

Machine Learning: Algorithms and Theory

Ulrike von Luxburg

Summer 2018

Department of Computer Science, University of Tübingen

(Version as of May 29, 2018)

Contents will be updated constantly, see course webpage.

Table of contents

Introduction to Machine Learning

What is machine learning?	16
Motivating examples and applications	17
Organisation of the course	38
Machine learning as inductive inference	40
Different learning scenarios	65
Recap: Linear algebra, probability theory	73
Warmup: k -nearest neighbor classification	76
The kNNalgorithm for classification	77
Formal setup	104

Table of contents (2)

Standard setup for supervised learning	105
Statistical and Bayesian Decision theory	117
Optimal prediction functions in closed form	133
... for classification under 0-1 loss	134
... for regression under L_2 loss	146
Basic learning principles: ERM, RRM	153

Linear methods for supervised learning

Linear methods for regression	169
Linear least squares regression	170
Feature representation of data	195
Least squares with linear combination of basis functions	205
Ridge regression: least squares with L_2 -regularization	213
Lasso: least squares with L_1 -regularization	231
(*) Probabilistic interpretation of linear regression	248

Table of contents (3)

Feature normalization	254
Selecting parameters by cross validation	257
Linear methods for classification	268
Intuition and feature representation	269
(*) Linear discriminant analysis	276
Excursion: Probabilistic interpretation of ERM	299
Logistic regression	306
Linear Support vector machines	320
Intuition and primal	321
Excursion: convex optimization, primal, Lagrangian, dual	346
Deriving the dual problem	347
Important properties of SVMs	360
Kernel methods for supervised learning	
Positive definite kernels	370

Table of contents (4)

Intuition	371
Definition and properties of kernels	380
Reproducing kernel Hilbert space and feature maps	403
Support vector machines with kernels	425
Regression methods with kernels	444
Kernelized least squares	445
Kernel ridge regression	453
How to center and normalize in the feature space	457

More supervised learning algorithms

(*) Random Forests: SKIPPED	470
(*) Boosting: SKIPPED	471

Unsupervised learning

Dimensionality reduction and embedding	473
--	-----

Table of contents (5)

PCA and kernel PCA	474
Multi-dimensional scaling	524
Graph-based machine learning algorithms: introduction	541
Isomap	548
(*) Maximum Variance Unfolding: SKIPPED	563
(*) Johnson-Lindenstraus: SKIPPED	564
(*) Ordinal embedding (GNMDS, SOE, t-STE): SKIPPED	565
Clustering	566
K-means and kernel k-means	576
Standard k -means algorithm	577
Kernel k -means	600
Linkage algorithms for hierarchical clustering	604
A glimpse on spectral graph theory	613
Unnormalized Laplacians	615
Normalized Laplacians	629

Table of contents (6)

Cheeger constant and isoperimetric problems	635
Spectral clustering	644
Unnormalized spectral clustering	654
Normalized spectral clustering	677
(*) Correlation clustering: SKIPPED	684

Introduction to learning theory

The standard theory for supervised learning	686
Learning Theory: setup and main questions	687
Controlling the estimation error: generalization bounds	696
Capacity measures for function classes	710
Finite classes	711
Shattering coefficient	719
VC dimension	729
Rademacher complexity	741

Table of contents (7)

Controlling the approximation error	747
Getting back to Occam's razor	755
(*) Loss functions, proper and surrogate losses: SKIPPED	764
(*) Probabilistic interpretation of ERM: SKIPPED	765
(*) The No-Free-Lunch Theorem: SKIPPED	766

Low rank matrix methods

Introduction: recommender systems, collaborative filtering	779
Matrix factorization basics	783
Low rank matrix completion	791
Compressed sensing	831

Ranking from pairwise comparisons

Table of contents (8)

Introduction	874
Simple but effective counting algorithm	883
Learning to rank	902
Application: distance completion problem	913
Spectral ranking	925
Google page rank	929

The data processing chain: from raw data to machine learning

Preparing the data	947
Data aquisition: train versus test distribution	948
Converting raw data to training data	953
Data cleaning: missing values, outliers	955
Defining features, similarities, distance functions	963
Defining a similarity / kernel / distance function	968
Reducing the number of training points?	974

Table of contents (9)

Unsupervised dimensionality reduction	978
Data standardization	981
Clustering	983
Setting up the learning problem	985
Choice of a loss / risk function	986
Training	994
Selecting parameters by cross validation	997
Feature selection	1008
Evaluation of the results	1024
General guidelines for attacking ML problems	1055
Key steps in the processing chain	1056
High-level guidelines and principles	1059

Table of contents (10)

Online learning

Online learning	1062
Warmup	1063
Prediction with expert advice, weighted majority algorithm	1073
Follow the (perturbed,regularized) leader	1087
Perceptron and winnow	1091

Outlook: some of the research in our group

Wrap up

Excursion: Research, publications, reviewing	1101
Things we did not talk about	1113
Further machine learning resources	1115

Table of contents (11)

Mathematical Appendix

Recap: Probability theory	1121
Discrete probability theory	1122
Continuous probability theory	1160
Recap: Linear algebra	1174
Excursion to convex optimization: primal, dual, Lagrangian	1213
Convex optimization problems: intuition	1214
Lagrangian: intuitive point of view	1221
Lagrangian: formal point of view	1242

Literature

Here are some of the books we use during the lecture (just individual chapters):

General machine learning:

- ▶ *Shai Shalev-Shwartz, Shai Ben-David: Understanding Machine Learning. Cambridge University Press 2014.* Theory and algorithms.
- ▶ *Chris Bishop: Pattern recognition and Machine Learning. Springer 2006.* Focus on algorithms.
- ▶ *Hastie, Tibshirani, Friedman: Elements of statistical learning. Springer, 2009.* More from a traditional statistics point of view.
- ▶ *Mohri, Rostamizadeh, Talwalkar: Foundations of machine learning. MIT Press, 2012.* More theory than algorithms.

Literature (2)

- ▶ *Schölkopf, Smola: Learning with kernels. MIT Press, 2002.* SVMs and kernel algorithms, for computer science audience.
- ▶ *Steinwart, Christmann: Support Vector Machines. Springer, 2008.* SVMs and kernel algorithms, for maths audience.

Learning theory:

- ▶ *Devroye, Györfi, Lugosi: A Probabilistic Theory of Pattern Recognition. Springer, 1996.*
Covers all the basic results of statistical learning theory, but we won't use much of it in this lecture.

Low rank matrix methods:

- ▶ *Hastie, Tibshirani, Wainwright: Statistical learning with sparsity. CRC Press, 2015.* In depth.

The Bayesian / probabilistic viewpoint:

Literature (3)

- ▶ Kevin Murphy: *Machine learning, a probabilistic perspective.* MIT Press, 2012.

Information theoretic (and Bayesian) point of view on machine learning:

- ▶ MacKay: *Information theory, inference and learning algorithms.* Cambridge University Press, 2003

Optimization:

- ▶ Boyd / Vandenberghe for convex optimization
- ▶ Nocedal / Wright: Numerical optimization (focus on algorithms)

Introduction to Machine Learning

What is machine learning?

Motivating examples and applications

Hand-written digit recognition

Want to automatically recognize the postal code in the address field of a letter:

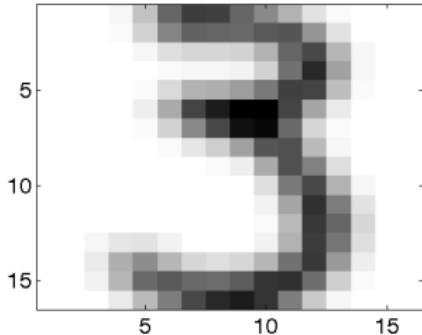
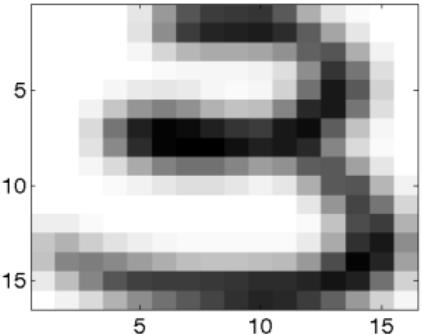
Fachbereich Informatik

Saud 14

72076 Tübingen

Hand-written digit recognition (2)

- ▶ Take a camera picture of the address
- ▶ Segment the image into individual letters and digits
- ▶ Image of a digit: 16×16 greyscale image, corresponds to a vector with 256 entries, each entry between 0 and 1 (0 = white, 1 = black)



- ▶ Goal: want to know which digits they are, that is we want to find a “correct” mapping $f : [0, 1]^{256} \rightarrow \{0, 1, 2, \dots, 9\}$.

Hand-written digit recognition (3)

- ▶ Problem: it is impossible to hand-design such a rule!

5 5 5 5 5 5 5

9 9 9 9 9 9 9

7 7 7 7 7 7 7

1 1 1 1 1 1 1

Hand-written digit recognition (4)

The machine learning approach:

- ▶ Present many “training examples” to the computer:
 $(X_i, Y_i)_{i=1, \dots, n}$ such that X_i = greyscale image,
 $Y_i \in \{0, 1, 2, \dots, 9\}$ the true class label
- ▶ The computer is supposed “to learn” a function f that correctly assigns digits to greyscale images

This problem is one of the “founding problems” of pattern recognition and machine learning.

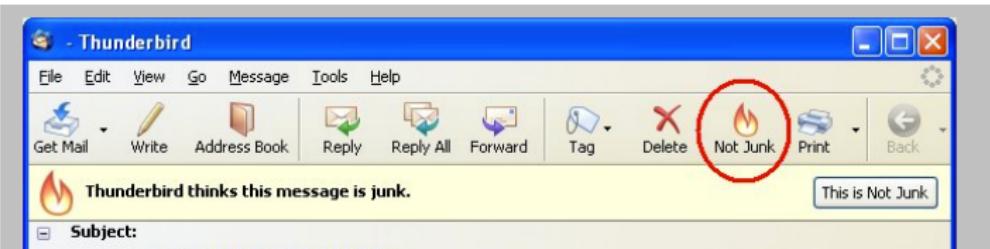
Object detection in general



Image: Christian Wojek et al, IEEE PAMI, 2013

Spam filtering

- Want to classify all emails into two classes: spam or not-spam
- Similar problem as above: hand-designed rules don't work in many cases
- So we are going to "train" the spam filter:



Spam filtering (2)

- ▶ Internally, the spam filter “updates its rules” based on the training example it gets.

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

Self-driving car

First breakthrough: The DARPA grand challenge in 2005:

- ▶ Build an autonomous car that can find its way 100 km through the desert.



© DARPA

- ▶ The winning team was the one of Sebastian Thrun (Stanford), one of the leading machine learning researchers.

Self-driving car (2)

Now: the google car:



Self-driving car (3)

How can it work?

- ▶ Many sensors, cameras, etc
- ▶ All of them generate low-level signals
- ▶ Need to “extract information” from each signal such as
 - ▶ “here is a wall / a car / a pedestrian”
 - ▶ “the cyclist extends his arm to signal a turn”
 - ▶ “Railways cross the street”
- ▶ Lots of information, noisy, unreliable, contradicting. Need to aggregate / combine this information.
- ▶ Need to predict what happens next (“cyclist wants to turn left”)
- ▶ Then the car can “decide” what to do next.

It took decades to work out all these steps, machine learning is the driving force behind the recent success.

Bioinformatics

Machine learning is used all over the place in bioinformatics:

- ▶ Classify different types of diseases based on microarray data

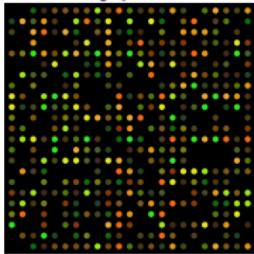
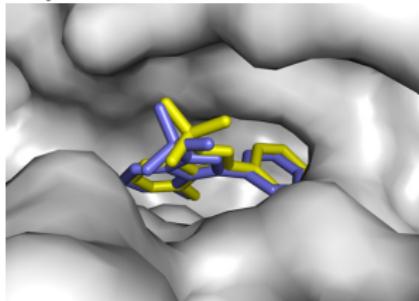
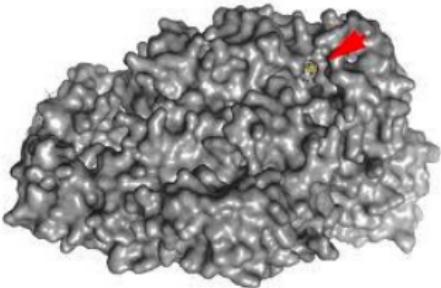


Image: Wikipedia

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



Images: BiochemLabSolutions.com

Bioinformatics (2)

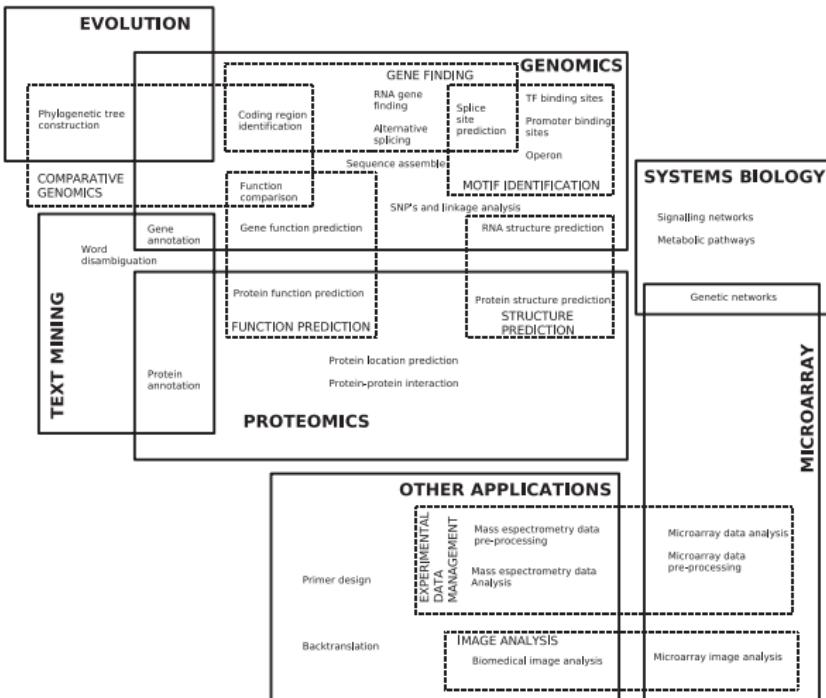
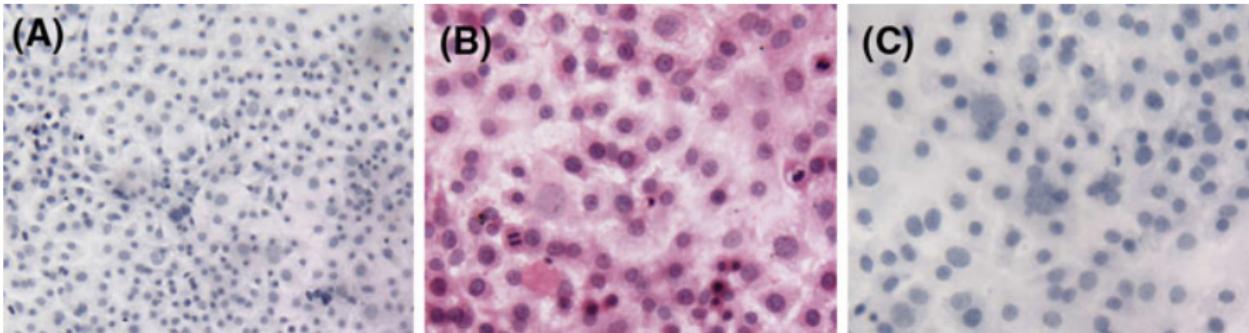


Figure from Larranaga et al., 2005

Medical image analysis

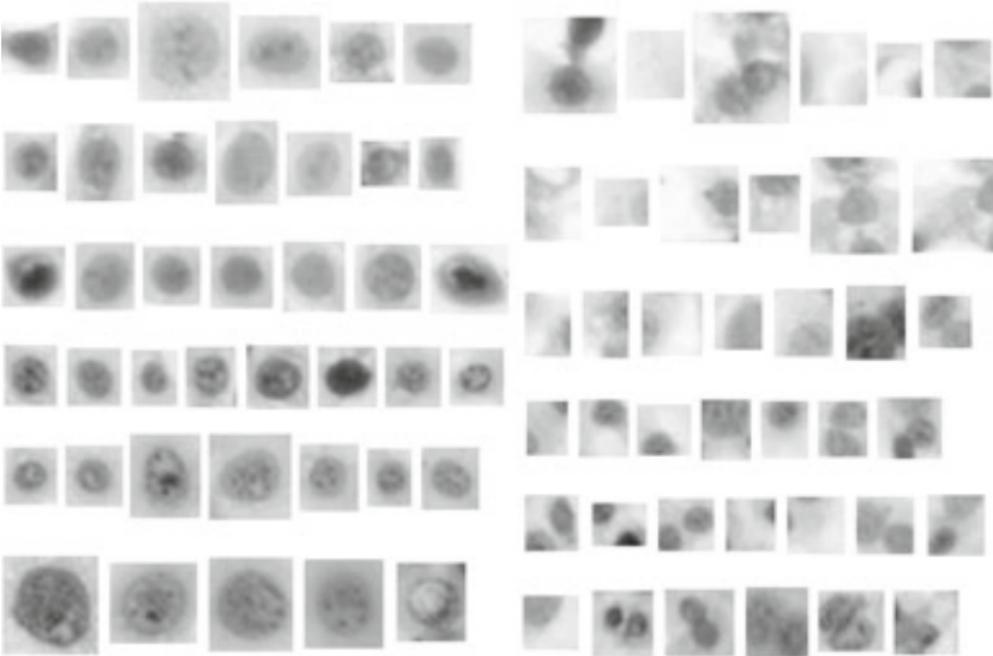
Example from cancer research: automatic detection and classification of cell nuclei in microscopy images:

Images look like this:



Medical image analysis (2)

The training data:

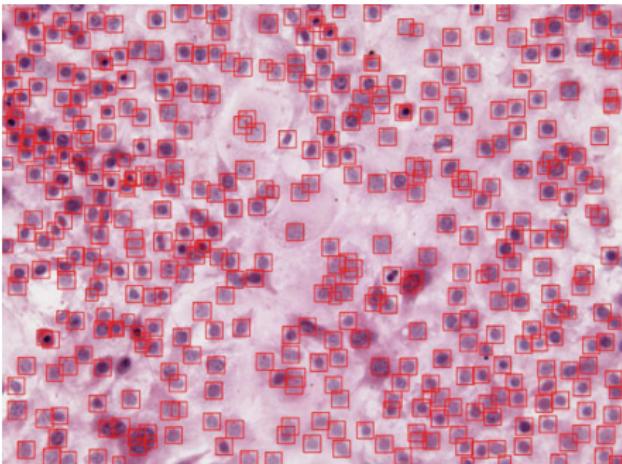


Positive training samples

Negative training samples

Medical image analysis (3)

The results:



Reference: The Application of Support Vector Machine Classification to Detect Cell Nuclei for Automated Microscopy (J.W. Han, T.P. Breckon, D.A. Randell, G. Landini), In Machine Vision and Applications, Springer, Volume 23, No. 1, pp. 15-24, 2012.

Language processing

2011: Computer “Watson” wins the american quiz show “jeopardy”. It is a bit like “Wer wird Millionär”, but not so much about facts and more about word games.



Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations

Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations
 - ~~~ eavesdropping

Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations
~~~ **eavesdropping**
- ▶ It's a poor workman who blames these

# Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations  
~~~ eavesdropping
- ▶ It's a poor workman who blames these ~~ tools

Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations
~~> eavesdropping
- ▶ It's a poor workman who blames these ~~> tools
- ▶ The USPS cost for mailing this, a minimum of $3\frac{1}{2} \times 5$ inches, is 28 cents; wish you were here!

Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations
~~> eavesdropping
- ▶ It's a poor workman who blames these ~~> tools
- ▶ The USPS cost for mailing this, a minimum of $3\frac{1}{2} \times 5$ inches, is 28 cents; wish you were here! ~~> postcard

Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations
~~> eavesdropping
- ▶ It's a poor workman who blames these ~~> tools
- ▶ The USPS cost for mailing this, a minimum of $3\frac{1}{2} \times 5$ inches, is 28 cents; wish you were here! ~~> postcard
- ▶ There are about 50 species of the hedgehog type of this plant, so named for its spiny fruit

Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations
~~> eavesdropping
- ▶ It's a poor workman who blames these ~~> tools
- ▶ The USPS cost for mailing this, a minimum of $3\frac{1}{2} \times 5$ inches, is 28 cents; wish you were here! ~~> postcard
- ▶ There are about 50 species of the hedgehog type of this plant, so named for its spiny fruit ~~> cactus

Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations
~~~ eavesdropping
- ▶ It's a poor workman who blames these ~~> tools
- ▶ The USPS cost for mailing this, a minimum of  $3\frac{1}{2} \times 5$  inches, is 28 cents; wish you were here! ~~> postcard
- ▶ There are about 50 species of the hedgehog type of this plant, so named for its spiny fruit ~~> cactus
- ▶ Milorad Cavic almost upset this man's perfect 2008 Olympics, losing to him by one hundredth of a second

# Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations  
~~~ eavesdropping
- ▶ It's a poor workman who blames these ~~> tools
- ▶ The USPS cost for mailing this, a minimum of $3\frac{1}{2} \times 5$ inches, is 28 cents; wish you were here! ~~> postcard
- ▶ There are about 50 species of the hedgehog type of this plant, so named for its spiny fruit ~~> cactus
- ▶ Milorad Cavic almost upset this man's perfect 2008 Olympics, losing to him by one hundredth of a second ~~> Michael Phelps

Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations
~~~ **eavesdropping**
- ▶ It's a poor workman who blames these ~~~ **tools**
- ▶ The USPS cost for mailing this, a minimum of  $3\frac{1}{2} \times 5$  inches, is 28 cents; wish you were here! ~~~ **postcard**
- ▶ There are about 50 species of the hedgehog type of this plant, so named for its spiny fruit ~~~ **cactus**
- ▶ Milorad Cavic almost upset this man's perfect 2008 Olympics, losing to him by one hundredth of a second ~~~ **Michael Phelps**
- ▶ A piece of wood from a tree, or to puncture with something pointed

# Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations  
    ~> **eavesdropping**
- ▶ It's a poor workman who blames these ~> **tools**
- ▶ The USPS cost for mailing this, a minimum of  $3\frac{1}{2} \times 5$  inches, is 28 cents; wish you were here! ~> **postcard**
- ▶ There are about 50 species of the hedgehog type of this plant, so named for its spiny fruit ~> **cactus**
- ▶ Milorad Cavic almost upset this man's perfect 2008 Olympics, losing to him by one hundredth of a second ~> **Michael Phelps**
- ▶ A piece of wood from a tree, or to puncture with something pointed ~> **stick**

# Language processing (2)

Example questions:

- ▶ One definition of this is entering a private place with the intent of listening secretly to private conversations  
    ~> **eavesdropping**
- ▶ It's a poor workman who blames these ~> **tools**
- ▶ The USPS cost for mailing this, a minimum of  $3\frac{1}{2} \times 5$  inches, is 28 cents; wish you were here! ~> **postcard**
- ▶ There are about 50 species of the hedgehog type of this plant, so named for its spiny fruit ~> **cactus**
- ▶ Milorad Cavic almost upset this man's perfect 2008 Olympics, losing to him by one hundredth of a second ~> **Michael Phelps**
- ▶ A piece of wood from a tree, or to puncture with something pointed ~> **stick**

# Breakthroughs in recent years, based on deep learning

... in areas with large amounts of structured data:

- ▶ Machine translation
- ▶ Speech recognition
- ▶ Object recognition

# Another recent success story: Deep mind's AlphaGo

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



## Another recent success story: Deep mind's AlphaGo (2)

AlphaGo combines an advanced tree search with deep neural networks. These neural networks take a description of the Go board as an input and process it through 12 different network layers containing millions of neuron-like connections. We trained the neural networks on 30 million moves from games played by human experts. AlphaGo learned to discover new strategies for itself, by playing thousands of games between its neural networks, and adjusting the connections using a trial-and-error process known as reinforcement learning.

# Organisation of the course

- ▶ Info Sheet
- ▶ Important: doodle for tutorial groups
- ▶ Literature
- ▶ Material covered: see table of contents
- ▶ Material NOT covered: neural networks and deep learning;  
Bayesian / probabilistic approaches; reinforcement learning
- ▶ Language: english / german
- ▶ Prerequisites: maths, nothing else (see first exercises)

# Machine learning as inductive inference

# What is machine learning?

First explanation:

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

# Deduction vs. Induction

WHO KNOWS WHAT INDUCTION AND DEDUCTION MEAN?

## Deduction vs. Induction (2)

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

Example:

- ▶ Premise 1: every person in this room is a student.
- ▶ Premise 2: every student is older than 10 years.
- ▶ Conclusions: 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. For example, 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 throw lots of things, very often.
- ▶ In all our experiments, the things fell down and not up.
- ▶ So we conclude that likely, things always fall down.

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

## Deduction vs. Induction (4)

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

Example:

- ▶ You come 10 minutes late to every lecture I give.
- ▶ The first 7 times I don't complain.
- ▶ You conclude that I don't care and it won't have any consequences.
- ▶ BUT you cannot be sure ...

# Machine learning as inductive inference

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

**Machine learning tries to automate the process of inductive inference.**

## Machine learning as inductive inference (2)

In most applications of machine learning: focus is not so much on building models (“understanding the underlying process”) but more on predicting.

Example with the things that always fall down:

- ▶ We are not so much concerned *why* the stones always fall down and not up
- ▶ We just want to *predict* that they fall down

# Machine learning as inductive inference (3)

## Very important observation:

- ▶ Being able to predict is often easier than understanding the underlying mechanism.
- ▶ In particular, it is often not necessary to understand the underlying mechanism for being able to make good predictions!

### Example:

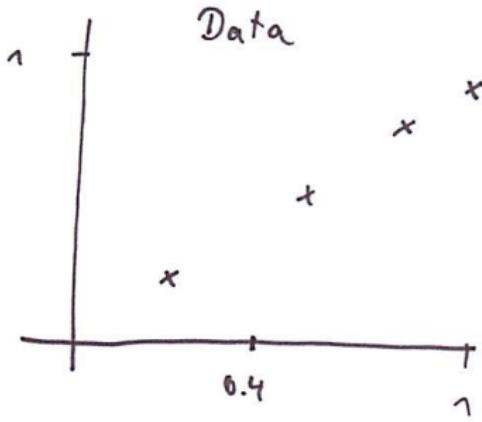
- ▶ We observe that the sun rises every day.
- ▶ So we predict that the sun is also going to rise the next day.
- ▶ To make this prediction, we don't need to understand why this is the case.

# Why should machine learning work at all?

Consider the following simple regression example:

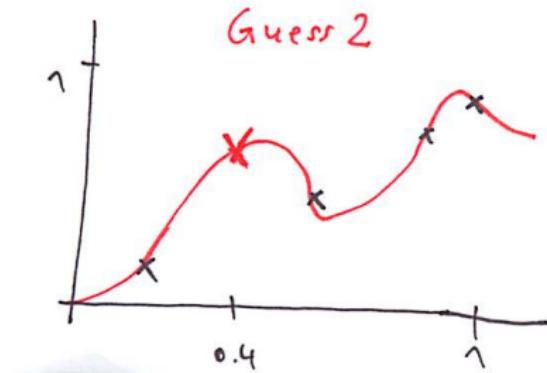
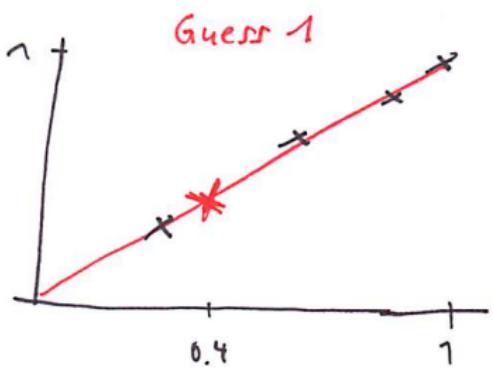
- ▶ Given: input-output pairs  $(X_i, Y_i)$ ,  $X_i \in \mathcal{X}$ ,  $Y_i \in \mathcal{Y}$ .
- ▶ Goal: learn to predict the  $Y$ -values from the  $X$ -values, that is we want to “learn” a suitable function  $f : \mathcal{X} \rightarrow \mathcal{Y}$ .

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



# Why should machine learning work at all? (2)

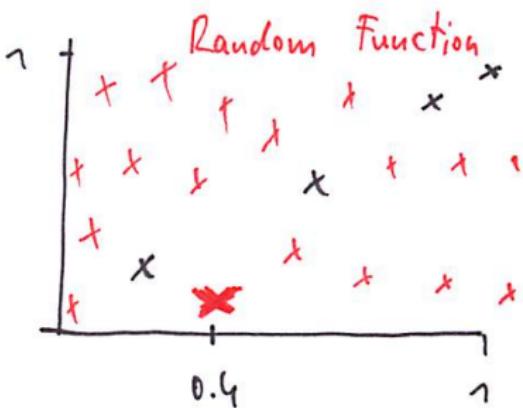
Here are two guesses:



WHICH ONE IS BETTER?

# Why should machine learning work at all? (3)

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



WHAT DO YOU PREDICT NOW?

# Why should machine learning work at all? (4)

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

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

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

# Why should machine learning work at all? (5)

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

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

# Inductive bias: very simple example

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

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

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

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

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

Formally:

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

## Inductive bias: very simple example (2)

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

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

## Inductive bias: very simple example (3)

A more formal way of stating this result is called the “No free lunch theorem”. There is a section on this topic in the slides on learning theory, you can read it if you want (we won’t discuss it in the lecture though).

# Inductive bias: very simple example (4)

## Case 2: model with an inductive bias.

- ▶ Assume that the true function is one out of two functions: either the constant one function 1 or the constant zero function 0.  
Our hypothesis space is  $\mathcal{F} = \{0, 1\}$ .
- ▶ Then, after we observed one training example, we know exactly which function is the correct one and can predict without error.

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

# Inductive bias: very simple example (5)

A bit too simplistic?

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

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

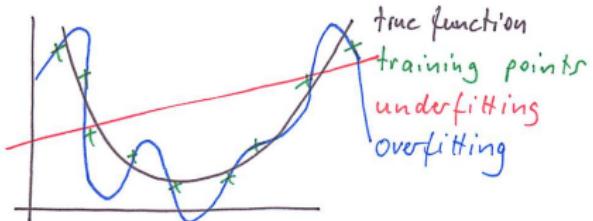
At the end of the course you will know all this, at least roughly ☺

# Overfitting, underfitting

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

Consider the following example:

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



# Overfitting, underfitting (2)

## Overfitting:

- ▶ We can always find a function that explains all training points very well or even exactly
- ▶ 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 ( $\rightsquigarrow$  see later for definitions)

## Underfitting:

- ▶ Model is too simplistic
- ▶ But estimated functions are stable with respect to noise
- ▶ Large approximation error, low estimation error ( $\rightsquigarrow$  see later for definitions)

# Excursion: Inductive bias in animal learning

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

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

Experiment 1:

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

# Excursion: Inductive bias in animal learning (2)

## Experiment 2:

- ▶ Same water, 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 accompanying audio-visual stimuli.
- ▶ In this setting, rats did NOT learn to avoid the water that makes them sick.
- ▶ Apparently, they cannot make a connection between “sound of the food” and “sickness”.

# Excursion: Inductive bias in animal learning (3)

## Explanation:

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

**In our words: the rat has an inductive bias!**

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

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

# Inductive bias, bottom line for now

**Any successful learning algorithm has an inherent inductive bias.**

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

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

# Different learning scenarios

# Supervised, semi-supervised, unsupervised learning

## Supervised learning:

- ▶ We are given input-output pairs  $(X_i, Y_i)$  as training data.
- ▶ The goal is to “learn” the function  $f : X_i \mapsto Y_i$ .
- ▶ “Supervised”: the teacher tells us what the true outcome should be on the given samples

The most important supervised learning tasks:

- ▶ **Classification:**  $Y_i \in \{0, 1\}$ , or  $Y_i \in \{1, 2, \dots, K\}$ .  
Applications: spam classification, digit recognition, ...
- ▶ **Regression:**  $Y_i \in \mathbb{R}$ .  
Example: predict how much a person is willing to pay for a laptop, based on his past shopping behavior

# Supervised, semi-supervised, unsupervised learning (2)

## Unsupervised learning:

- ▶ We are just given input values  $X_i$ , but no output values
- ▶ The learner is supposed to learn the “structure” of these inputs.

Examples:

- ▶ **Clustering:** partition the data into “meaningful groups”.

Applications:

- ▶ Different types of cancer based on gene-expression profiles
- ▶ Find communities in a social network
- ▶ Build a taxonomy of a document collection, based on the topics of the documents

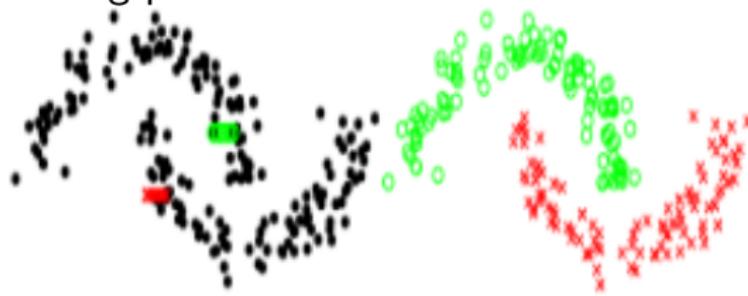
- ▶ **Outlier detection:** find data points that are “outliers”.

Application: intrusion detection, data preprocessing.

# Supervised, semi-supervised, unsupervised learning (3)

## Semi-supervised learning:

- ▶ We are given very many input values  $X_1, \dots, X_u$  (unlabeled points), and some input-output pairs  $(X_i, Y_i)$  ("labeled points").
- ▶ Goal is, as in supervised learning, to predict the function  $f : X_i \mapsto Y_i$ .
- ▶ But we want to exploit the extra knowledge we gain from the unlabeled training points.



# Supervised, semi-supervised, unsupervised learning (4)

- ▶ Applications:
  - ▶ Predict the function of certain proteins: lots of proteins are known (unlabeled points), but it is very expensive to run lab experiments to find out whether they have a certain functionality or not (labeled points)
  - ▶ Computer tomography: lots of images of patients are available, but only few of them are labeled by a medical doctor as to contain a tumor or not.

# Batch vs. Online vs. Active Learning

## Batch learning:

- ▶ We get a bunch of training data before we start the learning process.
- ▶ Then we learn on this whole training set.

## Online learning:

- ▶ Examples arrive online, one at a time.
- ▶ Each time a new training example arrives, we update our internal model.
- ▶ Prediction accuracy is measured in an online fashion as well (we have to predict constantly).
- ▶ Example: spam detection

# Batch vs. Online vs. Active Learning (2)

## Active learning:

- ▶ Instead of examples being given to us, we can actively choose which example should be the next one we want to learn (we can “actively ask” a teacher)
- ▶ Of course it is crucial to select a question that gives you as much information about the true underlying function as possible.
- ▶ Applications: domains where obtaining labels is expensive. Active learning is particularly interesting in combination with semi-supervised learning.

# Discriminative vs. generative approach

We distinguish two different machine learning scenarios:

- ▶ **Discriminative approach.** We are just interested in predicting the output variable  $Y$ .
- ▶ **Generative approach.** For some cases, we not only want to predict the labels, but we want to “understand” the underlying process, that is we want to model the joint distribution  $P(X, Y)$ , in particular the class conditional distributions  $P(X|Y = 1)$  and  $P(X|Y = 0)$ .

In general, solving discriminative problems is easier than generative problems.

## Recap: Linear algebra, probability theory

# Recap!

To be able to follow the lecture, you need to know the following material. If you cannot remember it, please recap. You can use the slides in the maths appendix, the tutorials, and any text book ...

Linear algebra:

- ▶ Vector space, basis, linear mapping, norm, scalar product
- ▶ Matrices: matrix multiplication, rank, inverse, eigenvalues and eigenvectors
- ▶ Eigen-decomposition of symmetric matrices, positive definite matrices

# Recap! (2)

## Probability theory:

- ▶ Discrete:
  - ▶ Basics: Probability distribution, conditional probability, Bayes theorem, random variables, expectation, variance, independence,
  - ▶ Distributions: joint, marginal, product distributions; Bernoulli, Binomial distribution, Poisson distribution, uniform distribution
  - ▶ multivariate random variables: variance, covariance
- ▶ Continuous (we keep it on the intuitive level):
  - ▶ Density, cumulative distribution function, expectation, variance
  - ▶ Normal distribution (univariate and multivariate), covariance matrix.

# Warmup: $k$ -nearest neighbor classification

# The kNN algorithm for classification

Literature:

Hastie, Tibshirani, Friedman Section 2.3.2

Duda, Hart Section 4.5

# A simple machine learning experiment

Data:

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

# A simple machine learning experiment (2)

Assume our machine learning algorithm has used the training data to construct a rule  $f_{alg}$  for predicting labels.

## Training error:

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

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

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

## A simple machine learning experiment (3)

- ▶ Define the training error of the classifier as the average error, over all training points:

$$\text{err}_{\text{train}}(f_{\text{alg}}) = \frac{1}{n} \sum_{i=1}^n \text{err}(X_i, Y_i, f_{\text{alg}}(X_i))$$

Remarks:

- ▶ This error obviously also depends on the training sample, but we drop this from the notation to make it more readable. To be really precise, one would have to write something like  $\text{err}_{\text{train}}(f_{\text{alg}}, (X_i, Y_i)_{i=1, \dots, n})$
- ▶ Later we will call this quantity the “empirical risk” of the classifier (with respect to the 0-1-loss).

# A simple machine learning experiment (4)

## Test error:

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

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

- ▶ Define the test error of the classifier as the average error, over all test points:

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

# A simple machine learning experiment (5)

Technical remarks:

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

# A simple machine learning experiment (6)

Remarks:

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

# The kNN classifier

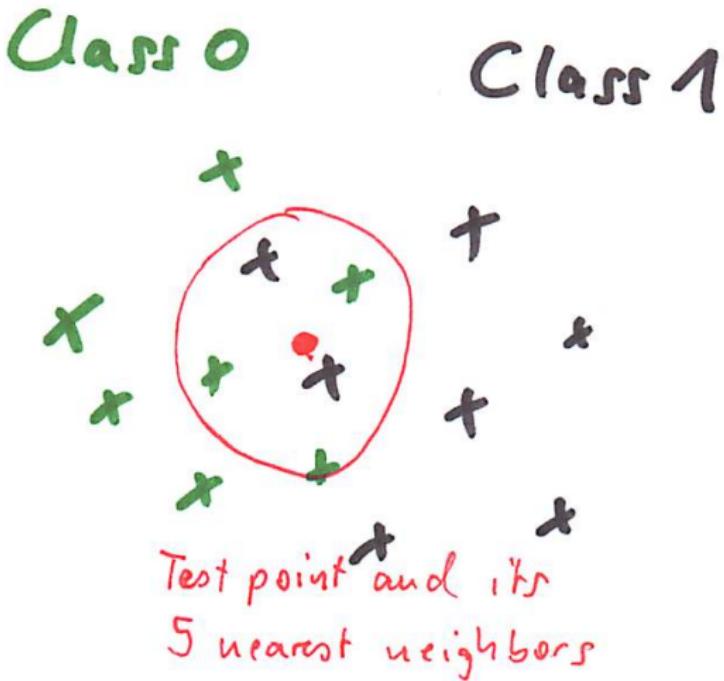
**Given:** Training points  $(X_i, Y_i)_{i=1,\dots,n} \subset \mathcal{X} \times \{0, 1\}$  and a distance function  $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ .

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

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

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

## Example



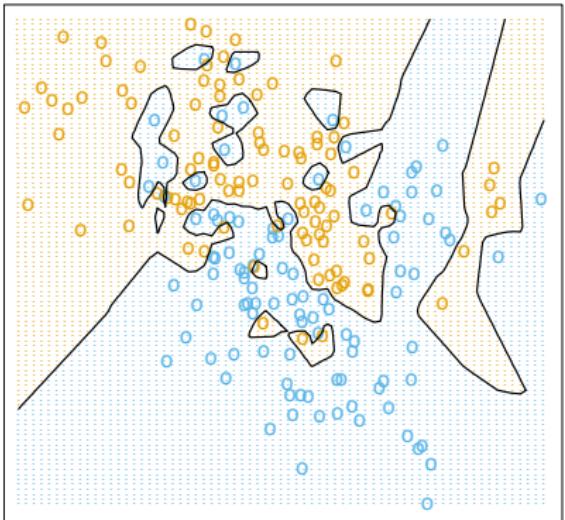
## Influence of the parameter $k$

The classification result will depend on the parameter  $k$ .

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

# Influence of the parameter $k$ (2)

1-Nearest Neighbor Classifier



15-Nearest Neighbor Classifier

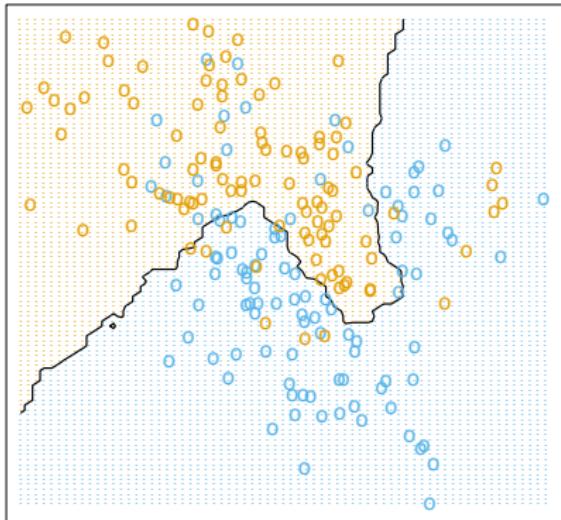


Figure from Hastie/Tibshirani/Friedman:

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

# Influence of the parameter $k$ (3)

Generally:

- ▶  $k$  too small  $\rightsquigarrow$  overfitting  
(Extreme case:  $k = 1$ , very wiggly and prone to noise, zero training error)
- ▶  $k$  too large  $\rightsquigarrow$  underfitting  
(Extreme case:  $k = n$ , then every point gets the same label, namely the overall majority label)
- ▶ Theoretical analysis can reveal:  $k$  should be roughly of order  $\log n$  as  $n \rightarrow \infty$  (we won't discuss it in this course though).

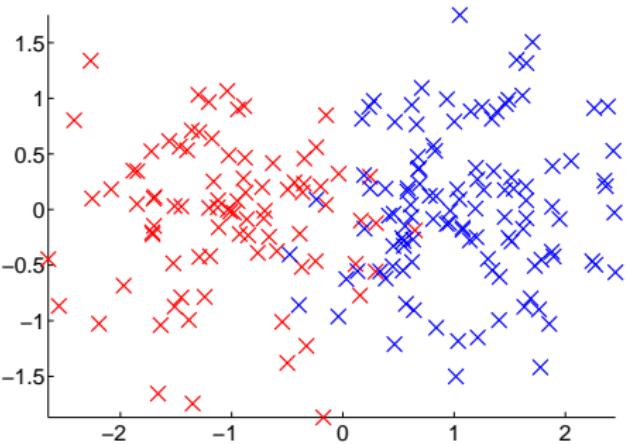
# Application: simple mixture of Gaussians

Recap:

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

## Application: simple mixture of Gaussians (2)

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



# Application: simple mixture of Gaussians (3)

Conduct the following experiment:

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

# Application: simple mixture of Gaussians (4)

Note:

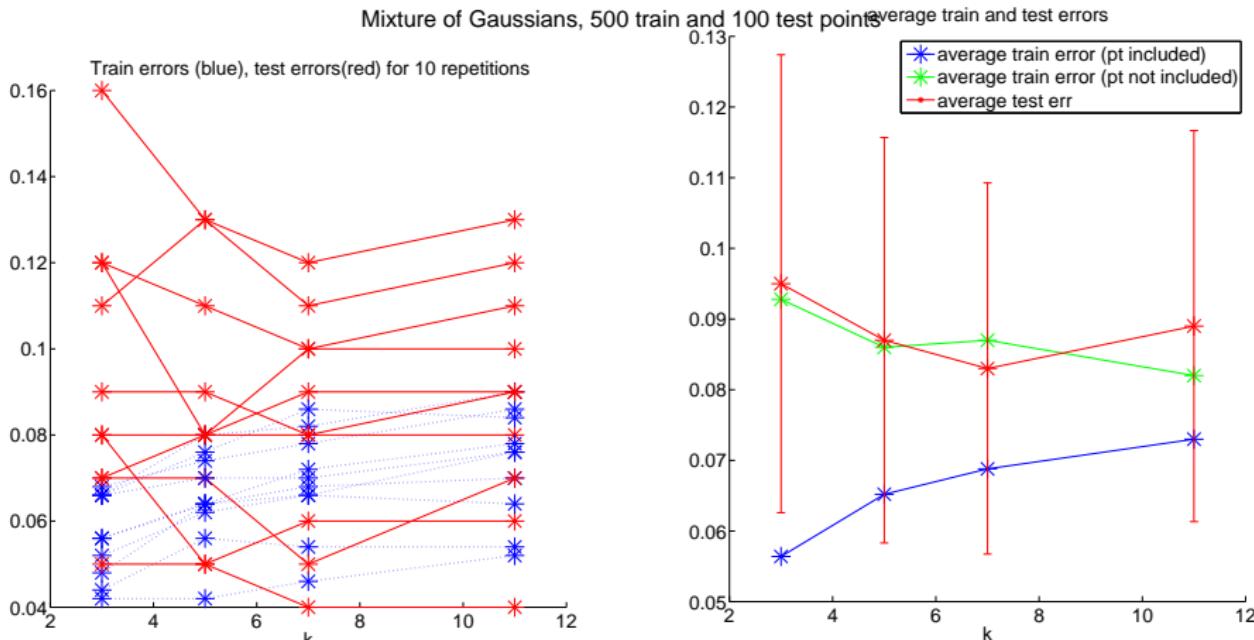
- ▶ For the kNN classifier, the training error and test error are about the same (it does not really “train” in the sense that it selects a function that is particularly good on the training data).
- ▶ Depending on whether a point is considered to be part of its own kNN neighborhood or not, the train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

## Application: simple mixture of Gaussians (5)

The following figure shows these train and test errors:

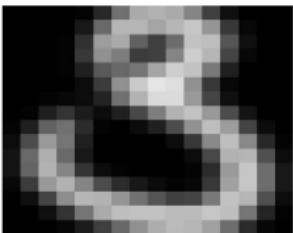
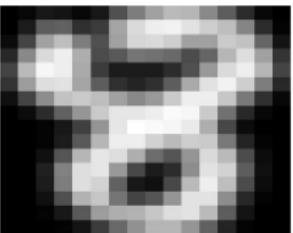
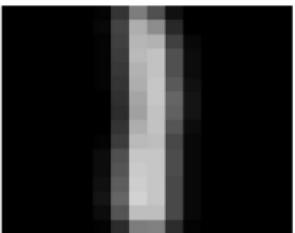
- ▶ Left figures: errors in each individual repetition
- ▶ Right figure: errors averaged over all repetitions

# Application: simple mixture of Gaussians (6)



# Application: hand written digits

Data:



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

Task 1: Learn to distinguish between 1 and 8

Task 2: Learn to distinguish between 3 and 8

# Application: hand written digits (2)

Setup:

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

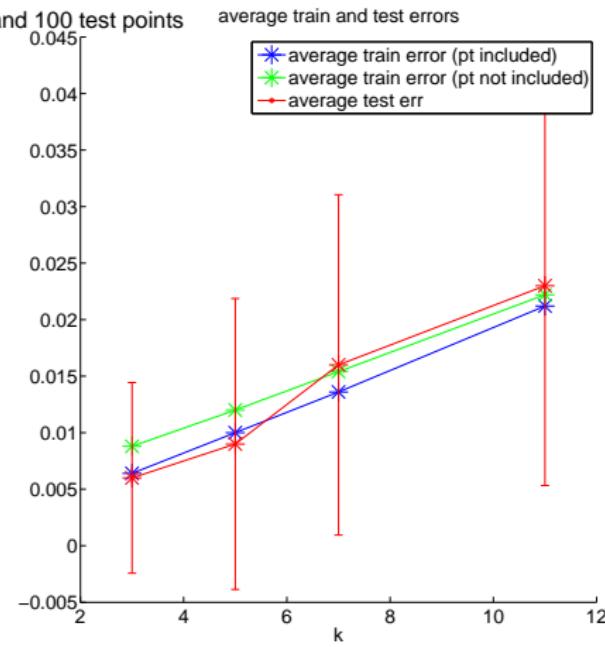
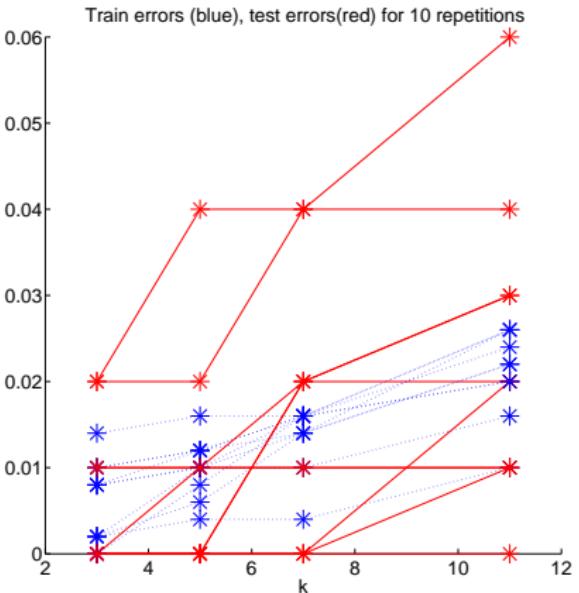
for  $X = (X_1, \dots, X_{256})^t$  and  $X' = (X'_1, \dots, X'_{256})^t$  we set

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

# Application: hand written digits (3)

## Results task 1 (digit 1 vs digit 8):

Digit 1 vs 8, 500 train and 100 test points



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

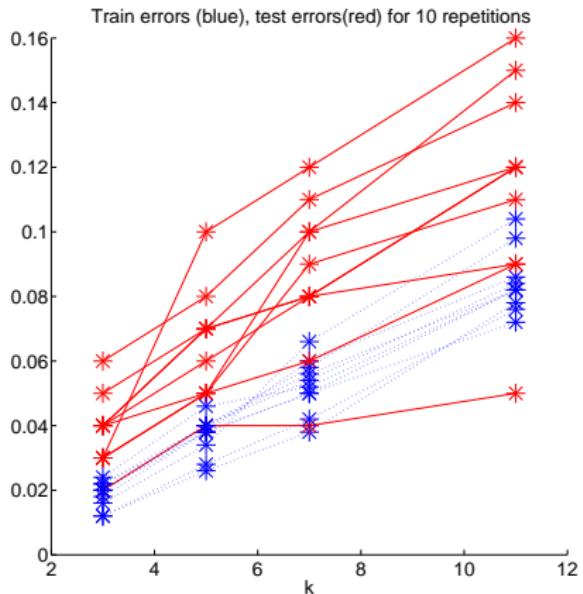
# Application: hand written digits (4)

WHAT DO YOU THINK, IS THIS GOOD OR BAD?

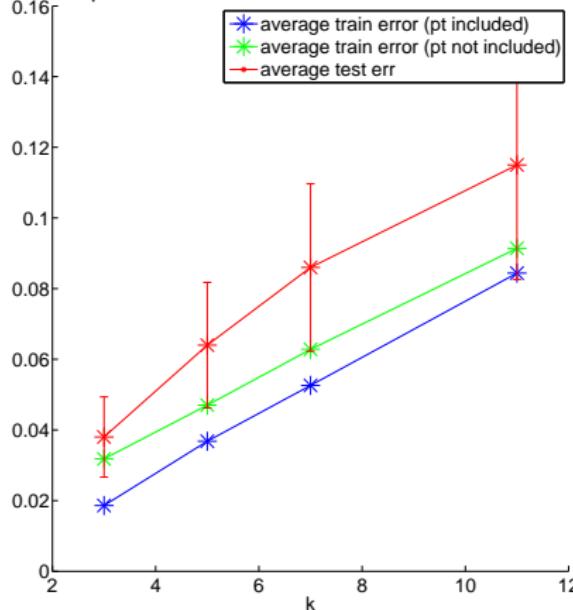
# Application: hand written digits (5)

## Results task 2 (digit 3 vs digit 8):

Digit 3 vs 8, 500 train and 100 test points



average train and test errors



These results are surprisingly good!!!

# Influence of the similarity function

The choice of the similarity function is crucial as well:

- ▶ The performance of kNNrules can only be good if the distance / similarity function encodes the “relevant information”.  
Example: you want to classify mushrooms as “edible” or “not edible” and as distance function between mushrooms you use the difference in weight ...
- ▶ in many applications it is not so obvious how to define a good distance / similarity function  
Example: you want to classify the genre of songs. How do you compute a similarity between different songs???

# Inductive bias

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

# Inductive bias

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

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

# Extensions

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

# Summary

- ▶ The kNN classifier is about the simplest classifier that exists.
- ▶ But often it performs quite reasonably.
- ▶ For any type of data, always consider the kNN classifier as a baseline.
- ▶ One can prove that in the limit of infinitely many data points, the kNN classifier is “consistent”, that is it learns the best possible function (see next lecture).

# Formal setup

# Standard setup for supervised learning

# Let's become more formal now

We now want to introduce the formal setup for supervised statistical learning.

# The underlying space

- ▶ Input space  $\mathcal{X}$ , output space  $\mathcal{Y}$ 
  - ▶ Sometimes, the spaces  $\mathcal{X}$  or  $\mathcal{Y}$  have some mathematical structure (topology, metric, vector space, etc), or we try to construct such a structure.
  - ▶ We assume that each space endowed with a sigma 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
  - ▶ output variables random as well

# A classifier / prediction function

... is simply a function  $f : \mathcal{X} \rightarrow \mathcal{Y}$ .

We now need to be able to measure how “good” a classifier / prediction function is.

Depends on the problem we want to solve, e.g.

- ▶ If  $\mathcal{Y}$  discrete, that is classification
- ▶ If  $\mathcal{Y} = \mathbb{R}$ , that is regression
- ▶ Other outputs are possible, 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?

# True Risk

The **true risk** (or **true expected loss**) of a prediction function  $f : \mathcal{X} \rightarrow \mathcal{Y}$  (with respect to loss function  $\ell$ ) 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 learning is to use some training data to construct a function  $f_n$  that has true risk as small as possible.**

# Bayes risk and Bayes classifier

What is the best function we can think of?

- ▶ The Bayes risk is defined as

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

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

- ▶ In case the infimum is attained, the corresponding function

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

is called the Bayes classifier / Bayes predictor.

# The training data and learning

Assume we are given supervised training data:

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

Note: this is a strong assumption!!!

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

# Consistency of a learning algorithm

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

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

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

- Convergence is in probability, for those who know what that means; if we have convergence almost surely, the algorithm is called **strongly consistent**.

## Consistency of a learning algorithm (2)

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

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

## Consistency of a learning algorithm (3)

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

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

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

# Statistical and Bayesian Decision theory

## Literature:

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

# What if we know all the underlying quantities?

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

# Running example: Male or female?

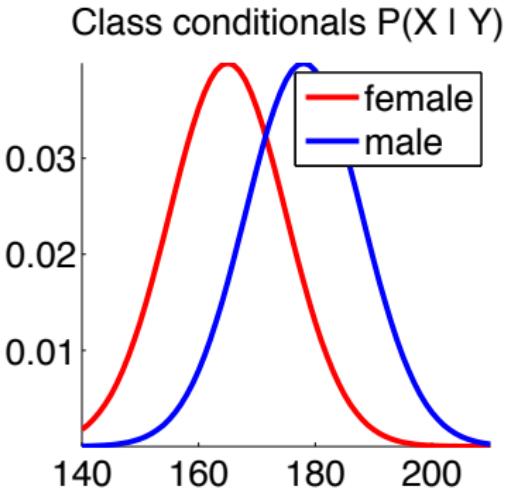
Predict sex of a person from body height:

HOW WOULD YOU PROCEED? ???

# Running example: Male or female?

Predict sex 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)

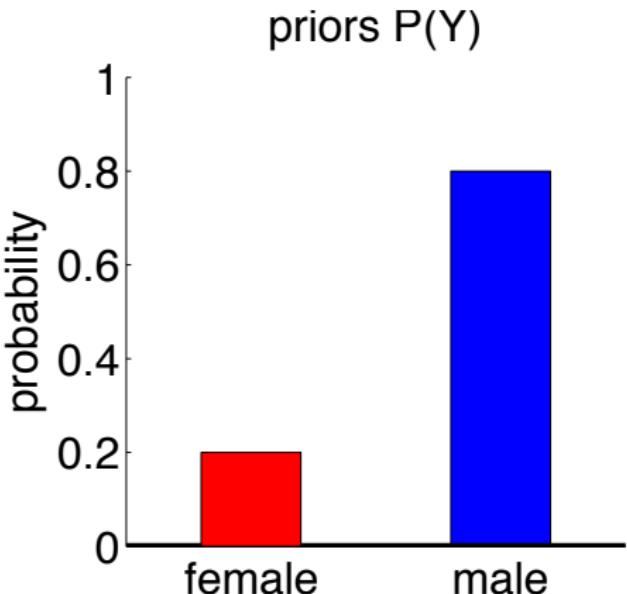
Decide based on class prior probabilities  $P(Y)$ .

- ▶ 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

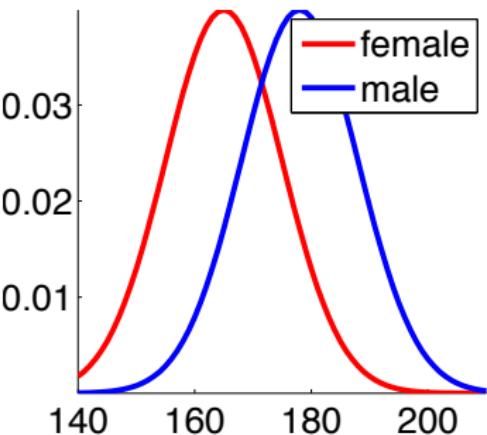


## Approach 2: maximum likelihood principle

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

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

Class conditionals  $P(X | Y)$



## 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

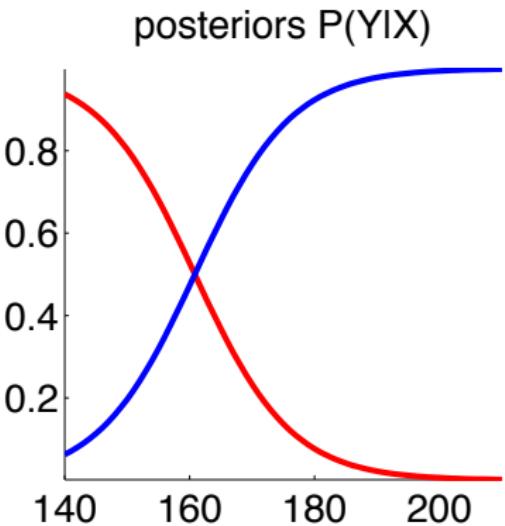
$$P(Y = m|X = x) = \frac{P(X = x|Y = m) \cdot P(Y = m)}{P(X = x)}$$

- ▶ 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  $x$

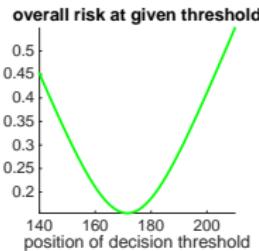
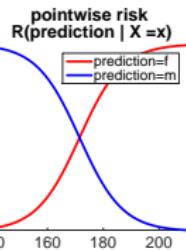
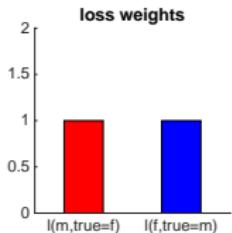
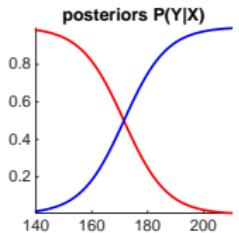
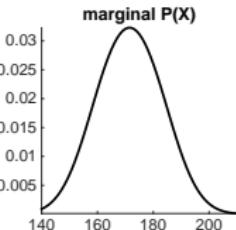
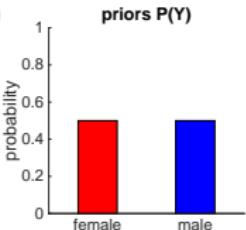
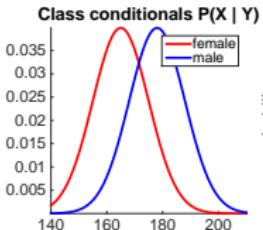
$$R(\hat{y}|X = x) = \ell(x, m, \hat{y})P(Y = m) + \ell(x, f, \hat{y})P(Y = f)$$

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

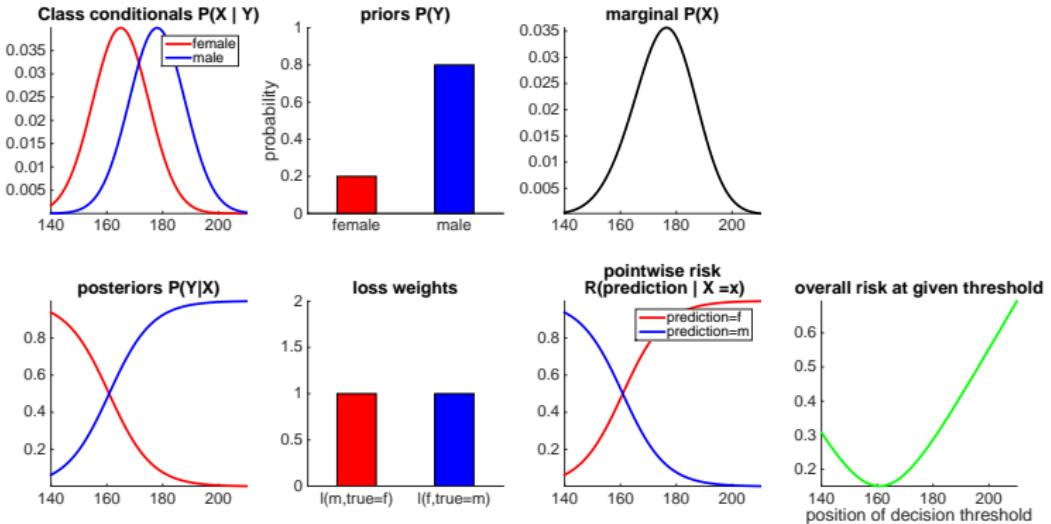
# Example: male vs female

Run `demo_bayesian_decision_theory.m`

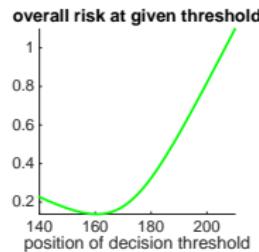
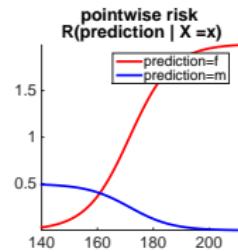
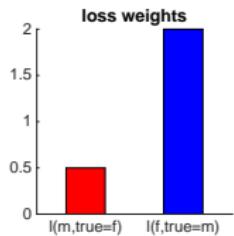
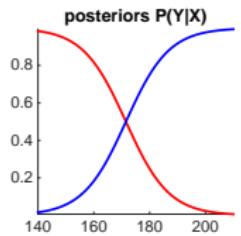
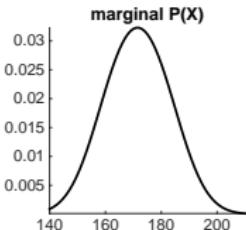
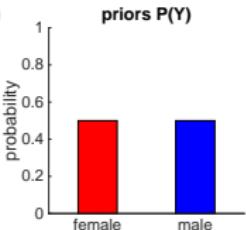
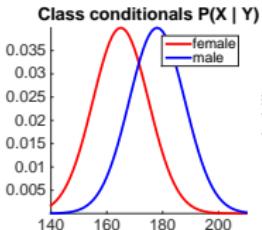
# Example: male vs female (2)



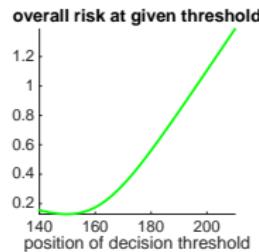
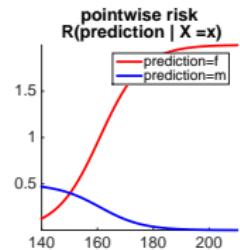
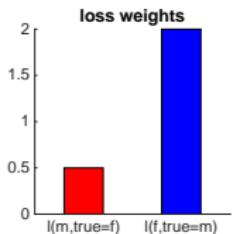
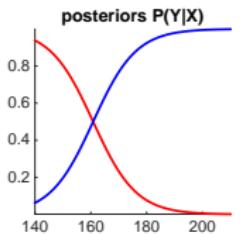
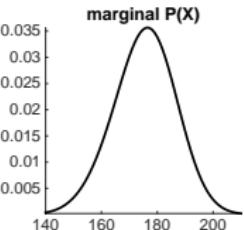
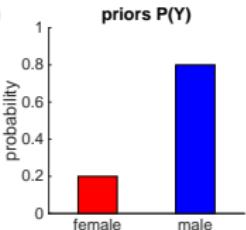
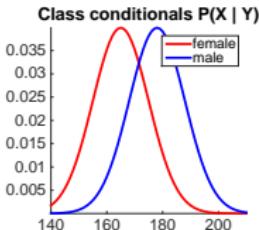
# Example: male vs female (3)



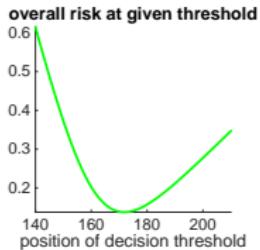
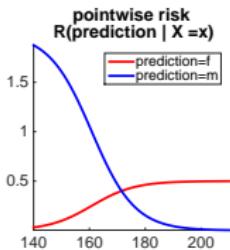
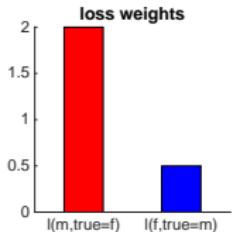
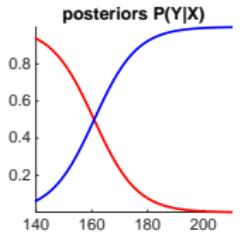
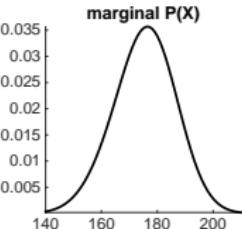
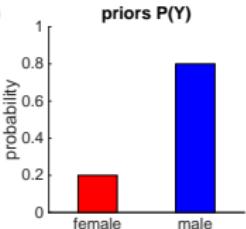
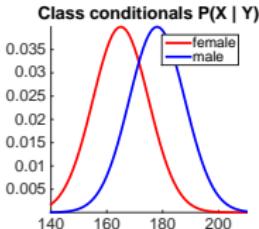
# Example: male vs female (4)



# Example: male vs female (5)



# Example: male vs female (6)



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

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

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

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

# Regression function (context of classification) (2)

Intuition:

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

WHY?

# Regression function (context of classification) (3)

## Proposition 1 (Unique decomposition)

The probability distribution  $P$  is uniquely determined by  $\mu$  and  $\eta$ .

**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  $\eta$  and  $\mu$ .

# Regression function (context of classification) (4)

**Formal proof for the general case:**

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

# Explicit form of the Bayes classifier

Consider the 0-1-loss function. Recall:

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

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

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

Now consider the following classifier:

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

# Explicit form of the Bayes classifier (2)

## Theorem 2 ( $f^\circ$ is the Bayes classifier)

Let  $f : \mathcal{X} \rightarrow \{0, 1\}$  be any (measurable) decision function and  $f^\circ$  the classifier defined above. Then  $R(f) \geq R(f^\circ)$ .

Remark:

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

# Explicit form of the Bayes classifier (3)

## Proof of Theorem 2:

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

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

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

## Explicit form of the Bayes classifier (4)

**Step 2:** Now compare the pointwise error of any other classifier  $f$  to the one of  $f^\circ$ :

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

To see the last step (\*\*):

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

## Explicit form of the Bayes classifier (5)

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

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

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

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



# Explicit form of the Bayes classifier (6)

Remarks:

- ▶ If we work with 0-1-loss and if we know the regression function, then we don't need to "learn", we can simply write down what the optimal classifier is.
- ▶ One can also explicitly compute the optimal classifier for many other loss functions, it is also going to depend on the regression function. We skip this.
- ▶ Problem in practice: we don't know the regression function.

# Plug-in classifier

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

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

Theoretical considerations:

- It can be shown that the plugin-approach is universally consistent. ☺

Practical considerations:

- Estimating densities is notoriously hard, in particular for high-dimensional input spaces. So unfortunately, the plugin-approach is useless for practice. ☹

... for regression under  $L_2$  loss

# Loss functions for regression

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

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

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

# Regression function (context of $L_2$ regression)

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

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

Recall: E stands for expectation, not for error...

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

We need one more intermediate result:

# Regression function (context of $L_2$ regression)

## (2)

### Proposition 3 (Decomposition)

We always have

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

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

### Proof.

(see Gyorfi, Kohler, Krzyzak, Walk: Distribution-free theory for nonparametric regression, p.2)

# Regression function (context of $L_2$ regression)

## (3)

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

# Regression function (context of $L_2$ regression)

(4)

$$\begin{aligned}
 \# &= E((f(x) - \eta(x))(\eta(x) - \gamma)) = \textcircled{1}: E(z) = E(E(z|G)) \\
 &= E\left(E((f(x) - \eta(x))(\eta(x) - \gamma)|X)\right) = \textcircled{2}: E(s(x)g(\gamma, x)|x) = g(x) \cdot E(g(x)|G) \\
 &= E((f(x) - \eta(x))) \cdot \underbrace{E(\eta(x) - \gamma | X)}_{= 0} = \\
 &\quad = \eta(x) - \underbrace{\underbrace{E(\gamma | X)}_{=\eta(x)}}_{= 0}
 \end{aligned}$$



# Explicit form of optimal solution under $L_2$ loss

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

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

## Theorem 4 (Explicit form of optimal $L_2$ -solution)

The function  $f^\circ$  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  $\geq 0$ , and it is  $= 0$  for  $f(X) = \eta(X)$ .
- ▶ So the whole right hand side is minimized by  $f(X) = \eta(X)$ .



## Basic learning principles: ERM, RRM

## Two major principles

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

There are two major approaches to supervised learning:

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

# Empirical risk minimization

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

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

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

# Empirical risk minimization (2)

Empirical risk minimization approach:

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

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

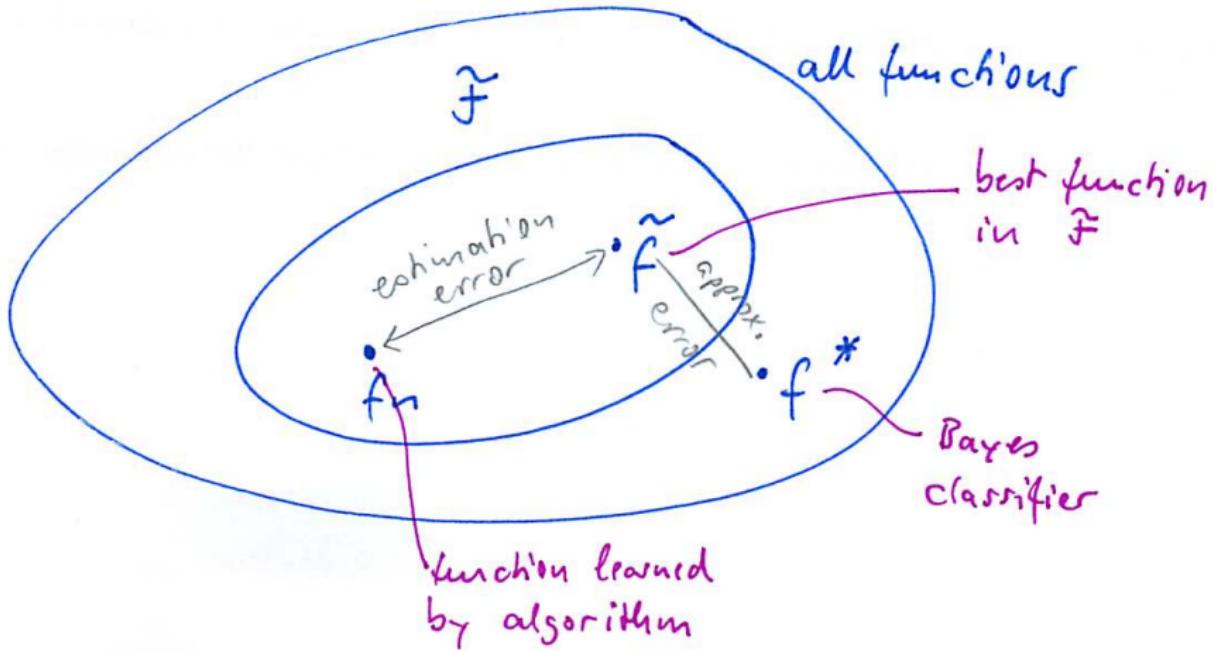
(might not be unique; for simplicity, let's assume the minimizer exists)

# Estimation vs approximation error

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

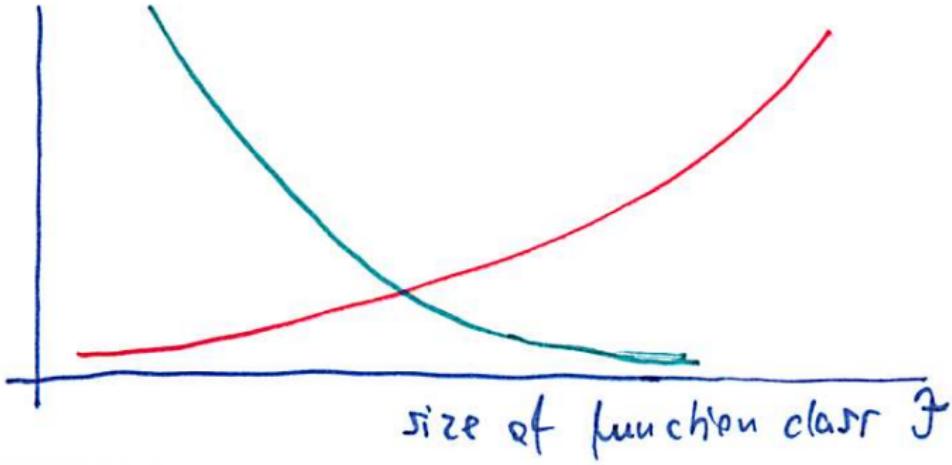
- ▶ Denote by  $\tilde{f}$  the true best function in the set  $\mathcal{F}$ , that is  
$$\tilde{f} = \operatorname{argmin}_{f \in \mathcal{F}} R(f).$$
- ▶ The quantity  $R(f_n) - R(\tilde{f})$  is called the **estimation error**. It is a random variable that depends on the random sample.
- ▶ The quantity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

# Estimation vs approximation error (2)



## Estimation vs approximation error (3)

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



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

# Overfitting vs Underfitting

Coming back to the terms underfitting and overfitting:

- ▶ Underfitting happens if  $\mathcal{F}$  is too small. In this case we have a small estimation error but a large approximation error.
- ▶ Overfitting happens if  $\mathcal{F}$  is too large. Then we have a high estimation error but a small approximation error.

# Bias-Variance tradeoff in $L_2$ -regression

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

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

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

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

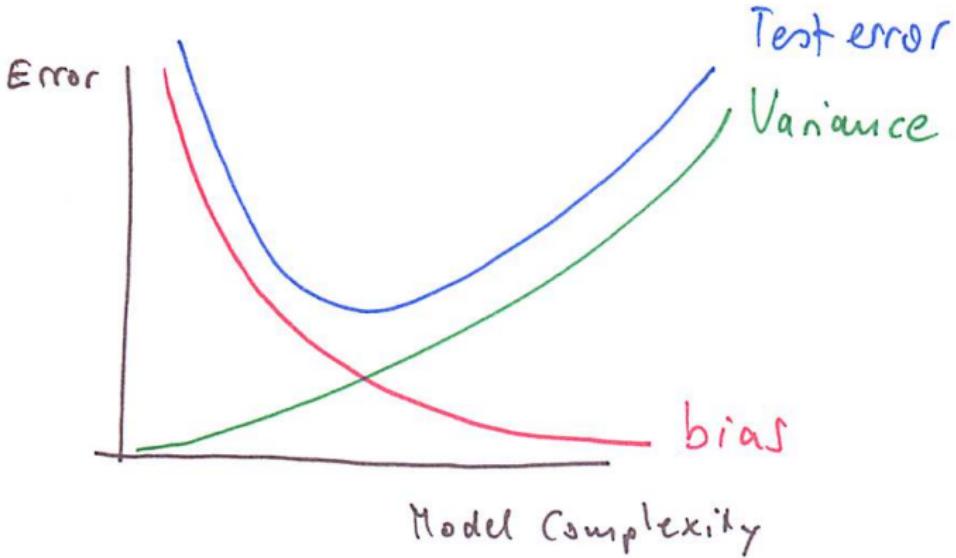
**Proof.** skipped (see Gyorfi, Kohler, Krzyzak, Walk: Distribution-free theory for nonparametric regression, p.24)

# Bias-Variance tradeoff in $L_2$ -regression (2)

Intuition:

- ▶ The 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 (3)



## ERM, remarks

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

# Regularized risk minimization

Crucial problem in ERM: choose  $\mathcal{F}$

Alternative approach:

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

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

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

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

# Regularized risk minimization (2)

Intuition:

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

## Regularized risk minimization (3)

EXERCISE: WHAT HAPPENS IF  $\lambda$  IS VERY SMALL? VERY LARGE?

# Linear methods for supervised learning

# Linear methods for regression

# Linear least squares regression

Literature:

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

# Linear setup

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

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

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

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

## Linear setup (2)

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

Formally, the linear least squares problem is the following:

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

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

## Example

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

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

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

# Concise notation

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

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

*d*

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

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

## Concise notation (2)

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

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

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

# Getting rid of $b$

We want to write the problem even more concisely.

- Define the matrices

$$\tilde{X} = \begin{pmatrix} x_{11} & \dots & x_{1d} & 1 \\ \vdots & & \vdots & \vdots \\ x_{n1} & \dots & x_{nd} & 1 \end{pmatrix} \quad n \quad \tilde{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_d \\ b \end{pmatrix} \quad d+1$$

- Then we have

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

## Getting rid of $b$ (2)

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

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

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

## Getting rid of $b$ (3)

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

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

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

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

# ML $\leadsto$ Optimization problem

We can see:

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

# Least squares regression is convex

Recap:

- Convex optimization problem (maths appendix, page 1231)

## Proposition 5 (Least squares is convex)

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

### Proof. Exercise

# Solution, case of full rank

Recap:

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

# Solution, case of full rank (2)

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

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

## Proof intuition.

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

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

## Solution, case of full rank (3)

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

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

## Solution, case of full rank (4)

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



## Solution, case of full rank (5)

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

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

# Solution, general case

Recap:

- ▶ Generalized inverse of a matrix (see maths appendix page 1216)

## Solution, general case (2)

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

Assume that  $X$  has rank  $< d$ .

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

**Proof (sketch).**

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

## Solution, general case (3)

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

## Solution, general case (4)

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

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

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

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



# Relationship between $n$ and $d$

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

WHAT DO YOU THINK IS BETTER:

- ▶  $n$  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  $d \ll n$ :

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

# Relationship between $n$ and $d$ (4)

Case  $d \gg n$ :

- ▶ Typically, 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 large because the dimension of the space is so large). This leads to overfitting.

# Summary: Linear least squares regression

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

# Feature representation of data

## Feature representation of data

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

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

# Feature representation of data (2)

## Example: bag of words representation for texts

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

## Feature representation of data (3)

$T_1$ : I like to play soccer. 1

$T_2$ : My soccer shoes are red.

$T_3$ : We moved to a new house.

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

# Feature representation of data (4)

## Example: strings in a feature representation

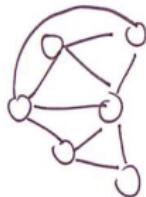
- ▶ Given a string
- ▶ Represent it by counting substrings (can also allow substrings with “gaps” in between)

# Feature representation of data (5)

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

Count the occurrence of certain subgraphs (called motifs):

graph 1:



Motifs: # in graph 1



10

# in graph 2

7



5

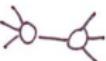
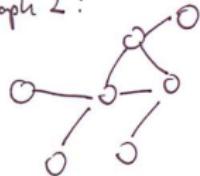
1



1

1

graph 2:



0

0

:

# Feature representation of data (6)

**Example: books and/or users in amazon.**

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

Representation of books by user data:

|       | user1 | user2 | user3 |
|-------|-------|-------|-------|
| book1 | 0     | 0     | 1     |
| book2 | 1     | 0     | 0     |
| book3 | 0     | 0     | 0     |
| ...   | 1     | 1     | 0     |
|       | 0     | 0     | 0     |

how often did user1 buy book 4

Representation of books by books

|       | book1 | book2 | book3 | ... |
|-------|-------|-------|-------|-----|
| book1 | 2     | 1     | 3     |     |
| book2 | 1     | 5     | 2     |     |
| book3 | 3     | 2     | 3     |     |
| :     |       |       |       |     |

how often was book 3 bought together with book 2

# Feature representation of data (7)

## Example: images

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

# Feature representation of data (8)

## General procedure that works very often:

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

## Feature representation of data (9)

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

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

# Least squares with linear combination of basis functions

Literature:

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

# Using non-linear basis functions

Idea:

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

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

where the functions  $\Phi_1, \dots, \Phi_D$  are arbitrary “basis functions”.

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

# Examples

## Example 1:

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

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

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

## Examples (2)

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

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

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

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

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

## Examples (3)

Example 2: Feature spaces

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

We might consider basis functions as the following ones:

- ▶  $\Phi_1$  counts the number of occurrences of the word “soccer”
- ▶  $\Phi_1$  counts the number of occurrences of the word “team”
- ▶  $\Phi_3$  tells you how many words the text has in total
- ▶ ... etc ...

# How to solve it

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

- ▶ Define the design matrix as follows:

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

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

# Advantages and disadvantages

- ▶ Note that in the given scenario, we did not choose our function basis to depend on the input. We chose the functions before we got to see the data points.
- ▶ If we have prior knowledge about our data, we can select a “good” set of basis functions.
- ▶ For any useful inference, the dimension  $D$  of the feature space has to be much smaller than the number  $n$  of data points.
  - ▶ WHY, AGAIN???
  - ▶ This is still quite restrictive. Just consider the case where our data is  $D$ -dimensional and we want to have a function space with polynomials of degrees one and two. There are already of the order  $D^2$  many basis polynomials of degree two ( $x_i x_j$  for  $i, j = 1, \dots, D$ ). So it can easily happen that we need more basis functions than we can cope with...

## Advantages and disadvantages (2)

- ▶ There is one way out of this trap, namely to regularize, in particular by enforcing sparsity. See Lasso below.

## Ridge regression: least squares with $L_2$ -regularization

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

# Idea

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

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

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

# Ridge regression problem

Consider the following regularization problem:

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

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

# Solution

## Theorem 8 (Solution of Ridge Regression)

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

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

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

# Solution (2)

## Proof.

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

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

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

## Solution (3)

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

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

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

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



## Example (by Matthias Hein)

- ▶ True function: periodic function + noise
- ▶ Basis functions  $\Phi$ : first 10 Fourier basis functions

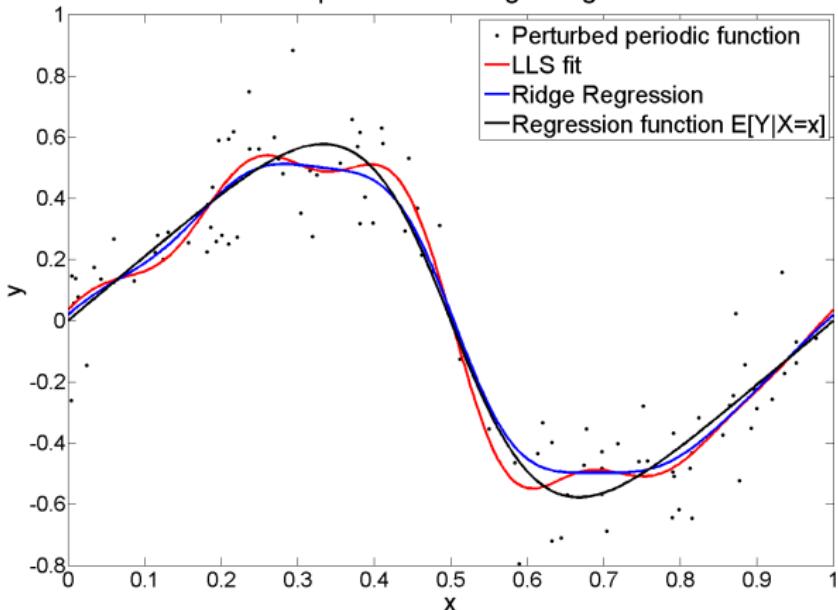
$$x \mapsto \sin(kx), \quad (k = 1, \dots, 10)$$

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

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

# Example (by Matthias Hein) (2)

Least Squares and Ridge Regression



## Choice of the parameter $\lambda$

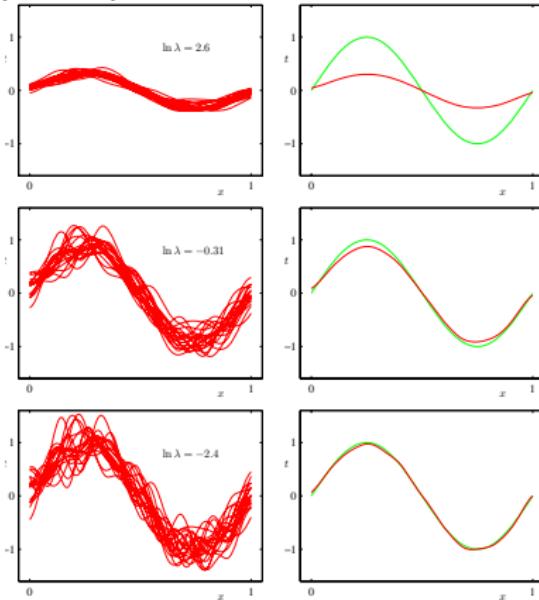
QUESTION: WHAT IS THE ROLE OF  $\lambda$ ? WHAT HAPPENS TO ESTIMATION AND APPROXIMATION ERROR IF IT IS HIGH / LOW?

# Choice of the parameter $\lambda$ (2)

Example (from Bishop's book):

Left: results for decreasing amount of regularization

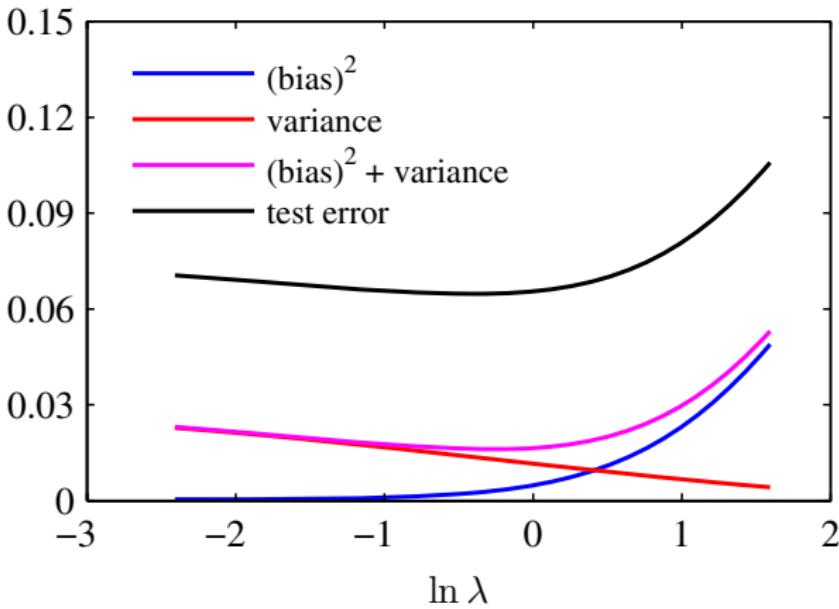
Right: True curve (green), average estimated curve (red)



# Choice of the parameter $\lambda$ (3)

Same example, bias-variance decomposition:

Larger regularization constant  $\lambda$  leads to less complex functions:



# (\*) Ridge regression as shrinkage method

Geometric interpretation of regularization via SVD:

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

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

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

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

- ▶ Standard least squares regression (without regularization) corresponds to  $\lambda = 0$ , and the fraction satisfies

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

## (\*) Ridge regression as shrinkage method (2)

- ▶ Regularized case:

- ▶ Case  $\sigma_i$  large: not much difference to non-regularized case:

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

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

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

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

## (\*) Ridge regression as shrinkage method (3)

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

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

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

## (\*) Ridge regression as shrinkage method (4)

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

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

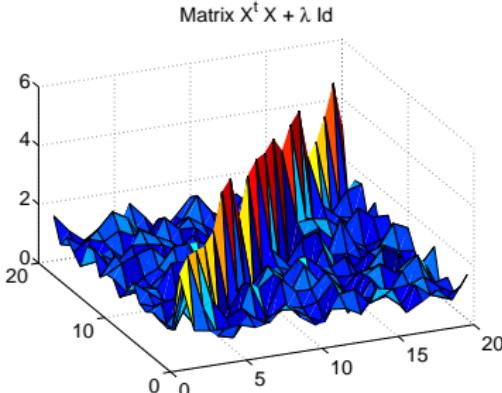
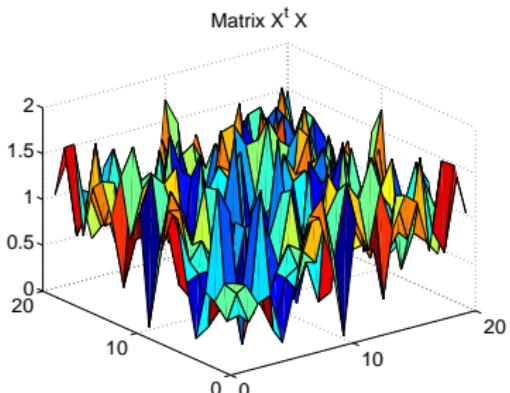
Read it on wikipedia if you are interested ☺

# History and Terminology

- ▶ Invented by Andrey Tikhonov, 1943, in the context of integral equations.  
Original publication: *Tikhonov, Andrey Nikolayevich. On the stability of inverse problems. Doklady Akademii Nauk SSSR, 1943*  
This type of regularization is often called **Tikhonov regularization** after its inventor.
- ▶ Introduced in statistics literature in the following paper:  
*Hoerl and Kennard. Ridge regression: Biased estimation for nonorthogonal problems. Technometrics, 1970.*

# History and Terminology (2)

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



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

# Summary: Ridge regression

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

## Lasso: least squares with $L_1$ -regularization

Books:

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

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

# Sparsity

- ▶ Consider the setting of linear regression with basis functions  $\Phi_1, \dots, \Phi_D$ .
- ▶ It is very desirable to obtain a solution function  $f_n := \sum_i w_i \Phi_i$  for which many of the coefficients  $w_i$  are zero. Such a solution is called “sparse”.
- ▶ Reasons:
  - ▶ Computational reasons: even if we have many basis functions, we just need to evaluate few of them.
  - ▶ Interpretability of the solution

# A naive regularizer for sparsity

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

## A naive regularizer for sparsity (2)

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

Use the regularizer

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

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

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

## A naive regularizer for sparsity (3)

$\Omega_0$  is a discrete function, and optimizing discrete functions is typically NP hard.

## Excursion: $p$ -norms

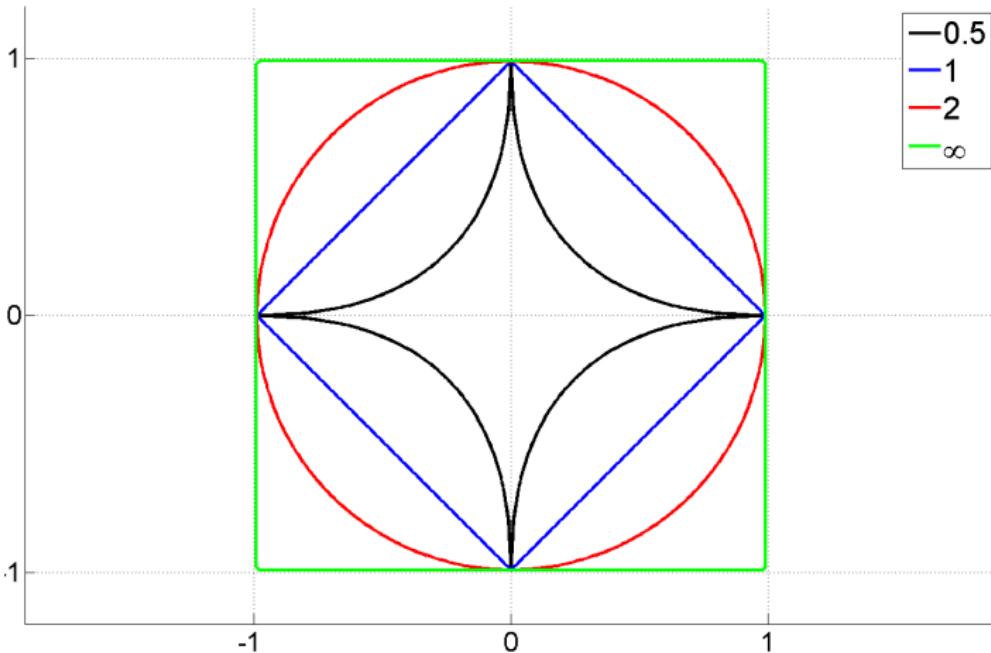
- ▶ For  $p > 0$ , define for a vector  $w \in \mathbb{R}^D$

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

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

## Excursion: $p$ -norms (2)

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



(Image by Matthias Hein)

## Excursion: $p$ -norms (3)

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

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

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

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

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

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

## Sparsity and the $L_1$ -norm

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

Question: Does it still tend to give sparse solutions?

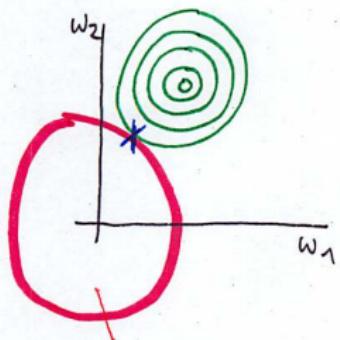
Answer is yes, see the illustration on the next slide:

# Sparsity and the $L_1$ -norm (2)

Illustration: Assume we restrict the search to functions with  $\|w\| \leq \text{const.}$  The blue cross shows the best solution

$w = (w_1, w_2)^t$ . It is not sparse for  $L_2$ , but sparse for  $L_1$ -norm regularization.

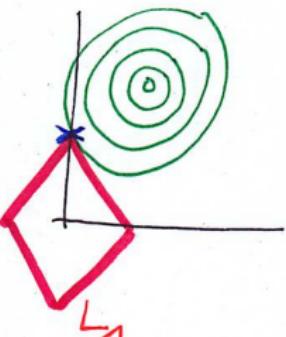
contour lines of the empirical error



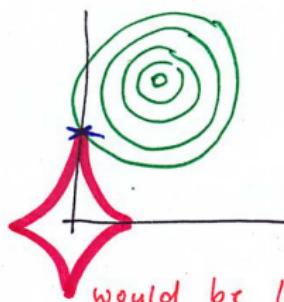
$L_2$  regularization:

set with  $\|w\|_2 \leq \text{const.}$

$w_1, w_2 \neq 0$ , not sparse



$w_1 = 0$ , sparse



would be  $L_p$  for  
 $p < 1$

even sparser, but not  
sparse any more.

## Sparsity and the $L_1$ -norm (3)

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

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

# The Lasso

Consider the following regularization problem:

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

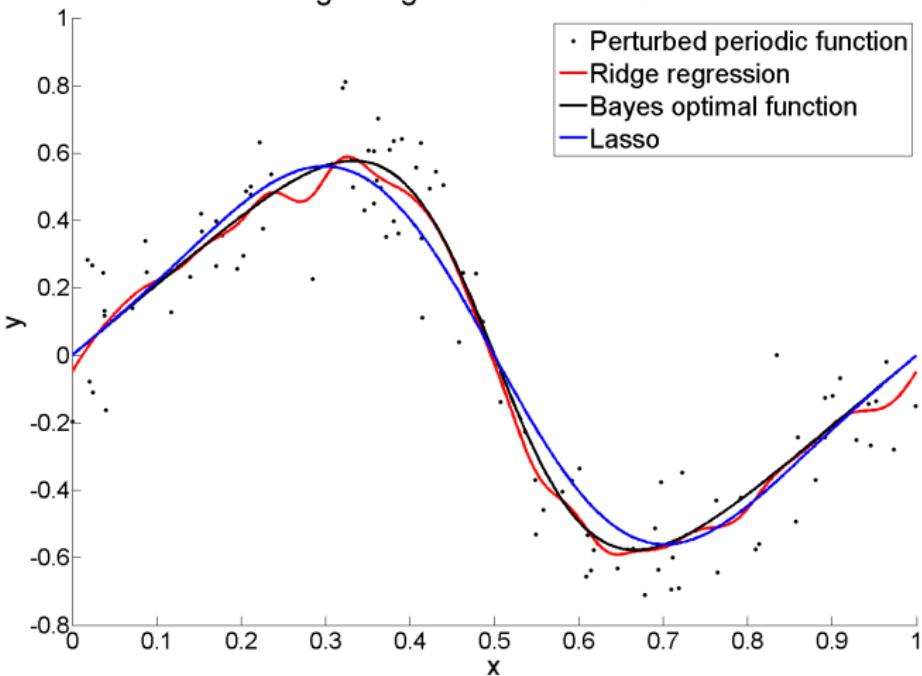
$$w_{n,\lambda} := \operatorname{argmin}_{w \in \mathbb{R}^D} \frac{1}{n} \|Y - \Phi w\|_2^2 + \lambda \|w\|_1.$$

# Solution of the Lasso problem

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

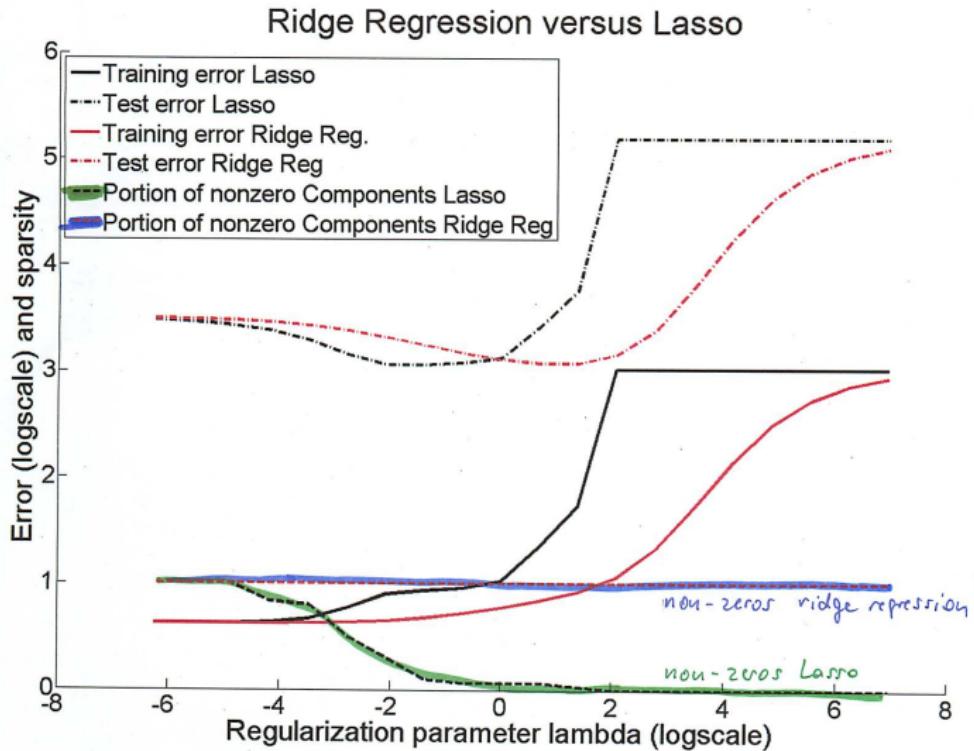
# Example

Ridge Regression versus Lasso



(Figure by Matthias Hein)

## Example (2)



(Figure by Matthias Hein)

# History

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

## Summary: the Lasso

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

## (\*) Probabilistic interpretation of linear regression

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

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

# Linear regression: ERM = maximum likelihood

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

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

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

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

# Linear regression: ERM = maximum likelihood (2)

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

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

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

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

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

# Linear regression: RRM = Bayesian MAP

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

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

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

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

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

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

# More generally: Bayesian interpretation of ERM and RRM

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

Examples:

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

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

## More generally: Bayesian interpretation of ERM and RRM (2)

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

# Feature normalization

## In practice: normalization

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

- ▶ Body height measured in mm or cm or even km

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

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

## In practice: normalization (2)

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

1. Centering:

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

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

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

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

# Selecting parameters by cross validation

## Cross validation - purpose

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

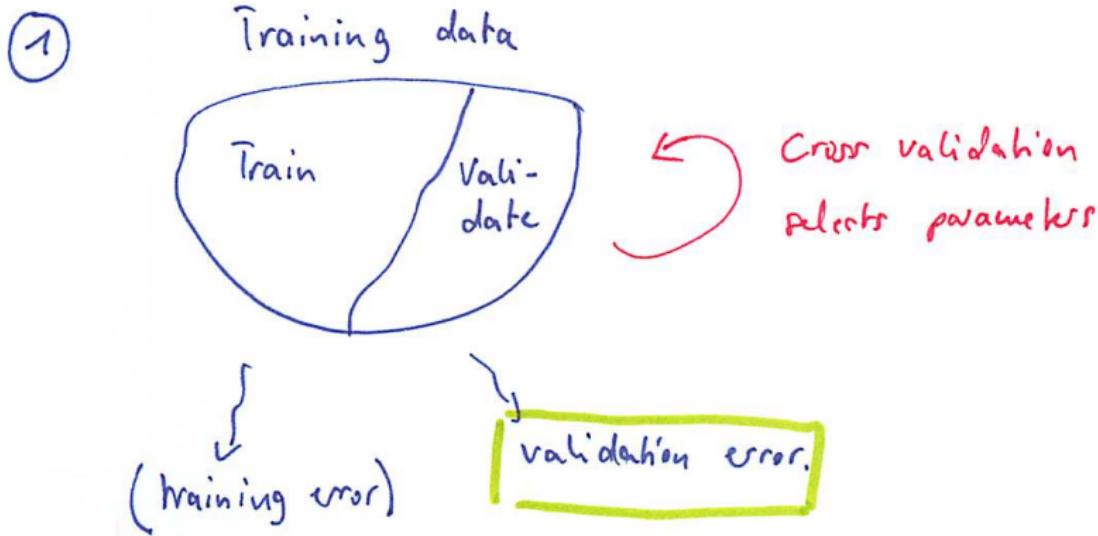
- ▶ Regularization parameter in ridge regression or Lasso
- ▶ Parameter  $C$  of the SVM
- ▶ Parameter  $\sigma$  in the Gaussian kernel
- ▶ Number of principle components in PCA
- ▶ But you also might want to figure out whether certain design choices make sense, for example whether it is useful to remove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

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

# K-fold cross validation

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

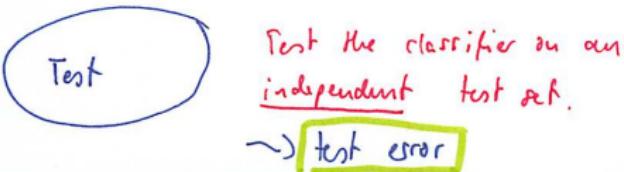
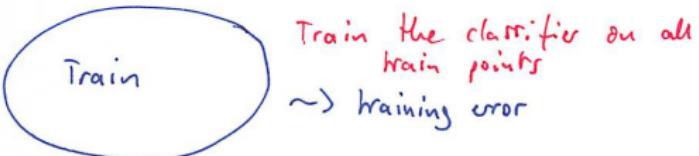
# K-fold cross validation (2)



# K-fold cross validation (3)

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

② With the parameters identified by cross validation:



## K-fold cross validation (4)

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

## K-fold cross validation (5)

What number of folds  $K$ ?

Not so critical, often people use 5 or 10.

# K-fold cross validation (6)

How to choose the set  $S$ ?

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

## K-fold cross validation (7)

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

# Advantages and disadvantages

Disadvantages of cross validation:

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

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

## Advantages and disadvantages (2)

Advantages:

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

Always, always, always do cross validation!!! Make sure the final test set is never touched while training (retraining for improving the test error is not allowed, then the data is spoiled).

# Linear methods for classification

# Intuition and feature representation

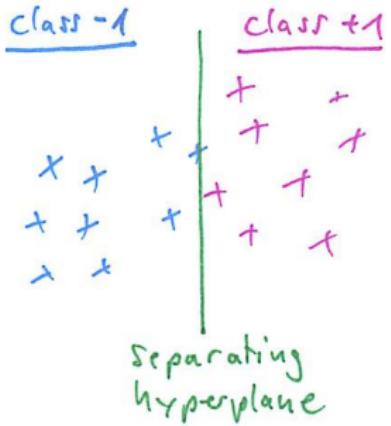
# Intuition

Given:

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

## Intuition (2)

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



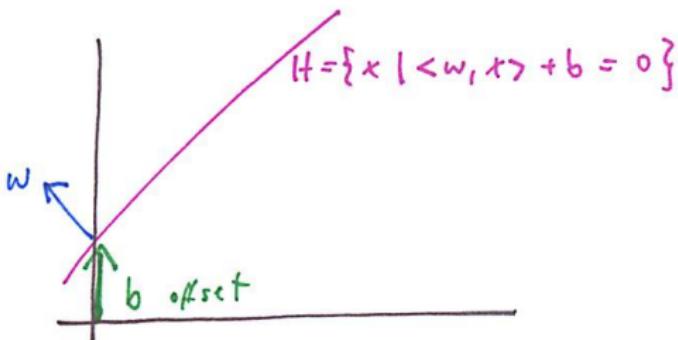
# Hyperplanes in $\mathbb{R}^d$

Now let's consider linear classification with hyperplanes.

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

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

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



# Classification using hyperplanes

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

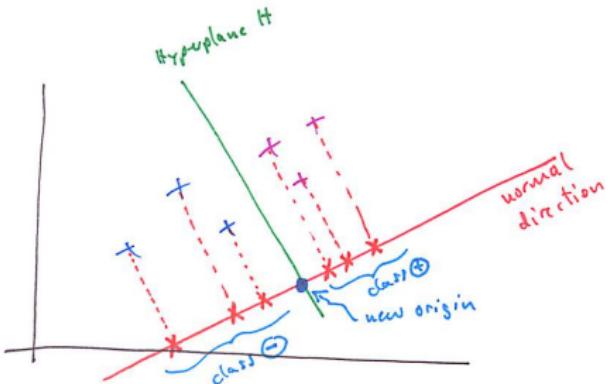
$$\text{sign}(\langle w, x \rangle + b) \in \{\pm 1\}$$

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

# Projection interpretation

Here is another way to interpret classification by hyperplanes:

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



# Loss functions for classification

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

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

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

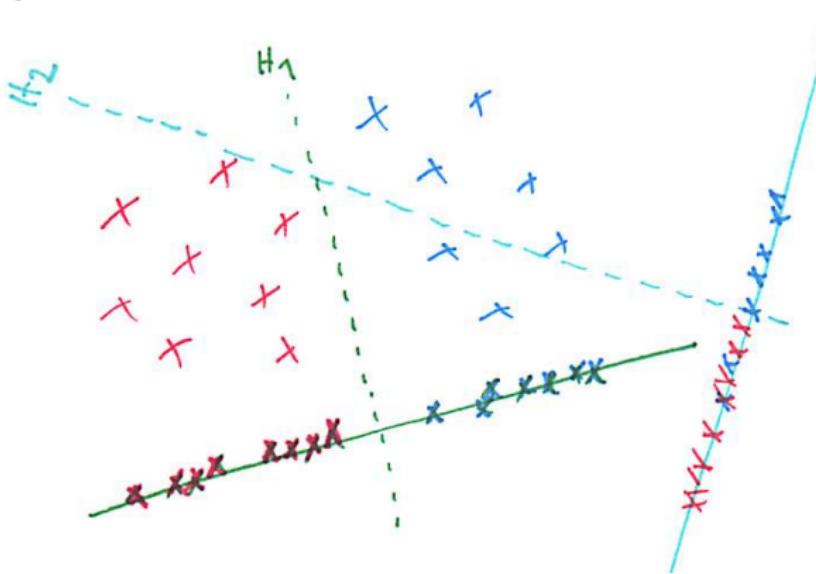
## (\*) Linear discriminant analysis

Literature:

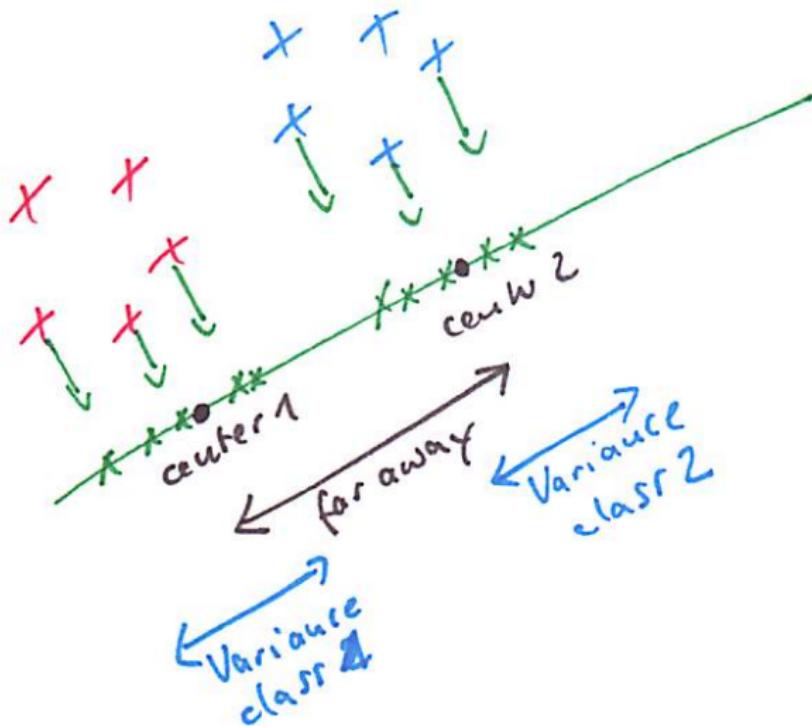
- ▶ Hastie/Tibshirani/Friedman Sec. 4.3
- ▶ Duda / Hart

# LDA: Geometric motivation

Different projections: which one is better for classification?



# LDA: Geometric motivation (2)

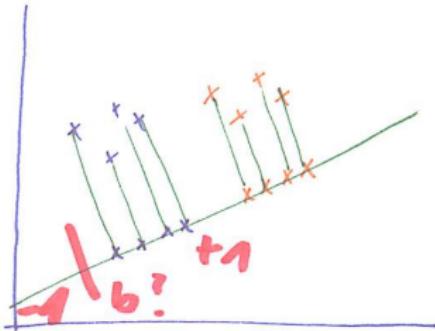
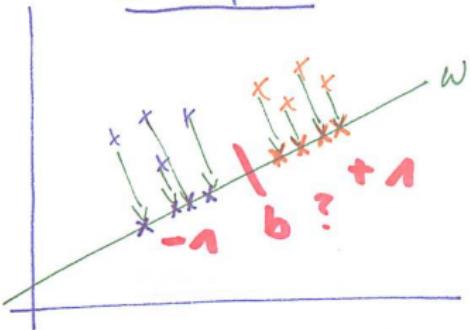


## LDA: Geometric motivation (3)

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

# LDA: Geometric motivation (4)

Role of  $b$ :



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

# LDA: Geometric motivation (5)

Observe the different roles of  $w$  and  $b$ :

## Step 1: finding a good separating direction $\rightsquigarrow w$

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

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

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

Note: the label information is used in both steps!

## Formally: the Fisher criterion

Define the following quantities for class  $+1$ :

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

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

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

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

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

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

## Formally: the Fisher criterion (2)

Now define the **Fisher criterion** as

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

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

# Fisher criterion in matrix form

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

- ▶ Define the between-class scatter matrix as

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

- ▶ Define the total within-class scatter matrix as

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

- ▶ The Fisher criterion can now be rewritten as

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

# Solution vector $w$

## Proposition 9 (Solution vector $w^*$ of LDA)

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

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

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

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

### Proof (sketch).

- Take the derivative:

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

## Solution vector $w$ (2)

- ▶ Set it to 0:

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

- ▶ Rewrite (plug in the definition of  $C_B$ ):

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

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

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

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



## Determining $b$

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

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

# LDA, alternative motivation by ERM

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

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

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

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

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

- ▶ No assumption on the underlying distributions.

Then we can prove the following nice theorem:

# LDA, alternative motivation by ERM (2)

## Theorem 10 (LDA as ERM)

Consider the following two optimization problems:

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

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

(2) The LDA problem:

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

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

# LDA, alternative motivation by ERM (3)

**Proof:** skipped.

# LDA, alternative motivation by ERM (4)

Comments:

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

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

# LDA, motivation by Bayesian decision theory

Let us make the following assumptions:

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

Then we can argue as follows:

- ▶ Bayesian decision theory: under these assumptions the optimal classifier selects according to whether
$$P(Y = 1 \mid X = x) \stackrel{?}{>} P(Y = -1 \mid X = x).$$
- ▶ Equivalently:
$$\log \left( P(Y = 1 \mid X = x) / P(Y = -1 \mid X = x) \right) \stackrel{?}{>} 0.$$

## LDA, motivation by Bayesian decision theory (2)

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

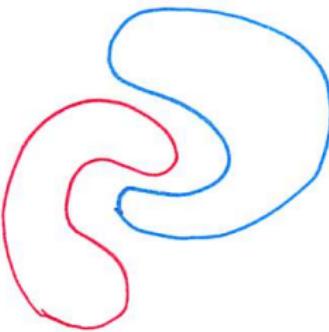
# LDA, motivation by Bayesian decision theory (3)

Insights:

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

# Limitations and generalizations

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



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

# Limitations and generalizations (2)

Generalizations:

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

# History

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

# Summary: Linear discriminant analysis (LDA)

Three different motivations:

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

All the motivations lead to the same algorithm:

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

# Excursion: Probabilistic interpretation of ERM

Chapter 8 in Murphy

## General idea

- ▶ Have seen in the case of LDA: Minimizing the  $L_2$  loss over the class of linear function is “the same” as finding the Bayesian decision theory approach for the probabilistic model with Gaussian class conditional priors, and Gaussian noise, and uniform class prior.
- ▶ Will see something similar in a minute for logistic regression.

Let's briefly look at the general concept.

# Excursion: Probabilistic interpretation of ERM

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

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

# Excursion: Probabilistic interpretation of ERM

## (2)

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

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

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

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

# Excursion: Probabilistic interpretation of ERM

## (3)

What does this tell us?

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

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

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

# Excursion: Probabilistic interpretation of ERM

## (4)

Why is this insight useful?

It helps to get more intuition:

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

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

# Excursion: Probabilistic interpretation of ERM

## (5)

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

# Logistic regression

Literature: Hastie/Tibshirani/Friedman Section 4.4

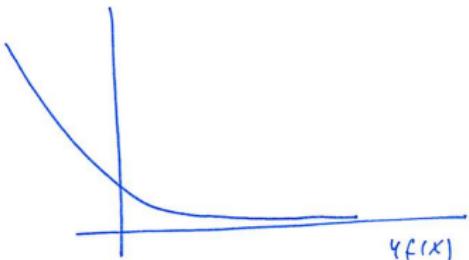
For the probabilistic point of view, see Chapter 8 in Murphy

# Logistic regression problem as ERM

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

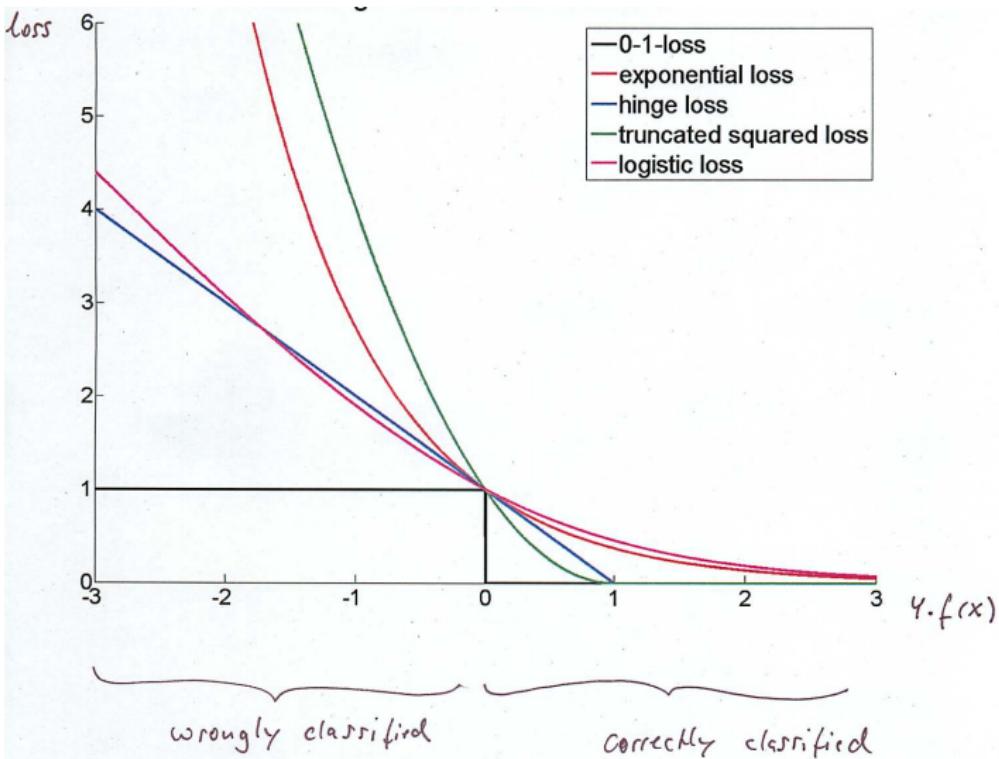
## Logistic regression problem as ERM (2)

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



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

# Logistic regression problem as ERM (3)



# Computing the ERM solution

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

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

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

The answer comes from the following Bayesian approach to classification.

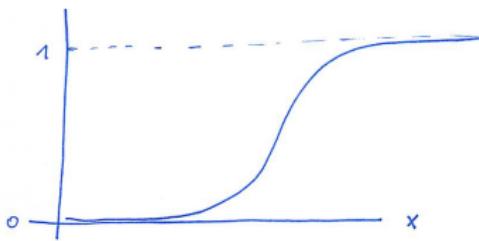
# The logistic model

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

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

with  $f(x) = \langle w, x \rangle + b$ . Here  $w$  and  $b$  are the parameters. The function  $1/(1 + \exp(-t))$  is called the **logistic function** and looks as follows:

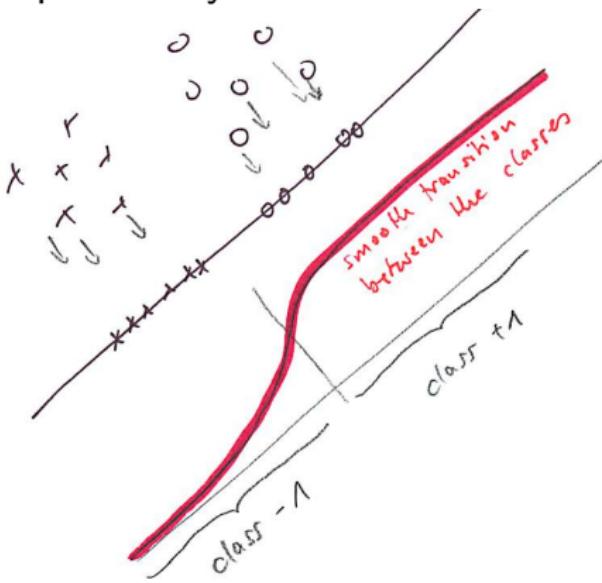
# The logistic model (2)



# The logistic model (3)

- ▶ Intuition:

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



## The logistic model (4)

The actual value of  $f(x)$  tells “how far” we are from the decision surface, that is “how sure” the classifier is about this class.  $|f(x)| \approx 0.5$  means that the classifier does not really know by itself,  $f(x)$  close to 0 or 1 means that the classifier is “pretty sure”.

## The logistic model (5)

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

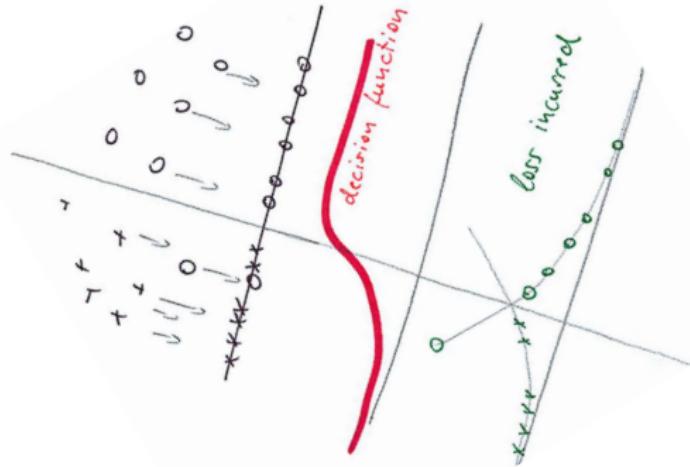
$$\ell(X, f(X), Y) = \log(1 + \exp(-Yf(X)))$$

This is the logistic loss function.

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

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

# The logistic model



Decision function:

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

Loss incurred for the respective decisions: logistic

# Adding regularization

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

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

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

# History of logistic regression

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

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

# Summary: logistic regression

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

# Linear Support vector machines

## Literature:

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

# Intuition and primal

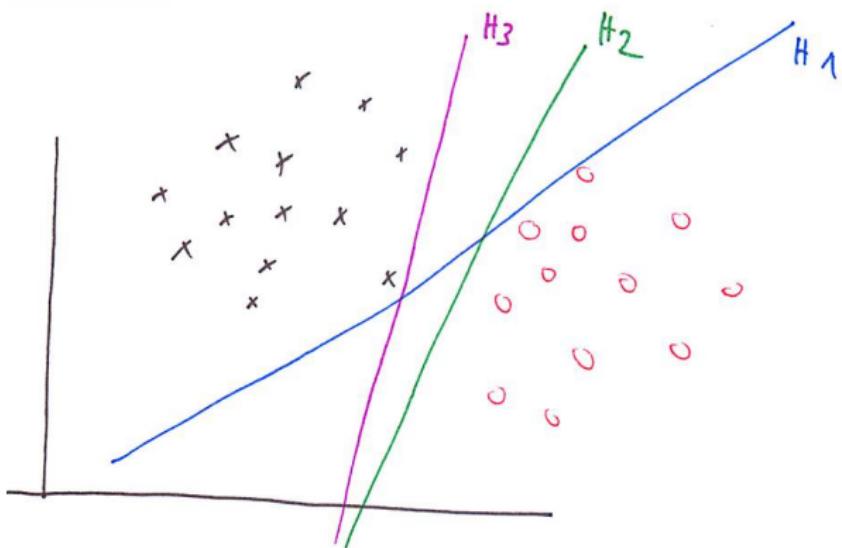
## Prelude

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

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

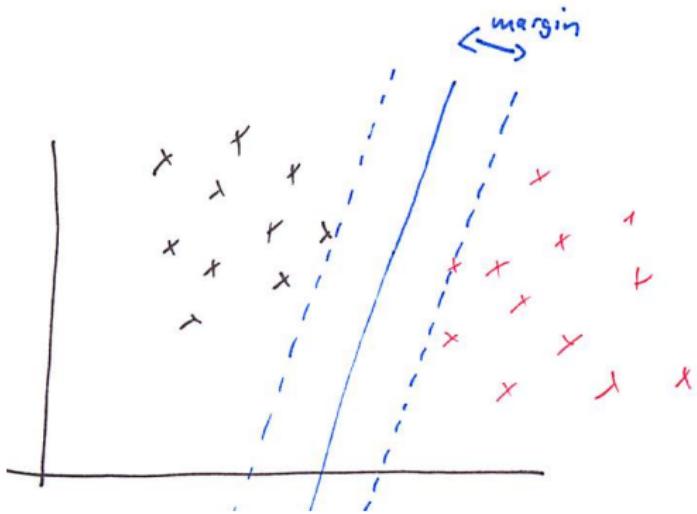
# Geometric motivation

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



## Geometric motivation (2)

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



Why might this make sense?

## Geometric motivation (3)

Why might this make sense:

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

# Canonical hyperplane

- We are interested in a linear classifier of the form

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

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

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

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

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

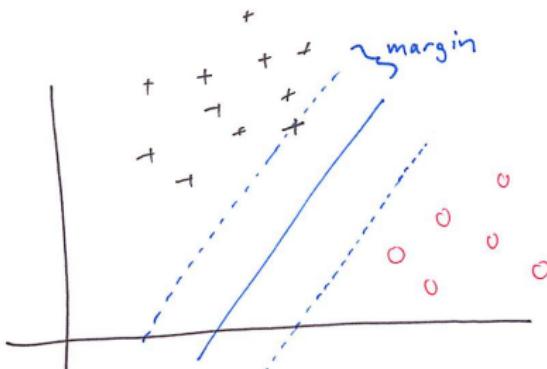
$$\min_{i=1, \dots, n} |\langle w, x_i \rangle + b| = 1$$

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

# The Margin

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

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



# The Margin (2)

## Proposition 11 (Margin)

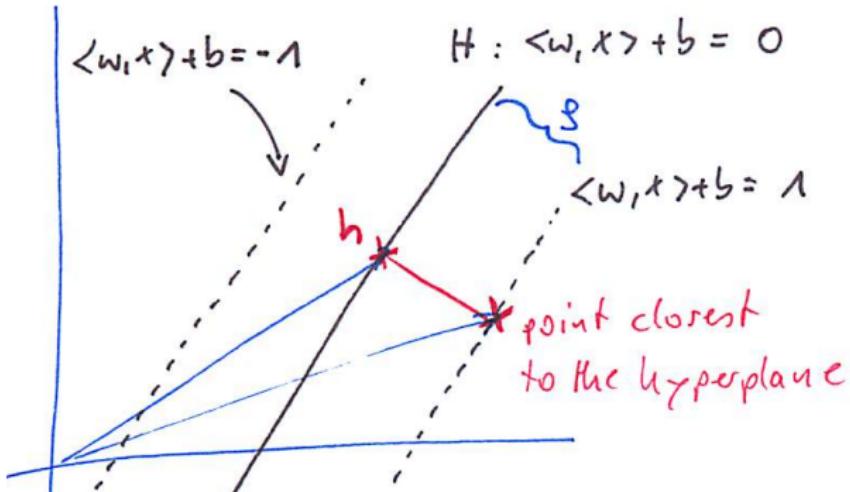
For a hyperplane in canonical representation, the margin  $\rho$  can be computed by  $\rho = 1/\|w\|$ .

### First proof.

Observe:

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

# The Margin (3)



# The Margin (4)

We also know that

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

because the line connecting  $x$  and  $h$  is in the normal direction  $w$  and has length  $\rho$ .

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

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

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

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



# The Margin (5)

## Alternative proof:

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

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

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

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

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

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

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

# Hard margin SVM

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

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

In formulas:

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

## Hard margin SVM (2)

Usually, we consider the following equivalent optimization problem:

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

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

# Hard margin SVM (3)

First remarks:

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

## Hard margin SVM (4)

However, big disadvantage:

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

# Soft margin SVM

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

$$\underset{w \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2} \|w\|^2 + \frac{C}{n} \sum_{i=1}^n \xi_i$$

$$\text{subject to } Y_i(\langle w, X_i \rangle + b) \geq 1 - \xi_i \quad \forall i = 1, \dots, n$$

$$\xi_i \geq 0 \quad \forall i = 1, \dots, n$$

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

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

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

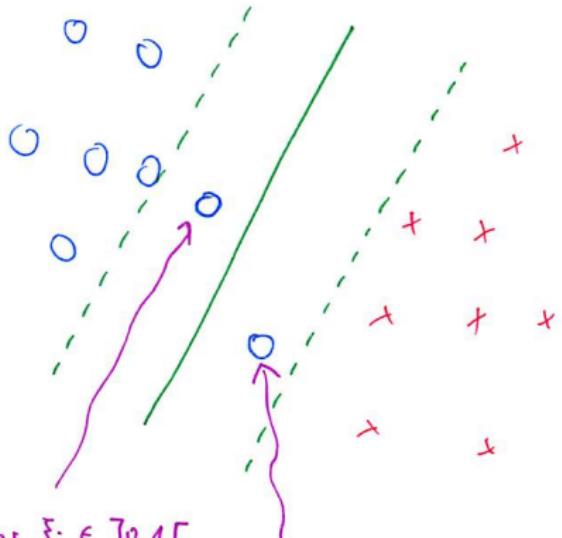
# Soft margin SVM (2)

Interpretation:

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

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

# Soft margin SVM (3)



has  $\xi_i \in [0, 1[$   
(correct side,  
inside margin)  
has  $\xi_i > 1$   
(wrong side)

# SVM as regularized risk minimization

We want to interpret the SVM in the regularization framework:

$$\text{minimize} \quad \underbrace{\frac{1}{2} \|w\|^2}_{\sim \text{ Regularization term}} + \underbrace{\frac{C}{n} \sum_{i=1}^n \xi_i}_{\sim \text{ Risk term}}$$

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

## SVM as regularized risk minimization (2)

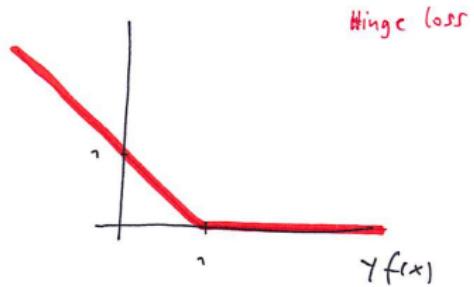
Consider the constraint  $Y_i(\langle w, X_i \rangle + b) \geq 1 - \xi_i$ . Exploiting  $\xi_i \geq 0$  we can rewrite it as follows:

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

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

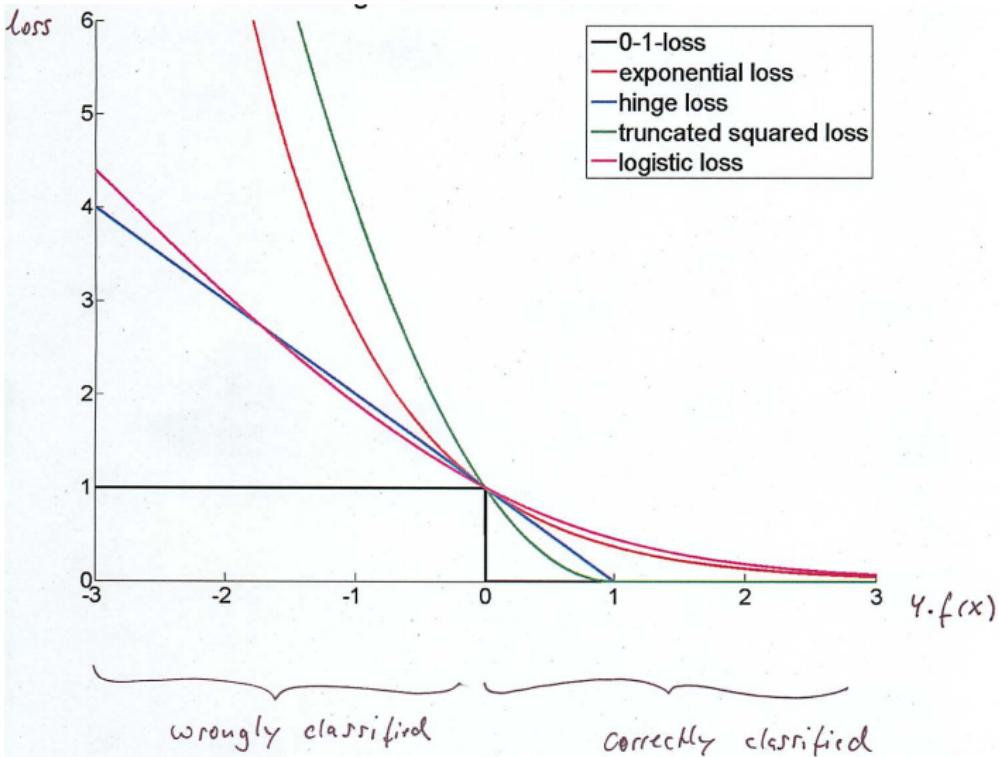
$$\ell(x, y, f(x)) = \max\{0, 1 - yf(x)\}$$

It looks as follows:



# SVM as regularized risk minimization (3)

Comparison to other loss functions:



## SVM as regularized risk minimization (4)

This loss function has a couple of interesting properties:

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

## SVM as regularized risk minimization (5)

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

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

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

It is a convention that we use  $C$  in SVMs, not  $\lambda$  ...

## SVM as regularized risk minimization (6)

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

# Summary so far: Linear SVM (primal)

What we have seen so far:

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

## Excursion: convex optimization, primal, Lagrangian, dual

see slides 1231ff. in the appendix

# Deriving the dual problem

# Dual of hard margin SVM

It turns out that all the important properties of SVM can only be seen from the dual optimization problem.

So let us derive the dual problem:

# Dual of hard margin SVM (2)

**Primal problem (the one we start with):**

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

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

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

# Dual of hard margin SVM (3)

**Formally, the dual problem is the following:**

Dual function:

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

Dual Problem:

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

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

## Dual of hard margin SVM (4)

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

In particular,

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

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

# Dual of hard margin SVM (5)

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

- ▶ First exploit (\*):

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

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

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

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

# Dual of hard margin SVM (6)

## Dual function:

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

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

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

## Dual of hard margin SVM (7)

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

- ▶ By now, (\*\*) is meaningless, because  $w$  disappeared already.  
So we drop it.
- ▶ But we need to carry the condition (\*) to the dual.

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

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

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

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

# Dual of the soft margin SVM

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

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

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

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

# Dual SVM in practice

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

# Dual SVM in practice (2)

**Recover the primal optimal variables  $w, b$  from the dual solution  $\alpha$ :**

- ▶ To compute  $w$ : directly use (\*\*):  $w = \sum_i \alpha_i Y_i X_i$
- ▶ To compute  $b$ :
  - ▶ Consider a point  $X_j$  for which  $0 < \alpha_j < C/n$  (active constraint). By the KKT conditions we know that

$$\alpha_j (Y_j (\langle w, X_j \rangle + b) - 1) = 0$$

$$\text{hence } Y_j (\langle w, X_j \rangle + b) = 1.$$

- ▶ Solving this for  $b$  and replacing  $w$  by the formula (\*\* ) we get

$$b = (1/Y_j - \langle w, X_j \rangle) = Y_j - \sum_{i=1}^n Y_i \alpha_i \langle X_i, X_j \rangle$$

## Dual SVM in practice (3)

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

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

with  $w$  and  $b$  as on the previous slide:

$$\begin{aligned}\langle w, X \rangle + b &= \left\langle \sum_i \alpha_i Y_i X_i, X \right\rangle + b \\ &= \sum_i \alpha_i Y_i \langle X_i, X \rangle + \left( Y_j - \sum_i Y_i \alpha_i \langle X_i, X_j \rangle \right)\end{aligned}$$

In practice, you don't need to take the intermediate steps of computing  $w$  and  $b$ , you can use this formula directly, it only depends on the  $\alpha_i$ .

# In practice: solve the primal or the dual?

In practice: Solve the primal or dual problem?

- ▶ Because we know that for quadratic problems we have strong duality, we could either solve the primal or the dual problem.
- ▶ The primal problem has  $d + 1$  variables (where  $d$  is the dimension of the space), and  $n$  constraints (where  $n$  is the number of training points). If  $d$  is small compared to  $n$ , then it makes sense to solve the primal problem.
- ▶ The dual problem has  $n$  variables and  $n + 1$  constraints. If  $d$  is large compared to  $n$ , then it is better to solve the dual problem. In most SVM libraries, this is the default.

# Important properties of SVMs

# Solution as linear combination

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

# Support vectors

Support vector property:

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

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

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

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

## Support vectors (2)

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

# Scalar products

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

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

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

# Exercise

It might be instructive to solve the following exercise:

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

**Primal problem:**

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

**Dual problem:**

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

## Exercise (2)

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

# History

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

# Summary: linear SVM

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

# Kernel methods for supervised learning

# Positive definite kernels

Introductory literature:

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

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

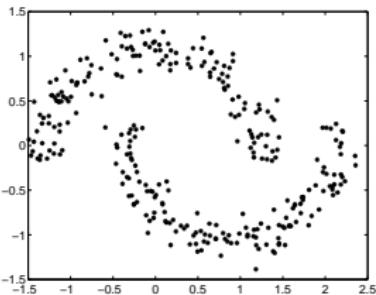
- ▶ Steinwart, Christmann: Support Vector Machines. Springer, 2008.

# Intuition

# Linear methods — disadvantages

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

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

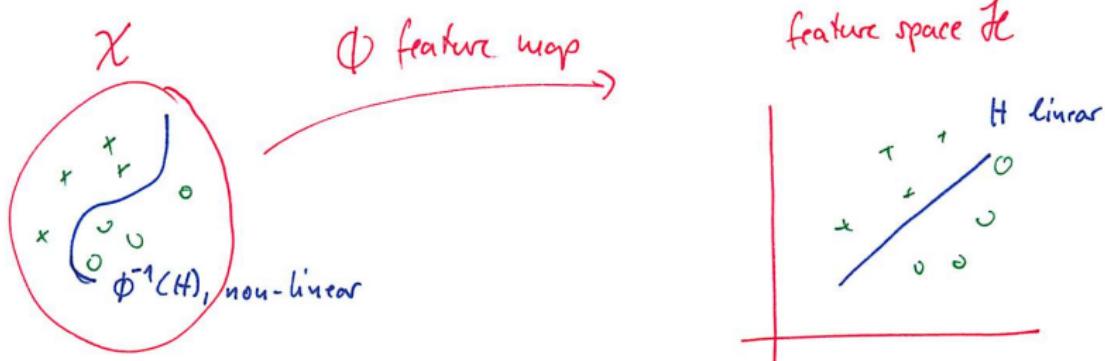


## Linear methods — disadvantages (2)

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

# Linear methods — disadvantages (3)

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



## Starting point: Key observation for SVMs

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

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

# Starting point: Key observation for SVMs (2)

Let us be more explicit:

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

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

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

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

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

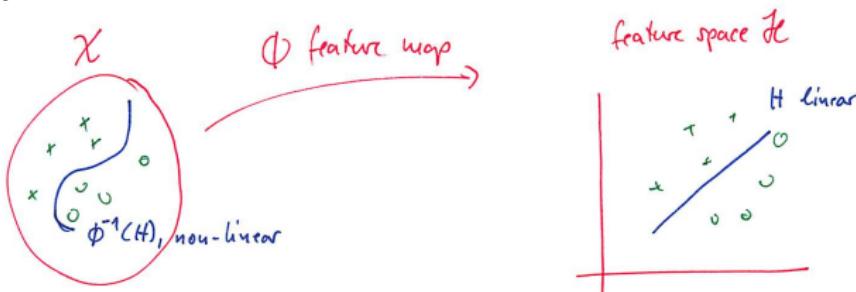
(and the same goes for the function that evaluates the results on test points).

# Idea: Kernels replacing feature maps

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

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

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



# Kernel methods — the overall picture

What we want to do:

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

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

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

# Kernel methods — the overall picture (2)

In the following we try to make this idea formal.

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

# Definition and properties of kernels

# Kernel function — definition

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

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

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

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

# Kernel function — definition (2)

Remarks:

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

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

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

# Scalar products lead to kernels

Observe:

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

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

is a valid kernel!

Proof sketch:

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

## Scalar products lead to kernels (2)

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

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

# Intuition: kernels as similarity functions

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

## Example: linear kernel

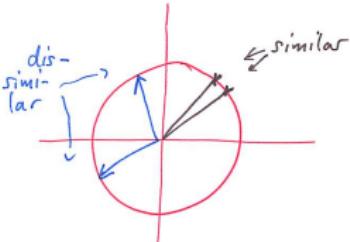
**The linear kernel.** The trivial kernel on  $\mathbb{R}^d$  defined by the standard scalar product:

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

Is obviously a kernel.

## Example: cosine similarity

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



- ▶ As a measure how similar the points are, we use the cosine of the angle between the two points.
  - ▶  $\cosine = 1 \iff$  points agree
  - ▶  $\cosine = 0 \iff$  points are orthogonal

## Example: cosine similarity (2)

- If the data points are normalized, then the cosine of the angle spanned by two points  $x$  and  $y$  is given by the scalar product  $\langle x, y \rangle$ .

Is obviously a kernel.

## Example: the Gaussian kernel

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

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

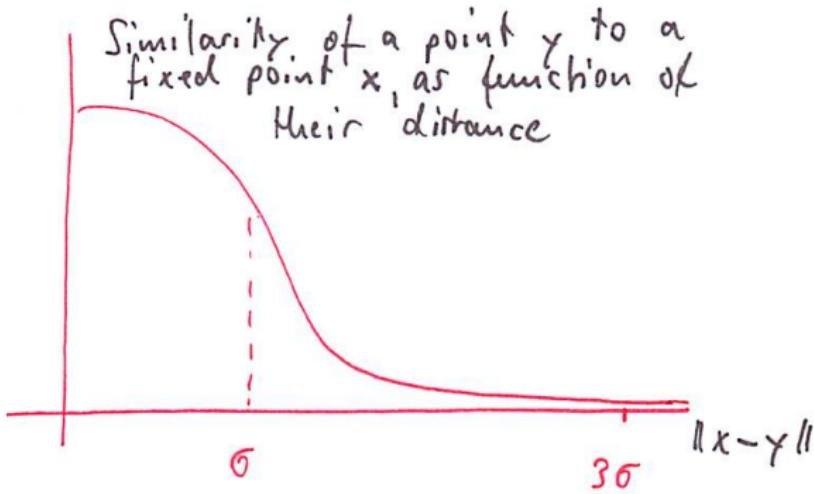
where  $\sigma > 0$  is a parameter.

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

See the text books if you are interested.

## Example: the Gaussian kernel (2)

Induced notion of similarity:



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

## Example: the Gaussian kernel (3)

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

## Example: polynomial kernel

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

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

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

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

## Example: kernels based on explicit feature maps

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

Induced similarity functions:

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

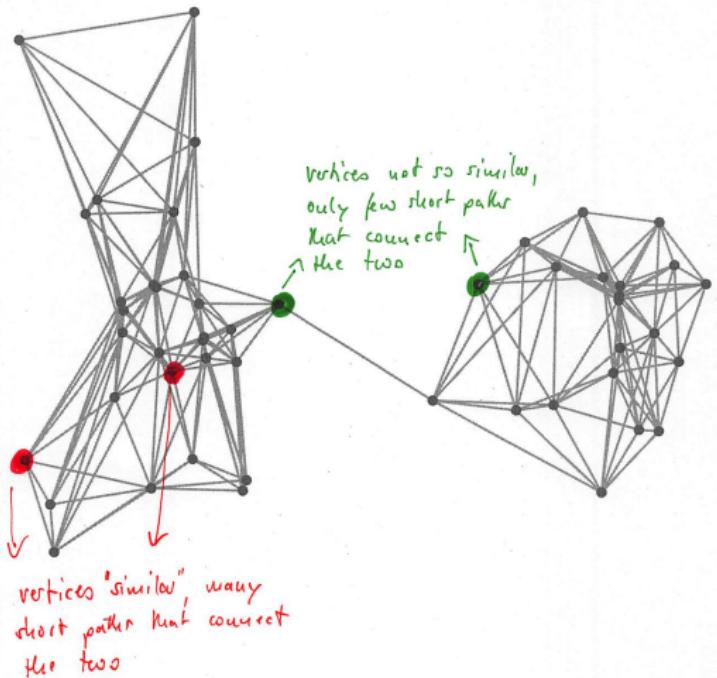
# Example: kernel between vertices in a graph

Application scenario:

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

## Example: kernel between vertices in a graph (2)

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



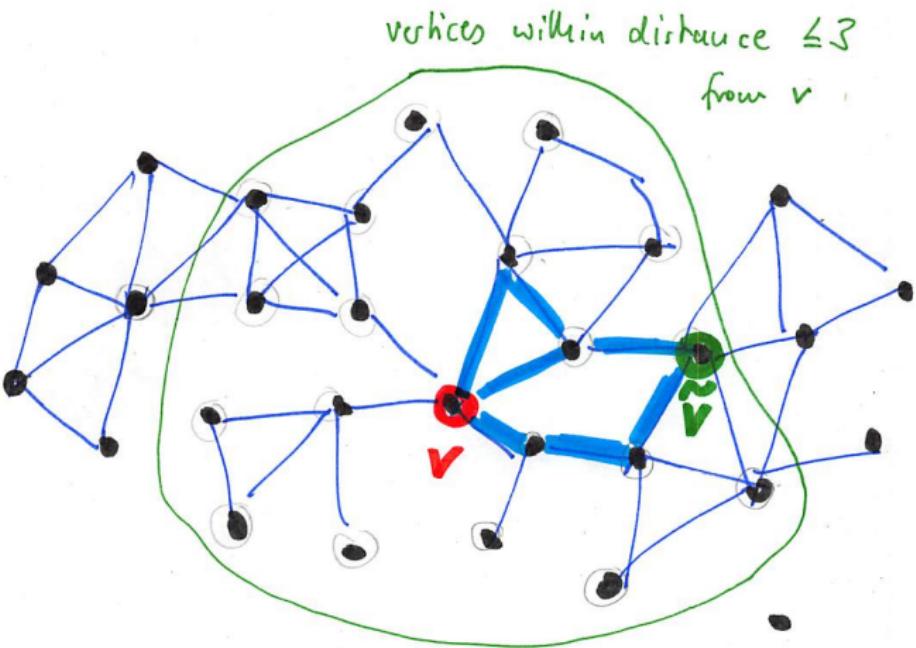
## Example: kernel between vertices in a graph (3)

To define a kernel between vertices on a graph:

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

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

## Example: kernel between vertices in a graph (4)



## Example: kernel between vertices in a graph (5)

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

# Simple rules for dealing with kernels

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

## Simple rules for dealing with kernels (2)

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

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

In particular,  $\tilde{k}(x, y) = f(x)f(y)$  is a kernel.

**Proof.** EXERCISE.

## (\*) Kernel matrix: pd or psd?

Due to a common confusion, let me stress again:

- ▶ A scalar product is positive definite. This means that the property  $\langle v, v \rangle > 0$  holds (with strict inequality!) for all  $v \neq 0$
- ▶ The kernel matrix is positive semi-definite in the sense that  $c'Kc \geq 0$  (greater or equal!).

WHY IS THIS NOT A CONTRADICTION?

## (\*) Kernel matrix: pd or psd? (2)

- ▶ Consider data  $X_1, \dots, X_n \in \mathbb{R}^d$ .
- ▶ Then the kernel matrix coincides with  $K = XX^t$ .
- ▶ Let  $v$  be an eigenvector of  $K$ . We have

$$v'XX'v = v'Kv = \lambda v'v = \lambda$$

- ▶ For eigenvectors with  $\lambda > 0$ : fine.
- ▶ For eigenvectors with  $\lambda = 0$ : Then  $X'v = 0$ , and the scalar product of the 0-vector with itself is 0. Fine as well.

In particular: The rank of the kernel matrix is at most the dimension of the underlying vector space. So if  $n > d$ , the kernel matrix must have eigenvalues 0. EXERCISE!

# Reproducing kernel Hilbert space and feature maps

# Kernels do what they are supposed to do

Here is the justification for why we defined kernels the way we did:

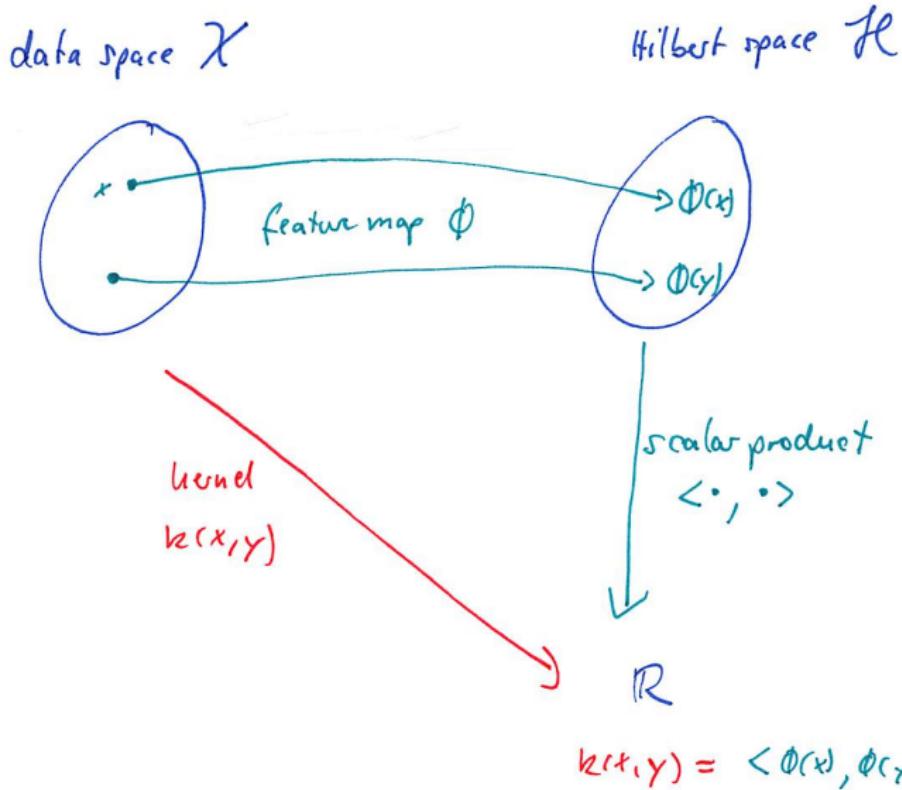
## Theorem 12 (Kernel implies embedding)

A function  $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  is a kernel if and only if there exists a Hilbert space  $\mathcal{H}$  and a map  $\Phi : \mathcal{X} \rightarrow \mathcal{H}$  such that  $k(x, y) = \langle \Phi(x), \Phi(y) \rangle$ .

If you have never heard of Hilbert spaces, just think of the space  $\mathbb{R}^d$ . The crucial properties are:

- ▶  $\mathcal{H}$  is a vector space with a scalar product  $\langle \cdot, \cdot \rangle_{\mathcal{H}}$
- ▶ Space is complete (all Cauchy sequences converge)
- ▶ (Scalar product gives rise to a norm:  $\|x\|_{\mathcal{H}} := \langle x, x \rangle_{\mathcal{H}}$ )

# Kernels do what they are supposed to do (2)



## Kernels do what they are supposed to do (3)

WHICH DIRECTION OF THE THEOREM IS EASY, WHICH ONE IS DIFFICULT?

# Kernels do what they are supposed to do (4)

## **Proof of “ $\Leftarrow$ ”**

Clear by definition of the kernel (we defined the kernel exactly such that this direction holds).

# Kernels do what they are supposed to do (5)

## Proof of “ $\Rightarrow$ ”

We have to prove the following:

Given  $\mathcal{X}$  and  $k$ , there exists a vector space  $\mathcal{H}$  with a scalar product  $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ , and a mapping  $\Phi : \mathcal{X} \rightarrow \mathcal{H}$  such that

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle_{\mathcal{H}}$$

for all  $x, y \in \mathcal{X}$ .

We now introduce the Reproducing Kernel Hilbert Space (RKHS), a scalar product on this space and a corresponding feature mapping  $\Phi$ . 

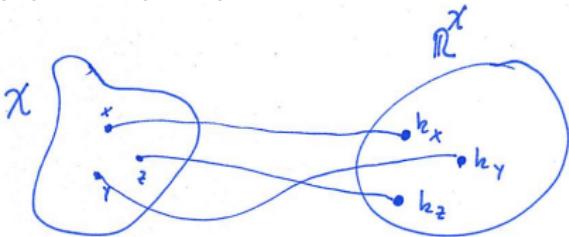
# Reproducing kernel Hilbert space

As vector space we are going to use a space of functions:

- ▶ Consider a mapping  $\Phi : \mathcal{X} \rightarrow \mathbb{R}^{\mathcal{X}}$  (where  $\mathbb{R}^{\mathcal{X}}$  denotes the space of all real-valued functions from  $\mathcal{X}$  to  $\mathbb{R}$ ), defined as

$$x \mapsto \Phi(x) := k_x := k(x, \cdot)$$

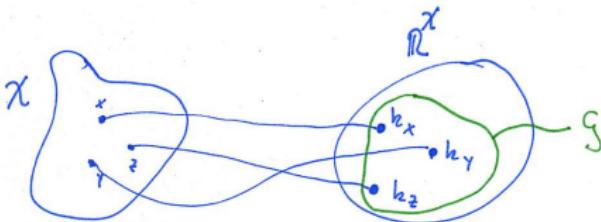
That is, the point  $x \in \mathcal{X}$  is mapped to the function  $k_x : \mathcal{X} \rightarrow \mathbb{R}$ ,  $k_x(y) = k(x, y)$ .



# Reproducing kernel Hilbert space (2)

- Now consider the images  $\{k_x | x \in \mathcal{X}\}$  as a spanning set of a vector space. That is, we define the space  $\mathcal{G}$  that contains all finite linear combinations of such functions:

$$\mathcal{G} := \left\{ \sum_{i=1}^r \alpha_i k(x_i, \cdot) \mid \alpha_i \in \mathbb{R}, r \in \mathbb{N}, x_i \in \mathcal{X} \right\}$$



# Reproducing kernel Hilbert space (3)

- ▶ Define a scalar product on  $\mathcal{G}$  as follows:

- ▶ For the spanning functions we define

$$\langle k_x, k_y \rangle = \langle k(x, \cdot), k(y, \cdot) \rangle := k(x, y)$$

- ▶ For general functions in  $\mathcal{G}$  the scalar product is then given as follows: If  $g = \sum_i \alpha_i k(x_i, \cdot)$  and  $f = \sum_j \beta_j k(y_j, \cdot)$  then

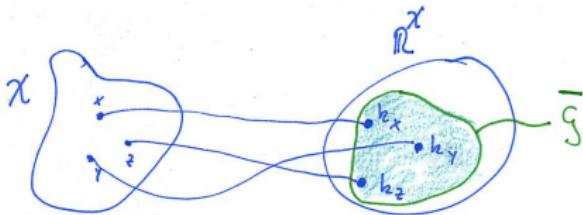
$$\langle f, g \rangle_{\mathcal{G}} := \sum_{i,j} \alpha_i \beta_j k(x_i, y_j)$$

To make sure that this is really a scalar product, we need to prove two things (EXERCISE!):

- ▶ Check that this is well-defined (not obvious because there might be several different linear combinations for the same function).
  - ▶ Check that it satisfies all properties of a scalar product (crucial ingredient is the fact that  $k$  is positive definite. )

# Reproducing kernel Hilbert space (4)

- ▶ Finally, to make  $\mathcal{G}$  a proper Hilbert space we need to take its topological completion  $\overline{\mathcal{G}}$ , that is we add all limits of Cauchy sequences.



- ▶ The resulting space  $\mathcal{H} := \overline{\mathcal{G}}$  is called the **reproducing kernel Hilbert space**.
- ▶ By construction, it has the property that

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

# Reproducing kernel Hilbert space (5)

## The reproducing property:

Let  $f = \sum_i \alpha_i k(x_i, \cdot)$ . Then  $\langle f, k(x, \cdot) \rangle = f(x)$ .

## Proof.

$$\begin{aligned}\langle k(x, \cdot), f \rangle &= \langle k(x, \cdot), \sum_i \alpha_i k(x_i, \cdot) \rangle \\ &= \sum_i \alpha_i \langle k(x_i, \cdot), k(x, \cdot) \rangle \\ &= \sum_i \alpha_i k(x_i, x) \\ &= f(x)\end{aligned}$$



# Reproducing kernel Hilbert space (6)

For those who know a bit of functional analysis:

- ▶ Let  $\mathcal{H}$  be a Hilbert space of functions from  $\mathcal{X}$  to  $\mathbb{R}$ . Then  $\mathcal{H}$  is a reproducing kernel Hilbert space if and only if all evaluation functionals  $\delta_x : \mathcal{H} \rightarrow \mathbb{R}$ ,  $f \mapsto f(x)$  are continuous.
- ▶ In particular, functions in an RKHS are pointwise well defined (as opposed to, say, function in an  $L_2$ -space which are only defined almost everywhere).
- ▶ Given a kernel, the RKHS is unique (up to isometric isomorphisms). Given an RKHS, the kernel is unique.
- ▶ There is a close connection to the Riesz representation theorem.

# The representer theorem

- ▶ In general, the RKHS is an infinite-dimensional vector space (a basis has to contain infinitely many vectors).
- ▶ The next theorem shows that in practice, we only have to deal with a finite-dimensional subspace.
- ▶ This subspace is still pretty large, later we discuss how to avoid overfitting!

# The representer theorem (2)

## Theorem 13 (Representer theorem)

Consider a regularized risk minimization problem of the form

$$\underset{f \in \mathcal{H}}{\text{minimize}} R_n(f) + \lambda \Omega(\|f\|_{\mathcal{H}}) \quad (*)$$

where  $\mathcal{X}$  arbitrary input space,  $\mathcal{Y}$  output space,  $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  a kernel,  $\mathcal{H}$  the corresponding feature space (RKHS) with norm  $\|\cdot\|_{\mathcal{H}}$ . For a given training set  $(X_i, Y_i)_{i=1,\dots,n} \subset \mathcal{X} \times \mathcal{Y}$ , let  $R_n$  be the empirical risk of a classifier  $f : \mathcal{X} \rightarrow \mathbb{R}$  with respect to a loss function  $\ell$ , and  $\Omega : [0, \infty[ \rightarrow \mathbb{R}$  a strictly monotonically increasing function. Then problem  $(*)$  always has an optimal solution of the form

$$f^*(x) = \sum_{i=1}^n \alpha_i k(X_i, x).$$

# The representer theorem (3)

## Proof intuition.

- ▶ Split the space  $\mathcal{H}$  into the subspace  $\mathcal{H}_{data} := \text{span}\{k_{X_1}, \dots, k_{X_n}\}$  (induced by the data) and its orthogonal complement  $\mathcal{H}_{comp}$ . Then  $\mathcal{H} = \mathcal{H}_{data} + \mathcal{H}_{comp}$ .
- ▶ Now express each vector  $f \in \mathcal{H}$  as  $f = f_{data} + f_{comp}$ .
- ▶ It is not difficult to see that all functions with the same  $f_{data}$  agree on all training points.
- ▶ So in particular, the loss function of  $f$  is not affected by  $f_{comp}$ .
- ▶ For fixed  $f_{data}$ , the norm of  $f$  is smallest if  $f_{comp}$  is 0.
- ▶ So if we had a solution  $f^*$  where  $f_{comp}$  would be non-zero, we could get a better solution by setting  $f_{comp}$  to zero.
- ▶ Thus we can always find an optimal solution has  $f_{comp} = 0$ .



## The representer theorem (4)

Intuitively, this theorem implies the following:

- ▶ We have seen that for any given kernel  $k$  there exists a feature space  $\mathcal{H}$ .
- ▶ However, this space was a function space that usually is an infinite-dimensional Hilbert space.
- ▶ The representer theorem now says that for any finite data set with  $n$  points, we don't need to deal with all the infinitely many dimensions, but we are only confronted with a space of at most  $n$  dimensions.
- ▶ As any  $n$ -dimensional subspace of a Hilbert space is isomorphic to  $\mathbb{R}^n$ , we can simply assume that our feature map goes to  $\mathbb{R}^n$ .
- ▶ This makes our lives much easier.

## Injective feature map

Note that without any further assumptions, the feature map  $\Phi : \mathcal{X} \rightarrow \mathcal{H}$  of a kernel  $k$  does not need to be injective!

(Simple counterexample:  $k(x, y) = \langle x, y \rangle^2$ ).

However, a kernel for which the feature map is not injective might not be too useful (WHY?)

A particular class of “nice” kernels are universal kernels:

## (\*) Universal kernels

A continuous kernel  $k$  on a compact metric space  $\mathcal{X}$  is called **universal** if the RKHS  $\mathcal{H}$  of  $k$  is dense in  $C(\mathcal{X})$ , that is for every function  $g \in C(\mathcal{X})$  and all  $\varepsilon > 0$  there exists a function  $f \in \mathcal{H}$  such that  $\|f - g\|_\infty \leq \varepsilon$ .

Intuition: with a universal kernel, we can approximate pretty much any function we like: all continuous functions, and all functions that can be approximated by continuous functions (such as step functions). In particular, we can separate any pair of disjoint compact subsets from each other.

Example:

- ▶ The Gaussian kernel with fixed kernel width  $\sigma$  on a compact subset  $\mathcal{X}$  of  $\mathbb{R}^d$  is universal.

## (\*) Universal kernels (2)

- ▶ Related statements can also be proved if we let  $\sigma \rightarrow 0$  slowly as  $n \rightarrow \infty$ .
- ▶ Polynomial kernels are not universal.

The kernel being universal is a necessary requirement if we want to construct learning algorithms that are uniformly Bayes consistent.

Universal kernels have many nice properties. For example, their feature maps are injective.

For details and proofs see the book by Steinwart / Christmann:  
Support Vector Machines. Springer 2008.

# Kernels — history

- ▶ Reproducing kernel Hilbert spaces play a big role in mathematics, they have been invented by Aronszajn in 1950. He already proved all of the key properties.  
*Aronszajn. Theory of Reproducing Kernels. Transactions of the American Mathematical Society, 1950*
- ▶ The feature space interpretation has first been published by Aizerman 1964, but in a different context. At that time the potential of the method had not been realized.  
*Aizerman, Braverman, Rozonoer: Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control, 1964.*

## Kernels — history (2)

- ▶ Then it was rediscovered in the context of the SVM in 1992:  
*Boser, Bernhard E.; Guyon, Isabelle M.; and Vapnik, Vladimir N.; A training algorithm for optimal margin classifiers. Conference on Learning Theory (COLT), 1992*
- ▶ Since then, kernels and the kernel trick became extremely popular, the first text books already appeared pretty soon, e.g. Schölkopf / Smola 2002 and Shawe-Taylor / Cristianini 2004.

# Kernel algorithms

In the following, we are now going to see a couple of algorithms that all use the kernel trick. The roadmap is always the same:

- ▶ Start with a linear algorithm
- ▶ Try to write this algorithm (both training and testing parts) in such a way that the only access to training points is in terms of scalar products (this is often possible but not always; sometimes it is simple, sometimes it is difficult).
- ▶ Then replace the scalar product by the kernel function.

In the machine learning lingo: we **kernelize** the algorithm.

# Support vector machines with kernels

## Literature:

- ▶ Schölkopf / Smola
- ▶ Shawe-Taylor / Cristianini
- ▶ A very theoretical / mathematically deep treatment of the theory of kernels and support vector machines is the following book:  
*Steinwart / Christmann: Support Vector Machines. Springer, 2008.*

# SVMs with kernels

- ▶ Consider the dual (!) SVM problem
- ▶ Have seen: the only way it accesses the training points in terms of scalar products
- ▶ So replace  $\langle X_i, X_j \rangle$  by  $k(X_i, X_j)$  everywhere
- ▶ The result is the dual of the “kernelized” SVM.

Formally, this looks as follows:

## SVMs with kernels (2)

Given input training points  $(X_i, Y_i)_{i=1,\dots,n}$  and a kernel function  $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ .

### Kernelized dual SVM problem:

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

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

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

Solving this problem gives the dual variables  $\alpha$ .

## SVMs with kernels (3)

**Computing labels at new points:** Have already seen how to compute the label of a test points for known  $\alpha$ :

- $w = \sum_i \alpha_i Y_i X_i$
- $b = Y_j - \sum_i Y_i \alpha_i \langle X_i, X_j \rangle$  for some  $j$  such that  $\alpha_j > 0$ .
- Label of test point  $X$  given by  $\langle w, X \rangle + b$

In kernel language:

$$\begin{aligned}\langle w, X \rangle + b &= \left\langle \sum_i \alpha_i Y_i X_i, X \right\rangle + b \\ &= \sum_i \alpha_i Y_i k(X_i, X) + \left( Y_j - \sum_i Y_i \alpha_i k(X_i, X_j) \right)\end{aligned}$$

**This is the approach that is typically used in practice.**

# The power of kernels

Why is the kernel framework so powerful? Let's look at one particular example, the Gaussian kernel.

- ▶ Have seen that the decision function of a kernelized SVM has the form

$$f(x) = \sum_i \beta_i k(x, X_i) + b$$

If  $k$  is a Gaussian kernel:

$$f(x) = \sum_i \beta_i \exp(-\|x - X_i\|^2/(2\sigma^2))$$

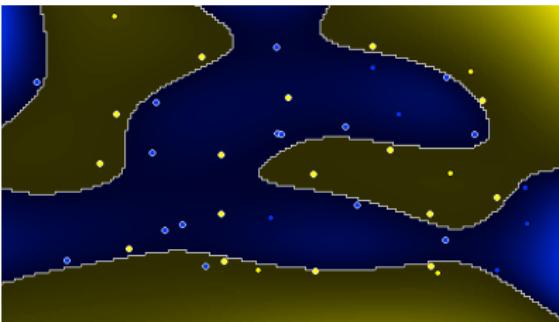
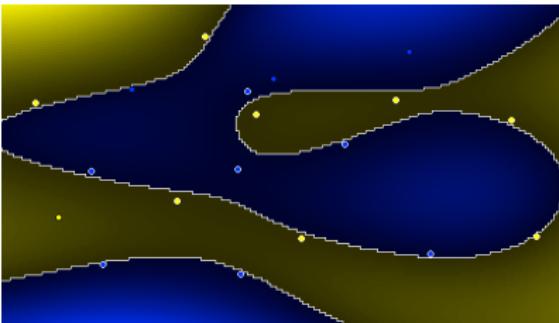
## The power of kernels (2)

- Important property: we can approximate any arbitrary continuous function  $g : \mathbb{R}^d \rightarrow \mathbb{R}$  by a sum of Gaussian kernels.



In particular, we can approximate any reasonable “decision surface” in  $\mathbb{R}^d$  by an SVM with Gaussian kernel:

# The power of kernels (3)



## The power of kernels (4)

- ▶ Kernels with this property are called “universal” kernels. One can prove that SVMs with universal kernels are universally consistent in the sense we defined in the very beginning of the lecture, that is they approximate the Bayes risk.
- ▶ Note: if the kernel is universal, the underlying function class is huge (all continuous functions can be approximated). All the more important is that we regularize!

# Regularization interpretation

Recall that we interpreted the linear primal SVM problem in terms of regularized risk minimization where the risk was the Hinge loss and we regularized by  $L_2$ -regularizer  $\|w\|^2$ .

How does it look for the kernelized SVM?

- ▶ The loss function is still the Hinge loss because the part with the variables  $\xi_i$  does not change.
- ▶ But the regularizer is now  $\|w\|^2$  where  $w$  is a vector in the feature space, and the norm is taken in the feature space.
- ▶ By the representer theorem,

$$\|w\|^2 = \left\langle \sum_i \beta_i \Phi(X_i), \sum_j \beta_j \Phi(X_j) \right\rangle = \beta^t K \beta$$

## Regularization interpretation (2)

- ▶ It is not so easy to gain intuition about this norm (obviously it depends on the kernel). But at least we can say that the regularization “restricts the size of the function space” (in the sense that there are fewer functions that can be expressed with  $w$  with low norm than with high norm).

## Regularization interpretation (3)

This regularization interpretation is really important, otherwise the SVM “could not work”:

- ▶ We implicitly embed our data in a very high-dimensional space.
- ▶ In high-dimensional spaces, it happens very easily that we overfit.
- ▶ The only way we can circumvent this is to regularize.
- ▶ This is what the kernelized SVM does.

# Why are SVMs so successful?

Before SVMs, there were neural networks. They are great, but they have a couple of drawbacks:

- ▶ Lots of parameters to tune (design choices to make: how many neurons, how many layers, etc)
- ▶ Training a neural network is a non-convex problem
- ▶ To be able to successfully work with neural networks one needs a large amount of experience.

Then came SVMs, they revolutionized the field. Why?

- ▶ Convex optimization problem, easy to implement
- ▶ Very few variables to tune ( $C$ , and maybe a kernel parameter such as  $\sigma$  in the Gaussian kernel), this can be done by cross validation

## Why are SVMs so successful? (2)

- ▶ Appealing from a conceptual side (large margin principle) and also from the mathematical point of view (support vector property, representer theorem, etc).
- ▶ The kernel framework boosts the potential of the SVM to the non-linear regime, but does not lead to excessive overfitting.
- ▶ Statistical learning theory shows many nice guarantees about the SVM (consistency, etc).

# Kernel SVMs in practice

If you want to use SVMs in practice, here is the vanilla approach:

- ▶ Come up with a good kernel that encodes a “natural notion” of similarity (sometimes easy, sometimes not).
- ▶ Train an SVM by some standard package (there are lots of SVM packages out there; for matlab, my favorite one is libSVM)
- ▶ **MAKE SURE YOU SET ALL PARAMETERS BY CROSS VALIDATION!**

The results are very sensitive to the choice of the regularization parameter  $C$  and the kernel parameters (such as  $\sigma$  for the Gaussian kernel).

Later in the lecture we will look at more preprocessing steps that you should use ( $\leadsto$  non-vanilla-version).

## (\*) Kernelizing the SVM primal

AS AN EXERCISE: IT IS POSSIBLE TO EXPRESS THE PRIMAL OPTIMIZATION PROBLEM IN TERMS OF KERNELS?

$$\begin{aligned} & \text{minimize}_{w \in \mathcal{H}} \|w\|_{\mathcal{H}}^2 + \frac{C}{n} \sum_{i=1}^n \xi_i \\ & \text{subject to } Y_i \left( \langle w, \Phi(X_i) \rangle_{\mathcal{H}} \right) \geq 1 - \xi_i \quad (i = 1, \dots, n) \end{aligned}$$

## (\*) Kernelizing the SVM primal (2)

- ▶ A priori, we cannot write the primal function just in terms of scalar products because it contains a scalar product between the variable we are looking for ( $w$ ) and the input points ( $X_i$ ).
- ▶ But according to the representer theorem, the solution vector  $w$  can always be written as a linear combination of input feature vectors, that is  $w = \sum_i \beta_i \Phi(X_i)$ .
- ▶ Consequently,

$$\|w\|^2 = \langle w, w \rangle = \sum_{i,j} \beta_i \beta_j k(X_i, X_j)$$

and

$$\langle w, \Phi(X_j) \rangle = \sum_i \beta_i \langle \Phi(X_i), \Phi(X_j) \rangle = \sum_i \beta_i k(X_i, X_j)$$

## (\*) Kernelizing the SVM primal (3)

- With this knowledge we can also kernelize the primal problem:

$$\underset{\beta \in \mathbb{R}^d, b \in \mathbb{R}, \xi \in \mathbb{R}^d}{\text{minimize}} \frac{1}{2} \sum_{i,j} \beta_i \beta_j k(X_i, X_j) + \frac{C}{n} \sum_{i=1}^n \xi_i$$

$$\text{subject to } Y_i \left( \sum_{j=1}^n \beta_j k(X_j, X_i) + b \right) \geq 1 - \xi_i \quad (i = 1, \dots, n)$$

## Summary: SVM with kernels

- ▶ Given data points in some space  $\mathcal{X}$  and a kernel function  $k$  on this space
- ▶ Want to solve classification.
- ▶ By the kernel trick, we embed our data points into some abstract feature space and use a linear classifier in this space.
- ▶ The inductive principle is that the margin in this feature space should be large.
- ▶ All this leads to a convex optimization problem that can be solved efficiently.
- ▶ There are lots of important properties (support vector property, representer theorem, etc).
- ▶ The kernel SVM is equivalent to regularized risk minimization with the Hinge loss and regularization by the squared norm in the feature space.

## Summary: SVM with kernels (2)

The kernel SVM is the most important classification algorithm that is out there. If you just remember one thing from this whole course, try to remember SVMs 😊

# Regression methods with kernels

# Kernelized least squares

## Least squares revisited

We had already seen in the beginning how to solve least squares regression in a feature space (at that point we called it differently, regression with basis functions):

Given data in some space  $\mathcal{X}$ , a mapping  $\Phi : \mathcal{X} \rightarrow \mathbb{R}^d$ . Already seen: The least squares problem in feature space

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \frac{1}{n} \sum_{i=1}^n (Y_i - \langle \Phi(X_i), w \rangle)^2$$

has the analytic solution  $w^* = (\Phi^t \Phi)^{-1} \Phi^t Y$ .

We are now going to rewrite everything using kernels.

# Kernelizing least squares (first method via representer theorem)

- ▶ The representer theorem tells us that the least squares problem always has a solution of the form

$$w^* = \sum_{j=1}^n \alpha_j \Phi(X_j).$$

- ▶ Plugging this in the objective gives

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \frac{1}{n} \sum_{i=1}^n (Y_i - \langle \Phi(X_i), w \rangle)^2$$

$$\iff \underset{\alpha \in \mathbb{R}^n}{\text{minimize}} \frac{1}{n} \sum_{i=1}^n \left( Y_i - \sum_{j=1}^n \alpha_j \underbrace{\langle \Phi(X_i), \Phi(X_j) \rangle}_{k_{ij}} \right)^2$$

# Kernelizing least squares (first method via representer theorem) (2)

- In matrix notation:

$$\underset{\alpha \in \mathbb{R}^n}{\text{minimize}} \frac{1}{n} \|Y - K\alpha\|^2$$

- By taking the derivative with respect to  $\alpha$  and exploiting that  $K$  is pd it is easy to see that the solution is given as  $\alpha^* = K^{-1}Y$  (EXERCISE!).
- To evaluate the solution on a new data point  $x$ , we need to compute

$$f(x) = \langle \Phi(x), w^* \rangle = \sum_j \alpha_j^* \langle \Phi(x), \Phi(X_j) \rangle = \sum_j \alpha_j^* k(x, X_j)$$

- So we can express the optimization problem, its solution and the evaluation function purely in terms of kernel functions. We have kernelized least squares.

## (\*) Kernelizing least squares (second method via SVD)

Recap: the kernel matrix is  $\Phi\Phi^t$

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

## (\*) Kernelizing least squares (second method via SVD) (2)

### Proposition 14 (Matrix Identities)

For any  $n \times d$ -matrix  $\Phi$  we have

$$(\Phi^t \Phi)^{-1} \Phi^t = \Phi^t (\Phi \Phi^t)^{-1}$$

### Proof of the proposition.

- ▶ Let  $\Phi = U \Sigma V^t$  the singular value decomposition of  $\Phi$ .
- ▶ It is straightforward to prove that  $(\Phi^t \Phi)^{-1} \Phi^t = V \Sigma^+ U^t$  (have seen this already when we derived least squares).
- ▶ It is even more straightforward to see that  $\Phi^t (\Phi \Phi^t)^{-1} = V \Sigma^+ U^t$ . 

## (\*) Kernelizing least squares (second method via SVD) (3)

Using this proposition and the fact that the kernel matrix  $K$  is given as  $\Phi\Phi^t$  we can rewrite the least squares solution as

$$\mathbf{w}^* = (\Phi^t \Phi)^{-1} \Phi^t Y = \Phi^t (\Phi \Phi^t) Y^{-1} = \Phi^t K^{-1} Y$$

Denote  $\alpha := K^{-1}Y$ . With this notation, the evaluation function is

$$\begin{aligned} f(x) &= \langle w^*, \Phi(x) \rangle = (w^*)^t \Phi(x) \\ &= (\Phi^t K^{-1} Y)^t \Phi(x) = Y^t K^{-1} \Phi^t \Phi(x) \\ &= \alpha^t \Phi \Phi^t(x) \\ &= \sum_{j=1}^n \alpha_j \Phi(X_j)^t \Phi(x) \\ &= \sum_{j=1}^n \alpha_j k(X_j, x) \end{aligned}$$

## (\*) Kernelizing least squares (second method via SVD) (4)

So we can express the optimization problem, its solution and the evaluation function purely in terms of kernel functions. We have kernelized least squares.

# Kernel ridge regression

# Ridge regression

Recall ridge regression in feature space:

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \frac{1}{n} \sum_{i=1}^n (Y_i - \langle w, \Phi(X_i) \rangle)^2 + \lambda \|w\|^2$$

Again use the representer theorem to express  $w$  as a linear combination of input points:

$$w = \sum_{j=1}^n \alpha_j \Phi(X_j)$$

## Ridge regression (2)

This leads to the following kernelized ridge regression problem:

$$\underset{\alpha \in \mathbb{R}^n}{\text{minimize}} \frac{1}{n} \|Y - K\alpha\|^2 + \lambda \alpha^t K \alpha$$

The solution is given by

$$\alpha = (n\lambda I + K)^{-1} Y$$

As before, we can compute the prediction for a new test point just using kernels.

# A subtle difference: Ridge regression vs. kernel ridge regression

- ▶ Given points in space  $\mathcal{X}$ . Want to compare:
  - ▶ Ridge regression using basis functions  $\Phi_i(x) = k(X_i, x)$
  - ▶ Kernel ridge regression using kernel  $k$  and feature map  $\Phi$
- ▶ In both cases, we work in the same function space, namely the one spanned by the functions  $\Phi_i$ .
- ▶ So no matter which function we use, the least squares error is the same in both approaches.
- ▶ **However, the regularizers are different:**
  - ▶ In the standard case we regularize by  $\|\alpha\|^2$ .
  - ▶ In the kernel case we regularize by  $\alpha^t K \alpha$ .
- ▶ This is as for linear and kernel SVMs ...

# How to center and normalize in the feature space

Literature:

- ▶ Shawe-Taylor / Cristianini Section 5.1

# What we want to do

- ▶ Have seen: many algorithms require that the data points are centered (have mean = 0) and are normalized.
- ▶ However, now we want to work in feature space, but without explicitly working with the coordinates in feature space.
- ▶ So how can we do this ???

## Centering in the feature space

To center points in the feature space, we would need to perform the following calculations:

- ▶ Compute center:  $\bar{\Phi} := 1/n \sum_i \Phi(x_i)$
- ▶ Replace  $\Phi(x_i)$  by  $\Phi(x_i) - \bar{\Phi}$

Now observe: if we compute the kernel matrix of the centered data points, we obtain the following:

## Centering in the feature space (2)

$$\begin{aligned}(\tilde{K})_{ij} &:= \langle \Phi(x_i) - \bar{\Phi}, \Phi(X_j) - \bar{\Phi} \rangle \\&= \langle \Phi(x_i) - 1/n \sum_{s=1}^n \Phi(x_s), \Phi(x_j) - 1/n \sum_{s=1}^n \Phi(x_s) \rangle \\&= k(X_i, X_j) - \frac{1}{n} \sum_{s=1}^n k(X_i, X_s) - \frac{1}{n} \sum_{s=1}^n k(X_j, X_s) \\&\quad + \frac{1}{n^2} \sum_{s,t=1}^n k(X_s, X_t)\end{aligned}$$

## Centering in the feature space (3)

In matrix notation, this means that we can compute the centered kernel matrix as follows:

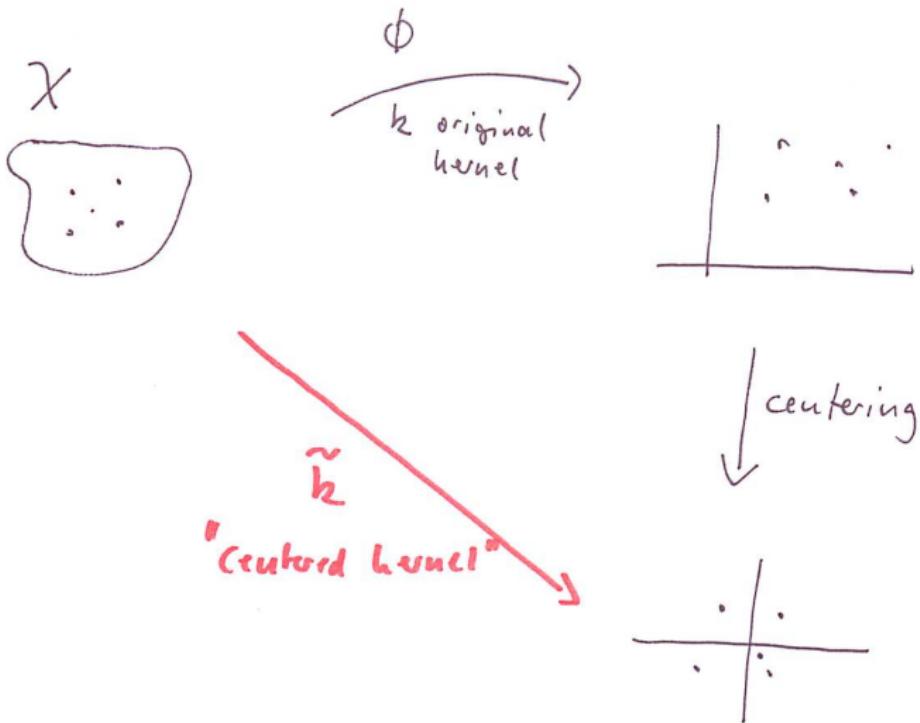
$$\tilde{K} = (K - \mathbb{1}_n K - K \mathbb{1}_n + \mathbb{1}_n K \mathbb{1}_n)$$

where  $\mathbb{1}_n$  is the  $n \times n$  matrix containing  $1/n$  as each entry.

Good news:

- ▶ We do not have to do the centering operation explicitly.
- ▶ We can implicitly center the data by replacing the “old” kernel matrix  $K$  by the new matrix  $\tilde{K}$ .

# Centering in the feature space (4)



# Normalizing in feature space

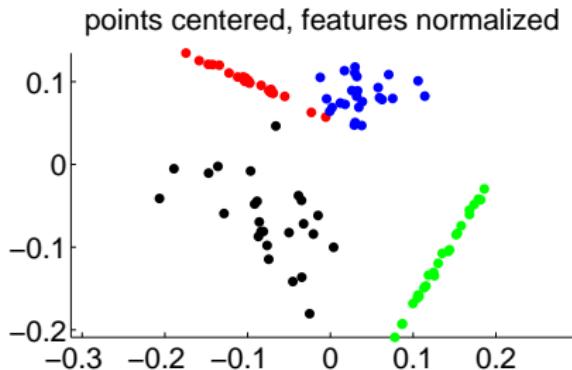
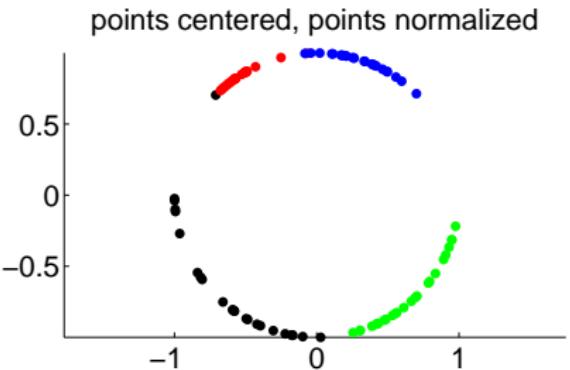
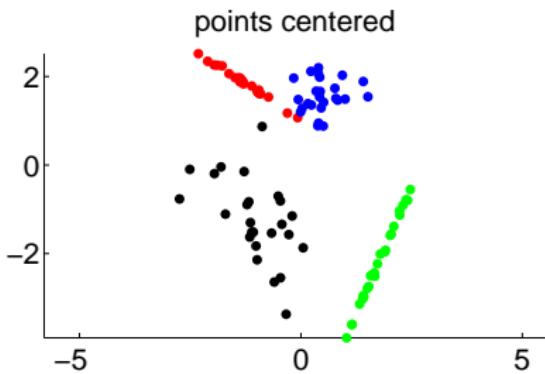
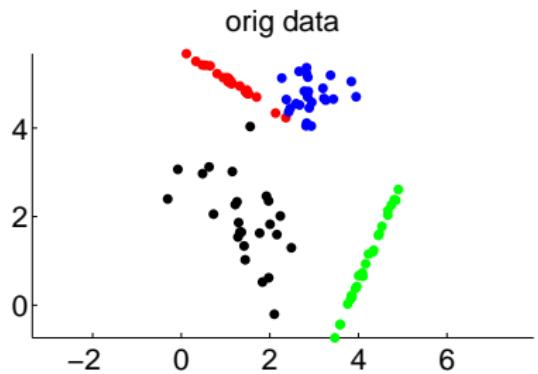
Assume our  $n$  data points are in  $\mathbb{R}^d$  and we stack them in a data matrix  $X$  as usual:

- ▶ Each row of  $X$  corresponds to one data point.
- ▶ The data matrix has dimensions  $n \times d$

Two different ways to normalize data:

- ▶ **Normalize the data points**, that is rescale each data point such that it has norm 1. This is equivalent to normalizing the **rows** of the centered matrix to have unit norm.
- ▶ **Normalize the individual features**, that is rescale all **columns** of the centered matrix to have norm 1. This is just a rescaling of the coordinate axes such that the variance in each coordinate direction is 1.

# Normalizing in feature space (2)



# Normalizing in feature space (3)

Note:

- ▶ Normalize the points
  - ▶ We will see below that there is a way to normalize points in the feature space.
  - ▶ Sometimes it helps, sometimes it hurts.
  - ▶ If in doubt, use cross-validation to see whether your results improve if you normalize or not.
- ▶ Normalize features:
  - ▶ Typically this never hurts, and often helps.
  - ▶ For kernel methods it is impossible to normalize the features (we don't know the embedding  $\Phi$  explicitly, in particular we don't know what the features are).

## Normalizing in feature space (4)

To normalize the points such that they have unit norm in the feature space:

- ▶ Assume the data points are already centered in feature space.
- ▶ Then define the normalized data point  
 $\hat{\Phi}(X) := \Phi(X)/\|\Phi(X)\|.$
- ▶ Observe:

$$\begin{aligned}\langle \hat{\Phi}(X), \hat{\Phi}(Y) \rangle &= \left\langle \frac{\Phi(X)}{\|\Phi(X)\|}, \frac{\Phi(Y)}{\|\Phi(Y)\|} \right\rangle \\ &= \frac{\langle \Phi(X), \Phi(Y) \rangle}{\|\Phi(X)\| \|\Phi(Y)\|} \\ &= \frac{k(x, y)}{\sqrt{k(x, x)k(y, y)}}\end{aligned}$$

## Normalizing in feature space (5)

So instead of first normalizing the points and then computing their kernels we can directly compute the kernels for the normalized data points.

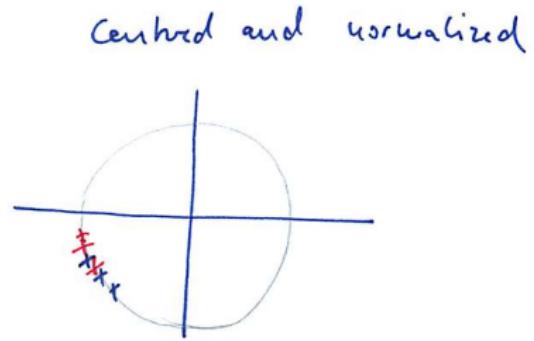
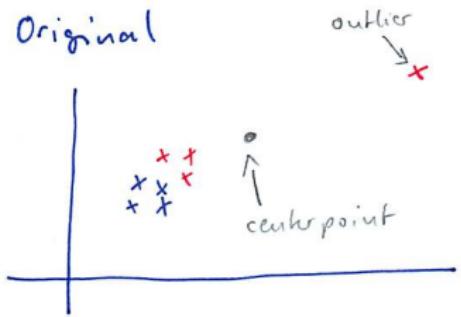
So to normalize the points in feature space, we simply replace the kernel function  $k$  by the normalized kernel function

$$\hat{k}(x, y) = \frac{k(x, y)}{\sqrt{k(x, x)k(y, y)}}.$$

(VERIFY THAT THIS IS INDEED A KERNEL)

# When it can go wrong

Standardizing the data is a preprocessing step. As any such step, it often helps, but sometimes can go wrong:



# More supervised learning algorithms

(\*) Random Forests: SKIPPED

(\*) Boosting: SKIPPED

# Unsupervised learning

# Dimensionality reduction and embedding

# PCA and kernel PCA

Classical PCA is covered in many statistics books:

- ▶ A complete book on PCA is Jolliffe: Principal Component Analysis. Springer, 2002.
- ▶ Chapter 8 in Mardia, Kent, Bibby: Multivariate Analysis. Academic Press, 1979. A classic.

Literature on kernel PCA:

- ▶ Chapter 14.2 of Schölkopf and Smola
- ▶ Chapter 6.2. of Shawe-Taylor and Cristianini

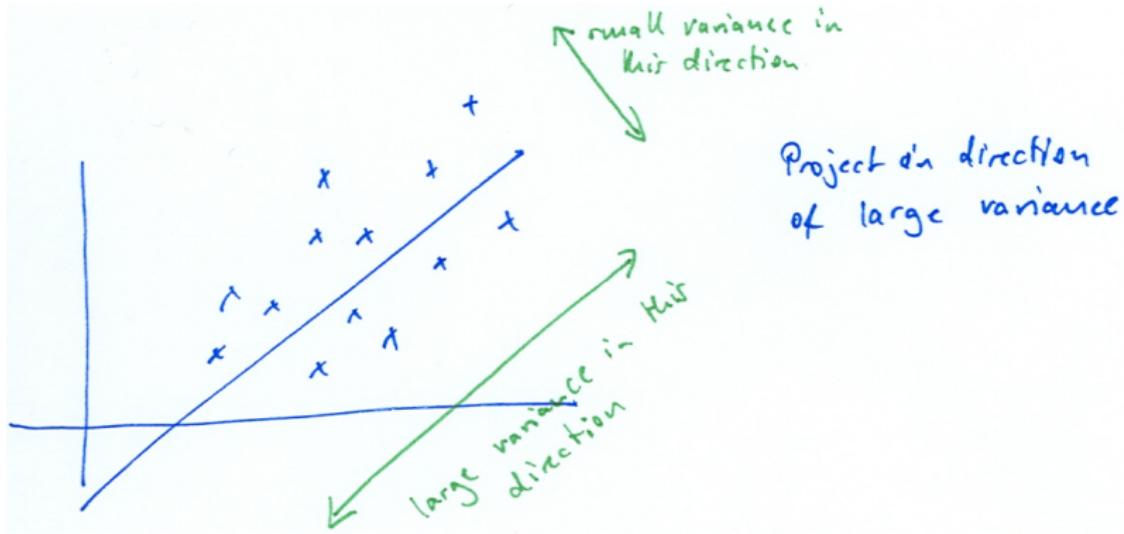
# Principal component analysis (PCA)

... is a “traditional” method for unsupervised dimensionality reduction. Is based on linear principles.

Goal:

- ▶ Given data points  $x_1, \dots, x_n \in \mathbb{R}^d$
- ▶ want to reduce the dimensionality of the data by throwing away “dimensions which are not important”.
- ▶ Result is set of new data points  $y_1, \dots, y_n \in \mathbb{R}^\ell$  with  $\ell < d$ .

# Principal component analysis (PCA) (2)



# Principal component analysis (PCA) (3)

Three approaches in this lecture:

- ▶ Traditional approach: Maximize the variance of the reduced data  $\leadsto$  Covariance matrix approach
- ▶ Traditional approach: Minimize the quadratic error  $\leadsto$  SVD approach
- ▶ Kernel PCA

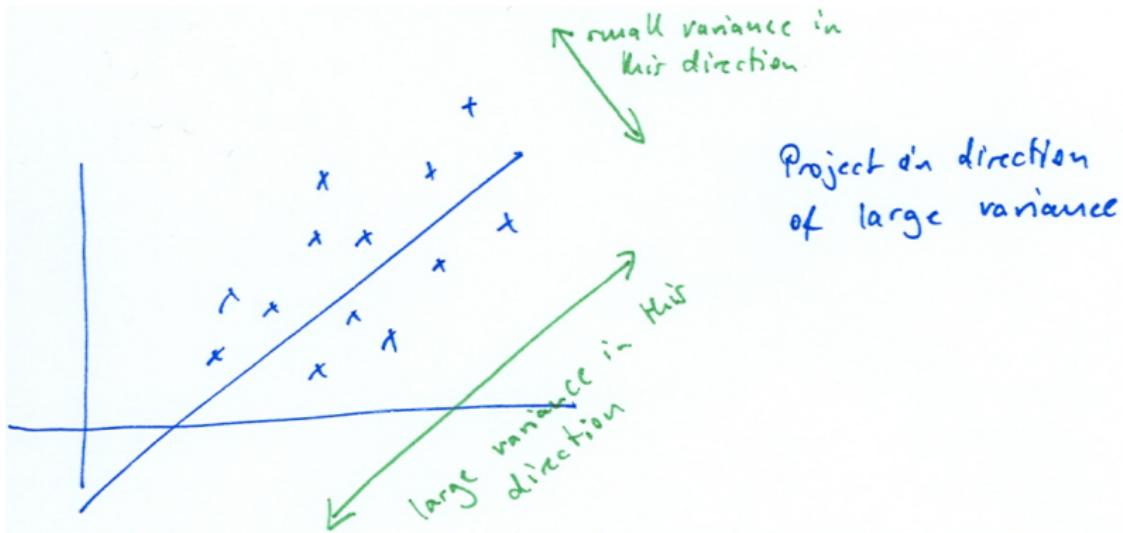
# Recap: Projections

... see slides in the appendix (slides 1224 ff.)

# Recap: Variance and Covariance

... see slides in the appendix (slides 1171 ff.)

# PCA by max variance approach: Idea



Want to find a linear projection on a low-dim space such that the overall variance of the resulting points is as large as possible.

## PCA by max variance approach: Idea (2)

Given: data points  $x_1, \dots, x_n \in \mathbb{R}^d$ , parameter  $\ell < d$  (the dimension of the space we want to project to).

Goal: find a projection  $\pi_S$  on an affine subspace  $S$  such that the variance of the projected points is maximized:  $\max_S \text{Var}_\ell(\pi_S(X))$ .

For simplicity, let us assume that the data points are centered:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i = 0$$

(If this is not the case, we can center the data points by setting  $\tilde{x}_i = x_i - \bar{x}$ .)

# PCA by max variance approach: Case $\ell = 1$

One-dimensional case:

We first of all assume that  $\ell = 1$ , that is we want to project the data points on a 1-dim space.

Have to solve the following optimization problem:

$$\begin{aligned} & \max_{a \in \mathbb{R}^d, \|a\|=1} \text{Var}(\pi_a(X)) \\ \iff & \max_{a \in \mathbb{R}^d} \sum_{i=1}^n (\pi_a(x_i))^2 \quad \text{subject to } a'a = 1 \\ \iff & \max_{a \in \mathbb{R}^d} \sum_{i=1}^n (a'x_i)^2 \quad \text{subject to } a'a = 1 \\ \iff & \max_{a \in \mathbb{R}^d} \|Xa\|^2 \quad \text{subject to } a'a = 1 \end{aligned}$$

# PCA by max variance approach: Case $\ell = 1$ (2)

To solve this:

- ▶ Write the Lagrangian:

$$L(a, \lambda) = \|Xa\|^2 - \lambda(a'a - 1) = a'X'Xa - \lambda(a'a - 1)$$

- ▶ Compute the partial derivatives wrt  $a$ :

$$\partial L / \partial a = X'Xa - \lambda a \stackrel{!}{=} 0$$

Thus necessary condition:  $a$  is an eigenvector of  $X'X$ .

- ▶ Substitute  $X'Xa = \lambda a$  in the original objective function:

$$a'X'Xa = \lambda a'a = \lambda$$

- ▶ This is maximal for  $a$  being the largest eigenvector of  $X'X$ .

Solution: If the data points are centered, then projecting on the largest eigenvector of  $C = X'X$  solves the problem for  $\ell = 1$ .

# PCA by max variance approach: Case $\ell > 1$

## Case $\ell > 1$ :

By similar arguments we can prove that we need to project the data on the space spanned by the  $\ell$  largest eigenvectors of  $X'X$ .

(and by “largest eigenvector” I mean the eigenvector corresponding to the largest eigenvalue).

# PCA: algorithm using covariance matrix C

Input: Data points  $x_1, \dots, x_n \in \mathbb{R}^d$ , parameter  $\ell \leq d$ .

- ▶ Center the data points, that is compute  $\tilde{x}_i = x_i - \bar{x}$  for all  $i$ .
- ▶ Compute the  $n \times d$  data matrix  $X$  with the centered data points  $\tilde{x}_i$  as rows, and the  $d \times d$  sample covariance matrix  $C = X'X$ .
- ▶ Compute the eigendecomposition  $C = VDV'$ .
- ▶ Define  $V_\ell$  as the matrix containing the  $\ell$  largest eigenvectors (i.e., the first  $\ell$  columns of  $V$  if the eigs in  $D$  are ordered decreasingly).
- ▶ Compute the new data points:
  - ▶ View 2:  $y_i = V_\ell' \tilde{x}_i \in \mathbb{R}^\ell$
  - ▶ View 1:  $z_i = P\tilde{x}_i + \bar{x} \in \mathbb{R}^d$  with  $P = V_\ell V_\ell'$

# PCA: algorithm using covariance matrix C (2)

Notation:

- ▶ The eigenvectors are called **principal axes** or **principal directions**.
- ▶ In View 1: the distance between a point and its projection is called the **reconstruction error** or **projection error**.

## Example: simple Gaussian toy data

demo\_pca.m

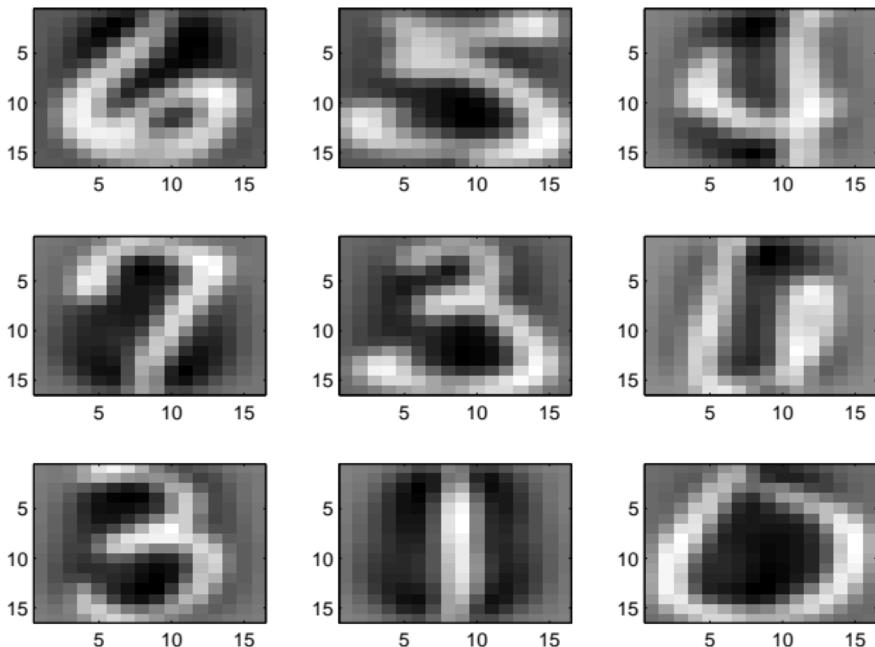
## USPS example

USPS handwritten digits, 16 x 16 greyscale images.

$\sim$  demo\_pca\_usps.m

# USPS example (2)

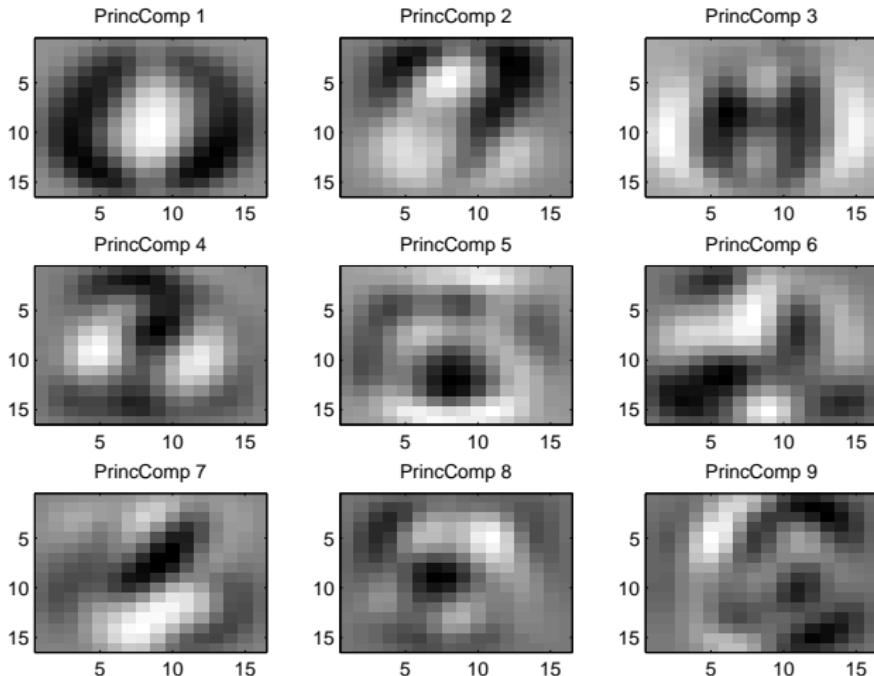
Some digits from the data set



# USPS example (3)

The first principal components (computed based on 500 digits):

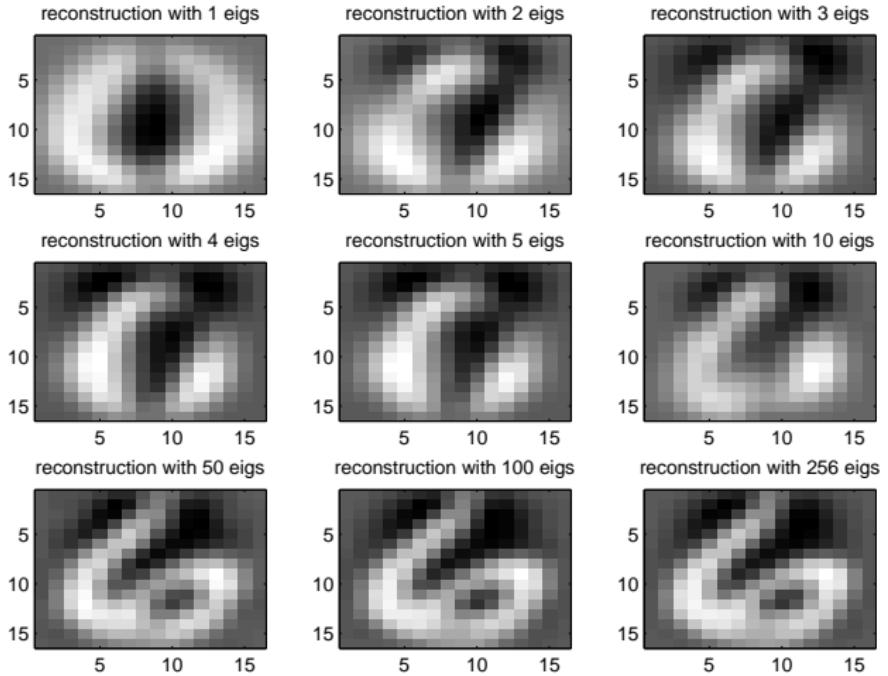
First principal components



# USPS example (4)

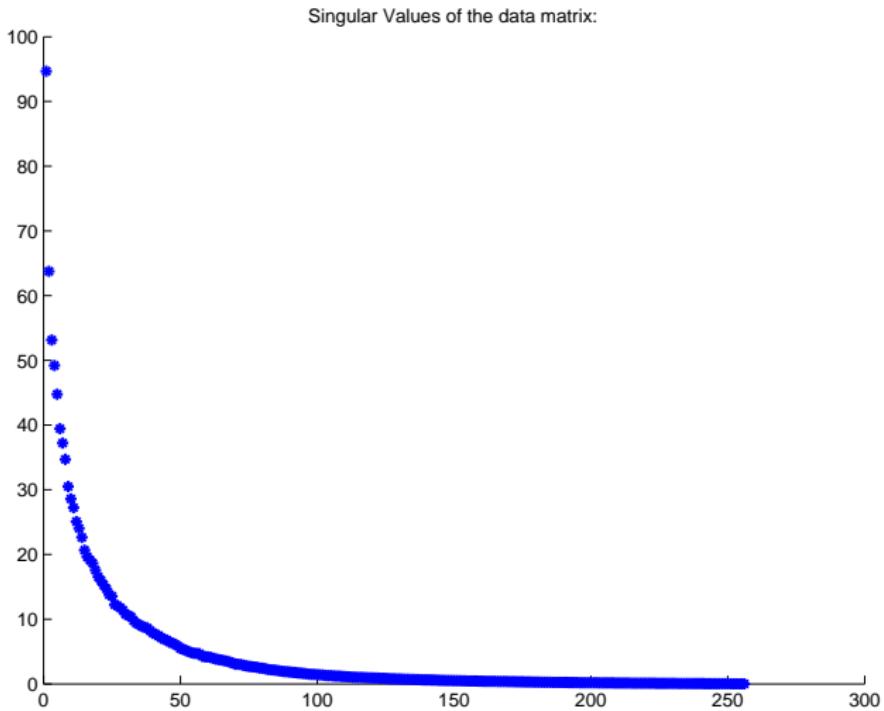
Reconstructing digits:

Reconstructed first digit



# USPS example (5)

All eigenvalues:



# Eigenfaces

Principal components for a data set of faces:



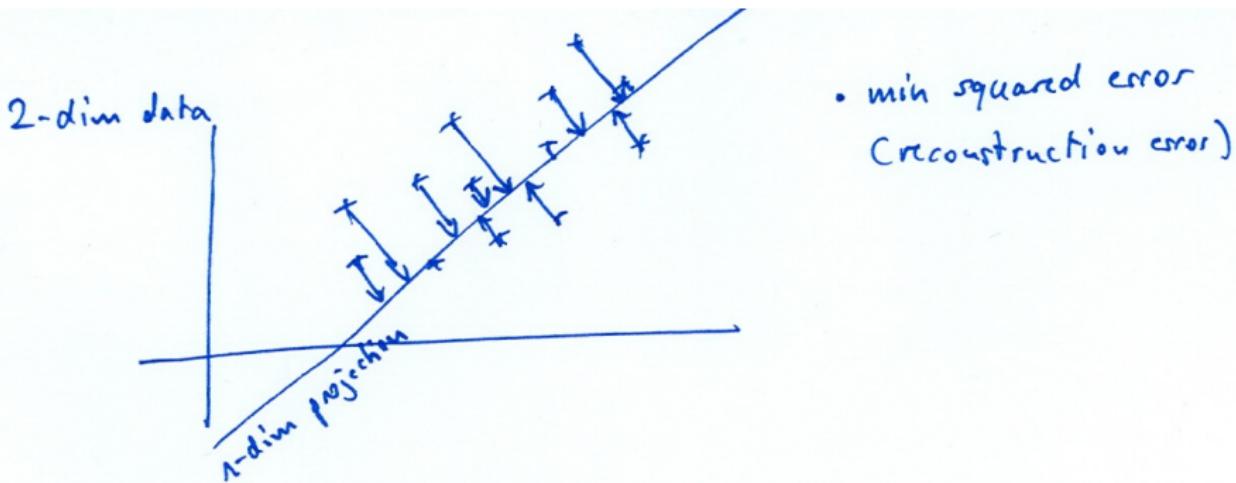
# PCA – min squared error approach

Second approach to PCA:

Find a projection  $\pi_S$  on an affine subspace  $S$  such that the squared distance between the points and their projections is minimized:

$$\min_S \sum_{i=1}^n \|x_i - \pi_S(x_i)\|^2.$$

## PCA – min squared error approach (2)

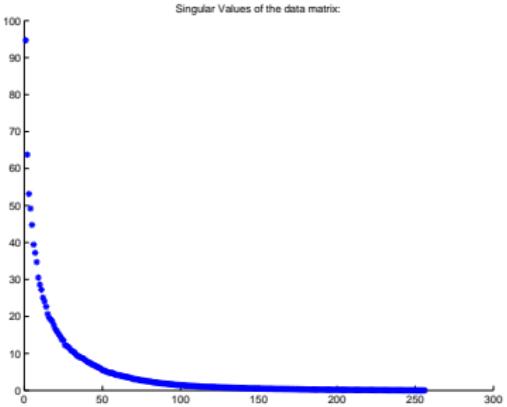


One can prove that this approach leads to exactly the same solution as the one induced by the max-variance criterion (we skip this derivation).

# Choosing the parameter $\ell$

- ▶ Heuristic: look at largest eigenvalues, and take the most “informative” ones. It also can be seen: the reconstruction error is bounded as

$$\sum_i \|x_i - \pi_\ell x_i\|^2 \leq \sum_{k=\ell+1}^n \lambda_k$$



## Choosing the parameter $\ell$ (2)

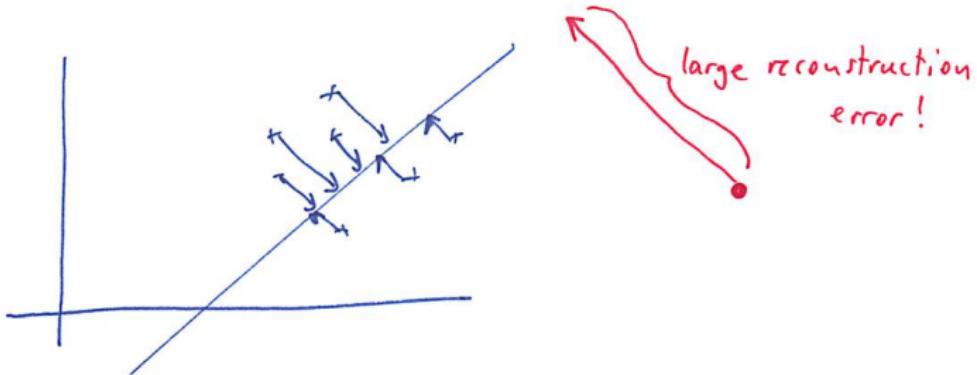
- If PCA is used as a preprocessing step for supervised learning, then use cross validation to set the parameter  $\ell$ !

Note: It is not a priori clear whether it is better to choose  $\ell$  large or small ...

# Global!

Keep in mind that PCA optimizes global criteria.

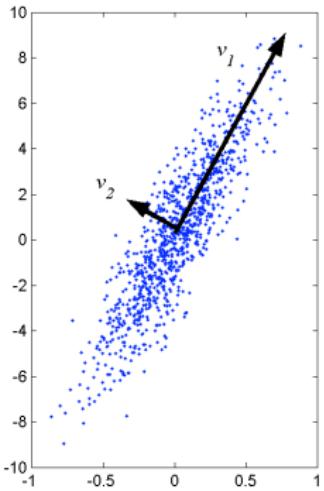
- ▶ No guarantees what happens to individual data points. This is different for some other dimensionality reduction methods (such as random projections and Johnson-Lindenstrauss).



- ▶ If the sample size is small, then outliers can have a large effect on PCA.

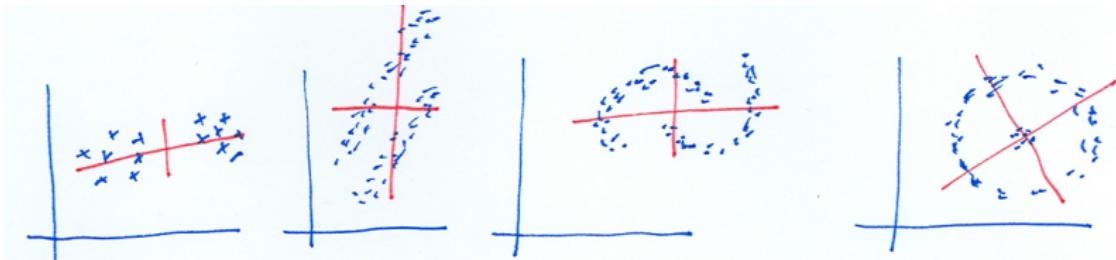
# When does it (not) make sense?

- ▶ The PCA works best if the data comes from a Gaussian.



## When does it (not) make sense? (2)

- ▶ But it can have very bad effects if the data is far from Gaussian:



# Towards kernel PCA

Now we want to kernelize the PCA algorithm to be able to have non-linear principal components.

Observe:

- ▶ PCA uses the covariance matrix — and this matrix inherently uses the actual coordinates of the data points.
- ▶ So how should we be able to kernelize PCA???
- ▶ The solution will be: there is a tight relationship between the covariance matrix and the kernel matrix.

## Recap: Covariance matrix vs. kernel matrix

Consider centered data points  $x_1, \dots, x_n$ , stacked in a data matrix  $X$  as rows. Denote the  $k$ -th column of the matrix by  $X^{(k)}$  (contains the  $k$ -th coordinate of all data points). Then:

- Covariance matrix is  $C = X'X$  because

$$C_{kl} = \text{Cov}_{1\text{dim}}(X^{(k)}, X^{(l)}) = \sum_{i=1}^n X_i^{(k)} X_i^{(l)} = (X'X)_{kl}$$

Also note that because  $X_i^{(k)} X_i^{(l)} = (x_i x_i')_{kl}$  this implies  
 $(X'X) = \sum_{i=1}^n (\underbrace{x_i x_i'}_{d \times d})$

- Kernel matrix is  $K = XX'$

(because  $(XX')_{ij} = \sum_{k=1}^d x_{ik} x_{jk} = x_i' x_j = \langle x_i, x_j \rangle$ ).

## Recap: Covariance matrix vs. kernel matrix (2)

What we now try to do is to express the eigenvalues/eigenvectors of  $C$  by those of  $K$  and vice versa.

# Expressing the eigs of $K$ by those of $C$

## Proposition 15 (Eigs of $K$ via eigs of $C$ )

Consider a set of points  $x_1, \dots, x_n \in \mathbb{R}^d$ . Assume that  $\lambda$  and  $a$  are an eigenvalue and corresponding eigenvector of the kernel matrix  $K$  (with respect to the standard scalar product). If  $v := X'a \neq 0$ , then it is an eigenvector of  $C$  with eigenvalue  $\lambda$ .

### Proof.

$$\begin{aligned}Ka &= \lambda a \\ \iff XX'a &= \lambda a \\ \implies X'XX'a &= \lambda X'a \\ \iff Cv &= \lambda v\end{aligned}$$

If  $v \neq 0$ , then it is an eigenvector of  $C$ .



## Expressing the eigs of $K$ by those of $C$ (2)

Main message: All eigs  $(\lambda, a)$  of  $K$  give rise to eigs  $(\lambda, v)$  of  $C$  unless  $v = X'a = 0$ .

Does it also work the other way round (express the eigenvectors of  $C$  by the ones of  $K$ )???

# Expressing the eigs of $C$ by those of $K$

## Proposition 16 (Eigs of $C$ via eigs of $K$ )

Assume that the points  $x_i$  are centered. Then to compute the  $\ell$ -th eigenvector  $v$  of  $C$  we can proceed as follows:

- ▶ compute the matrix  $K$
- ▶ compute its  $\ell$ -th eigenvector  $a = (a_1, \dots, a_n)' \in \mathbb{R}^n$
- ▶ make sure that  $\|a\| = 1$ .
- ▶ set  $v = \frac{1}{\sqrt{\lambda}} \sum_j a_j x_j$

**Proof in several steps:**

## Expressing the eigs of $C$ by those of $K$ (2)

**Step 1: non-zero eigenvectors of  $C$  are linear combinations of input points:**

$$\lambda v = Cv \text{ and } C = \sum_{j=1}^n x_j x'_j \text{ implies}$$

$$\textcolor{brown}{v} = \frac{1}{\lambda} Cv = \frac{1}{\lambda} \sum_{j=1}^n x_j x'_j v = \frac{1}{\lambda} \sum_{j=1}^n x_j \langle x_j, v \rangle =: \sum_j a_j x_j$$

# Expressing the eigs of $C$ by those of $K$ (3)

**Step 2: express eig of  $C$  as eig of  $K$ :**

$$\begin{aligned} Cv \stackrel{!}{=} \lambda v &\iff (\sum_{j=1}^n x_j x'_j) (\sum_{i=1}^n a_i x_i) \stackrel{!}{=} \lambda \sum_{i=1}^n a_i x_i \\ &\iff \sum_{i,j=1}^n x_j x'_j a_i x_i \stackrel{!}{=} \lambda \sum_{i=1}^n a_i x_i \end{aligned}$$

Now we multiply the equation with  $x'_s$  (for some  $s$ ) and obtain:

$$\begin{aligned} x'_s (\sum_{ij} a_i x_j \langle x_j, x_i \rangle) &\stackrel{!}{=} \lambda \sum_i a_i x'_s x_i \\ \iff \sum_{ij} a_i \langle x_s, x_j \rangle \langle x_j, x_i \rangle &\stackrel{!}{=} \lambda \sum_i a_i \langle x_i, x_s \rangle \\ \iff (K^2 a)_s &= \lambda (Ka)_s \end{aligned}$$

## Expressing the eigs of $C$ by those of $K$ (4)

Thus we obtain:

$$\begin{aligned} v = \sum_i a_i x_i \text{ eig of } C &\iff \forall s : (K^2 a)_s = \lambda(Ka)_s \\ &\iff K^2 a = \lambda K a \\ &\iff K a = \lambda a \text{ (as } K \text{ is positive definite)} \\ &\iff a \text{ is eigenvector of } K \text{ with eigenvalue } \lambda. \end{aligned}$$

# Expressing the eigs of $C$ by those of $K$ (5)

## Step 3: Normalization:

Let  $a \in \mathbb{R}^d$  be an eigenvector of  $K$  with  $\|a\| = 1$ . Then the length of the corresponding eigenvector  $v \in \mathbb{R}^d$  of  $C$  is given by

$$\begin{aligned}\|v\|^2 &= \left\| \sum_j a_j x_j \right\|^2 = \left\langle \sum_j a_j x_j, \sum_i a_i x_i \right\rangle = \sum_{i,j} a_i a_j \langle x_i, x_j \rangle \\ &= \sum_{i,j} a_i a_j k(x_i, x_j) = a' K a = a' \lambda a = \lambda\end{aligned}$$

To obtain a unit eigenvector  $v$ , we have to normalize the eigenvector  $a$  obtained from  $K$  with  $1/\sqrt{\lambda}$ .



# Exercise: eigs of $C$ and $K$

QUESTION:

- ▶  $C$  is a  $d \times d$ -matrix, so its eigendecomposition has  $d$  eigenvalues.
- ▶  $K$  is a  $n \times n$  matrix with  $n$  eigenvalues.
- ▶ But intuitively spoken, we just showed that we can convert the eigenvalues of  $K$  to those of  $C$  and vice versa.

HOW CAN THIS BE, IF THEIR NUMBERS ARE DIFFERENT???

# Expressing the projection on eigs of $C$ using $K$

- ▶ Assume we want to project on eigenvector  $v$  of  $C$ . Have already seen that we can write  $v = \sum_i a_i x_i$ . Thus:

$$\pi_v(x_i) = v'x_i = \left\langle \sum_j a_j x_j, x_i \right\rangle = \sum_j a_j \langle x_j, x_i \rangle$$

- ▶ If we want to project on a subspace spanned by  $\ell$  vectors  $v_1, \dots, v_\ell \in \mathbb{R}^d$ , compute each of the  $\ell$  coordinates by this formula as well.
- ▶ To compute the projections, we only need scalar products ☺

# Finally: kernel PCA

Input: Data points  $X_1, \dots, X_n$ , parameter  $\ell$

- ▶ Compute the kernel matrix  $K$
- ▶ Center the data in feature space by computing the centered kernel matrix  $\tilde{K} = K - \mathbb{1}_n K - K \mathbb{1}_n + \mathbb{1}_n K \mathbb{1}_n$
- ▶ Compute the eigendecomposition  $\tilde{K} = ADA'$ . Let  $A_k$  denote the  $k$ -th column of  $A$  and  $\lambda_k$  the corresponding eigenvalue.
- ▶ Define the matrix  $V_\ell$  which has the columns  $A_k / \sqrt{\lambda_k}$ ,  $k = 1, \dots, \ell$
- ▶ Compute the low dim representation points  $y_i = (y_i^1, \dots, y_i^\ell)'$  with the formula  $y_i^s = \sum_{j=1}^n v_j^s \tilde{K}_{ji}$  (for all  $s = 1, \dots, \ell$ ).

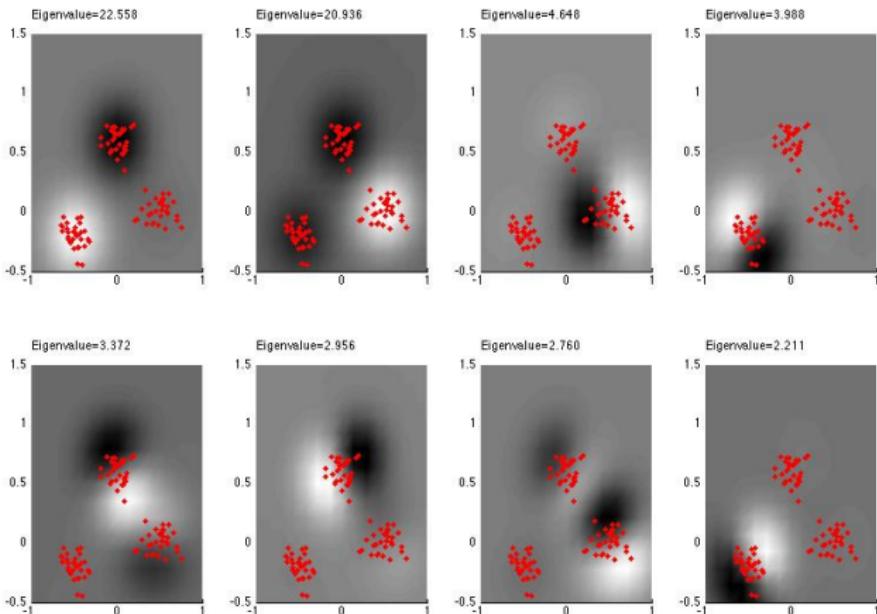
Output:  $y_1, \dots, y_n \in \mathbb{R}^\ell$

# kPCA toy example: three Gaussians

demo\_kpca\_bernhard.m

- ▶ Data drawn from 2-dim Gaussians (red crosses)
- ▶ kernel used is Gaussian kernel
- ▶ Now can compute the first eigenvectors in kernel feature space
- ▶ For test points, plot the coordinate which results when projecting on this eigenvector in the feature space (grey scale)

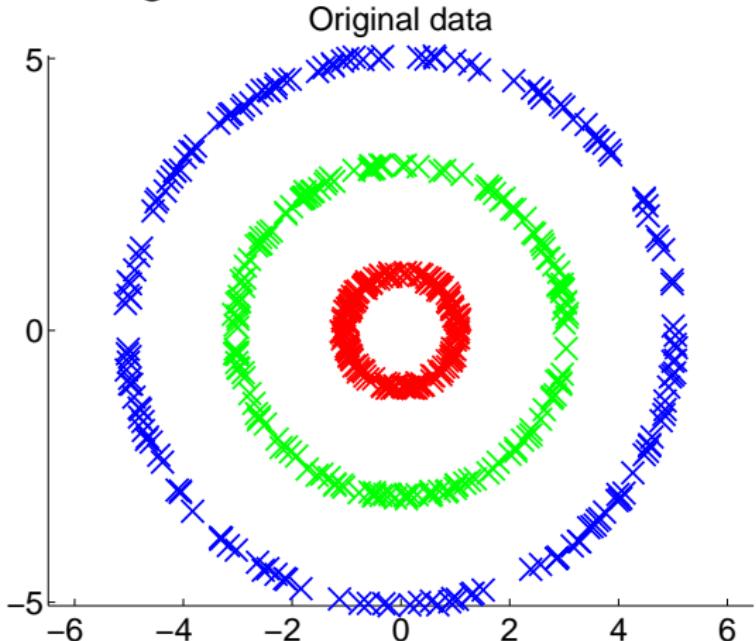
# kPCA toy example: three Gaussians (2)



# kPCA toy example: rings

demo\_kpca\_toy.m:

Consider the following three-dimensional data set:



## kPCA toy example: rings (2)

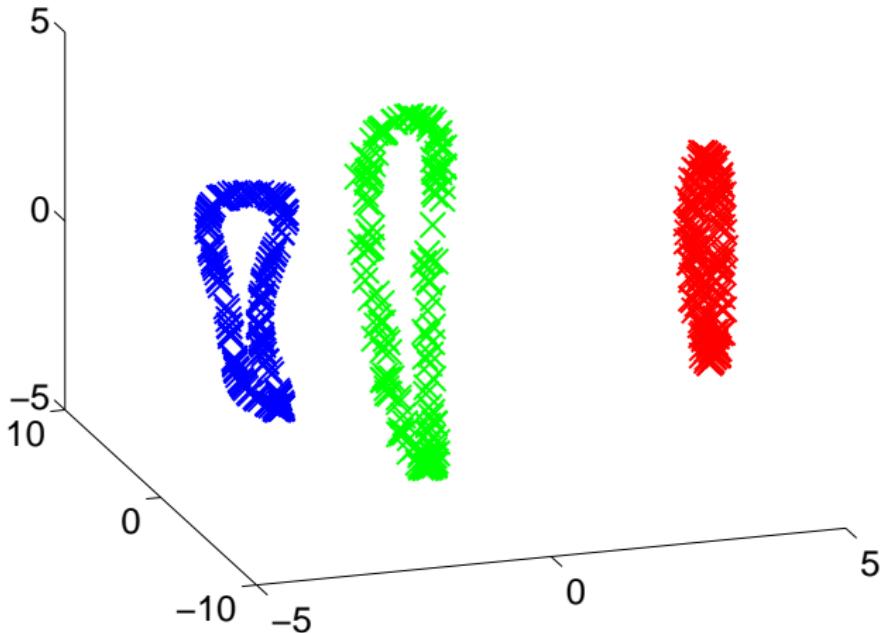
Now apply kPCA:

- ▶ Choose the Gaussian kernel with  $\sigma = 2$ .
- ▶ Note: we implicitly work in the RKHS, which has  $n$  dimensions
- ▶ So it makes sense to choose  $\ell = 3$  (even though the original data set just had  $d = 2$ ).

# kPCA toy example: rings (3)

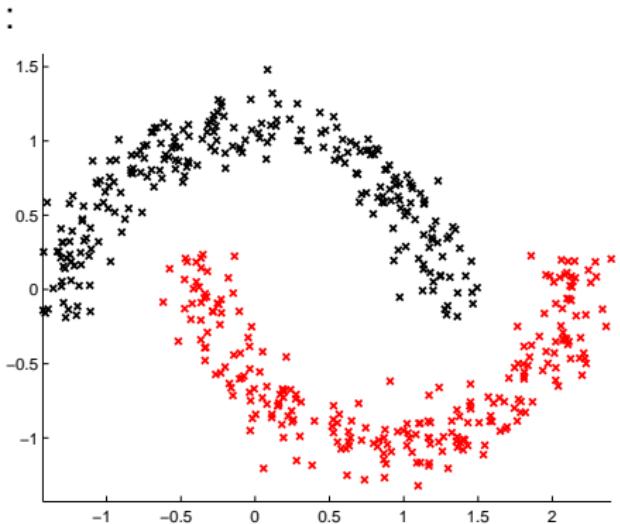
Here is the result:

3 dim kPCA

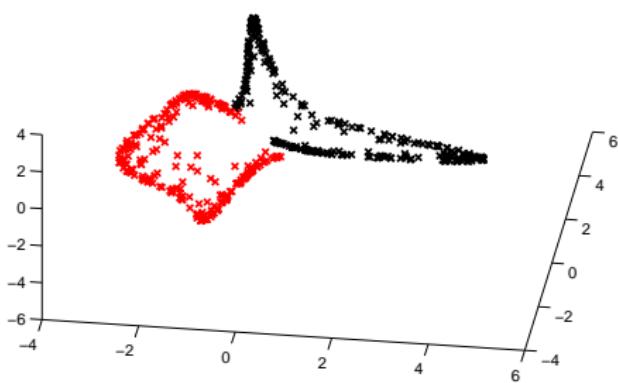


Surprising, isn't it???

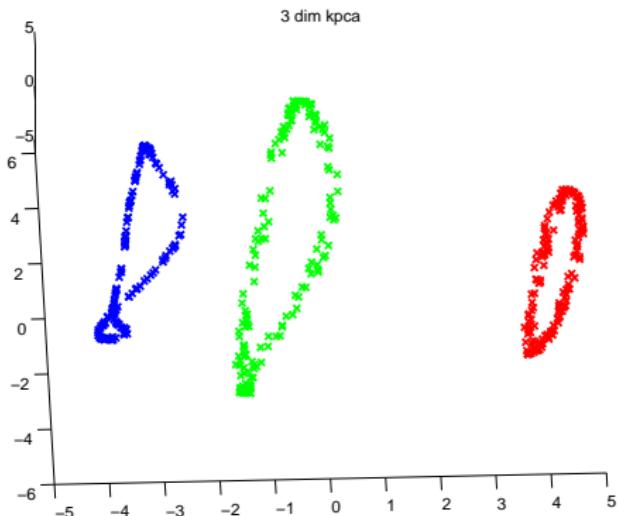
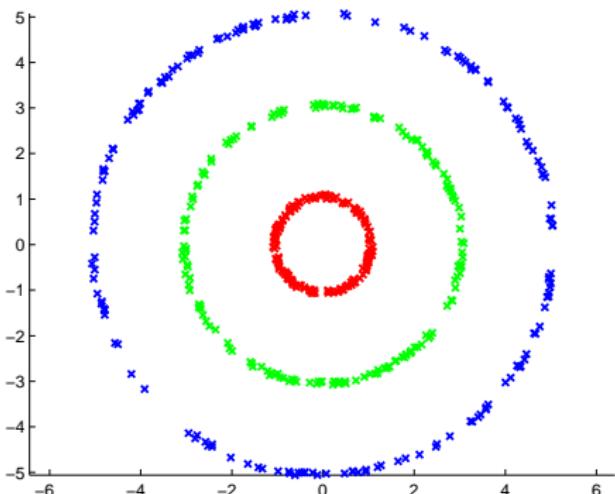
# More toy examples



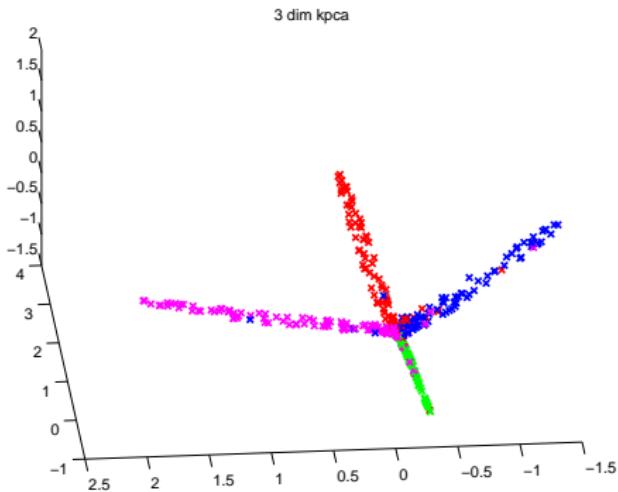
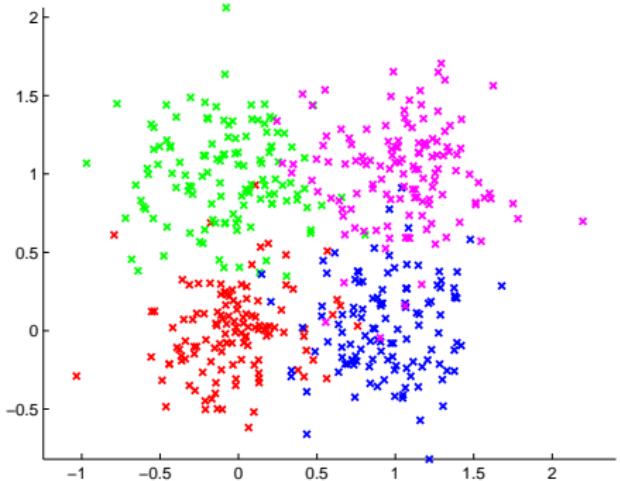
3 dim kpca



## More toy examples (2)



# More toy examples (3)



# History

- ▶ Classical PCA was invented by Pearson:  
*On Lines and Planes of Closest Fit to Systems of Points in Space. Philosophical Magazine, 1901.*
- ▶ It is one of the most popular “classical” techniques for data analysis.
- ▶ Kernel PCA was invented pretty much 100 years later ☺  
*B. Schölkopf, A. Smola, and K.-R. Müller. Kernel Principal component Analysis. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, Advances in Kernel Methods—Support Vector Learning, pages 327–352. MIT Press, Cambridge, MA, 1999.*

# Summary: PCA and kernel PCA

## Standard PCA:

- ▶ Technique to reduce the dimension of a data set in  $\mathbb{R}^d$  by linear projections.
- ▶ First explanation: throw away dimensions with “low variance”
- ▶ Second explanation: minimize the squared error.
- ▶ Can be computed by an eigendecomposition of the empirical covariance matrix.

## Kernel PCA:

- ▶ Use the kernel trick to make PCA non-linear.

# Multi-dimensional scaling

Literature:

Multi-dimensional scaling:

- ▶ Is a classic that is covered in many books on data analysis.
- ▶ A whole book on the subject: *Borg, Groenen: Modern multidimensional scaling. Springer, 2005.*

Isomap:

- ▶ The original paper is: *J. Tenenbaum, V. De Silva, J. Langford. A global geometric framework for nonlinear dimensionality reduction. Science, 2000.*

# Embedding problem

- ▶ Assume we are given a distance matrix  $D \in \mathbb{R}^{n \times n}$  that contains distances  $d_{ij} = \|x_i - x_j\|$  between data points.
- ▶ Can we “recover” the points  $(x_i)_{i=1,\dots,n} \in \mathbb{R}^d$ ?

This problem is called **(metric) multi-dimensional scaling**.

A more general way of asking: Given abstract “objects”  $x_1, \dots, x_n \in \mathcal{X}$ , can we find an **embedding**  $\Phi : \mathcal{X} \rightarrow \mathbb{R}^d$  (for some  $d$ ) such that  $\|\Phi(x_i) - \Phi(x_j)\| = d_{ij}$ ?

DO YOU BELIEVE IT ALWAYS WORKS?

# Embedding problem (2)

Answer will be:

- ▶ We can find a correct point configuration if the distances really come from points  $\in \mathbb{R}^d$ . In this case we say that  $D$  is a **Euclidean distance matrix**. See next slide for how this works.
- ▶ For general distance matrices  $D$ , we cannot achieve such an embedding without distorting the data. There is a huge bulk of literature on approximate embeddings, but we won't cover it in this lecture.

## Embedding problem (3)

WHY DO YOU THINK SUCH AN EMBEDDING MIGHT BE USEFUL?

## Embedding problem (4)

Why might we be interested in such an embedding?

- ▶ Visualization!
- ▶ Many algorithms are just defined for Euclidean data. If we want to apply them, we need to find a Euclidean representation of our data.
- ▶ Identify low-dimensional structure, see Isomap below.

What might be problematic about it?

- ▶ We might introduce distortion to the data ...

# MDS in various flavors

- ▶ **Classic MDS:** we assume that the given distance matrix is Euclidean.
  - ▶ If the matrix is Euclidean, embedding will be exact.
  - ▶ If the matrix is not Euclidean, embedding will make some errors.
- ▶ **Metric MDS:** we are given any distance matrix (might be non-Euclidean). We try to find an embedding that approximately preserves all distances.
  - ▶ In case the original matrix is non-Euclidean, perfect reconstruction is impossible, so we definitely will make approximation errors.
  - ▶ In case the original matrix is Euclidean, we might still make errors due to the formulation of the problem, see below.
- ▶ **Non-metric MDS:** we are not given distances, but ordinal information, see below.

## Classic MDS

Assume we are given a Euclidean distance matrix  $D$ . Will now see how to express the entries of the Gram matrix  $S = (\langle x_i, x_j \rangle)_{ij=1,\dots,n}$  in terms of entries of  $D$ :

- ▶ By definition:

$$\begin{aligned} d_{ij}^2 &= \|x_i - x_j\|^2 = \langle x_i - x_j, x_i - x_j \rangle \\ &= \langle x_i, x_i \rangle + \langle x_j, x_j \rangle - 2\langle x_i, x_j \rangle \end{aligned}$$

- ▶ Rearranging gives

$$\begin{aligned} \langle x_i, x_j \rangle &= \frac{1}{2} \left( \underbrace{\langle x_i, x_i \rangle}_{=d(0,x_i)^2} + \underbrace{\langle x_j, x_j \rangle}_{=d(0,x_j)^2} - d_{ij}^2 \right) \end{aligned}$$

## Classic MDS (2)

- We are free to choose the origin 0 as we want. For simplicity, we choose the first data point  $x_1$  as the origin. This gives:

$$\langle x_i, x_j \rangle = \frac{1}{2} \left( d_{1i}^2 + d_{1j}^2 - d_{ij}^2 \right)$$

- So we can express the entries of the Gram matrix  $S$  with  $s_{ij} = \langle x_i, x_j \rangle$  in terms of the given distance values.
- Because  $S$  it is positive definite, we can decompose  $S$  in the form  $S = XX'$  where  $X \in \mathbb{R}^{n \times d}$ .  
**EXERCISE: HOW EXACTLY DO YOU DO THIS? WHAT IS THE DIMENSION  $d$  GOING TO BE?**
- The rows of  $X$  are what we are looking for, that is we set the embedding of point  $x_i$  as the  $i$ -th row of the matrix  $X$ .

# Classic MDS implementation

This is how it finally works:

- ▶ Compute the matrix  $S$  with  $s_{ij} = \frac{1}{2} \left( d_{1i}^2 + d_{1j}^2 - d_{ij}^2 \right)$ .
- ▶ Compute the eigenvalue decomposition  $S = V\Lambda V'$ .
- ▶ Define  $X = V\sqrt{\Lambda}$ .

Alternatively, if you want to fix some dimension  $d \leq n$ , set  $V_d$  to be the first  $d$  columns of  $V$  and  $\Lambda_d$  the  $d \times d$  diagonal matrix with the first  $d$  eigenvalues on the diagonal, and then set  $X = V_d\sqrt{\Lambda_d}$ .

- ▶ Row  $i$  of  $X$  then gives the coordinates of the embedded point  $x_i$ .

# Classic MDS implementation (2)

## How to choose $d$ ?

- ▶ If the data points come from  $\mathbb{R}^d$ , then the matrix  $S$  is going to have rank  $d$ , that is there are  $d$  eigenvalues  $> 0$  and  $n - d$  eigenvalues equal to 0.
- ▶ Hence, looking at the spectrum of  $S$  gives you an idea to choose  $d$ . In case of classic MDS, you can just read off  $d$  from the matrix, in the more general case of metric MDS you simply “choose it reasonable” (as in PCA).

# Demos

demo\_mds.m

# Metric MDS

Metric MDS refers to the problem where the distance matrix  $D$  is no longer Euclidean, but we still believe (hope) that a good embedding exists.

- ▶ If the distance matrix  $D$  is not Euclidean, we will not be able to recover an exact embedding.
- ▶ Instead, one defines a “stress function”. Below is an example for such a stress function:

$$\text{stress}(\text{embedding}) = \frac{\sum_{ij} (\|x_i - x_j\| - d_{ij})^2}{\sum_{ij} \|x_i - x_j\|}$$

Many more stress functions are considered in the literature.

- ▶ Then we try to find an embedding  $x_1, \dots, x_n$  with small stress by a standard non-convex optimization algorithm, say gradient descent.

## Metric MDS (2)

When using metric MDS, there are two sources of error:

- ▶ The distance matrix is not Euclidean, so we will not be able to recover a perfect embedding.
- ▶ The optimization problems are highly non-convex and suffer all kinds of problems of local optima.

Using metric MDS only makes sense if the data is “nearly Euclidean”, and results should always be treated with care.

# Non-metric MDS

- ▶ Instead of distance values, we are just given distance comparisons, that is we know whether  $d_{ij} < d_{ik}$  or vice versa.
- ▶ The task is then to find an embedding such that these ordinal relationships are preserved.
- ▶ Our group is working on this problem ☺



**Which of the bottom images is most similar to the top image?**

# History of MDS

- ▶ Metric MDS: Torgerson (1952) - The first well-known MDS proposal. Fits the Euclidean model.
- ▶ Non-metric MDS: Shepard (1962) and Kruskal (1964)

# Outlook: general embedding problems

There exists a huge literature on embedding metric spaces in Euclidean spaces:

- ▶ Given certain assumptions on the metric ...
  - ▶ In what space can I embed (dimension???)
  - ▶ What are the guarantees on the distortion?

Some literature pointers:

- ▶ Theorem of Bourgain: Any  $n$ -point metric space can be embedded in Euclidean space with distortion  $O(\log n)$ . By the theorem of Johnson-Lindenstrauss, we can achieve dimensionality of  $O(\log n)$  as well.
- ▶ An overview paper on the area is: *Piotr Indyk and Jiri Matousek. Low-distortion embeddings of finite metric spaces. In: Handbook of Discrete and Computational Geometry, 2004.*

# Summary MDS

- ▶ Given a distance matrix  $D$ , MDS tries to construct an embedding of the data points in  $\mathbb{R}^d$  such that the distances are preserved as well as possible.
- ▶ If  $D$  is Euclidean, a perfect embedding can easily be constructed.
- ▶ If  $D$  is not Euclidean, MDS tries to find an embedding that minimizes the “stess”. The resulting problem is highly non-convex. Solutions should be treated with care.

# Graph-based machine learning algorithms: introduction

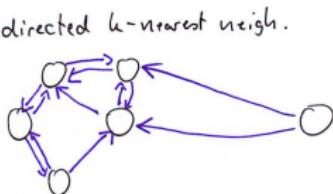
# Neighborhood graphs

Given the similarity or distance scores between our objects, we want to build a graph based on it.

- ▶ Vertices = objects
- ▶ Edges between objects in the same “neighborhood”

Different variants:

- ▶ **directed  $k$ -nearest neighbor graph:** connect  $x_i$  by a directed edge to its  $k$  nearest neighbors (or to the  $k$  points with the largest similarity) .

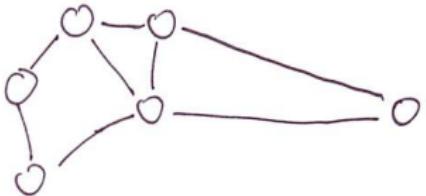


## Neighborhood graphs (2)

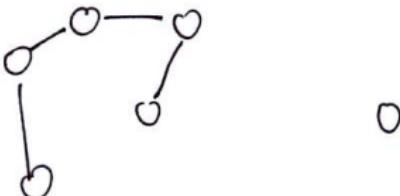
Note that this graph is not symmetric. Many algorithms need undirected graphs (in particular, spectral methods). To make it undirected:

- ▶ **Standard  $k$ -nearest neighbor graph:** put an edge between  $x_i$  and  $x_j$  if  $x_i$  is among the  $k$  nearest neighbors of  $x_j$  OR vice versa.
- ▶ **Mutual  $k$ -nearest neighbor graph:** put an edge between  $x_i$  and  $x_j$  if  $x_i$  is among the  $k$  nearest neighbors of  $x_j$  AND vice versa.

standard undirected



mutual undirected



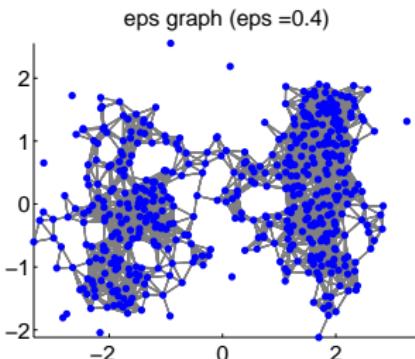
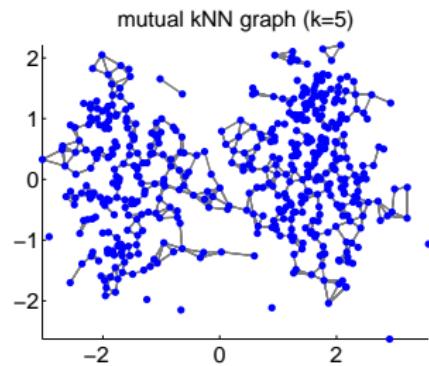
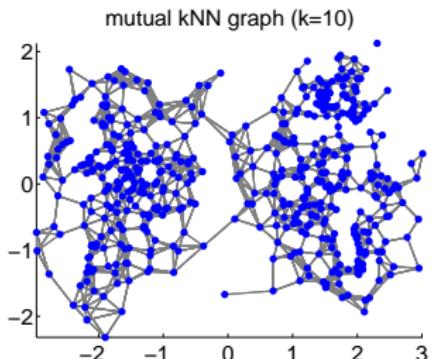
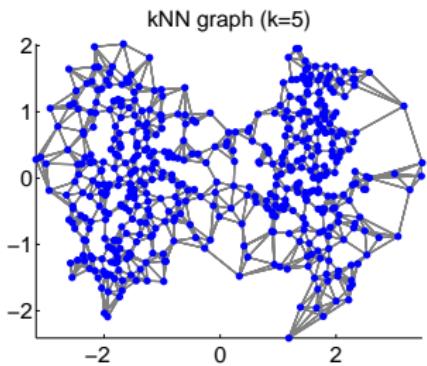
## Neighborhood graphs (3)

Alternatively, we can use the  $\varepsilon$ -graph:

- ▶ Connect each point to all other points that have distance smaller than  $\varepsilon$  (or similarity larger than some threshold  $e$ )

Note: all these neighborhood graphs can be built based on similarities or based on distances.

# Neighborhood graphs (4)



# Neighborhood graphs (5)

Edge weights:

- ▶ A priori, all the graphs above are unweighted.
- ▶ On kNN graphs, it often makes sense to use **similarities as edge weights**.  
(Reason: edges have very diverse “lengths”, and we want to tell this to the algorithm; e.g., spectral clustering is allowed to cut long edges more easily than short edges)
- ▶ **Never use distances as weights!** (this destroys the “logic” behind a neighborhood graph: no edge means “far away”, and no edge is the same as edge weight 0 ... )
- ▶ For  $\varepsilon$ -graphs, edge weights do not make so much sense because all edges are more or less “equally long”. The edge weights then do not carry much extra information.

# Neighborhood graphs (6)

Why are we interested in similarity graphs?

- ▶ Sparse representation of the similarity structure
- ▶ Graphs are well-known objects, lots of algorithms to deal with them.
- ▶ Similarity graph encodes local structure, goal of machine learning (unsupervised learning) is to make statements about its global structure.
- ▶ There exist many algorithms for machine learning on graphs:
  - ▶ Clustering: Spectral clustering (see later)
  - ▶ Dimensionality reduction: Isomap, Laplacian eigenmaps, Maximum Variance Unfolding
  - ▶ Semi-supervised learning: label propagation (not treated in the lecture)

# Isomap

## Literature:

- ▶ Original paper:

*J. Tenenbaum, V. De Silva, J. Langford. A global geometric framework for nonlinear dimensionality reduction. Science, 2000.*

# Isomap

We often think that data is “inherently low-dimensional”:

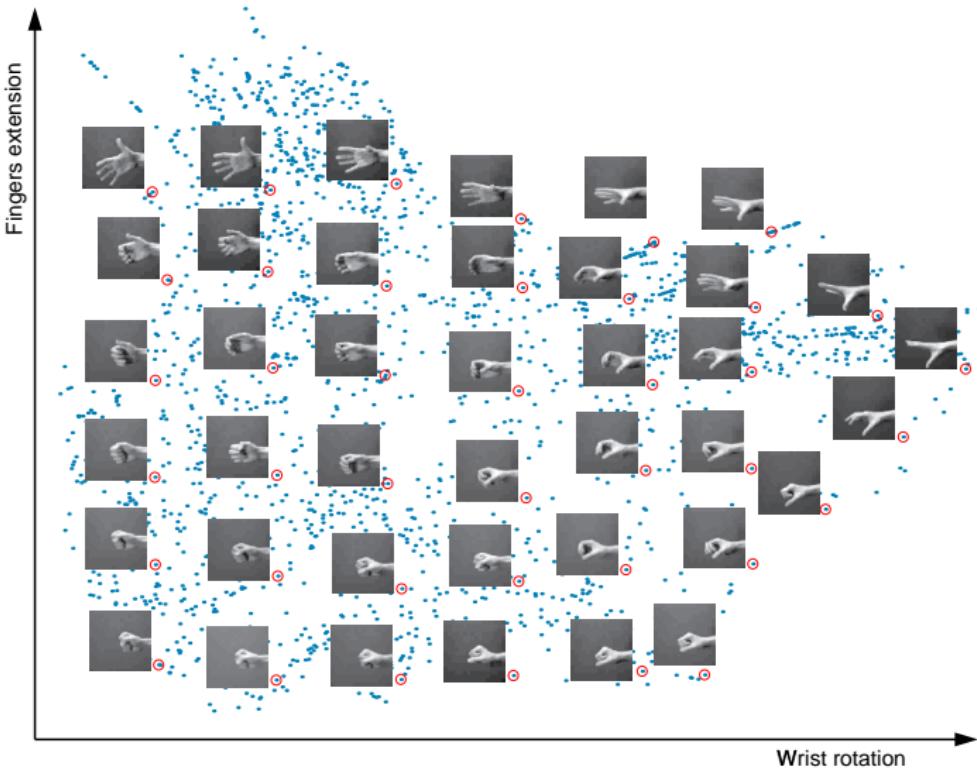
- ▶ Images of a tea pot, taken from all angles. Even though the images live in  $\mathbb{R}^{256}$ , say, we believe they sit on a manifold corresponding to a circle:



## Isomap (2)

- ▶ A phenomenon generates very high-dimensional data, but the “effective number of parameters” is very low

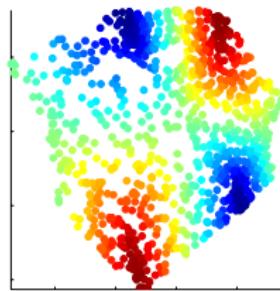
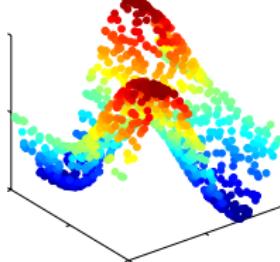
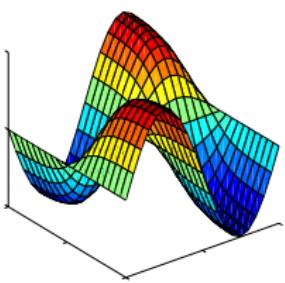
# Isomap (3)



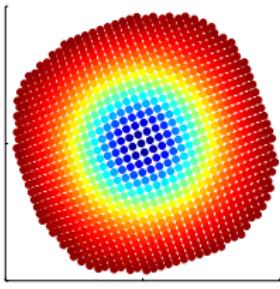
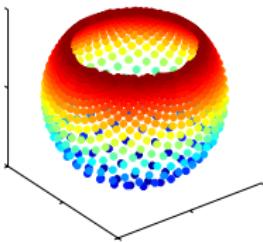
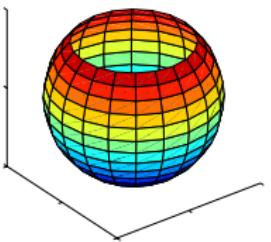
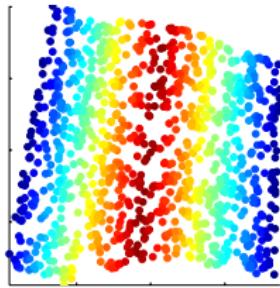
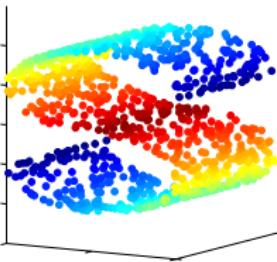
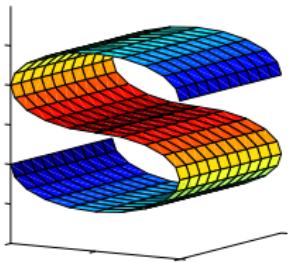
# Isomap (4)

More abstractly:

- ▶ We assume that the data lives in a high-dimensional space, but effectively just sits on a low-dimensional manifold
- ▶ We would like to find a mapping that recovers this manifold.
- ▶ If we could do this, then we could reduce the dimensionality in a very meaningful way.



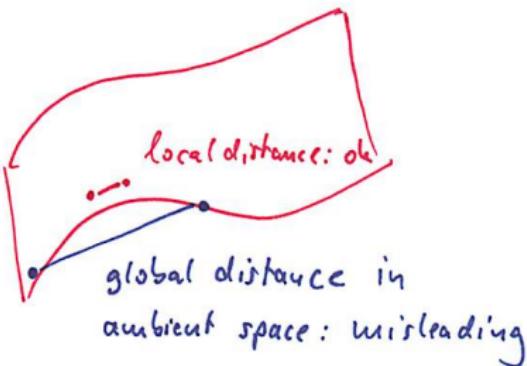
# Isomap (5)



# The Isomap algorithm

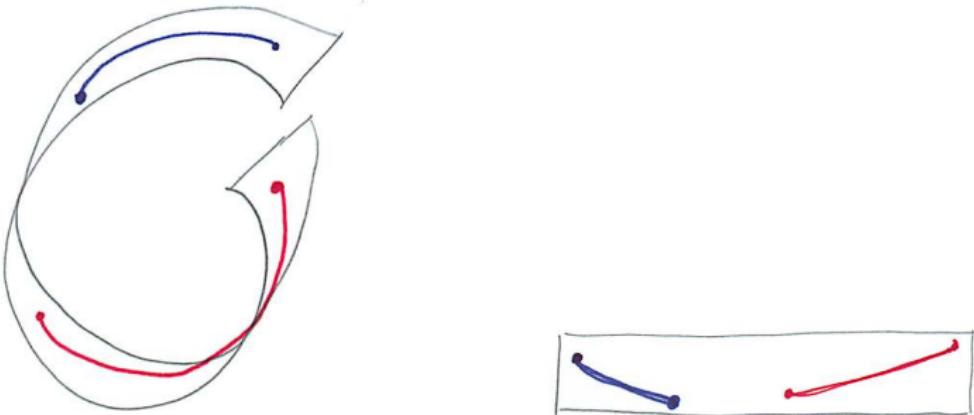
Intuition:

- ▶ In a small local region, Euclidean (extrinsic) distances between points on a manifold approximately coincide with the intrinsic distances. We want to keep the local distances unchanged.
- ▶ This is no longer the case for large distances: we want to keep the intrinsic (geodesic) distances rather than the ones in the ambient space.



## The Isomap algorithm (2)

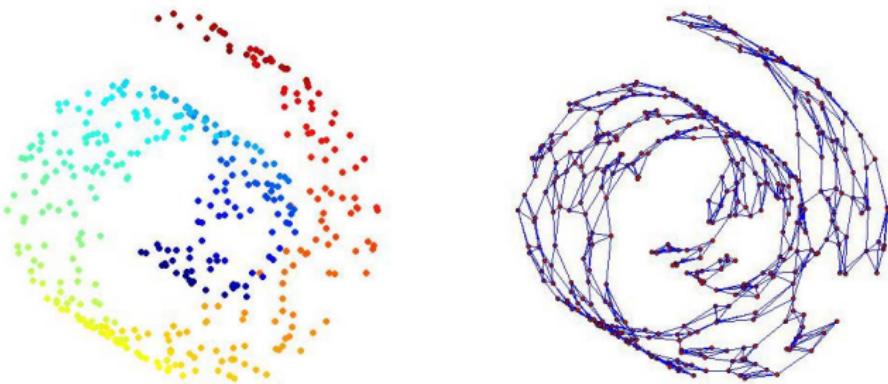
- If we want to “straighten” a manifold, we need to embed it in such a way that the Euclidean distance after embedding corresponds to the geodesic distance on the manifold.



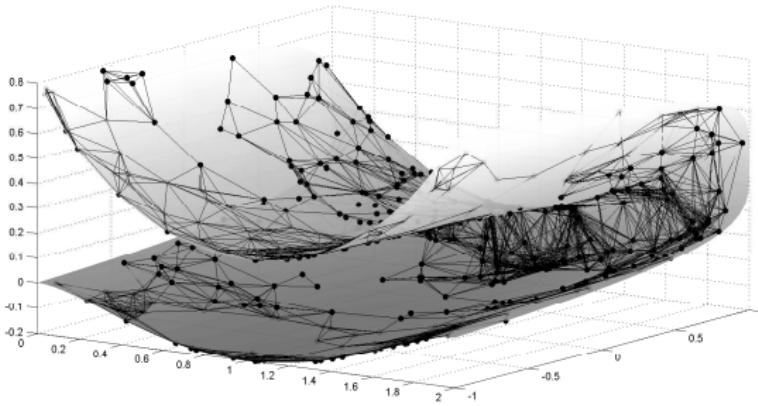
So we would like to discover the geodesic distances in the manifold.

# The Isomap algorithm (3)

- ▶ To discover the geodesic distances in the manifold:
  - ▶ Build a kNN graph on the data
  - ▶ Use the shortest path distance in this graph.
  - ▶ Idea is: it goes “along” the manifold.



# The Isomap algorithm (4)



(figure by Matthias Hein)

# The Isomap algorithm (5)

The algorithm:

- ▶ Given some abstract data points  $X_1, \dots, X_n$  and a distance function  $d(x_i, x_j)$ .
- ▶ Build a  $k$ -nearest neighbor graph where the edges are weighted by the distances. **These are the local distances.**
- ▶ In the kNN graph, compute the shortest path distances  $d_{sp}$  between all pairs of points and write them in the matrix  $D$ .  
**They correspond to the geodesic distances.**
- ▶ Then apply metric MDS with  $D$  as input. **Finds embedding that preserves the geodesic distances.**

## Theoretical guarantees

In the original paper (supplement) the authors have proved:

- ▶ If the data points  $X_1, \dots, X_n$  are sampled uniformly from a “nice” manifold, then as  $n \rightarrow \infty$  and  $k \approx \log n$ , the shortest path distances in the kNN graph approximate the geodesic distances on the manifold.
- ▶ Under some geometric assumptions on the manifold, MDS then recovers an embedding with distortion converging to 0.

(Attention, the dimension is an issue here. Typically, we cannot embed a manifold without distortion in a space of the intrinsic dimension, we need to choose the dimension larger).

# Demos

- ▶ `demo_isomap.m`
- ▶ Toolbox by Laurens van der Maaten:
  - ▶ `addpath(genpath('/Users/ule/matlab_ule/downloaded_packages_not_in_path/dim_reduction_toolbox/'));` then call `drgui`
  - ▶ Use swiss roll and helix, play with  $k$

# History

- ▶ Manifold methods became fashionable in machine learning in the early 2000s.
- ▶ Isomap was invented in 2000.
- ▶ Since then, a large number of manifold-based dimensionality reduction techniques has been invented:  
Locally linear embedding, Laplacian eigenmaps, Hessian eigenmaps, Diffusion maps, Maximum Variance Unfolding, and many more ...
- ▶ There exists a nice matlab toolbox that implements all these algorithms, written by Laurens van der Maaten.  
[http://homepage.tudelft.nl/19j49/Matlab\\_Toolbox\\_for\\_Dimensionality\\_Reduction.html](http://homepage.tudelft.nl/19j49/Matlab_Toolbox_for_Dimensionality_Reduction.html)

To call the demo:

- ▶ `addpath(genpath('/Users/ule/matlab_ule/downloaded_packages_not_in_path/dim_reduction_toolbox/'))`
- ▶ `call drgui`

# Summary Isomap

- ▶ Unsupervised learning technique to extract the manifold structure from distance / similarity data
- ▶ Intuition: local distances define the intrinsic geometry, shortest paths in a kNN graphs correspond to geodesics.
- ▶ MDS then tries to find an appropriate embedding.

(\*) Maximum Variance Unfolding: SKIPPED

(\*) Johnson-Lindenstrauß: SKIPPED

(\*) Ordinal embedding (GNMDS, SOE, t-STE):  
SKIPPED

# Clustering

# Data clustering

Data clustering is one of the most important problems of unsupervised learning.

- ▶ Given just input data  $X_1, \dots, X_n$
- ▶ We want to discover groups (“clusters”) in the data such that points in the same cluster are “similar” to each other and points in different clusters are “dissimilar” of each other.
- ▶ Important: a priori, we don’t have any information (training labels) about these groups, and often we don’t know how many groups there are (if any).

# Data clustering (2)

## Applications:

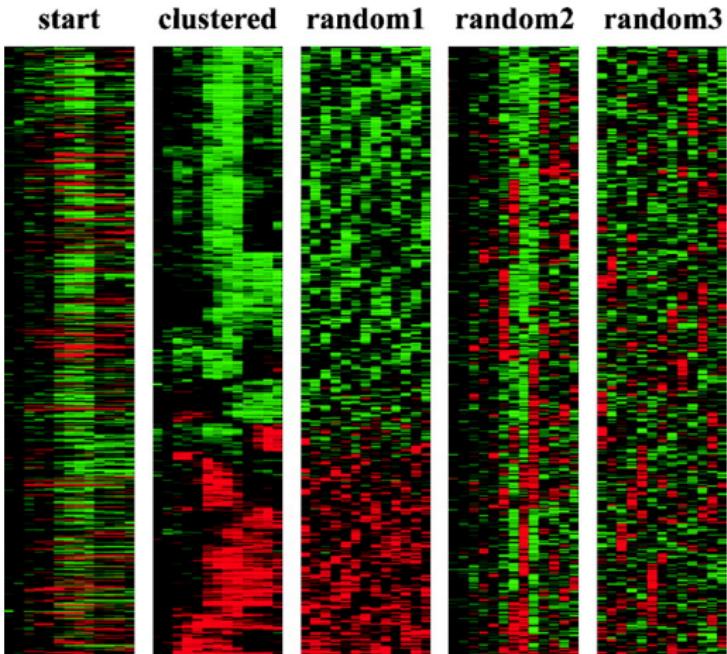
- ▶ Find “genres” of songs
- ▶ Find different “groups” of customers
- ▶ Find two different types of cancer, based on gene expression data
- ▶ Discover proteins that have a similar function
- ▶ ...

# Data clustering (3)

Two main reasons to do this:

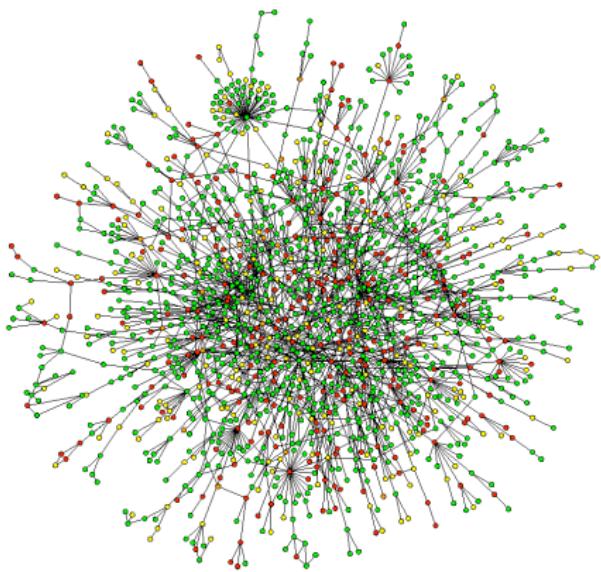
- ▶ Improve your understanding of the data! Exploratory data analysis.
- ▶ Reduce the complexity of the data. Vector quantization.  
For example, instead of training on a set of  $10^6$  customers, use 1000 “representative” customers.
- ▶ Break your problem into subproblems and treat each cluster individually.

# Example: Clustering gene expression data



M. Eisen et al., PNAS, 1998

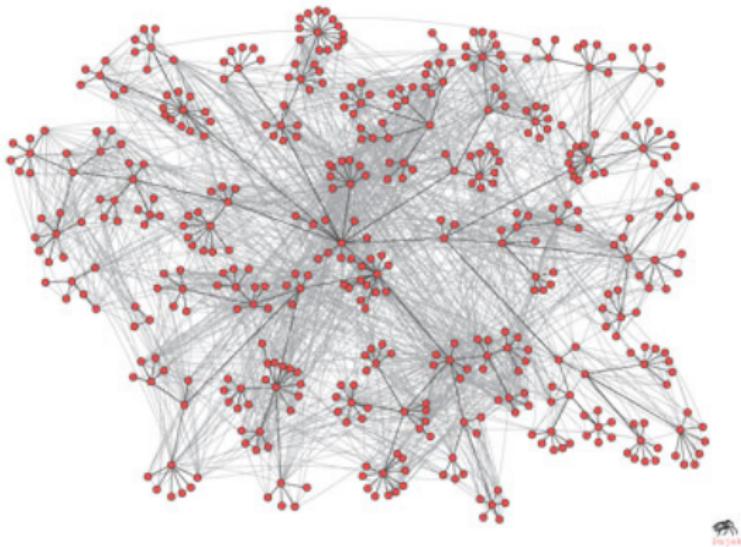
# Example: Protein interaction networks



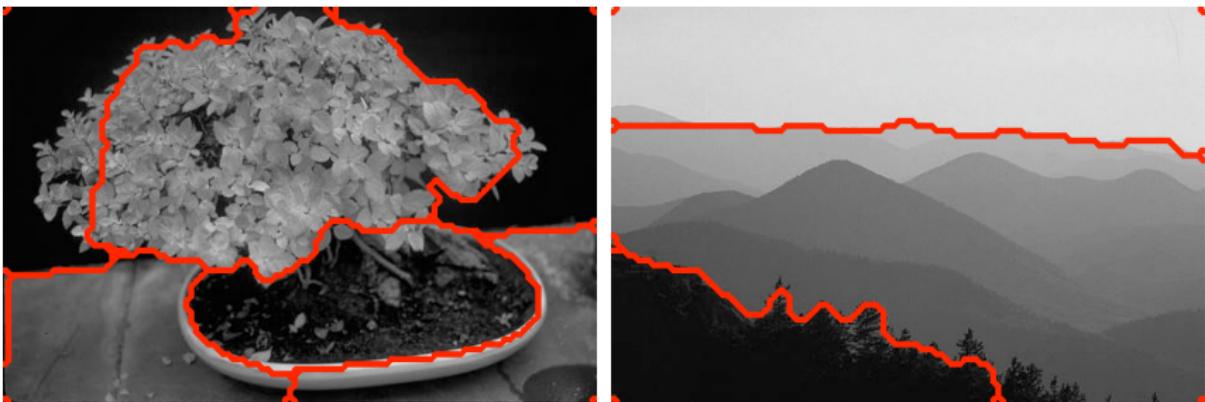
(from <http://www.math.cornell.edu/~durrett/RGD/RGD.html>)

# Example: Social networks

Corporate email communication (Adamic and Adar, 2005)

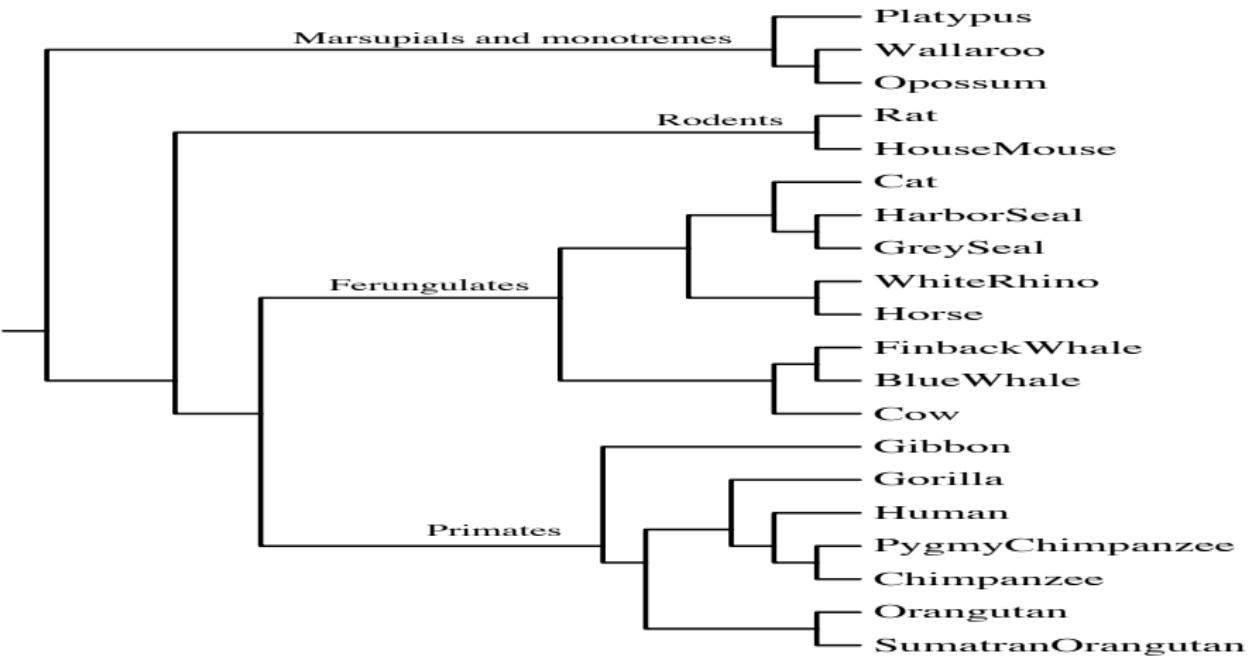


## Example: Image segmentation



(from Zelnik-Manor/Perona, 2005)

# Example: Genetic distances between mammals



cf. Chen/Li/Ma/Vitanyi (2004)

# Most common approach

Have two goals:

- ▶ Want distances between points in the same cluster to be small
- ▶ Want distances between points in different clusters be large

Naive approach:

- ▶ Define a criterion that measures these distances and try to find the best partition with respect to this criterion: Example:

$$\text{minimize} \frac{\text{average within-cluster distances}}{\text{average between-cluster distances}}$$

Problem:

- ▶ Which objective to choose?
- ▶ Most such optimization problems are NP hard (combinatorial optimization).

# K-means and kernel k-means

Literature:

Tibshirani/Hastie/Friedmann

# Standard $k$ -means algorithm

## *k*-means objective

- ▶ Assume we are given data points  $X_1, \dots, X_n \in \mathbb{R}^d$
- ▶ Assume we want to separate it into  $K$  groups.
- ▶ We want to construct  $K$  class representatives (class means)  $m_1, \dots, m_K$  that represent the groups.
- ▶ Consider the following objective function:

$$\min_{\{m_1, \dots, m_K \in \mathbb{R}^d\}} \sum_{k=1}^K \sum_{i \in C_k} \|X_i - m_k\|^2$$

That is, we want to find the centers such that the sum of squared distances of data points to the closest centers are minimized.

## $k$ -means objective (2)



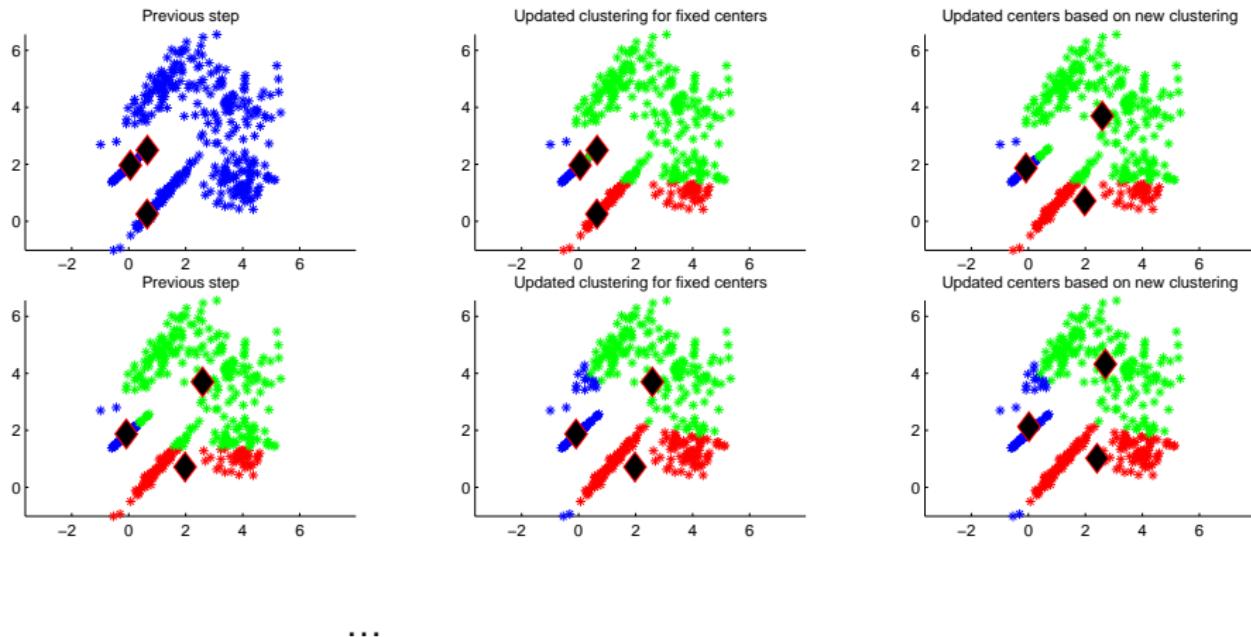
$x$  = cluster centers

# Lloyd's algorithm ( $k$ -means algorithm)

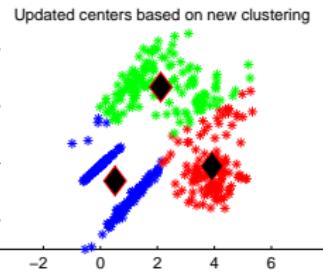
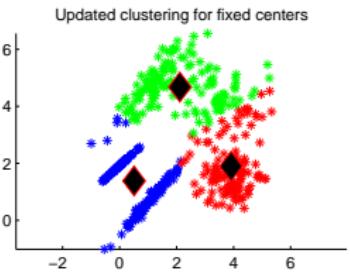
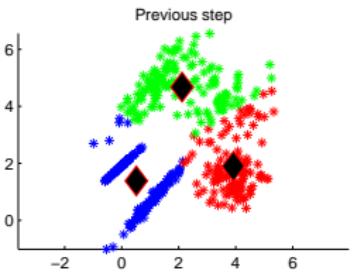
The following heuristic is typically used to find a local optimum of the  $k$ -means objective function:

- ▶ Start with randomly chosen centers.
- ▶ Repeat the following two steps until convergence:
  - ▶ Assign all points to the closest cluster center.
  - ▶ Define the new centers as the mean vectors of the current clusters.

# Lloyd's algorithm ( $k$ -means algorithm) (2)



# Lloyd's algorithm ( $k$ -means algorithm) (3)



# Lloyd's algorithm ( $k$ -means algorithm) (4)

## The formal $k$ -means algorithm:

- 1 **Input:** Data points  $X_1, \dots, X_n \in \mathbb{R}^d$ , number  $K$  of clusters to construct.
- 2 Randomly initialize the centers  $m_1^{(0)}, \dots, m_K^{(0)}$ .
- 3 **while** not converged
- 4 Assign each data point to the closest cluster center, that is define the clusters  $C_1^{(i+1)}, \dots, C_K^{(i+1)}$  by

$$X_s \in C_k^{(i+1)} \iff \|X_s - m_k^{(i)}\|^2 \leq \|X_s - m_l^{(i)}\|^2, l = 1, \dots, K$$

- 5 Compute the new cluster centers by

$$m_k^{(i+1)} = \frac{1}{|C_k^{(i+1)}|} \sum_{s \in C_k^{(i+1)}} X_s$$

- 6 **Output:** Clusters  $C_1, \dots, C_K$

# Lloyd's algorithm ( $k$ -means algorithm) (5)

matlab demo: `demo_kmeans()`

# *K*-means algorithm — Termination

## Proposition 17 (Termination)

Given a finite set of  $n$  points in  $\mathbb{R}^d$ . Then the  $k$ -means algorithm terminates after a finite number of iterations.

### Proof sketch.

- ▶ In each iteration of the while loop, the objective function decreases.
- ▶ There are only finitely many partitions we can inspect.
- ▶ So the algorithm has to terminate.



# $K$ -means algorithm — Solutions can be arbitrarily bad

## Proposition 18 (Bad solution possible)

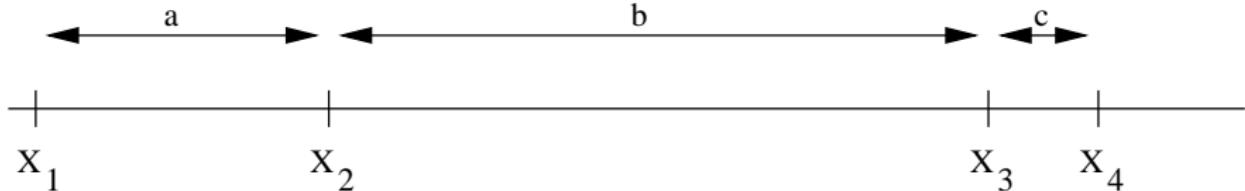
The algorithm ends in a local optimum which can be an arbitrary factor away from the global solution.

### Proof.

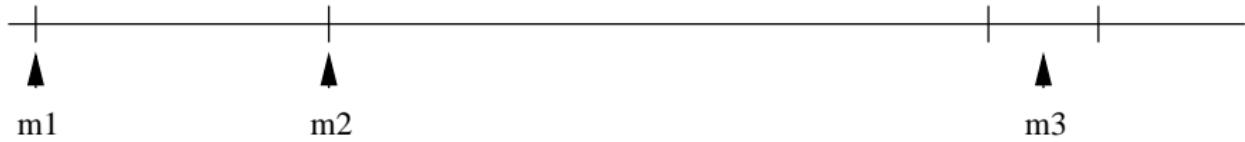
- ▶ We give an example with four points in  $\mathbb{R}$ , see figure on the next slides.
- ▶ By adjusting the parameters  $a$  and  $b$  and  $c$  we can achieve an arbitrarily bad ratio of global and local solution.

# $K$ -means algorithm — Solutions can be arbitrarily bad (2)

Data set: four points on the real line:



Optimal solution: (initialization:  $X_1, X_2, X_3$ ; value of solution:  $c^2 / 2$ )



Bad solution: (initialization:  $X_1, X_3, X_4$ ; value of solution:  $a^2 / 2$ )



# K-means algorithm — Initialization

Methods to select initial centers:

- ▶ Most common: randomly choose some data points as starting centers.
- ▶ Much better: Farthest first heuristic:

- 1  $S = \emptyset$  #  $S$  set of centers
- 2 Pick  $x$  uniformly at random from the data points  
 $S = \{x\}$
- 3 **while**  $|S| < k$
- 4   **for all**  $x \in \mathcal{X} \setminus S$
- 5     Compute  $D(x) := \min_{s \in S} \|x - s\|^2$ .
- 6     Select the next center  $y$  with probability proportional to  $D(x)$  among the remaining data points.

## *K*-means algorithm — Initialization (2)

The  $k$ -means algorithm with this heuristic is called kmeans++ and satisfies nice approximation guarantees.

- ▶ Initialize the centers using the solution of an even simpler clustering algorithm.
- ▶ Ideally have prior knowledge, for example that certain points are in different clusters.

# *K*-means algorithm — Heuristics for practice

As it is the standard procedure for highly non-convex optimization problems, in practice **we restart the algorithm many times with different initializations**. Then we use the best of all these runs as our final result.

# *K*-means algorithm — Heuristics for practice (2)

Common problem:

- ▶ In the course of the algorithm it can happen that a center “loses” all its data points (no point is assigned to the center any more).
- ▶ In this case, one either restarts the whole algorithm, or randomly replaces the empty center by one of the data points.

## *K*-means algorithm — Heuristics for practice (3)

Local search heuristics to improve the result once the algorithm has terminated:

- ▶ Restart many times with different initializations.
- ▶ Swap individual points between clusters.
- ▶ Remove a cluster center, and introduce a completely new center instead.
- ▶ Merge clusters, and additionally introduce a completely new cluster center.
- ▶ Split a cluster in two pieces (preferably, one which has a very bad objective function). Then reduce the number of clusters again, for example by randomly removing one.

# $k$ -means minimizes within-cluster distances

Another way to understand the  $k$ -means objective:

Proposition 19 ( $k$ -means and within-cluster distances)

The following two optimization problems are equivalent:

1. Find a discrete partition of the data set such that the within-cluster-distances are minimized:

$$\min_{\{C_1, \dots, C_K\}} \sum_{k=1}^K \frac{1}{|C_k|^2} \sum_{i \in C_k, j \in C_k} \|X_i - X_j\|^2$$

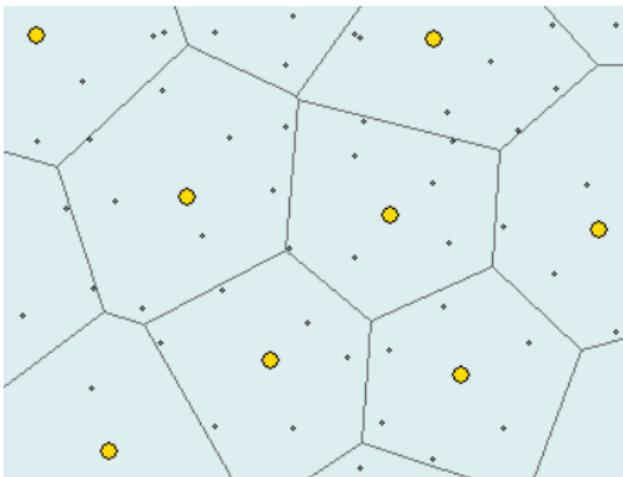
2. Find cluster centers such that the distances of the data points to these centers are minimized:

$$\min_{m_1, \dots, m_K \in \mathbb{R}^d} \sum_{k=1}^K \sum_{i \in C_k} \|X_i - m_k\|^2$$

**Proof.** Elementary, but a bit lengthy, we skip it.

# $k$ -means leads to Voronoi partitions

Observe that the partition induced by the  $k$ -means objective corresponds to a Voronoi partition of the space:

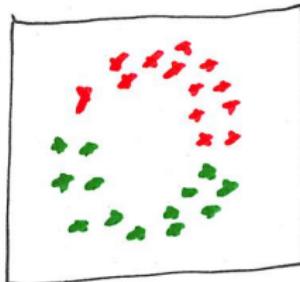
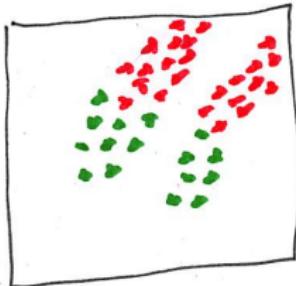
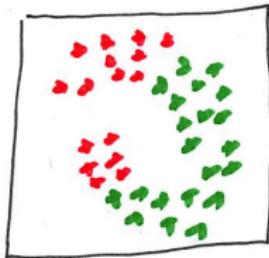
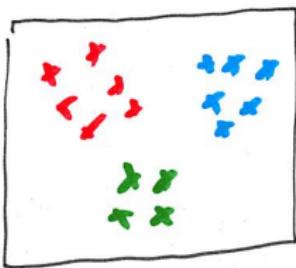


## *k*-means leads to Voronoi partitions (2)

This has two important consequences:

- ▶ all cluster boundaries are linear (WHY MIGHT THIS BE INTERESTING?)
- ▶ The *k*-means algorithm always constructs convex clusters! This gives intuition about when it works and when it doesn't work:

## *k*-means leads to Voronoi partitions (3)



## *K*-means — computational complexity

- ▶ Finding the global solution of the  $k$ -means optimization problem is **NP hard** (both if  $k$  is fixed or variable, and both if the dimension is fixed or variable).

This is curious because one can prove that there only exist polynomially many Voronoi partitions of any given data set. The difficulty is that we cannot construct any enumeration to search through them.

See the following paper and references therein:

*Mahajan, Meena and Nimbhorkar, Prajakta and Varadarajan, Kasturi: The planar  $k$ -means problem is NP-hard. WALCOM: Algorithms and Computation, 2009.*

## *K*-means — computational complexity (2)

- ▶ On the other hand, optimizing the  $k$ -means objective has **polynomial smoothed complexity**.

*Arthur, David and Manthey, Bodo and Röglin: k-Means has polynomial smoothed complexity. FOCS 2009.*

- ▶ With careful seeding, one can achieve **constant-factor approximations**:
  - ▶ Consider the random farthest first rule for initialization (`kmeans` with this initialization is called `kmeans++`).
  - ▶ Then, the expected objective value is at most a factor  $O(\log k)$  worse than the optimal solution.
  - ▶ Reference:  
*Arthur, D., Vassilvitskii, S.: k-means++: the advantages of careful seeding. In: Symposium on Discrete Algorithms (SODA), 2007.*

## More variants of $K$ -means

- ▶  $K$ -median: here the centers are always data points. Can be used if we only have distances, but no coordinates of data points.
- ▶ weighted  $K$ -means: introduce weights for the individual data points
- ▶ kernel- $K$ -means: the kernelized version of  $K$ -means (note that all boundaries between clusters are linear)
- ▶ soft  $K$ -means: no hard assignments, but “soft” assignments (often interpreted as “probability” of belonging to a certain cluster)
- ▶ Note:  $K$ -means is a simplified version of the EM-algorithm which fits a Gaussian mixture model to the data.

## Kernel $k$ -means

Literature: Shawe-Taylor/Cristianini Section 8.2.2

## Idea

You guessed it already: it is possible to kernelize the  $k$ -means algorithm.

- ▶ Sets are separated by hyperplanes, so this looks like a promising candidate for kernelization.
- ▶ Kernel  $k$ -means would hopefully be able to generate more general cluster shapes than just convex clusters.

However, we are going to skip it here. Kernel  $k$ -means is closely related to the next algorithm we are going to study: spectral clustering. If you are interested in details, consider the following paper:

*I. S. Dhillon, Y. Guan, and B. Kulis, Kernel  $k$ -means, spectral clustering and normalized cuts. KDD, 2004.*

# History of $k$ -means

- ▶ The algorithm was developed in the 1960s at Bells Lab by S. Lloyd (but only got published much later):  
*S. Lloyd, Least square quantization in PCM, IEEE Trans. Infor. Theory, 1982.*
- ▶ There is still active research about heuristics and theoretical guarantees of the algorithm, see for example the references on the “computational complexity” slide.
- ▶ The  $k$ -means algorithm is the most popular traditional clustering algorithm.
- ▶ Kernel  $k$ -means: around 2004. See e.g.  
*Dhillon, I. and Guan, Y. and Kulis, B., Kernel  $k$ -means, spectral clustering and normalized cuts. KDD, 2004.*

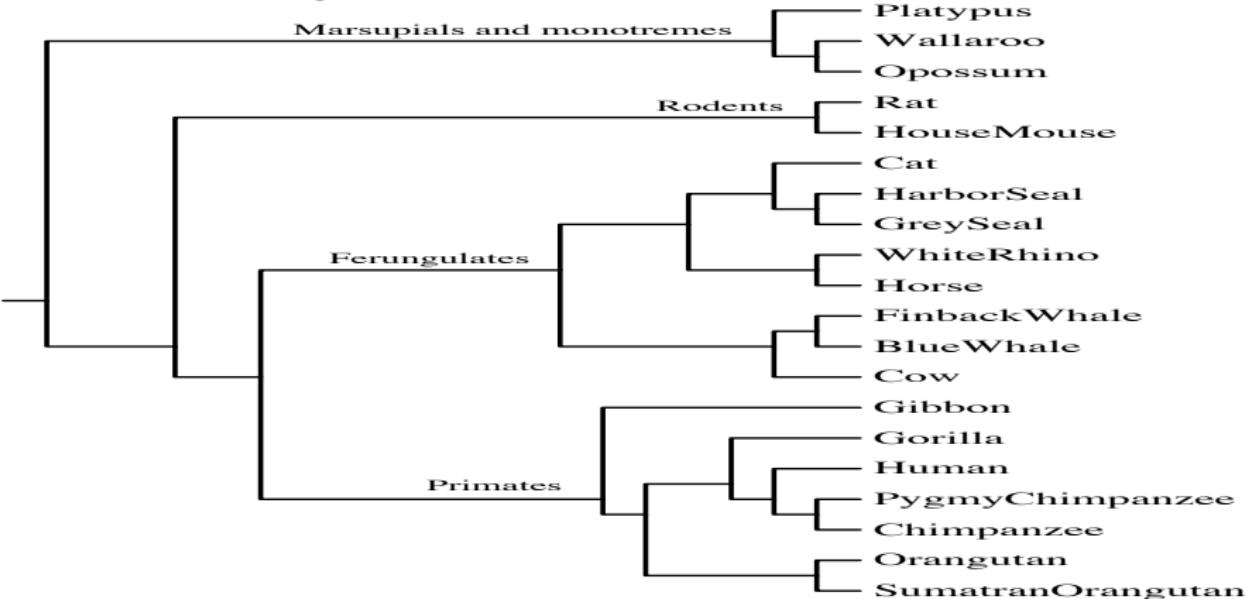
# Summary $K$ -means

- ▶ Represent clusters by cluster centers
- ▶ Highly non-convex NP hard optimization problem
- ▶ Heuristic: Lloyd's  $k$ -means algorithm
- ▶ Very easy to implement, hence very widely used.
- ▶ In my opinion:  $k$ -means works well for vector quantization (if you want to find a large number of clusters, say 100 or so). It does not work so well for small  $k$ , here you should consider spectral clustering.

# Linkage algorithms for hierarchical clustering

# Hierarchical clustering

Goal: obtain a complete hierarchy of clusters and sub-clusters in form of a **dendrogram**



cf. Chen/Li/Ma/Vitanyi (2004)

# Simple idea

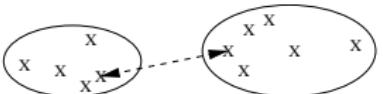
Agglomerative (bottom-up) strategy:

- ▶ Start: each point is its own cluster
- ▶ Then check which points are closest and “merge” them to form a new cluster
- ▶ Continue, always merge two “closest” clusters until we are left with one cluster only

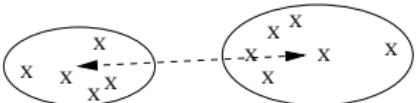
## Simple idea (2)

To define which clusters are “closest”:

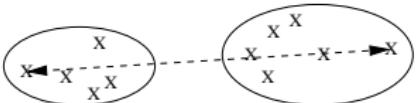
Single linkage:  $dist(C, C') = \min_{x \in C, y \in C'} d(x, y)$



Average linkage:  $dist(C, C') = \frac{\sum_{x \in C, y \in C'} d(x, y)}{|C| \cdot |C'|}$



Complete linkage:  $dist(C, C') = \max_{x \in C, y \in C'} d(x, y)$



# Linkage algorithms – basic form

## Input:

- Distance matrix  $D$  between data points (size  $n \times n$ )
- function  $dist$  to compute a distance between clusters (usually takes  $D$  as input)

**Initialization:** Clustering  $\mathcal{C}^{(0)} = \{C_1^{(0)}, \dots, C_n^{(0)}\}$  with  $C_i^{(0)} = \{i\}$ .

**While** the current number of clusters is  $> 1$ :

- find the two clusters which have the smallest distance to each other
- merge them to one cluster

**Output:** Resulting dendrogram

# Examples

... show matlab demos ...

`demo_linkage_clustering_by_foot()`

`demo_linkage_clustering_comparison()`

# Linkage algorithms tend to be problematic

Observations from practice:

- ▶ Linkage algorithms are very vulnerable to outliers
- ▶ One cannot “undo” a bad link

Theoretical considerations:

- ▶ Linkage algorithms attempt to estimate the density tree
- ▶ Even though this can be done in a statistically consistent way, estimating densities in high dimensions is extremely problematic and usually does not work in practice.

# History and References

- ▶ The original article: S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 2:241 - 254, 1967.
- ▶ A complete book on the topic: N. Jardine and R. Sibson. *Mathematical taxonomy*. Wiley, London, 1971.
- ▶ Nice, more up-to-date overview with application in biology: J. Kim and T. Warnow. Tutorial on phylogenetic tree estimation. ISMB 1999.

# Linkage algorithms — summary

- ▶ Attempt to estimate the whole cluster tree
- ▶ There exist many more ways of generating different trees from a given distance matrix.
- ▶ Advantage of tree-based algorithms: do not need to decide on “the correct” number of clusters, get more information than just a flat clustering
- ▶ However, one should be very careful about the results because they are very unstable, prone to outliers and statistically unreliable.

# A glimpse on spectral graph theory

## Literature:

- ▶ U. Luxburg. Tutorial on Spectral Clustering, Statistics and Computing, 2007.
- ▶ F. Chung: Spectral Graph Theory (Chapters 1 and 2).
- ▶ D. Spielman: Spectral Graph Theory, 2011.  
Book chapter whose official reference I was unable to verify, but you can simply download it from Dan Spielman's homepage (be aware that there are many papers on this subject by him, make sure you download the one that looks like a tutorial).

# What is it about?

General idea:

- ▶ many properties of graphs can be described by properties of the adjacency matrix and related matrices (“graph Laplacians”).
- ▶ In particular, the eigenvalues and eigenvectors can say a lot about the “geometry” of the graph.

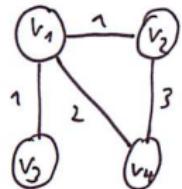
# Unnormalized Laplacians

# Unnormalized Graph Laplacians: Definition

Consider an undirected graph with non-negative edge weights  $w_{ij}$ .

Notation:

- ▶  $W :=$  the weight matrix of the graph
- ▶  $D := \text{diag}(d_1, \dots, d_n)$  the **degree matrix** of the graph
- ▶  $L := D - W$  the **unnormalized graph Laplacian matrix**



$$W = \begin{pmatrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 0 & 3 \\ 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \end{pmatrix} \quad D = \begin{pmatrix} 4 & & & \\ & 4 & & \\ & & 1 & \\ 0 & & & 5 \end{pmatrix} \quad L = \begin{pmatrix} 4 & -1 & -1 & -2 \\ -1 & 4 & 0 & -3 \\ -1 & 0 & 1 & 0 \\ -2 & -3 & 0 & 5 \end{pmatrix}$$

# Unnormalized Laplacians: Key property

## Proposition 20 (Key property)

Let  $G$  be an undirected graph. Then for all  $f \in \mathbb{R}^n$ ,

$$f^t L f = \frac{1}{2} \sum_{i,j=1}^n w_{ij} (f_i - f_j)^2.$$

**Proof.** Simply do the calculus:

## Unnormalized Laplacians: Key property (2)

$$\begin{aligned} f^t L f &= f^t D f - f^t W f \\ &= \sum_i d_i f_i^2 - \sum_{i,j} f_i f_j w_{ij} \\ &= \frac{1}{2} \left( \sum_i \left( \sum_j w_{ij} \right) f_i^2 - 2 \sum_{ij} f_i f_j w_{ij} + \sum_j \left( \sum_i w_{ij} \right) f_j^2 \right) \\ &= \frac{1}{2} \sum_{ij} w_{ij} (f_i - f_j)^2 \end{aligned}$$

□

# Why is it called “Laplacian”?

Where does the name “graph Laplacian” come from?

$$f^t L f = \frac{1}{2} \sum w_{ij} (f_i - f_j)^2$$

Interpret  $w_{ij} \sim 1/d(X_i, X_j)^2$

$$f^t L f = \frac{1}{2} \sum ((f_i - f_j)/d_{ij})^2$$

looks like a discrete version of the standard Laplace operator

$$\langle f, \Delta f \rangle = \int |\nabla f|^2 dx$$

Hence the graph Laplacian measures the variation of the function  $f$  along the graph:  $f^t L f$  is low if points that are close in the graph have similar values  $f_i$ .

# Unnormalized Laplacians: Spectral properties

## Proposition 21 (Simple spectral properties)

For an undirected graph with non-negative edge weights, the graph Laplacian has the following properties:

- ▶  $L$  is symmetric and positive semi-definite.
- ▶ Smallest eigenvalue of  $L$  is 0, corresponding eigenvector is  $\mathbb{1} := (1, \dots, 1)^t$ .
- ▶ Thus eigenvalues  $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$ .

# Unnormalized Laplacians: Spectral properties (2)

## Proof.

**Symmetry:**  $W$  is symmetric (graph is undirected),  $D$  is symmetric, so  $L$  is symmetric.

**Positive Semi-Definite:** by key proposition:

$$f^t L f = \frac{1}{2} \sum_{ij} w_{ij} (f_i - f_j)^2 \geq 0$$

**Smallest Eigenvalue/vector:** It is indeed an eigenvector because

$$L\mathbf{1} = D\mathbf{1} - W\mathbf{1} = 0$$

It is the smallest because all eigs are  $\geq 0$ . □

# Unnormalized Laplacians and connected components

## Proposition 22 (Relation between spectrum and clusters)

Consider an undirected graph with non-negative edge weights.

- ▶ Then the (geometric) multiplicity of eigenvalue 0 is equal to the number  $k$  of connected components  $A_1, \dots, A_k$  of the graph.
- ▶ The eigenspace of eigenvalue 0 is spanned by the characteristic functions  $\mathbb{1}_{A_1}, \dots, \mathbb{1}_{A_k}$  of those components (where  $\mathbb{1}_{A_i}(j) = 1$  if  $v_j \in A_i$  and  $\mathbb{1}_{A_i}(j) = 0$  otherwise).

# Unnormalized Laplacians and connected components (2)

## Proof, case k=1.

- ▶ Assume that the graph is connected.
- ▶ Let  $f$  be an eigenvector with eigenvalue 0.
- ▶ Want to show:  $f$  is a constant vector.

Here is the reasoning:

- ▶ By definition:  $Lf = 0$ .
- ▶ Exploiting this and the key proposition:

$$0 = f^t L f = \sum_{ij} w_{ij} (f_i - f_j)^2$$

- ▶ The right hand side can only be 0 if all summands are 0.

## Unnormalized Laplacians and connected components (3)

- ▶ Hence, for all pairs  $(i, j)$ :
  - ▶ either  $w_{ij} = 0$  (that is,  $v_i$  and  $v_j$  are not connected by an edge in the graph),
  - ▶ or  $f_i = f_j$ .

Consequently: if  $v_i$  and  $v_j$  are connected in the graph, then  $f_i = f_j$ .

In particular,  $f$  is constant on the whole connected component.

# Unnormalized Laplacians and connected components (4)

**Proof, case  $k > 1$ .**

- If the graph consists of  $k$  disconnected components, both the adjacency matrix and the graph Laplacian are block diagonal. In particular, each little block is the graph Laplacian of the corresponding connected component.

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_k \end{pmatrix}$$

# Unnormalized Laplacians and connected components (5)

- For each block (= each connected component), by the case  $k = 1$  we know that there is exactly one eigenvector for eigenvalue 0, and it is constant:

$$v_1(L_1) = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \quad v_1(L_2) = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \dots$$

- For the matrix  $L$  we then know that there are  $k$  eigenvalues 0, each one coming from one of the blocks. Padding the eigenvectors with zeros leads to the cluster indicator vectors:

# Unnormalized Laplacians and connected components (6)

$$v_1(L) = \begin{pmatrix} 1 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$v_2(L) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

...



# Unnormalized Laplacians and connected components (7)

QUESTION:

Consider a graph with  $k$  connected components:

WHY IS THERE NO CONTRADICTION BETWEEN PROPOSITION 22 (SECOND STATEMENT) AND PROPOSITION 21???

# Normalized Laplacians

# Normalized graph Laplacian

For various reasons (see below) it is better to normalize the graph Laplacian matrix.

Two versions:

- ▶ The “symmetric” normalized graph Laplacian

$$L_{sym} = D^{-1/2} L D^{-1/2}$$

(where the square root of the diagonal matrix  $D$  can be computed entry-wise).

- ▶ The “random walk graph Laplacian”

$$L_{rw} = D^{-1} L$$

We will now see that both normalized Laplacians are closely related, and have similar properties as the unnormalized Laplacian.

# Normalized Laplacians: First properties

Proposition 23 (Adapted key property)

For every  $f \in \mathbb{R}^n$  we have

$$f' L_{\text{sym}} f = \frac{1}{2} \sum_{i,j=1}^n w_{ij} \left( \frac{f_i}{\sqrt{d_i}} - \frac{f_j}{\sqrt{d_j}} \right)^2.$$

**Proof.** Similar to the unnormalized case. ☺

# Normalized Laplacians: First properties (2)

## Proposition 24 (Simple spectral properties)

Consider an undirected graph with non-negative edge weights.  
Then:

1.  $\lambda$  is an eigenvalue of  $L_{rw}$  with eigenvector  $u$   
 $\iff \lambda$  is an eigenvalue of  $L_{sym}$  with eigenvector  $w = D^{1/2}u$ .
2.  $\lambda$  is an eigenvalue of  $L_{rw}$  with eigenvector  $u$   
 $\iff \lambda$  and  $u$  solve the generalized eigenproblem  $Lu = \lambda Du$ .
3. 0 is an eigenvalue of  $L_{rw}$  with the constant one vector  $\mathbb{1}$  as eigenvector. 0 is an eigenvalue of  $L_{sym}$  with eigenvector  $D^{1/2}\mathbb{1}$ .
4.  $L_{sym}$  and  $L_{rw}$  are positive semi-definite and have  $n$  non-negative real-valued eigenvalues  $0 = \lambda_1 \leq \dots \leq \lambda_n$ .

# Normalized Laplacians: First properties (3)

## Proof.

Part (1): multiply the eigenvalue equation  $L_{\text{sym}}w = \lambda w$  with  $D^{-1/2}$  from the left and substitute  $u = D^{-1/2}w$ .

Part (2): multiply  $L_{\text{rw}}u = \lambda u$  with  $D$  from the left.

Part (3): Just plug it in the corresponding eigenvalue equations.

Part (4): The statement about  $L_{\text{sym}}$  follows from the adapted key property, and then the statement about  $L_{\text{rw}}$  follows from (2). ☺

# Normalized Laplacians and connected components

## Proposition 25 (Relation between spectrum and clusters)

Let  $G$  be an undirected graph with non-negative weights. Then the multiplicity  $k$  of the eigenvalue 0 of both  $L_{\text{rw}}$  and  $L_{\text{sym}}$  equals the number of connected components  $A_1, \dots, A_k$  in the graph. For  $L_{\text{rw}}$ , the eigenspace of 0 is spanned by the indicator vectors  $\mathbb{1}_{A_i}$  of those components. For  $L_{\text{sym}}$ , the eigenspace of 0 is spanned by the vectors  $D^{1/2}\mathbb{1}_{A_i}$ .

### Proof.

Analogous to the one for the unnormalized case. □

# Cheeger constant and isoperimetric problems

# Cheeger constant

Let  $G$  be an undirected graph with non-negative edge weights  $w_{ij}$ ,  $S \subset V$  be a subset of vertices,  $\bar{S} := V \setminus S$  its complement. Define:

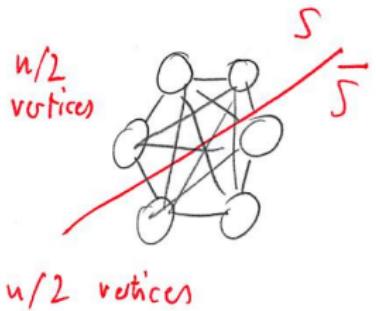
- ▶ Volume of the set:  $\text{vol}(S) := \sum_{s \in S} d(s)$
- ▶ Cut value:  $\text{cut}(S, \bar{S}) := \sum_{i \in S, j \in \bar{S}} w_{ij}$
- ▶ Cheeger constant:

$$h_G(S) := \frac{\text{cut}(S, \bar{S})}{\min\{\text{vol}(S), \text{vol}(\bar{S})\}}$$
$$h_G := \min_{S \subset V} h_G(S)$$

# Cheeger constant (2)

Example: a clique (=fully connected graph, including self-loops) with  $n$  vertices has  $h_G = \Theta(1)$ :

- $S$  that contains  $n/2$  vertices:



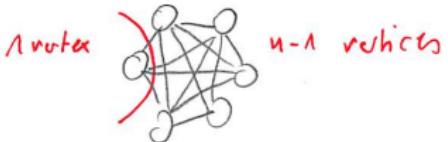
$$\text{cut}(S, \bar{S}) = \frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$$

$$|V(S)| = |V(\bar{S})| = \frac{n}{2} \cdot n = \frac{n^2}{2}$$

$$\Rightarrow h_G(S) = \frac{1}{2} = \Theta(1)$$

# Cheeger constant (3)

- $S$  that contains 1 vertex:



$$\text{cut}(\mathcal{S}, \bar{\mathcal{S}}) = n - 1$$

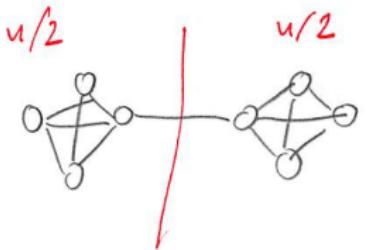
$$\text{vol}(\mathcal{S}) = 1 \cdot n \quad , \quad \text{vol}(\bar{\mathcal{S}}) = (n-1) \cdot n$$

$$\Rightarrow h_G(\mathcal{S}) = \frac{n-1}{n} = \Theta(1)$$

- Similarly for other sets  $S$ .

# Cheeger constant (4)

Example: two cliques with  $n/2$  vertices each, connected by a single edge. Results in  $h_G = \mathcal{O}(1/n^2)$



$$\text{cut}(\mathcal{S}, \bar{\mathcal{S}}) = 1$$

$$\text{vol}(\mathcal{S}) = \text{vol}(\bar{\mathcal{S}}) = \frac{n}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$$

$$\Rightarrow h_G(\mathcal{S}) = \frac{1}{\frac{n^2}{4}} = \Theta\left(\frac{1}{n^2}\right)$$

# Cheeger constant (5)

Intuition:

small Cheeger cuts are achieved for cuts that split the graph into reasonably big, tightly connected subgraphs (so that numerator is large) which are well clustered (denominator small).

# Isoperimetric problem

**In general geometry:**

Find a set  $S$  with given volume such that its boundary  $\partial S$  has as small a volume as possible.

E.g., among the subsets of area 1, the circle has the smallest circumference.

# Isoperimetric problem (2)

**In the context of graphs:**

Find a subset  $S \subset V$  with volume at least some given value and with  $\text{vol}(S) \leq \text{vol}(\bar{S})$  such that  $\text{cut}(S, \bar{S})$  is as small as possible.

This is equivalent to finding the Cheeger constant of a graph.

Intuition: The Cheeger constant measures whether the graph contains two well-separated clusters of volume at least some given value.

# Relation of the Cheeger constant and $\lambda_2$

## Theorem 26 (Cheeger inequality for graphs)

Consider a connected, undirected, unweighted graph. Let  $\lambda_2$  be the second-smallest eigenvalue of  $L_{\text{sym}}$ . Then

$$\frac{\lambda_2}{2} \leq h_G \leq \sqrt{2\lambda_2}$$

### Intuition:

- ▶ The Cheeger constant describes the cluster properties of a graph.
- ▶ The Cheeger constant is controlled by the second eigenvalue.

**Proof.** Blackboard / skipped (see Chung: Spectral Graph Theory).



Theorem ( $\lambda_2$  - bound), see Chung p 25 Lemma 2.1 :

Let  $\lambda_2$  be the second-smallest eig. of  $L_{\text{sym}}$ . And

$$h_G := \min_{S \subset V} \frac{\text{cut}(S, \bar{S})}{\min\{\text{vol}(S), \text{vol}(\bar{S})\}}. \quad \text{Then}$$

$$2h_G = \lambda_2.$$

Proof: By Rayleigh's principle,

$$\lambda_2 = \inf_{v \perp v_n} \frac{v^T L_{\text{sym}} v}{\|v\|^2} = \inf_{w \perp D^{1/2} v_n} \frac{v^T (D^{1/2} L D^{1/2}) v}{\|v\|^2}$$

Substitute  $w = D^{1/2} v$  to get

$$\lambda_2 = \inf_{w \perp D w} \frac{w^T L w}{\|D^{1/2} w\|^2} = \inf_{w \perp D w} \frac{\sum_{i,j} (w_i - w_j)^2}{\sum_j d_j w_j^2}$$

To construct an upper bound on  $\lambda_2$ , we now consider a particular vector  $w^*$ .

Let  $S^* := \arg\min_{S \subset V} \frac{\text{cut}(S, \bar{S})}{\min\{\text{vol}(S), \text{vol}(\bar{S})\}}$  the set that

minimizes the Cheeger cut, define the vector  $u$  by

$$u_i := \begin{cases} 1/\sqrt{\text{vol}(S^*)} & \text{if } i \in S^* \\ -1/\sqrt{\text{vol}(\bar{S}^*)} & \text{if } i \notin S^* \end{cases}$$

Note that  $u \perp D w$  because

$$\langle u, d \rangle = \underbrace{\sum_{i \in S^*} \frac{1}{\sqrt{\text{vol}(S^*)}} d_i}_{1} - \underbrace{\sum_{i \notin S^*} \frac{1}{\sqrt{\text{vol}(\bar{S}^*)}} d_i}_{1} = 0$$

Now we get

$$\lambda_2 \leq \frac{\sum_{i,j} (u_i - u_j)^2}{\sum_j d_j u_j^2} = \dots \text{see next page} = 2h_G$$

$$\begin{aligned}
 (\#) \quad & \frac{\sum_{i \in S^*} (u_i - u_j)^2}{\sum_j d_j u_j^2} = \text{cut by definition of } u \\
 & = \frac{\sum_{\substack{i \in S^* \\ j \in \bar{S}^*}} (u_i - u_j)^2}{\sum_{i \in S^*} \left( \frac{1}{\text{vol } S^*} \right)^2 + \sum_{j \in \bar{S}^*} \left( \frac{1}{\text{vol } \bar{S}^*} \right)^2} \\
 & \text{w.r.t. edges from } S^* \text{ to } \bar{S}^* \\
 \frac{1}{\text{vol } S^*} & = \frac{\sum_{i \in S^*} \left( \frac{1}{\text{vol } S^*} \right)^2 d_i}{\sum_{i \in S^*} \left( \frac{1}{\text{vol } S^*} \right)^2 d_i + \sum_{j \in \bar{S}^*} \left( \frac{1}{\text{vol } \bar{S}^*} \right)^2 d_j} = \frac{1}{\text{vol } \bar{S}^*} \\
 & = \text{cut}(S^*, \bar{S}^*) \left( \frac{1}{\text{vol } S^*} + \frac{1}{\text{vol } \bar{S}^*} \right) \\
 & \leq \text{cut}(S^*, \bar{S}^*) + 2 \max \left\{ \frac{1}{\text{vol } S^*}, \frac{1}{\text{vol } \bar{S}^*} \right\} \\
 & = 2 \frac{\text{cut}(S^*, \bar{S}^*)}{\min \{\text{vol } S^*, \text{vol } \bar{S}^*\}} = 2 b_G.
 \end{aligned}$$

# Spectral clustering

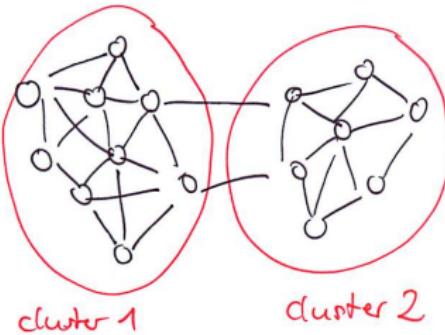
## Literature:

- ▶ U. Luxburg. Tutorial on Spectral Clustering, Statistics and Computing, 2007.
- ▶ The more recent editions of Tibshirani/Hastie/Friedman also contain a chapter on it.

# Clustering in graphs

General problem:

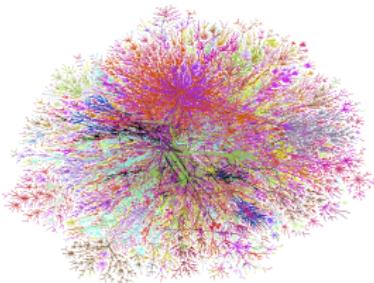
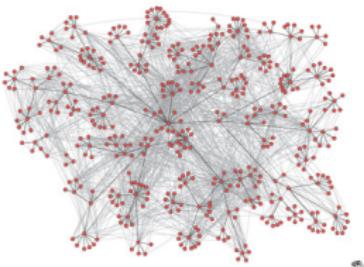
- ▶ Given a graph
- ▶ Want to find “clusters” in the graph:
  - ▶ many connections inside the cluster
  - ▶ few connections between different clusters



# Clustering in graphs (2)

Examples:

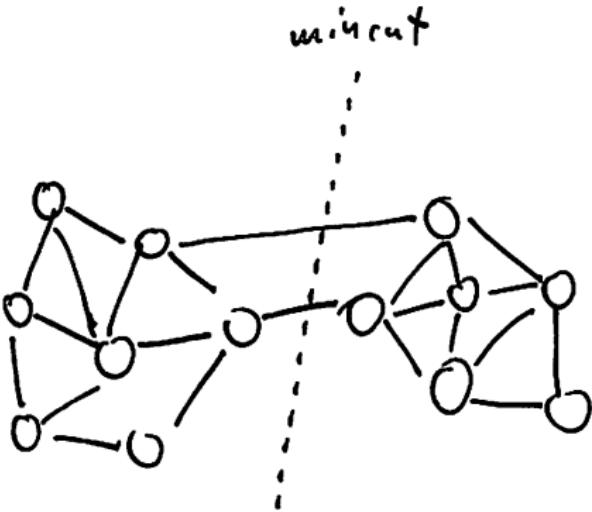
- ▶ Find “communities” in a social network (e.g., to analyze communication patterns in a company; to place targeted ads in facebook)
- ▶ Find groups of jointly acting proteins in a protein-interaction network
- ▶ Find groups of similar films ( $\leadsto$  “genres” )
- ▶ Find subgroups of diseases (for more specific medical treatment)



# Clustering in graphs (3)

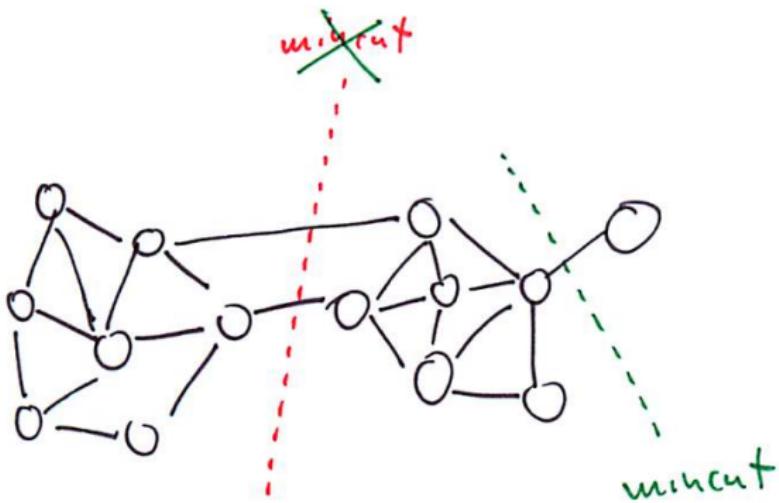
First idea:

- ▶ “Few connections between clusters”  $\approx$  “small cut”.
- ▶ Thus clustering  $\approx$  find a mincut in the graph.



# Clustering in graphs (4)

Problem: outliers



# Clustering in graphs (5)

Better idea: find two sets such that

- ▶ the cut between the sets is small
- ▶ each of the clusters is “reasonably large”

Some background on complexity of cut problems:

- ▶ Finding any mincut (without extra constraints) is easy and can be done in polynomial time.
- ▶ Finding the balanced mincut is NP hard
- ▶ Can we do something in between?

# RatioCut criterion

## Idea:

- ▶ want to define an objective function that measures the quality of a “nearly balanced cut”:
  - ▶ the smaller the cut value, the smaller the objective function
  - ▶ the more balanced the cut, the smaller the objective function

## Measuring the balancedness of a cut:

- ▶ Consider a partition  $V = A \cup B$ .
- ▶ Define  $|A| :=$  number of vertices in  $A$
- ▶ Introduce the **balancing term**  $1/|A| + 1/|B|$ .
- ▶ Observe: The balancing term is small when  $A$  and  $B$  have (approximately) the same number of vertices:

## RatioCut criterion (2)

- ▶ Example:  $n$  vertices in total.
  - ▶ Case  $|A| = n/2, |B| = n/2$ . Then  
 $1/|A| + 1/|B| = 4/n = O(1/n)$ .
  - ▶ Case  $|A| = 1, |B| = n - 1$ . Then  
 $1/|A| + 1/|B| = 1 + 1/(n - 1) = O(1)$ .
- ▶ In general: Under the constraint that  $|A| + |B| = n$ , the term  $\frac{1}{|A|} + \frac{1}{|B|}$  is minimal if  $|A| = |B|$ .

Formally, this can be seen by taking the derivative of the function  $f(a) = 1/a + 1/(n - a)$  with respect to  $a$  and setting it to 0.

## RatioCut criterion (3)

**Combining cut and balancing:** Define:

$$\text{cut}(A, B) := \sum_{i \in A, j \in B} w_{ij}$$

$$\text{RatioCut}(A, B) = \text{cut}(A, B) \left( \frac{1}{|A|} + \frac{1}{|B|} \right)$$

RatioCut gets smaller if

- ▶ the cut is smaller
- ▶ the clusters are more balanced

This is what we wanted to achieve.

## RatioCut criterion (4)

**Target** is now: find the cut with the minimal RatioCut value in a graph.

**Bad news:**

finding the global minimum of RatioCut is NP hard ☹

**Good news:**

But there exists an algorithm that finds very good solutions in most of the cases: spectral clustering ☺

# Unnormalized spectral clustering

# Goal: minimize RatioCut

Consider the following problem:

**Given an undirected graph  $G$  with non-negative edge weights. What is the minimal RatioCut in the graph?**

On the following slides we want to show how we can use spectral graph theory to achieve what we want.

# Relaxing balanced cut

Consider a graph with an even number  $n$  of vertices. For  $A \subset V$ , denote  $\bar{A} = V \setminus A$ . We want to solve the following problem:

$$\min_{A \subset V} \text{cut}(A, \bar{A}) \quad \text{subject to } |A| = |\bar{A}| \quad (*)$$

We want to rewrite the problem in a more convenient way.

Introduce  $f = (f_1, \dots, f_n)^t \in \mathbb{R}^n$  with

$$f_i = \begin{cases} +1 & \text{if } i \in A \\ -1 & \text{if } i \notin A. \end{cases}$$

Now observe:

$$\text{cut}(A, \bar{A}) = \sum_{i \in A, j \in \bar{A}} w_{ij} = \frac{1}{4} \sum_{i,j=1}^n w_{ij} (f_i - f_j)^2 = \frac{1}{2} f^t L f$$

## Relaxing balanced cut (2)

So we can rewrite problem (\*) equivalently as follows:

$$\min_f f^t L f \quad \text{subject to} \quad \sum_{i=1}^n f_i = 0 \quad \text{and} \quad f_i = \pm 1 \quad (**)$$

So far, we did not change the problem at all, we just wrote it in a different way.

It still looks difficult because it is a **discrete optimization problem**.

## Relaxing balanced cut (3)

Now we are going to **relax** the problem: we simply replace the difficult condition  $f_i = \pm 1$  by the two conditions  
 $f_i \in \mathbb{R}$  and  $\|f\| = 1$ :

$$\min_f f^t L f \quad \text{subject to} \quad \sum_{i=1}^n f_i = 0 \quad \text{and } f_i \in \mathbb{R} \quad \text{and } \|f\| = 1 \quad (\#)$$

Finally, observe that  $\sum_i f_i = 0 \iff f \perp \mathbb{1}$  where  $\mathbb{1}$  is the constant-one vector  $(1, 1, \dots, 1)$ . We obtain:

$$\min_f f^t L f \quad \text{subject to} \quad f \perp \mathbb{1} \quad \text{and } f_i \in \mathbb{R} \quad \text{and } \|f\| = 1 \quad (\#\#)$$

## Relaxing balanced cut (4)

The final observation is now:

- ▶  $\mathbb{1}$  is the smallest eigenvector of the matrix  $L$
- ▶ So Rayleigh's principle tells us that the solution to Problem (#) is  $f^*$  being the second-smallest eigenvector of  $L$ .

To transform the solution of the relaxed problem into a partition we simply consider the sign:

$$i \in A : \iff f_i^* \geq 0$$

# Relaxing balanced cut (5)

So we end up with the following algorithm:

## HardBalancedCutRelaxation( $G$ )

- 1 Input: Weight matrix (or adjacency matrix)  $W$  of the graph
- 2  $D :=$  the corresponding degree matrix
- 3  $L := D - W$  (the corresponding graph Laplacian)
- 4 Compute the second-smallest eigenvector  $f$  of  $L$
- 5 Define the partition  $A = \{i \mid f_i \geq 0\}$ ,  $\bar{A} = V \setminus A$
- 6 Return  $A, \bar{A}$

## Relaxing balanced cut (6)

Remarks about the relaxation approach:

- ▶ Our original problem was NP hard.
- ▶ We now solve a **relaxed problem** (in polynomial time, see below).
- ▶ In general, relaxing a problem does not lead to any guarantees about whether the solution of the relaxed problem is close to the solution of the original problem.
- ▶ This is also the case for spectral clustering. We can construct example graphs for which the relaxation is arbitrarily bad. However, such examples are very artificial.
- ▶ However, in practice the spectral relaxation works very well!!!

# Relaxing RatioCut

Now we want to solve the soft balanced mincut problem of optimizing ratiocut:

$$\min_{A \subset V} \text{RatioCut}(A, \bar{A}) \quad (*)$$

This goes along the same lines as the hard balanced mincut problem:

- ▶ Define particular values of  $f_i$ , namely

$$f_i = \begin{cases} +(|\bar{A}|/|A|)^{1/2} & \text{if } i \in A \\ -(|A|/|\bar{A}|)^{1/2} & \text{if } i \in \bar{A} \end{cases}$$

- ▶ Observe that we can write  $\text{RatioCut}(A, \bar{A}) = \dots = f^t L f$ .

## Relaxing RatioCut (2)

- ▶ So the RatioCut problem is equivalent to

$$\min_f f^t L f \text{ subject to } f_i \text{ "of the form given above"} \quad (**)$$

- ▶ Also observe that any  $f$  of the form given above satisfies  $\sum_{i \in V} f_i = \dots = 0$ . So any such  $f$  satisfies  $f \perp \mathbb{1}$ .
- ▶ So the RatioCut problem is also equivalent to

$$\min_f f^t L f \text{ subject to } f \perp \mathbb{1} \text{ and } f_i \text{ "of the form given above"} \quad (**)$$

- ▶ Now we relax the condition  $f_i$  "of the form given above" to  $f_i \in \mathbb{R}$ , and apply Rayleigh to see that we need to compute the second eigenvector.

## Relaxing RatioCut (3)

- As before, we assign points to  $A$  and  $\bar{A}$  according to the sign of the resulting  $f^*$ .

In pseudo-code, this algorithm is exactly the one we have already seen above. It is called (unnormalized) spectral clustering.

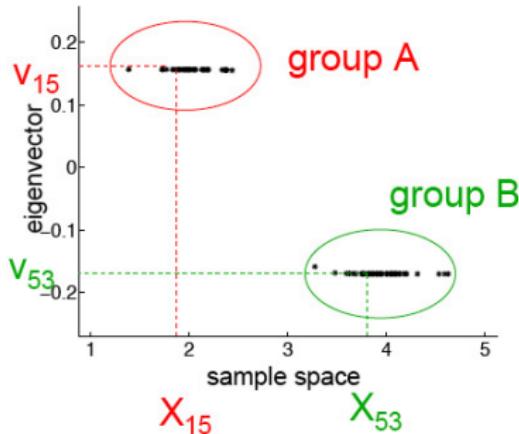
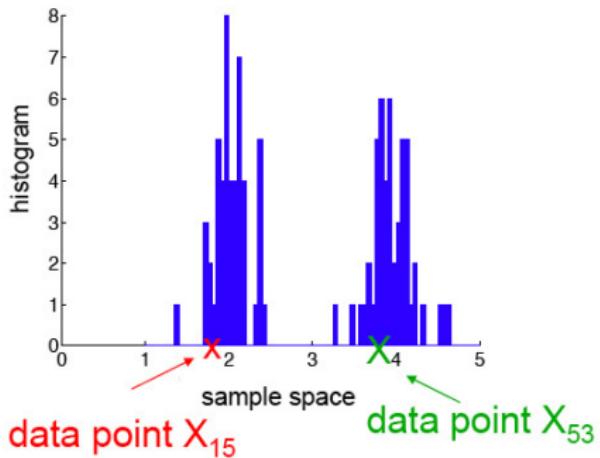
# (Unnormalized) Spectral Clustering, case two clusters

## UnnormalizedSpectralClustering( $G$ )

- 1 Input: Weight matrix (or adjacency matrix)  $W$  of the graph
- 2  $D :=$  the corresponding degree matrix
- 3  $L := D - W$  (the corresponding graph Laplacian)
- 4 Compute the second-smallest eigenvector  $f$  of  $L$
- 5 Define the partition  $A = \{i \mid f_i \geq 0\}$ ,  $\bar{A} = V \setminus A$
- 6 Return  $A, \bar{A}$

# Examples

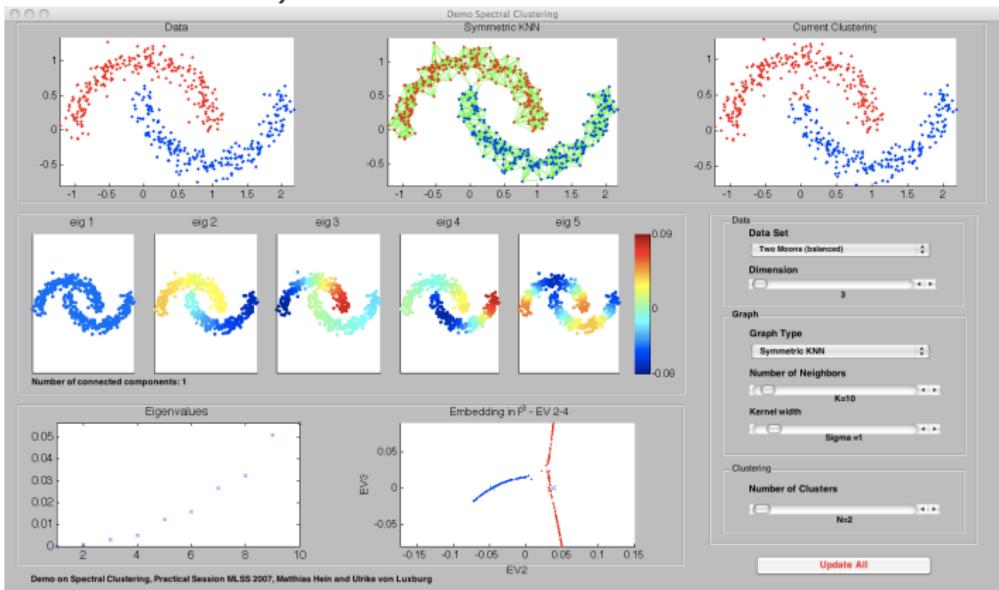
A couple of data points drawn from a mixture of Gaussians on  $\mathbb{R}$ .



$$\text{eigenvector } v = (-0.18, -0.18, 0.17, 0.18, \dots, \color{red}{0.17}, \dots -0.18, \dots)$$

# Examples (2)

For more examples, see the **INTERACTIVE SPECTRAL CLUSTERING DEMO** (can be downloaded from my webpage; you need Matlab to run it):



# Unnormalized spectral clustering for $k$ clusters

One can extend the algorithm to the case of  $k$  clusters.

General idea:

- ▶ The first  $k$  eigenvectors encode the cluster structure of  $k$  disjoint clusters.  
(To see this, consider the case of  $k$  perfectly disconnected clusters)
- ▶ To extract the cluster information from the first  $k$  eigenvectors, we construct the so-called **spectral embedding**:
  - ▶ Let  $V$  be the matrix that contains the first  $k$  eigenvectors as columns.
  - ▶ Now define new points  $Y_i \in \mathbb{R}^k$  as the  $i$ -th row of matrix  $V$ .

# Unnormalized spectral clustering for $k$ clusters (2)

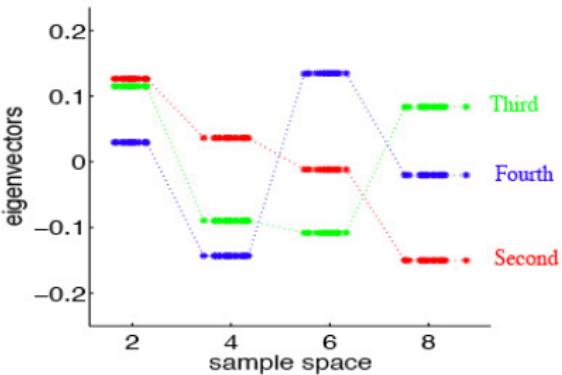
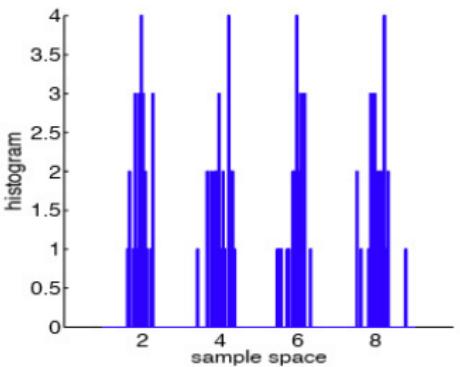
- ▶ Note: in the “ideal case” (disconnected clusters) the  $Y_i$  are the same for all points in the same cluster.

$$V = \begin{pmatrix} & & b_2 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{pmatrix}$$

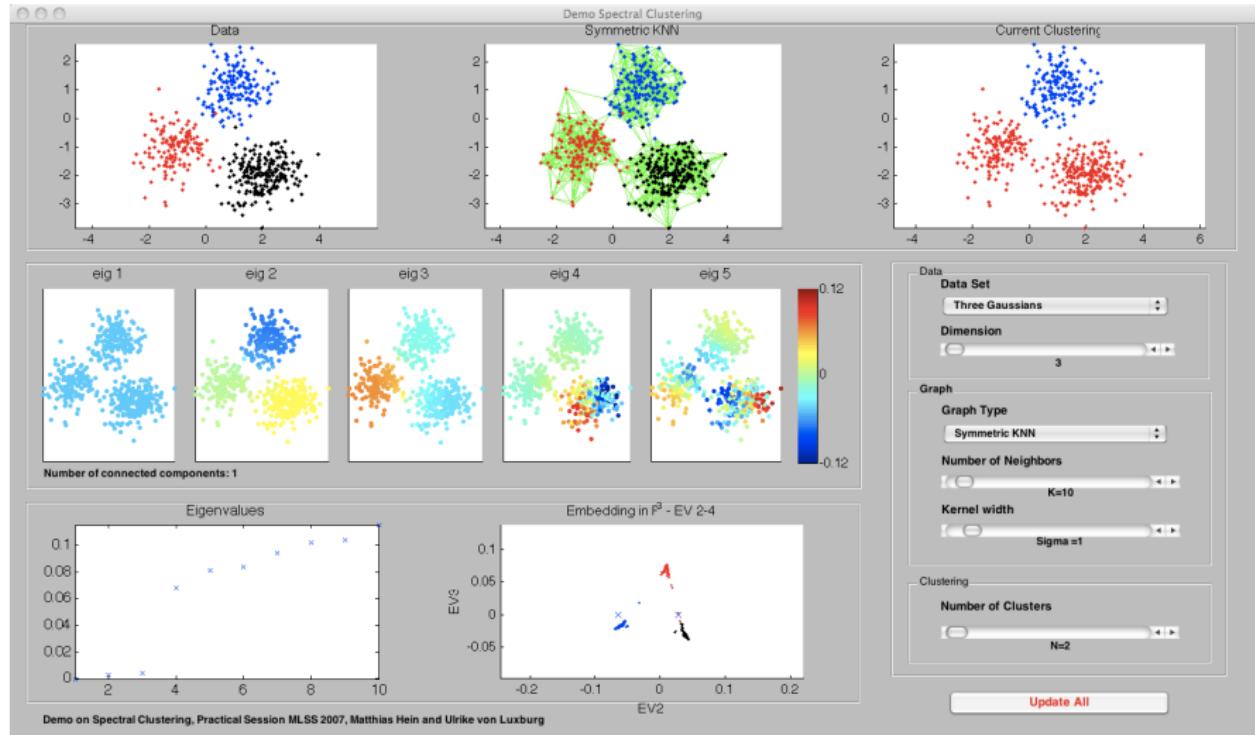
$\left. \right\} (1 \ 0 \ 0)$   
 $\left. \right\} (0 \ 1 \ 0)$   
 $\left. \right\} (0 \ 0 \ 1)$

- ▶ Idea: they are “nearly the same” if we still have nice (but not perfect) clusters.
- ▶ In particular, any simple algorithm can recover the cluster membership based on the embedded points  $Y_i$ . We use  $k$ -means to do so.

# Unnormalized spectral clustering for $k$ clusters (3)



# Unnormalized spectral clustering for $k$ clusters (4)



# Unnormalized spectral clustering for $k$ clusters (5)

## UnnormalizedSpectralClustering( $G$ )

- 1 Input: Weight matrix (or adjacency matrix)  $W$  of the graph
- 2  $D :=$  the corresponding degree matrix
- 3  $L := D - W$  (the corresponding graph Laplacian)
- 4 Compute the  $n \times k$  matrix  $V$  that contains the first  $k$  eigenvectors as columns.
- 5 Define the new data points  $Y_i \in \mathbb{R}^k$  to be the rows of the matrix  $V$ . **This is sometimes called the spectral embedding.**
- 6 Now cluster the points  $(Y_i)_{i=1,\dots,n}$  by the  $k$ -means algorithm.

# Unnormalized spectral clustering for $k$ clusters

## (6)

Some more intuition:

- ▶ Seems funny: we first say we want to use spectral clustering, and in the end we run  $k$ -means.
- ▶ The point is that the spectral embedding is such a clever transformation of the original data that after this transformation the cluster structure is “obvious”, we just have to extract it by a simple algorithm.

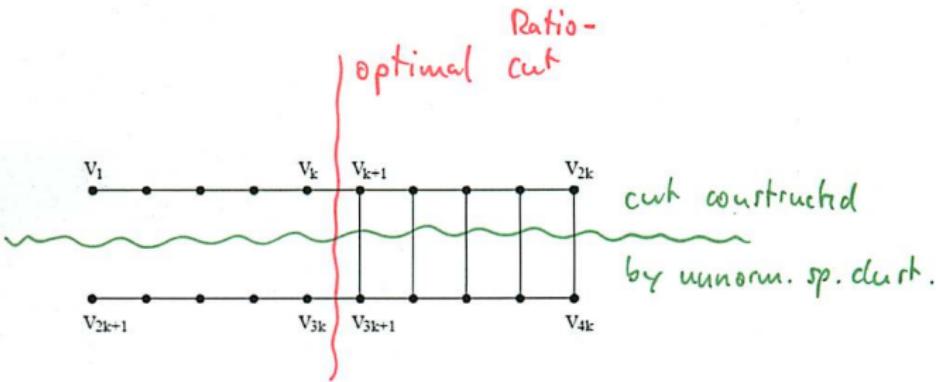
# Analysis: Running time

The bottleneck of the algorithm is the computation of the eigenvector:

- ▶ In general, the first eigenvectors of a symmetric matrix can be computed in time  $O(n^3)$
- ▶ However, one can do much better on sparse matrices (running time then depends on the sparsity and on other conditions such as the “spectral gap”).

# Analysis: Approximation guarantees

- As hinted above: we cannot guarantee that the solution we find by unnormalized spectral clustering is close to the minimizer of RatioCut
- In the following example, the cut constructed by spectral clustering is  $c \cdot n$  times larger than the best RatioCut:



Note that this example relies heavily on symmetry.

Guattery, S., Miller, G. (1998). On the quality of spectral separators. SIAM Journal of Matrix Anal. Appl., 1998.

## Analysis: Approximation guarantees (2)

- ▶ However, despite the lack of approximation guarantees, it performs extremely well in practice. In terms of clustering performance, it is the state of the art and one of the most widely used “modern” algorithms for clustering.

# Normalized spectral clustering

## Normalized cut criterion

We have seen that the unnormalized spectral clustering algorithm solves the (relaxed) problem of minimizing Ratiocut.

For various reasons (see later), it turns out to be better to consider the following objective function called **normalized cut**:

$$\text{Ncut}(A, B) = \text{cut}(A, B) \left( \frac{1}{\text{vol}(A)} + \frac{1}{\text{vol}(B)} \right)$$

This looks very similar to RatioCut, but we measure the size of the sets  $A$  and  $B$  not by their number of vertices, but by the weight of their edges:

$$\text{vol}(A) = \sum_{i \in A} d_i$$

## Normalized cut criterion (2)

By a derivation that is very similar to what we have seen for Ratiocut:

- ▶ relaxing the problem to minimizing Ncut leads to clustering the eigenvectors of the random walk Laplacian  $L_{rw}$ .
- ▶ For computational reasons, one replaces the eigendecomposition of  $L_{rw}$  by the one of  $L_{sym}$  (WHY?)

# Minimizing normalized cut

Relaxation approach, very similar to the one for Ratiocut, leads to the following algorithm:

## NormalizedSpectralClustering( $G$ )

- 1 Input: Weight matrix (or adjacency matrix)  $W$  of the graph
- 2  $D :=$  the corresponding degree matrix
- 3  $L_{sym} := D^{-1/2}(D - W)D^{-1/2}$  (the normalized graph Laplacian)
- 4 Compute the second-smallest eigenvector  $f$  of  $L_{sym}$
- 5 Compute the vector  $g = D^{-1/2}f$
- 6 Define the partition  $A = \{i \mid g_i \geq 0\}$ ,  $\bar{A} = V \setminus A$
- 7 Return  $A, \bar{A}$

# Normalized vs. unnormalized spectral clustering

You should always prefer the normalized spectral clustering algorithm. There are several theoretical (and practical) results that show:

- ▶ The normalized graph Laplacian is better behaved.
- ▶ The unnormalized one can lead to systematically wrong solutions.

Details omitted.

# History

- ▶ Has been discovered and rediscovered several times since the 1970ies, but went pretty much unnoticed.
- ▶ Breakthrough papers:
  - ▶ *Shi, J. and Malik, J. Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2000.*
  - ▶ *Meila, M. and Shi, J. A random walks view of spectral segmentation. AISTATS, 2001.*
  - ▶ *Ng, A., Jordan, M., and Weiss, Y. On spectral clustering: analysis and an algorithm. NIPS, 2002.*
- ▶ By now, it has been established as the most popular “modern” clustering algorithm, with many theoretical results underpinning its usefulness.

# Spectral clustering summary

- ▶ Spectral clustering tries to solve a balanced cut problem:
  - ▶ minimize Ratiocut ( $\leadsto$  unnormalized spectral clustering)
  - ▶ minimize Ncut ( $\leadsto$  normalized spectral clustering)
- ▶ Both these problems are discrete optimization problems and NP hard to solve.
- ▶ Spectral clustering solves a relaxed version of these problems.
- ▶ In theory, there are no approximation guarantees — the relaxed solution can be miles away from the one we want.  
In practice, it works very well and is state of the art in many applications.
- ▶ Running time complexity can be as bad as  $O(n^3)$ , but for sparse graphs it is very fast.

**Normalized spectral clustering is THE modern state of the art clustering algorithm.**

(\*) Correlation clustering: SKIPPED

# Introduction to learning theory

# The standard theory for supervised learning

# Learning Theory: setup and main questions

Literature on learning theory:

- ▶ High-level: U. von Luxburg and B. Schölkopf. Statistical Learning Theory: Models, Concepts, and Results. 2011.
- ▶ More technical: Bousquet, Boucheron, Lugosi: Introduction to statistical learning theory, 2003
- ▶ The “classic” book (technical): Devroye, Györfi, Lugosi: A probabilistic theory of pattern recognition. Springer, 1996
- ▶ Some of the general text books cover different aspects, for example the books by Shalev-Shwartz, Ben-David and Mohri, Rostamizadeh, Talwalkar.

# Statistical learning theory

On an abstract level, SLT tries to answer questions such as:

- ▶ Which learning tasks can be performed by computers in general (positive and negative results)?
- ▶ What kind of assumptions do we have to make such that machine learning can be successful?
- ▶ What are the key properties a learning algorithm needs to satisfy in order to be successful?
- ▶ Which performance guarantees can we give on the results of certain learning algorithms?

**In the following we focus on the case of binary classification, for which the theory is well-understood.**

# The framework

**The data.** Data points  $(X_i, Y_i)$  are an i.i.d. sample from some underlying (unknown) probability distribution  $P$  on the space  $\mathcal{X} \times \{\pm 1\}$ .

**Goal.** Our goal is to learn a deterministic function  $f : \mathcal{X} \rightarrow \{\pm 1\}$  such that the expected loss (risk) according to some given loss function  $\ell$  is as small as possible. In classification, the natural loss function is the 0-1-loss.

# The framework (2)

## Assumptions we (do not) make:

- ▶ We do not make any assumption on the **underlying distribution**  $P$  that generates our data, it can be anything.
- ▶ True **labels** do not have to be a deterministic function of the input (consider the example of predicting male/female based on body height).
- ▶ Data points have been sampled **i.i.d.**
- ▶ Data does not change over time (the ordering of the training points does not matter, and the distribution  $P$  does not change),
- ▶ The distribution  $P$  is unknown at the time of learning.

# Recap: Bayes classifier

## The Bayes classifier

The Bayes classifier for a particular learning problem is the classifier that achieves the minimal expected risk.

Have already seen: if we knew the underlying distribution  $P$ , then we also know the Bayes classifier (just look at the regression function).

The challenge is that we do not know  $P$ . The goal is now to construct a classifier that is “as close to the Bayes classifier” as possible. Now let’s become more formal.



# Convergence and consistency

Assume we have a set of  $n$  points  $(X_i, Y_i)$  drawn from  $P$ . Consider a given function class  $\mathcal{F}$  from which we are allowed to pick our classifier. Denote:

- ▶  $f^*$  the Bayes classifier corresponding to  $P$ .
- ▶  $f_{\mathcal{F}}$  the best classifier in  $\mathcal{F}$ , that is

$$f_{\mathcal{F}} = \operatorname{argmin}_{f \in \mathcal{F}} R(f)$$

- ▶  $f_n$  the classifier chosen from  $\mathcal{F}$  by some training algorithm on the given sample of  $n$  points.

Now consider the following definitions:

## Convergence and consistency (2)

1. A learning algorithm is called **consistent with respect to  $\mathcal{F}$  and  $P$**  if the risk  $R(f_n)$  converges in probability to the risk  $R(f_{\mathcal{F}})$  of the best classifier in  $\mathcal{F}$ , that is for all  $\varepsilon > 0$ ,

$$P(R(f_n) - R(f_{\mathcal{F}}) > \varepsilon) \rightarrow 0 \text{ as } n \rightarrow \infty.$$

2. A learning algorithm is called **Bayes-consistent with respect to  $P$**  if the risk  $R(f_n)$  converges to the risk  $R(f^*)$  of the Bayes classifier, that is for all  $\varepsilon > 0$ ,

$$P(R(f_n) - R(f^*) > \varepsilon) \rightarrow 0 \text{ as } n \rightarrow \infty.$$

3. A learning algorithm is called **universally consistent with respect to  $\mathcal{F}$  (resp. universally Bayes-consistent)** if it is consistent with respect to  $\mathcal{F}$  (resp. Bayes-consistent) for all probability distributions  $P$ .

## Convergence and consistency (3)

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

- ▶ Here consistency means that our decisions are not affected systematically from the fact that we only get to see a finite sample, rather than the full space. In other words, all “finite sample effects” cancel out once we get to see enough data.
- ▶ If a learning algorithm is consistent, it means that it does not overfit when it gets to see enough data (low estimation error, low variance).
- ▶ Consistency with respect to  $\mathcal{F}$  does not tell us anything about underfitting (bias, approximation error; this depends on the choice of  $\mathcal{F}$ ).

# Empirical risk minimization (ERM)

True risk of a function:  $R(f) = E(\ell(X, f(X), Y))$

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

Empirical risk minimization: given  $n$  training points  $(X_i, Y_i)_{i=1,\dots,n}$  and a fixed function class  $\mathcal{F}$ , select the function  $f_n$  that minimizes the training error on the data:

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

Earlier we informally discussed that ERM won't work if the function class  $\mathcal{F}$  is "too large". We are now going to make this formal.

# Controlling the estimation error: generalization bounds

# Law of large numbers and concentration

Recall from probability theory:

## Proposition 27 (Law of large numbers, simplest version)

Let  $(Z_i)_{i \in \mathbb{N}}$  be a sequence of independent random variables that have been drawn according to some probability distribution  $P$ , denote its expectation as  $E(Z)$ . Then (under mild assumptions)

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

## Law of large numbers and concentration (2)

Even more, there exist very strong guarantees on how fast this convergence takes place:

**Proposition 28** (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  $\varepsilon > 0$

$$P\left(\left|\frac{1}{n} \sum_{i=1}^n Z_i - E(Z)\right| \geq \varepsilon\right) \leq 2 \exp(-2n\varepsilon^2).$$

## Law of large numbers and concentration (3)

Now consider our scenario of binary classification.

### Proposition 29 (Risks converge for fixed function)

Fix a function  $f_0 \in \mathcal{F}$ . Then, for this **fixed function**  $f_0$ ,

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

DO YOU SEE WHY?

# Law of large numbers and concentration (4)

Proof:

- ▶ Apply the Hoeffding bound to the variables  $Z_i := \ell(f_0(X_i), Y_i)$ . This leads to convergence in probability.
- ▶ (For those who know about probability theory: To get almost sure convergence, you need to do one extra step, namely apply the Borel-Cantelli lemma. Key is that  $\sum_{n=1}^{\infty} \exp(-2n\varepsilon^2)$  is finite. Exercise.)

## Law of large numbers and concentration (5)

Question: Let  $f_n$  be the function selected by empirical risk minimization. Does the LLN imply that

$$R_n(f_n) - R(f_n) \rightarrow 0 \quad ???????????.$$

# Law of large numbers and concentration (6)

NO!!!

Here is a simple counter-example:

- ▶  $\mathcal{X} = [0, 1]$  with uniform distribution; labels deterministic with  $x < 0.5 \implies y = -1$  and  $x \geq 0.5 \implies y = +1$
- ▶ Draw  $n$  training points
- ▶ Define  $f_n$  as follows: for all points in the training sample, predict the training label; for all other points predict -1.

Here we have  $R_n(f_n) = 0$  but  $R(f_n) = 0.5$  for all  $n$ .

DO YOU SEE WHY WE CANNOT APPLY THE LLT, WHERE DOES IT GOE WRONG???

# Uniform convergence

Want to have a condition that is sufficient for the convergence of the empirical risk of the data-dependent function  $f_n$ :

- ▶ We require that **for all functions in  $\mathcal{F}$** , the empirical risk (as measured on the data) has to be close to the true risk.
- ▶ Formally: for any  $\varepsilon > 0$ , we want that with high probability,

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

- ▶ Note the logic behind this condition: if  $R_n(f)$  and  $R(f)$  are close for all functions  $f \in \mathcal{F}$ , they particularly will be close for the function  $f_n$  that has been chosen by the classification algorithm.

## Uniform convergence (2)

Note that in the counter-example above, this requirement is clearly not satisfied.

WHY EXACTLY?

## Uniform convergence (3)

Definition:

We say that the law of large number holds uniformly over a function class  $\mathcal{F}$  if for all  $\varepsilon > 0$ ,

$$P\left(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| \geq \varepsilon\right) \rightarrow 0 \text{ as } n \rightarrow \infty.$$

# Uniform convergence: sufficient for consistency

Relatively easy to see:

Proposition 30 (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 is that uniform convergence is also necessary, this is in fact a very deep result:

**Theorem 31** (Vapnik & Chervonenkis, 1971)

Let  $\mathcal{F}$  be any function class. Then empirical risk minimization is uniformly consistent with respect to  $\mathcal{F}$  if and only if uniform convergence holds:

$$P\left(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| > \varepsilon\right) \rightarrow 0 \text{ as } n \rightarrow \infty, \quad (1)$$

The proof is beyond the scope of this lecture.

# Uniform convergence: necessary for consistency (2)

But the big question is now:

How do I know whether we have uniform consistency for some function class  $\mathcal{F}$ ???

# Capacity measures for function classes

# Finite classes

# Capacity measures: intuition

Have seen:

- ▶ If a function class is too large (as in the counter-example), then we don't have uniform convergence.
- ▶ If a function class is small (say, it only consists of a single function), then we have uniform convergence.

We now want to come up with ways to measure the size of a function class — in such a way that we can bound the term

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

# Generalization bound for finite classes

Recall the Hoeffding bound for a fixed function  $f_0$ :

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

Now consider a function class with finitely many functions:  
 $\mathcal{F} = \{f_1, \dots, f_m\}$ . We get:

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

## Generalization bound for finite classes (2)

Have:

$$\Pr\left(\sup_{f \in \mathcal{F}} |R(f) - R_n(f)| \geq \varepsilon\right) \leq \underbrace{2m \exp(-2n\varepsilon^2)}_{=: \delta} \quad (*)$$

Let us try to write this more intuitively: Define the right hand side as  $\delta$  and solve for  $\varepsilon$ :

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

With this,  $(*)$  becomes the following generalization bound:

## Generalization bound for finite classes (3)

### Theorem 32 (Generalization bound for finite classes)

Assume  $\mathcal{F}$  is finite and contains  $m$  functions. 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)

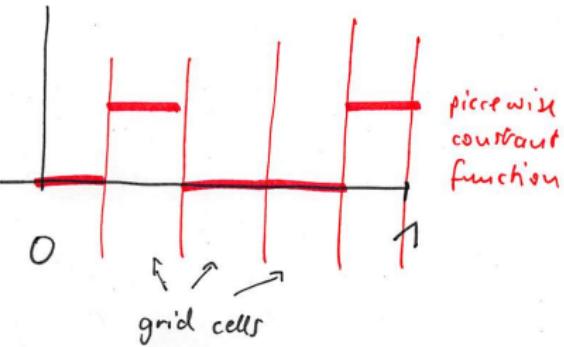
Let's digest this bound:

- ▶ 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 whole bound only holds with probability  $1 - \delta$ . When we decrease  $\delta$  (higher confidence), the bound gets worse.
- ▶ If  $m$  is fixed, and the confidence value  $\delta$  is fixed, and  $n \rightarrow \infty$ , then the empirical risk converges to the true risk. The speed of convergence is of the order  $1/\sqrt{n}$ .
- ▶ If you want to grow your function space with  $n$  in order to be able to fit more accurately if you have more data, you need to make sure that  $(\log m)/n \rightarrow 0$  if you want to get consistency.

# Generalization bound for finite classes (5)

## EXERCISE:

Consider  $\mathcal{X} = [0, 1]$ , split it into a grid of  $k$  cells of the same size. As function class, consider all functions that are piecewise constant (0 or 1) on all cells.



- ▶ Prove that ERM is consistent if  $k$  is fixed.
- ▶ Consider the case that  $k$  grows with  $n$ . How fast can  $k$  grow such that we still have consistency?
- ▶ What about the bias?

# Generalization bound for finite classes (6)

Bottom line:

- ▶ For finite function classes, we can measure the size of  $\mathcal{F}$  by its number  $m$  of functions.
- ▶ This leads to a generalization bound with plausible behavior.

However, what should we do if  $\mathcal{F}$  is infinite (say, space of all linear functions)? Then the approach above does not work ... WHY?

# Shattering coefficient

# Shattering coefficient: definition

We now want to measure the capacity of an infinite class of functions. The most basic such capacity measure is the following:

**Definition:** For a given sample  $X_1, \dots, X_n \in \mathcal{X}$  and a function class  $\mathcal{F}$  define  $\mathcal{F}_{X_1, \dots, X_n}$  as the set of those functions that we get by restricting  $\mathcal{F}$  to the sample:

$$\mathcal{F}_{X_1, \dots, X_n} := \{f|_{X_1, \dots, X_n}; f \in \mathcal{F}\}$$

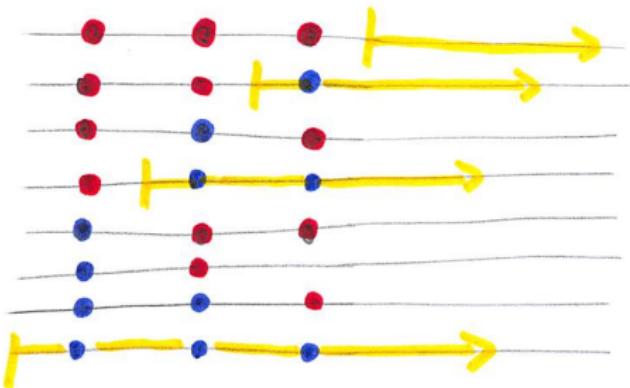
The **shattering coefficient  $\mathcal{N}(\mathcal{F}, n)$**  of a function class  $\mathcal{F}$  is defined as the maximal number of functions in  $\mathcal{F}_{X_1, \dots, X_n}$ :

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

# Shattering coefficient: definition (2)

Example 1:  $\mathcal{X} = \mathbb{R}$ ,  $\mathcal{F}$  as below (positive class = right half-space)

$$\mathcal{X} = \mathbb{R}$$



$$\tilde{\mathcal{F}} = \left\{ \mathbb{1}_{[a, \infty]} \mid a \in \mathbb{R} \right\}$$

(all half-spaces with the "infinite end" at the right side)

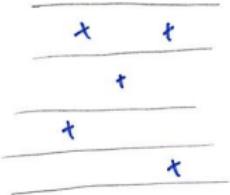
With  $\tilde{\mathcal{F}}$  we can only realize 4 out of the 8 possible labelings.  
Thus  $N(\tilde{\mathcal{F}}, 3) = 4$ .

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

# Shattering coefficient: definition (3)

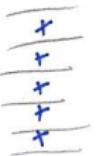
Example 2:  $\mathcal{X} = \mathbb{R}^2$ ,  $\mathcal{F}$  such that positive class = space above a horizontal line

Data 1 :



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

Data 2 :



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

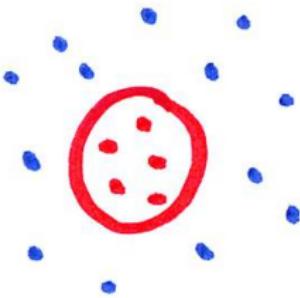
Data 3:

$x \ x \ x \ x \ x$

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

## Shattering coefficient: definition (4)

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



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

# Shattering coefficient: generalization bound

Theorem 33 (Generalization bound with shattering coefficient)

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

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

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

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

# Proof of Theorem 33 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 34 (Symmetrization lemma)

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

(Proof elementary, omitted)

## Proof of Theorem 33 by symmetrization (2)

What is the point of symmetrization?

- ▶ The right hand side only depends on the values of the functions  $f$  on the two samples:  
If two functions  $f$  and  $g$  coincide on all points of the original sample and the ghost sample, that is  $f(x) = g(x)$  for all  $x$  in the samples, then  $R_n(f) = R_n(g)$  and  $R'_n(f) = R'_n(g)$ .
- ▶ So the supremum over  $f \in \mathcal{F}$  in fact only runs over finitely many functions: all possible binary functions on the two samples.
- ▶ The number of such functions is bounded by the shattering coefficient  $\mathcal{N}(\mathcal{F}, 2n)$ .
- ▶ Now Theorem 33 is a consequence of Theorem 32.

# Discussion of the generalization bound with shattering coefficient

- ▶ The bound is analogous to the one for finite function classes, just the number  $m$  of functions has been replaced by the shattering coefficient.
- ▶ Intuitively, the shattering coefficient measures “how powerful” a function class is, how many different labelings of a data set it can possibly realize.
- ▶ Overfitting happens if a function class is very powerful and can in principle fit everything. Then we don’t get consistency, the shattering coefficient is large.  
The smaller the shattering coefficient, the less prone we are to overfitting (in the extreme case of one function, we don’t overfit).

## Discussion of the generalization bound with shattering coefficient (2)

- ▶ To prove consistency of a classifier, we need to establish that  $\log \mathcal{N}(\mathcal{F}, 2n)/n \rightarrow 0$  as  $n \rightarrow \infty$ .

Intuitively: the number of possibilities in which a data set can be labeled has to grow at most polynomially in  $n$ .

- ▶ Shattering coefficients are complicated to compute and to deal with. To prove consistency, we would need to know how fast the shattering coefficients grow with  $n$  (exponentially or less).

We now study a tool that can help us with this.

# VC dimension

## VC dimension: Definition

**Definition:** We say that a function class  $\mathcal{F}$  **shatters** a set of points  $X_1, \dots, X_n$  if  $\mathcal{F}$  can realize all possible labelings of the points, that is  $|\mathcal{F}_{X_1, \dots, X_n}| = 2^n$ .

The **VC dimension of  $\mathcal{F}$**  is defined as the largest number  $n$  such that there exists a sample of size  $n$  which is shattered by  $\mathcal{F}$ . Formally,

$$\text{VC}(\mathcal{F}) = \max\{n \in \mathbb{N} \mid \exists X_1, \dots, X_n \in \mathcal{X} \text{ s.t. } |\mathcal{F}_{X_1, \dots, X_n}| = 2^n\}.$$

If the maximum does not exist, the VC dimension is defined to be infinity.

(VC stands for Vapnik-Chervonenkis, the people who invented it)

# VC dimension: Definition (2)

Example: positive class = closed interval

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

- Can find a set of two points that can be shattered:



- Can we find a set of three points that can be shattered?

No!



Can never realize their labeling

$$\Rightarrow VC(\mathcal{F}) = 2$$

# VC dimension: Definition (3)

Example: positive class = interior of a axis aligned rectangle

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

- 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

• • •      => VC < 5

# VC dimension: Definition (4)

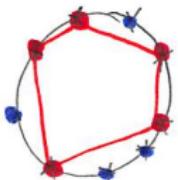
Example: positive class = interior of a convex polygon

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

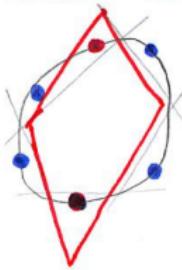
Then:  $VC = 2d + 1$ .



Lower bound:  $2d+1$  points on a circle can be shattered:



If we have less than  $d$  red points



If we have more than  $d$  red points

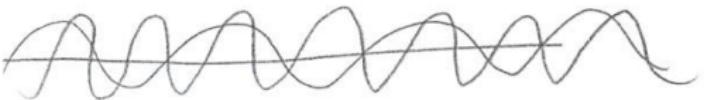
Upper bound: more technical, prove that max. Number of shattered points is achieved for points on a circle.

# VC dimension: Definition (5)

Example: sine waves

$$\chi = \mathbb{R}, \quad \mathcal{F} = \{ \text{sgn}(\sin(tx)) \mid t \in \mathbb{R} \}$$

Then  $VC = \infty$ .



## VC dimension: Definition (6)

Finally, examples that are relevant for practice (SVMs!):

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

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

# VC dimension: Sauer-Shelah Lemma

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

Proposition 35 (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

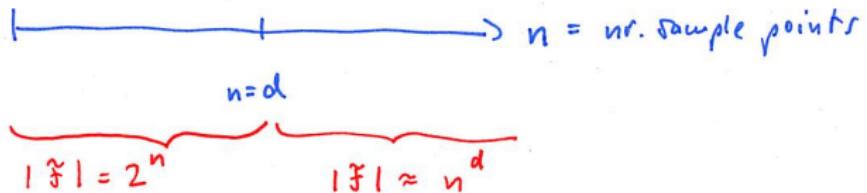
$$\mathcal{N}(\mathcal{F}, n) \leq \left(\frac{en}{d}\right)^d.$$

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

## Theorem 36 (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 37

Empirical risk minimization is consistent with respect to  $\mathcal{F}$  if and only if  $\text{VC}(\mathcal{F})$  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  $\varepsilon$ ?

- ▶ 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  $d/\varepsilon^2$  many sample points.

# Rademacher complexity

# Rademacher complexity

The shattering coefficient is a purely combinatorial object, it does not take into account what the actual probability distribution is. This seems suboptimal.

**Definition:** Fix a number  $n$  of points. Let  $\sigma_1, \dots, \sigma_n$  be i.i.d. tosses of a fair coin (result is -1 or 1 with probability 0.5 each). The Rademacher complexity of a function class  $\mathcal{F}$  with respect to  $n$  is defined as

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

The expectation is both over the draw of the random points  $X_i$  and the random labels  $\sigma_i$ .

It measures how well a function class can fit random labels.

## Rademacher complexity (2)

There exist a number of generalization bounds for Rademacher complexities, and they tend to be sharper than the ones by combinatorial concepts like shattering coefficients. They typically look like this:

Theorem 38 (Rademacher generalization bound)

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

$$R(f) \leq R_n(f) + 2 \text{Rad}_n(\mathcal{F}) + \sqrt{\frac{\log(1/\delta)}{2n}}$$

Proofs are beyond the scope of this lecture.

## Rademacher complexity (3)

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

# Generalization bounds: conclusions

- ▶ Generalization bounds are a tool to answer the question whether a learning algorithm is consistent.
- ▶ Consistency refers to the estimation error, not the approximation error.
- ▶ Typically, generalization bounds have the following form:  
*With probability at least  $1 - \delta$ , for all  $f \in \mathcal{F}$*

$$R(f) \leq R_n(f) + \text{capacity term} + \text{confidence term}$$

The capacity term measures the size of the function class.  
The confidence term deals with how certain we are about our statement.

- ▶ There are many different ways to measure the capacity of function classes, we just scratched the surface.

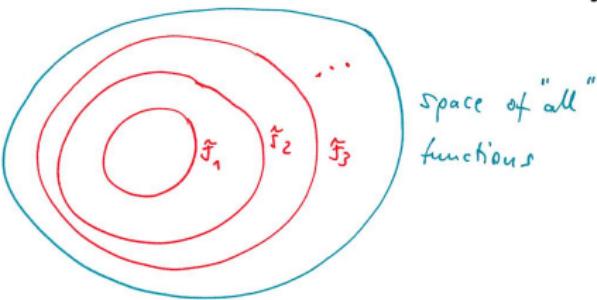
## Generalization bounds: conclusions (2)

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

# Controlling the approximation error

# Nested function classes

- ▶ So far, we always fixed a function class  $\mathcal{F}$  and investigated whether the estimation error in this class vanishes as we get to see more data.
- ▶ However, we need to take into account the approximation error as well.
- ▶ Idea is now: consider function classes that slowly grow with  $n$ :



# Nested function classes (2)

- ▶ If we have few data, the class is supposed to be small to avoid overfitting (generalization bound!)
- ▶ Eventually, when we see enough data, we can afford a larger function class without overfitting. The larger the class, the smaller our approximation error.

There are two major approaches to this:

- ▶ Structural risk minimization: explicit approach
- ▶ Regularization: implicit approach

# Structural risk minimization (SRM)

- ▶ Consider a nested sequence of function spaces:  $\mathcal{F}_1 \subset \mathcal{F}_2 \subset \dots$
- ▶ We now select an appropriate function class and a good function in this class simultaneously:

$$f_n := \operatorname{argmin}_{m \in \mathbb{N}, f \in \mathcal{F}_m} R_n(f) + \text{capacity term}(\mathcal{F}_m)$$

- ▶ The capacity term is the one that comes from a generalization bound.
- ▶ If the nested function classes approximate the space of “all” functions, one can prove that such an approach can lead to universal consistency.

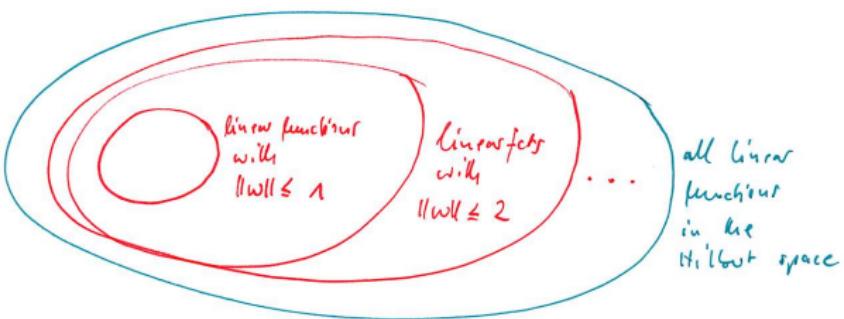
# Regularization

Recap: regularized risk minimization:

$$\text{minimize } R_n(f) + \lambda \cdot \Omega(f)$$

where  $\Omega$  punishes “complex” functions.

The trick is now: Regularization is an implicit way of performing structural risk minimization.



# Regularization (2)

Proving consistency for regularization is technical but very elegant:

- ▶ Make sure that your overall space of functions  $\mathcal{F}$  is dense in the space of continuous functions  
Example: linear combinations of a universal kernel.
- ▶ Consider a sequence of regularization constants  $\lambda_n$  with  $\lambda_n \rightarrow 0$  as  $n \rightarrow \infty$ .
- ▶ Define function class  $\mathcal{F}_n := \{f \in \mathcal{F} ; \lambda_n \cdot \Omega(f) \leq \text{const}\}$
- ▶ Choose  $\lambda_n \rightarrow 0$  so slow that  $\log \mathcal{N}(\mathcal{F}_n, n)/n \rightarrow 0$ .
  - ▶ On the one hand, this ensures that in the limit we won't overfit, the estimation error goes to 0.
  - ▶ On the other hand, if  $\lambda_n \rightarrow 0$ , then  $\mathcal{F}_n \rightarrow \mathcal{F}$  because  $\lambda \Omega(f) < c \implies \Omega(f) \leq c/\lambda_n \rightarrow \infty$ .  
Hence, the approximation error goes to 0, so we won't underfit.

## Regularization (3)

If you want to see the mathematical details, I recommend the following paper:

Steinwart: Support Vector Machines Are Universally Consistent.  
Journal of Complexity, 2002.

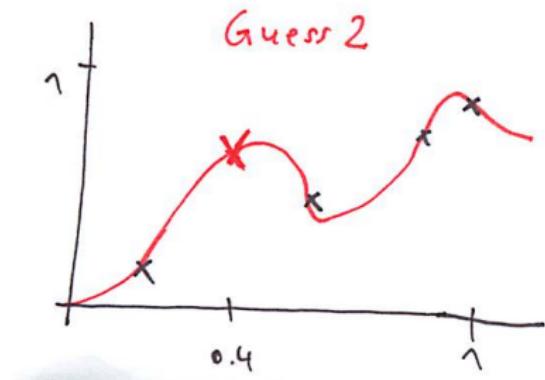
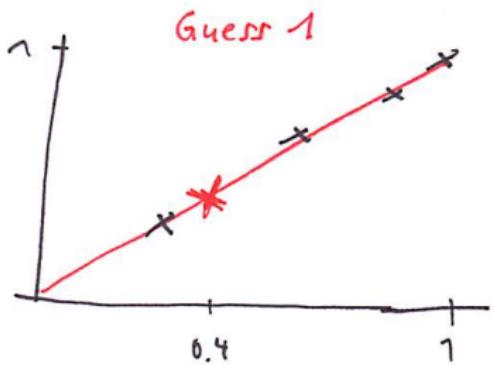
## Brief history

- ▶ The first proof that there exists a learning algorithm that is universally Bayes consistent was the Theorem of Stone 1977, about the kNN classifier.
- ▶ The combinatorial tools and generalization bounds have essentially been developed in the early 1970ies already (Vapnik, Chervonenkis, 1971, 1972, etc) and refined in the years around 2000.
- ▶ The statistics community also proved many results, in particular rates of convergence. There the focus is more on regression rather than classification.
- ▶ By and large, the theory is well understood by now, the focus of attention moved to different areas of machine learning theory (for example, online learning, unsupervised learning, etc).

## Getting back to Occam's razor

# Examples revisited

Remember the examples we discussed in the first lecture?



The question was which of the two functions should be preferred.

Many of you had argued that unless we have a strong belief that the right curve is correct, we should prefer the left one due to "simplicity".

## Examples revisited (2)

This principle is often called “Occam’s razor” or “principle of parsimony”:

*When we choose from a set of otherwise equivalent models, the simpler model should be preferred.*

Intuitive argument:

“Occam’s razor helps us to shave off those concepts, variables or constructs that are not really needed to explain the phenomenon. By doing that, developing the model will become easier, and there is less chance of introducing inconsistencies, ambiguities and redundancies. “

These formulations can be found in many papers and text books, I don’t know the original source ...

# Occam's razor vs. learning theory

However:

- ▶ The main message of learning theory was that we need to control the size of the function class  $\mathcal{F}$ .
- ▶ We had not at all talked about “simplicity” of functions!

Is this a contradiction? Is Occam's razor wrong???

# Occam's razor vs. learning theory (2)

First point of view: we don't need "simplicity":

- ▶ Consider an example of a function class that just contains 10 functions, all of which are very "complicated" (not "simple").
- ▶ For the estimation error, this would be great, we would soon be able to detect which of the function minimizes the ERM, with high probability.
- ▶ If the function class also happens to be able to describe the underlying phenomenon (low approximation error), this would be perfect.
- ▶ In this case, we do not need simple functions!!!

## Occam's razor vs. learning theory (3)

Second point of view: **Spaces of simple functions tend to be small.**

Example: Polynomials in one variable, with a discrete set of coefficients:

$$f(x) = \sum_{k=1}^d a_k x^k \text{ with } a_k \in \{-1, -0.99, -0.98, \dots, 0.98, 0.99, 1\}$$

There are about 200 polynomials of degree 1,  
 $200^2$  polynomials of degree 2,  
 $200^d$  polynomials of degree d.

Here, the spaces get larger the more “parameters” we have.

## Occam's razor vs. learning theory (4)

Both points of view come together if we talk about data compression.

- ▶ A space with few functions can be represented with few bits (say, by a small lookup table).
- ▶ A space with “simple” functions can be represented with few bits as well (encode all the parameters).
- ▶ A space of “complex” function cannot be compressed.

Intuitive conclusion:

- ▶ Spaces of simple functions are small, spaces of complex functions tend to be large.
- ▶ Learning theory tells us that we should prefer small function spaces.
- ▶ This often leads to spaces of simple functions.

## Occam's razor vs. learning theory (5)

This intuition can be made rigorous and formal:

- ▶ Sample compression bounds in statistical learning theory
- ▶ The whole branch of learning based on the “Minimum description length principle” (comprehensive book in this area by Peter Grünwald)

# Occam's razor vs. learning theory (6)

Bottom line:

- ▶ The quantity that is important is not so much the simplicity of the functions but rather the size of the function space.
- ▶ But spaces of simple functions tend to be small and are good candidates for learning.
- ▶ Occam's razor slightly misses the point, but is a good first proxy. It is not always correct, but often...

(\*) Loss functions, proper and surrogate losses:  
SKIPPED

(\*) Probabilistic interpretation of ERM:  
SKIPPED

## (\*) The No-Free-Lunch Theorem: SKIPPED

### Literature:

- ▶ The way I present it is taken from the paper: Ho and Pepyne. Simple explanation of the no-free lunch theorem and its implications. *Journal of Optimization Theory and Applications*, 2002.
- ▶ The book by Shalev-Shwartz and Ben-David discusses the NFL as well in Sec. 5.1, albeit with a different formulation.
- ▶ More general version can be found in Chapter 7 in Devroye/Györfi / Lugosi.

# Intuition

- ▶ Intuitively the no free lunch theorem (NFL) says that there does not exist a single best classifier that outperforms any other classifier on all learning problems.
- ▶ There exist many different versions to state this formally, below we describe the easiest one.

# NFL, simple version

- ▶ Assume that the space of all input points just consists of a finite set  $\mathcal{X} = \{x_1, \dots, x_m\}$ . Assume that the marginal distribution over these points is the uniform one (that is, each value  $x_i$  is equally likely).
- ▶ Assume that we consider binary classification, that is  $\mathcal{Y} = \{\pm 1\}$ , and that the labels are deterministic functions of the input.
- ▶ Particularly, there exists some function  $f : \mathcal{X} \rightarrow \mathcal{Y}$  that does not make any error.

# NFL, simple version (2)

Now consider the following table:

- Rows correspond to all possible true functions (there are  $2^m$  such functions)
- Columns correspond to all possible estimated functions
- The entries  $r_{ij}$  give the true error of function  $f_i$  when the true function is  $f_j$ .

| estimated \ true | $f_1$ | $f_2$ | $f_3$ | $\dots$ | $f_{2^m}$ |
|------------------|-------|-------|-------|---------|-----------|
| $f_1$            |       |       |       |         |           |
| $f_2$            |       |       |       |         |           |
| $f_3$            |       |       |       |         |           |
| $\vdots$         |       |       |       |         |           |
| $f_{2^m}$        |       |       |       |         |           |

$r_{ij}$

## NFL, simple version (3)

Proposition 39 (Risk in each row is the same)

In each row of the table, each risk value occurs the same number of times.

Proof.

- ▶  $r_{ij} = 0$  exactly once (if  $f_i = f_j$ )
- ▶  $r_{ij} = 1/m$  exactly  $m$  times
- ▶  $r_{ij} = 2/m$  exactly  $\binom{m}{2}$  times
- ▶ ...



# NFL, simple version (4)

## Proposition 40 (Simple NFL)

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

Proof. Obvious consequence of the previous proposition.



## NFL, simple version (5)

### Proposition 41 (Simple NFL with training data)

In the model introduced above: Assume we are given a training set  $(X_i, Y_i)_{i=1,\dots,n}$ . Then, on average over all test distributions, all classifiers that are consistent with the training set perform the same.

Proof.

- ▶ In the table above, eliminate all columns that are not consistent with the training data.
- ▶ Among the remaining ones, all distributions over test labels are possible.
- ▶ Then the result follows by a similar argument as above. ☺

# NFL, simple version (6)

Note that much more general theorems exist, for example for the standard machine learning scenario where we draw data from joint distribution  $P$  on  $\mathcal{X} \times \mathcal{Y}$  and  $\mathcal{X}$  is any space you want ...

# NFL, simple version (7)

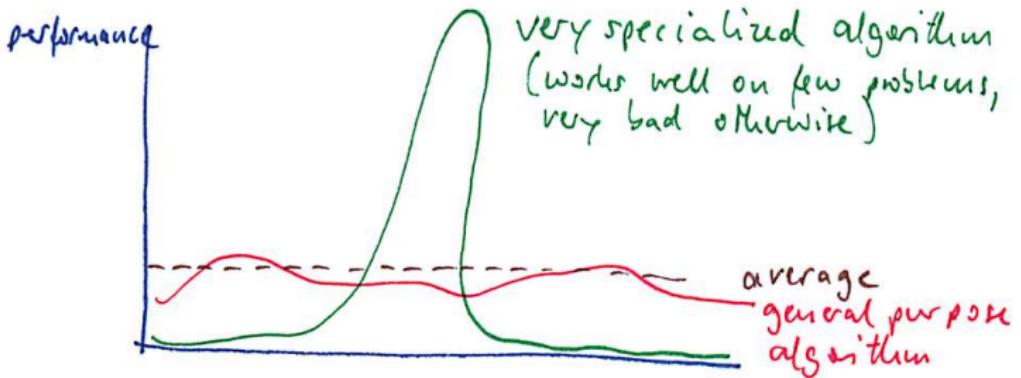
Discussion:

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

## NFL, simple version (8)

- ▶ No: the key is that in practice we do not see “all possible data sets”. As soon as we make assumptions on the data sets, the NFL breaks down (“Making assumptions” means to delete some columns from the above matrix, and then the proof breaks down).
- ▶ This shows once more how important it is to incorporate these assumptions to the machine learning algorithm  $\leadsto$  inductive bias!

# NFL, simple version (9)



all learning problems

the integral over all these curves is exactly  
the same by NFL.

# History / Literature

- ▶ Wolpert, David. *The Lack of A Priori Distinctions between Learning Algorithms*. *Neural Computation*, 1996.
- ▶ Many generalizations since then (both to the field of machine learning and optimization in general).
- ▶ Ho and Pepyne. *Simple explanation of the no-free lunch theorem and its implications*. *Journal of Optimization Theory and Applications*, 2002.
- ▶ Chapter 7 in Devroye/ Györfi / Lugosi

# Low rank matrix methods

# Introduction: recommender systems, collaborative filtering

# Recommender systems

Goal: give recommendations to users, based on their past behavior:

- ▶ Recommend movies (e.g., netflix)
- ▶ recommend music (e.g., lastfm)
- ▶ recommend products to buy (e.g., amazon)

ANY IDEAS HOW WE COULD DO THIS?

# Recommender systems (2)

Content-based approach:

- ▶ Model products based on **explicit features**. Use these features to define a similarity function between products
- ▶ If a user likes product A, then recommend products similar to A.

Prominent example: Pandora Radio. You start with a song you like, and then Pandora plays similar songs.

# Recommender systems (3)

Collaborative approach:

- ▶ Forget about explicitly modeling users or features.
- ▶ Instead, **implicitly** model similarity of users and products based on past shopping behavior.
- ▶ Consider user/product matrix with ratings. Defines an implicit similarity between users (or products).
- ▶ Then recommend similar items to similar users.

Prominent example: lastfm

ADVANTAGES / DISADVANTAGES OF THE TWO?

# Matrix factorization basics

Hastie, Tibshirani, Wainwright: Statistical learning with sparsity.  
2015. Chapter 7.2

# Recap: singular value decomposition (SVD)

Recall PCA:

- ▶ Eigenvalue decomposition for a symmetric matrix
- ▶ Best rank- $k$  approximation of the matrix: based on highest  $k$  eigenvalues

Now want to do something more general for arbitrary (non-square) matrices.

# Recap: singular value decomposition (SVD) (2)

Every (!) matrix can be decomposed as follows:

$$\begin{matrix} d \\ n \end{matrix} = \begin{matrix} n \\ n \end{matrix} \cdot \begin{matrix} n \\ d \end{matrix} \cdot \begin{matrix} d \\ d \end{matrix}$$

left singular vectors      singular values      right singular vectors

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

$U$  is the matrix of left singular vectors,  $V$  the right singular vectors, and the diagonal of  $\Sigma$  contains the singular values.

## Recap: singular value decomposition (SVD) (3)

There is a simple relationship between SVD and PCA:

For any matrix  $A$ ,

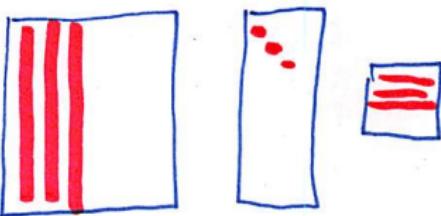
- ▶ the left singular vectors are the eigenvectors of  $AA'$
- ▶ the right singular vectors are the eigenvectors of  $A'A$
- ▶ the non-zero singular values are the square roots of the eigenvalues of both  $AA'$  and  $A'A$ .

PROOFS: EXERCISE!

# SVD for rank-k approximation

Consider the following “Top-k-SVD” procedure:

- ▶ Given a  $n \times d$  matrix  $A$ .
- ▶ Compute the SVD such that the singular values are sorted in decreasing order.
- ▶ Keep the first  $k$  columns of  $U$  and  $V$ . Call the resulting matrices  $U_k \in \mathbb{R}^{n \times k}$  and  $V_k \in \mathbb{R}^{d \times k}$  (such that  $V'_k \in \mathbb{R}^{k \times d}$ ).
- ▶ Keep the singular values  $\sigma_1, \dots, \sigma_k$  and write them in a diagonal matrix  $\Sigma_k \in \mathbb{R}^{k \times k}$ .
- ▶ Now define  $A_k := U_k \Sigma_k V'_k$ .



# SVD for rank- $k$ approximation (2)

Intuitive interpretation:

- ▶ Assume that  $A$  is a matrix recording ratings of  $n$  users about  $d$  products.
- ▶ Then the top- $k$  right singular vectors can be interpreted as basic “customer types”. Each customer is a weighted mixture of the basic customers.
- ▶ Similarly, the top- $k$  left singular vectors can be interpreted as basic “product types”.

## SVD for rank-k approximation (3)

The Frobenius norm of matrix is defined as  $\|B\|_F := \sqrt{\sum_{ij} b_{ij}^2}$ .

### Theorem 42 (SVD as rank-k approximation)

The matrix  $A_k$  defined by the first  $k$  singular values/vectors solves the following rank- $k$ -approximation problem:

Given  $A$  and  $k$ , find the matrix  $A_k$  with rank at most  $k$  such that  $\|A - A_k\|_F$  is minimized.

Proof: EXERCISE (consider the hints in Exercise 7.2. p. 196 in “statistical learning with sparsity”).

EXERCISE: COMPARE THIS RESULT WITH THE CORRESPONDING RESULT FOR PCA!

## SVD for rank- $k$ approximation (4)

Digest again what the intuitive interpretation is:

- ▶ We know the full product / user matrix.
- ▶ The  $k$  top singular vectors define  $k$  types of users / products.
- ▶ Based on these few types, we can “explain” the behavior of everybody (up to a small approximation error).

# Low rank matrix completion

## Literature

- Hastie, Tibshirani, Wainwright: Statistical learning with sparsity. 2015. Chapter 7

Some important orginal papers:

- Candes, Recht: Exact matrix completion via convex optimization. Foundations of computational mathematics, 2009.
- Matrix Completion from a Few Entries R. Keshavan, Andrea Montanari, and Sewoong Oh: Matrix completion from noisy entries. JMLR, 2010
- Mazumder, Hastie, Tibshirani: Spectral regularization algorithms for learning large incomplete matrices. JMLR, 2010.

# Netflix problem

General problem:

- ▶ Consider a huge matrix of user ratings of movies. Rows correspond to movies, columns correspond to users, entries are ratings on a scale from 1 to 5.
- ▶ We only know few entries in this matrix.
- ▶ The matrix completion problem is to estimate the missing entries in order to recommend new movies to a user.

## Netflix problem (2)

History of the Netflix challenge:

- ▶ Launched in 2006
- ▶ Data: About 20.000 movies, 500.000 users,  $10^9$  ratings (that is, about 1% of the entries are known)
- ▶ Goal: predict the missing entries, error measure RMSE (root mean squared error  $\sqrt{\sum_{i=1}^n (z_i - \hat{z}_i)^2 / n}$ )
- ▶ First team that beats Netflix's own algorithm by an improvement of at least 10% wins a prize of 1 million dollars.
- ▶ Was finally achieved in 2009.

# Matrix completion problem

General setup:

- ▶ Consider an  $m \times n$  matrix which is unknown.
- ▶ We get to see some entries in the matrix.
- ▶ Assume that the position of the revealed entries is random (no adversarial setting).
- ▶ Goal is to estimate the unknown entries as well as possible.

CAN YOU THINK OF EASY / DIFFICULT CASES? IS IT ALWAYS POSSIBLE?

## Matrix completion problem (2)

We need to make assumptions to be able to solve this problem (inductive bias!). If the entries are not related to each other (say, independent random numbers), there is no way in which we could predict missing entries.

## Matrix completion problem (3)

High-level idea from learning theory: A useful inductive bias is one that leads to a “small” set of possible matrices.

Here is what everybody uses:

We are going to look for a matrix that has low rank.

(Just as a sanity check: a matrix with independent random entries typically has high rank, the eigenvalues follow the semi-circle law.)

# Matrix completion, first formulations

Denote by  $\Omega$  the set of entries of a matrix  $Z$  that have been observed: we know the values  $z_{ij}$  for all  $(i, j) \in \Omega$ . We would like to solve the following problem:

$$\text{minimize} \operatorname{rank}(M) \text{ subject to } m_{ij} = z_{ij} \text{ for } (i, j) \in \Omega.$$

or a slightly weaker version

$$\text{minimize} \operatorname{rank}(M) \text{ subject to } \sum_{(i,j) \in \Omega} (m_{ij} - z_{ij})^2 \leq \delta$$

or the regularization version

$$\text{minimize} \sum_{(i,j) \in \Omega} (m_{ij} - z_{ij})^2 + \lambda \operatorname{rank}(M)$$

Is NP hard ☺

CAN YOU SEE WHAT MAKES IT SO DIFFICULT?

# Matrix completion, first approach using SVD

Here is a straight forward heuristic by which we can try to solve the optimization problem:

## *Hard-Impute*

- ▶ Have an initial guess for the missing entries  $\leadsto$  matrix  $Z_1$
- ▶ Compute the SVD of  $Z_1$ , keep the first  $r$  singular components  $\leadsto Z_2$
- ▶ Fill in the missing entries with the ones of  $Z_2$ , and start over again ...

Sometimes this works reasonably.

But let's try to think about alternatives ... one option for non-convex optimization problems is always to construct a convex relaxation (have seen this before, at least twice, where? )

# Trace as convex relaxation of rank

Let us try to find a convex relaxation of the rank function:

- If  $\sigma := \sigma(A)$  denotes the vector of singular values of matrix  $A$ , then

$$\text{rank}(A) = \|\sigma\|_0$$

- Recall the standard approach in sparse regression (Lasso). We relaxed the 0-norm to the 1-norm, which is convex:

$$\|\sigma\|_1 = \sum_i |\sigma_i|.$$

We now use this as a norm for matrices, it is called the nuclear norm or the trace norm:

$$\|A\|_{tr} := \|\sigma(A)\|_1.$$

One can prove that the nuclear norm is the tightest convex relaxation of the rank of a matrix.

# Trace norm regularization

Now consider the following optimization problems:

$$\text{minimize } \|M\|_{tr} \text{ subject to } \sum_{(i,j) \in \Omega} (m_{ij} - z_{ij})^2 \leq \delta \quad (*)$$

$$\text{minimize } \frac{1}{2} \sum_{(i,j) \in \Omega} (m_{ij} - z_{ij})^2 + \lambda \|M\|_{tr} \quad (**)$$

These two problems are essentially the same, once in the natural formulation (\*) and once in the regularization / Lagrangian formulation (\*\*).

# Trace norm regularization (2)

Two big questions:

- ▶ Can we give an efficient algorithm that can find the global optimum, either in formulation (\*) or in (\*\*)?
- ▶ If yes, what can we say about the theoretical properties of the global optimum, how close is it going to be to the matrix we are looking for? In particular, **how many entries do we need to observe to find a good reconstruction?**

## Solving (\*), naive algorithm: semi-definite program

The first formulation of the problem is a **semi-definite program**. In principle, SDPs can be solved in polynomial time, but “polynomial” can still be very long... There are general-purpose solvers for such problems, but they are so slow that they only work for small instances.

We skip the details.

# Solving $(\ast\ast)$ efficiently: soft-impute

- ▶ Start with initial guesses for the missing values.
- ▶ Compute the SVD, “soft-threshold” the singular values by some threshold  $\lambda$
- ▶ Repeat until convergence.

Let's look at the details:

# Solving (\*\*) efficiently: soft-impute (2)

Soft-thresholding:

- ▶ Given the SVD of a matrix  $Z = UDV'$ , denote the singular values by  $d_i$ .
- ▶ We define  $S_\lambda(Z) := U D_\lambda V'$  where  $D_\lambda$  is the diagonal matrix with diagonal entries  $(d_i - \lambda)_+ := \max(d_i - \lambda, 0)$
- ▶ Soft thresholding decreases the trace norm and also often decreases the rank of a matrix.

## Proposition 43

The solution of the optimization problem

$$\min_M \|Z - M\|_F^2 + \lambda \|Z\|_{tr}$$

is given by  $S_\lambda(Z)$ .

## Solving (\*\* ) efficiently: soft-impute (3)

(Note that in this theorem,  $Z$  is known completely, there are no missing entries yet).

Proof: see Mazumder, Hastie, Tibshirani: Spectral regularization algorithms for learning large incomplete matrices. JMLR, 2010.

## Solving (\*\* ) efficiently: soft-impute (4)

Now we want to use a similar approach to complete the matrix  $Z$  in case it is just partially observed (problem (\*\*)).

Introduce notation:

- ▶ Denote by  $\Omega$  the set of matrix entries that are known.
- ▶ Define the “projection”  $P_\Omega(Z)$  as the matrix that has the original values  $z_{ij}$  at all the observed positions of  $Z$ , and 0 otherwise (that is, fill the unobserved entries with zeros).
- ▶ With this definition,  
$$\sum_{(i,j) \in \Omega} (z_{ij} - m_{ij})^2 = \|P_\Omega(Z) - P_\Omega(M)\|_F.$$
- ▶ Define  $P_\Omega^\perp(Z)$  as the “projection” of the matrix  $Z$  on the entries that are NOT in  $\Omega$  (so that  $Z = P_\Omega(Z) + P_\Omega^\perp(Z)$ ).

# Solving (\*\*) efficiently: soft-impute (5)

---

**Algorithm 1** SOFT-IMPUTE
 

---

1. Initialize  $Z^{\text{old}} = 0$ .
2. Do for  $\lambda_1 > \lambda_2 > \dots > \lambda_K$ :
  - (a) Repeat:
    - i. Compute  $Z^{\text{new}} \leftarrow \mathbf{S}_{\lambda_k}(P_{\Omega}(X) + P_{\Omega}^{\perp}(Z^{\text{old}}))$ .
    - ii. If  $\frac{\|Z^{\text{new}} - Z^{\text{old}}\|_F^2}{\|Z^{\text{old}}\|_F^2} < \varepsilon$  exit.
    - iii. Assign  $Z^{\text{old}} \leftarrow Z^{\text{new}}$ .
  - (b) Assign  $\hat{Z}_{\lambda_k} \leftarrow Z^{\text{new}}$ .
3. Output the sequence of solutions  $\hat{Z}_{\lambda_1}, \dots, \hat{Z}_{\lambda_K}$ .

Inuition:

- ▶ Inner loop (a) for fixed lambda: clamp the observed values of the matrix, fill the rest by a low-rank approximation, until convergence
- ▶ Outer loop (2): start with a case that is easy ( $\lambda_i$  large, matrix low rank) and work our way towards the more difficult situation of smaller  $\lambda_i$

# Solving (\*\* ) efficiently: soft-impute (6)

Properties:

- ▶ It can be proved that this algorithm always converges to the global solution of (\*\* ) (for a suitable choice of the sequence  $\lambda_k$ ).
- ▶ It can be implemented efficiently even for huge matrices (just a couple of hours for Netflix). Main trick: can decompose the dense matrix in a sum of a sparse matrix and a low rank matrix. Can exploit this cleverly in the algorithm.

$$\mathcal{P}_\Omega(\mathbf{Z}) + \mathcal{P}_{\Omega^\perp}^\perp(\mathbf{Z}^{\text{old}}) = \underbrace{\mathcal{P}_\Omega(\mathbf{Z}) - \mathcal{P}_\Omega(\mathbf{Z}^{\text{old}})}_{\text{sparse}} + \underbrace{\mathbf{Z}^{\text{old}}}_{\text{low rank}}$$

- ▶ R-package softImpute

Details: Mazumder, Hastie, Tibshirani

# Solving (\*\*) efficiently: soft-impute (7)

Soft-impute on the Netflix data:

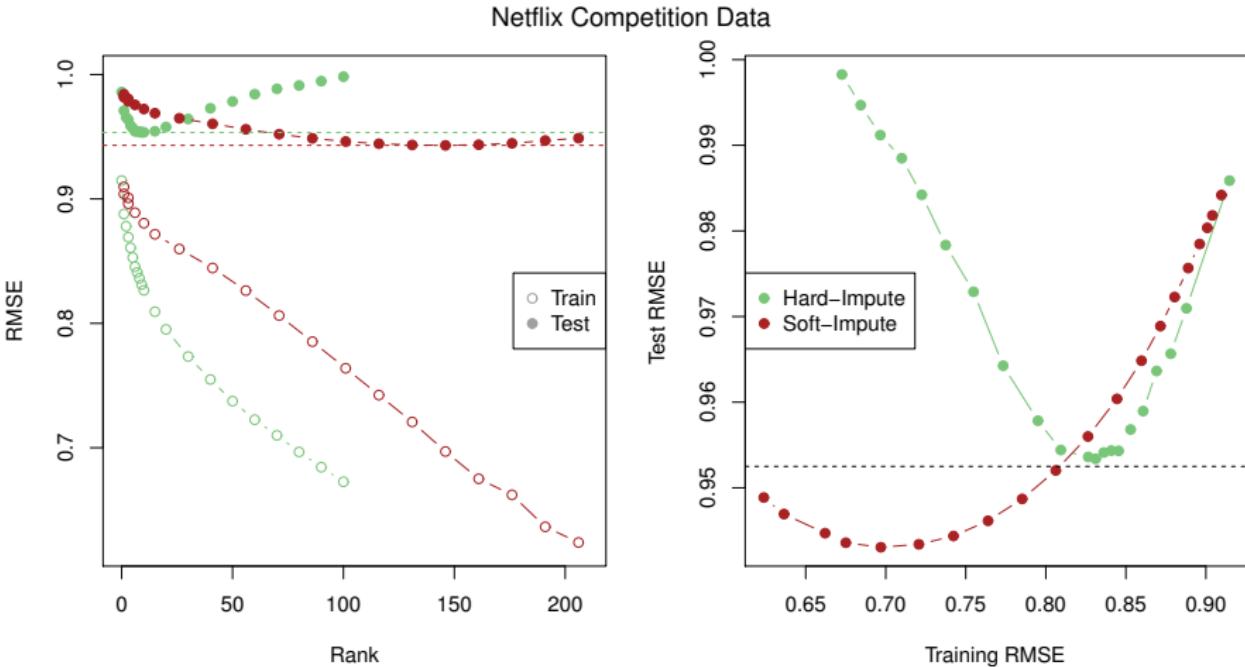


Figure from "Statistical learning with sparsity". Dotted line on rhs = Netflix's own algorithm, baseline.

Sanity check: random guessing would have RMSE≈2

# Theoretical results on matrix completion

Setting:

- ▶ Consider an arbitrary matrix  $M$ . For simplicity, assume that  $M$  is square of size  $p \times p$ .
- ▶ Assume that we observed  $n$  entries of the matrix, drawn uniformly at random.
- ▶ Question: how large does  $n$  need to be such that we can successfully reconstruct the matrix  $M$  (exact or approximately)?

# First observations

Problem of empty columns:

If there is a column (or a row) that does not have any observed entry, it will be impossible to reconstruct the matrix.

QUESTION TO YOU: We sample  $n$  elements. Each element belongs to one of  $p$  columns. How large does  $n$  need to be such that, with reasonably high probability, we have at least one element per column?

## First observations (2)

ANSWER: this is the coupon collector problem, we need at least  $p \log p$  samples.

## First observations (3)

Number of “parameters” for a rank- $r$  matrix:

- ▶ A rank- $r$  matrix of dimension  $p \times p$  can be described by  $r$  vectors of length  $p \sim rp$  parameters
- ▶ In general, we cannot assume that we can “compress” these  $rp$  entries any further
- ▶ So it is plausible that we won’t be able to perfectly reconstruct such a matrix if we observe less than  $rp$  entries of the matrix.

(of course, this argument is really hand-waving, but often such hand-waving intuition helps to get a first feeling for a problem; the next step is then to make it precise)

## More subtle observation: coherency

Consider the following matrix

$$\mathbf{Z} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Assume we want to solve EXACT recovery:

- ▶ It is of rank one (so the problem should be easy, few parameters to learn).
- ▶ However, there is only one important entry.
- ▶ If we sample of order less than of the order  $p^2$  entries, likelihood is high that we never get to see this particular entry. So if we are after exact recovery, we have a problem.

## More subtle observation: coherency (2)

The problem in this example:

- ▶ Some entries are much more important than others.
- ▶ In mathematical terms: the eigenvectors of this matrix are too much aligned with the standard basis of  $\mathbb{R}^p$ .

## More subtle observation: coherency (3)

To deal with this problem:

“measure” up to which extent the eigenvectors of the matrix are aligned with the standard basis: this leads to the notion of coherence.

**Definition:** Let  $U$  be a subspace of  $\mathbb{R}^d$  of dimension  $r$  and  $P_U$  the orthogonal projection on  $U$ . Then the coherence of  $U$  wrt to the standard basis  $(e_i)_i$  is defined as

$$\mu(U) := \frac{d}{r} \max_{i=1,\dots,d} \|P_U e_i\|^2$$

FOR MATRIX COMPLETION, IS IT BETTER TO HAVE SMALL OR LARGE COHERENCE?

## More subtle observation: coherency (4)

To get intuition, consider the case where  $U$  is spanned by one vector:

- ▶ The smaller  $\mu(U)$ , the “easier” the matrix will be to recover.
- ▶ Maximum value of coherence occurs if  $U = \text{span}(e_i)$  (more generally, if  $e_i \in U$ ). Then we have  $\|P_u e_i\| = 1$ .
- ▶ Minimum value of coherence occurs if  $U$  is spanned by the vector  $(1/\sqrt{n}, \dots, 1/\sqrt{n})'$ .
- ▶ Intuition: More generally, coherence is low (good) if all entries of the vector have about the same order of magnitude. **Then each entry contains about the same amount of information**, so sampling few entries should be fine.

## More subtle observation: coherency (5)

Examples:

The matrix we started with:

$$M = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & 0 & \\ 0 & & & \end{pmatrix}, \quad \sigma_1 = 1, \quad \sigma_2 = \dots = \sigma_n = 0, \quad v_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \Rightarrow \|P_{v_1} e_1\| = 1$$

$\Rightarrow$  maximal coherence

# More subtle observation: coherency (6)

Same matrix, just flipped entries:

$$M = \begin{pmatrix} 0 & 1 & \dots & 1 \\ 1 & 0 & \dots & 0 \\ \vdots & & \ddots & 0 \\ 1 & 0 & \dots & 0 \end{pmatrix}, \quad \sigma_1, \sigma_2 \geq 0, \sigma_3 \dots \sigma_n = 0$$

$$v_1 = \begin{pmatrix} 0.3 \\ 0.4 \\ 0.4 \\ 0.4 \\ 0.4 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 0.9 \\ -0.1 \\ -0.1 \\ -0.1 \\ -0.1 \end{pmatrix}, \quad V := \begin{pmatrix} v_1 & v_2 \end{pmatrix}$$

$$\|P_V e_1\| = 1$$

(because  $e_1 \in \text{span}\{v_1, v_2\}$ )  
 $\Rightarrow$  maximal coherence

As a sanity check, the all ones matrix:

$$M = \begin{pmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{pmatrix}, \quad \sigma_1 = n, \sigma_2 = \dots = \sigma_n = 0$$

$$v_1 = \frac{1}{\sqrt{n}} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \quad \|P_{V_1} e_i\| = \frac{1}{\sqrt{n}}$$

$\Rightarrow$  low coherence  
 (as small as it can be)

## More subtle observation: coherency (7)

A random rank-r matrix:

$$\mathbf{M} = \text{rand}(n, r), \quad \mathbf{M} = \mathbf{M} \cdot \mathbf{M}' \quad (\text{random rank-}r \text{ matrix})$$

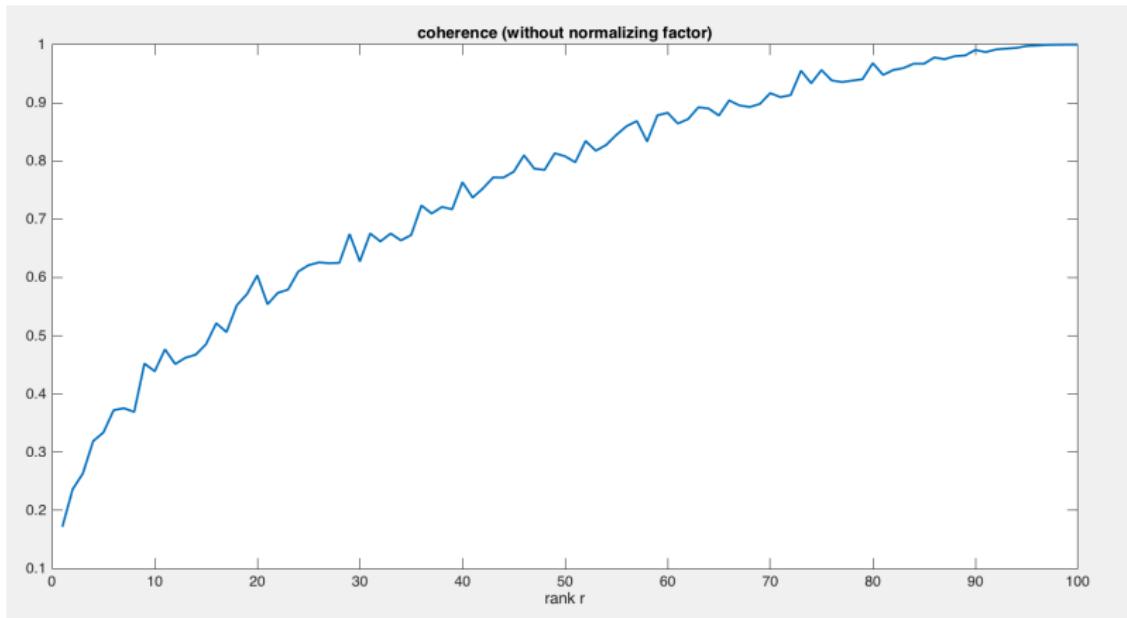
$$\sigma_1, \dots, \sigma_r > 0, \quad \sigma_{r+1} \dots \sigma_n = 0$$

- If  $r$  low, coherence low as well
- If  $r$  high, coherence gets higher (extreme case  
 $r=n$ : Then we project on the full space, hence  
coherence = 1)

See plot on next page:

## More subtle observation: coherency (8)

Plot of  $\max_{i=1 \dots n} \|P_V e_i\|$ , random rank-r matrix, n=100:



## Guarantee for exact recovery

One can prove guarantees of the following flavor ( $p$  size of matrix,  $r$  rank):

With high probability, exact recovery is possible if the number of observed entries is at least

$$N \geq Crp \log p$$

Here,  $C$  is a constant that depends on the coherence of the matrix:

- ▶ Coherence low:  $N \approx rp \log p$
- ▶ Coherence high:  $C \cdot r \approx p$ , such that we need to sample about  $p^2 \log p$  entries.

Proofs are beyond the scope of this lecture.

GIVEN OUR PREVIOUS OBSERVATIONS, DO YOU THINK THIS IS A GOOD OR A BAD RESULT?

# Guarantee for exact recovery (2)

Concretely, the coherency-based theorem for exact recovery looks as follows (Candes, 2009):

**Theorem 1.1** Let  $M$  be an  $n_1 \times n_2$  matrix of rank  $r$  with singular value decomposition  $U\Sigma V^*$ . Without loss of generality, impose the conventions  $n_1 \leq n_2$ ,  $\Sigma$  is  $r \times r$ ,  $U$  is  $n_1 \times r$  and  $V$  is  $n_2 \times r$ . Assume that

**A0** The row and column spaces have coherences bounded above by some positive  $\mu_0$ .

**A1** The matrix  $UV^*$  has a maximum entry bounded by  $\mu_1 \sqrt{r/(n_1 n_2)}$  in absolute value for some positive  $\mu_1$ .

Suppose  $m$  entries of  $M$  are observed with locations sampled uniformly at random. Then if

$$m \geq 32 \max\{\mu_1^2, \mu_0\} r(n_1 + n_2) \beta \log^2(2n_2) \quad (1.2)$$

for some  $\beta > 1$ , the minimizer to the problem

$$\begin{array}{ll} \text{minimize} & \|X\|_* \\ \text{subject to} & X_{ij} = M_{ij} \quad (i, j) \in \Omega. \end{array} \quad (1.3)$$

is unique and equal to  $M$  with probability at least  $1 - 6 \log(n_2)(n_1 + n_2)^{2-2\beta} - n_2^{2-2\beta^{1/2}}$ .

(here  $\|\cdot\|_*$  denotes the trace norm).

# Guarantee for approximate recovery

Guarantee from Keshavan et al, 2010:

*low rank matrix*      *matrix with independent noise*      *number of observed entries*

**Theorem 1.2** Let  $N = M + Z$ , where  $M$  is a  $(\mu_0, \mu_1)$ -incoherent matrix of rank  $r$ , and assume that the subset of revealed entries  $E \subseteq [m] \times [n]$  is uniformly random with size  $|E|$ . Further, let  $\Sigma_{\min} = \Sigma_r \leq \dots \leq \Sigma_1 = \Sigma_{\max}$  with  $\Sigma_{\max}/\Sigma_{\min} \equiv \kappa$ . Let  $\hat{M}$  be the output of OPTSPACE on input  $N^E$ . Then there exists numerical constants  $C$  and  $C'$  such that if

$$|E| \geq Cn\sqrt{\alpha}\kappa^2 \max\{\mu_0 r\sqrt{\alpha} \log n; \mu_0^2 r^2 \alpha \kappa^4; \mu_1^2 r^2 \alpha \kappa^4\}, \approx n^{r^2 \log n} \cdot n$$

then, with probability at least  $1 - 1/n^3$ ,

$$\frac{1}{\sqrt{mn}} \underbrace{\|\hat{M} - M\|_F}_{\text{error of recovery}} \leq C' \kappa^2 \frac{n\sqrt{r\alpha}}{|E|} \underbrace{\|Z^E\|_2}_{\text{amount of noise}}. \quad (3)$$

provided that the right-hand side is smaller than  $\Sigma_{\min}$ .

$\Sigma_i$ : singular values of  $M$

# Simulations: noise-free setting

Assume you want to run simulations for low rank matrix completion, under controlled conditions.

HOW COULD YOU GENERATE A “RANDOM” LOW RANK MATRIX TO PLAY WITH?

## Simulations: noise-free setting (2)

- ▶ Model the ground truth by a simple low rank model:
  - ▶ Generate the  $p \times r$  matrices  $U$  and  $V$  with independent random entries, normally distributed according to  $N(0, 1)$  (in the figures below:  $p = 20$  or  $40$ , and  $r = 1$  or  $5$ . )
  - ▶ Define  $Z = U \cdot V'$ .  
(WHAT IS THE RANK OF THESE MATRICES? WHAT CAN YOU SAY ABOUT THE SINGULAR VALUES?)
- ▶ Generate the toy data: Sample  $n$  random entries from this matrix.
- ▶ Try to recover the ground truth:
  - ▶ Use soft-impute to complete the matrices, results in  $\hat{Z}$ .
  - ▶ Check whether  $\|\hat{Z} - Z\| \approx 0$
  - ▶ Repeat this experiment many times and report fraction of correctly recovered matrices.

# Simulations: noise-free setting (3)

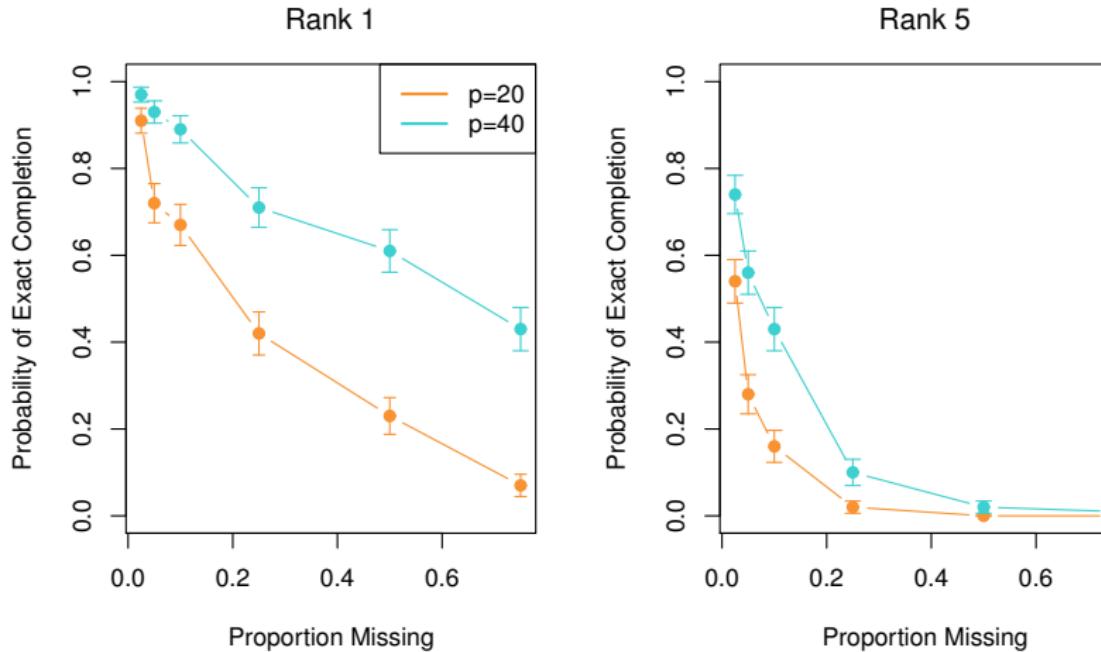


Figure from "Statistical learning with sparsity"

WHAT CAN YOU SEE?

# Simulations: noisy setting

Noisy setting:

- ▶ Generate the matrix  $Z$  as before.
- ▶ Now add Gaussian noise with standard deviation 0.5 to each of the entries of  $Z$ , this results in  $Z_{noisy}$ .
- ▶ Now try to reconstruct  $Z$ , when you observe entries from  $Z_{noisy}$  (using Soft-impute).
- ▶ Plot

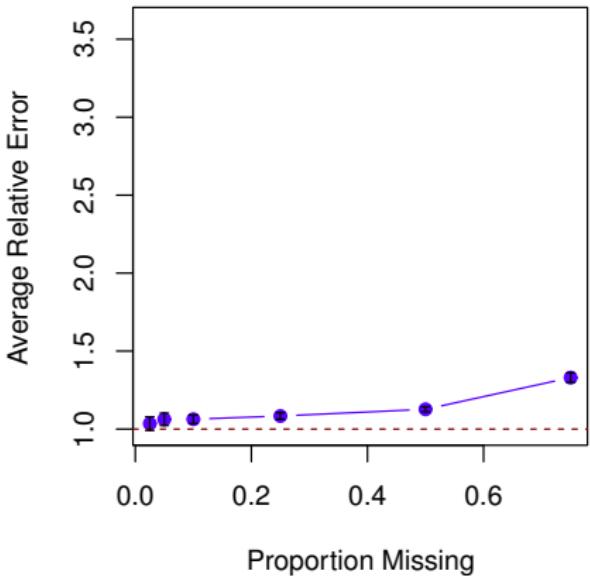
$$\text{average relative error} := \frac{\text{average Frobenius norm error}}{\text{noise standard deviation}}$$

(WHAT IS THE BEST AVERAGE RELATIVE ERROR YOU COULD HOPE FOR? )

Results:

# Simulations: noisy setting (2)

Rank 1



Rank 5

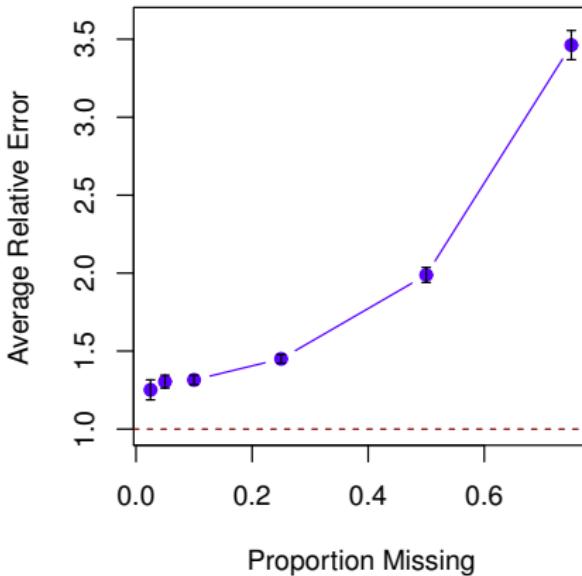


Figure from "Statistical learning with sparsity"

# Outlook / literature

- ▶ We just scratched the surface, there are many more variants of the problem, and also many more algorithms.
- ▶ If you are interested, the book “Statistical learning with sparsity” is a good starting point.

## History:

- ▶ PhD thesis Fazel 2002: nuclear norm as surrogate for rank
- ▶ Nati Srebro et al, 2005, nuclear norm relaxations, with first generalization bounds.
- ▶ Candes, Recht: Exact matrix completion via convex optimization. Foundations of computational mathematics (FOCM), 2009. Bounds in exact case.
- ▶ Netflix challenge: 2006 - 2009.

# Compressed sensing

Book chapters:

- Hastie, Tibshirani, Wainwright: Statistical learning with sparsity. 2015. Chapter 10.
- Shalev-Shwartz, Ben-David: Understanding Machine Learning, Section 23.3.

# Motivation

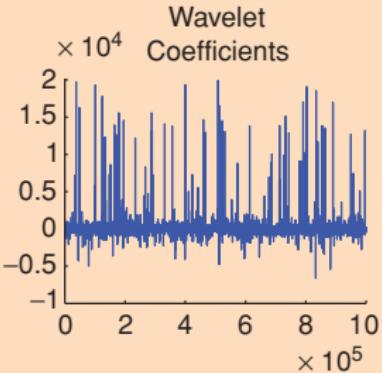
Consider the camera in your phone:

- ▶ If you take a picture, it first generates a raw image that stores the image in a standard pixel-based representation (e.g., rgb values for each pixel).
- ▶ Then it compresses the picture by representing it in a suitable basis (say, a wavelet basis) and generates a compressed version of the image (say, a jpg file).

# Motivation (2)



(a)



(b)



(c)

**Figure 1.2** (a) Original megapixel image with pixel values in the range  $[0, 255]$  and (b) its wavelet transform coefficients (arranged in random order for enhanced visibility). Relatively few wavelet coefficients capture most of the signal energy; many such images are highly compressible. (c) The reconstruction obtained by zeroing out all the coefficients in the wavelet expansion but the 25,000 largest (pixel values are thresholded to the range  $[0, 255]$ ). The differences from the original picture are hardly noticeable.

## Motivation (3)

Idea: it would be great if we could skip the first step and directly capture the data in the better representation.

This is called compressed (compressive) sensing.

Applications:

- ▶ Cameras with little power / storage. Take a picture with less pixels, but achieve the same quality in the end.
- ▶ MRI / tomography: scans parts of the body, scanning time is proportional to the number of measurements. Want to speed up scanning (take less pictures) but still have the same quality.

# Setup

Assume we observe a vector  $x \in \mathbb{R}^d$ .

- ▶ Typically it is not be sparse in the standard basis, that is  $\|x\|_0$  is close to  $d$
- ▶ But it might be sparse in a different basis: There exists an orthonormal matrix  $U$  such that  $x = U\alpha$  and  $\alpha$  is a sparse vector:  $\|\alpha\|_0 =: s$  is small
- ▶ If we would know the basis  $U$  and would have a technical way to measure the signal in this basis directly, this would be great.
- ▶ Goal is now: construct a basis that does the job.

## Setup (2)

Notation in the following:

- ▶  $d$  dimension of the original space (high)
- ▶  $s$  true sparsity of the signal in the basis  $U$  (low)
- ▶  $k$  the sparsity we actually achieve (hopefully low as well)

We always have  $s \leq k \leq d$ .

## Key steps in compressed sensing

1. We design  $k$  “linear measurements”  $w_1, \dots, w_k \in \mathbb{R}^d$  (in applications, this is done by specific hardware, see later).
2. Nature picks an unknown signal  $x \in \mathbb{R}^d$  ( $d$  large)
3. We directly receive the measurement results  
 $\tilde{x}_1 = \langle w_1, x \rangle$ ,  $\tilde{x}_2 = \langle w_2, x \rangle$ , ..., resulting in the measurement vector  $\tilde{x} \in \mathbb{R}^k$  ( $k$  reasonably small).
4. We are now supposed to reconstruct  $x \in \mathbb{R}^d$  from  $\tilde{x} \in \mathbb{R}^k$ .

Note: The goal is to design a single (!) set of measurement vectors  $w_1, \dots, w_k$  that works well for all (!) signals  $x$  in the sense that we are able to reconstruct with little error.

## Example: Single pixel camera

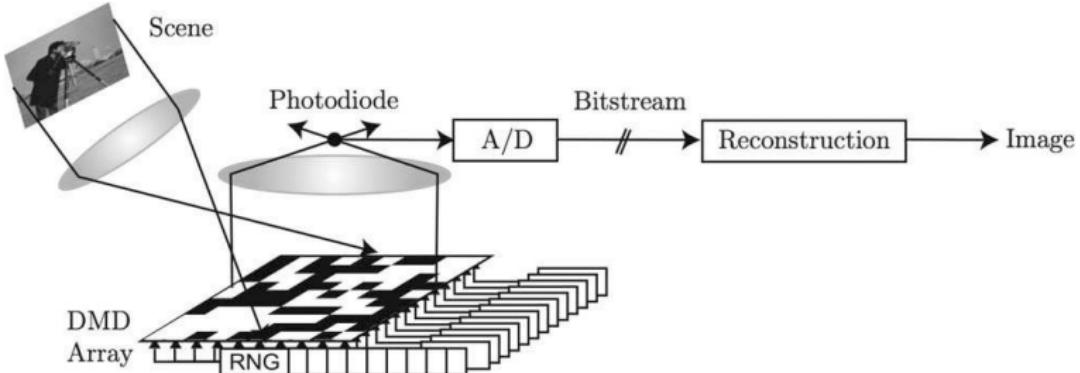
- ▶ Standard camera: record millions of pixels and then apply compression (e.g. jpeg compression) after the picture has been taken.
- ▶ New approach: we use only a single pixel detector to create images and we gather only a small fraction of the information, effectively compressing the image while taking it.

## Example: Single pixel camera (2)

How does it work? See figure below:

- ▶ uses an array of bacteria-sized mirrors to acquire a random sample of the incoming light.
- ▶ Each mirror can be tilted in one of two ways, either to reflect the light toward the single sensor or away from it.
- ▶ Thus the light that the sensor receives is a weighted average of many different pixels, all combined into one pixel.

## Example: Single pixel camera (3)



**Figure 6.** A schematic diagram of the “one-pixel camera.” The “DMD” is the grid of micro-mirrors that reflect some parts of the incoming light beam toward the sensor, which is a single photodiode. Other parts of the image (the black squares) are diverted away. Each measurement made by the photodiode is a random combination of many pixels. In “**One is Enough**” (p.114), 1600 random measurements suffice to create an image comparable to a 4096-pixel camera. (Figure courtesy of Richard Baraniuk.)

Original resolution:  $d = 64 \times 64 = 4096$ .

Compressed:  $k = 1600$  measurements.

## Example: Single pixel camera (4)

- Traditionally: many photodiodes on a grid, each of them measures a part of the image.
- Now: instead of a grid of photodiodes, have a grid of mirrors.
- Mirrors can be on and off. The mirrors that are on project their part of the image on one point, the new single photodiode (=the single pixel). **This photodiode thus receives a linear combination of the images on all the mirrors that are on.**
- One configuration of the mirrors corresponds to one linear measurement vector  $w_i$ .
- Repeat this procedure for  $k$  different mirror configurations  $w_1, \dots, w_k$  and store the  $k$  measurements the photodiode receives.
- Now try to reconstruct the original image from the vector of  $k$  measurements.

## Example: Single pixel camera (5)

Cool result: By taking of the order  $\Theta(s \log(d/s))$  snapshots, with a different random selection of pixels each time, the single-pixel camera is able to acquire a recognizable picture with a resolution comparable to  $d$  pixels.



**One is Enough.** A photograph taken by the “single-pixel camera” built by Richard Baraniuk and Kevin Kelly of Rice University. (a) A photograph of a soccer ball, taken by a conventional digital camera at  $64 \times 64$  resolution. (b) The same soccer ball, photographed by a single-pixel camera. The image is derived mathematically from 1600 separate, randomly selected measurements, using a method called compressed sensing. (Photos courtesy of R. G. Baraniuk, Compressive Sensing [Lecture Notes], Signal Processing Magazine, July 2007. ©2007 IEEE.)

## Example: Single pixel camera (6)

Nice blog discussion by Terrence Tao on this topic:

[https://terrytao.wordpress.com/2007/04/13/  
compressed-sensing-and-single-pixel-cameras/](https://terrytao.wordpress.com/2007/04/13/compressed-sensing-and-single-pixel-cameras/)

# Compressed sensing

Another way to describe it:

- ▶ We don't measure the signal  $x$  directly, but just a "compressed" version of it, namely we measure

$$\tilde{x} = Wx \in \mathbb{R}^k$$

where  $W$  is a  $k \times d$ -matrix with  $k \ll d$ .

- ▶ Now we want to reconstruct the high-dimensional signal  $x$  from the low-dimensional representation  $\tilde{x}$ .

WHAT WOULD BE THE NAIVE WAY OF RECONSTRUCTION?  
WHY DOESN'T IT WORK?

## Compressed sensing (2)

- ▶ To reconstruct, we would need to solve the linear system  
 $\tilde{x} = Wx$  for  $x$ .
- ▶ However, the latter is heavily underdetermined (we have  $k$  equalities but  $d$  unknowns, with  $k \ll d$ ). There are infinitely many solutions to this linear system.

The trick is going to be:

- ▶ We need to make assumptions on the  $x$  we are looking for.
- ▶ In particular, we assume that  $x$  is sparse in some basis. We will see below that this does the job.

# Compressed sensing main result

Theorem 44 (Compressed sensing with random measurements)

Fix a signal length  $d$  and a sparsity level  $s$ . Let  $W$  be a  $k \times n$  matrix with  $k = \Theta(s \log(d/s))$ , with each of its entries chosen independently from a standard normal distribution  $N(0, 1)$ . Then, with high probability over the choice of  $W$ , every  $s$ -sparse signal can be efficiently recovered from  $\tilde{x} = Wx$  by the following optimization problem:

$$\text{minimize} \|x\|_1 \text{ subject to } \tilde{x} = Wx$$

## Compressed sensing main result (2)

Some hand-wavy intuition for the result  $k = \Theta(s \log(d/s))$ :

- ▶ We know that we are looking for a vector of length  $d$  which has only  $s$  non-zero components (in some appropriate basis). But we don't know which are the non-zero components.
- ▶ There are  $\binom{d}{s}$  subsets of size  $s$  among the  $d$  components.
- ▶ To recover the vector we could try out all different such subsets, and then reconstruct based on these subsets.
- ▶ Any efficient algorithm to do so would need to be able to distinguish at least  $\log \binom{d}{s} \approx \Theta(s \log(d/s))$  many situations.

(This argument is similar to the proof for the lower bound in comparison-based sorting ...)

# Compressed sensing main result (3)

Key steps in the proof of the theorem:

1. If we choose the matrix  $W$  such that it has the “restricted isometry property” (see below), then any  $k$ -sparse vector  $x$  can be reconstructed from its compressed image  $\tilde{x}$  with only little distortion, by an inefficient algorithm using  $\ell_0$ -norm optimization.
2. The reconstruction of  $x$  from  $\tilde{x}$  can be calculated equally well (!) using  $\ell_1$ -norm optimization (rather than  $\ell_0$ -norm). This is very surprising!
3. It is easy to find matrices  $W$  that have the RIP property: we can use a random matrix with  $k = \Omega(s \log d)$  where  $s$  is the sparsity of the signal and  $d$  the original dimension of the space.

# Proof step 1: define RIP matrices

**Definition (Restricted Isometry Property):**

A matrix  $W \in \mathbb{R}^{k \times d}$  is  $(\varepsilon, 2s)$ -RIP if for all  $x \neq 0$  with  $\|x\|_0 \leq s$  we have

$$\left| \frac{\|Wx\|_2^2}{\|x\|_2^2} - 1 \right| \leq \varepsilon$$

Intuitively:

Multiplying an RIP-matrix to a sparse vector does not considerably change the norm of the vector, no matter which vector we choose.

# Proof step 1: Perfect reconstruction from RIP using $\ell_0$

The following theorem shows that RIP matrices yield a lossless compression for sparse vectors:

## Theorem 45 (Reconstruction based on 0-norm)

Let  $W$  be  $(\varepsilon, 2s)$ -RIP for some  $\varepsilon < 1$ ,  $x$  with  $\|x\|_1 \leq s$  (that is,  $x$  is sparse in the standard basis of  $\mathbb{R}^d$ ),  $y = Wx$  the compression of  $x$  by matrix  $W$ . Then the reconstruction

$$\tilde{x} := \operatorname{argmin}\{\|v\|_0 ; v \in \mathbb{R}^d, Wv = y\}$$

coincides exactly with  $x$ .

# Proof step 1: Perfect reconstruction from RIP using $\ell_0$ (2)

Proof of the theorem, by contradiction:

- ▶ Assume that  $\tilde{x} \neq x$ .
- ▶ By definition of  $\tilde{x}$  we have

$$\|\tilde{x}\|_0 \leq \|x\|_0 \leq s$$

In particular,  $\|x - \tilde{x}\|_0 \leq 2s$ .

- ▶ Now apply the RIP property to the vector  $z := x - \tilde{x}$ . Recall that  $Wx = W\tilde{x}$ , hence  $Wz = 0$ .
- ▶ Then the RIP property gives

$$\left| \frac{\|Wz\|_2^2}{\|z\|_2^2} - 1 \right| = |0 - 1| = 1 > \varepsilon$$

## Proof step 1: Perfect reconstruction from RIP using $\ell_0$ (3)

Note that this theorem immediately gives a first algorithm for reconstructing  $x$  from its sparse representation  $\tilde{x}$ .

WHICH ONE? IS IT A GOOD ONE?

# Proof step 1: Perfect reconstruction from RIP using $\ell_0$ (4)

We need to solve an  $\ell_0$ -optimization problem. This is combinatorial, hard, undesirably ...

## Proof step 2: Perfect reconstruction from RIP using $\ell_1$

Now comes the very surprising result: we get exact (!) recovery even if we replace the  $\ell_0$  norm by the  $\ell_1$  norm:

### Theorem 46 (Reconstruction based on 1-norm)

Under the same assumptions as before:

$$\operatorname{argmin}\{\|v\|_0 ; v \in \mathbb{R}^d, Wv = y\} = \operatorname{argmin}\{\|v\|_1 ; v \in \mathbb{R}^d, Wv = y\}$$

Proof: omitted, a nice writeup can be found in Chapter 23.3 of Shalev-Shwartz and Ben-David.

WHY IS THIS SURPRISING? WHY IS IT INTERESTING?

## Proof step 2: Perfect reconstruction from RIP using $\ell_1$ (2)

The theorem gives us a pretty efficient (=polynomial) way to exactly (!) reconstruct the original signal from the sparse one, by solving a linear program!

Remark:

There exists an even stronger version of the theorem which does not assume that the original vector is  $s$ -sparse. Essentially, the statement says that we can perfectly recover the  $s$  largest components.

## Proof step 3: Constructing RIP matrices

We still haven't clarified how we actually construct the compression matrix  $W$ :

### Theorem 47 (Random matrices are RIP)

- (i) Let  $s \leq d$  an integer,  $\varepsilon, \delta \in ]0, 1[$ . Choose  $k \geq \text{const} \cdot \frac{s \log(d/(\delta\varepsilon))}{\varepsilon^2}$ . Now choose  $W \in \mathbb{R}^{k \times d}$  such that each entry is drawn randomly from a normal distribution  $N(0, 1/s)$ . Then, with probability  $1 - \delta$  (over the choice of the matrix), the matrix  $W$  is  $(\varepsilon, s)$ -RIP.
- (ii) More generally, if  $U$  is any  $d \times d$  orthonormal matrix, then with probability  $1 - \delta$ , the matrix  $WU$  is  $(\varepsilon, s)$ -RIP.

Proof: omitted, a nice writeup can be found in Chapter 23.3 of Shalev-Shwartz and Ben-David.

## Proof step 3: Constructing RIP matrices (2)

Remark: This result is closely related to the theorem of Johnson-Lindenstrauss, which is widely used in randomized algorithms.

# More intuition: a different way to tell the same story

Compressed sensing is advantageous whenever

- ▶ signals are sparse in a known basis
- ▶ measurements (or computation at the sensor end) are expensive
- ▶ but computations at the receiver end are cheap.

## More intuition: a different way to tell the same story (2)

- ▶ One measures a relatively small number of **random linear combinations of the signal values** — much smaller than the number of signal samples nominally defining it.
- ▶ However, because the underlying signal is compressible, the nominal number of signal samples is still an overestimate of the effective number of degrees of freedom of the signal.
- ▶ As a result, the signal can be reconstructed with good accuracy from relatively few measurements by a clever nonlinear procedure.

## More intuition: a different way to tell the same story (3)

When does it work?

**Transform sparsity:** The desired image should have a sparse representation in a known transform domain (i.e., it must be compressible by transform coding).

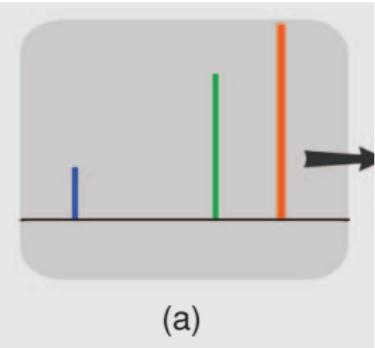
**Incoherence of undersampling artifacts:** The artifacts in linear reconstruction caused by undersampling should be incoherent (noise like) in the sparsifying transform domain.

**Nonlinear reconstruction:** The image should be reconstructed by a nonlinear method that enforces both sparsity of the image representation and consistency of the reconstruction with the acquired samples.

## Example: time series

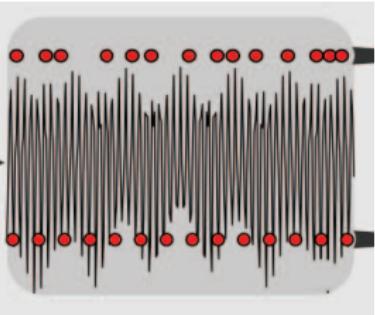
The following example is taken from Lustig, M., Donoho, D. L., Santos, J. M., Pauly, J. M. (2008). Compressed sensing MRI. Signal Processing Magazine, IEEE, 25(2), 72-82.

Sparse signal, as it would be in the appropriate basis (say, a vector of Fourier coefficients of a time series).



## Example: time series (2)

Signal in the “default basis” (say, the time series itself, not sparse):



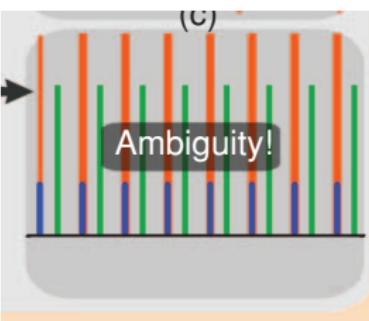
Assume it is too costly to sample the whole time series completely.

## Example: time series (3)

Obvious first idea: equispaced undersampling.

- ▶ Just measure (“sense”) the signal at equispaced positions (in the image on the previous slide, at the positions indicated by the red dots at the bottom).
- ▶ Replace the remaining entries with 0.
- ▶ Go over to the sparse basis and represent the signal there.

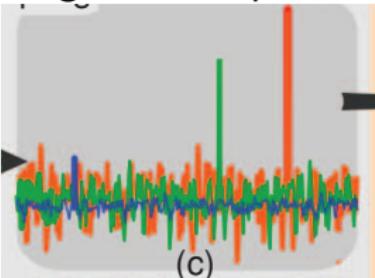
Result: artifacts called “aliasing”. It does not work at all!



## Example: time series (4)

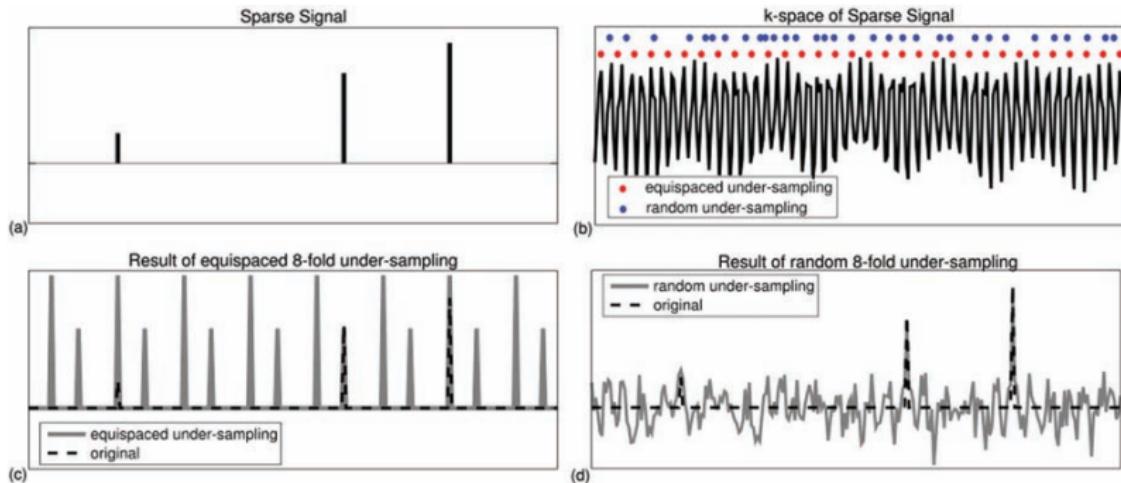
### The compressed sensing approach: Random undersampling

- ▶ Instead of sampling at equispaced positions, randomly pick some entries (in the image before, this is indicated by the red dots at the top).
- ▶ Try to represent the image in the sparse basis:



Works! If we threshold the small Fourier coefficients, we are left with the sparse representation of the signal.

## Example: time series (5)



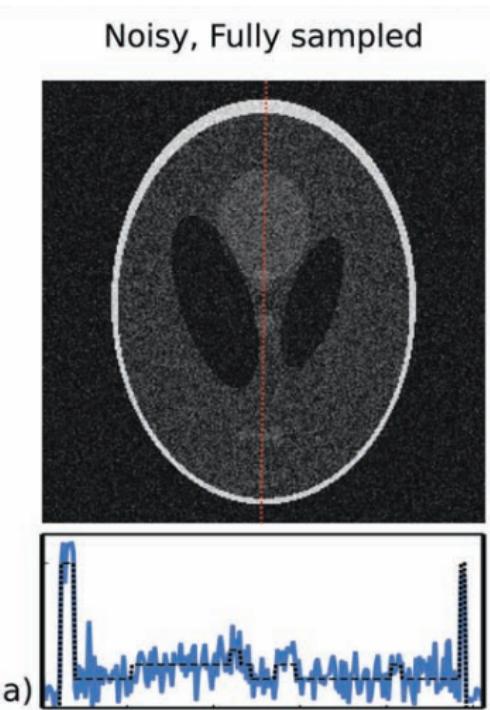
**Figure 2.** Reconstructing a sparse wave train. (a) The frequency spectrum of a 3-sparse signal. (b) The signal itself, with two sampling strategies: regular sampling (red dots) and random sampling (blue dots). (c) When the spectrum is reconstructed from the regular samples, severe “aliasing” results because the number of samples is 8 times less than the Shannon-Nyquist limit. It is impossible to tell which frequencies are genuine and which are impostors. (d) With random samples, the two highest spikes can easily be picked out from the background. (Figure courtesy of M. Lustig, D. Donoho, J. Santos and J. Pauly, *Compressed Sensing MRI*, Signal Processing Magazine, March 2008. © 2008 IEEE.)

## Example: Images

Example taken from: MacKenzie, Dana. Compressed sensing makes every pixel count. What is happening in the mathematical sciences 7 (2009): 114-127.

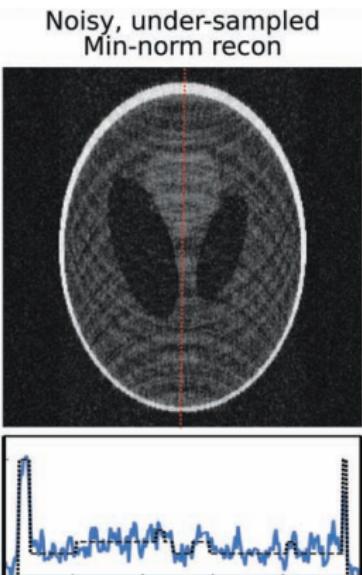
Original noisy image. Shown is the image itself and (I guess) the coefficients in Fourier (Wavelet?) basis. Signal is sparse in this basis (but of course, it was not recorded in this basis, here the transform to the sparse basis happened afterwards):

## Example: Images (2)



## Example: Images (3)

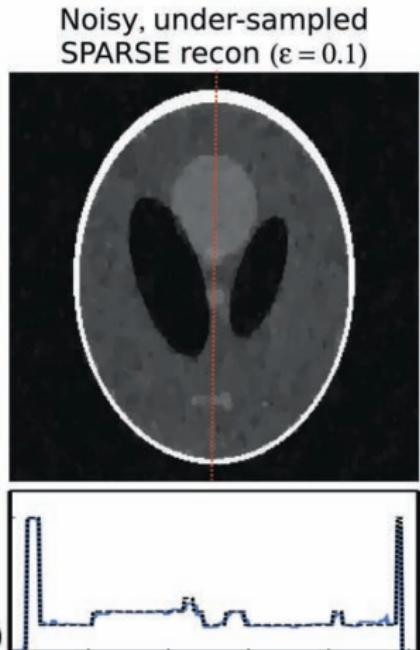
Now use random undersampling to record the picture, and reconstruct based on  $l_2$ -minimization:



Many artifacts.

## Example: Images (4)

Random undersampling, reconstruction based on  $\ell_1$ -minimization:



Nice :-)

# Relation to standard information theory

Shannon sampling theorem (1949):

- ▶ A time-varying signal with no frequencies higher than  $d$  hertz can be perfectly reconstructed by sampling the signal at regular intervals of  $1/2d$  seconds (that is, we sample at  $2d$  different time points).
- ▶ A signal with frequencies higher than  $d$  hertz cannot be reconstructed uniquely if we sample with this rate; there is always a possibility of aliasing (two different signals that have the same samples).

Compressed sensing: makes stronger assumptions than Shannon:

- ▶ The achievable resolution is controlled not only by the maximal number of frequencies (the dimension  $d$  of the space), but by the “information content” (the sparsity  $s$  of the signal).

## Relation to standard information theory (2)

- If we know that among the  $d$  different frequencies only  $s$  of them really occur, then we can reconstruct the signal from a small number of measurements.

# Outlook

- ▶ Active area of research
- ▶ Lots of actual applications!!! Cameras, MRI scanning, etc

# Ranking from pairwise comparisons

# Introduction

Text books (but I don't like both chapters so much):

- ▶ Mohri et al. chapter 9
- ▶ Shalev-Shwartz/Ben-David, Chapter 17.4

Papers: see the individual sections.

# Introduction, informal

- ▶ Ranking candidates for a job offering
- ▶ Ranking of the world's best tennis players
- ▶ Ranking of search results in google
- ▶ Ranking of molecules according to whether they could serve as a drug for a certain disease

IN WHICH SENSE ARE THESE PROBLEMS DIFFERENT, IN WHICH SENSE SIMILAR?

## Introduction, informal (2)

- ▶ top-k ranking vs full ranking
- ▶ sampling with or without replacement
- ▶ active vs. passive selection of comparisons
- ▶ distributed or not
- ▶ ground truth exists or not

Problems run under many different names: rank aggregation, ranking, tournaments, voting, ... and are tackled in many different communities (machine learning, computational social choice, theoretical computer science, etc).

# Introduction, more formal

- ▶ Given  $n$  objects  $x_1, \dots, x_n$ .
- ▶ In the simplest case, we assume that there exists a “true” total order  $\prec$  on the objects, that is there exists a permutation  $\pi$  such that  $x_{\pi(1)} \prec x_{\pi(2)} \prec \dots \prec x_{\pi(n)}$ .
- ▶ Goal is to learn this permutation from partial observations of the ranking. In the simplest case, observations are of the form  $x_k \prec x_l$  for certain pairs  $(k, l)$ .

## Introduction, more formal (2)

Distance functions between permutations:

Given two permutations  $\pi$  and  $\hat{\pi}$  of the same set of objects. Want to compute how different these rankings are.

- ▶ **Kendall- $\tau$  distance:** Count the number of pairs  $(i, j)$  that are in different order in the two permutations:

$$d_\tau(\pi, \hat{\pi}) := \frac{2}{n(n-1)} \sum_{i=1}^n \sum_{j=i+1}^n \mathbb{1}\{\text{sign}(\pi(i) - \pi(j)) \neq \text{sign}(\hat{\pi}(i) - \hat{\pi}(j))\}$$

- ▶ **Spearman- $\rho$  distance:** Count for each object by how much it is “displaced” in one permutation with respect to the other:

$$d_\rho(\pi, \hat{\pi}) = \sum_{i=1}^n |\pi(i) - \hat{\pi}(i)|$$

## Introduction, more formal (3)

- ▶ **Top-k differences.** Assume we are just interested in whether the top  $k$  objects in the two rankings coincide. Denote by  $S_k$  the set of first  $k$  objects in  $\pi$ , and by  $\hat{S}_k$  the corresponding set in  $\hat{\pi}$ . We define the distance

$$d_k(\pi, \hat{\pi}) := |S_k \Delta \hat{S}_k| := |(S_k \cup \hat{S}_k) \setminus (S_k \cap \hat{S}_k)|$$

Note that it only looks at the unordered sets, not at the order within the sets.

- ▶ **Normalized discounted cumulative gain (NDCG):** We take the ranking  $\pi$  as “reference ranking”. Then we compare it to the second ranking  $\hat{\pi}$ , but we weight errors among the top items of  $\pi$  more severely than errors for items at the bottom of the ranking  $\pi$ . Many different ways in which this can be done ...

## Introduction, more formal (4)

There exists a large variety of probabilistic model assumptions in the literature. Here are some typical examples:

- ▶ We assume there exists a true ranking. When asking a user to provide an answer to the question  $x_i ? x_j$ , he gives an incorrect answer with probability  $p$  (where  $p$  is independent of  $x_i, x_j$ ).
- ▶ We assume that the objects  $x_i$  can be represented by a real number  $u(x_i)$ , for example a utility score. Then we define  $x_i \prec x_j := u(x_i) < u(x_j)$ . The likelihood to observe an incorrect answer depends on the distance  $u(x_i) - u(x_j)$ . Many different versions, for example the BTL model below.

## Introduction, more formal (5)

- ▶ Model for paired comparisons: Bradley-Terry-Luce (BTL) model. Each object has a score  $u(x_i)$  (utility value, skill, ...). Probability of answers to comparisons are modeled by a logistic model:

$$P(x_i \succ x_j) = \frac{1}{1 + \exp(-(u(x_i) - u(x_j)))}$$

- ▶ Mallows model (probability distribution over all permutations): Assume that  $\pi$  is the true ranking. Then the probability to observe a ranking  $\hat{\pi}$  is chosen proportional to  $\alpha^{d_\tau(\pi, \hat{\pi})}$  where  $\alpha \in ]0, 1]$  is a parameter and  $d_\tau$  is the Kendall- $\tau$  distance. Choosing  $\alpha = 1$  implies the uniform distribution over all permutations, the closer  $\alpha$  is to 0, the more the mass concentrates around  $\pi$ .

## Introduction, more formal (6)

**Default statistical approach.** Given a probabilistic model, a straight forward idea is to use a **maximum likelihood estimator**: Find the permutation that maximizes the likelihood of the observed data. However, it is often infeasible due to computational complexity (need to have a clever way to try out all permutations).

**Default algorithmic approach:** Given the observations, find the permutation that is as consistent as possible with your observations (minimizes a loss function). For example, assume in a sports tournament that everybody played against everybody. Now find a ranking that violates as few outcomes as possible. This problem is NP hard, there exists a PTAS for it (Kenyon-Mathieu and Schudy: How to rank with few errors. STOC 2007).

## Simple but effective counting algorithm

Based on the paper:

Shah, Wainwright: Simple, robust and optimal ranking from pairwise comparisons. Arxiv, 2015.

# The model

Ground truth model is very general:

- ▶  $n$  objects
- ▶ For each pair of objects, assume a parameter  $p_{ij} := P(i \succ j)$ .  
Assume that  $P(i \succ j) + P(i \prec j) = 1$  (no ties).
- ▶ Define the score that measures the probability that object  $i$  beats a randomly chosen object  $j$ :

$$\tau_i := \frac{1}{n} \sum_{j=1}^n P(i \succ j)$$

This score  $\tau_i$  can be interpreted as the probability that object  $i$  wins against a randomly chosen object  $j$  (under the uniform probability distribution of objects). We consider the ranking induced by these scores as the true ranking. Note: high score = top of the list.

## The model (2)

Observation model:

- ▶ Assume that the number of times that a pair  $(i, j)$  is observed is distributed according to a binomial distribution  $\text{Bin}(r, p_{\text{obs}})$  (where  $r \in \mathbb{N}$  and  $p_{\text{obs}} \in [0, 1]$  are global parameters independent of  $i$  and  $j$ ).
- ▶ To generate the observations we proceed as follows:
  - ▶ For each pair  $(i, j)$ , we draw a random variable  $n_{ij} \sim \text{Bin}(r, p_{\text{obs}})$ . This is the number of times that we are going to observe comparisons between  $i$  and  $j$ .
  - ▶ Now we ask  $n_{ij}$  times independently whether  $i \prec j$  or  $i \succ j$ . We get the answers with probabilities according to  $p_{ij}$ .

This model is very general, it encompasses most of the more specialized models that exist in the literature.

Goal: Given a set of comparisons, find either the top-k ranking or the full ranking.

# The counting algorithm

Simple counting algorithm:

- ▶ Define  $\hat{\tau}_i$  as the number of times that object  $i$  has won over another object  $j$ , based on the observed comparisons.
- ▶ Define the estimated ranking (or the top-k set) as the order induced by the estimated scores  $\hat{\tau}_i$ .

This algorithm is about the simplest thing you can come up with, it is sometimes called **Borda count** or **Copeland method** in the literature.

# Bounds for exact recovery of top-k items

Define the following parameter:

$$\Psi_k(n, r, p_{\text{obs}}) := \underbrace{(\tau_k - \tau_{k+1})}_{=: \text{separation para.}} \cdot \sqrt{\underbrace{\frac{n \cdot p_{\text{obs}} \cdot r}{\log n}}_{\text{sampling para.}}}$$

- ▶ The separation parameter measures how well-separated the first  $k$  items are from the remaining ones.
- ▶ The sampling parameter is a complexity term that depends on the number  $n$  of objects and the expected number  $n \cdot p_{\text{obs}} \cdot r$  of observations per object.
- ▶ We will see below that the larger  $\Psi_k$ , the easier it is to discover the true ranking. (DOES IT MAKE SENSE?)

# Bounds for exact recovery of top-k items (2)

## Theorem 48 (Exact top k recovery)

- (a) Upper bound: Denote by  $S_k$  the set of true top-k times, and by  $\hat{S}_k$  the estimated set of top-k items according to the counting algorithm. If  $\Psi_k(n, r, p_{\text{obs}}) \geq 8$ , then  $\hat{S}_k = S_k$  with probability at least  $1 - 1/n^{14}$ .
- (b) Lower bound: If  $\Psi_k(n, r, p_{\text{obs}}) \leq 1/7$ ,  $n \geq 7$  and  $r \cdot p_{\text{obs}} > \log n / (2n)$ . Then there exist instances such that any algorithm that attempts to recover the top-k items will err with probability at least  $1/7$ .

# Digesting the theorem

Let's digest the upper bound by constructing a simple example:

Example 1:

- ▶ Assume the true ordering is  $o_1 \succ o_2 \succ \dots \succ o_n$ , that is the best player is  $o_1$ . Our goal is to find the best player (that is,  $k = 1$ ).
- ▶ Assume a noise-free setting: a better player always wins against a worse player, that is  $p_{ij} = 1$  if  $o_i \succ o_j$  and 0 otherwise.
- ▶ Then  $\tau_i = (n - i)/n$ , and in particular  $\tau_1 - \tau_2 = 1/n$ .
- ▶ Consider the case where we observe each pair exactly  $r$  times (we set  $p_{\text{obs}} = 1$ , so the number of observations is deterministic as well).

## Digesting the theorem (2)

- ▶ Upper bound in the theorem: perfect recovery works if  $\Psi := (\tau_1 - \tau_2) \cdot \sqrt{\frac{nr}{\log n}} \geq \text{const.}$ . In our case,  $\tau_1 - \tau_2 = 1/n$ , and solving the equation leads to  $r \geq n \log n$ . That is, the upper bound guarantees perfect recovery if we observe each pair (!) at least  $n \log n$  times. So overall we have to make  $n^3 \log n$  comparisons.

On the other side, the lower bound says the following:

- ▶ Assume that we have separation  $\tau_1 - \tau_2 = 1/n$  as in our example. Then, if we observe less than  $r = n^3 \log n$  we cannot guarantee recovery.
- ▶ The point is that the lower bound is a worst case statement: for the worst of all examples, we cannot guarantee recovery if we observe less than  $n^3 \log n$  examples.

## Digesting the theorem (3)

- ▶ For example, we can construct an example that has separation  $1/n$  as well, but has lots of noise:

Example 2: Assume that for  $i > 2$  we have

$P(1 \succ i) = 1/2 + 2/(n - 2)$ ,  $P(2 \succ i) = 1/2 + 2/(n - 2)$ ,  
and all other pairwise probabilities are  $1/2$ . This clearly is a  
very difficult case.

Taken jointly, upper and lower bound say:

- ▶ The counting algorithm gives perfect recovery if we get to see  $n^3 \log n$  comparisons (this is for the case of separation  $1/n$ ).
- ▶ There are instances where it fails if we get to see less than this amount of samples.
- ▶ In this sense, the counting algorithm is optimal (up to constants in the bounds).

## Digesting the theorem (4)

- ▶ But of course, the query complexity (number of comparisons we need) is huge. The problem is not that we have a bad algorithm (this is what the lower bound tells). The problem is that we make too little assumptions, so there is no structure we can exploit.

(As a side remark: the lower bound also holds if we make the Bradley-Terry-Luce assumptions, one can construct an example that satisfies their assumptions and still needs many comparisons).

## Digesting the theorem (5)

Just as comparison: In our example 1, we are in a completely noiseless case.

- ▶ WHAT IS THE NUMBER OF COMPARISONS WE NEED TO SORT A SEQUENCE?
- ▶ WHAT IS THE NUMBER OF COMPARISONS WE NEED TO FIND THE TOP ITEM IN A LIST OF ITEMS?

So we are miles away from this good performance. HOW CAN THIS BE, WHAT IS THE DIFFERENCE?

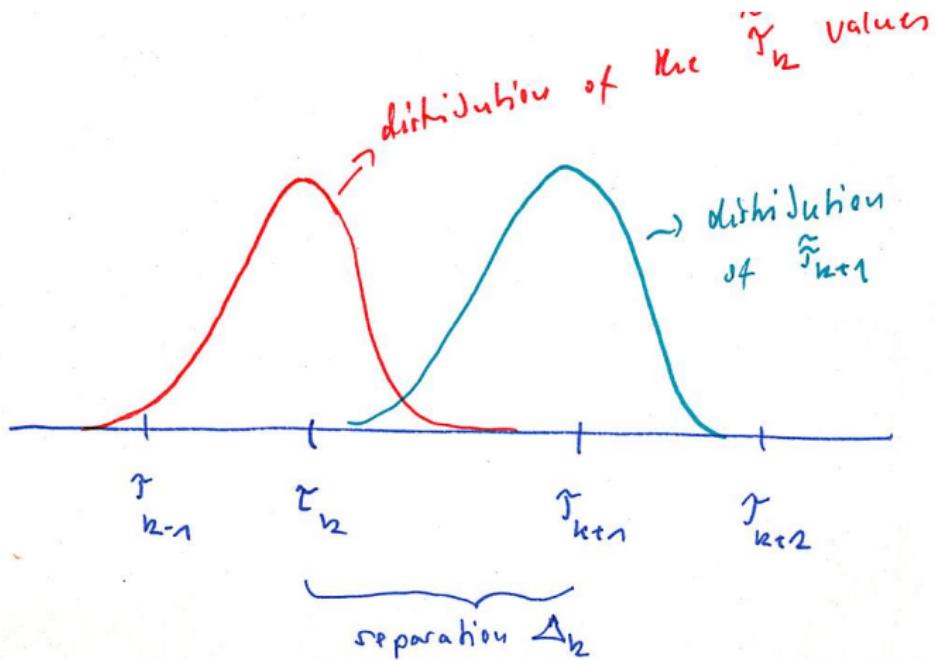
# Proof sketch, upper bound

Let's briefly look at the proof:

- ▶ For each pair  $(i, j)$ , we have a certain number of independent observations.
- ▶ The parameter  $\hat{\tau}_i$  is an average over these observations.
- ▶ This average is highly concentrated around its expectation. Applying standard concentration inequalities (Bernstein), one can show that the deviations of the random variables are small.
- ▶ In particular, we can then bound the probability that one of the top-k items “is beaten” (in terms of  $\hat{\tau}_i$ ) by one of the not-top-k items.

See the following figure:

# Proof sketch, upper bound (2)



## Proof sketch, lower bound

We construct one particular example in a clever way:

- ▶ For each  $a$  in  $\{k-1, k, k+1, \dots, n\}$  let  
 $S^*(a) := \{1, 2, \dots, k - 1\} \cup \{a\}$ . This is supposed to be the true top-k set.
- ▶ Define the probabilities

$$P_a(i \succ j) = \begin{cases} 1/2 & \text{if } i, j \in S^*(a) \text{ or } i, j \notin S^*(a) \\ 1/2 + \delta & \text{if } i \in S^*(a) \text{ and } j \notin S^*(a) \\ 1/2 - \delta & \text{if } i \notin S^*(a) \text{ and } j \in S^*(a) \end{cases}$$

- ▶ Note that the true  $\tau$ -values give the correct top-k set.
- ▶ Our goal is to identify the true permutation based on observations, that is we want to find the correct parameter  $a$  that has been used.

## Proof sketch, lower bound (2)

To construct the lower bound, we now want to show that no matter which algorithm we use to estimate the correct top-k set in our example, it always errs with a constant probability.

To this end we use a tool from information theory: **Fano's inequality**. Essentially it says that if we want to recover a certain parameter, we need to receive a certain amount of “signal” or “information”.

- ▶ Assume that  $a$  is chosen uniformly from  $k, \dots, n$ . Then we sample observations according to the model  $P_a$ .

## Proof sketch, lower bound (3)

- ▶ Fano's inequality now states that any algorithm that estimates  $a$  by some  $\hat{a}$  has to make an error of at least

$$P(a \neq \hat{a}) \geq 1 - \frac{I(a, \text{observation}) + \log 2}{\log(n - k + 1)}$$

So we need to bound the mutual information  $I(a, \text{observation})$ , which boils down to a sum of Kullback-Leibler divergences  $D(P_a || P_b)$ . They can be computed by standard methods.

Details skipped.

# Exact recovery of full ranking

The bound for top-k ranking can immediately be turned into a bound of exact full ranking. The main observation is that a ranking is correct if the top-k rankings for all  $k = 1, \dots, n - 1$  are correct. This immediately leads to:

## Theorem 49 (Upper bound, full permutation)

Let  $\hat{\pi}$  be the permutation induced by the estimated scores  $\hat{\tau}$ , and  $\pi$  the one by the true scores  $\tau$ . If  $\Psi_k(n, r, p_{\text{obs}}) \geq 8$  for all  $k = 1, \dots, n - 1$ , then  $P(\hat{\pi} = \pi) \geq 1 - 1/n^{13}$

Proof: union bound with the previous theorem (union bound leads to power 13 instead of 14).

## Approximate recovery

Result looks surprisingly similar. Just the separation term now not depends just on  $\tau_k - \tau_{k+1}$ , but on all  $\tau$ -values in a certain neighborhood of  $k$  (where the size of the neighborhood depends on the error we are allowed to make).

We still get the same kind of worst case query complexity.

Details skipped.

# Discussion

- ▶ On a high level, the theorem shows two things:
  - ▶ Ranking from noisy data is difficult if we don't make any assumptions.
  - ▶ You cannot improve on the counting algorithm — unless you do make more assumptions.
- ▶ In practice, the query complexity of  $n^3 \log n$  is completely out of bounds, there is no way you can collect that many comparisons in a realistic setting. So what is obviously needed are algorithms that work well with less queries in realistic settings (assumptions).

# Learning to rank

# Learning to rank

- ▶ Objects  $x_1, \dots, x_n$ .
- ▶ Observations of the form  $x_i \prec x_j$ . Encode this as follows:
  - ▶ Consider the space  $S$  of all unordered pairs of objects.
  - ▶ Output variable  $y_{ij} = \begin{cases} +1 & \text{if } x_i \prec x_j \\ -1 & \text{otherwise} \end{cases}$
- ▶ Goal: learn a classifier that makes as few mistakes as possible.

## Naive idea: ERM

The first naive algorithm we can think of is to perform empirical risk minimization on the set of permutations: that is we pick the permutation that agrees most with our observations.

We have mentioned already that this is NP hard to do (**computational complexity**), but let's look at how many queries we would need (**query complexity**).

# Generalization bounds for learning to rank

## Proposition 50 (VC dim of permutations)

Consider a set  $V$  of  $n$  objects, and the set  $S$  of all unordered pairs of objects. Denote by  $\Pi$  the set of permutations of  $V$ . Each permutation  $\pi$  induces a classifier  $f_\pi : S \rightarrow \{-1, 1\}$  on the set  $S$  (as described above). Then the space  $\mathcal{F} := \{f_\pi \mid \pi \in \Pi\}$  has VC-dimension  $n - 1$ .

**Proof:** Step 1: Prove that  $VC < n$ . To this end, consider any subset  $S' \subset S$  with  $|S'| = n$ . Want to show that it cannot be shattered by the function class  $\mathcal{F}$ . We construct a proof by contradiction.

- ▶ Assume we have a set  $S' \subset S$  of  $n$  pairs of objects that can be shattered by  $\mathcal{F}$ .

## Generalization bounds for learning to rank (2)

- ▶ Consider the comparison graph of  $S'$ : Vertices = all  $n$  objects; undirected edge from object  $i$  to  $j$  if  $\{i, j\} \in S'$ .
- ▶ The graph has  $n$  vertices and  $n$  edges by construction, so it needs to contain an undirected cycle. Now observe that **we cannot shatter the pairs in  $S'$  that correspond to the edges in the cycle**: we cannot realize the function that corresponds to  $x_i \prec x_j \prec \dots \prec x_l \prec x_i$ , because the latter implies  $x_i \prec x_i$ . ↴

## Generalization bounds for learning to rank (3)

Step 2: Prove that  $VC \geq n - 1$ . To this end, need to find at least one subset  $S' \subset S$  with  $|S'| = n - 1$  that can be shattered. Using the same construction as above, we simply choose  $S'$  such that the graph is a tree. Can always be done. This does it.



Remark: naively, the set  $\Pi$  consists of  $n!$  many permutations, so the shattering coefficient is  $n!$ . The log-shattering coefficient is then  $\log(n!) = n \log n$ , So the first natural guess is that the VC dim might be  $n \log n$ . We now see that it is even  $n - 1$ .

## Generalization bounds for learning to rank (4)

Our standard VC-generalization bound for a class with VC-dimension  $d$  over a sample of  $m$  comparisons is that with probability at least  $1 - \delta$ , any permutation  $\pi$  satisfies

$$R(f) \leq R_n(f) + 2\sqrt{\frac{d \log(2em/d) - \log(\delta)}{m}}.$$

As a rule of thumb: how many sample points do you need to achieve an error of about  $\varepsilon$  at most? Here is the argument:

- ▶ Ignore all log terms and constants.
- ▶ Then the error  $\varepsilon$  is of the order  $\varepsilon := \sqrt{d/m}$ . Solving for  $m$  tells us that we need to observe of the order  $d/\varepsilon^2$  comparisons.  
In our case with  $d = n$  this means that we need to observe about  $m := n/\varepsilon^2$  comparisons to achieve an error of at most  $\varepsilon$ , with high probability.

Seems pretty good!

# Generalization bounds for learning to rank (5)

Comparison to the bound in the Shah/Wainwright approach  
(simple counting algorithm):

- ▶ Note: the bound in the Shah/Wainwright approach was:  
recovery works if we see about  $n^3 \log n^3$  comparisons (with similar results for approximate recovery and recovery of the full ranking).
- ▶ Now we have a VC bound that says that of the order  $n$  examples are enough for good classification performance.
- ▶ Both approaches make only minimalistic / no assumptions whatsoever on the structure of the numbers we want to sort.

WHERE IS THE CATCH?

# Generalization bounds for learning to rank (6)

- ▶ Note that the Shah/Wainwright bound talks about **identifying** the correct ranking, while the VC bound just talks about **predicting** the outcome of comparisons.
- ▶ Consider an example that is difficult in the Shah / Wainwright framework: all  $\pi_{ij}$ -values close to 1/2, so the  $\tau$ -values are very similar to each other.
  - ▶ Shah/Wainwright: need many samples to find the actual ranking.
  - ▶ Learning to rank: the bound only considers the estimation error of the classifier, when applied to predict unobserved comparisons. [As a side remark: in the given example even the Bayes classifier would have a poor performance close to random guessing. The generalization bound just tells us that we need not so many comparisons to come close to the performance of the actual Bayes classifier. ]

## Generalization bounds for learning to rank (7)

- ▶ So the bounds are difficult to compare, neither the estimation error of the predictor nor its approximation error are directly related to the difficulty of the ranking problem.

# SVM ranking

As ERM is infeasible computationally, we could use a linear SVM instead:

- ▶ Encode an ordered pair of objects by a feature vector  $x_{ij} := e_i - e_j \in \mathbb{R}^n$  and the outcome  $y_{ij}$  as described above.
- ▶ Get training points of the form  $(x_{ij}, y_{ij})$ .
- ▶ Classify using a linear hyperplane (that is, find a vector  $w \in \mathbb{R}^n$ ) such that  $\text{sign}(\langle w, z_{ij} \rangle)$  makes as few errors as possible. Use an SVM to find this hyperplane.
- ▶ In particular, the predicted ordering can then be recovered by the ordering of the coordinates of  $w$ .

Can also prove margin-type generalization bounds for SVM ranking, skipped.

## Application: distance completion problem

This is a topic we are actually working on right now in my research group.

# Setup: Triplet comparisons

A scenario beyond simple ranking:

- ▶ Points  $X_1, \dots, X_n$  from  $\mathbb{R}^d$
- ▶ We don't know any numeric information such as vector representations or distance values
- ▶ We just get to see binary variables that compare distances:

$$d(X_i, X_j) < d(X_k, X_l) = \text{ true or false}$$

So in the ranking language, we get a partial ranking between the distances of the objects.

# Setup: Triplet comparisons (2)

Why is this interesting?

It is often easy to say that things “are pretty similar” or “not similar at all”, but it is hard to come up with good ways to quantify this.

Example user ratings: easier to compare

$$\text{dist}(\text{}) < \text{dist}(\text{})$$

... than to give numeric distance values:

$$\text{dist}(\text{}) = 0.1$$

$$\text{dist}(\text{}) = 0.7$$

# Setup: Triplet comparisons (3)

In the following we consider:

- ▶ Triple questions:  $d(\textcolor{red}{X}_i, \textcolor{blue}{X}_j) \stackrel{?}{\leq} d(\textcolor{red}{X}_i, \textcolor{green}{X}_k)$
- ▶ Quadruple questions:  $d(\textcolor{red}{X}_i, \textcolor{blue}{X}_j) \stackrel{?}{\leq} d(X_k, \textcolor{green}{X}_l)$

# Distance completion problem

Distance comparison problem:

- ▶  $n$  objects from  $\mathbb{R}^d$
- ▶ All we observe are a subset of all triple comparisons of the form  $d(X_i, X_j) < d(X_i, X_k)$ .
- ▶ Want to estimate the full ranking between all distances  $d_{ij}$ .

The full distance ranking can then for example be used to find the nearest neighbors of each data points, and then we can apply classification algorithms, regression algorithms, clustering algorithms, etc.

# Query complexity of the distance completion problem

Given  $n$  objects, how many randomly chosen triple comparisons do I need in order to estimate the true distance ranking reliably?

# Query complexity, first observations

First observations about ranking  $m$  objects.

- ▶ There are of the order  $m := n^2$  distances. A comparison-based sorting algorithm would need  $\Theta(m \log m) \Theta(n^2 \log n^2)$  many (actively chosen !) comparisons. In the noiseless case, it would produce the perfect ranking.
- ▶ If we use ERM to recover an approximate ranking, we would just need  $m/\varepsilon^2 = n^2/\varepsilon^2$  many queries to learn the ranking up to error  $\varepsilon$  (ignoring that the computational complexity is much too high).
- ▶ If we apply the simple counting algorithm by Shah/Wainwright, we also get a query complexity of  $m^3 \log m = n^6 \log n$  (of randomly chosen comparisons). Would also work in a noisy case.

## Query complexity, first observations (2)

In any application in real-world, query complexities of order  $n^2$  are prohibitive ...

## Query complexity, first observations (3)

However:

- ▶ Observe that the three approaches I mentioned above do not make any assumption on the objects that need to be ordered, it can be any arbitrary collection of numbers.
- ▶ We know more about our data: the things we want to order are Euclidean distances. Can we exploit this in some way?

# Query complexity, exploit structure

The answer is yes: we can exploit the structure of the problem.

- ▶ Consider the set of points  $X_1, \dots, X_n \in \mathbb{R}^d$ , and a certain subset of triple questions.
- ▶ Observe that a triple comparison gives a relationship in form of a hyperplane:  $d_{ij} < d_{ik}$  is equivalent to saying: if we consider the hyperplane between point  $X_j$  and  $X_k$ , then point  $X_i$  is on the same side as  $X_j$ .
- ▶ We can now build equivalence classes of point sets: the ones that satisfy the same hyperplane conditions.
- ▶ It is now possible to “count” the number of equivalence classes (non-trivial!). The result is: there are of the order  $2^{dn \log n}$  such equivalence classes (where  $d$  is the dimension of the space).
- ▶ This means that the log-shattering coefficient of the set of all Euclidean (!) distance completions is just  $dn \log n$ .

## Query complexity, exploit structure (2)

- ▶ So we the standard shattering-coefficient generalization bound says that with high probability, if we want to approximate the correct distance ranking up to error  $\varepsilon$ , we need of the order  $dn \log n / \varepsilon^2$  many triple questions, close to linear!!!
- ▶ Note that this is much better than the  $n^2 \log n$  requirements we had without making any assumption.

This is a very nice example to demonstrate that exploiting the structure in the problem helps (at least in theory).

And this is also a nice example for the type of questions we work in in my group - we just proved this result a couple of weeks ago...

# Query complexity, exploit structure (3)

Outlook:

- ▶ Note that while this shows that in theory we only need few triples to recover the full distance ranking for Euclidean points, we don't know how to do it in practice.
- ▶ We would need to have an algorithm that does ERM on the set of equivalence classes ...

# Spectral ranking

Based on the following paper:

Fogel, d'Aspremont, Vojnovic: SerialRank: Spectral ranking using seriation. NIPS 2014.

# Spectral ranking

Setting as before: we observe pairwise comparison, want to output a ranking.

Define the comparison matrix:

$$C_{ij} = \begin{cases} 1 & \text{if } i \succ j \\ -1 & \text{if } i \prec j \\ 0 & \text{if no data exists} \end{cases}$$

Define a similarity matrix as follows:

$$S_{ij} := \sum_{k=1}^n \frac{1 + C_{i,k}C_{j,k}}{2}$$

(counts the number of matching comparisons of  $i$  and  $j$  with other items  $k$ )

# Spectral ranking (2)

SpectralRanking algorithm:

- ▶ Compute the similarity matrix  $S$  based on the observed data
- ▶ Construct the unnormalized Laplacian  $L$  and compute its second eigenvector.
- ▶ Rank all items according to the corresponding entries in this eigenvector.

## Spectral ranking (3)

There are lots of theoretical results on this algorithm:

- ▶ Assume we get to see all pairwise comparisons, answered truthfully, and there are no ties. Then SpectralRanking recovers the correct ranking perfectly.  
(not interesting from an algorithmic point of view, we could just do topological sort in this case).
- ▶ Given a comparison matrix for  $n$  objects, with at most  $m$  corrupted entries (selected uniformly at random). Then if  $m = O(\sqrt{\delta n})$ , then the SerialRank algorithm will produce the ground truth ranking with probability at least  $1 - \delta$ .  
This is the interesting statement.

Proofs are based on some old work by Atkinson 1998, we skip them.

## Google page rank

The setting here is not a pairwise-comparison setting. But no student should leave this university without knowing google page rank, so let's discuss it anyway.

# The setting

Want to build a search engine:

- ▶ Query comes in
- ▶ First need to find all documents that match the query
- ▶ Then need to decide which to display on the top of the list. So we need to rank the search results according to their “relevance”.

Early attempts looked at the content of the documents (count how often the keyword occurs, etc).

The new idea by the google founders was to instead look at the link structure of the webpages.

# Page Rank

Published by Brin, Page, 1998.

Main idea:

- ▶ A webpage is important if many important links point **to** that page.
- ▶ A link is important if it comes **from** an page that is important.

Results of a search query should then be ranked according to importance.

## Page Rank (2)

Given a directed graph  $G = (V, E)$ , potentially with edge weights  $s_{ij}$ , define:

- ▶ Out-degree:  $d_{\text{out}}(i) = \sum_{\{k|i \rightarrow k\}} s_{ik}$
- ▶ In-degree:  $d_{\text{in}}(j) = \sum_{\{k|k \rightarrow j\}} s_{kj}$

Define the ranking function  $r$  for all vertices:

$$r(j) = \sum_{i \in \text{parents}(j)} \frac{r(i)}{d_{\text{out}}(i)} \quad (*)$$

This is an implicit definition. We need to find a way to solve this for  $r(j)$ , for all  $j$ .

## Page Rank (3)

Define the matrix  $A$  with entries

$$a_{ij} = \begin{cases} 1/d_{\text{out}}(i) & \text{if } i \rightarrow j \\ 0 & \text{otherwise} \end{cases}$$

and the vector  $r$  with the relevance scores as entries.

Observe that  $(r^t \cdot A)_j = \sum_i r_i a_{ij}$ , so we can rewrite  $(*)$  as

$$r^t = r^t \cdot A$$

So  $r$  is a left eigenvector of  $A$  with eigenvalue 1.

The page rank idea consists of ranking vertices according to this eigenvector.

In the following we will consider two things:

- ▶ Interpretation as a random surfer model
- ▶ How to compute the eigenvector.

# Random walks on a graph

To give the random surfer interpretation to pagerank, we first need to learn about random walks on a graph:

- ▶ Consider a directed graph  $G = (V, E)$  with  $n$  vertices.
- ▶ A **random walk** on the graph is a time-homogenous, discrete-time Markov chain. At each point in time, we randomly jump from one vertex to a neighboring vertex. The probability to end in one of the neighbors only depends on the current vertex, not on the past beyond this.
- ▶ It is fully described by **transition matrix  $P$**  with entries

$$p_{ij} = P(X_{t+1} = v_j | X_t = v_i).$$

For a weighted graph with similarity edge weights  $s_{ij}$ , have

$$p_{ij} = s_{ij}/d_i \text{ and in particular } P = D^{-1}S.$$

## Random walks on a graph (2)

Initial distribution:

At time point 0, we start the random walk at a random vertex according to probability row vector  $\mu = (\mu_1, \dots, \mu_n)$  with  $\mu_i \geq 0$ ,  $\sum \mu_i = 1$ . The special case where we start at a deterministic vector corresponds to the case where  $\mu = (0, \dots, 0, 1, 0, \dots, 0)$ .

# Random walks on a graph (3)

$k$ -step distribution:

- ▶ At time  $t = 0$ , the state distribution is  $\mu$ , our initial distribution.
- ▶ At time  $t = 1$ , the distribution is  $\mu P$ .
- ▶ At time  $t = 2$ , the distribution is  $(\mu P)P = \mu P^2$ . Note that entry  $ij$  of the matrix  $P^2$  describes all possible ways to get with exactly 2 steps from vertex  $i$  to  $j$ .
- ▶ In general, the matrix  $P^k$  describes the  $k$ -step transition probabilities.

# Random walks on a graph (4)

Stationary distribution:

Intuition: if the random walk runs for a long time, it will converge to an equilibrium distribution. It is called the stationary distribution or the invariant distribution.

Definition of a stationary distribution: If we start in a stationary distribution  $\pi$  and perform one step of the random walk, we have again the stationary distribution. In formulas:  $\pi$  is a stationary distribution of  $P$  if

$$\pi P = \pi$$

# Random walks on a graph (5)

Convergence to the stationary distribution:

Consider a graph  $G$  that is weighted, undirected, connected, and not bipartite. Denote by  $P$  the transition matrix. Then:

- ▶ The stationary distribution of  $P$  is given as the (normalized) degree vector:  $\pi_i = d_i / (\sum_j d_j)$  (CAN YOU SEE WHY?)
- ▶  $\lim_{t \rightarrow \infty} P(X_t = v_i) = \pi_i$
- ▶ The matrix  $P^t$  converges to the matrix  $\mathbb{1}\pi$  with constant columns  $\pi$ . The speed of convergence depends on the eigengap between the first and second eigenvalue of  $P$ :
  - ▶ Largest eigenvalue is always 1, second largest eigenvalue  $\lambda_2$  satisfies  $\lambda_2 < 1$  (note: it might not be real-valued!). Note that right and left eigenvalues are the same, just the eigenvectors differ.
  - ▶ Perron-Frobenius theorem:  $P^t = \mathbb{1}\pi + O(n^c |\lambda_2|^t)$ .

# The random surfer model

Recall the definition of the matrix  $A$  for pagerank. Observe:

- ▶ the matrix  $A$  is the transition matrix of a random walk on the graph of the internet.
- ▶ the ranking vector  $r$  is its stationary distribution.

If done naively, two big problems:

- ▶ dangling nodes (e.g., pdf pages).
- ▶ disconnected components

## The random surfer model (2)

Solution to both problems:

we introduce a random restart (“teleportation”):

- ▶ with probability  $\alpha$  close to 1, we walk along edges of the graph.
- ▶ With probability  $1 - \alpha$ , we teleport: we jump to any other random webpage.

Transition matrix is then given as

$$\alpha P + (1 - \alpha) \frac{1}{n} \mathbb{1}$$

where  $n$  is the number of vertices and  $\mathbb{1}$  the constant one matrix.

The ranking is then the stationary distribution of this matrix.

## How to compute it: the power method

Need to compute an eigenvector of a matrix of size  $n \times n$  where  $n$  is the number of webpages in the internet (2014: one billion webpages).

Computing an eigenvector of a symmetric matrix has worst case complexity of about  $O(n^3)$  (and btw, our current matrix is not symmetric).

IS THERE ANY REASON TO BELIEVE THAT THIS MIGHT WORK?

# How to compute it: the power method (2)

The simplest way to compute eigenvectors: the power method

- ▶ Let  $A$  be any diagonalizable matrix.
- ▶ Goal: want to compute eigenvector corresponding to the largest eigenvalue.
- ▶ Observe: Denote by  $v_1, \dots, v_n$  a basis of eigenvectors of matrix  $A$ . Consider any vector  $v = \sum_i a_i v_i$ . Then

$$Av = A\left(\sum_i a_i v_i\right) = \sum_i a_i(Av_i) = \sum_i a_i \lambda_i v_i$$

If we apply  $A$   $k$  times, then:

$$A^k v = \sum_i a_i \lambda_i^k v_i = \underbrace{a_1 \lambda_1^k \left( v_1 + \sum_{i=2}^n \frac{a_i}{a_1} \frac{\lambda_i^k}{\lambda_1^k} v_i \right)}_{\text{dominates}} \underbrace{\left( \frac{a_i}{a_1} \frac{\lambda_i^k}{\lambda_1^k} \right)}_{\text{vanishes}}$$

# How to compute it: the power method (3)

The Power Method, vanilla version:

- 1 Initialize  $q_0$  by any random vector with  $\|q_0\| = 1$
- 2 **while** not converged
- 3      $z^{(k)} := Aq^{(k)}$
- 4      $q^{(k)} := z^{(k)} / \|z^{(k)}\|$

Caveat:

- ▶ Won't work if  $q_0 \perp$  first eigenvector
- ▶ Does not necessarily converge if the multiplicity of the largest eigenvalue is larger than 1.
- ▶ Speed of convergence depends on the gap between the first and second eigenvalue, namely  $\lambda_2/\lambda_1$ .

# How to compute it: the power method (4)

Implementation of page rank is a simple power iteration:

- We Initialize with constant vector  $r = e = (1, \dots, 1)^t$ .
- We iterate until convergence:

$$\begin{aligned}r_{k+1}^t &= r_k^t(\alpha A + (1 - \alpha)e v^t) \\&= \alpha r_k^t A + (1 - \alpha) \underbrace{r_k^t e}_{=1} v^t \\&= \alpha \underbrace{r_k^t A}_{\text{sparse}} + (1 - \alpha)v^t\end{aligned}$$

Comments:

- $v$  is the “personalization vector” ( $\approx$  probability over all webpages of whether the surfer would like to see that page)

## How to compute it: the power method (5)

- ▶  $1 - \alpha$  is the teleportation parameter.
- ▶ in the last line, we essentially have to perform one sparse matrix-vector multiplication, this can be done in parallel.
- ▶ Speed of convergence depends on the gap between first and second eigenvalue. Personalization adds speed because if the spectrum of  $P$  is  $\{1, \lambda_2, \lambda_3, \dots\}$ , then the spectrum of the personalize matrix is  $\{1, \alpha\lambda_2, \alpha\lambda_3, \dots\}$ .  
Thus we have a tradeoff:  $\alpha$  large  $\leadsto$  small gap, slow convergence, but structure of the web graph well represented.

# The data processing chain: from raw data to machine learning

# Preparing the data

## Data aquisition: train versus test distribution

# Train versus test distribution

Machine learning starts with acquiring good data. The results of your learning algorithm can only be as good as the information in your data!!!

If you want to train a classifier to perform a certain task and you start collecting data for training, you should ask yourself the following questions:

Are all the potential test cases covered in the training data, with all the variety that exists?

- ▶ If you want to classify digits, are all of them going to be upright? If not, add digits in all orientations to your system.

## Train versus test distribution (2)

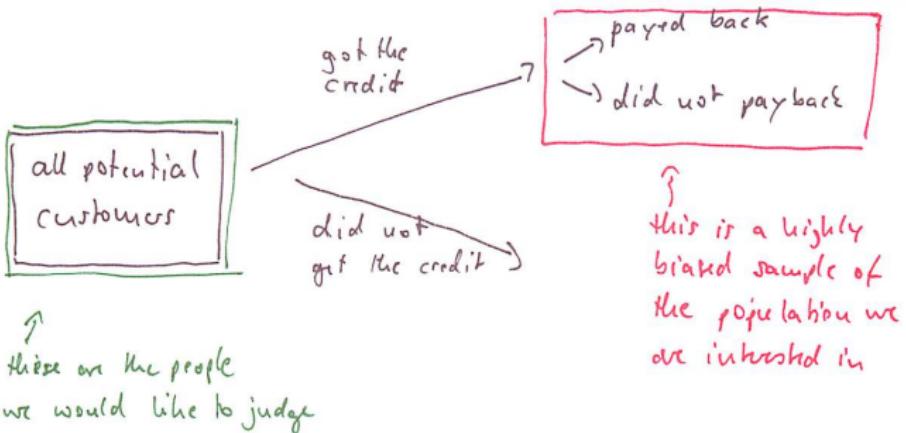
Is your training set “representative” for the test cases: is the distribution of your training inputs roughly the same as the distribution of your test inputs?

- ▶ If you want to predict whether general customers like a certain product, it is not a good idea to just collect opinions from students, say.
- ▶ Yet another sampling bias: We take a questionnaire in the machine learning class, about whether you like the class or not.
  - ▶ Because we take it towards the end of the semester, the people who disliked it most are no longer present (because they already dropped the lecture).
  - ▶ And the people who never attend the lecture because they think it is enough to read the slides at home are also not present.

## Train versus test distribution (3)

- ▶ Much more subtle: credit scoring rules. Such rules are used by the banks to predict whether a customer is likely to pay back a loan in time. The problem is that the available training data (persons plus the knowledge whether they payed back) is highly biased, because the banks only gave the credits to pre-selected people in the first place (so if their “old” selection rule never gave a credit to females, say, then they never get positive training examples for females who pay back the credit).

## Train versus test distribution (4)



At least, try to be aware of any “in-balancedness” in your training set.

# Converting raw data to training data

# Raw data to training data

Often there is a considerable amount of decisions to take when you go from raw data to training data.

Example: you want to detect faces in images, and you have a set of images (with and without faces).

- ▶ What exactly do you use as training examples? The whole image? The part of the image that contains a face / not a face? All  $16 \times 16$  patches of your image?
- ▶ What representation of the image do you use in the first place? What color space? What resolution?
- ▶ What do you do with different brightnesses? Do you also use the additional data provided by the camera as input (exposition time, aperture, focus, etc)?

# Data cleaning: missing values, outliers

# Outlier detection

Data often contains “outliers”:

*An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism.*

Unfortunately, outliers can have a very big influence on the outcome of certain algorithms.

DO YOU KNOW AN EXAMPLE FOR THIS?

Consequently, we sometimes might want to remove outliers from our data.

# Outlier detection (2)

There exist many algorithms for outlier detection:

- ▶ The classical statistics / model-based approach:
  - ▶ Fit some model to the data, say a mixture of Gaussians
  - ▶ Then the outliers are points which have very low probability under this model.
  - ▶ For most ML applications, this is not applicable at all ... data too complex ... try to avoid building explicit models ...
- ▶ Model-free approach:
  - ▶ Want to find a set  $S$  which has two properties:  
it is as small as possible, , but it contains most of the data points.
  - ▶ To identify outliers, we have to find such a set  $S$ . We then say that all points that fall outside of  $S$  are outliers.

## Outlier detection (3)

- ▶ The one-class SVM falls into this framework (see the book of Schölkopf/Smola).
- ▶ Many other approaches exist.

# Outlier detection (4)

Very important to understand:

- ▶ “Right” or “wrong” does not really exist (outlier detection is an unsupervised technique!)
- ▶ Depending on the algorithm (and, as a matter of fact, the underlying distance function) completely different points might get marked as outliers.
- ▶ Note that “outliers” are not always bad and undesired, to the contrary: these might be points that are particularly interesting.
  - ▶ For example, if we want to assess the side effects of some medical treatment, there might be just a very small number of patients who have severe side effects, but these are the ones we care about.
  - ▶ If we throw outliers away, we might lose the most important data!

## Outlier detection (5)

Always be suspicious if people generously remove outliers! I tend not to use outlier detection, unless I really know what I am doing...

# Missing values

Often data is not complete, you have missing values. Two big cases:

- ▶ Missing at random: this is the easier case, no bias introduced by the fact that data is missing.
- ▶ Missing not at random: values could miss systematically, and the fact that a value is missing might contain information:
  - ▶ You run a population survey. One of the questions is about the income. People might not want to disclose their income if it is really high or really low. So the value is not missing at random, which introduces a bias.

What to do? No standard recipe ...

- ▶ Throw away corrupted entries (ok if missing at random, not ok if missing not at random)
- ▶ Impute missing values in some clever way.

## Missing values (2)

- ▶ Use an algorithm that can cope with missing values (example: if you use a feature vector to describe your data, and one entry in the vector is missing, this is a problem if you want to compute a scalar product).

# Defining features, similarities, distance functions

# Choice of features

- ▶ Choose reasonable features if you can ☺
  - ▶ If you want to classify texts into different topics, a bag of words seems reasonable. A “bag of letters” would not be reasonable.
- ▶ Don't be shy with including features. If in doubt, always include a questionable feature. Many supervised algorithms like SVMs are good at identifying useful features and can work with thousands of features (not always, but often).
- ▶ Choosing good features can be a very tough problem (in fact, half of the computer vision community does research on what are good features for image classification. Famous are the SIFT features: they are scale and rotation invariant local features on images (google it if you are interested)).

# Dealing with categorial features

- ▶ Assume you have a feature that is not a numerical value but a "category".  
Example: you want to describe books, the categories are "detective story", "novel", "children's book", ....
- ▶ The naive attempt would be to say "detective story" = 1, "novel" = 2, "children's book" = 3, ...
- ▶ But in many cases this is a bad idea. Note that the numerical values 1, 2, 3 suggest that a detective story is closer to novel than to a children's book (as similarity between feature vectors we use the scalar product, and it implicitly encodes this kind of intuition). **MAKE SURE YOU UNDERSTAND THIS POINT!**

# Dealing with categorial features (2)

- ▶ Alternative: use binary encodings:  
For each category, you introduce one yes/no feature.

Example:

Feature 1: is it a detective story? (0 or 1)

Feature 2: is it a novel? (0 or 1)

And so on.

# Sparse feature vectors

- ▶ Sometimes the feature vector is very sparse (e.g. in a bag of words approach).
- ▶ You might use dimensionality reduction first (eg., SVD) or cluster features into meaningful groups.
- ▶ You might also build disjunctive features (replace two features by one feature that contains the sum of the two, or a logical disjunction in case of categorial features).
- ▶ Sometimes (but not very often) people also use a hash function to map the original feature vector to a condensed representation. This is called “feature hashing”.

# Defining a similarity / kernel / distance function

# Importance of a good choice

- ▶ One of the basic principles of all the learning algorithms we have seen is that points which are “similar” or “close” tend to belong to the same class (have a similar label).
- ▶ The notion of “similarity” or “closeness” has to be given to the ML algorithm as input.
- ▶ This is the place where a lot of the prior knowledge you have about your problem is made accessible to your algorithm.
- ▶ The choice of the similarity function / kernel / distance function is crucial! If this function is not well-suited, there is nothing you can save by the best learning algorithm in the world

# Defining a similarity /kernel / distance function

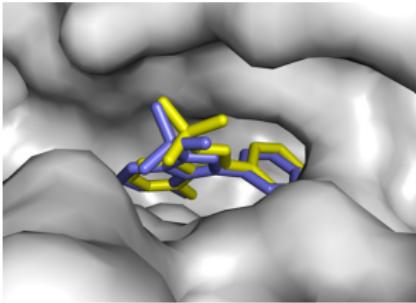
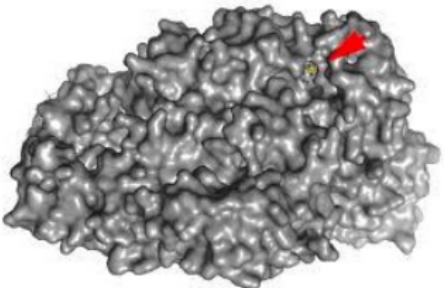
**In a kernel approach (or distance or similarity based approach):**

- ▶ Any prior knowledge you have has to go into the similarity / kernel / distance function.
- ▶ Keep in mind: ML tries to classify such that points that are similar / close tend to end up in the same classes.

# Defining a similarity /kernel / distance function (2)

Example from bioinformatics:

- We want to classify proteins as “drugable” or “not drugable” (this means whether they have the potential to be used in a medical context).



Images: BiochemLabSolutions.com

# Defining a similarity /kernel / distance function (3)

- ▶ As similarity function between proteins we use a score produced by the BLAST sequence alignment algorithm. The intuition is that proteins with a similar primary sequence have a similar function.
- ▶ Alternatively, we might take into account that the important information is not so much the primary sequence but the 3d structure of the protein. In this case, we need to define a similarity function that takes the 3d structure of the protein into account.

# Defining a similarity /kernel / distance function (4)

Example digit classification:

- ▶ To define the similarity between two hand-written digits, we should ignore the color of the pen in which the digit has been written.

# Reducing the number of training points?

# Data compression

- ▶ Sometimes the data set we have is too big to be processed (here I refer to the number  $n$  of points, not to their dimension  $d$ ).
- ▶ Ideally, we would like to have a smaller data set, but we don't want to lose much information.
- ▶ Most importantly, by reducing the data set we do not want to introduce substantial artefacts or distortion to the data.

Two approaches that are both used commonly:

# Data compression (2)

## Subsampling

- ▶ randomly select  $n' < n$  points from the original data set and just train on the smaller set.
- ▶ One might want to repeat this procedure several times and average in the end.
- ▶ Here the distribution of the data is the same as before, but the variance of the result will be higher (simply because we have a lower sample size).

# Data compression (3)

## Vector quantization:

- ▶ Run an algorithm to select a set of  $n'$  “representative points” from the original data set.
- ▶ A common approach is to use the  $k$ -means algorithm with  $k = n'$  for this task.
- ▶ This approach introduces a change to the data distribution, the centers selected by  $k$ -means no longer follow the original distribution of the data.

One indicator why this is the case: It can be very efficient (from the  $k$ -means point of view) to just put a small number of representatives in high-density regions, but cover the outliers by a representative each.

# Unsupervised dimensionality reduction

# Dimensionality reduction

Dimensionality reduction is a very useful preprocessing step if the dimensionality of the data is high. Unless one removes too many dimension, it very often helps to improve classification accuracy (intuitively, we remove noise from the data).

There are two considerably different approaches to this problem:

- ▶ unsupervised dimensionality reduction (see below)
- ▶ supervised dimensionality reduction. This is usually called “feature selection”, see later today.

## Dimensionality reduction (2)

Unsupervised dimensionality reduction: we have seen two algorithms so far:

- ▶ (kernel) PCA / SVD. This is THE standard approach in this context.
- ▶ Isomap. Is usually not so much used for data preprocessing, but more for unsupervised learning on its own.

There exist many, many more methods.

But be aware: it can always happen that your (unsupervised) dimensionality reduction destroys important information, see the discussion in the PCA section for examples.

# Data standardization

# Data standardization

It is very common to use data standardization (centering and normalizing), and almost never hurts.

We have already discussed standardization in the context of (kernel) PCA:

- ▶ Center the data points
- ▶ Normalize rows or columns of your data matrix, see the discussion in the context of kernel PCA.

# Clustering

# Clustering

Sometimes it makes sense to understand the cluster structure of your data.

- ▶ Many small clusters to reduce the size of your data set (vector quantization).
- ▶ Few big clusters of data. You might then treat the classes differently, or use learning algorithms that exploit the cluster structure. Often, this is simply used as an explorative preprocessing step, in particular as a sanity check for your data (or to discover unknown aspects of your data).

# Setting up the learning problem

## Choice of a loss / risk function

# Weighted loss and risk functions

The default loss function for classification is the 0-1-loss.

However, there are a couple of standard cases where we need to incorporate some weights into loss functions.

## Unbalanced classes.

- ▶ Assume your training data consists of 1000 points, but just 10 of them are from class +1, and the other 990 from the class -1.
- ▶ If you now use a standard loss function, it is very likely that the best classifier is the one that simply predicts -1 everywhere. WHY?
- ▶ To circumvent this problem you have to reweight the loss function such that training errors that mispredict points of class 1 get much more punished than the other way round.

## Weighted loss and risk functions (2)

- As an example, you can define the training error (empirical error) as

$$R_n(f) = \sum_{i: Y_i=1} \ell(X_i, Y_i, f(X_i)) + \gamma \cdot \sum_{i: Y_i=-1} \ell(X_i, Y_i, f(X_i))$$

where  $\gamma$  is a parameter. High  $\gamma$  means that errors in class -1 get punished much more severely.

# Weighted loss and risk functions (3)

## Importance-weighted classification.

- ▶ It also might be the case that a correct result is very important for some training points, but not so important for other training points.

Example: you might care much more about “good customers” than about not so good customers. So you might be more careful with annoying some customers by adds than others.

- ▶ In such cases you can assign different weights  $w_i$  to the training points and then define the empirical error as

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

# Weighted loss and risk functions (4)

## Cost sensitive classification.

- ▶ In many applications, the kind of errors we make are not symmetric.
- ▶ Example: spam classification
  - ▶ If a spam mail ends up in your inbox, no much harm done.
  - ▶ But if an important non-spam email ends in your spam folder, this can be a disaster.
- ▶ Here the loss function itself contains weights, that is

$$\ell(X_i, Y_i, f(X_i)) = \begin{cases} 1 & \iff Y_i = \text{spam}, f(X_i) = \text{ham} \\ \gamma & \iff Y_i = \text{ham}, f(X_i) = \text{spam} \end{cases}$$

where  $\gamma$  is a parameter (say,  $10^3$ ).

# Designing your own loss function

Discrete loss functions like the weighted 0-1-loss are NP hard to optimize. Instead, most ML algorithms use a different loss function (called surrogate loss).

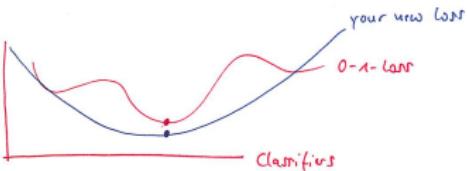
If you want to design such a surrogate loss function, here are some considerations you should take into account (this is not part of the standard ML processing chain, and often rather difficult, but I wanted to mention the keywords):

- ▶ Ideally, your new loss function should be convex, otherwise you are going to have a hard time optimizing it.

## Designing your own loss function (2)

- ▶ There are two (slightly different) properties of loss functions: being **proper** and **classification calibrated**. On a very high level, both definitions try to ensure that for any classifier  $f$  and the Bayes classifier  $f^*$  we have

$$f \neq f^* \implies R(f) > R(f^*)$$



- ▶ Formally, the definitions of “proper” and “calibrated” are not exactly the same but are in a close relationship.

## Designing your own loss function (3)

- ▶ All this is pretty recent work, and the design of loss and risk functions is a very active field of research, see for example the publications by Peter Bartlett (now in Brisbane) or Bob Williamson (Australian National University) .

# Training

# Multi-class approaches

Assume we are given a classification problem with  $K$  classes, labeled  $1, \dots, K$ . Further, assume that the ordering of these labels is not important (“closeness” of the labels does not have any meaning).

## One-versus-all

- ▶ For each  $k \in \{1, \dots, K\}$  we train binary classification problems where the first class contains all points of class  $k$  and the other class all remaining points.
- ▶ We then get  $K$  classifiers  $f_k$ . The final decision for a class  $k$  is then the one which gives the highest score to class  $k$ :

$$f_{final}(x) = \operatorname{argmax}_{k=1, \dots, K} f_k(x)$$

## One-vs-one.

## Multi-class approaches (2)

- ▶ We train each class against each other class, this then gives  $K(K - 1)/2$  classifiers  $f_{kl}$  in the end.
- ▶ The final classification is then by majority vote.

$$f_{final}(x) = \operatorname{argmax}_{l=1, \dots, K} \sum_{k=1, \dots, l-1, l+1, \dots, K} \mathbb{1}_{f_{lk}(x) > 0}$$

### Comments.

- ▶ One can prove that both approaches lead to Bayes consistent classifiers if the underlying binary classifiers are Bayes consistent.
- ▶ There also exist more complicated schemes, but in practice they don't really perform better than the simple ones.
- ▶ Multi-class scenarios are also an active field of research.

# Selecting parameters by cross validation

## Cross validation - purpose

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

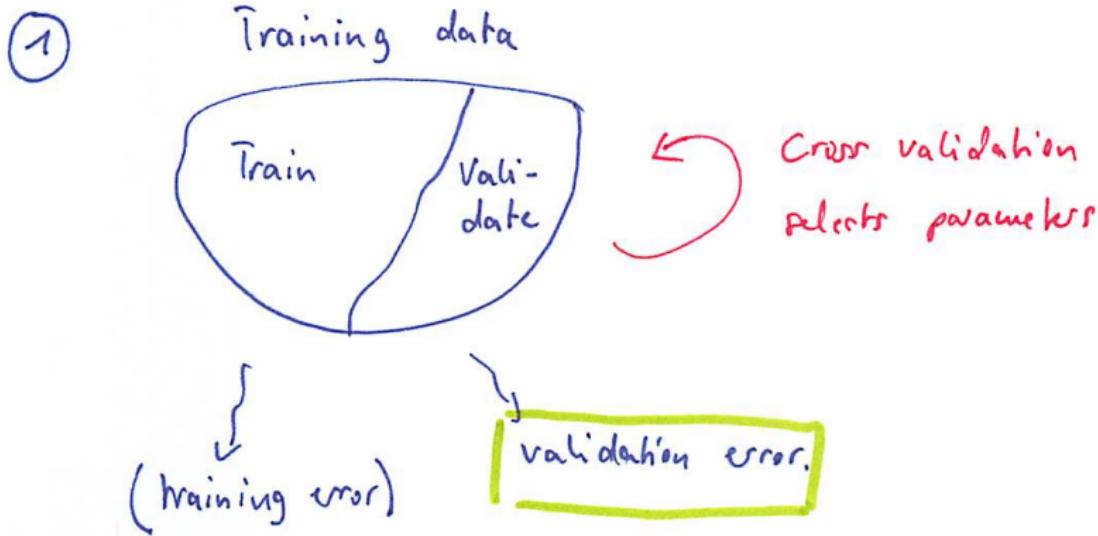
- ▶ Regularization parameter in ridge regression or Lasso
- ▶ Parameter  $C$  of the SVM
- ▶ Parameter  $\sigma$  in the Gaussian kernel
- ▶ Number of principle components in PCA
- ▶ But you also might want to figure out whether certain design choices make sense, for example whether it is useful to remove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

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

# K-fold cross validation

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

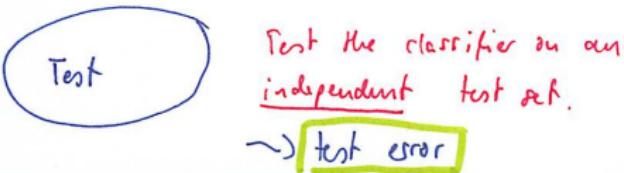
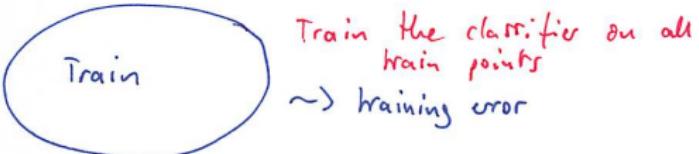
# K-fold cross validation (2)



# K-fold cross validation (3)

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

② With the parameters identified by cross validation:



## K-fold cross validation (4)

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

# K-fold cross validation (5)

What number of folds  $K$ ?

Not so critical, often people use 5 or 10.

# K-fold cross validation (6)

How to choose the set  $S$ ?

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

## K-fold cross validation (7)

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

# Advantages and disadvantages

Disadvantages of cross validation:

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

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

## Advantages and disadvantages (2)

Advantages:

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

Always, always, always do cross validation!!! Make sure the final test set is never touched while training (retraining for improving the test error is not allowed, then the data is spoiled).

# Feature selection

## Literature:

- ▶ Text book: Shalev-Shwartz/Ben-David, Chapter 25
- ▶ An great overview paper: *Guyon, Elisseeff: Introduction to variable and feature selection. JMLR, 2003.*
- ▶ A whole book (based on a feature selection challenge): Guyon et. al: Feature Extraction: Foundations and Applications. Springer, 2006.

# Feature selection problem

- ▶ Given high-dimensional data vectors.
- ▶ Goal is to reduce the number of features, but in a supervised way.
- ▶ We don't want to lose (much) classification accuracy.
- ▶ Reasons for doing this:
  - ▶ Curse of dimensionality
  - ▶ Computational reasons
  - ▶ We simply might want to "understand" our classifier. For example, in a medical context people don't want to have a black-box classifier that simply suggests a certain treatment. They want to know what are the reasons for this choice.

# Feature selection, first thoughts

- ▶ Ideally, what we would like to do is to take all subsets of features and figure out which of them leads to the best classifier.
- ▶ However, this is not a good idea. WHY?

# Filter methods

General procedure:

- ▶ Take the training data (points and labels)
- ▶ Try to identify “good features” just based on this data (see below for how this works).
- ▶ Then train your classifiers with the good features only.

General properties:

- ▶ Faster than wrapper methods, less overfitting than wrapper methods
- ▶ But independent of the actual classifier we use, which might not be such a good idea

# Filter methods (2)

Scores:

- ▶ Usually, filter methods try to compute “dependency scores” between sets of features and labels.
- ▶ Example:
  - ▶ Assume we want to classify mushrooms as “edible” or “poisonous”. We collect many features: size, smell, color, shape, ... , and also have the true labels in our training set.
  - ▶ If we now figure out that mushrooms are edible if and only if they are brown then we just need the color feature for perfect prediction.
- ▶ Note: this kind of feature selection is supervised (we need the label information)!

## Filter methods (3)

- ▶ To assess how “indicative” a subset of features is for a given labels, there exist many scores:
  - ▶ based on correlation
  - ▶ based on Fisher information
  - ▶ based on information theoretic measures such as mutual information

# Filter methods (4)

Sequential forward selection:

- ▶ Start with an empty set.
- ▶ Then, one after another, add “the best” of the remaining features, according to some score.
- ▶ Do this as long as a second, overall score significantly improves by adding features. Then stop.

Drawbacks:

- ▶ it can happen that two features are just indicative if they are both in the set of features (but each alone is not very indicative). But the naive sequential method might miss this.
- ▶ You can never “undo” a choice.

## Filter methods (5)

Sequential backward selection:

Analogous, just start with all features and keep on removing “unimportant ones”.

## Filter methods (6)

More complicated procedures, for example based on branch and bound methods:

try to search through the tree of all feature subsets, but just evaluating few of them ... difficult and used seldom.

# Wrapper methods

As opposed to filter methods, the wrapper methods repeatedly train the actual learning algorithm we want to use.

- ▶ Train the classifier with different sets of features and compute the cross validation error.
- ▶ To select the final set of features, use the “smallest set” that still produces a good cross validation error.

Advantage: We only select features if they are really useful for the actual algorithm we use.

Disadvantage:

- ▶ Computationally very expensive (we have to retrain the classifier over and over again).

## Wrapper methods (2)

- ▶ Very prone to overfitting: one can interpret the feature selection method as a very large blowup of the hypothesis space, so overfitting happens easily.

# Feature selection checklist from Guyon / Elisseeff 2003

1. Do you have domain knowledge? If yes, construct a better set of ad hoc features.
2. Are your features commensurate? If no, consider normalizing them.
3. Do you suspect interdependence of features? If yes, expand your feature set by constructing conjunctive features or products of features, as much as your computer resources allow you.
4. Do you need to prune the input variables (e.g. for cost, speed or data understanding reasons)? If no, construct disjunctive features or weighted sums of features (e.g. by clustering or matrix factorization, see Section 5).

## Feature selection checklist from Guyon / Elisseeff 2003 (2)

5. Do you need to assess features individually (e.g. to understand their influence on the system or because their number is so large that you need to do a first filtering)? If yes, use a variable ranking method (Section 2 and Section 7.2); else, do it anyway to get baseline results.
6. Do you need a predictor? If no, stop.
7. Do you suspect your data is “dirty” (has a few meaningless input patterns and/or noisy outputs or wrong class labels)? If yes, detect the outlier examples using the top ranking variables obtained in step 5 as representation; check and/or discard them.

# Feature selection checklist from Guyon / Elisseeff 2003 (3)

8. Do you know what to try first? If no, use a linear predictor.  
Use a forward selection method (Section 4.2) with the “probe” method as a stopping criterion (Section 6) or use the 0-norm embedded method (Section 4.3). For comparison, following the ranking of step 5, construct a sequence of predictors of same nature using increasing subsets of features. Can you match or improve performance with a smaller subset? If yes, try a non-linear predictor with that subset.
9. Do you have new ideas, time, computational resources, and enough examples? If yes, compare several feature selection methods, including your new idea, correlation coefficients, backward selection and embedded methods (Section 4). Use linear and non-linear predictors. Select the best approach with model selection (Section 6).

## Feature selection checklist from Guyon / Elisseeff 2003 (4)

10. Do you want a stable solution (to improve performance and/or understanding)? If yes, sub-sample your data and redo your analysis for several bootstraps.

# Literature on feature selection

- ▶ Text book: Shalev-Shwartz/Ben-David, Chapter 25
- ▶ A great overview paper: Guyon, Elisseeff: *Introduction to variable and feature selection*. JMLR, 2003.
- ▶ A whole book (based on a feature selection challenge): Guyon et. al: Feature Extraction: Foundations and Applications. Springer, 2006.

# Evaluation of the results

# Counting performance measures

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

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

# Counting performance measures (2)

Confusion table:

|      |          | predicted              |                        |
|------|----------|------------------------|------------------------|
|      |          | positive               | negative               |
| true | positive | true positive<br>$tp$  | false negative<br>$fn$ |
|      | negative | false positive<br>$fp$ | true negatives<br>$tn$ |

$\sum =: P$  positive examples

$\sum =: N$  negative examples

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

## Counting performance measures (3)

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

## Counting performance measures (4)

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

## Counting performance measures (5)

If the classes are highly unbalanced, one sometimes uses:

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

## Counting performance measures (6)

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

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

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

# ROC and AUC

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

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

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

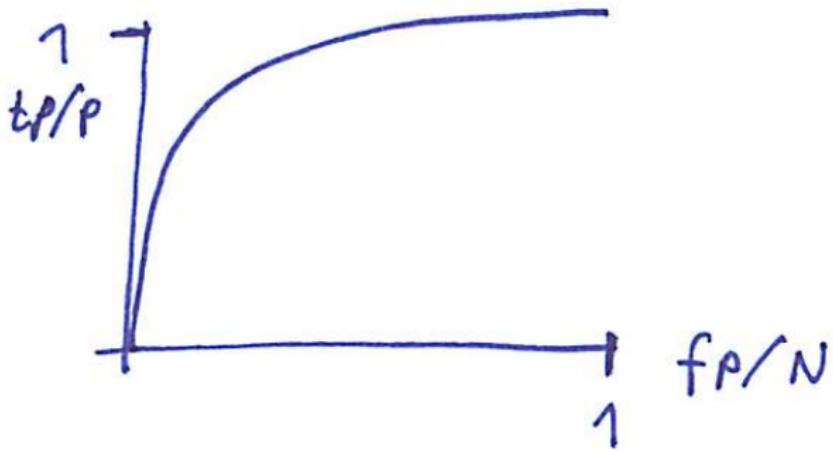
Instead, one is interested in true positives and false positives. To be able judge classifiers based on both criteria simultaneously, we can now use ROC curves.

# ROC and AUC (2)

## ROC (=Receiver-operator characteristic) curve:

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

## ROC and AUC (3)

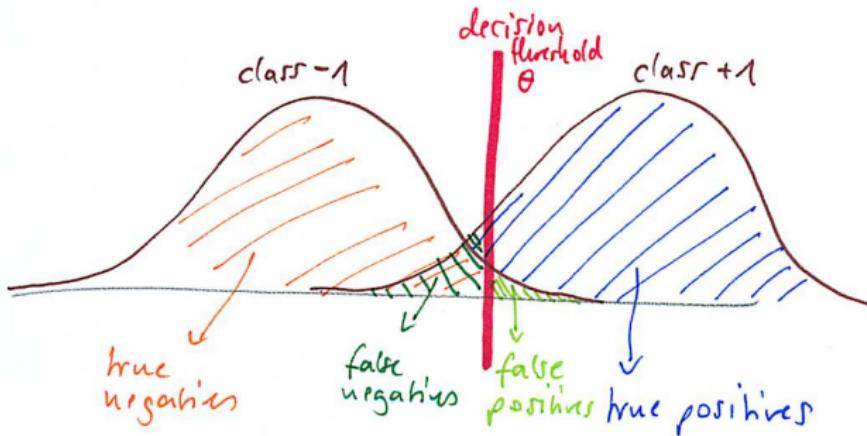


Note: indeed,

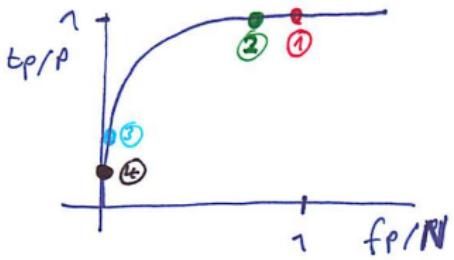
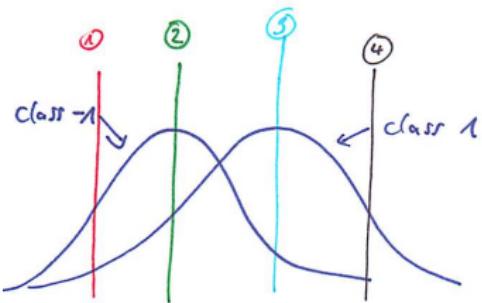
- $tp/P \in [0, 1]$
- $fp/N \in [0, 1]$

# ROC and AUC (4)

Intuition with normal distributions:

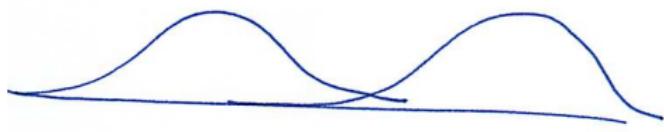


# ROC and AUC (5)

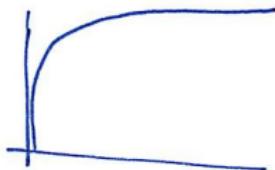


# ROC and AUC (6)

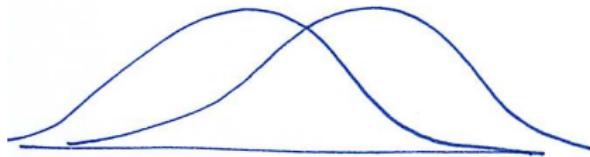
"good" Data



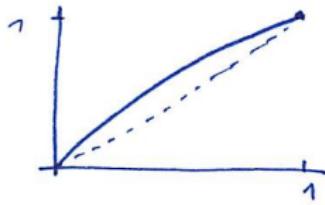
ROC



"bad" Data



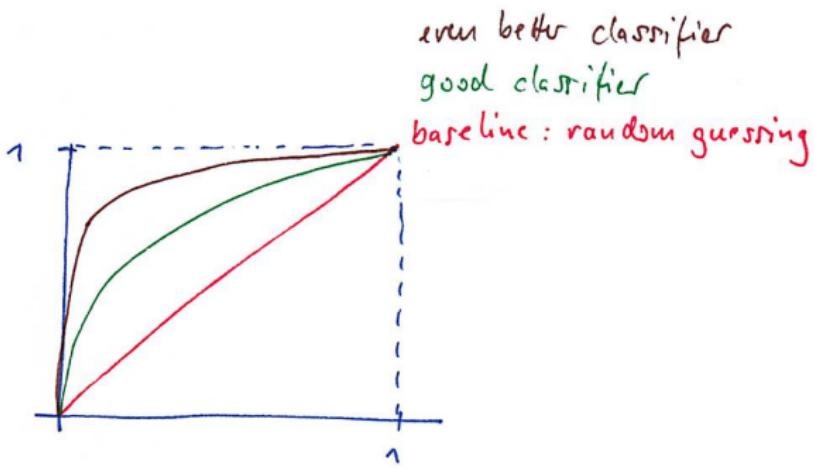
ROC



# ROC and AUC (7)

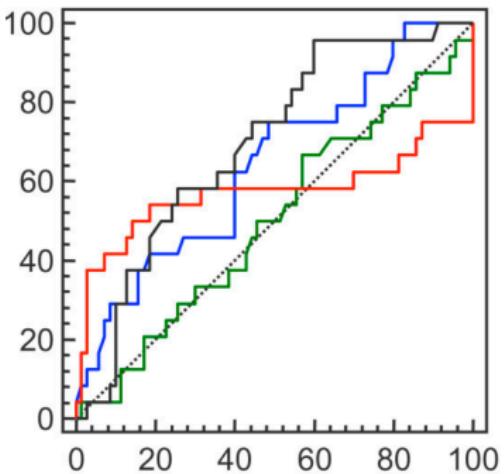
ROC for comparing classifiers:

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



## ROC and AUC (8)

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

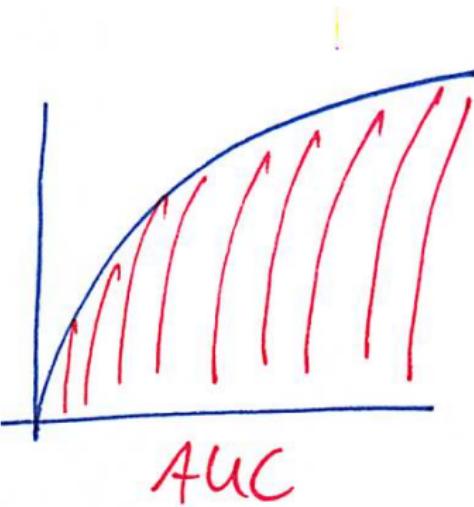


- In this case, you might still be able to say in what parameter range one classifier is better than the other.
- Or you might want to use AUC.

# ROC and AUC (9)

## AUC (Area under the ROC curve):

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



# ROC and AUC (10)

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

# Multi-class performance measures

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

# Comparing many classifiers

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

# Comparing many classifiers (2)

Table 1: Average classification error rates.

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

## Comparing many classifiers (3)

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

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

# Statistical tests for comparing classifiers

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

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

Permutation test:

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

# Statistical tests for comparing classifiers (2)

Comment:

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

## Some critical remarks

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

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

## Some critical remarks (2)

And one more (same source):

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

## Some critical remarks (3)

If you want to compare classifiers:

- ▶ Always try to compare them *for a particular task* (the “best classifier” does not exist, see no free lunch theorem).
- ▶ Always test on a variety of data sets, under many different conditions, on many different data sets.
- ▶ Be fair: try to implement all classifiers in the best way you can. If available, use implementations by the people who invented the algorithms (often a considerable amount of works goes into fine-tuning a classifier).
- ▶ Most people won’t believe you if you say that your classifier “consistently outperforms” the other classifiers.

## Some critical remarks (4)

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

## Some critical remarks (5)

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

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

# Some critical remarks (6)

Finally:

- ▶ Eventually, some of the very good algorithms tend to get identified in the community (SVM, spectral clustering, ... ), simply because many people get positive experiences by using them (but others might go completely unnoticed).
- ▶ Machine learning challenges:
  - ▶ People put a particular machine learning problem online (data and some evaluation protocol). Then everybody can commit his/her solutions. The winner gets a prize (or just the fame).
  - ▶ For examples see [www.kaggle.com](http://www.kaggle.com)

## Some references

There is lots of research on how to compare classifiers.

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

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

## Some references (2)

A couple of other references:

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

# General guidelines for attacking ML problems

## Key steps in the processing chain

# The key steps

As we have seen:

- ▶ Machine learning is not about applying one given algorithm to your data set.
- ▶ It requires a lot of tuning, playing, trial and error.
- ▶ If your problem is easy, simple approaches might work.
- ▶ If your problem is difficult, you might not be successful if you don't use a sophisticated processing chain.

## The key steps (2)

Here is the list of what I consider the minimal things you should always perform:

- ▶ Data preprocessing:
  - ▶ Figure out whether your training data has systematic biases.
  - ▶ Standardize your features
  - ▶ If your data is high-dimensional, try unsupervised dimensionality reduction (PCA) and supervised feature selection.
- ▶ The learning approach:
  - ▶ Consider to incorporate weights in your loss function if your classes are unbalanced.
  - ▶ Use regularization to prevent overfitting.
  - ▶ Always use cross-validation to set your parameters / decide on design principles.
- ▶ Make sure you run the final test on an independent set!

# High-level guidelines and principles

# High-level guidelines

Try not to loose sight of the following, very general (and abstract) principles:

- ▶ Try to incorporate any prior knowledge about your data into the process of learning.
- ▶ Try to understand the inductive bias used by your learning algorithm. (Remember, learning is impossible without an inductive bias). Is it appropriate for your problem?
- ▶ When solving a problem of interest, do not solve a more general problem as an intermediate step. (V. Vapnik).
- ▶ Play, play, play!

# Online learning

# Online learning

Literature:

On the level of text books:

- ▶ Chapter 7 in Mohri/Rostamizadeh/Tawalkar
- ▶ Chapter 21 in Shalev-Shwartz/Ben-David, Textbook level introduction.

Research level, focus on theory:

- ▶ Cesa-Bianchi, Lugosi: Prediction, learning and games. A comprehensive research-level book, focus is on theory.
- ▶ Lecture notes by Sasha Rakhlin, focus is on theory.

# Warmup

# Online learning: intuition

Consider the example of spam classification:

- ▶ Training examples arrive in a stream, one after the other.
- ▶ At each point in time, we have a current working model by which we can predict the label of the new instance (spam or not-spam).
- ▶ The goal is to make as few mistakes as possible.

But the circumstances are quite different than in the batch setting:

- ▶ The distribution over these examples can change over time
- ▶ We play against an adversary who tries to exploit all the weaknesses of our current predictions.

# Online learning: intuition (2)

Online learning, a bit more formally:

- ▶ We maintain a “model”  $f_t$  by which we predict labels.
- ▶ At each point  $t$  in time, we first observe a point  $x_t$ .
- ▶ Then we have to predict what we believe is the correct label of  $x_t$ , by applying our current model function:  $p_t := f_t(x_t)$ .
- ▶ Then we get to see the true label  $y_t$  of this point
- ▶ We incur an error if  $p_t \neq y_t$ .

# No assumptions on input sequence

To accommodate all the different possibilities for the input sequences (changing over time, adversary, etc) we proceed as follows:

We do not make ANY assumption on how the sequence is being generated.

There could be an adversary who, at each point in time, comes up with a “difficult” example  $(X_i, Y_i)$ . He can choose  $X_i$  and  $Y_i$  completely as he wishes (that is, he can also give wrong labels).

?!?!??!?!??!?!??!

## No assumptions on input sequence (2)

Let's start with a couple of examples:

Example: Random outcomes

- ▶ Adversary picks  $X_i$  uniformly at random from  $[0, 1]$
- ▶ Then he throws a coin and gives us the corresponding result as  $Y_i$ .
- ▶ In this scenario there won't be any way that we can predict the label better than random guessing.

## No assumptions on input sequence (3)

Example: fictitious play

- ▶ The adversary is allowed to choose the label of the new point.
- ▶ We assume that he knows our learning strategy, that is he know what is our current function  $f_t$  that we use to predict.
- ▶ He now chooses some  $X_t$ , finds out what we would have predicted by  $f_t(X_t)$ , and chooses  $Y_t$  exactly as the opposite label.
- ▶ **We are going to predict the wrong label at every single step!**

Looks pretty hopeless ... but looking at it the right way shows that it isn't ...

# Minimizing regret

What would be a good way to measure the success of an online learning algorithm? Above we have already seen that the absolute number of errors is not a good measure for how successful an algorithm is.

Assume that we can choose or model function  $f_t$  from some function class  $\mathcal{F}$ . We now define the **regret with respect to  $\mathcal{F}$** :

$$\text{Regret}(T) := \sup_{f \in \mathcal{F}} \sup_{(x_1, y_1), \dots, (x_T, y_T)} \left| \sum_{t=1}^T |p_t - y_t| - \sum_{t=1}^T |f(x_t) - y_t| \right|$$

In words, we measure how much we regret not having used another predictor in our class  $\mathcal{F}$ .

## Minimizing regret (2)

Consider again the two examples from above:

Example 1 (random outcome):

- ▶ No function in the class would be better than what we could do, so the regret would be (close to) 0. This makes sense and is the result we would expect: if there is nothing we can learn, then we should not be punished for not being able to learn.

Example 2 (fictitious play):

- ▶ In the example where the adversary chooses a label to fool us, the regret can be up to  $T$  (depending on the function class), and there is nothing we can do against it.
- ▶ This is not what we would hope to get — after all, we should be as clever as the adversary, so why is he allowed to fool us?

## Minimizing regret (3)

- ▶ To sidestep the latter problem, we introduce one more component: we allow the learner to randomize his predictions.

## Minimizing regret (4)

Randomized learner:

- ▶ At each point in time, the learner maintains a probability distribution over the function space  $\mathcal{F}$ .
- ▶ When he is asked to make a prediction, he randomly chooses a strategy according to his internal probability.
- ▶ The adversary is allowed to know the current probability distribution (in order to be able to act adversarially), but he does not have any influence on the randomness that generates our label.

# Prediction with expert advice, weighted majority algorithm

Literature: Chapter 21 in Shalev-Shwartz/Ben-David

# Prediction with expert advice

- ▶ We are given a (finite) class  $\mathcal{F}$  of  $d$  different prediction functions. The functions are called “*experts*”.
- ▶ Your job is to find out how to combine the opinion of all experts to extract a good decision.

Example applications:

- ▶ Ask different experts for an opinion about whether to buy or sell stock at the stock market
- ▶ Consider different wheater forecast services and figure out what the best prediciton is

# Deterministic weighted majority

First idea:

- ▶ We start with all experts having the same weight.
- ▶ We predict the label of the new point by the weighted majority among our experts.
- ▶ If an expert makes a mistake, we reduce its weight by a factor of  $1/2$ .

Intuition:

- ▶ Assume there exists one expert that is very good.
- ▶ All other experts (the ones that predict worse than the best expert) get exponentially smaller than the best one.
- ▶ So in the long run, most of the weight will be accumulated by the best expert.
- ▶ So in the long run, our prediction will be close to the one by the best expert.

# Deterministic weighted majority (2)

Weighted majority algorithm ( $N$  denotes number of experts,  $\beta$  update factor):

WEIGHTED-MAJORITY( $N$ )

```
1  for  $i \leftarrow 1$  to  $N$  do
2       $w_{1,i} \leftarrow 1$ 
3  for  $t \leftarrow 1$  to  $T$  do
4      RECEIVE( $x_t$ )
5      if  $\sum_{i: y_{t,i}=1} w_{t,i} \geq \sum_{i: y_{t,i}=0} w_{t,i}$  then
6           $\hat{y}_t \leftarrow 1$ 
7      else  $\hat{y}_t \leftarrow 0$ 
8      RECEIVE( $y_t$ )
9      if  $(\hat{y}_t \neq y_t)$  then
10         for  $i \leftarrow 1$  to  $N$  do
11             if  $(y_{t,i} \neq y_t)$  then
12                  $w_{t+1,i} \leftarrow \beta w_{t,i}$ 
13             else  $w_{t+1,i} \leftarrow w_{t,i}$ 
14     return  $\mathbf{w}_{T+1}$ 
```

## Deterministic weighted majority (3)

We can give the following guarantee on the performance of the algorithm ( $\beta$  is the step size):

### Theorem 7.3

Fix  $\beta \in (0, 1)$ . Let  $m_T$  be the number of mistakes made by algorithm WM after  $T \geq 1$  rounds, and  $m_T^*$  be the number of mistakes made by the best of the  $N$  experts. Then, the following inequality holds:

$$m_T \leq \frac{\log N + m_T^* \log \frac{1}{\beta}}{\log \frac{2}{1+\beta}}. \quad (7.6)$$

Proof: based on simple counting arguments, see the book of Mohri et al.

# Deterministic weighted majority (4)

Bottom line:

- ▶ number of mistakes is about a constant times the number of mistakes of the best expert in hindsight
- ▶ This is remarkable, we don't make any assumption on the input sequence whatsoever, it can be random, adversarial, whatever you want.
- ▶ However, by adversarial fictitious play we can always construct sequences that have regret of the order  $\Theta(T)$ .

## Randomized weighted majority

The weighted majority algorithm does not take into account how “sure” our experts are, in the sense that our decision is always the same, no matter how much more weight we have for one or the other decision.

This can be improved by introducing randomization. The model is as follows:

- ▶ The learner is **allowed to randomize** his choice: at time  $t$ , the learner comes up with a probability distribution over the experts, that is vector of probabilities  $w_1^{(t)}, \dots, w_d^{(t)}$ .
- ▶ When he gets to see the point  $X_t$ , he chooses one of his experts, say  $f_i$ , according to his probability distribution, and uses this expert to predict the label  $f_i(X_t)$ .

## Randomized weighted majority (2)

- ▶ This decision leads to a loss of  $\mathbb{1}_{f_i(X_t) \neq Y_t}$ . Because the strategy is randomized, we consider the expected error:

$$\sum_{i=1}^d w_t^{(i)} \mathbb{1}_{f_i(X_t) \neq Y_t}$$

# Randomized weighted majority (3)

The following algorithm is often called the exponential weights algorithm:

## Weighted-Majority

**input:** number of experts,  $d$  ; number of rounds,  $T$

**parameter:**  $\eta = \sqrt{2 \log(d)/T}$

**initialize:**  $\tilde{\mathbf{w}}^{(1)} = (1, \dots, 1)$

**for**  $t = 1, 2, \dots$

    set  $\mathbf{w}^{(t)} = \tilde{\mathbf{w}}^{(t)}/Z_t$  where  $Z_t = \sum_i \tilde{w}_i^{(t)}$

    choose expert  $i$  at random according to  $\mathbb{P}[i] = w_i^{(t)}$

    receive costs of all experts  $\mathbf{v}_t \in [0, 1]^d$

    pay cost  $\langle \mathbf{w}^{(t)}, \mathbf{v}_t \rangle$

**update rule**  $\forall i, \tilde{w}_i^{(t+1)} = \tilde{w}_i^{(t)} e^{-\eta v_{t,i}}$

# Randomized weighted majority (4)

**Theorem 21.11.** Assuming that  $T > 2\log(d)$ , the Weighted-Majority algorithm enjoys the bound

$$\sum_{t=1}^T \langle \mathbf{w}^{(t)}, \mathbf{v}_t \rangle - \min_{i \in [d]} \sum_{t=1}^T v_{t,i} \leq \sqrt{2 \log(d) T}.$$

Proof: Skipped, based on deriving upper and lower bounds on the potential function  $W_t := \sum_{i=1}^N w_{t,i}$ . See book by Mohri et al.

Discussion:

- ▶ The number of mistakes the algorithm makes in  $T$  rounds is of the order  $\sqrt{T}$ .
- ▶ There exists a lower bound that shows that this bound is optimal — there does not exist any algorithm that achieves a better asymptotic guarantee.

## Randomized weighted majority (5)

- ▶ Formally, for this bound to hold we need to know the number  $T$  of rounds in advance (in the algorithm,  $\eta$  depends on  $T$ ). Can get around this by the “doubling trick”.
- ▶ The term  $\log(d)$  measures the “capacity” of the function class  $\mathcal{F}$ . The whole setting can be generalized to infinite function classes as well. The capacity term  $\log d$  is then replaced by the “Littlestone dimension”.
- ▶ Using concentration inequalities, the expected loss can be replaced by the actual loss.

# Outlook: Multi-armed bandits

One very important feature of the expert advice setting:  
In each round, we get full feedback on the loss that each single strategy would have produced. This is unrealistic in many scenarios.

## Example 1: routing

- ▶ You drive to work every day.
- ▶ There are  $d$  different routes you can take.
- ▶ Goal is, in the long run, to have a good strategy for selecting the route. The learning setting is as follows:
  - ▶ Each day you choose one of them, and you observe how much time it takes you.
  - ▶ But you don't get any feedback on how much time it would have taken to take any of the other routes!  $\leadsto$  **limited feedback**

# Outlook: Multi-armed bandits (2)

## Example 2: online advertising

- ▶ You have a choice of different ads to show to a user.
- ▶ After the user has seen the add, you record whether he clicked on it or not.
- ▶ However, but you don't have any feedback on what the user would have done if you had shown him a different add.

## Outlook: Multi-armed bandits (3)

The setting where you only see the outcome of the selected expert is called “multi-armed bandits” setting.

- ▶ You have a number of slot machines.
- ▶ At each time  $t$ , you can pull the arm of one machine.
- ▶ You observe the amount of money you gain.
- ▶ But you don't know what you would have gained at another machine.

Note that an interesting new aspect comes into the game:  
exploration vs. exploitation ...

We don't discuss bandits any further in this lecture.

Follow the (perturbed,regularized) leader

## Follow the leader

Here is yet another, pretty simple strategy called **follow the leader**: At each point in time, simply choose the expert that, up to now, accumulated the smallest cumulative loss.

However, this rule can easily be fooled:

- ▶ Consider a scenario of two experts. The first one always predicts 0, the other one always 1.
- ▶ The adversary plays fictitious play.
- ▶ Then each of the experts is wrong in half of the cases, but follow the leader is wrong in all of the cases, leading to a regret of size  $\Omega(T)$ .

But there is a simple trick to mend this: try to ensure that follow the leader gets a bit more stable.

... follow the regularized and perturbed leader:

## Follow the leader (2)

Follow the perturbed leader:

- ▶ Denote by  $L_{i,t}$  the cumulative loss that expert  $i$  would have incurred up to time  $t$ .
- ▶ Now add some noise  $Z_i$  to it (introduce some perturbation):

$$\tilde{L}_{i,t} := L_{i,t} + Z_{i,t}$$

- ▶ Now choose the prediction of the expert with the smallest  $\tilde{L}_{i,t}$ -value.

## Follow the leader (3)

Follow the regularized leader:

- ▶ Instead of adding a noise term, we add a regularization term to each of the cumulative losses.
- ▶ This could, for example, be the norm of the coefficients in case of linear classification.
- ▶ Makes the solutions more stable over time.

## Perceptron and winnow

Literature: book by Mohri et al.

# Online linear classification: perceptron

Consider linear classification in an online setting:

- ▶ Data points live in  $\mathbb{R}^d$ , but are received one after the other
- ▶ Goal is to maintain a hyperplane that separates the classes.

Main idea:

- ▶ You maintain a weight vector  $w$
- ▶ Whenever your prediction makes a mistake, you slightly update your weight vector:
  - ▶ If the algorithm makes an error, this means that  $y_t w_t x_t < 0$ .
  - ▶ After the update, we have  $y_t w_{t+1} x_t = y_t w_t y_t x_t + \eta \|x_t\|^2$ , so the inner product becomes “more positive”, moving the weight vector in the correct direction.

# Online linear classification: perceptron (2)

PERCEPTRON( $\mathbf{w}_0$ )

```
1    $\mathbf{w}_1 \leftarrow \mathbf{w}_0$        $\triangleright$  typically  $\mathbf{w}_0 = \mathbf{0}$ 
2   for  $t \leftarrow 1$  to  $T$  do
3       RECEIVE( $\mathbf{x}_t$ )
4        $\hat{y}_t \leftarrow \text{sgn}(\mathbf{w}_t \cdot \mathbf{x}_t)$ 
5       RECEIVE( $y_t$ )
6       if ( $\hat{y}_t \neq y_t$ ) then
7            $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t$      $\triangleright$  more generally  $\eta y_t \mathbf{x}_t$ ,  $\eta > 0$ .
8       else  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$ 
9   return  $\mathbf{w}_{T+1}$ 
```

## Online linear classification: perceptron (3)

There are many different ways in which the perceptron algorithm can be analyzed:

- ▶ It can be shown that it is equivalent to stochastic gradient descent.
- ▶ One can prove margin-like generalization bounds that describe the performance if the data can be linearly separated by a hyperplane with a certain margin.

## Online linear classification: perceptron (4)

Variants: there are many, many variants of the perceptron:

- ▶ You can kernelize it.
- ▶ You can teach it to “forget the past” (“forgetron”).
- ▶ many more ...

The perceptron is about the oldest learning algorithm that exists (Roseblatt, 1958; margin bound by Novikoff 1962; kernel version by Aizerman 1964).

# Winnow

Winnow also constructs linear classifiers. But it uses multiplicative updates rather than linear ones:

- ▶ If prediction of an expert is correct, its weight is increased.
- ▶ If prediction of an expert is incorrect, its weight gets decreased.

## Winnow (2)

WINNOW( $\eta$ )

```
1    $w_1 \leftarrow \mathbf{1}/N$ 
2   for  $t \leftarrow 1$  to  $T$  do
3       RECEIVE( $\mathbf{x}_t$ )
4        $\hat{y}_t \leftarrow \text{sgn}(\mathbf{w}_t \cdot \mathbf{x}_t)$ 
5       RECEIVE( $y_t$ )
6       if ( $\hat{y}_t \neq y_t$ ) then
7            $Z_t \leftarrow \sum_{i=1}^N w_{t,i} \exp(\eta y_t x_{t,i})$ 
8           for  $i \leftarrow 1$  to  $N$  do
9                $w_{t+1,i} \leftarrow \frac{w_{t,i} \exp(\eta y_t x_{t,i})}{Z_t}$ 
10      else  $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$ 
11  return  $\mathbf{w}_{T+1}$ 
```

The principle is reminiscent to weighted majority in the expert setting...

# History

- ▶ There is a close connection of online learning to game theory, and a number of important results have already been proved in the 1950ies (regret of order  $\sqrt{T}$ , but linear dependency on  $d$ )
- ▶ Perceptron: dates back to Rosenblatt 1958, first theoretical analysis was by Minski and Papert 1969 and Novikov 1952
- ▶ Weighted majority and its variants are due to Littlestone and Warmuth, 1989 and the following years (first analysis with  $\log d$  dependency).
- ▶ the last 10 years or so the COLT community had a strong focus on analyzing online algorithms.
- ▶ Littlestone/Warmuth 1999
- ▶ EXP3 (bandit setting breakthrough) around 2000

Still very active area of research.

# **Outlook: some of the research in our group**

# Wrap up

# Excursion: Research, publications, reviewing

# Publication culture in Computer Science

Keywords to discuss:

journals, conferences, lecture notes, arxiv, technical reports

# Publication culture in Computer Science (2)

Top conferences in Machine Learning: NIPS, ICML, COLT

Top journals in Machine Learning: JMLR, MLJ, IEEE IT

Top journal in Statistics: Annals of Statistics

# Reviewing: the process itself

... in journals:

Keywords to discuss:

- ▶ blind, double-blind,
- ▶ editor (in chief, associate)
- ▶ how the whole process works: submission, editor, associate editor, 3 reviewers, associate editor, decision (accept, minor/major revision, reject), notification

# Reviewing: the process itself (2)

... in conferences:

Keywords are:

- ▶ program chair = editor in chief
- ▶ area chair = associate editor (mainly in large conferences)
- ▶ programm committee: sometimes this means reviewer, sometimes area chair.
- ▶ Biggest issue: scaling! (NIPS 2016: 2500 submissions, 3200 reviewers, 100 area chairs)
- ▶ How the process works: submission, program chairs, bidding + paper assignment to reviewers, reviews come in, discussion among reviewers, discussion among area chairs/program chairs, decision (accept/reject).

# Points to address in a review

Two different parties are addressed in a review:

- ▶ Authors: want constructive and fair feedback about their paper
- ▶ Editors: need arguments for their decision

Points typically addressed in a review:

- ▶ **Quality.** Is the paper technically sound? Are claims well-supported? Is this a complete piece of work, or merely a position paper? Are the authors careful (and honest) about evaluating both the strengths and weaknesses of the work?
- ▶ **Clarity.** Is the paper clearly written? Is it well-organized? Does it adequately inform the reader? (A superbly written paper provides enough information for the expert reader to reproduce its results.)

## Points to address in a review (2)

- ▶ **Originality.** Are the problems or approaches new? Is this a novel combination of familiar techniques? Is it clear how this work differs from previous contributions? Is related work adequately referenced?
- ▶ **Significance.** Are the results important? Does the paper address a difficult problem in a better way than previous research? Does it advance the state of the art? Does it provide unique data, unique conclusions on existing data, or a unique theoretical or pragmatic approach?

# Points to address in a review (3)

Typical structure of a review:

- ▶ **Summarize** what you believe are the main contributions of the paper, in own words (about 1 paragraph), and summarize your opinion about the paper (few sentences)
- ▶ **Give detailed evaluation:** address the four points (quality, clarity, originality, significance), and summarize pros / cons.
- ▶ **Give minor comments** to the authors (typos, unclear formulations, parts that cannot be understood, wrong formulas, etc)
- ▶ **Private comments to the editor** (not seen by the authors). Here one can declare conflicts of interests, whether one knows the authors, how thoroughly one has done the review (eg, checked proof details).

# Example reviews

... see files ...

# Judging the quality of researchers

People (in particular, administration), always tries to invent numbers to measure quality. Per se, this is impossible, but you should know a couple of terms:

Journal:

- ▶ impact factors, bogus!!!!
- ▶ editorial board
- ▶ “standing in the community”

... of a researcher:

- ▶ citation numbers, h-index
- ▶ prizes
- ▶ is he/she in editorial boards

# Finding a PhD position

Check carefully where you go:

- ▶ Your supervisor should have published regularly during the last couple of years, in good conferences / journals.
- ▶ Ideally, the group should not be too small (just you) or too large (30 people). In the latter case, insist to find out who would supervise you, because it definitely won't be the head of the group.
- ▶ Check how much and where all the other PhD students publish.
- ▶ How many international guests the group had during the last year.
- ▶ To which conferences the people in the group go regularly. Is there travel money?
- ▶ Are there any regular activities going on? Reading groups, seminars, ...?

## Finding a PhD position (2)

- ▶ Where do the other PhD students come from? From the same university? Are there postdocs who came after their PhDs?
- ▶ How long does a PhD in that group take, usually.
- ▶ How is the supervision organized in the group?
- ▶ Find out how much freedom the people have in selecting what they want to work on.
- ▶ How much time do people have for research, what other obligations are there (teaching, project work, ...)?
- ▶ Ask in advance whether you will get the opportunity to talk to another PhD student in the group. Listen to what they say “between the lines”. Ask all the questions above to the head of the group, and once more to the PhD student.
- ▶ Ultimately, you also need to have the impression that you like the place and get along with your potential supervisor...

## Things we did not talk about

# Things that are important but we did not cover

- ▶ Many important algorithms, of course
- ▶ Probabilistic approaches (Graphical models, Bayesian learning, Gaussian processes, etc). See the book by Kevin Murphy.
- ▶ Bio-inspired approaches (neural networks, deep networks, genetic algorithms, etc).
- ▶ Reinforcement learning
- ▶ Information-theoretic approaches. See book by David MacKay.

# Further machine learning resources

# Online classes and videos

There exist a number of online lectures (MOOC = massive online open course) about machine learning, taught by some of the best machine learners in the world:

- ▶ Stanford, Caltech, New York, ...

Machine learning summer schools:

- ▶ many of them exist, see for example [www.mlss.cc](http://www.mlss.cc).
- ▶ Nearly all of them get videotaped and are available at youtube or [videolectures.net](http://videolectures.net)

Videolectures:

- ▶ Most machine learning conferences, workshops, summer schools, etc are videotaped. The videos are online, many of them at [videolectures.net](http://videolectures.net) or youtube.
- ▶ So if you are interested in a particular topic, try to find a video at [videolectures](http://videolectures.net), likelihood is high that you are going to be successful.

# Machine learning software

There exist many software tools for machine learning, from very applied and simple to more sophisticated and closer to research. I don't have a favorite one.

But be aware: using such tools only can get you that far.

- ▶ You don't have the freedom to make all the design choices you want.
- ▶ At least, try to understand what they do if you press a button...

# Machine learning research

The top conferences in machine learning (with reviewed papers):

- ▶ Neural Information Processing Systems (NIPS)s
- ▶ International Conference on Machine Learning (ICML)
- ▶ International Conference on Learning Theory (COLT)

Also good:

- ▶ AISTATS
- ▶ UAI
- ▶ KDD (more data mining oriented)

The top journals:

- ▶ Journal of machine learning research (JMLR), the flagship journal
- ▶ Machine Learning Journal (MLJ), good as well.

Want more? Plan for the next semesters ...

# Mathematical Appendix

# Recap: Probability theory

## Literature:

- ▶ In general, any book on probability theory
- ▶ On the homepage you can also find the link to a probability recap writeup for a CS course at Stanford University (written by Arian Maleki and Tom Do).

# Discrete probability theory

# Discrete probability measure

- ▶  $\Omega$  = space of “elementary events”, “sample space”.  
This space is called “discrete” if it has finitely many elements.
- ▶ “Space of events”: In the discrete case this is simply the power set  $\mathcal{P}(\Omega)$  of  $\Omega$ , that is all possible subsets of  $\Omega$ .  
(In general it is more complicated, the space of events has to be a “ $\sigma$ -algebra”).
- ▶ Probability measure:  $P : \mathcal{P}(\Omega) \rightarrow [0, 1]$  such that the following three rules are satisfied (“Axioms of Kolmogorov”)
  - ▶  $P(A) \geq 0$  for all events  $A \subset \mathcal{P}(\Omega)$
  - ▶  $P(\Omega) = 1$
  - ▶ “sigma-additivity”: Let  $S_1, S_2, \dots \subset \Omega$  be at most countably many disjoint sets. Then  $P(S_1 \cup S_2 \cup \dots) = \sum_i P(S_i)$

Note: in the discrete case, the probability measure is uniquely defined on all of  $\mathcal{P}(\Omega)$  if we know  $P(\omega)$  for all elementary events  $\omega \in \Omega$ .

# Discrete probability measure (2)

Example: throwing a die

- ▶ Elementary events:  $\{1, 2, \dots, 6\}$
- ▶ Probability of the elementary events:  
 $P(1) = P(2) = \dots = P(6) = 1/6.$
- ▶ Probabilities of all other subsets of  $\Omega$  can be computed based on the elementary events due to the sigma-additivity.

Example:  $P(1, 2, 5) = P(1) + P(2) + P(5) = 3 \cdot 1/6 = 1/2.$

# Conditional probabilities

Define the probability of event A under the condition that event B has taken place:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Example with a die: compute the probability  $P(\{3\} \mid \text{"uneven"})$ .

Solution:

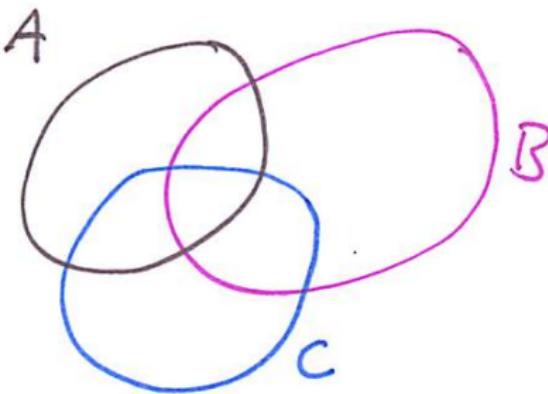
$A = \{3\}$ ,  $B = \{1, 3, 5\}$ ,  $P(A \cap B) = P(\{3\}) = 1/6$ ,  $P(B) = 1/2$ ,  
this implies  $P(\{3\} \mid \text{"uneven"}) = (1/6)/(1/2) = 1/3$ .

# Important formulas

- **Union bound.** Let  $A_1, \dots, A_k$  be any events. Then

$$P(A_1 \cup A_2 \cup \dots \cup A_k) \leq \sum_{i=1}^k P(A_i)$$

Intuitive reason:



## Important formulas (2)

- ▶ **Formula of total probability.** Let  $B_1, \dots, B_k$  be a disjoint decomposition of the probability space, that is all  $B_i$  are disjoint and  $B_1 \cup \dots \cup B_k = \Omega$ . Then:

$$P(A) = \sum_{i=1}^k P(A \cap B_i) = \sum_{i=1}^k P(A \mid B_i)P(B_i)$$

# Important formulas (3)

- ▶ Bayes' formula:

$$P(B \mid A) = \frac{P(B \cap A)}{P(A)} = \frac{P(A \mid B) \cdot P(B)}{P(A)}$$

Example:

The probability that a woman has breast cancer is 1%. The probability that the disease is detected by a mammography is 80 % (true positive rate). The probability that the test detects the disease although the patient does not have it is 9.6% (false positive rate). If a woman at age 40 is tested as positive, what is the probability that she indeed has breast cancer?

# Important formulas (4)

Define the following events:

$A :=$  mammography is positive

$B :=$  woman has breast cancer

Given:

- ▶  $P(B) = 0.01$
- ▶  $P(A | B) = 0.80$
- ▶  $P(A | \neg B) = 0.096$
- ▶ Need to compute  $P(A)$ . Here we use the total probability:

$$\begin{aligned}P(A) &= P(A|B)P(B) + P(A|\neg B)P(\neg B) \\&= 0.8 \cdot 0.01 + 0.096 \cdot 0.99 = 0.103\end{aligned}$$

Now we plug this into Bayes theorem and obtain

$$P(B|A) = \frac{0.80 \cdot 0.01}{0.103} = 0.078$$

# Random variables

A **random variable** is a function  $X : \Omega \rightarrow \mathbb{R}$ .

Example:

- ▶ We have 5 red and 5 black balls in an urn
- ▶ We draw 3 balls randomly without replacement
- ▶ Random variable  $X = \text{number of red balls we got}$

A **random variable is called discrete** if its image is discrete (it can take at most finitely many values).

## Random variables (2)

A random variable  $X : \Omega \rightarrow \mathbb{R}$  induces a probability distribution  $P_X$  on its image: for any (measurable) set  $A \subset \mathbb{R}$  we define

$$P_X(A) = P(X \in A)$$

The measure  $P_X$  is called the **distribution of the random variable**.

# Important discrete probability distributions

- ▶ **Bernoulli distribution:** we throw a biased coin once. It takes value 1 with probability  $p$  and value 0 with probability  $(1 - p)$ .
- ▶ **Binomial distribution  $B(n, p)$ :** We throw a biased coin  $n$  times independently from each other. The binomial random variable counts how often we got 1. It is defined as

$$P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$$

It has expected value  $np$  and variance  $np(1 - p)$ .

## Important discrete probability distributions (2)

- ▶ Poisson distribution  $Pois(\lambda)$ .

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

The Poisson distribution counts the occurrence of “rare events” in a fixed time interval (like radioactive decay),  $\lambda$  is the intensity parameter.

It has expected value  $\lambda$  and variance  $\lambda$ .

# Independence

Two events  $A, B$  are called independent if  
 $P(A \cap B) = P(A) \cdot P(B)$ .

Note that this implies that  $P(A \mid B) = P(A)$ .

Two random variables  $X, Y : \Omega \rightarrow \mathbb{R}$  are called independent if for all events  $A, B$  we have that

$$P(X \in A, Y \in B) = P(X \in A) \cdot P(Y \in B).$$

Example:

- ▶ Throw a coin twice.  $X$  = result of the first toss,  $Y$  = result of the second toss. These two random variables are independent.

## Independence (2)

- ▶ Throw a coin twice.  $X$  = result of the first toss,  $Y$  = sum of the two results. These two random variables are not independent.

# Expectation

For a discrete random variable  $X : \Omega \rightarrow \{r_1, \dots, r_k\}$  its expectation (mean value) is defined as

$$E(X) := \sum_{i=1}^k r_i \cdot P(\{X = r_i\})$$

Intuition: the expectation is the “average result”, where the results are weighted according to their probabilities.

Examples:

- We throw a die,  $X$  is the result. Then

$$E(X) = \sum_{i=1}^6 i \cdot \frac{1}{6} = 3.5.$$

- We throw a biased coin, heads occurs with probability  $p$ , tails with probability  $1 - p$ . We assign the random variable  $X = 1$  for heads and  $X = 0$  for tails. Then

$$E(X) = 0 \cdot (1 - p) + 1 \cdot p = p.$$

# Expectation (2)

Important formulas and properties:

- The expectation is linear: for random variables  $X_1, \dots, X_n$  and real numbers  $a_1, \dots, a_n \in \mathbb{R}$ ,

$$E\left(\sum_{i=1}^n a_i X_i\right) = \sum_{i=1}^n a_i E(X_i)$$

- Expectation and independence: If  $X, Y$  are independent, then

$$E(X \cdot Y) = E(X) \cdot E(Y).$$

# Variance

The variance of a random variable is defined as

$$\text{Var}(X) = E((X - E(X))^2) = E(X^2) - (E(X))^2$$

For a discrete random variable with possible values  $r_1, \dots, r_n$ , it is given as

$$\text{Var}(X) = \sum_{i=1}^n (r_i - E(X))^2 \cdot P(X = r_i)$$

The variance measures how much the random variable “varies” about its mean.

# Variance (2)

Example:

- ▶ We throw a biased coin, heads occurs with probability  $p$ , tails comes with probability  $1 - p$ . We assign the random variable  $X = 1$  for heads and  $X = 0$  for tails.
- ▶ We have already seen:  $E(X) = p$ .
- ▶ Now let's compute the variance:

$$\text{Var}(X) = (1 - p)^2 p + (0 - p)^2 (1 - p) = (1 - p)p$$

Important properties of the variance:

- ▶  $\text{Var}(X) \geq 0$ .
- ▶ For random variables  $X$  and scalars  $a, b \in \mathbb{R}$  we have  
$$\text{Var}(aX + b) = a^2 \text{Var}(X)$$

## Variance (3)

- If  $X, Y$  are independent random variables, then

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y).$$

# Standard deviation

The standard deviation of a random variable is just the square root of the variance:

$$\text{std}(X) = \sqrt{\text{Var}(X)}$$

# Covariance and correlation

The covariance of two real-valued random variables  $X$  and  $Y$  is defined as

$$\begin{aligned}\text{Cov}(X, Y) &= E((X - E(X))(Y - E(Y))) \\ &= E(XY) - E(X)E(Y)\end{aligned}$$

It provides (one particular) measure of how related the two random variables are: whether we can use a linear (!) function to predict one of them from the other one.

Properties:

- ▶  $\text{Cov}(X, Y) = \text{Cov}(Y, X)$
- ▶  $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y) + 2 \text{Cov}(X, Y).$
- ▶ If  $\text{Cov}(X, Y) = 0$ , the random variables are called uncorrelated.
- ▶  $X, Y$  independent  $\implies X, Y$  uncorrelated (but not vice versa)

# Covariance and correlation (2)

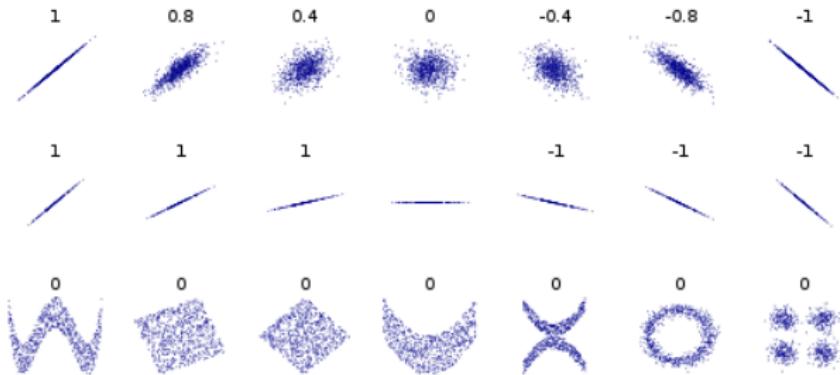
The **correlation coefficient** is defined as

$$\text{Cor}(X, Y) := \rho(X, Y) := \text{Cov}(X, Y) / (\text{std}(X)\text{std}(Y))$$

- ▶ rescales the covariance to a number between  $-1$  and  $1$
- ▶  $\rho = 1$  iff  $Y = aX + b$  for  $a > 0, b \in \mathbb{R}$
- ▶  $\rho = -1$  iff  $Y = aX + b$  for  $a < 0, b \in \mathbb{R}$

# Covariance and correlation (3)

Examples (point sets and their correlation coefficient, taken from wikipedia):



# Covariance and correlation (4)

(\*) Covariance and correlation cares about linear relationships:

- Random variables  $X, Y$
- Find  $a, b \in \mathbb{R}$  to linearly estimate  $Y$  from  $X$  by  

$$Y \approx aX + b$$
- Measure mean squared error:  $E((Y - aX - b)^2)$
- Is minimized if we choose  $a = \text{cov}(X, Y) / \text{var}(X)$ ,

$$b = E(Y) - \frac{\text{cov}(X, Y)}{\text{var}(X)} \cdot E(X)$$

- Then best linear predictor:

$$\hat{Y} = E(X) + \frac{\text{cov}(X, Y)}{\text{var}(X)} (X - E(X))$$

$\hat{Y}$  is linear fit of  $X$ , coefficient is given by covariance (correlation)

If variables are standardized to mean = 0, var = 1, the formula is:

$$\hat{Y} = \text{cov}(X, Y) \cdot X$$

(\*) Even if  $Y$  is a deterministic function of  $X$ , the covariance can be 0

## Covariance and correlation (5)

Exercise: consider a symmetric random variable  $X$  (such that the distribution of  $X$  and  $-X$  are the same), and define  $Y = X^2$ . Then  $Cov(X, Y) = 0$

## (\*) Important inequalities

- ▶ **Markov's inequality:** Let  $X$  be a non-negative random variable and  $t > 0$ . Then

$$P(X \geq t) \leq \frac{E(X)}{t}$$

- ▶ **Chebyshev's inequality:**

$$P(|X - E(X)| \geq t) \leq \frac{\text{Var}(X)}{t^2}$$

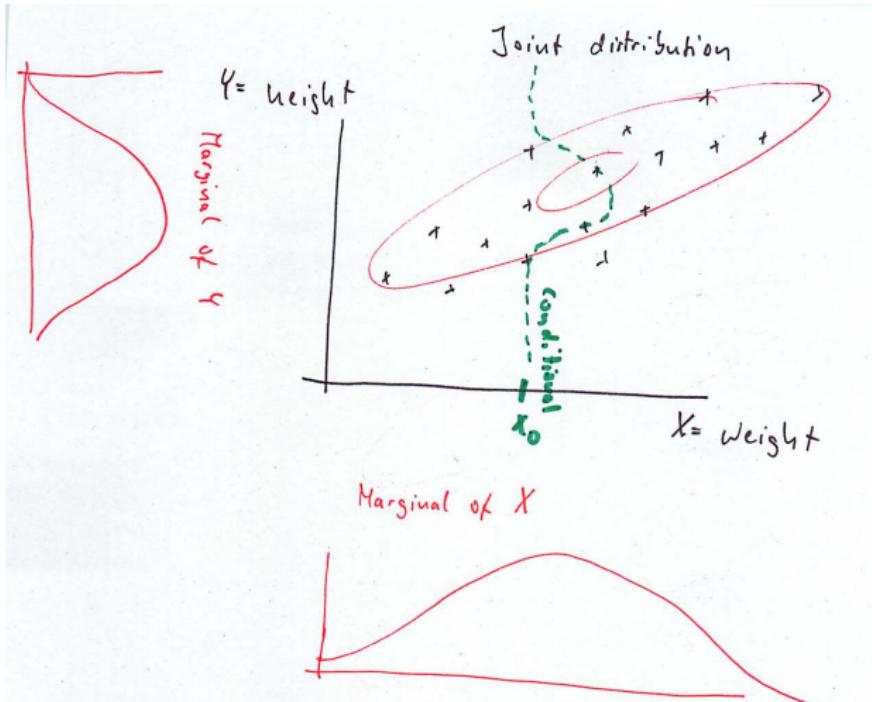
# Joint, marginal and product distribution

We want to look at the “joint distribution” of two random variables.  
Example:

- ▶ We “sample” people:  $\Omega = \text{set of all people}$
- ▶  $X = \text{their weight (in kg)}, Y = \text{their height (in cm)}$ .
- ▶ The **joint distribution** measures how the pair of random variables  $(X, Y) : \Omega \rightarrow \mathbb{R}^2$  is distributed.

# Joint, marginal and product distribution (2)

- The distribution of  $X$  is called the **marginal distribution** of  $X$ , similarly for  $Y$ .



## Joint, marginal and product distribution (3)

- ▶ Note that for given marginal distributions, there exist many joint distributions that respect the marginals!

## Joint, marginal and product distribution (4)

A particular joint distribution is the **product distribution**: it gives the joint distribution of  $X$  and  $Y$  if they are independent of each other:

- ▶ Consider two discrete random variables  $X, Y : \Omega \rightarrow \mathbb{R}$ .

- ▶ Define the product distribution

$$P((X, Y) = (x, y)) = P(X = x) \cdot P(Y = y).$$

The construction works analogously for a product of finitely many spaces.

## (\*) Conditional independence

Consider three discrete random variables  $X, Y, Z : \Omega \rightarrow \mathbb{R}$ . We say that  $X$  and  $Y$  are conditionally independent given  $Z$  if

$$\begin{aligned} & P(X \in A, Y \in B \mid Z \in C) \\ &= P(X \in A \mid Z \in C) \cdot P(Y \in B \mid Z \in C) \end{aligned}$$

for all sets  $A, B, C \subset \Omega$  with  $P(Z \in C) > 0$ .

# Variance and covariance of multivariate random variables

Variance and covariance for **1-dim random variables**  $X \in \mathbb{R}$ :

$$\text{Var}(X) = E((X - E(X))^2)$$

$$\text{Cov}(X, Y) = E\left((X - E(X))(Y - E(Y))\right)$$

They can be estimated from sample points  $x_1, \dots, x_n$  and  $y_1, \dots, y_n$  as follows:

$$\bar{x} := 1/n \sum_{i=1}^n x_i$$

$$\hat{\text{Var}}(X) = 1/n \sum_{i=1}^n (x_i - \bar{x})^2$$

# Variance and covariance of multivariate random variables (2)

$$\hat{\text{Cov}}(X, Y) = 1/n \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

Note that for variance and covariance, one sometimes normalizes the estimator  $\hat{\text{Var}}$  resp.  $\hat{\text{Cov}}$  with the factor of  $1/(n - 1)$  instead of  $1/n$  to achieve an unbiased estimate (we skip this issue here).

# Variance and covariance of multivariate random variables (3)

Now consider  $d$ -dim random variables:  $X = (X^{(1)}, \dots, X^{(d)})'$ .

The **expectation**  $E(X)$  of a  $d$ -dim random variable is the vector that contains the coordinate-wise expectations.

The **overall variance** over all  $d$  dimensions is the sum of the variances of the individual dimensions:

$$\text{Var}_d(X) = \sum_{i=1}^d E(\|X^{(i)} - E(X^{(i)})\|^2)$$

The **covariance matrix** of  $X$  is a  $d \times d$ -matrix  $C$  which encodes the covariances between the individual dimensions of the distribution:

# Variance and covariance of multivariate random variables (4)

$$C_{kl} = \text{Cov}(X^{(k)}, X^{(l)})$$

EXAMPLE: SHOE SIZE / HEIGHT / AGE OF A PERSON

# Variance and covariance of multivariate random variables (5)

These quantities can be estimated from the data:

$$\hat{\text{Var}}_d(X) = 1/n \sum_{k=1}^d \sum_{i=1}^n (x_i^{(k)} - \bar{x}^{(k)})^2$$

$$(\hat{C})_{kl} = \frac{1}{n} \sum_{i=1}^n (x_i^{(k)} - \bar{x}^{(k)})(x_i^{(l)} - \bar{x}^{(l)})$$

- $\hat{C}$  is called the empirical covariance matrix or the sample covariance matrix.

# Variance and covariance of multivariate random variables (6)

- If the data points are centered, and we define matrix  $\mathbf{X}$  containing the points as rows, then the empirical covariance matrix  $\hat{C}$  coincides with  $\mathbf{X}'\mathbf{X}$  because

$$C_{kl} = \sum_{i=1}^n X_i^{(k)} X_i^{(l)} = (\mathbf{X}'\mathbf{X})_{kl}$$

- In the following, we often drop the “hat” and the word “empirical” ...

## (\*) Conditional expectation

Example:

- ▶  $X, Y$  two independent throws of a die,  $Z = X + Y$ .
- ▶ Want to compute the expectation of  $Z$  under the condition that  $X$  was 3.
- ▶ We write  $E(Z \mid X = 3)$

If we don't fix the outcome value of  $X$ , then we write  $E(Z \mid X)$ , this is a random variable (because we don't know the random outcome of  $X$ ).

Formally, this is a pretty complicated mathematical object. For those who have not seen it before, we just treat it in an intuitive manner.

## Continuous probability theory

Probability theory gets more complicated once we go beyond the discrete regime. In this class, we try to keep it on a somewhat intuitive level.

# Density and cumulative distribution

Consider a random variable  $X : \Omega \rightarrow \mathbb{R}$ . We say that  $X$  has **density function**  $p : \mathbb{R} \rightarrow \mathbb{R}$  if for all (measurable) subsets  $A \subset \mathbb{R}$  we have

$$P(X \in A) = \int_A p(x)dx$$



# Density and cumulative distribution (2)

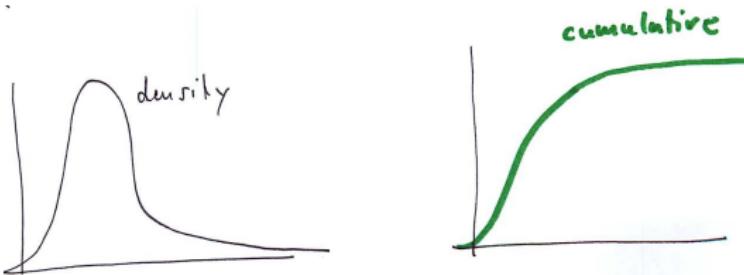
- ▶ Intuitively, a density is something like a “continuous histogram”.
- ▶ Sometimes the density is abbreviated as “pdf” (“probability density function”) in the literature.
- ▶ Density functions are always non-negative and integrate to 1. They don't have to be continuous.
- ▶ Not every random variable can be described by a density, but in this course we won't discuss this.

# (\*) Cumulative distribution function

A real-valued random variable can always be described by its **cumulative distribution function** (sometimes abbreviated as “cdf” in the literature).

For a random variable  $X : \Omega \rightarrow \mathbb{R}$  it is defined as

$$g : \mathbb{R} \rightarrow \mathbb{R}, \quad g(x) = P(X \leq x)$$



# Uniform distribution

The uniform distribution on  $[0, 1]$ : for  $0 \leq a < b \leq 1$  we define

$$P(X \in [a, b]) = b - a$$

Its density is constant.

# Normal distribution (univariate)

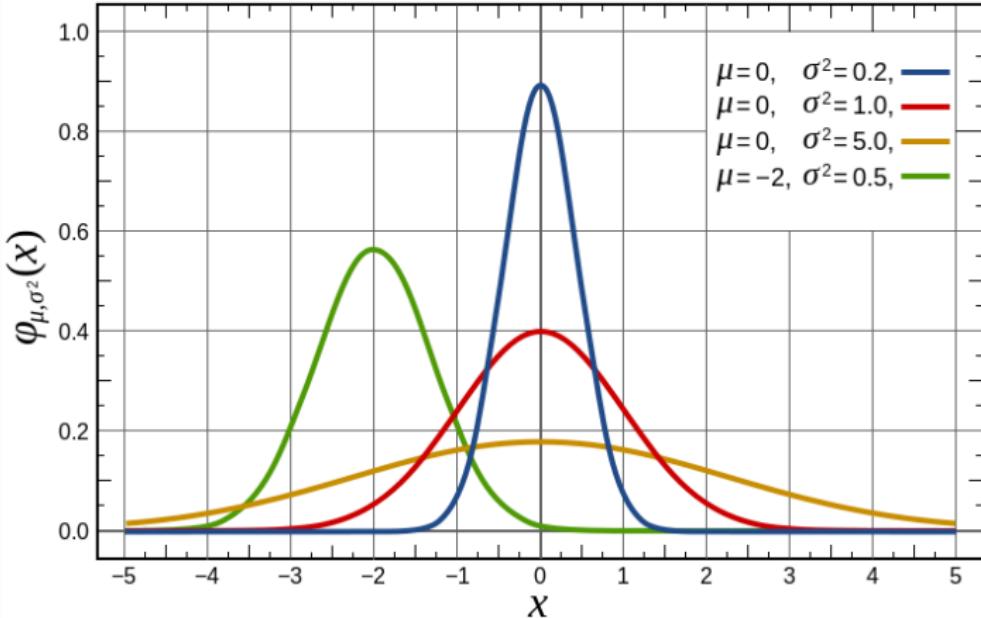
The most important continuous distribution on  $\mathbb{R}$  is the normal distribution, abbreviated  $N(\mu, \sigma^2)$ .

- ▶ It has two parameters: its expectation  $\mu$  and its variance  $\sigma^2$ .
  - ▶  $\mu$  controls the location of the distribution
  - ▶  $\sigma$  controls the “width” of the distribution
- ▶ The density function of  $N(\mu, \sigma^2)$  is given as

$$f_{\mu, \sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

- ▶ The special case of mean 0 and variance 1 is called the “standard normal distribution”. Sometimes the normal distribution is also called a Gaussian distribution.

# Normal distribution (univariate) (2)



# Multivariate normal distribution

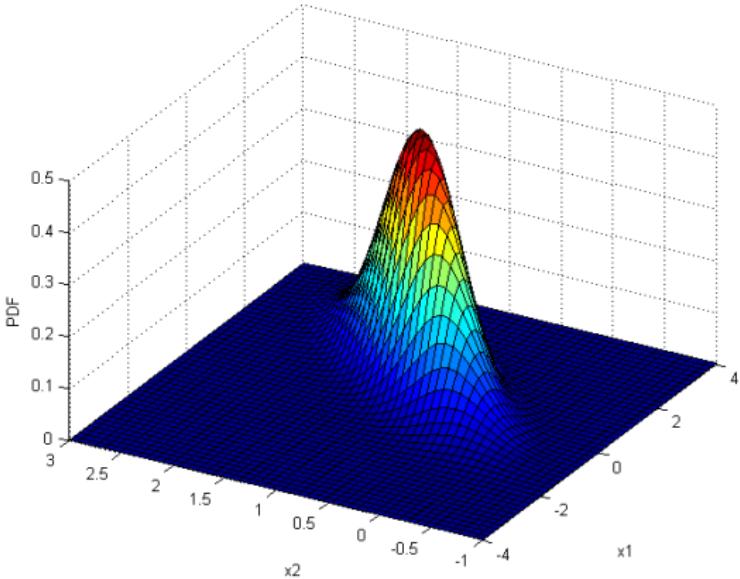
The multivariate normal is defined for the  $d$ -dimensional space  $\mathbb{R}^d$ , it is abbreviated by  $N(\mu, \Sigma)$ .

- ▶ It has two parameters: the expectation vector  $\mu \in \mathbb{R}^d$ , and the covariance matrix  $\Sigma \in \mathbb{R}^{d \times d}$ . The covariance matrix is always positive definite.
- ▶ The density function is defined as follows:

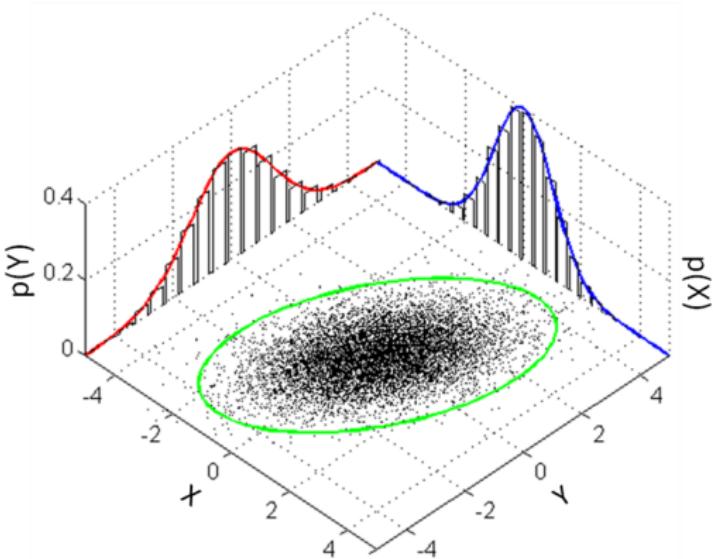
$$f_{\mu, \Sigma}(x) = \frac{1}{\sqrt{2\pi \det(\Sigma)}} \exp\left(-\frac{1}{2}(x - \mu)' \Sigma^{-1} (x - \mu)\right)$$

- ▶ The eigenvectors and eigenvalues of the covariance matrix control the shape of the Gaussian.
- ▶ Each of the marginal distributions is a univariate normal distribution.

# Multivariate normal distribution (2)



# Multivariate normal distribution (3)



# Mixture of Gaussians

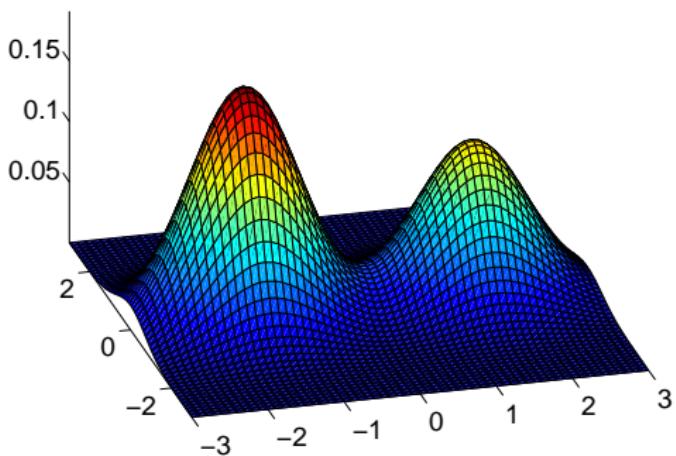
When generating toy data for machine learning applications, one often uses a mixture of Gaussian distributions:

Given mean vectors  $\mu_1, \dots, \mu_k \in \mathbb{R}^d$ , positive definite covariance matrices  $\Sigma_1, \dots, \Sigma_k \in \mathbb{R}^{d \times d}$ , and mixing coefficients  $\alpha_1, \dots, \alpha_k > 0$  with  $\sum \alpha_i = 1$ , the density function of the mixture of Gaussians as follows:

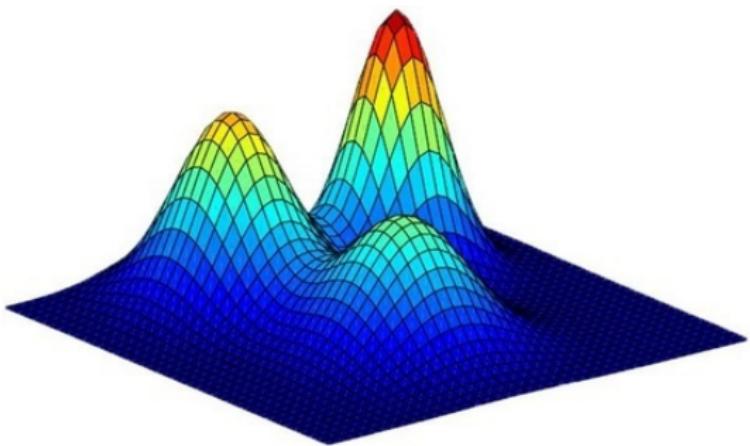
$$f(x) = \sum_{i=1}^k \alpha_i f_{\mu_i, \Sigma_i}$$

# Mixture of Gaussians (2)

density



# Mixture of Gaussians (3)



# Expectation

In the continuous domain, sums are going to be replaced by integrals. For example, the expectation of a random variable  $X$  with density function  $p(x)$  is defined as

$$E(X) = \int_{\mathbb{R}} x \cdot p(x) dx$$

## Recap: Linear algebra

### Literature:

- ▶ In general, any introductory book on linear algebra
- ▶ On the homepage you can also find the link to a short linear algebra recap writeup (by Zico Kolter and Chuong Do).

# Vector space

A **vector space**  $V$  is a set of “vectors” that supports the following operations:

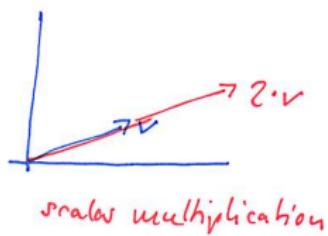
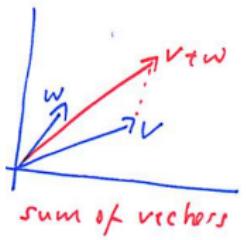
- ▶ We can add and subtract vectors: For  $v, w \in V$  we can build  $v + w, v - w$
- ▶ We can multiply vectors with scalars: For  $v \in V, a \in \mathbb{R}$  we can build  $av$ .
- ▶ These operations satisfy all kinds of formal requirements (associativity, commutativity, identity element, inverse element, and so on).

# Vector space (2)

Most prominent example:  $V = \mathbb{R}^d$ .

$$\begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{pmatrix} + \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{pmatrix} = \begin{pmatrix} v_1 + w_1 \\ \vdots \\ v_d + w_d \end{pmatrix}$$

$$a \cdot \begin{pmatrix} v_1 \\ \vdots \\ v_d \end{pmatrix} = \begin{pmatrix} a \cdot v_1 \\ \vdots \\ a \cdot v_d \end{pmatrix}$$



# Basis

A **basis** of a vector space is a set of vectors  $b_1, \dots, b_d \in V$  that satisfies two properties:

- ▶ Any vector in  $V$  can be written as a linear combination of basis vectors:

For any  $v \in V$  there exist  $a_1, \dots, a_d \in \mathbb{R}$  such that

$$v = \sum_{i=1}^d a_i b_i$$

- ▶ The vectors in the basis cannot be expressed in terms of each other, they are linearly independent:

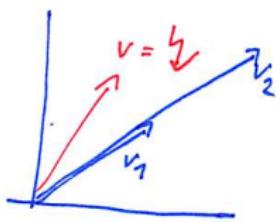
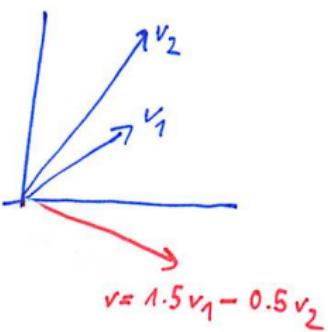
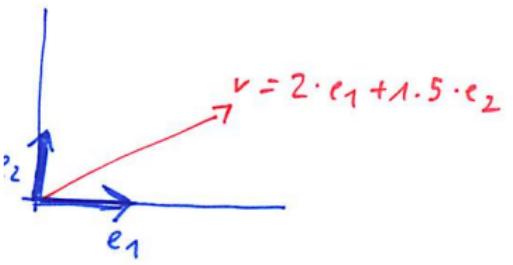
$$\sum_{i=1}^d a_i b_i = 0 \implies a_i = 0 \text{ for all } i = 1, \dots, d.$$

The number of vectors in a basis is called the **dimension** of the vector space.

# Basis (2)

Example:

- $e_1 := (1, 0)$  and  $e_2 := (0, 1)$  form a basis of  $\mathbb{R}^2$
- $v_1 := (1, 1)$  and  $v_2 := (1, 2)$  form a basis of  $\mathbb{R}^2$
- $v_1 := (1, 1)$  and  $v_2 := (2, 2)$  do not form a basis of  $\mathbb{R}^2$ .



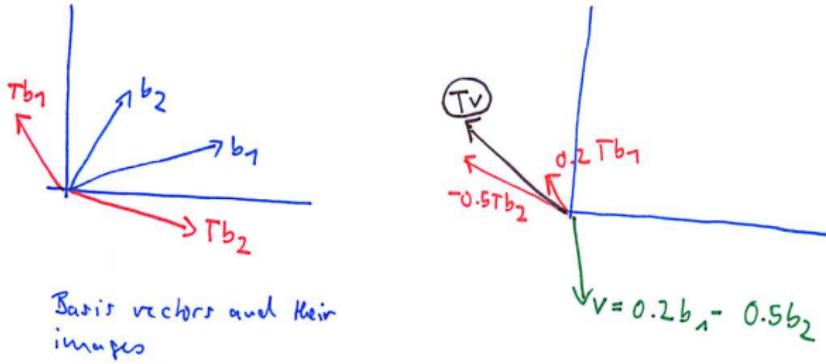
# Linear mappings

A **linear mapping**  $T : V \rightarrow V$  satisfies

$$T(av_1 + bv_2) = aT(v_1) + bT(v_2) \text{ for all } a, b \in \mathbb{R}, v_1, v_2 \in V.$$

Typical linear mappings are: stretching, rotation, projections, etc., and combinations thereof.

Note: to figure out what a linear mapping does, it is enough to know what it does on the basis vectors: for  $v = \sum_i a_i b_i$  we know by linearity that  $T(v) = T(\sum_i a_i b_i) = \sum_i a_i T(b_i)$



Basis vectors and their images

# Matrices

$m \times n$ -matrix A:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & & a_{mn} \end{pmatrix}$$

## Matrices (2)

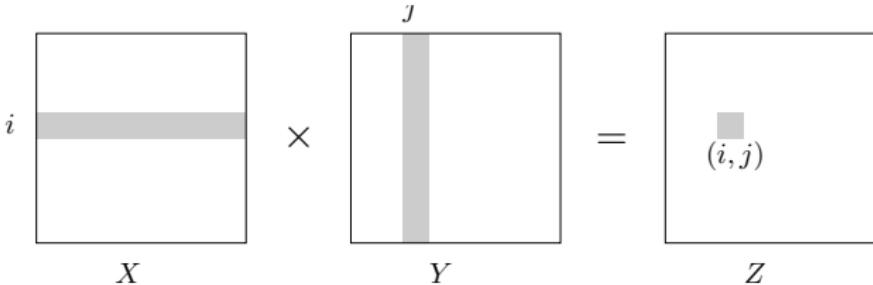
Transpose of a matrix , written as  $A^t$  or  $A'$  is the matrix where we exchange rows with columns (that is, instead of  $a_{ij}$  we have  $a_{ji}$ ).

# Matrices (3)

We can multiply two matrices if their “dimensions” fit:

$X = m \times n$ -matrix,  $Y$   $n \times k$  matrix. Then  $Z : X \cdot Y$  is a  $m \times k$ -matrix with entries

$$z_{ij} = \sum_{s=1}^n x_{is} y_{sk}$$



## Matrices (4)

Special case where  $Y$  is a vector of length  $n \times 1$  is called **matrix-vector-multiplication**:

$$z = Xy \text{ with } z_i = \sum_j x_{ij}y_j$$

# Linear mappings correspond to matrices

Linear mappings correspond to **matrices**:

Intuition: the columns of the matrix contain the images of the basis vectors:

Basis  $e_1, \dots, e_d$ , linear mapping  $T$

→ corresponding matrix:  $\begin{pmatrix} | & | & | \\ T_{e_1} & T_{e_2} & \dots & T_{e_d} \\ | & | & | \end{pmatrix} =: A$

- ▶ Matrix-vector multiplication is then the same as applying the mapping to the vector.
- ▶ Multiplication of two matrices is the same as applying the mappings one after the other.

# The rank of a matrix

Many equivalent definition: The **rank** of a matrix is ...

- ▶ ... the largest number of independent columns in the matrix
- ▶ ... the largest number of independent rows in the matrix
- ▶ ... the dimension of the image space of the linear mapping that corresponds to the matrix
- ▶ ... in case the matrix is symmetric: the rank is the number of non-zero eigenvalues of the matrix (see below).

A  $n \times n$ -matrix is said to have **full rank** if it has rank  $n$ .

A  $n \times n$ -matrix is said to have **low rank** if its rank is “small” compared to  $n$  (this is not a formal definition, it is often used informally).

# Inverse of a matrix

- ▶ For some matrices  $A$  we can compute the inverse matrix  $A^{-1}$ .  
It is the unique matrix that satisfies

$$A \cdot A^{-1} = A^{-1} \cdot A = Id$$

where  $Id$  is the identity matrix (1 on the diagonal, 0 everywhere else).

- ▶ A matrix is called invertible if it has an inverse matrix.
- ▶ A square matrix is invertible if and only if it has full rank.

# Norms and scalar products

Some vector spaces have additional structure: norms or even scalar products. In particular, this is true for  $\mathbb{R}^d$ .

Given two vectors  $v = (v_1, \dots, v_n)^t$  and  $w = (w_1, \dots, w_n)^t \in \mathbb{R}^n$ , their scalar product is defined as  $\langle v, w \rangle = \sum_{i=1}^n v_i w_i$ .

The norm  $\|v\|$  of a vector  $v \in \mathbb{R}^d$  is defined as  $\|v\|^2 = \langle v, v \rangle$ .

Intuition:

- ▶ The scalar product is related to the angle between the two vectors:
  - ▶  $\langle v, w \rangle = 0 \iff v \perp w$  (vectors are orthogonal)
  - ▶ If  $v$  and  $w$  have norm 1, then  $\langle v, w \rangle$  is the cosine of the angle between the two vectors.
- ▶ The norm is the length of a vector.

## Norms and scalar products (2)

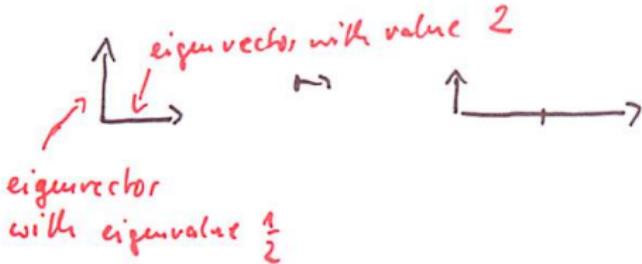
A matrix  $A$  is called **orthogonal** if all its columns are orthogonal to each other. It is called **orthonormal** if additionally, all its columns have norm 1.

For orthogonal matrices, we always have  $A^t = A^{-1}$ .

# Eigenvalues and eigenvectors

A vector  $v \in \mathbb{R}^n, v \neq 0$  is called an **eigenvector** of  $A \in \mathbb{R}^{n \times n}$  with eigenvalue  $\lambda$  if  $Av = \lambda v$ .

Intuition: in the direction of  $v$ , the linear mapping corresponding to  $A$  is stretching by factor  $\lambda$ .



Taken together, all eigenvectors with eigenvalue  $\lambda$  form a subspace called the **eigenspace** associated to eigenvalue  $\lambda$ . The dimension of this subspace is called the **geometric multiplicity** of  $\lambda$ .

# Eigenvalues and eigenvectors (2)

Just for completeness:

Eigenvectors can also be defined as the roots of the **characteristic polynomial**  $f(\lambda) := \det(A - \lambda I) \stackrel{!}{=} 0$ . The degree of this polynomial is  $d$  (the dimension of the space). The multiplicity of this root is called the **algebraic multiplicity of  $\lambda$** .

The algebraic multiplicity is always larger or equal to the geometric one. In case of strict inequality, the matrix cannot be diagonalized.

Simple example where the two multiplicities do not agree:  
the nilpotent matrix  $[0, 1; 0, 0]$  has eigenvalue 0 with geometric multiplicity 1, but algebraic multiplicity 2. It cannot even be diagonalized over  $\mathbb{C}$ .

# Eigenvalue decomposition of a symmetric matrix

Diagonalization:

- ▶ A matrix  $A$  is called **diagonalizable** if there exists a basis of eigenvectors.
- ▶ In this case, we can write the matrix in the form

$$A = VDV^t$$

where  $V$  is an orthonormal matrix that contains the eigenvectors as columns, and  $D$  is a diagonal matrix containing the eigenvalues.

- ▶ One can also write the matrix in the form

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

where  $\lambda_i$  are the eigenvalues and  $v_i$  the eigenvectors.

# Eigenvalue decomposition of a symmetric matrix (2)

- ▶ Intuitively, a matrix is diagonalizable if it performs “Strecken und Spiegeln”, but no rotation.

Symmetric matrices are always diagonalizable and have real-valued eigenvalues. Their eigenvectors (of different eigenvalues) are always perpendicular to each other

## (\*) Singular Value Decomposition (SVD)

If a matrix is not square, we cannot compute eigenvalues. But there exists a closely related concept, the singular values:

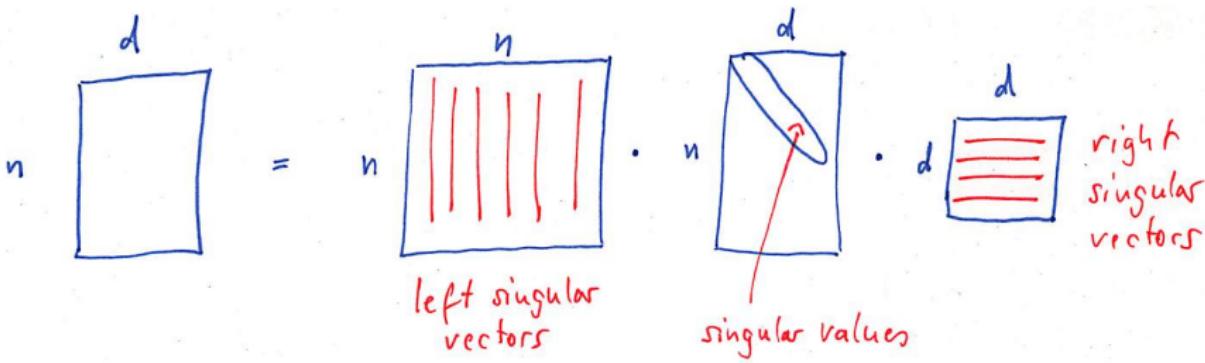
Any matrix  $\Phi \in \mathbb{R}^{n \times d}$  can be decomposed as follows:

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

where

- ▶  $U \in \mathbb{R}^{n \times n}$  is orthogonal. Its columns are called **left singular vectors**.
- ▶  $\Sigma \in \mathbb{R}^{n \times d}$  is a diagonal matrix containing the **singular values**  $\sigma_1, \dots, \sigma_d$  on the diagonal
- ▶  $V \in \mathbb{R}^{d \times d}$  is an orthogonal matrix. Its columns are called **right singular vectors**.

# (\*) Singular Value Decomposition (SVD) (2)

 $\phi$  $=$  $u$  $\Sigma$  $v^t$

## (\*) Singular Value Decomposition (SVD) (3)

There is a close relation between the singular values of  $\Phi$  and the eigenvalues of the (symmetric!) matrices  $\Phi\Phi^t$  and  $\Phi^t\Phi$ :

- ▶ The left singular vectors of  $\Phi$  are the eigenvectors of  $\Phi\Phi^t$ .  
CAN YOU SEE WHY?  
$$\Phi\Phi^t = (U\Sigma V')(U\Sigma V')' = U\Sigma VV'\Sigma U' = U\Sigma^2 U'.$$
- ▶ The right singular vectors of  $\Phi$  are the eigenvectors of  $\Phi^t\Phi$ .
- ▶ The non-zero singular values of  $\Phi$  are the square roots of the non-zero eigenvalues of both  $\Phi^t\Phi$  and  $\Phi\Phi^t$ .

## (\*) Singular Value Decomposition (SVD) (4)

Note in particular:

- ▶ An SVD exists for any matrix!
- ▶ The singular values are unique.
- ▶ The singular vectors are “as unique” as in an eigenvector decomposition (that is, up to scalar multiplication, and in case of higher multiplicity the singular vectors span a whole space).

# Positive Definite Matrices

A symmetric matrix  $A$  is called **positive semi-definite** if all its eigenvalues are  $\geq 0$ . In case of strict inequality it is called **positive definite**.

Equivalent formulations:

- ▶ Positive definite  $\iff v^t A v > 0$  for all  $v \in \mathbb{R}^n \setminus \{0\}$ .
- ▶ Positive semi-definite  $\iff v^t A v \geq 0$  for all  $v \in \mathbb{R}^n \setminus \{0\}$ .
- ▶ Positive semi-definite  $\iff$  we can decompose the matrix in the form  $A = XX'$ .

## (\*) Generalized inverse

Consider a symmetric matrix  $A \in \mathbb{R}^{d \times d}$ .

- ▶ Let  $\lambda_1, \dots, \lambda_d$  the eigenvalues and  $v_1, \dots, v_d$  a corresponding set of eigenvectors of  $A$ . We can write  $A$  in the spectral decomposition as

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

- ▶ In case the matrix has rank  $d$ , all its eigenvalues are non-zero. Then we can write the inverse of  $A$  as

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

## (\*) Generalized inverse (2)

- ▶ In case the matrix is not of full rank, it is not invertible.  
However, we can define the **Moore-Penrose generalized inverse** as

$$\textcolor{brown}{A}^+ := \sum_{i:\lambda_i \neq 0} \frac{1}{\lambda_i} v_i v_i^t$$

(intuitively, this is the inverse of the matrix A restricted to the subspace orthogonal to its nullspace).

## (\*) Generalized inverse (3)

### Properties of the generalized inverse:

In general we don't have that  $AA^+ = I$  or  $A^+A = I$ .

But we have the following slightly weaker properties:

- ▶  $AA^+A = A$  and  $A^+AA^+ = A^+$
- ▶  $(A^+)^+ = A$
- ▶  $A^+A$  and  $AA^+$  are both symmetric.
- ▶ If  $A$  is invertible, then  $A^{-1} = A^+$ .
- ▶  $AA^+$  is an orthogonal projection on the  $\text{ran}(A)$  (the image of the matrix  $A$ ), and  $A^+A$  is an orthogonal projection on  $\text{ran}(A^t)$ .

# (\*) Generalized inverse (4)

## Some intuition:

Consider a linear operator  $A$  that is a projection on some lower-dimensional subspace. As example, consider the projection of the three-dim space to the two-dim plane:

$$A(x_1, x_2, x_3)' := (x_1, x_2)' \in \mathbb{R}^2$$

Call the projection  $A$  and consider a “reconstruction” operator  $A^{rec}$ .

- ▶ Note that from the result of the projection, it is impossible to reconstruct the original point exactly (this is why the matrix  $A$  is not invertible).
- ▶ However, I can reconstruct another point that would give the same projection result: for example, I can simply define

$$A^{rec}(x_1, x_2) := (x_1, x_2, 17) \in \mathbb{R}^3$$

## (\*) Generalized inverse (5)

- ▶ Note that if I apply the projection again after reconstruction, I get the same result as after the first projection: I have

$$AA^{rec}A = A$$

The Moore-Penrose pseudoinverse is one particular such reconstruction operator.

## (\*) Rayleigh principle

### Proposition 51 (Rayleigh principle)

Let  $A \in \mathbb{R}^{n \times n}$  be a symmetric matrix with eigenvalues  $\lambda_1 \geq \dots \geq \lambda_n$  and eigenvectors  $v_1, \dots, v_n$ . Then

$$\lambda_1 = \max_{v \in \mathbb{R}^n} \frac{v^t A v}{\|v\|^2} = \max_{v \in \mathbb{R}^n : \|v\|=1} v^t A v.$$

The eigenvector  $v_1$  is the vector for which this maximum is attained. Moreover,

$$\lambda_{k+1} = \max_{v \perp \{v_1, \dots, v_k\} : \|v\|=1} v^t A v.$$

This theorem holds analogously for minimization problems. In this case, the solution is given by the smallest eigenvalue / vector.

# (\*) Rayleigh principle (2)

## Proof intuition.

- ▶ Let  $\lambda$  be any eigenvalue with eigenvector  $v$ . Then  $v^t A v = v^t (\lambda v) = \lambda$  (because  $v^t v = 1$ ).
- ▶ So among all eigenvectors  $v_1, \dots, v_n$ , the eigenvector  $v_1$  leads to the largest value  $\lambda_1$ .
- ▶ Now consider an arbitrary unit vector  $w \in \mathbb{R}^n$ . Because  $A$  is symmetric, there exists a basis of eigenvectors  $v_1, \dots, v_n$ . In particular, there exist coefficients  $c_i$  such that  $w = \sum_i c_i v_i$  and  $\|c\| = 1$ .
- ▶ Then  $w^t A w = \dots = \sum_{i,j=1}^n c_i c_j v_i^t A v_j$
- ▶ But for  $i \neq j$  we get  $v_i^t A v_j = v_i^t \lambda v_j = 0$  (because different eigenvectors are perpendicular to each other).
- ▶ so  $w^t A w = \sum_i c_i^2 v_i^t A v_i = \sum_i c_i^2 \lambda_i$ .

## (\*) Rayleigh principle (3)

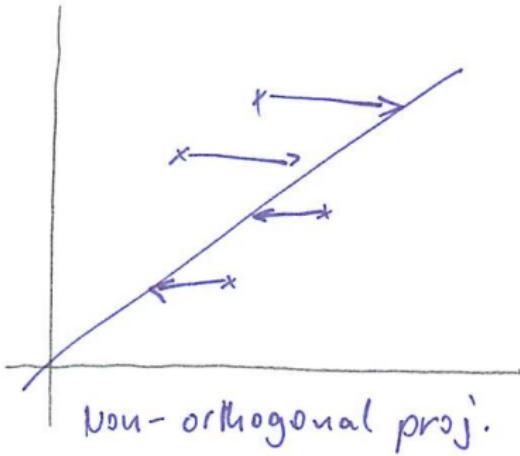
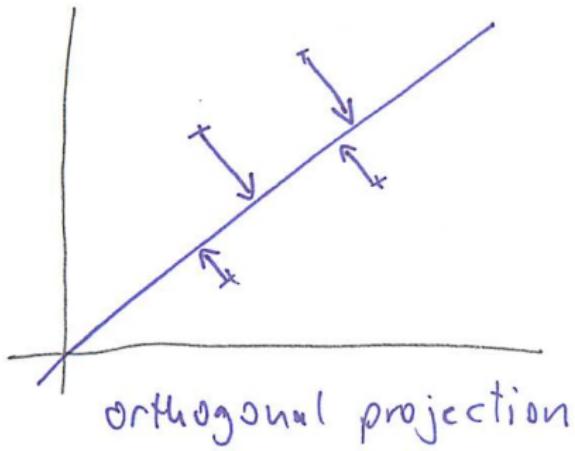
- ▶ Among all  $c$  with  $\|c\| = 1$ , the maximum of this expression is attained for  $c_1 = 1, c_2 = \dots = c_n = 0$ .



# Projections

A linear mapping  $\pi : E \rightarrow F$  between vector spaces is a **projection** if and only if  $\pi^2 = \pi$ .

It is an **orthogonal projection** if and only if it is a projection and  $\text{nullspace}(\pi) \perp \text{image}(\pi)$ .

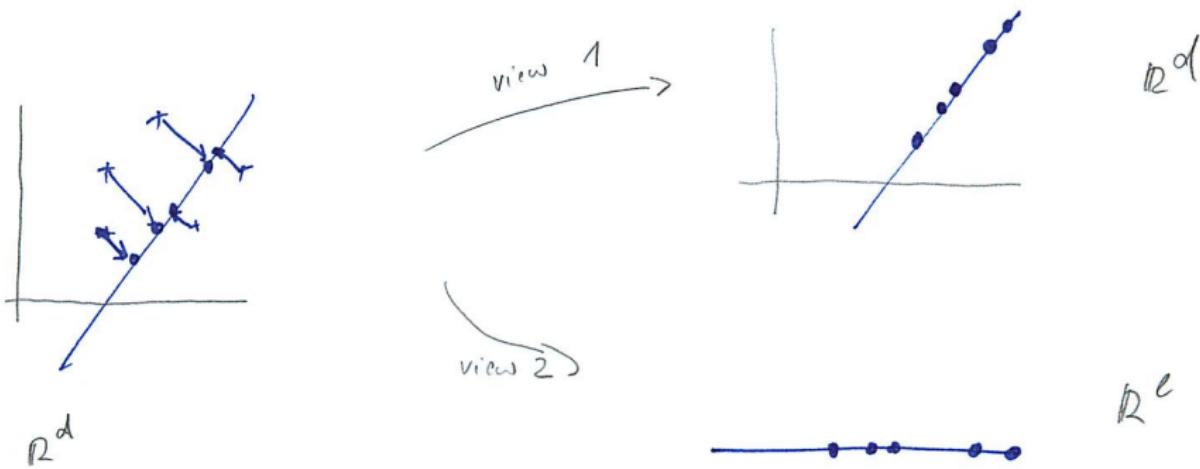


# Projections (2)

We always have two points of view of a projection:

View 1: Represent the projected points still as elements of the original space, that is  $P : \mathbb{R}^d \rightarrow \mathbb{R}^d$ .

View 2: Represent the projected points just as elements of the low-dim space, that is  $\pi : \mathbb{R}^d \rightarrow \mathbb{R}^\ell$



# Projections (3)

Projection on a one-dimensional space:

The orthogonal projection on a one-dimensional space spanned by vector  $a$  can be expressed as

$$\pi : \mathbb{R}^d \rightarrow \mathbb{R}, \pi(x) = a'x$$

(this is in View 2).

# Projections (4)

**Projection on a  $\ell$ -dimensional subspace:** Want to project on an  $\ell$ -dim subspace  $S$  with ONB  $v_1, \dots, v_\ell$ .

**View 2:** Define the matrix  $V$  with the vectors  $v_1, \dots, v_\ell$  as columns. Then compute the low-dim representation as

$$\pi : \mathbb{R}^d \rightarrow \mathbb{R}^\ell, x \mapsto V'x$$

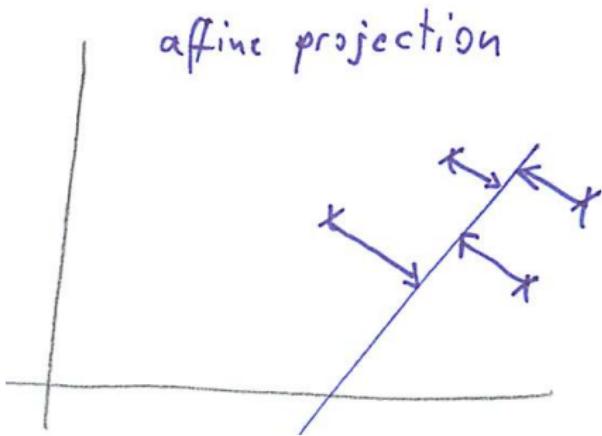
**View 1:** Define  $P := VV'$  (with  $V$  as above) and set

$$\pi : \mathbb{R}^d \rightarrow \mathbb{R}^d, x \mapsto Px$$

# Projections (5)

Affine projections:

Linear projections always map 0 to 0. If we want to perform an orthogonal projection on an affine (= shifted) space  $\tilde{S} = S + \mu$ , we need to express the mapping as  $Tx = P(x - \mu) + \mu$ .



## (\*) Matrix norms

There are many different ways in which one can define a norm of a matrix:

- **Operator norm, spectral norm:** Consider a matrix as linear operator on a normed vector space  $V$  with norm  $\|\cdot\|$ . Then define

$$\|A\| = \sup_{\{x \in V; \|x\|=1\}} \|Ax\|$$

If  $A$  is normal (that is,  $AA^* = A^*A$ ), then the operator coincides with

$$\max\{|\lambda|; \lambda \text{ eigenvalue of } A\}.$$

This is sometimes called the **spectral norm**.

## (\*) Matrix norms (2)

- ▶ Frobenius norm, Hilbert-Schmidt norm, nuclear norm:

$$\|A\|_F := \sqrt{\sum_{ij} a_{ij}^2} = \sqrt{\text{trace}(A * A)} = \sqrt{\sum_i \sigma_i^2}$$

where  $\sigma_i$  are the singular values of the matrix.

# Excursion to convex optimization: primal, dual, Lagrangian

## Literature:

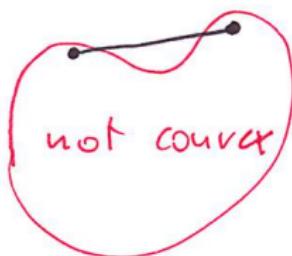
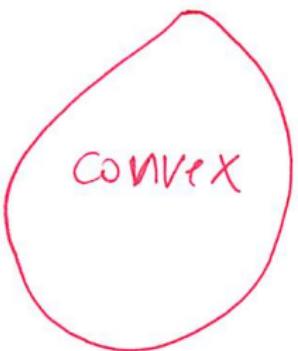
- ▶ Appendix E in the book by Bishop
- ▶ Section 6.3 in the book by Schölkopf / Smola
- ▶ Your favorite book on convex optimization, for example:
  - ▶ Boyd, S. and L. Vandenberghe. Convex Optimization. Cambridge University Press, 2004. Comprehensive, yet easy to read.

# Convex optimization problems: intuition

# Convex optimization problems

Convex sets:

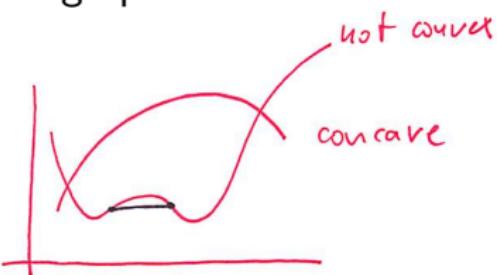
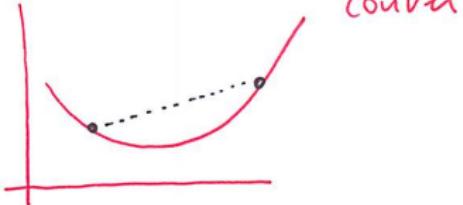
- ▶ A subset  $S$  of a vector space is called convex if for all  $x, y \in S$  and for all  $t \in [0, 1]$  it holds that  $tx + (1 - t)y \in S$ .
- ▶ In words: for any two points  $x, y \in S$ , the straight line connecting these two points is contained in the set  $S$ .



# Convex optimization problems (2)

Convex functions:

- ▶ A function  $f : S \rightarrow \mathbb{R}$  that is defined on a convex domain  $S$  is called convex if for all  $x, y \in S$  and  $t \in [0, 1]$  we have
$$f(tx + (1 - t)y) \leq tf(x) + (1 - t)f(y).$$
- ▶ Intuitively, this means that if we look at the graph of the function and we connect two points of this graph by a straight line, then this line is always above the graph.



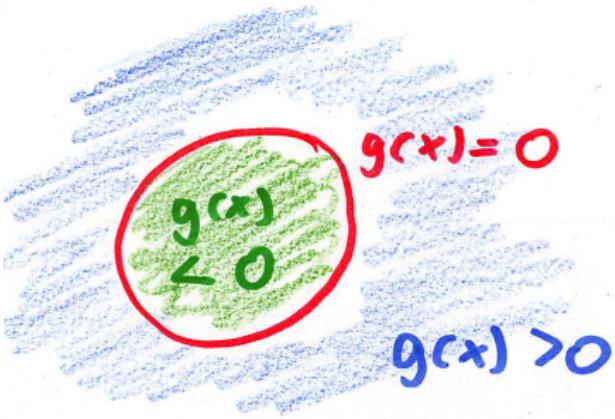
# Convex optimization problems (3)

Examples:

- ▶ functions of one variable that are twice differentiable are convex iff their second derivative is non-negative.
- ▶ Functions of several variables that are twice differentiable are convex if their Hessian matrix is positive (semi)-definite.

## Convex optimization problems (4)

Observe: For convex functions  $g$ , the sublevel sets of the form  $\{x | g(x) \leq 0\}$  are convex.



(Funnily, this is not true the other way round: you can have all sublevel sets convex, but yet the function is not convex. )

# Convex optimization problems (5)

Convex optimization problem:

- ▶ An optimization problem of the form

$$\text{minimize } f(x)$$

$$\text{subject to } g_i(x) \leq 0 \quad (i = 1, \dots, k)$$

is called convex if the functions  $f$ ,  $g_i$  are convex.

- ▶ Sometimes one also considers equality constraints of the form  $h_j = 0$ . They can always be replaced by two inequality constraints  $h_j \leq 0$  and  $-h_j \leq 0$ . However, the only situation in which  $h_j$  and  $-h_j$  are both convex occurs if  $h$  is a linear function.
- ▶ Convex optimization problems have the desirable property that any local minimum is already a global minimum.

# Convex optimization problems (6)

Important terms:

- ▶ The function  $f$  over which we optimize is called the **objective function**
- ▶ The functions  $g_i$  are called the **constraints**.
- ▶ The set of points where all constraints are satisfied is called the **feasible set**.

# Lagrangian: intuitive point of view

# Convex optimization problem

We now want to derive a “recipe” by which many convex optimization problems can be analyzed / rewritten / solved. We don’t consider formal proofs, but just derive the concepts in an intuitive way.

In particular, for the ease of presentation assume that all functions are continuously differentiable (all statements hold in more general settings as well, but one would need convex analysis for this).

# Recap: gradient of a function

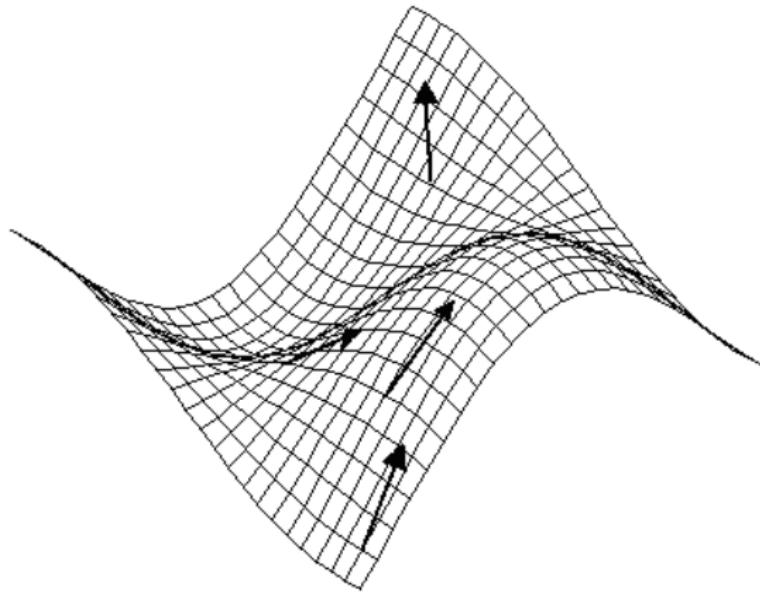
Consider a function  $f : \mathbb{R}^d \rightarrow \mathbb{R}$ .

- The **gradient** of  $f$  is the vector of partial derivatives:

$$\nabla f(x) = (\partial/\partial x_1, \dots, \partial/\partial x_d)'(x)$$

- For each  $x$ , the gradient  $\nabla f(x)$  points in the direction where the function increases most:

## Recap: gradient of a function (2)



Gradient Vectors Shown at Several Points on the Surface of  $\cos(x) \sin(y)$

# Lagrange multiplier for equality constraints

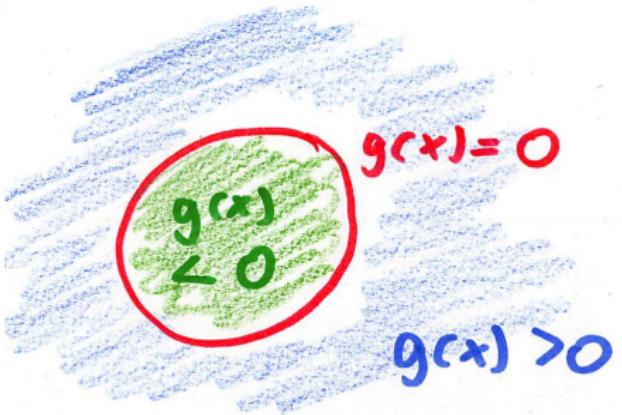
Consider the following convex optimization problem:

$$\begin{aligned} & \text{minimize } f(x) \\ & \text{subject to } g(x) = 0 \end{aligned}$$

where  $f$  and  $g$  are convex.

# Lagrange multiplier for equality constraints (2)

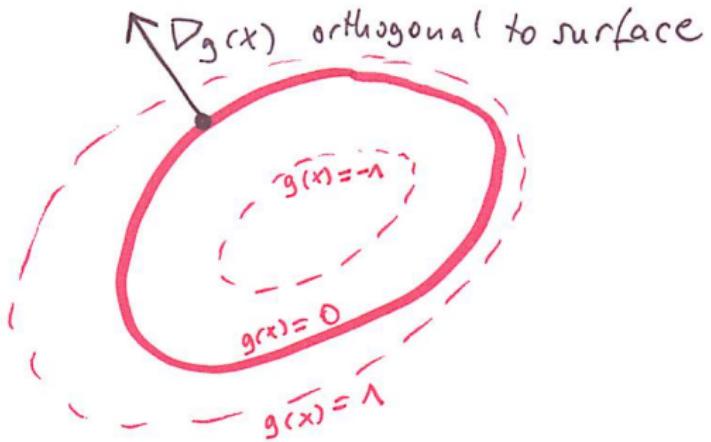
Recall: if  $g$  is convex, then its sublevel-sets are convex:



Sublevel set:  $\{x | g(x) \leq c\}$  (the green set in the figure)

# Lagrange multiplier for equality constraints (3)

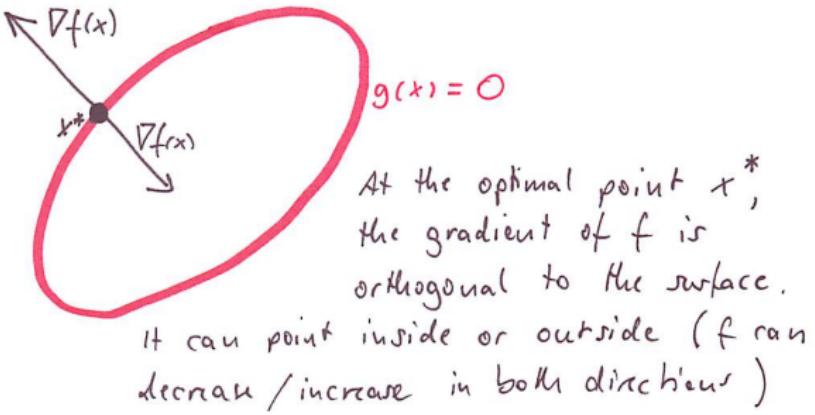
**Gradient (equality constraint):** For any point  $x$  on the “surface”  $\{g(x) = 0\}$  the gradient  $\nabla g(x)$  is orthogonal to the surface itself.



Intuition: to increase / decrease  $g(x)$ , you need to move away from the surface, not walk along the surface.

# Lagrange multiplier for equality constraints (4)

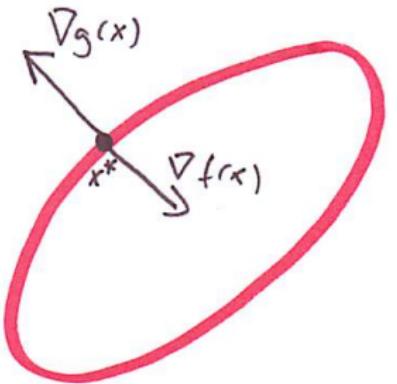
**Gradient (objective function):** Consider the point  $x^*$  on the surface  $\{g(x) = 0\}$  for which  $f(x)$  is minimized. This point must have the property that  $\nabla f(x)$  is orthogonal to the surface.



Intuition: otherwise we could move a little along the surface to decrease  $f(x)$ .

## Lagrange multiplier for equality constraints (5)

Consequence: at the optimal point,  $\nabla g(x)$  and  $\nabla f(x)$  are parallel, that is there exists some  $\nu \in \mathbb{R}$  such that  $\nabla f(x) + \nu \nabla g(x) = 0$ .



At the optimal point  $x^*$ ,  
 $\nabla g$  and  $\nabla f$  are parallel to  
each other (and can point  
either in the same or the  
opposite direction).

# Lagrange multiplier for equality constraints (6)

We now define the **Lagrangian** function

$$L(x, \nu) = f(x) + \nu g(x)$$

where  $\nu \in \mathbb{R}$  is a new variable called **Lagrange multiplier**. Now observe:

- ▶ The condition  $\nabla f(x) + \nu \nabla g(x) = 0$  is equivalent to  
 $\nabla_x L(x, \nu) = 0$
- ▶ The condition  $g(x) = 0$  is equivalent to  $\nabla_\nu L(x, \nu) = 0$ .

To find an optimal point  $x^*$  we need to find a **saddle point** of  $L(x, \nu)$ , that is a point such that both  $\nabla_x L(x, \nu)$  and  $\nabla_\nu L(x, \nu)$  vanish.

## Simple example

Consider the problem to minimize  $f(x)$  subject to  $g(x) = 0$ , where  $f, g : \mathbb{R}^2 \rightarrow \mathbb{R}$  are defined as

$$\begin{aligned}f(x_1, x_2) &= x_1^2 + x_2^2 - 1 \\g(x_1, x_2) &= x_1 + x_2 - 1\end{aligned}$$

Observe: it is hard to solve this problem by naive methods because it is unclear how to take care of the constraints!

### Solution by the Lagrange approach:

Write it in the standard form:

$$\begin{aligned}&\text{minimize } x_1^2 + x_2^2 - 1 \\&\text{subject to } x_1 + x_2 - 1 = 0\end{aligned}$$

## Simple example (2)

The Lagrangian is

$$L(x, \nu) = \underbrace{x_1^2 + x_2^2 - 1}_{f(x_1, x_2)} + \nu \underbrace{(x_1 + x_2 - 1)}_{g(x_1, x_2)}$$

Now compute the derivatives and set them to 0:

$$\nabla_{x_1} L = 2x_1 + \nu \stackrel{!}{=} 0$$

$$\nabla_{x_2} L = 2x_2 + \nu \stackrel{!}{=} 0$$

$$\nabla_\nu L = x_1 + x_2 - 1 \stackrel{!}{=} 0$$

If we solve this linear system of equations we obtain  
 $(x_1^*, x_2^*) = (0.5, 0.5)$ .

# Lagrange multiplier for inequality constraints

Consider the following convex optimization problem:

$$\begin{aligned} & \text{minimize } f(x) \\ & \text{subject to } g(x) \leq 0 \end{aligned}$$

where  $f$  and  $g$  are convex.

We now distinguish two cases: constraint is “active” or “inactive”:

## Lagrange multiplier for inequality constraints (2)

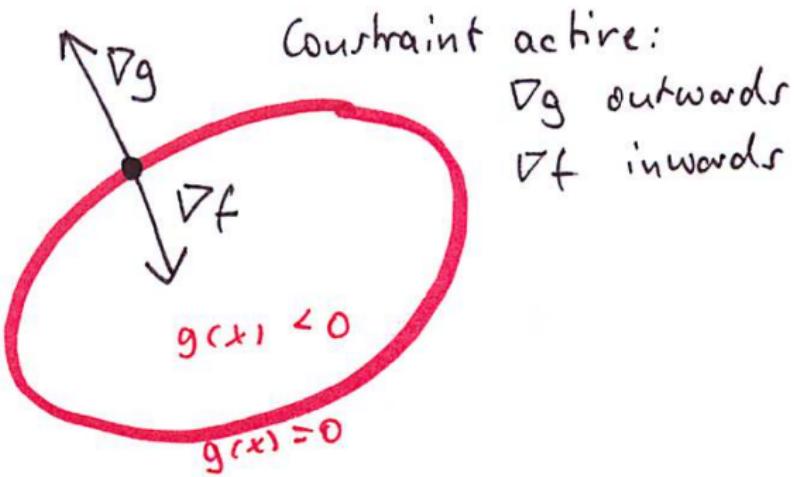
Case 1: Constraint is “active”, that is the optimal point is *on* the surface  $g(x) = 0$ .

Again  $\nabla f$  and  $\nabla g$  are parallel in the optimal point.

But furthermore, the direction of derivatives matters:

- ▶ The derivative of  $g$  points outwards (at any point on the surface  $g = 0$ ). This is always the case if  $g$  is convex.
- ▶ Then the derivative of  $f$  is directed inwards (otherwise we could decrease the objective by walking inside).

# Lagrange multiplier for inequality constraints (3)



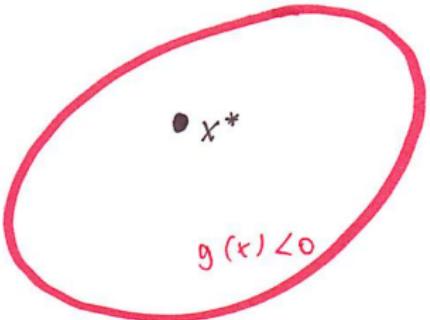
So we have  $\nabla f(x) = -\lambda \nabla g(x)$  for some value  $\lambda > 0$ .

## Lagrange multiplier for inequality constraints (4)

Case 2: Constraint is “inactive”, that is the optimal point is not on the surface  $g(x) = 0$  but somewhere in the interior.

- ▶ Then we have  $\nabla f = 0$  at the solution (otherwise we could decrease the objective value).
- ▶ We do not have any condition on  $\nabla g$  (it is as if we would not have this constraint).

Constraint inactive. No condition on  $\nabla g$



## Lagrange multiplier for inequality constraints (5)

We can summarize both cases using the Lagrangian again. We now define the Lagrangian

$$L(x, \lambda) = f(x) + \lambda g(x)$$

where the Lagrange multiplier has to be positive:  $\lambda \geq 0$ .

- ▶ Case 1: constraint active,  $\lambda > 0$ .
  - ▶ Need to find a saddle point:  $\nabla_x L(x, \lambda) = \nabla_\lambda L(x, \lambda) = 0$ .
- ▶ Case 2: constraint inactive,  $\lambda = 0$ .
  - ▶ Then  $L(x, \lambda) = f(x)$ . Hence  $\nabla_x L(x, \lambda) = \nabla_x f(x) \stackrel{!}{=} 0$ ,  
 $\nabla_\lambda L(x, \lambda) \equiv 0$ .
- ▶ So in both cases we have again a saddle point of the Lagrangian.

# Lagrange multiplier for inequality constraints (6)

Also in both cases we have  $\lambda g(x^*) = 0$ .

- ▶ Constraint active:  $\lambda > 0, g(x^*) = 0$ .
- ▶ Constraint inactive:  $\lambda = 0, g(x^*) \neq 0$ .

This is called the **Karush-Kuhn-Tucker (KKT) condition**.

## Simple example

What are the side lengths of a rectangle that maximize its area, under the assumption that its perimeter is at most 1?

We need to solve the following optimization problem:

$$\text{maximize } x \cdot y \text{ subject to } 2x + 2y \leq 1$$

Bring the problem in standard form:

$$\text{minimize}(-x \cdot y) \text{ subject to } 2x + 2y - 1 \leq 0$$

Form the Lagrangian:

$$L(x, y, \lambda) = -xy + \lambda(2x + 2y - 1)$$

## Simple example (2)

Saddle point conditions / derivatives:

$$\partial L / \partial x = -y + 2\lambda \stackrel{!}{=} 0$$

$$\partial L / \partial y = -x + 2\lambda \stackrel{!}{=} 0$$

$$\partial L / \partial \lambda = 2x + 2y - 1 \stackrel{!}{=} 0$$

Solving this system of three equations gives  $x = y = 0.25$ .

## Simple example (3)

Now need to see: when does this approach work, when does it not work, what can we prove about it?

Lagrangian: formal point of view

# Lagrangian and dual: formal definition

Consider the primal optimization problem

$$\text{minimize } f_0(x)$$

$$\text{subject to } f_i(x) \leq 0 \quad (i = 1, \dots, m)$$

$$h_j(x) = 0 \quad (j = 1, \dots, k)$$

Denote by  $x^*$  a solution of the problem and by  $p^* := f_0(x^*)$  the objective value at the solution.

# Lagrangian and dual: formal definition (2)

Define the corresponding **Lagrangian** as follows:

- ▶ For each equality constraint  $j$  introduce a new variable  $\nu_j \in \mathbb{R}$ , and for each inequality constraint  $i$  introduce a new variable  $\lambda_i \geq 0$ . These variables are called **Lagrange multipliers**.
- ▶ Then define

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^m \lambda_i f_i(x) + \sum_{j=1}^k \nu_j h_j(x)$$

Define the **dual function**  $g : \mathbb{R}^m \times \mathbb{R}^k \rightarrow \mathbb{R}$  by

$$g(\lambda, \nu) = \inf_x L(x, \lambda, \nu)$$

# Dual function as lower bound on primal

## Proposition 52 (Dual function is concave)

No matter whether the primal problem is convex or not, the dual function is always concave in  $(\lambda, \nu)$ .

**Proof.** For fixed  $x$ ,  $L(x, \lambda, \nu)$  is linear in  $\lambda$  and  $\nu$  and thus concave. The dual function as a pointwise infimum over concave functions is concave as well. ☺

Note that concave is good, because we are going to maximize this function later on.

## Dual function as lower bound on primal (2)

Proposition 53 (Dual function as lower bound on primal)

For all  $\lambda_i \geq 0$  and  $\nu_j \in \mathbb{R}$  we have  $g(\lambda, \nu) \leq p^*$ .

### Proof.

- ▶ Let  $x_0$  be a feasible point of the primal problem (that is, a point that satisfies all constraints).
- ▶ For such a point we have

$$\sum_{i=1}^m \underbrace{\lambda_i}_{\geq 0} \underbrace{f_i(x_0)}_{\leq 0} + \sum_{j=1}^k \nu_j \underbrace{h_j(x_0)}_{=0} \leq 0$$

## Dual function as lower bound on primal (3)

- This implies

$$L(x_0, \lambda, \nu) = f_0(x_0) + \sum_{i=1}^m \lambda_i f_i(x_0) + \sum_{j=1}^k \nu_j h_j(x_0) \leq f_0(x_0)$$

Note that this property holds in particular when  $x_0$  is  $x^*$ .

- Moreover, for any  $x_0$  (and in particular for  $x_0 := x^*$ ) we have

$$\inf_x L(x, \lambda, \nu) \leq L(x_0, \lambda, \nu)$$

- Combining the last two properties gives

$$g(\lambda, \nu) = \inf_x L(x, \lambda, \nu) \leq L(x^*, \lambda, \nu) \leq f_0(x^*)$$



# Dual optimization problem

Have seen: the dual function provides a lower bound on the primal value. Finding the highest such lower bound is the task of the dual problem:

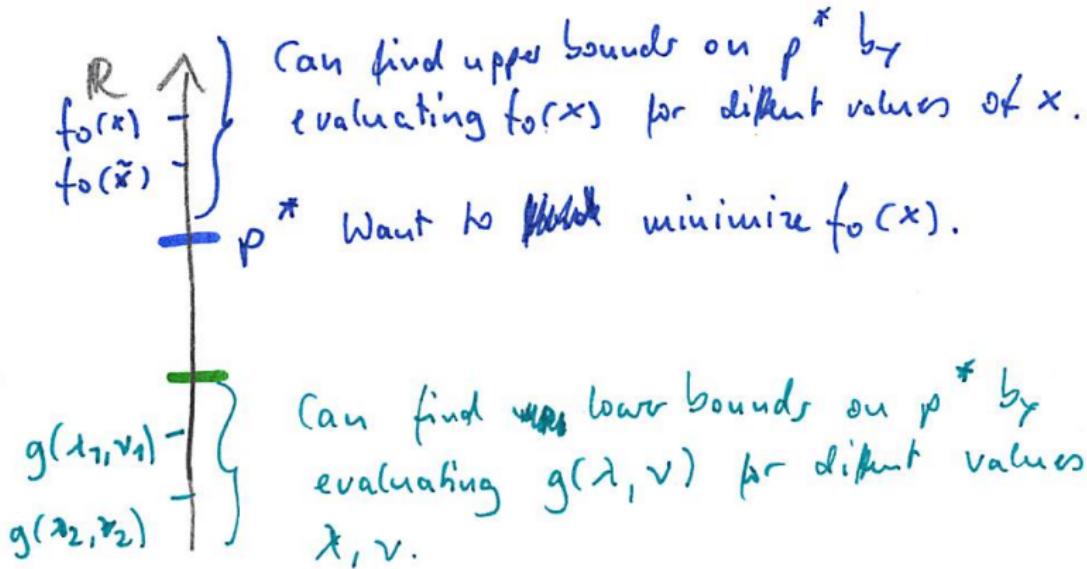
We define the **dual optimization problem** as

$$\max_{\lambda, \nu} g(\lambda, \nu) \text{ subject to } \lambda_i \geq 0, \nu_j \in \mathbb{R}$$

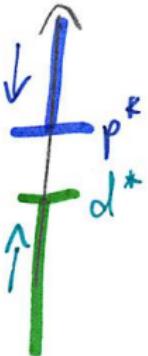
Denote the solution of this problem by  $\lambda^*, \nu^*$  and the corresponding objective value  $d^* := g(\lambda^*, \nu^*)$ .

# Dual optimization problem (2)

Dual vs Primal, some intuition:



## Dual optimization problem (3)



Primal problem: find best upper bound on  $p^*$   
 $(= \text{find } p^*)$

Dual problem: find best lower bound on  $p^*$   
 $(= \text{find } d^*)$

# Weak duality

## Proposition 54 (Weak duality)

The solution  $d^*$  of the dual problem is always a lower bound for the solution of the primal problem, that is  $d^* \leq p^*$ .

**Proof.** Follows directly from Proposition 53 above. ☺

We call the difference  $p^* - d^*$  the **duality gap**.

# Strong duality

- ▶ We say that **strong duality** holds if  $p^* = d^*$ .
- ▶ This is not always the case, just under particular conditions. Such conditions are called **constraint qualifications** in the optimization literature.
- ▶ Convex optimization problems often satisfy strong duality, but not always.

## Strong duality (2)

Examples:

- ▶ Linear problems have strong duality
- ▶ Quadratic problems have strong duality ( $\leadsto$  support vector machines)
- ▶ There exist many convex problems that do not satisfy strong duality. Here is an example:

$$\begin{aligned} & \text{minimize}_{x,y} \exp(-x) \\ & \text{subject to } x/y \leq 0 \\ & \quad y \geq 0 \end{aligned}$$

One can check that this is a convex problem, yet  $p^* = 1$  and  $d^* = 0$ .

# Strong duality: how to convert the solution of the dual to the one of the primal

By strong duality:  $p^* = d^*$ , that is we get the same objective values. But how can we recover the primal variables  $x^*$  that lead to this solution, if we just know the dual variables  $\lambda^*, \nu^*$  of the optimal dual solution?

EXERCISE!

# Strong duality implies saddle point

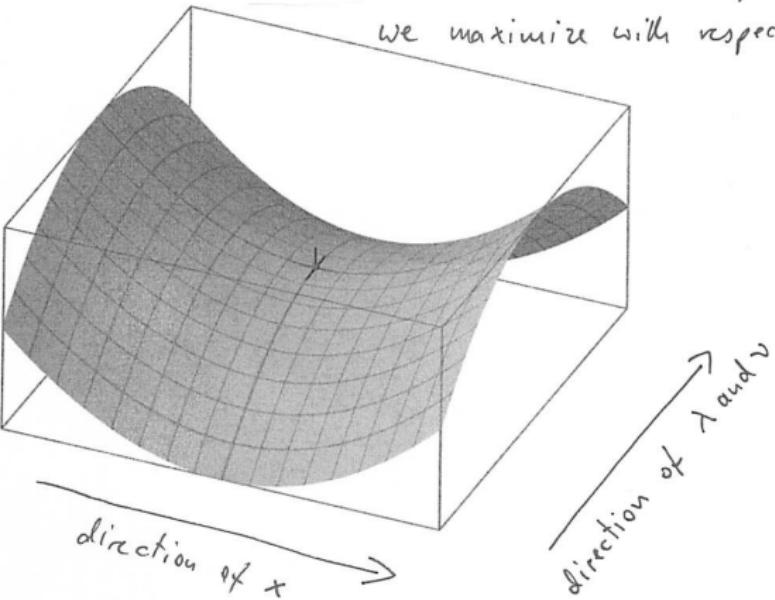
## Proposition 55 (Strong duality implies saddle point)

Assume strong duality holds, let  $x^*$  be the solution of the primal and  $(\lambda^*, \nu^*)$  the solution of the dual optimization problem. Then  $(x^*, \lambda^*, \nu^*)$  is a saddle point of the Lagrangian.

# Strong duality implies saddle point (2)

Lagrangian saddlepoint

we minimize with respect to  $x$ ,  
we maximize with respect to  $\lambda, \gamma$



# Strong duality implies saddle point (3)

## Proof.

- ▶ We first have to show that  $x^*$  is a minimizer of  $L(x, \lambda^*, \nu^*)$ :
  - ▶ By the strong duality assumption we have  $f_0(x^*) = g(\lambda^*, \nu^*)$ .
  - ▶ With this we get

$$f_0(x^*) = g(\lambda^*, \nu^*) = \inf_x L(x, \lambda^*, \nu^*) \leq L(x^*, \lambda^*, \nu^*) \leq f_0(x^*)$$

(last inequality follows from Proposition 54).

- ▶ Because we have the same term on the left and side, we have equality everywhere.
- ▶ So in particular,  $\inf_x L(x, \lambda^*, \nu^*) = L(x^*, \lambda^*, \nu^*)$ .
- ▶ Then we have to show that  $(\lambda^*, \nu^*)$  are maximizers of  $L(x^*, \lambda, \nu)$ .
  - ▶ This follows from the definition of  $(\lambda^*, \nu^*)$  as solutions of  $\max_{\lambda, \nu} \min_x L(x, \lambda, \nu)$ .

# Strong duality implies saddle point (4)

- Taken together we get

$$L(x^*, \lambda, \nu) \leq L(x^*, \lambda^*, \nu^*) \leq L(x, \lambda^*, \nu^*)$$

That is,  $(x^*, \lambda^*, \nu^*)$  is a **saddle point** of the Lagrangian:

- It is a minimum for  $x$  (with fixed  $\lambda^*, \nu^*$ ).
- It is a maximum for  $(\lambda, \nu)$  (with fixed  $x^*$ ).



# Saddle point always implies primal solution

## Proposition 56 (Saddlepoint implies primal solution)

If  $(x^*, \lambda^*, \nu^*)$  is a **saddle point** of the Lagrangian, then  $x^*$  is always a solution of the primal problem.

**Proof.** Not very difficult, but we skip it. ☺

Remarks:

- ▶ This proposition always holds (not only under strong duality).
- ▶ This proposition gives sufficient conditions for optimality.  
Under additional assumptions (constraint qualifications) it is also a necessary condition.

# Why is this whole approach useful?

- ▶ Whenever we have a saddle point of the Lagrangian, we have a solution of our constraint optimization problem. This is great, because otherwise we would not know how to solve it.
- ▶ If strong duality holds, we even know that any solution must be a saddle point. So if we don't find a saddle point, then we know that no solution exists.
- ▶ If your original minimization problem is not convex, at least its dual is a concave maximization problem (or, by changing the sign, a convex minimization problem). If the duality gap is small, then it might make sense to solve the dual instead of the primal (you will not find the optimal solution, but maybe a solution that is close).
- ▶ As we will see for support vector machines, the Lagrangian framework sometimes gives important insights into properties of the solution.