to learn to produce the correct output. Yet, looking at the weights themselves, they do not change a lot.
- ▶ In particular, the network does not learn any new data representation but essentially sticks to the (random) representation that it got assigned through initialization.
- ▶ In this sense, the model behaves like a random feature model.

Wide shallow networks in practice

- ▶ The similarity to a random feature model already hints that shallow very wide networks might not be the best of all learning scenarios
- ▶ NTK theory gives a powerful explanation for optimization in very wide networks, but it does not fully explain feature learning in practical finite-width networks.
- ▶ In particular, numerical experiments show that the performance of networks trained in the lazy regime degrades and in particular does not exceed that of some classical linear methods.

What are the limitations of NTK theory?

NTK theory gives a clean and rigorous explanation of optimization in the very-wide limit. However, the NTK regime has important limitations.

- ▶ The network stays close to its initialization.
- ▶ The tangent features remain essentially fixed.
- ▶ The model behaves like a kernel method.
- ▶ Representation learning is largely absent.

So in the end, NTK theory explains the solutions of very wide networks, but it does not really explain feature learning as it takes place in practical networks.

Theory regarding depth?

Theory regarding deep networks

- ▶ There exist a number of isolated results regarding very deep networks. But they are very specific to individual architectures, and they don't really tell any coherent story.
- ▶ So currently, there is no convincing line of work that might explain why networks need to be deep to be successful.
- ▶ I would say, currently it is a widely open research question

Take home

Learning in the under-parameterized regime

The setup: we tend to have not so many data points

- ▶ We need to explicitly encode prior knowledge about the data, in particular in terms of the feature representation and the similarity/distance/kernel function
- ▶ In case of few data points, we need to control the capacity of the underlying function class of the learning algorithm
- ▶ In particular, we need to be careful to not overfit.

Learning in the under-parameterized regime (2)

The success of this approach can be judged by some guiding principles:

- ▶ Traditional bias-variance decomposition
- ▶ Try to work with small function classes
- ▶ or use regularization
- ▶ make sure your results are as stable as possible

In particular, these principles are often enforced explicitly.

Learning in the over-parameterized regime

Setup:

- ▶ We have a huge number of sample points to start with.
- ▶ We typically do not have an explicit feature representation. Features are learned by the network (but then also hard to interpret). Eg images (raw features are pixels) or text (raw features are tokens).
- ▶ We train a complex neural network architecture, often to achieve 0 training error, and we often keep training much longer.

Learning in the over-parameterized regime (2)

If this approach is successful, the reason is typically that some of the “old” principles are still at work, but happen implicitly, perhaps even without the algorithm designer knowing about it:

- ▶ Bias-Variance decomposition beyond the interpolation threshold
- ▶ Implicit regularization by the choice of the algorithm
- ▶ Stability of the optimizers
- ▶ The focus of analysis changes from a statistics analysis towards analyzing properties of the optimization algorithm.

Old principles in new disguise.

Learning in the over-parameterized regime (3)

On the downside ...

- ▶ most theoretical results have been established on rather simple models, and it is not proven that they also hold in complex models. A
- ▶ there might be some other, powerful principles that we haven't been able to identify.
- ▶ In particular, there is no magic involved. But it is sometimes hard to see behind the curtain and understand why something works or doesn't work.
- ▶ We are still very far from really understanding learning in the over-parameterized regime!!!

Which approach to use?

- ▶ Many machine learning applications still need the under-parameterized regime: we simply don't have enough data to successfully train a complex neural network!
- ▶ Neural networks are super-helpful if we have / can generate a lot of data. The prime examples are image data and text data, but there are also many other cases, for example simulation-based inference in science.
- ▶ As a ML engineer, you need to be confident in both worlds!

New failure modes

Deep learning exhibits failure modes that partly already present in the classical regime, but are more pronounced in the new regime:

- ▶ Adversarial examples
- ▶ Spurious features
- ▶ Shortcut learning
- ▶ ...

The reason is that

- ▶ We use raw data without much pre-processing and just hope that ML will discover whatever is necessary
- ▶ We use complex architectures, and training them is hard to gain intuition about
- ▶ The resulting models are black box and impossible to understand, even as ML engineers.

Combined with the fact that there doesn't exist strong theory to guide us in our choices, this means:

Carefully testing, benchmarking, and validating plays a central role now!!!

Machine Learning in the context of society

Fairness

Literature:

- Book 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.

Is it fair?

Example: Compas

The field of fairness in machine learning was, to a large extent, triggered by the discussion around the COMPAS algorithm in 2016:

- ▶ The algorithm COMPAS is used in the US nationwide to decide whether defendants awaiting trial should be released on bail (German: on bail = “auf Kautions 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)
- ▶ In 2016, ProPublica published the report "Machine bias: There is software across the country to predict future criminals. And it is biased against black". It started a heated debate, and this debate started the field of fairness in machine learning.
- ▶ Let us look at the two points of view in this debate:

Example: Compas (2)

First point of view, condensed from the ProPublica arguments: 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). In maths:

$$P(\text{high risk score} | \text{not reoffend}, \text{race}=\text{black}) \gg P(\text{high risk score} | \text{not reoffend}; \text{race}=\text{white})$$

Intuitively: Even though all 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 (3)

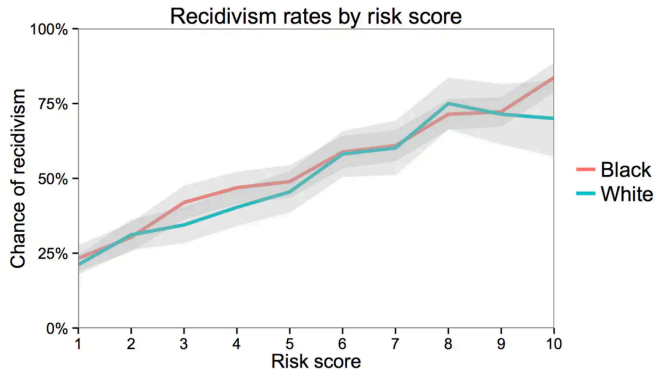
The COMPAS builders point of view: the score is not biased:

If you have a certain score s , the probability to reoffend is about the same for white and black persons:

$$\forall s = 1, \dots, 10 :$$

$$P(\text{reoffend} | \text{score} = s; \text{race}=\text{white}) \approx P(\text{reoffend} | \text{score} = s; \text{race}=\text{black})$$

Example: Compas (4)



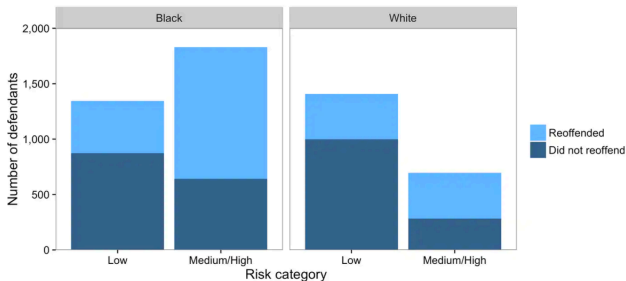
Recidivism rate by risk score and race. White and black defendants with the same risk score are roughly equally likely to reoffend. The gray bands show 95 percent confidence intervals.

Consequence: when judges see a defendant's risk score, they need not consider the defendant's race when interpreting it.

Example: Compas (5)

Both stories sound convincing, isn't it?

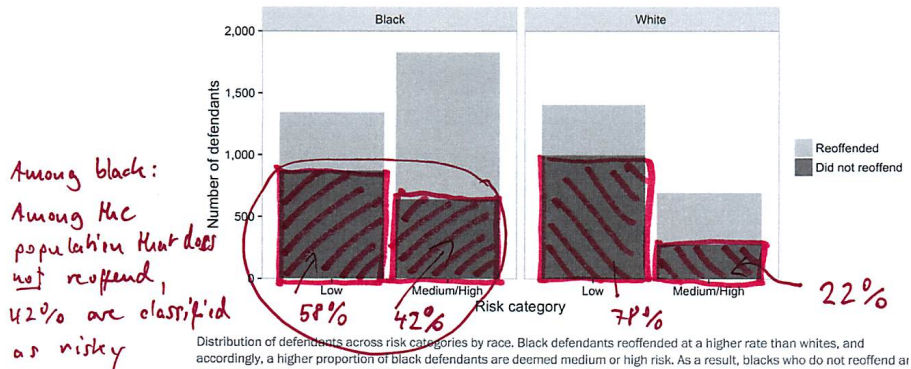
Let's dig into numbers and consider the summary statistics of the data:



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 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.

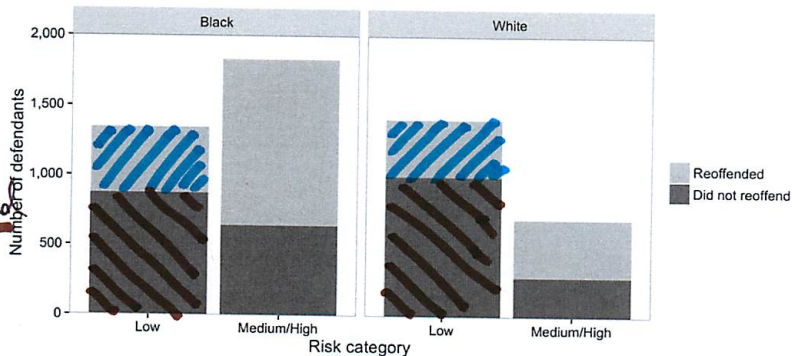


Example: Compas (7)

Second 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).

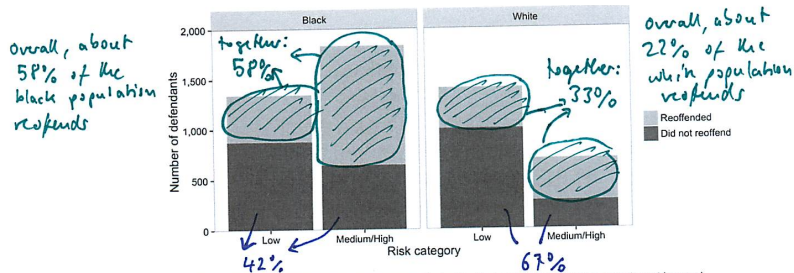
For low risk,
about the same
ratio of
reoffend / not reoffend

Similarly for
high risk.



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 we will see below: it is impossible to construct non-trivial classifier that is “fair” with respect to both points of view.

First important takeaway: it is not obvious at all how to measure fairness!

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:](https://www.washingtonpost.com/news/monkey-cage/wp/2016/10/17/can-an-algorithm-be-racist-our-analysis-is-more-cautious-than-propublicas/)

[//www.washingtonpost.com/news/monkey-cage/wp/2016/10/17/can-an-algorithm-be-racist-our-analysis-is-more-cautious-than-propublicas/](https://www.washingtonpost.com/news/monkey-cage/wp/2016/10/17/can-an-algorithm-be-racist-our-analysis-is-more-cautious-than-propublicas/)

Example: College admission and affirmative action

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 and affirmative action (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 and affirmative action (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 and affirmative action (4)

Oct 2019: judges concluded: what Harvard does is lawful:

“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” (quote from the Harvard webpage, potentially biased)

Sources:

<https://admissionscase.harvard.edu/>

Fair ML book, Sec. on “Counterfactual discrimination analysis”

Second important takeaway: trying to be “more fair” might come “at the cost of accuracy”, and other groups might feel treated more unfairly.

Machine learning systems are often 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: 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)

About 10 years ago: Amazon, where men hold 74 percent of the company's managerial positions, used a tool to screen applications, but then it got discontinued: it had a severe 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.

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., compas) 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!!!

It is an issue. Always. And typically there is no simple solution.

But there are at least some things that we can do, see below.

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.

- ▶ **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?

By selecting features we influence the way we model the problem. The selection of measurements might systematically favor/ disfavor certain groups!

See the examples in the section on data earlier in the lecture.

Sources of unfairness: target variable

Problem of target measurement vs target construct:

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.

See the examples in the section on data earlier in the lecture.

Discussion

Fairness is not a technical requirement, but one that comes from society!

Nothing about fairness is obvious and simple.

Formal notions of fairness, and why we cannot satisfy all of them at the same time

Measuring group fairness in a simplified setup

As we have seen in the examples above: there is no unique definition of fairness.

Below are some popular definitions, many more definitions exist.

Measuring group fairness in a simplified setup (2)

Assume the 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”).

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 (called independence in fairml book) (2)

Alternative (more general) definition that can be used beyond the discrete cases:

The two random variables $\hat{Y}$ and A are stochastically independent:

$$\hat{Y} \perp A$$

Demographic parity (called independence in fairml book) (3)

Requiring 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

$$P(\hat{Y} = 1 | Y = 0, A = 0) = P(\hat{Y} = 1 | Y = 0, A = 1)$$

(Alternatively, if one cares more about one direction than the other, one could just request one of the two.)

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 ($A = 1$).
- ▶ 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 (called separation in fairml book) (2)

More general definition:

Conditionally on the random variable Y , the two random variables $\hat{Y}$ and A are independent:

$$\hat{Y} \perp A \mid Y$$

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...

Equalized odds (called separation in fairml book) (3)

Recap: For discrete random variables A, B, C we say $A \perp B \mid C$ iff

$$P(A = a, B = b \mid C = c) = P(A = a \mid C = c) \cdot P(B = b \mid C = c)$$

Equivalently:

$$P(A = a \mid B = b, C = c) = P(A = a \mid C = c)$$

In words: Once C is known, additionally knowing B gives no further information about A .

Predictive parity (called sufficiency in fairml book)

Predictive parity criterion:

$$\forall y : P(Y = 1 | \hat{Y} = y, A = 0) = P(Y = 1 | \hat{Y} = y, A = 1).$$

If the prediction for a person is $\hat{Y} = y$, then the probability that the true value satisfies $Y = 1$ 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 (called sufficiency in fairml book) (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

Impossibility results

There exist all kinds of statements that in non-trivial circumstances, the different criteria typically cannot hold at the same time:

(For literature and proofs, see 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).)

Demographic parity vs Predictive parity

Proposition. Assume that all events in the joint distribution of $(A, \hat{Y}, Y)$ have positive probability. Assume that A and Y are not independent. Then DP and PP cannot both hold at the same time.

Proof:

$$\begin{aligned} P(A = a, Y = y, \hat{Y} = \hat{y}) &= P(A = a \mid Y = y, \hat{Y} = \hat{y}) P(Y = y, \hat{Y} = \hat{y}) \\ &\quad \text{(exploit PP)} \\ &= P(A = a \mid \hat{Y} = \hat{y}) P(Y = y, \hat{Y} = \hat{y}) \\ &\quad \text{(exploit DP)} \\ &= P(A = a) P(Y = y, \hat{Y} = \hat{y}) \end{aligned}$$

Thus $A \perp (Y, \hat{Y})$, which implies $A \perp Y$. This is a contradiction to our assumption. ⚡



Equalized odds vs predictive parity

Proposition. Assume that all events in the joint distribution of $(A, \hat{Y}, Y)$ have positive probability. Assume that A is not independent of Y . Then equalized odds and predictive parity cannot both hold.

Proof:

By similar arguments as above one can see that $A \perp \hat{Y} \mid Y$ and $A \perp Y \perp \hat{Y}$ imply that $A \perp (Y, \hat{Y})$, which as above implies both $A \perp Y$. This is a contradiction to our assumption.



Demographic Parity vs Predictive parity

Proposition. Assume Y is binary, A is not independent of Y , $\hat{Y}$ not independent of Y . Then demographic parity and predictive parity cannot both hold.

Proof: Also elementary, skipped.

Indiviudal 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 indiviudals 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.

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	Kleinberg, Mullainathan, Raghavan (2016)
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.

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:

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 (2)

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 (3)

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.

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:

$$\text{unf}(f) = P(f(x) > 0 \mid A = 1) - P(f(x) > 0 \mid A = 0)$$

and its empirical version as

$$\widehat{\text{unf}}(f) = \frac{1}{n} \sum_{i|a_i=1} \mathbb{1}_{f(x_i)>0} - \sum_{i|a_i=0} \mathbb{1}_{f(x_i)>0}$$

Train for accuracy and fairness (2)

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. 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

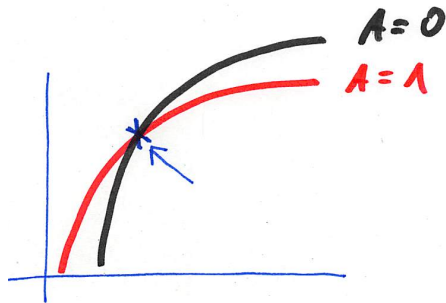
Fix models in hindsight (3)

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 fairness 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.

Discussion

- ▶ Fairness is always an issue in machine learning.
- ▶ 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).
- ▶ Keep in mind: If in a particular application, the AI systems don't meet the desired amount of fairness, it is always an option to not use an AI system!

See also: Sam Corbett-Davies, Sharad Goel: The Measure and Mismeasure of Fairness: A Critical Review of Fair Machine Learning. Arxiv, 2018

Explainability for tabular data

Literature: as an introductory textbook, I recommend the book by Christoph Molnar: Interpretable Machine Learning (exists online).

Introduction

Why do we need explainable machine learning?

Black box models with millions of parameters

Hard to debug, hard to trust

Might want to "understand" what the model does:

- in science
- for critical applications (medicine)
- in society (loan, university admission, ...)

AI regulation requires transparency!

Purposes of explainable AI

Would like to be able to query for "explanations" why the ML model acts/decides/predicts in a specific way.

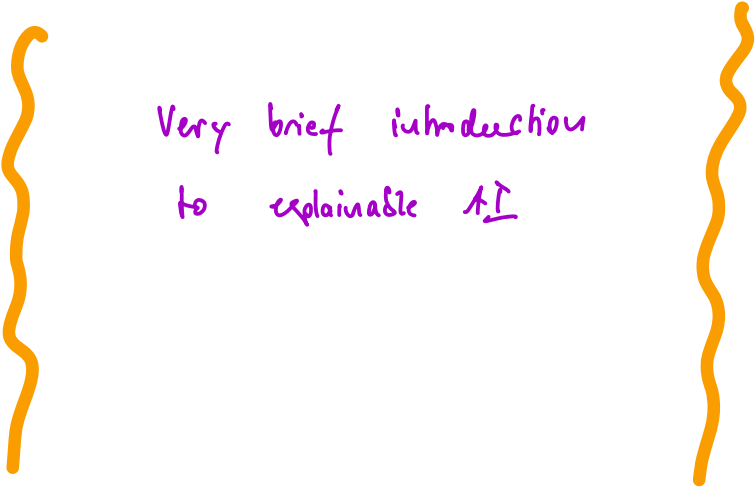
Might serve to

- debug your code
- establish trust (society applications; medicine)
- balance power/control between model provider and "customer"
- give hints for concrete actions (eg recourse)
- reveal scientific insights (truth?)

AI Regulation

EU just passed the AI Act:

- AI applications are assigned to a risk category
- High-risk applications need to satisfy specific transparency criteria. Explainability is one of them!



Very brief introduction
to explainable $\uparrow \downarrow$

Setup for now

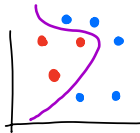
Machine learning on feature-based data (not text, not images)

- Patients described by numeric values
- Loan applicants "



Data is high-dimensional (many features) $\begin{pmatrix} 1 \\ -2.3 \\ 1.5 \\ 17 \\ 32 \end{pmatrix}$ features

Task: classification or regression



Interpretable vs. explainable ML

The gold standard: interpretable ML

Use models that are so simple that we can understand/judge their underlying decision mechanism:

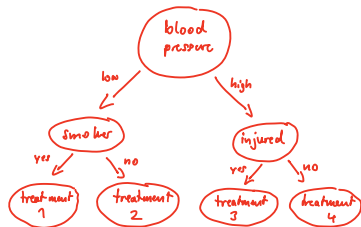
- decision trees:

- linear models:

$$f(x) = \sum a_i x_i$$

- generalized additive models (GAMs)

$$f(x) = \sum f_i(x_i)$$



But ...

Interpretable ML might not always be powerful enough
to solve complex prediction problems.

Local post-hoc explanations

- The model is so complex that we cannot understand it as a whole
- Instead: trying to explain why the model made a particular decision
- We invoke another algorithm, the explanation algorithm:
 - it receives a data point as input, x_0 , and the prediction $f(x_0)$
 - the function f
 - it can sample $x \in X$ and query $f(x)$
- This alg tries to justify "post hoc" the decision $f(x_0)$

Example: Mr. Schmidt doesn't receive the loan because his income is too low.

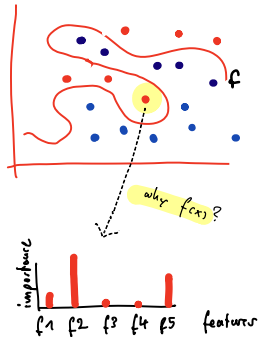
Feature attribution algorithms

- Consider a complex decision model f (cannot explain it as a whole)
- Request local post-hoc explanation for individual decisions:

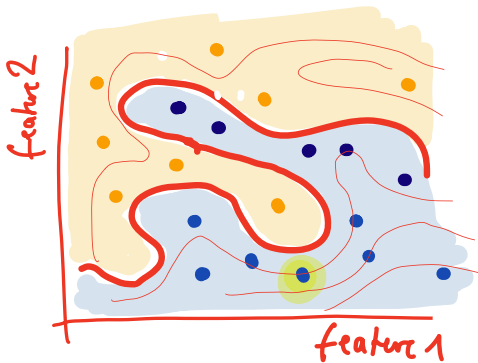
why $f(x)$?

- "Explanation" often comes in form of a feature attribution:

which of the input features was important for the particular decision?



Rationale for creating such explanations



If small changes in feature i of x change the decision about x , then feature i is considered important.

Often boils down to approximating f locally at x by a simple model, for example a linear function.

Global vs local explanations



Want to understand/explain
the decision mechanism/rationale
globally



Want to explain only
decisions for individual
local data points

Explaining the decision fct f vs the data-generating process



we don't care about the data,
but really about the particular
decision fct f

(eg, credit decisions)



We want to relate the
data to the decision fct,
for example discover a
causal reason of a
particular disease
(eg, in science)

Model-specific vs model-agnostic explanations



Explanations depend on the model architecture, say the specific architecture of a neural network



Shap, LIME, Anchors, ...

Explanation does not make (strong) assumptions on the form of f or how it has been generated

Data attribution vs feature attribution



Which of the other points in the training set were particularly important for the prediction $f(x)$?



Which of the features of x were particularly important for the prediction $f(x)$?

Cooperative

vs

adversarial scenario



explanation provider and
explanation recipient share the
same goal

Examples:

- Model/code debugging
- Scientific discovery



... have different goals

Example: loan application

No incentive to provide
"true explanation"

Engineering vs. science of XAI

- One can **engineer** many different heuristics, criteria, algorithms that spit out explanations.
- Many of them often produce explanations that "make sense"

But what kind of **scientific guarantee** can we give?

- When are explanations "true"?
- By which standards should we evaluate explanations?
- What kind of guarantee can we give, under which assumptions?

When is the scientific approach important?

AI act: (eg, credit scoring)

Medicine

Critical applications

...

When is an engineering approach justified?

Exploratory data analysis, say in science

Code debugging

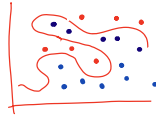
Whenever a useful explanation can help, but even many wrong explanations do not really hurt

Can we assert that an explanation is true?



? ! ?
.
.
.
~~~~~  
incomplete  
knowledge

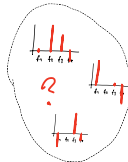
The algorithm's world



Geometry:  
distances, features, labels

? ! ?  
.  
.  
.  
} ambiguity

... rather not ...



## Local posthoc explanations (feature attributions)

## Local posthoc explanations (feature attributions)

First issues with feature importance scores: correlations and interactions

# General issues

In the following we will discuss many popular methods to assign “feature importance scores” for particular decisions.

Before we explain the methods, we already point out some general issues that will come up in many of them.



## Correlation between features in the data domain

Whenever methods try to isolate individual features as “reasons” for a decision, this might get distorted if features are correlated in the input domain.

Examples for correlated features:

- ▶ Extreme examples: we have the same feature twice.
- ▶ years of full time employment and gender
- ▶ Address and income

The machine learning model might either just pick one of the features (and not use the others); or it might use both with different weights; or in the extreme case, it might even use both features with opposite weights, which means that they cancel in the decision function, but not in terms of gradient or so.

Obviously, this might be tricky to deal with for algorithms that develop feature importance scores.

## Variable interactions in the decision function

Beyond correlated features in the input domain, features can also be coupled in the output domain: features might "interact".

Examples for feature interactions:

- ▶ XOR function (which of the two features is more important??? Impossible to answer, we need to see both. )
- ▶ Two genes that only jointly trigger a disease
- ▶ etc

More generally, we can have  $k$ -th order feature interactions where  $k$  features need to interact.

It is again very unclear what a feature importance score would do in such a situation, and it might even depend on tiny implementation details.

## Many more issues...

More discussion about these and other issues:

Barocas, Selbst, Raghavan: The hidden assumptions behind counterfactual explanations and principal reasons, FAccT 2020

## Counterfactual explanations

Original paper:

Wachter, Mittelstadt, and Russell. Counterfactual explanations without opening the black box: Automated decisions and the GDPR. 2017.

# Counterfactual explanations

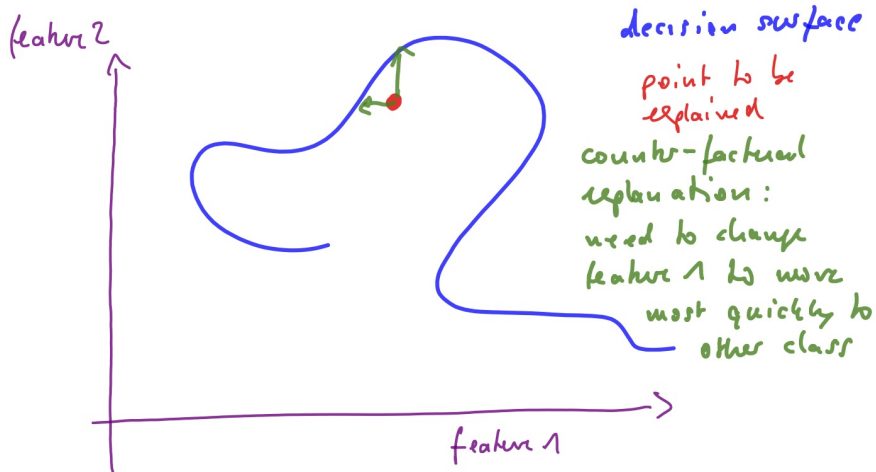
Main idea: what would be the smallest change to the input such that the decision function would return a different result?

Examples for counterfactual explanations:

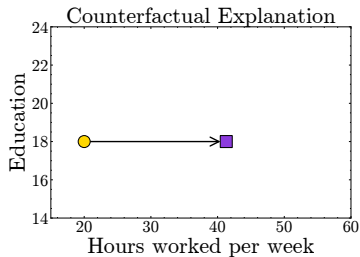
- ▶ If your income would be 500 Euro more, you would get the credit.
- ▶ If your gender would be male, you would be admitted to the program. (?!?)

To construct such an 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.

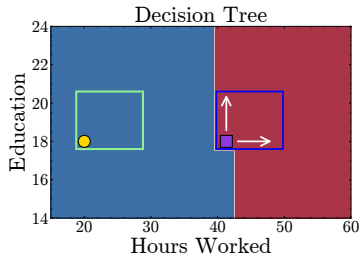
## Counterfactual explanations (2)



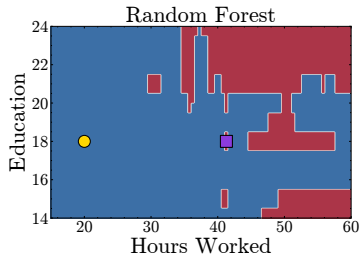
# Attention: Counterfactuals of complex functions might not be robust



(a) A recipient obtains a counterfactual explanation.



(b) What the recipient believes the decision boundary looks like.



(c) How the decision boundary may actually look like.

If the decision function is complex (right plot), users of explanations might misinterpret their meaning ...

## Are counterfactuals helpful for recourse?

- ▶ It is often argued that counterfactual explanations can be used for recourse: to know which features you would need to change to get accepted.
- ▶ But:
  - ▶ “You would get the credit if you were 10 years younger.”  
This is not actionable.
  - ▶ “You would get the credit if you earned 10.000 Euros more per year”.  
... but by that time, you also might have become 5 years older, so it is unclear whether the counterfactual is also valid if the age changes as well.  
... and by that time, the bank might have retrained its model, so it does something different anyways...
- ▶ So while a counterfactual explanation might give you a hint for recourse, it doesn't give you the full picture.



# SHAP

Literature: the original paper is:

Lundberg and Lee. A unified approach to interpreting model predictions. NeurIPS 2017.

There are thousands of publications related to SHAP, some of which I cite below, including some of our own papers.

## Intuition from game theory

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.

## Intuition from game theory (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).

## Definition of SHAP

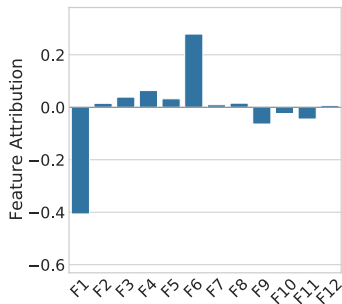
More formally: the SHAP value  $\Phi_i$  of the  $i$ -th feature of function  $f$  at point  $x$  is given as follows

$$\Phi_i(f, x) = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|! (|F| - |S| - 1)!}{|F|!} \left( v(x, S \cup \{i\}) - v(x, S) \right)$$

where  $v$  is the "value function".

## Definition of SHAP (2)

By this approach, we obtain importance scores for all features, can plot them as a bar plot, and can select the “important ones”:



**(a) SHAP**

(at least, this is the story, see below).

# Illustration with pizza

- ▶ Want to compute which toppings are most influential for the taste of a particular pizza
- ▶ Consider a large set  $\{1, \dots, d\}$  of possible toppings.

$$p = (p_1, p_2, p_3, p_4, \dots, p_d)$$

Handwritten red labels with arrows pointing to the components of the vector  $p$ :

- $p_1$ : cheese
- $p_2$ : tomato
- $p_3$ : salami
- $p_4$ : pepperoni
- $p_d$ : mushrooms



A particular pizza is described by a  $d$ -dim binary vector that says whether this topping is on the pizza or not, e.g.  $p = (0, 1, 1, 0, 0, \dots, 1)$ .

- ▶ The taste  $f(p)$  of a particular pizza  $p$  is a number between 1 and 10, say.

## Illustration with pizza (2)

- For a fixed pizza  $p$ , and a fixed set  $S$  of toppings, denote by  $p_S$  the values of  $p$  at coordinates  $S$ :

$$S = \{1, 3, 4\}, \quad p = \{\underline{1}, 1, \underline{0}, \underline{0}, 0, 1\}$$

$$\Rightarrow \quad p_S = \{1, 0, 0\}$$

- Given a particular pizza  $p = (p_1, \dots, p_d)$  and a particular set  $S \subset \{1, \dots, d\}$  of toppings, the value function aims to rate the contribution of the toppings listed in  $p_S$  to the general taste of the pizzas.

Concretely, we look at all pizzas  $P$  that agree with  $p$  in terms of the toppings  $S$ , that is  $P_S = p_S$ . We then define the value  $v(p, S)$  as the average taste of these pizzas:

$$v(p, S) := E(f(P) \mid P_S = p_S)$$

## Illustration with pizza (3)

(tricky detail: which average exactly? See below when we discuss observational vs interventional)

- To compute the importance of feature  $i$ , we now consider all possible subsets  $S$  of toppings, and compare their values with and without the ingredient  $i$ :

$$v(S \cup \{i\}) - v(S)$$

- We then sum this over all possible subsets  $S \subset \{1, \dots, d\}$  as in the SHAP formula.



## Observational value function, observational SHAP

Fix a point  $x$ . Fix a subset  $S \subset \{1, \dots, d\}$  of features. Consider all points  $X \in \mathcal{X}$  whose  $S$ -features coincide with the ones of  $x$ :  $X_S = x_S$ , sample from them, and compute the average function value  $f(X)$ :

$$v_{\text{obs}}(x, S) = E(f(X) \mid X_S = x_S) = \int f(x_S, x_{\bar{S}}) p(x_{\bar{S}} \mid x_S) dx_{\bar{S}}$$

(where  $\bar{S}$  denotes the complement of  $S$ ).

Easy in theory. But in practice it is super-difficult. We would need a huge amount of data points to approximate the conditional distributions.

## Observational value function, observational SHAP (2)



In the pizza example: to compute the value  $v(p, S)$  of the set  $S = \{\text{mushrooms}, \text{ham}\}$  for a pizza  $p$  that has  $p_S = (1, 1)$  we look at the set of all possible pizzas on the menu that have mushrooms and ham on them, and compute their average taste. If no pizza is on the menu that has, say, mushrooms, ham, and pineapple, then such a pizza is not taken into account in the average.

## Interventional value function, interventional SHAP

Interventional / marginal value function:

$$v_{\text{int}}(x, S) = E_X(f(X) \mid \text{do}(X_S = x_S)) = E_{X_{\bar{S}}}(f(x_S, X_{\bar{S}})) = \int f(x_S, x_{\bar{S}}) p(x_{\bar{S}}) dx_{\bar{S}}$$

Here we simply set the values of  $X_S$  to  $x_S$ , and then sample the remaining features  $X_{\bar{S}}$  from the marginal distribution of this feature. In this way, we break all dependencies between the features in  $S$  and its complement.

This is called the interventional SHAP because we intervene (in the causal framework: we use the do-operator to set the features in  $S$ ).

This is the approach that pretty much everybody uses: it is practical because sampling is simpler. But: it does include out-of-distribution points and breaks dependencies, which might be questionable ...

## Interventional value function, interventional SHAP (2)



In the pizza example: To compute the value  $v(p, S)$  of the set  $S = \{\text{mushrooms}, \text{ham}\}$  for a pizza  $p$  that has  $p_S = (1, 1)$ , we simply take all existing pizzas, and compare them in two versions: (1) we remove mushrooms and ham from the pizza; (2) we force mushrooms and ham on the pizza; then we compare their taste. We average over all pizzas.

In this setup, if the menu has a pizza with pineapple but none with pineapple and mushrooms, we would still include the pizza with pineapple, mushrooms, ham in the average (out of distribution).

## Interventional value function, interventional SHAP (3)

Observe that if the features are stochastically independent, then interventional and observational SHAP coincide!

## SHAP recovers GAM-components

A GAM (generalized additive model) for data points  $x = (x_1, \dots, x_d) \in \mathbb{R}^d$  is a decision function of the form

$$f(x) = c_0 + \sum_{i=1}^d f_i(x_i)$$

The functions  $f_i$  are called the component functions or shape functions. They are typically not linear functions (otherwise the whole function would be linear), but for example small decision trees or low-order polynomials.

We will later discuss more about GAMs, but for now we illustrate the connection between GAMs and SHAP:

## SHAP recovers GAM-components (2)

### Theorem 58 (SHAP for GAMs)

Let the decision function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  be a GAM as on the previous page. Assume that the component functions  $f_i$  are centered, that is  $E(f_i(x)) = 0$ . Then the interventional SHAP values coincide with the GAM component functions:

$$\Phi_i(f) = f_i(x)$$

For observational SHAP, the same is true if the features are stochastically independent (in which case observational and interventional SHAP coincide).

## SHAP recovers GAM-components (3)

Proof:

Because  $f$  is a GAM we have

$$\begin{aligned} v(S; x) &= E_{X_{\bar{S}}} \left( \beta_0 + \sum_{j \in S} f_j(x_j) + \sum_{j \in \bar{S}} f_j(X_j) \right) \\ &= \beta_0 + \sum_{j \in S} f_j(x_j) + \sum_{j \in \bar{S}} E(f_j(X_j)). \end{aligned}$$

This implies (when taking the difference, most terms cancel):

$$v(S \cup \{i\}; x) - v(S; x) = f_i(x_i) - E(f_i(X_i)).$$

Observe that the rhs does NOT depend on  $S$  any more! Hence, if we now assemble the SHAP formula we get:



## SHAP recovers GAM-components (4)

$$\begin{aligned}\Phi_i(x) &= \sum_{S \subseteq F \setminus \{i\}} w_{|S|} (v(S \cup \{i\}; x) - v(S; x)) \\ &= \sum_{S \subseteq F \setminus \{i\}} w_{|S|} (f_i(x_i) - E(f_i(X_i))) \\ &= (f_i(x_i) - E(f_i(X_i))) \cdot \sum_{S \subseteq F \setminus \{i\}} w_{|S|} \\ &= f_i(x_i) - E(f_i(X_i)).\end{aligned}$$

where the  $w_{|S|}$  are the constants of the Shapley formula. It is easy to see that they sum up to 1 (last equality).

Finally, if we assume the component functions to be centered, then  $E(f_i(X_i)) = 0$ .



# SHAP recovers GAM-components (5)

What does it mean for practice?

- ▶ SHAP values are meaningful if your function is a GAM and you use interventional SHAP.
- ▶ They might be misleading if your function is not a GAM!!! See below...
- ▶ Similar relationships can be proved for higher order GAMs and higher-order SHAP values, see the following paper:

Bordt, Luxburg: From Shapley Values to Generalized Additive Models and back.  
AISTATS, 2023.

# SHAP problems with variable dependencies and variable interactions

## Variable interactions:

- ▶ Consider the california housing data set which is about predicting the price of a house in california. Houses are described by lots of features, including longitude and latitude of the position of the house. By themselves, none of the two features is very informative. But jointly, they can be used to derive whether the house is at a beach, which makes a house much more expensive.
- ▶ Such an effect could not easily be described by a GAM. The function would need to look at **interaction terms** that depend on both latitude and longitude at the same time.
- ▶ Yet, SHAP will attribute a medium importance to both longitude and latitude and ignore all interactions.

## SHAP problems with variable dependencies and variable interactions (2)

Dependencies between features:

- ▶ Similarly, assume the data set also has a separate feature "ocean proximity". Once we know longitude and latitude, we don't need this variable any more, and can replace it by the others. In this example, there is a high dependency between the variables.
- ▶ For dependent variables, SHAP tends to distribute its "importance" to all the variables, making the importance value of each individual feature pretty meaningless.

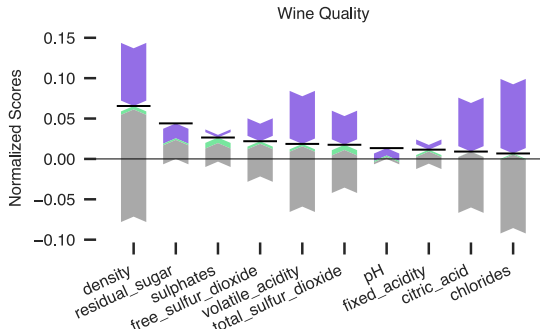
## SHAP problems with variable dependencies and variable interactions (3)

Feature rankings can be meaningless:

- ▶ Because of the two effects (interactions, dependencies), the ranking of features by SHAP can be pretty meaningless.
- ▶ In particular, it can be the case that the features that are really important achieve a low rank because they are correlated with other features.
- ▶ Simply picking the features with the highest SHAP scores is not a good idea for feature selection!

# SHAP problems with variable dependencies and variable interactions (4)

Instead, one can try to estimate the effects separately: standalone contribution, dependencies, interactions:



(Wine data set, see description next page)

## SHAP problems with variable dependencies and variable interactions (5)

Figure caption: The original scores in Figure 3 (left, black horizontal lines) suggest that density and residual sugar are most relevant, but that citric acidity is irrelevant. The scores are prone to misinterpretation. First, one may erroneously infer that variables with large scores, like residual sugar, also have large standalone predictive power. The DIP decomposition in the figure reveals that residual sugar is relevant due to cooperation instead, and thus, its role in predicting the target can only be understood in combination with other features.

Digging deeper, a pairwise DIP decomposition further reveals a positive cooperative impact between residual sugar and density; an in-depth analysis shows that the features are positively correlated (adding sugar increases the density) but have opposing effects on wine quality that cancel out unless both features are observed.

Disentangling Interactions and Dependencies in Feature Attribution. Gunnar König, Eric Günther, Ulrike von Luxburg, AISTATS, 2025.

## My conclusion: avoid SHAP whenever possible!!!

- ▶ SHAP is very widely used. I believe it is because (1) it has a nice story (game theory), and (2) it provides easy code that produces nice plots.
- ▶ SHAP is nice for GAMs - but for GAMs we might not need local interpretability methods.
- ▶ SHAP can be super misleading in general, and this is not easy to fix.
- ▶ Many mis-interpretations happen with SHAP, the literature is full of them.
- ▶ So unless you really know what you are doing, I recommend to avoid SHAP.

Many of the criticisms for SHAP also concern other methods for feature attribution (in particular, the correlation / interaction issues).



# LIME

Some references on LIME and its properties:

Original LIME paper: M. T. Ribeiro, S. Singh, and C. Guestrin. Why should I trust you? Explaining the predictions of any classifier. In SIGKDD, 2016.

Some of our work:

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 tabular data

LIME = Local Interpretable Model-agnostic Explanations.

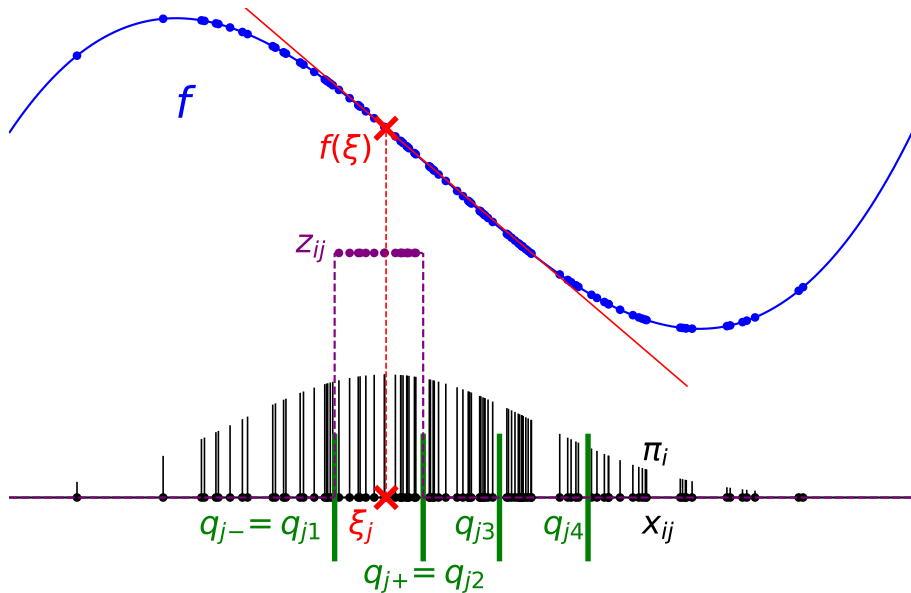
- ▶ Assume we are in a high-dim Euclidean feature space. We have learned a complex function (e.g, random forest; deep network).
- ▶ We obviously cannot globally replace the complicated function by a simple function that is based on very few “explainable” features and use this interpretable model instead (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:

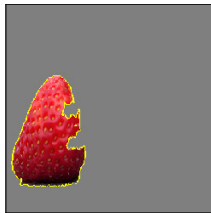
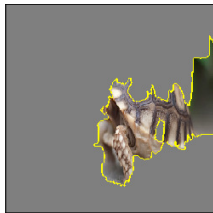
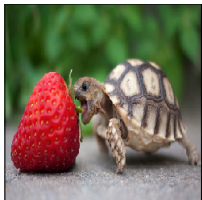
- ▶ Fix a point  $x$  for which you want to explain the decision of a complicated function  $f$
- ▶ Sample points  $x_1, \dots, x_m$  locally around  $x$ , evaluate the function  $f(x_1), \dots, f(x_m)$ , and apply a binning procedure.
- ▶ Locally approximate the complicated decision function  $f$  by 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  $(\text{in red})$ , we want to build a local model for  $f$  (in blue), given

# Local post-hoc explanations: LIME on images



(b) explaining 'terrapien' (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.

## Criticism, discussion about local posthoc explanations

Literature on manipulation:

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

More general:

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

## Explanations can be manipulated

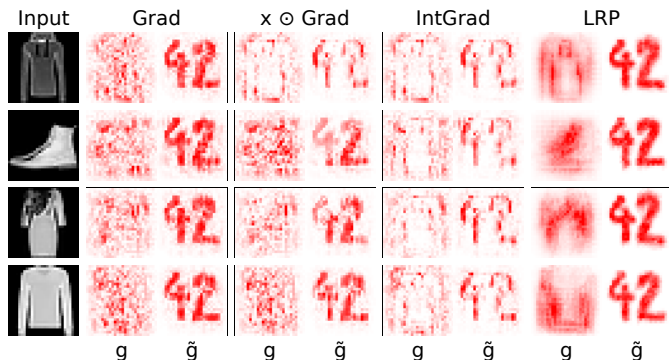
Explanation algorithms can easily be tricked or manipulated or cheated upon. The main idea is as follows:

- ▶ Assume we are given a function  $f$ , and we are supposed to provide local explanations for  $f$ . We now construct a function  $\tilde{f}$  that makes the same decisions as  $f$  on the data, but off-distribution behaves very differently.
- ▶ Real-world data often lives on a low-dimensional manifold. On this manifold, we keep the decision function  $f$  as it is. But locally around each data point there are many off-manifold points.
- ▶ An algorithm that creates explanations for  $x_0$  typically samples from a neighborhood of  $x_0$ , often points that are off-manifold.
- ▶ So by systematically changing  $\tilde{f}$  off-manifold, we can change the explanations (eg the gradients of a function), and thus the explanation at point  $x_0$



## Explanations can be manipulated (2)

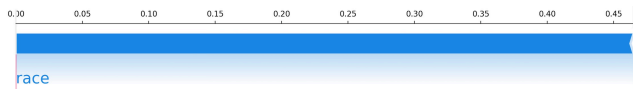
Example for manipulated gradient-based explanations, always outputting an image of 42 as gradient explanation:



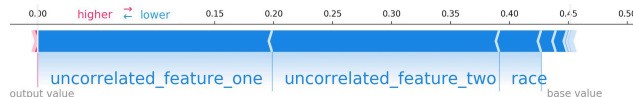
Anders et al: Fairwashing explanations with off-manifold detergent. ICML 2020.

## Explanations can be manipulated (3)

Example of manipulated SHAP explanations: a function  $f$  that heavily depends on feature “race” (Figure (a)) gets manipulated such that it looks innocent (Figure (b)), reporting other features as important:



(a) Example SHAP explanation for biased classifier  $f$

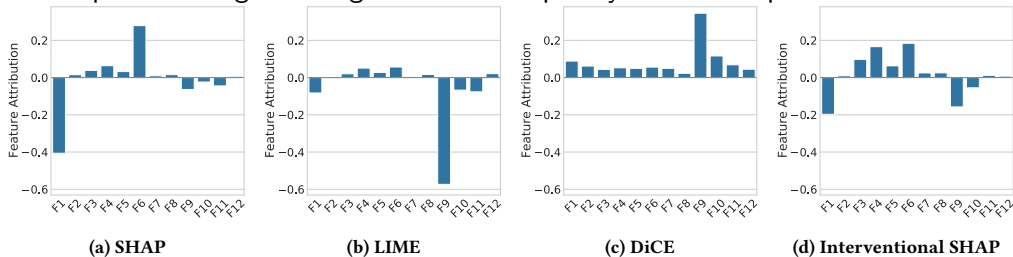


(b) SHAP explanation of the same instance after the attack (e)

D. Slack et al: Fooling lime and shap: Adversarial attacks on post hoc explanation methods. IAES 2020

# Different explanation algorithms give completely different explanations

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. 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.

## Informative explanations only exist for simple functions

- ▶ More generally, we have developed a framework when a local explanation is “informative”.
- ▶ We can prove that informative explanations only exist for simple functions, questioning the whole point of local posthoc-explanations...

Informative Post-Hoc Explanations Only Exist for Simple Functions. Günther, Szabados, Bhattacharjee, Bordt, Luxburg, Arxiv 2025.

## Explanation cards

Based on all the negative results on local posthoc explanations, there is one constructive way to go forward: Explanation cards.

- ▶ Explanation Cards augment standard explanations with complementary information about robustness and validity, as well as clear instructions for interpretation.
- ▶ The complementary information can render otherwise uninformative explanations practically useful, while also helping to detect cases where they are not.
- ▶ The interpretation instructions in explanation cards shift responsibility from users to providers: Rather than expecting users to recognize what can and can- not be concluded from an explanation, providers must make this explicit upfront.

Eric Günther, Balázs Szabados, Gunnar König, Kristof Meding, Sebastian Bordt, Ulrike von Luxburg: We Need Explanation Cards to Connect Explanation Algorithms to the Real World. Arxiv. 2026

## Summary

My personal take:

If the explanations are designed for a different end users, supply them with explanation cards.

If it is important to get reliable explanations (for example, in a societal context), attempt to use interpretable algorithms, do not allow for complicated models with post-hoc explanations.

Other people have different opinions here...

## Outlook: explaining image classifiers

- ▶ There are many methods to explain image classifiers, many of them are based on gradients.
- ▶ General idea: compute the gradient of the output function w.r.t. the input pixels. This can show which of the pixels most affect the prediction.

Some references:

- ▶ Saliency maps: Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, Simonyan, et al, ICLR Workshop paper, 2014
- ▶ Integrated gradients: Axiomatic Attribution for Deep Networks. Sundararajan et al, ICML, 2017
- ▶ Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization Selvaraju et al., ICCV, 2017
- ▶ SmoothGrad: removing noise by adding noise Smilkov et al. , arXiv, 2017

## Outlook: language models

Many people try to ask language models to explain themselves or explain their reasoning ... I am not so convinced ... and definitely not trustworthy if an explanation is really important.

Vastly open field ...



## Outlook: mechanistic interpretability of complex architectures

There exist approaches to understand the architecture of a particular, trained neural network. People try to extract what sub-components of the network really do and relate it to the task at hand. For example, one tries to identify certain sub-tasks that are performed by particular attention heads in a complex transformer architecture. Some small successes, but in the end big questions: can the approach be generalized beyond toy applications? And how trustworthy are the findings?

# Legal issues

## Reasons for regulation of AI systems

- ▶ Safety and harm prevention: AI can cause accidents, discrimination, etc at scale
- ▶ Accountability: who is liable when an AI system causes harm?
- ▶ Fundamental rights: privacy, non-discrimination
- ▶ Market integrity: consumer protection, level playing field for all actors
- ▶ Trust: public acceptance of AI requires oversight and legal certainty

## General comments: how does legislation work?

- ▶ Law operates at a high level: sets norms and principles, not technical specifications
- ▶ Does not define *how* to comply; this is left to standards bodies, industry guidelines, and case law
- ▶ Many key concepts deliberately vague (e.g., what does “transparent” mean for an ML model?)
- ▶ Interpretation is refined over time through court rulings and decisions by national regulatory bodies
- ▶ Compliance is an ongoing process, not a one-time checklist

# European AI Act

# The EU AI Act

- ▶ World's first comprehensive AI regulation
- ▶ Applies to providers, deployers, importers of AI systems placed on the EU market or affecting EU persons
- ▶ Goals: ensure AI is safe, transparent, non-discriminatory; foster innovation via legal certainty
- ▶ Timeline: proposed Apr 2021  $\leadsto$  political agreement Dec 2023  $\leadsto$  in force Aug 2024  $\leadsto$  stepwise roll-out until Aug 2026

# Reading the EU AI Act

- ▶ Full text of the regulation, it is worth looking at it:  
<https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689>
- ▶ The first part of this document contains the “Recitals”. These numbered paragraphs explain the background, motivation, aims, and interpretation of the law. They are not binding.
- ▶ The actual legally binding part starts somewhere in the middle (search for “Chapter I: General provisions”)

## Risk-based approach

The AI act does not regulate “algorithms”, but their “applications”. Applications are categorized into different risk categories:

- ▶ **Unacceptable risk (prohibited)**: real-time biometric surveillance in public spaces, social scoring by governments, subliminal manipulation, exploitation of vulnerable groups
- ▶ **High risk**: safety-critical systems (medical devices, critical infrastructure, autonomous vehicles) and high-impact systems (credit scoring, recruitment, education, law enforcement, border control)
- ▶ **Limited risk** (eg chatbots, deepfakes) and **minimal risk** (recommender systems, spam filters, games); little or no mandatory obligations

The major part of the AI Act discusses systems with high risk.



# Obligations for high-risk systems

- ▶ Risk management system throughout the entire lifecycle
- ▶ Data governance: high-quality training, validation, and test data; documented data practices
- ▶ Technical documentation and automatic logging of system behavior
- ▶ Transparency toward deployers; instructions for use
- ▶ Human oversight: system must be designed so humans can intervene or override

# General purpose AI systems / foundation models

- ▶ Got introduced just in the last round of debates of the AI Act (because of ChatGPT entering the market; circumventing all the discussions that had happened before).
- ▶ Definition in the legal text (Article 3(63)): “an AI model that is trained with a large amount of data using self-supervision at scale, that displays significant generality and is capable of competently performing a wide range of distinct tasks regardless of the way the model is placed on the market and that can be integrated into a variety of downstream systems or applications, except AI models that are used for research, development or prototyping activities before they are placed on the market.”
- ▶ The law explicitly mentions a threshold when systems start having systemic risk: training compute  $> 10^{25}$  FLOPs (approximately GPT-4 scale and above).
- ▶ All GPAI providers must: provide technical documentation, comply with copyright law, publish a summary of training data
- ▶ Additional obligations for systemic-risk models:
  - ▶ Adversarial testing / red-teaming before release

## General purpose AI systems / foundation models (2)

- ▶ Report serious incidents to the European Commission
- ▶ Cybersecurity measures; energy efficiency reporting

# European GDPR

# What is the GDPR?

- ▶ General Data Protection Regulation
- ▶ EU regulation, in force since May 2018
- ▶ Regulates the processing of *personal data* relating to natural persons.
- ▶ Applies to any entity processing personal data of EU residents, regardless of where the entity is located
- ▶ Relevance for ML: training data frequently contains personal data

# Data consent

- ▶ Specifies what are lawful reasons for processing personal data: consent, contract, legal obligation, legitimate interest, . . .
- ▶ Special categories (health, biometrics, ethnicity, political opinions, . . . ) require *explicit* consent
- ▶ Data minimization: collect only what is necessary for the stated purpose
- ▶ Purpose limitation: data may not be reused beyond original purpose
- ▶ **ML implication: scraping publicly available data does not automatically grant permission to use it for training!**

## Right to be forgotten

- ▶ Individuals can request erasure of their personal data
- ▶ **ML challenge: once data is encoded in model weights, deletion is non-trivial**
- ▶ Active research area — *machine unlearning*: removing the influence of specific training points from a model

# European laws around intellectual properties and copyright



# Data used for training

- ▶ **EU Copyright Directive** (2019)
- ▶ Using any type of data requires a legal basis! Could be a license, a setup as public domain (eg creative commons), you own data, there exists a data sharing agreement, ...
- ▶ GDPR applies additionally whenever training data contains personal information
- ▶ Scraping without permission increasingly contested, for example in the law cases between the *New York Times v. OpenAI*, *Getty Images v. Stability AI*

# Intellectual property issues

- ▶ **Ownership of AI-generated output:** current consensus: no copyright without human authorship; AI has no legal personhood
- ▶ **Training as copyright infringement:** active legal dispute; courts in US and EU have not yet settled this
- ▶ **Trade secrets:** model weights and training pipelines can be protected as proprietary IP
- ▶ **Patents:** AI cannot be listed as inventor

## Regulation outside the EU

## Regulation in other countries?

- ▶ **UK:** no new dedicated AI law; existing sector regulators cover many AI cases
- ▶ **Canada:** currently relying on existing privacy, consumer protection, human rights and sector-specific rules (there was an attempt to create AI regulation, but it got stopped 2025).
- ▶ **USA:** no federal AI law; some sector-specific rules apply, for example for medical AI, or for consumer protection, for discrimination, ...
- ▶ **China:** Algorithmic Recommendation Provisions (2022); Generative AI Regulation (2023); emphasis on content control and alignment with state interests

In general, no global consensus regarding regulation.

## Key takeaways

- ▶ **Know your data:** legal basis, data provenance, personal data, ...
- ▶ **Know your system:** determine in which risk category it is under the AI Act ~→ compliance obligations
- ▶ **Know your outputs:** generated content carries copyright risk for the deployer

The legal landscape is rapidly evolving. Landmark court decisions and implementing regulations still pending!