

Lecture Notes on Machine Learning

Ulrike von Luxburg

Summer 2014

Department of Computer Science, University of Hamburg

(Version as of May 26, 2014)

Contents will be updated during the lectures. Up-to-date material to be found on the lecture's webpage. We skipped some sections or parts, these are marked by (*). You don't need to know these parts for the exam!

Table of contents

Introduction to Machine Learning

What is machine learning?	7
Motivating examples and applications	8
Machine learning as inductive inference	41
Different learning scenarios	65

Warmup: k -nearest neighbor classification	75
The kNN algorithm for classification	76

Recap: Maths basics (Linear algebra and probability theory, slides in the appendix)

Supervised learning

Table of contents (2)

Formal setup	102
Statistical and Bayesian Decision theory	103
Case of known distribution	107
Risk minimization framework: loss, risk, ERM, RRM	123
Settings for all supervised learning problems	124
Particular setup for classification	142
Particular setup for regression	157
(*) Generative vs. discriminative approaches SKIPPED	167
The No-Free-Lunch Theorem	178
Linear Methods for regression	190
Linear least squares regression	191
Least squares with linear combination of basis functions	219
Ridge regression: least squares with L_2 -regularization	226

Table of contents (3)

Lasso: least squares with L_1 -regularization	250
Linear Methods for classification	267
Intuition and feature representation	268
Linear discriminant analysis	283
Logistic regression	306
Logistic regression, ERM viewpoint	307
Logistic regression, Bayesian (model-based) viewpoint	313
Regularized logistic regression	326
Linear Support vector machines (Intuition and primal)	330
Excursion to optimization: primal, dual, Lagrangian	354
Lagrangian: intuitive point of view	355
Lagrangian: formal point of view	371
Linear Support vector machines (Dual and Properties)	388
Important properties of SVMs	400

Table of contents (4)

Kernel methods for supervised learning	408
Positive definite kernels	409
Intuition	410
Definition and properties of kernels	419
Reproducing kernel Hilbert space and feature maps	438
Support vector machines with kernels	455
Regression methods with kernels	474
Kernelized least squares	475
Kernel ridge regression	483
How to center and normalize in the feature space	487

Unsupervised learning

PCA and kernel PCA	501
Multi-dimensional scaling and Isomap	551
Multi-dimensional scaling	552
Graph-based machine learning algorithms	568

Table of contents (5)

Isomap	575
--------------	-----

Clustering

K-means and kernel k-means	600
Standard k -means algorithm	601
Kernel k -means	624
Linkage algorithms for hierarchical clustering	628
A glimpse on spectral graph theory	637
Unnormalized Laplacians	639
Normalized Laplacians	651
Cheeger constant and isoperimetric problems	657
Spectral clustering	662
Unnormalized spectral clustering	672
Normalized spectral clustering	695

The data processing chain: from raw data to machine learning

Table of contents (6)

Preparing the data	703
Data aquisition: train versus test distribution	704
Converting raw data to training data	709
Data cleaning: missing values, outliers	711
Defining features, similarities, distance functions	718
Defining a similarity / kernel / distance function	724
Reducing the number of training points?	730
Unsupervised dimensionality reduction	734
Data standardization	737
Setting up the learning problem	739
Choice of a loss / risk function	740
Training	747
Selecting parameters by cross validation	750
Feature selection	760

Table of contents (7)

Evaluation of the results	775
General guidelines for attacking ML problems	804
Key steps in the processing chain	805
High-level guidelines and principles	808

Wrap up

Things we did not talk about	811
Further machine learning resources	813
Future lectures in Hamburg	817

Reinforcement learning: taught by Norman Hendrich

Mathematical Appendix

Recap: Probability theory	821
---------------------------------	-----

Table of contents (8)

Discrete Probability Theory	822
Continuous probability theory	851
Recap: Linear algebra	859

Literature

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

Books with a focus on algorithms:

- ▶ *Chris Bishop: Pattern recognition and Machine Learning. Springer 2006.*
- ▶ *Hastie, Tibshirani, Friedman: Elements of statistical learning, 2nd edition. Springer 2009.*
More from a traditional statistics point of view.
- ▶ *Duda, Hart: Pattern classification. 2000.*
A bit outdated, but still good.
- ▶ *Schölkopf, Smola: Learning with kernels. MIT Press, 2002.*
Mainly SVMs and kernel algorithms.

Literature (2)

- ▶ *Shawe-Taylor, Cristianini: Kernel methods for pattern recognition. Cambridge University Press, 2004.*

For completeness, here are two more text books (we won't cover them because we don't have time):

- ▶ *Kevin Murphy: Machine learning, a probabilistic perspective. MIT Press, 2012.*
The Bayesian / probabilistic viewpoint.
- ▶ *MacKay: Information theory, inference and learning algorithms. Cambridge University Press, 2003*
Information theoretic point of view on machine learning.

Learning theory:

Literature (3)

- ▶ *Devroye, Györfi, Lugosi: A Probabilistic Theory of Pattern Recognition. Springer, 1996.*
Covers the main results of statistical learning theory, we won't use much of it in this lecture.
- ▶ *Mohri, Rostamizadeh, Talwalkar: Foundations of Machine Learning. MIT Press, 2012*

Optimization:

- ▶ Boyd / Vandenberghe for convex optimization

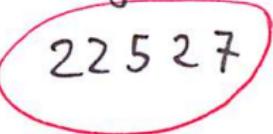
Introduction to Machine Learning

What is machine learning?

Motivating examples and applications

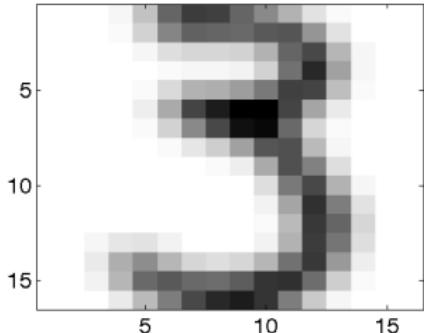
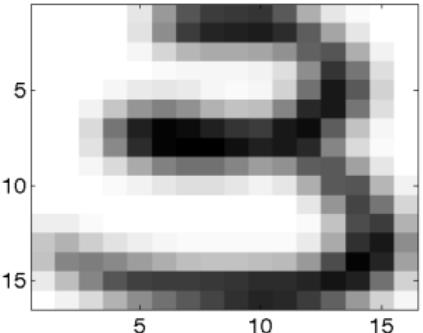
Hand-written digit recognition

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

Fachbereich Informatik
Vogt-Kölln-Str. 30
 22527 Hamburg

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

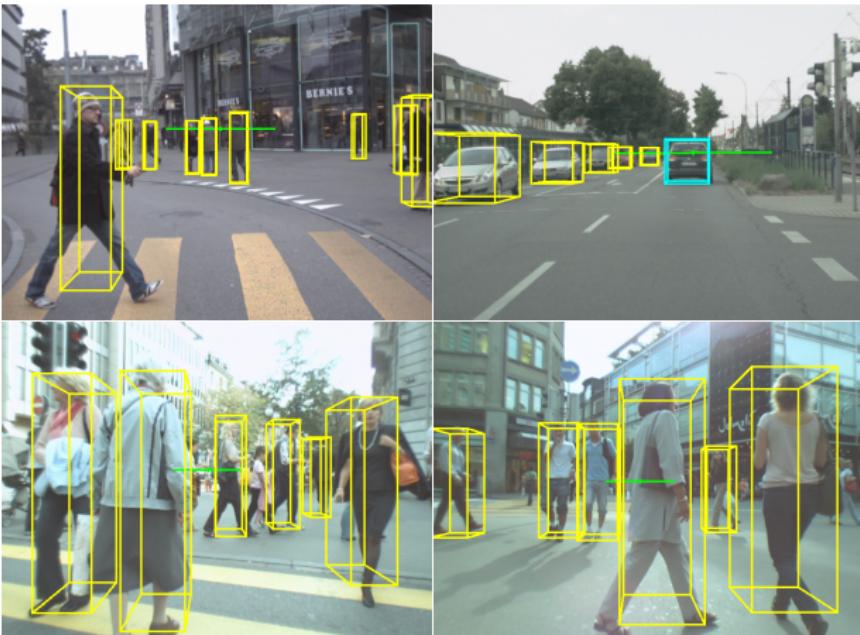
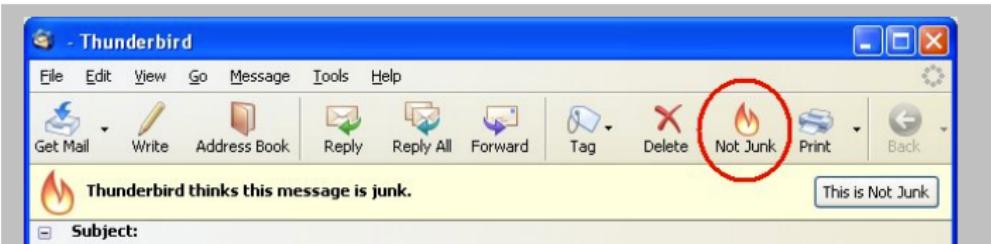


Image: Christian Wojek et al, IEEE PAMI, 2013

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

Recommender Systems

Amazon: “If you like this book, you might also like this other book.”

How such a recommender system built?

- ▶ Collect shopping data of all users
- ▶ Based on the shopping data, learn a “rule” that predicts what item a certain customer might want to buy.
- ▶ This rule is based both on the joint shopping data of all users (this is our training input we use to “build a model”), and on the past shopping behavior of the particular customer (this is input X_i)

Robotics: autonomous car

The DARPA grand challenge in 2005:

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



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

... VIDEOS ...

Videos in teaching/videos_for_teaching

Start of the race: GrandChallenge1.flv, start at min. 1:50

Crashes: GrandChallenge1.flv, start at min 0:37

Beerbottle pass and final: GrandChallenge2.flv, start at min 4:16

Robotics: autonomous car (2)

How does such a thing “work”?

- ▶ lots of sensors, cameras, etc
- ▶ all of them generate signals
- ▶ Need to “extract information” from each sensor like
 - ▶ “here is a wall”
 - ▶ “here the slope is very steep”

Each such extraction step is done by machine learning.

- ▶ Now you get lots of second order information of all these detection systems. This information is very noisy, unreliable, might be contradicting. So you need to find a way to combine all this. This is done by machine learning as well.

All these things sound not so difficult, but the DARPA success is really the end of a very long way of trial and error.

Machine learning techniques were the crucial ingredient.

Robotics: robot playing table tennis

- ▶ Robot is supposed to do very complicated movements.
- ▶ Impossible to “hand-endcode” all the rules how to do this.

Example: playing table tennis

... VIDEO ... VIDEO ...

[teaching/videos_for_teaching/JanPeters_RobotLearning](#)

Lots of reinforcement learning is involved!

Bioinformatics

Machine learning is used all over the place in bioinformatics:

- ▶ Classify different types of diseases based on microarray data

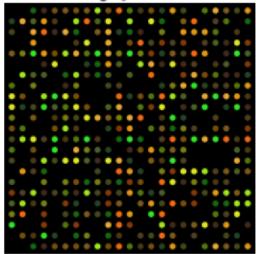
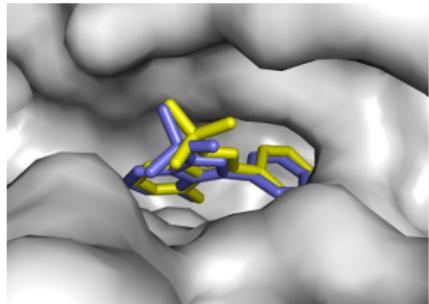
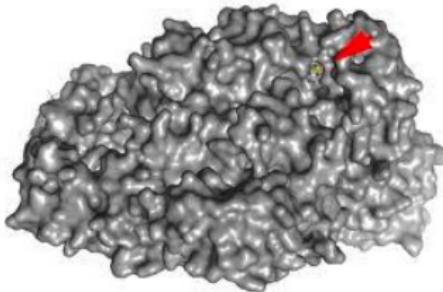


Image: Wikipedia

- ▶ Want to find proteins with “similar functions”.



Images: BiochemLabSolutions.com

Bioinformatics (2)

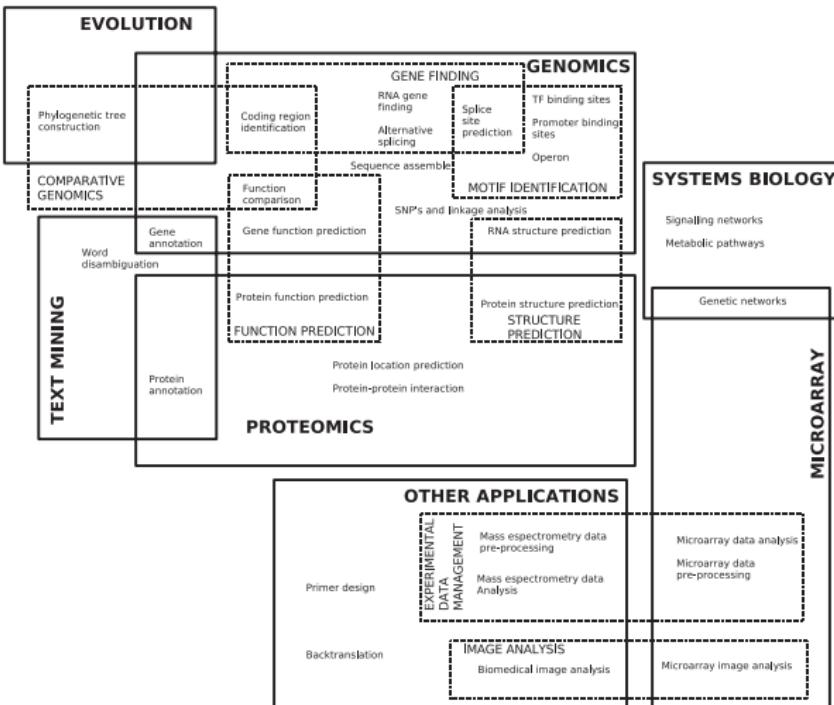
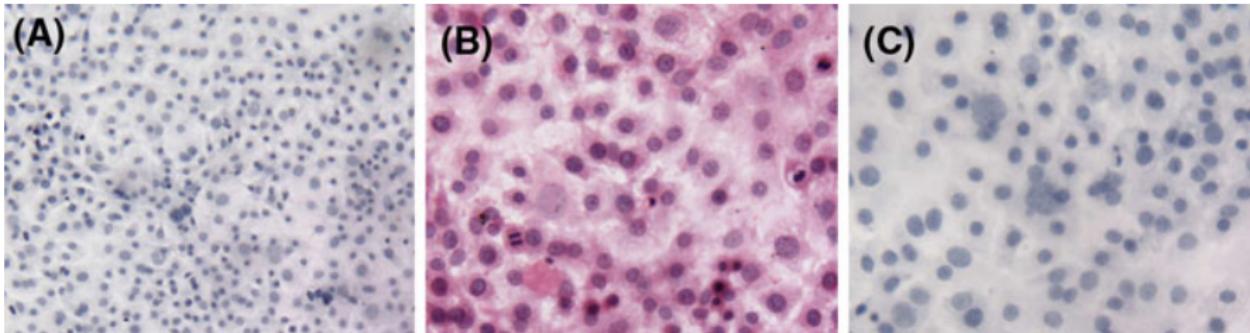


Figure from Larranaga et al., 2005

Medical image analysis

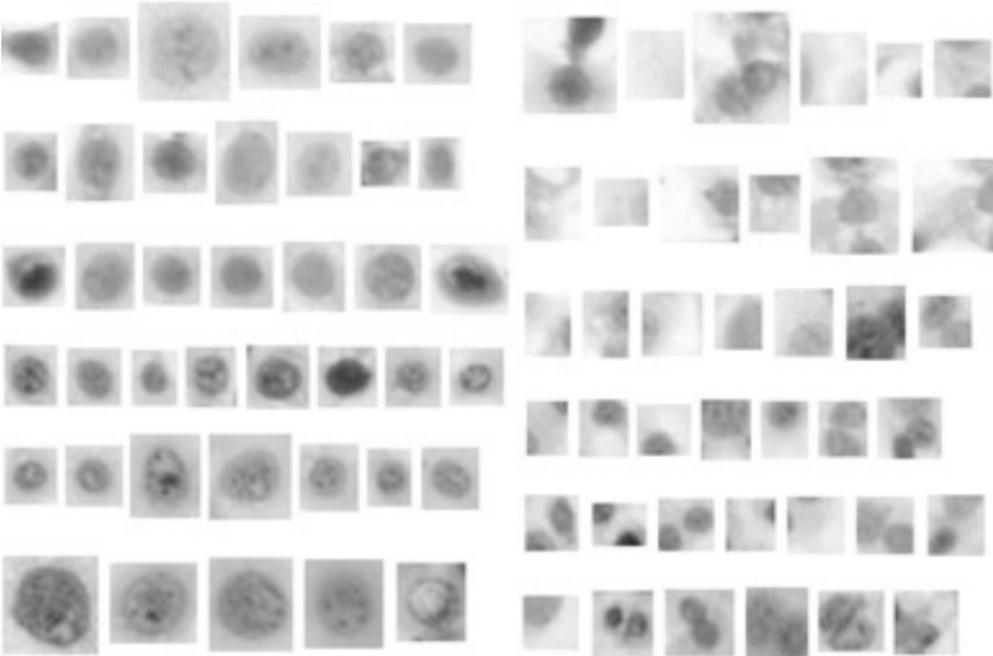
Example from cancer research: automatic detection and classification of cell nuclei in microscopy images:

Images look like this:



Medical image analysis (2)

The training data:

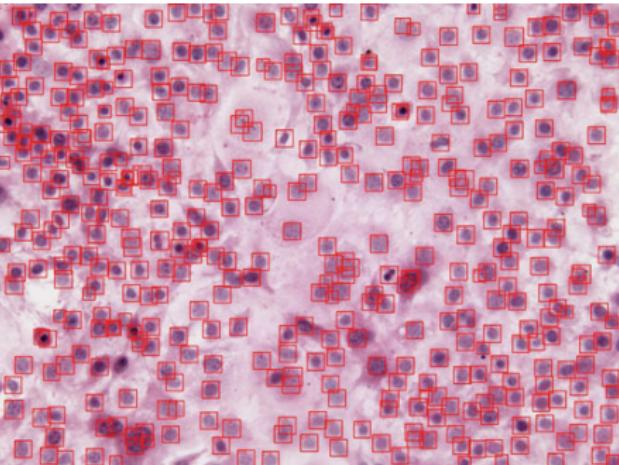


Positive training samples

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

Language processing (3)

VIDEO ... VIDEO ... VIDEO

watson_jeopardy.flv, start at min 17:36

Amazing, isn't it??? How can this work???

Internally, Watson is full of machine learning!

Last but not least, and I hate to mention it: NSA

The surveillance methods used by secret services such as the NSA are bound to use machine learning internally:

- ▶ You want to find “suspicious” activities out of billions of “normal” email
- ▶ This is a typical unsupervised learning problem ☹

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:

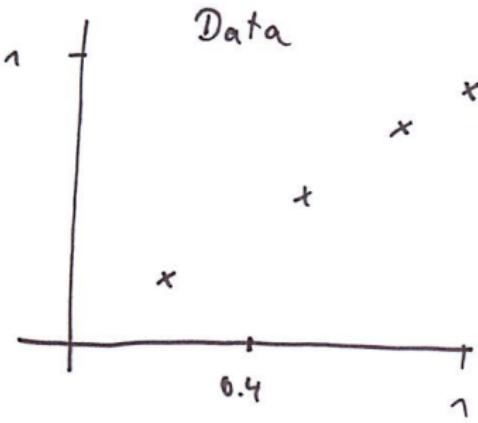
- ▶ We observe that the sun rises every day.
- ▶ So we predict that the sun is also going to rise the next day.
- ▶ To make this prediction, we don't need to understand why this is the case.

Why should machine learning work at all?

Consider the following simple classification examples:

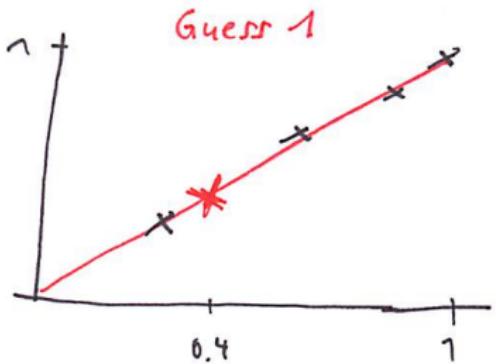
- ▶ Given: input-output pairs (X_i, Y_i) .
- ▶ Goal: learn to predict the Y -values from the X -values, that is we want to “learn” a suitable function $f : \mathcal{X} \rightarrow \mathcal{Y}$.

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



Why should machine learning work at all? (2)

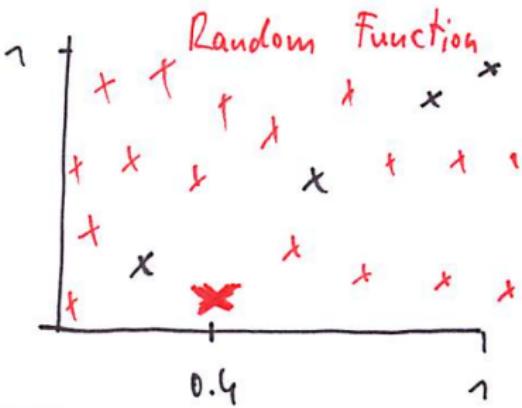
Here are two guesses:



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” (say, continuous and has a flat slope).

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' .
 - ▶ 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)

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 (4)

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 in fact not contained in \mathcal{F} after all.

In fact, 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.**

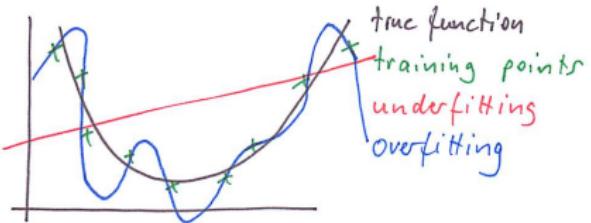
At the end of the course you will know all this, at least roughly ☺
Here is some more intuition:

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 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” .

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

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!

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

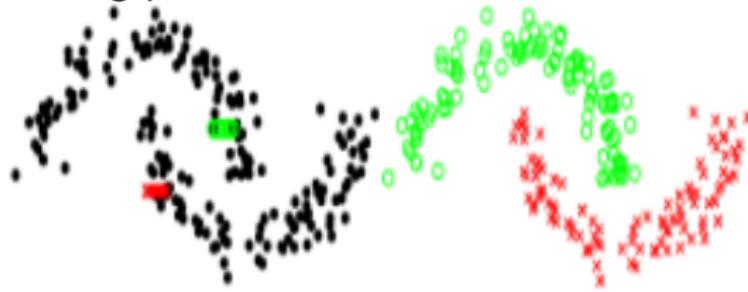
- ▶ **Outlier detection:** find data points that are “outliers”.

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 labels 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.

Reinforcement learning

Agent is supposed to learn how to interact and “behave optimally” in a complex environment with many possible actions and consequences. At the same time, he must get to know the environment and learn how to act in a useful way.

Examples:

- ▶ Rescue robots, say in a mining accident. Robot is supposed to find its way through the mine and accomplish certain tasks
- ▶ Control problems: We've seen the video of the robot arm that learns to throw the ball in the cup. At each point in time, the robot arm has “to decide” how to move and direct its forces in order to accomplish the task.
- ▶ Backgammon.

Reinforcement learning (2)

General model:

- ▶ Model the interaction of the agent with the environment
- ▶ At each point in time, agent is in a certain state, gets a certain amount of input (“observations”) and can perform actions that bring him to another state.
- ▶ Each action gives him a certain amount of reward.
- ▶ Goal is to learn a strategy such that he obtains as much accumulated reward as possible

Key problem: exploration — exploitation tradeoff

Reinforcement learning will be treated in the last part of the course by Dr. Norman Hendrich.

Warmup: k -nearest neighbor classification

The kNN algorithm for classification

Literature:

Hastie, Tibshirani, Friedman Section 2.3.2

Duda, Hart Section 4.5

Setting up 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.

Setting up 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 this particular point:

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

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

Setting up a simple machine learning experiment (3)

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

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

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

Setting up 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 this particular point:

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

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

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

Setting up a simple machine learning experiment (5)

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.
- ▶ We will see in the next lecture how typically training and test errors behave.

The kNN classifier

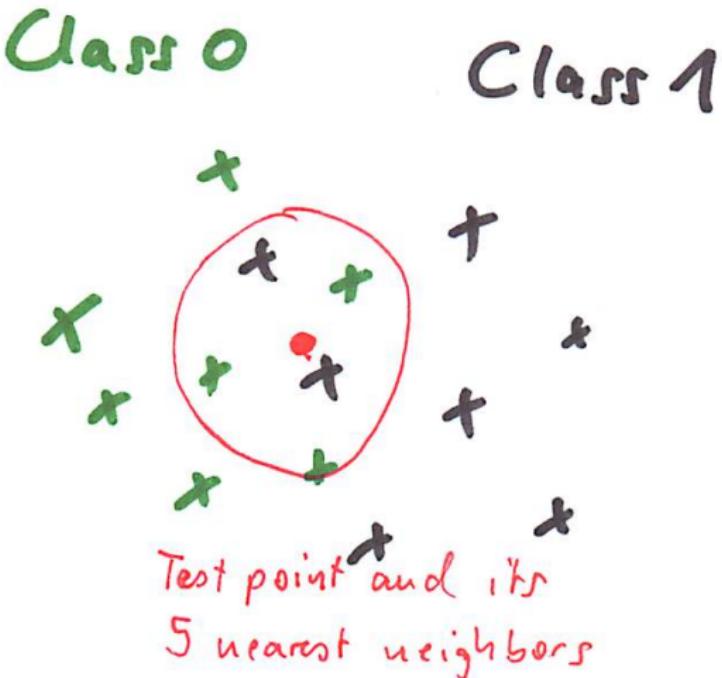
Given: Training points $(X_i, Y_i)_{i=1,\dots,n} \subset \mathcal{X} \times \{0, 1\}$ and a distance function $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$.

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

- ▶ 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 $kNN(X')$.
- ▶ 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



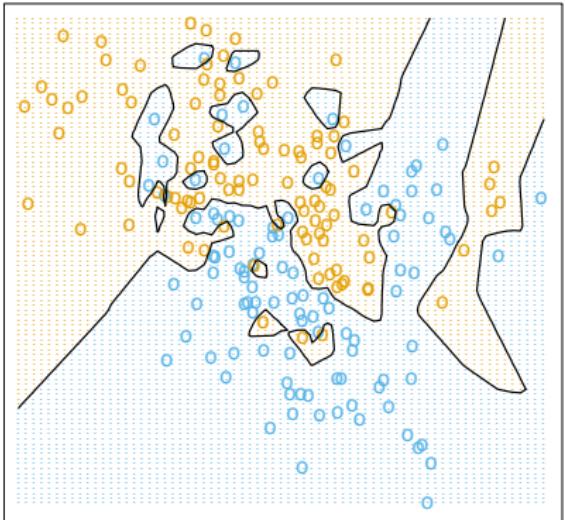
Influence of the parameter k

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

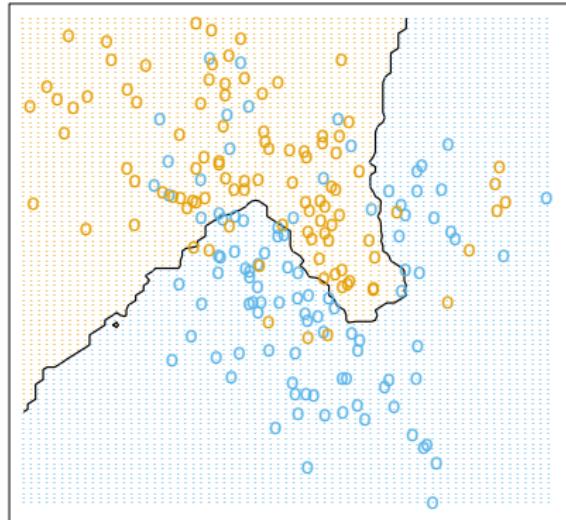


Figure from Hastie/Tibshirani/Friedman

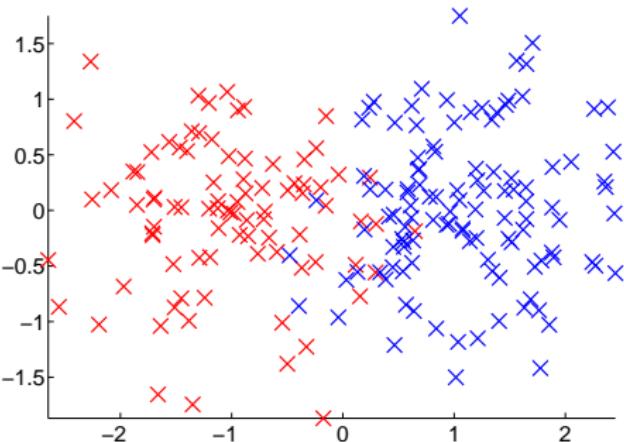
Influence of the parameter k (3)

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)
- ▶ Based on theoretical considerations: k should be about order $\log n$ as $n \rightarrow \infty$ (see later in the course)

Application: simple mixture of Gaussians

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



Application: simple mixture of Gaussians (2)

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 (3)

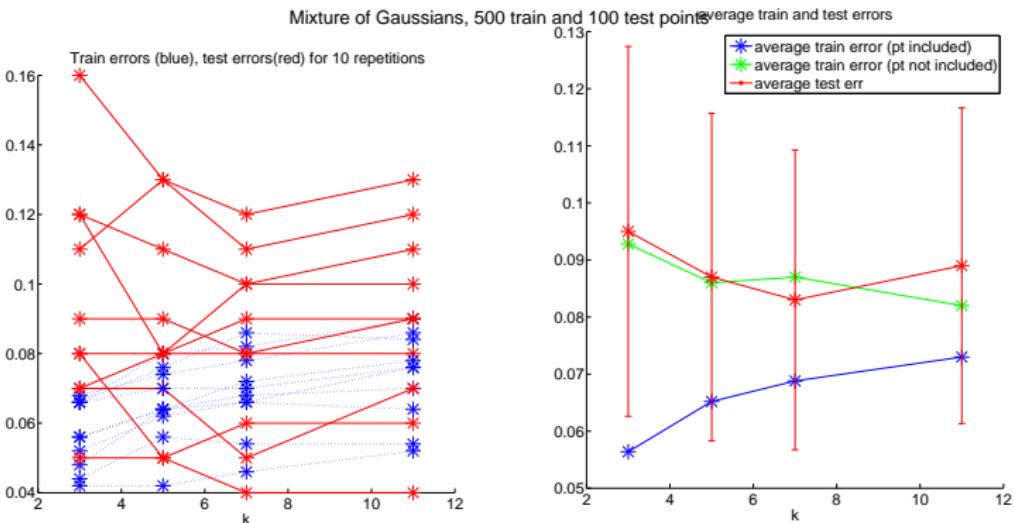
Note:

- ▶ 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 (4)

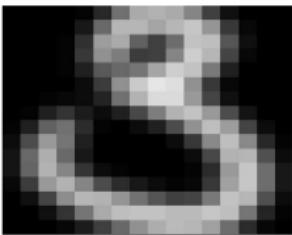
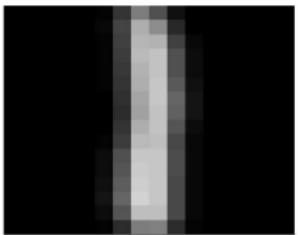
The following figure shows these train and test errors:

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



Application: hand written digits

Data:



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

Task 1: Learn to distinguish between 1 and 8

Task 2: Learn to distinguish between 3 and 8

Application: hand written digits (2)

Setup:

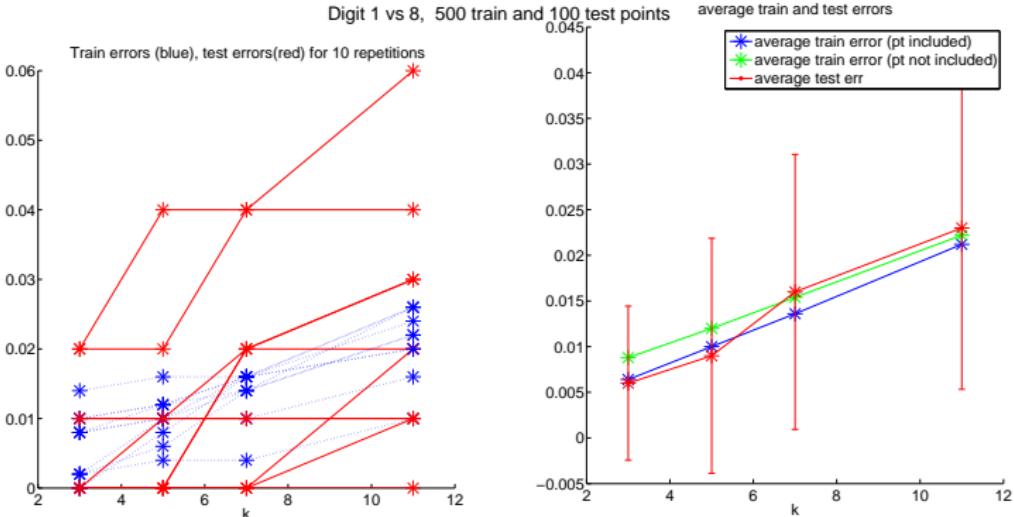
- ▶ To apply the kNN rule we need to define a distance function between digits.
- ▶ For simplicity, we use the Euclidean distance between the vectors:

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

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

Application: hand written digits (3)

Results task 1 (digit 1 vs digit 8):

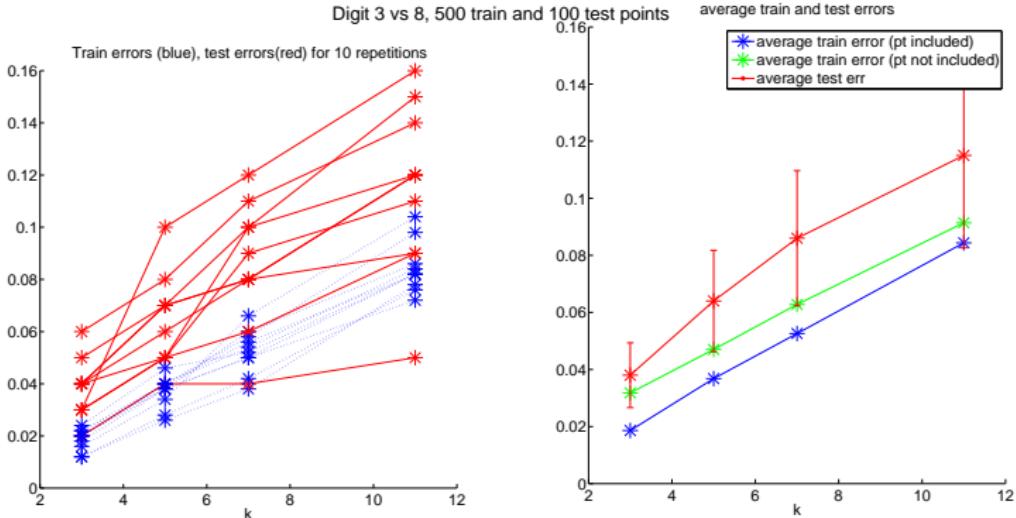


(x -Axis: parameter k , y -axis: error)

WHAT DO YOU THINK, IS THIS GOOD OR BAD?

Application: hand written digits (4)

Results task 2 (digit 3 vs digit 8):



These results are surprisingly good!!!

Influence of the similarity function

The choice of the similarity function is crucial as well:

- ▶ The performance of kNN rules 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).

Recap: Maths basics (Linear algebra and probability theory, slides in the appendix)

Supervised learning

Formal setup

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)

The statistical setup

- ▶ Let \mathcal{X} and \mathcal{Y} be some input and output space.
Our goal will be to learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$.
- ▶ Consider a **probability distribution P** over $\mathcal{X} \times \mathcal{Y}$. All training and test points are going to be sampled according to this probability distribution. P is the joint distribution over (input,output)-pairs.
- ▶ In particular, note that the output values Y_i are not necessarily deterministic functions of X_i . Instead, the output can be non-deterministic or noisy.

The statistical setup (2)

Example:

- ▶ You want to classify people as “male” or “female” based on their height.
- ▶ X = height of the person, Y is the label “male” or “female”.
- ▶ Obviously, the label cannot be deterministic (there exist both men and women of height 170 cm, say).

The statistical setup (3)

At the time of learning, the probability distribution P is unknown, all we get are samples $(X_i, Y_i)_{i=1,\dots,n}$ from this distribution.

But let us first see how learning would work if we knew all distributions.

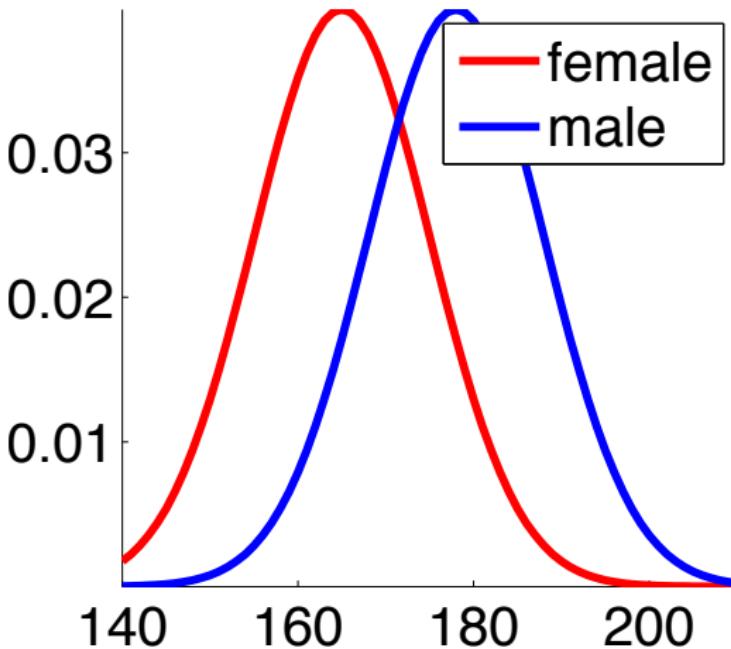
Case of known distribution

Male vs. female

Let's continue with the example to classify "male" vs. "female". We consider a couple of different approaches.

Male vs. female (2)

Class conditionals $P(X | Y)$



GIVEN THIS INFORMATION, HOW WOULD YOU LABEL THE

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

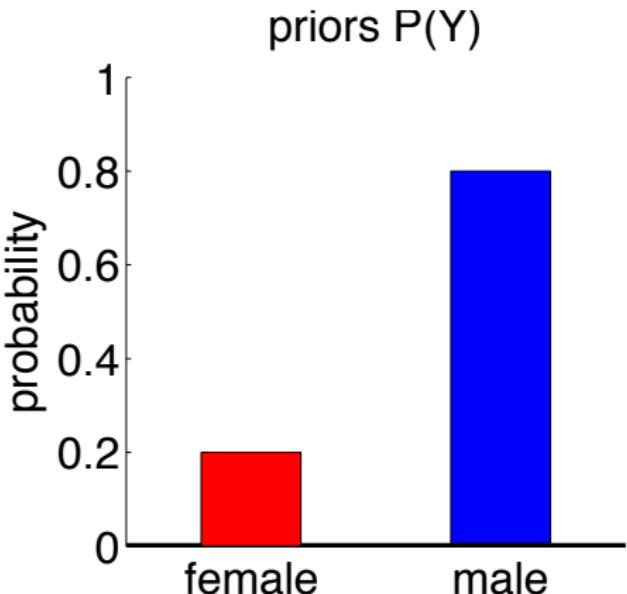
Decide based on prior probabilities $P(Y)$.

- ▶ 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

$$f_n(X) = \begin{cases} 1 & \text{if } P(Y = 1) > P(Y = 0) \\ 0 & \text{otherwise} \end{cases}$$

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

Visually: select the higher bar



Approach 2: maximum likelihood principle

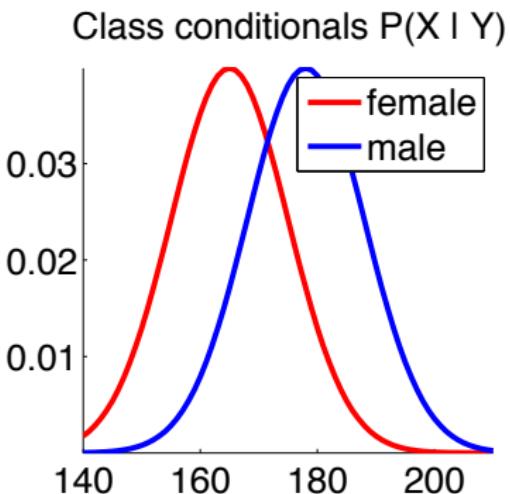
Decide based on the likelihood functions $P(X|Y)$ (maximum likelihood approach).

- ▶ Compute the **class conditional distributions** $P(X|Y=1)$ and $P(X|Y = 0)$.
- ▶ Then predict the label with the higher likelihood:

$$f_n(x) = \begin{cases} 1 & \text{if } P(X = x|Y = 1) > P(X = x|Y = 0) \\ 0 & \text{otherwise} \end{cases}$$

Approach 2: maximum likelihood principle (2)

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

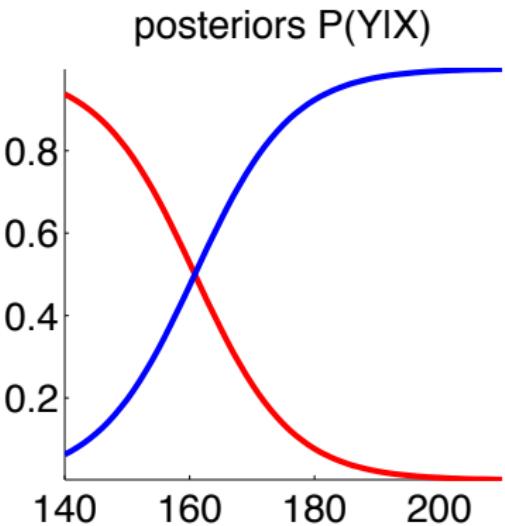
$$P(Y = 1|X = x) = \frac{P(X = x|Y = 1) \cdot P(Y = 1)}{P(X = x)}$$

- ▶ Predict by the following rule:

$$f_n(x) = \begin{cases} 1 & \text{if } P(Y = 1|X = x) > P(Y = 0|X = x) \\ 0 & \text{otherwise} \end{cases}$$

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

$$\ell(\text{predict } a_i \text{ if true label is } Y = y_i)$$

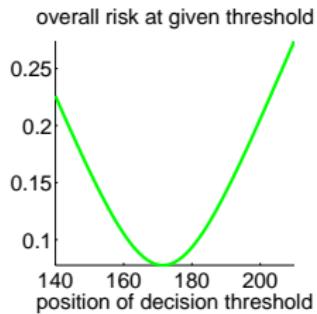
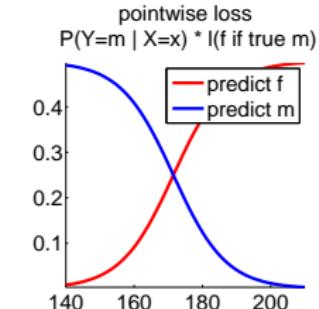
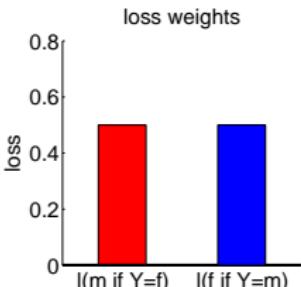
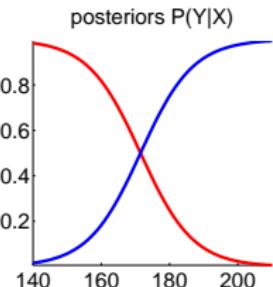
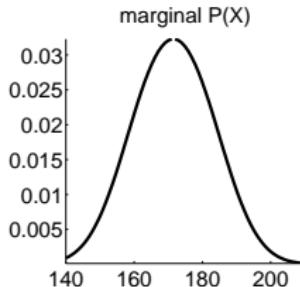
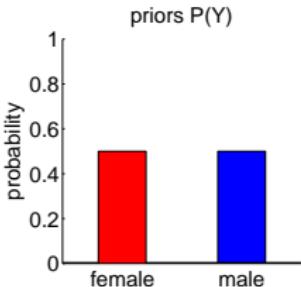
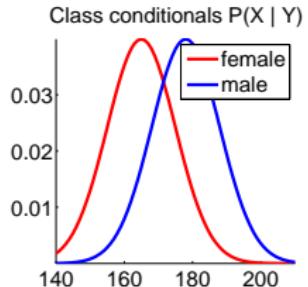
that specifies the costs of predicting a_i when the true label is y_i

- ▶ Consider the expected conditional risk

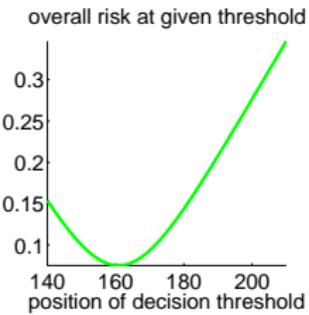
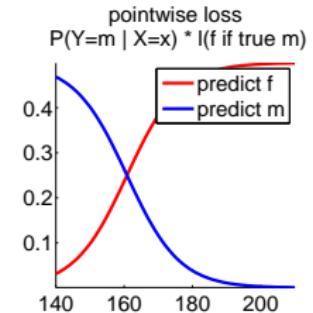
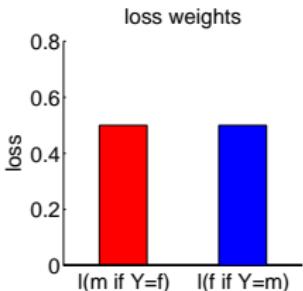
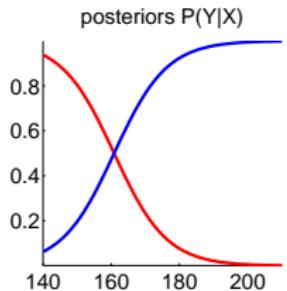
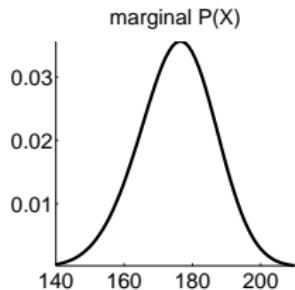
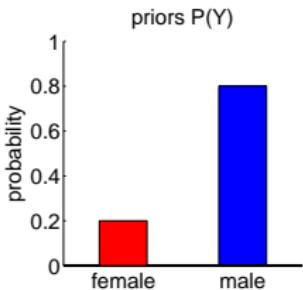
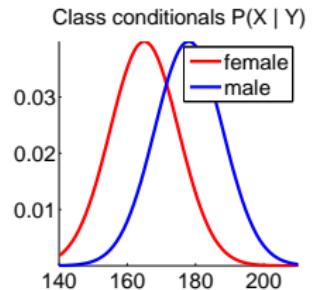
$$\begin{aligned} R(a_i|X=x) &= \ell(a_i \text{ if true } Y=1)P(Y=1|X=x) \\ &\quad + \ell(a_i \text{ if true } Y=0)P(Y=0|X=x) \end{aligned}$$

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

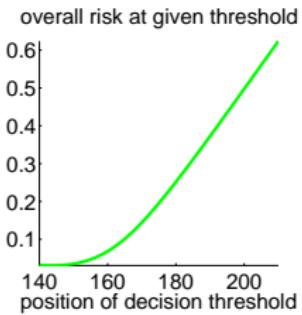
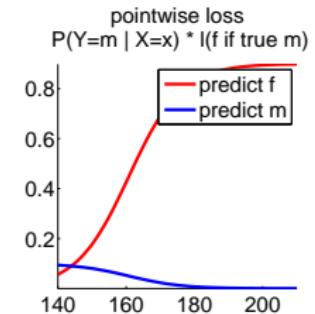
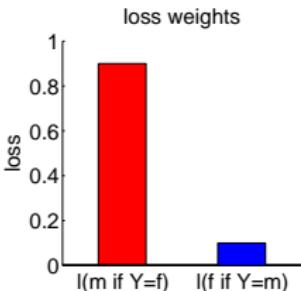
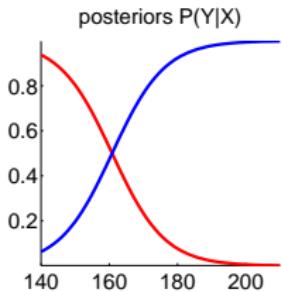
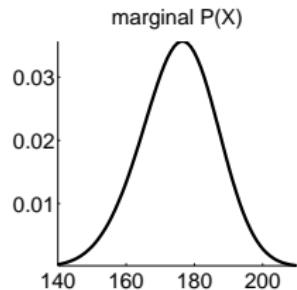
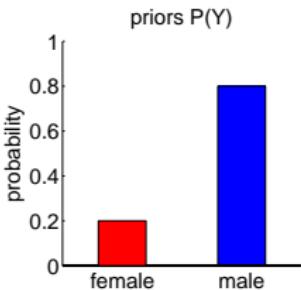
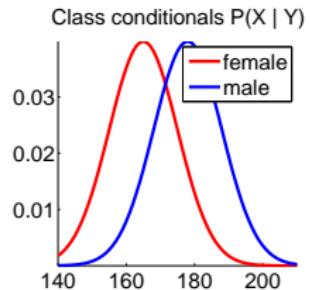
Example: male vs female



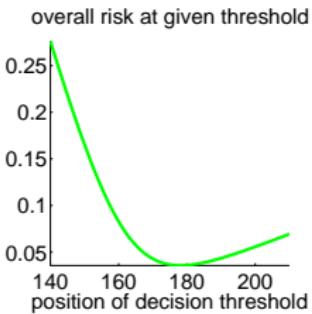
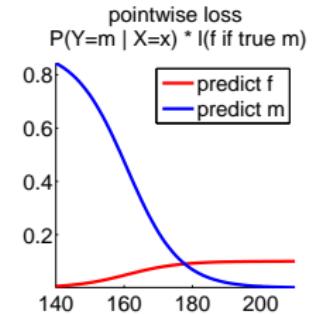
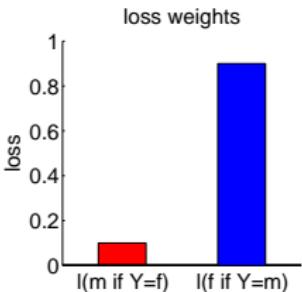
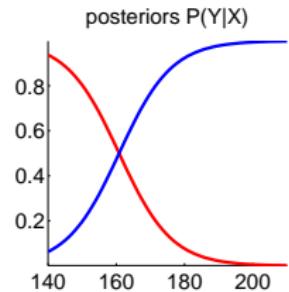
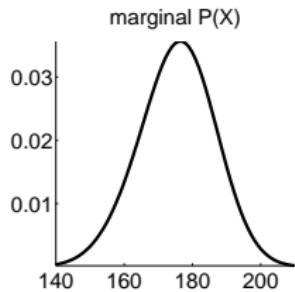
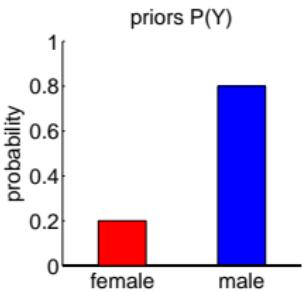
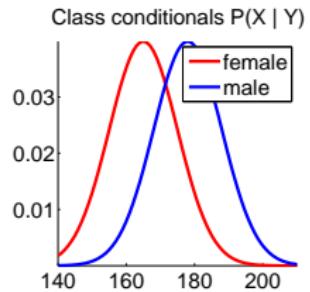
Example: male vs female (2)



Example: male vs female (3)



Example: male vs female (4)



What if P is unknown?

At the time of learning, the probability distribution P is unknown, all we get are samples $(X_i, Y_i)_{i=1,\dots,n}$ from this distribution.

If we want to argue in the framework outlined above, we need to use a **generative approach**:

What if P is unknown? (2)

- ▶ We make certain model assumptions on the underlying distributions (e.g., classes are Gaussians), that is we assume that the true probability distribution comes from a certain set \mathcal{P} of distributions.
- ▶ Then we use the training data to select the best model:
 - ▶ Maximum likelihood approach: we do not prefer certain $P \in \mathcal{P}$ over others and simply select the distribution that gives the highest likelihood to the data:

$$P^* = \underset{P}{\operatorname{argmax}} \in \prod_{i=1}^n P(Y_i \mid X = X_i)$$

- ▶ Sometimes people also want to give certain preferences to $P \in \mathcal{P}$ in form of a prior distribution. This leads then to a Bayesian approach: XXX SKIPPED XXX
- ▶ Then we classify using the approaches of above.

Risk minimization framework: loss, risk, ERM, RRM

Literature:

The proofs of the classification section are taken from:

Devroye, Lugosi, Györfi: Probabilistic theory of pattern recognition
(first pages of Section 2).

The proofs of the regression section are taken from:

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

Settings for all supervised learning problems

Discriminative approach

The generative approach we have seen in the last part is often difficult in practice, and sometimes overkill.

We now want to settle for a **discriminative approach**:

Setting:

- ▶ Consider a probability distribution P over $\mathcal{X} \times \mathcal{Y}\{0, 1\}$.
- ▶ We want to learn a classifier $f : \mathcal{X} \rightarrow \{0, 1\}$ that makes “as little errors as possible”.
- ▶ In particular, we do not want to explicitly model the underlying probability distributions.

We now want to formalize what it means to have “as little error as possible”.

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

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

Loss function (2)

Note:

- ▶ In some applications, it is important that the loss also depends on x (for example, in a medical context)
- ▶ In some applications, it is important that the loss depends on the order of y and y' (for example, in spam classification)

True Risk

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

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

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

Bayes risk

What is the best function we can think of?

- The Bayes risk is defined as

$$R^* := \inf_f \{ R(f) \mid f \text{ measurable} \}$$

(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^* is called the Bayes learner.

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

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

$$\lim_{n \rightarrow \infty} R(f_n) = R^*$$

(where the convergence is in probability, for those who know what that means).

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

Consistency (2)

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 (3)

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

Since then quite a number of 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.

Two principles

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 the one that has the smallest empirical risk:

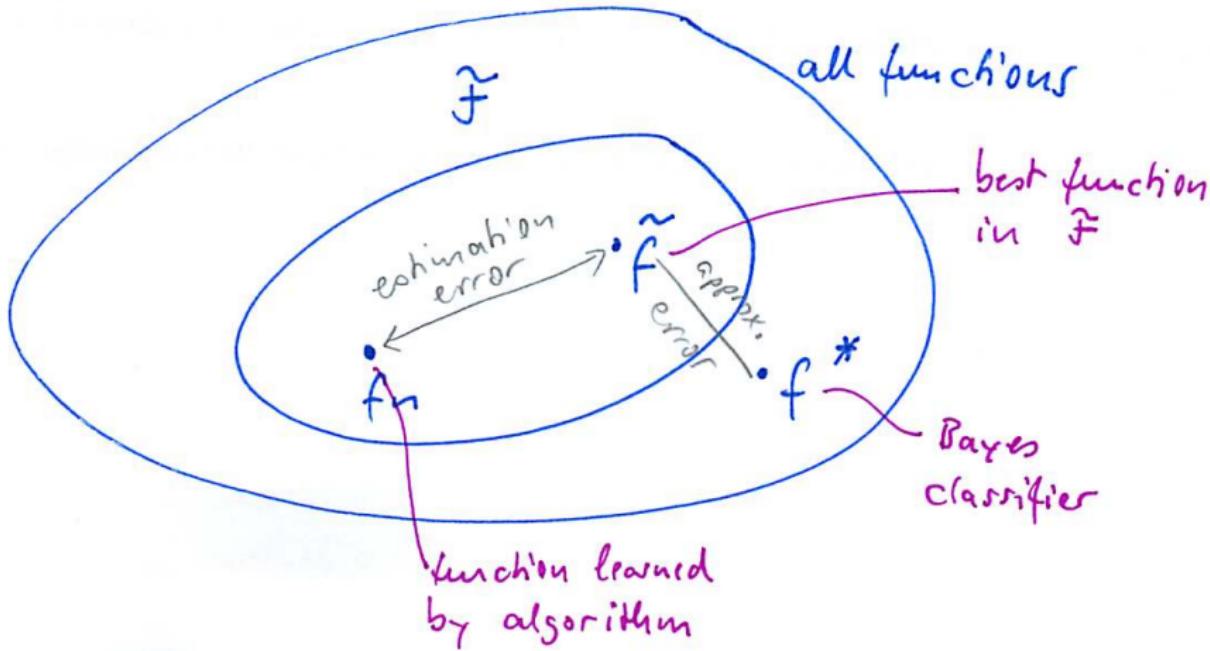
$$f_n := \operatorname{argmin}_{f \in \mathcal{F}} R_n(f)$$

Empirical risk minimization (3)

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} .

Empirical risk minimization (4)



Empirical risk minimization (5)

Coming back to the words 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.

Empirical risk minimization (6)

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** $\Omega(f)$ that measures how “complex” a function is. Examples:
 - ▶ $\Omega = \text{degree of the polynomial}$
 - ▶ $\Omega = \text{maximal slope of a differentiable function}$
- ▶ Then choose $f \in \mathcal{F}$ to minimize the **regularized risk**

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

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.

Particular setup for classification

A list of loss functions for classification

- ▶ The most important loss function for classification is the **0-1-loss**, defined as

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

- ▶ However, it can be hard to minimize zero-one-losses (\leadsto discrete optimization problem, NP hard).
- ▶ For this reason, all successful machine learning algorithms do the following two things:
 - ▶ Instead of a binary output they return real valued outputs, and use the sign of the output as the final label.
 - ▶ They consider a convex relaxation, a “surrogate loss function”, of the zero-one-loss function.

A list of loss functions for classification (2)

The most important convex surrogate loss functions are:

- ▶ **Hinge loss (soft margin loss)**, used in support vector machines:

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

- ▶ **exponential loss**, used in boosting: $\exp(-y \cdot f(x))$
- ▶ **logistic loss**, used in logistic regression: $\log(1 + \exp(-y \cdot f(x)))$

A list of loss functions for classification (3)

Remarks:

- ▶ Surrogate loss functions lead to optimization problems that are computationally tractable. But do they also give the correct result? Does it hurt to replace the 0-1-loss by a surrogate?

This problem has been solved in *Bartlett, Jordan, McAuliffe: Convexity, classification and risk bounds. Journal of the American Statistical Association, 2006.*

If time permits, we revisit these issues later in the course.

Regression function (context of 0-1-loss classification)

Consider a probability distribution over $\mathcal{X} \times \{0, 1\}$. We want to describe this distribution in terms of two other quantities:

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

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

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

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

Regression function (context of 0-1-loss classification) (2)

Intuition:

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

Regression function (context of 0-1-loss classification) (3)

Proposition 1 (Unique decomposition)

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

Intuition (discrete case): We can rewrite

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

and similarly

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

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

Regression function (context of 0-1-loss 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

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

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

Now consider the following classifier:

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

Explicit form of the Bayes classifier (2)

Theorem 2 (f° is the Bayes classifier)

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

Remark:

- ▶ The theorem shows that $f^\circ = f^*$.
- ▶ Consequence: in the particular case of classification with the 0-1-loss, we have an explicit formula for the Bayes classifier.

Explicit form of the Bayes classifier (3)

Proof:

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

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

To get from second to third line, observe that $f(x)$ is a deterministic function.

Explicit form of the Bayes classifier (4)

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

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

To see the last step:

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

Explicit form of the Bayes classifier (5)

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

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

Because this holds for any individual value of x , it also holds on average 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.

Naive approach based on the Bayes classifier

Plugin-approach. 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. ☹

Particular setup for regression

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:

- ▶ Squared loss (L_2 -loss): $\ell(x, f(x), y) = (f(x) - y)^2$
- ▶ L_1 -loss function: $\ell(x, f(x), y) = |f(x) - y|$
- ▶ ε -insensitive loss function:

$$\ell(x, f(x), y) = \begin{cases} |f(x) - y| - \varepsilon & \text{if } |f(x) - y| \geq \varepsilon \\ 0 & \text{otherwise} \end{cases}$$

- ▶ Huber's robust loss function:

$$\ell(x, f(x), y) = \begin{cases} \frac{1}{2\varepsilon}(f(x) - y)^2 & \text{if } |f(x) - y| \geq \varepsilon \\ |f(x) - y| - \varepsilon/2 & \text{otherwise} \end{cases}$$

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)

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

Regression function (context of L_2 regression)

(4)

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



Explicit form of optimal solution under L_2 loss

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

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

Theorem 4 (Explicit form of optimal L_2 -solution)

The function f° minimizes the L_2 -risk.

Proof. Follows directly from Proposition 3:

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



Bias-Variance tradeoff

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:

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

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).

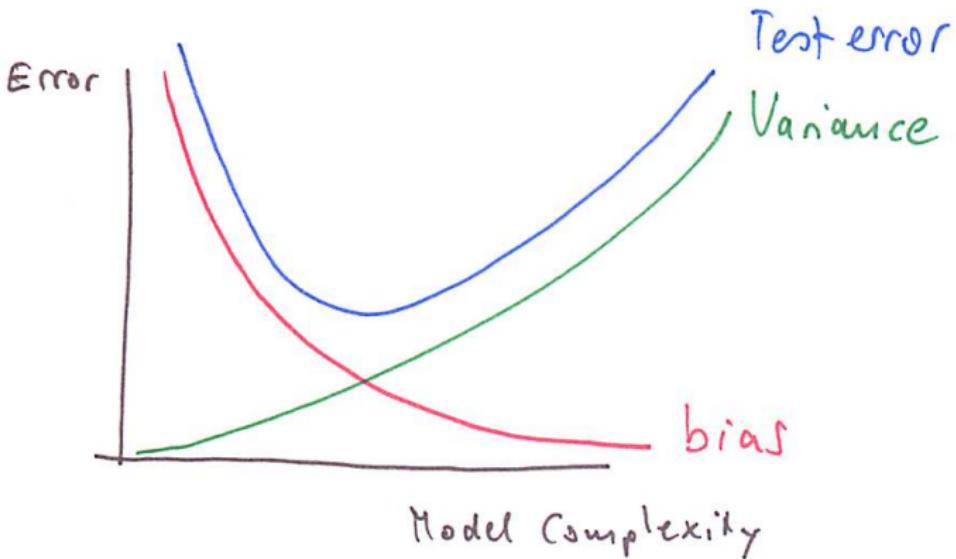
Proof. skipped, see the book (see Gyorfi, Kohler, Krzyzak, Walk:
Distribution-free theory for nonparametric regression, p.2)

Bias-Variance tradeoff (2)

Intuition:

- ▶ The variance term measures how much the regression function f_n varies with the noise in the data.
It has the same connotation as the estimation error we discussed earlier.
- ▶ The bias term measures how good the approximation is in case we don't have noise.
It has the same connotation as the approximation error we discussed earlier.

Bias-Variance tradeoff (3)



(*) Generative vs. discriminative approaches
SKIPPED

Intro: ERM vs Bayesian decision theory

Have seen two approaches to select a good classifier:

- ▶ Generative approach (fit a probability model and apply bayesian decision theory)
- ▶ Empirical risk minimization (minimize a loss function over a certain class of functions)

Are these two approaches related or completely different?

Intro: ERM vs Bayesian decision theory (2)

Generative approach:

- ▶ we can make **assumptions on the underlying probability distribution**
- ▶ We cannot not make any assumptions on the function class that realizes these different classifiers!

Discriminative approach:

- ▶ we can make **assumptions on the function class \mathcal{F} and the loss function.**
- ▶ We do not make assumptions on the probability distribution!

In some cases, we can bring both aspects together: in the Bayesian framework, a particular probabilistic assumption leads to the same classifier as assumptions on a function class in the ERM framework.

Generative ML \implies discriminative ERM

Generative approach based on maximum likelihood:

- ▶ Assume that the data distribution comes from a family \mathcal{P} of distributions.
- ▶ If we know which was the true distribution P , we would simply classify according to the corresponding regression function.
- ▶ Instead, we now first have to estimate the distribution, say by maximum likelihood:

$$P_0 := \operatorname{argmax}_{P \in \mathcal{P}} \prod_{i=1}^n P(Y_i \mid X = X_i)$$

Generative ML \implies discriminative ERM (2)

Discriminative interpretation 1:

- ▶ The set of probability distributions \mathcal{P} gives rise to a set of corresponding regression functions. Call this set \mathcal{N} . Now define the set $\mathcal{F} := \{\mathbb{1}_{\eta \geq 0.5} \mid \eta \in \mathcal{N}\}$.
- ▶ The set \mathcal{F} is the set of decision functions that corresponds to the regression functions in η .

Assumptions on the family of probability distributions in a generative approach induce a specific function class in the discriminative approach.

Generative ML \implies discriminative ERM (3)

Discriminative interpretation 2:

- ▶ The generative approach first selects the probability distribution that best represents the data, and then it classifies according to the regression function.

$$\max_{P \in \mathcal{P}} \prod_{i=1}^n P(Y_i \mid X = X_i)$$

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

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

XXX f is here the regression function corresponding to the distribution XXXX

Generative ML \implies discriminative ERM (4)

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

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

Fitting a particular model to the data corresponds to minimizing a particular loss function over a particular function class \mathcal{F} .

Generative ML \implies discriminative ERM (5)

What does this tell us?

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

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

Note that it does not always work the other way round (we cannot find a probability distribution for any kind of loss function ℓ).

Generative ML \implies discriminative ERM (6)

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.

Examples

Linear discriminant analysis:

- ▶ Assume that our classes are normally distributed with the same covariance structure.
- ▶ Consider the ratio $P(Y = 1 \mid X = x)/P(Y = 0 \mid X = x)$.
- ▶ Computations show that
 - ▶ the decision boundary is always a hyperplane
 - ▶ Among all possible hyperplanes, the true decision boundary is the one that minimizes the squared loss (in a certain sense, see later).

Generative MAP \implies RRM

A similar reasoning can show:

- ▶ if we select the probability distribution not according to maximum likelihood, but according to a Bayesian approach, then the approach corresponds to regularized risk minimization.
- ▶ The loss is defined as above
- ▶ The regularizer is the log-prior distribution of the parameter.

The No-Free-Lunch Theorem

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 f_j when the true function is f_i .

| estimated
^{true} | f_1 | f_2 | f_3 | f_4 | \dots | f_{2^m} |
|---|-------|-------|-------|-------|---------|-----------|
| f_1 | | | | | | |
| f_2 | | | | | | |
| f_3 | | | | | | |
| f_4 | | | | | | |
| : | | | | | | |
| f_{2^m} | | | | | | |

entries (r_{ij}) = risk if true
function is f_i
and estimated
function is f_j

NFL, simple version (3)

Proposition 5 (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 $f_i = \hat{f}_j$)
- ▶ $r_{ij} = 1/m$ exactly m times
- ▶ $r_{ij} = 2/m$ exactly $\binom{m}{2}$ times
- ▶ ...



NFL, simple version (4)

Proposition 6 (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 7 (Simple NFL with training data)

In the model introduced above: Assume we are given a training set $(X_i, Y_i)_{i=1,\dots,n}$. Then, on average over all test distributions, all classifiers that are consistent with the training set perform the same.

Proof.

- ▶ In the table above, eliminate all columns that are not consistent with the training data.
- ▶ Among the remaining ones, all distributions over test labels are possible.
- ▶ Then the result follows by a similar argument as above. ☺

NFL, simple version (6)

Note that much more general theorems exist, for example for the standard machine learning scenario where we draw data from joint distribution P on $\mathcal{X} \times \mathcal{Y}$ and \mathcal{X} is any space you want ...

NFL, simple version (7)

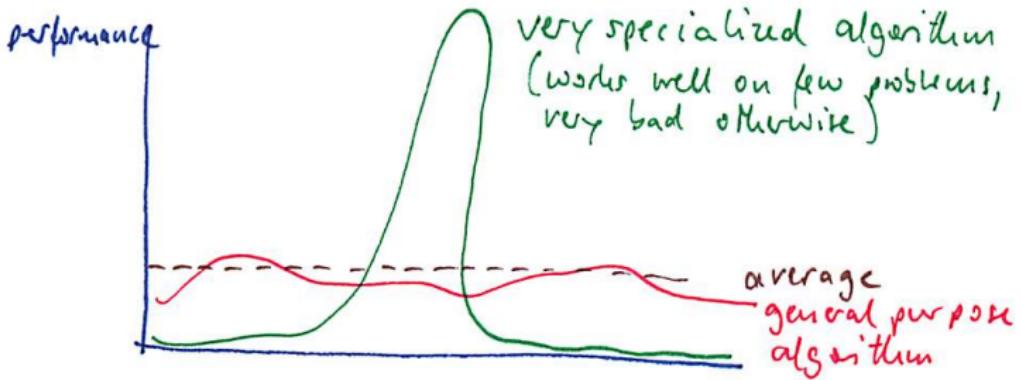
Discussion:

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

NFL, simple version (8)

- ▶ No: the key is that in practice we do not see “all possible data sets”. As soon as we make assumptions on the data sets, the NFL breaks down (“Making assumptions” means to delete some columns from the above matrix, and then the proof breaks down).
- ▶ This shows once more how important it is to incorporate these assumptions to the machine learning algorithm \leadsto inductive bias!

NFL, simple version (9)



The integral over all these curves is exactly
the same by NFL.

History / Literature

- ▶ *Wolpert, David. The Lack of A Priori Distinctions between Learning Algorithms. Neural Computation, 1996.*
- ▶ *Many generalizations since then (both to the field of machine learning and optimization in general).*
- ▶ *Ho and Pepyne. Simple explanation of the no-free lunch theorem and its implications. Journal of Optimization Theory and Applications, 2002.*
- ▶ *Chapter 7 in Devroye/ Györfi / Lugosi*

Linear Methods for regression

Linear least squares regression

Literature:

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

Starting point

- ▶ We want to solve a regression problem with respect to the least squares loss (also called L_2 -loss).
- ▶ Have seen: if we optimized over the space of all (measurable) functions and had infinite amount of data, the solution would be given by the regression function $\eta(x) = E(Y \mid X = x)$.
- ▶ But now: only have a finite sample, and we don't want to optimize over all functions (overfitting). So we settle for a small function space and just consider linear functions.

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)

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 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:

$$f(X) = \frac{2}{10} \text{height} + 0 \cdot \text{weight} + 0 \cdot \text{income} + 0 \cdot \text{children}$$

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:

$$X = \begin{pmatrix} X_{11} & \cdots & X_{1d} \\ \vdots & & \vdots \\ X_{n1} & \cdots & X_{nd} \end{pmatrix} \Bigg\}^n \quad d \quad y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \Bigg\}^n \quad w = \begin{pmatrix} w_1 \\ \vdots \\ w_d \end{pmatrix} \Bigg\}^d$$

- Notation: the i -th training point is X_i , its entries are denoted as X_{i1}, \dots, X_{id} .
- Now we can write much more concisely:

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

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

$$\frac{1}{n} \sum_{i=1}^n \left(Y_i - (\langle w, X_i \rangle + b) \right)^2$$

Getting rid of b

We want to write the problem even more concisely.

- ▶ Define the matrices

$$\tilde{X} = \begin{pmatrix} X_{11} & \cdots & X_{1d} & 1 \\ \vdots & & \vdots & \vdots \\ X_{n1} & \cdots & X_{nd} & 1 \end{pmatrix} \quad \left. \right\} n \quad \tilde{w} = \begin{pmatrix} w_1 \\ \vdots \\ w_d \\ b \end{pmatrix} \quad \left. \right\} d+1$$

- ▶ Then we have

$$\langle \tilde{w}, \tilde{X}_i \rangle = \sum_{k=1}^{d+1} \tilde{w}_k \tilde{X}_{ik} = \sum_{k=1}^d w_k X_{ik} + b$$

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

Getting rid of b (2)

- ▶ Want to find w, b of the original problem $\#\#$.
- ▶ Transform the problem as on the last slide:
find a vector $\tilde{w} \in \mathbb{R}^{d+1}$ such that $\frac{1}{n} \sum_{i=1}^n (Y_i - \langle \tilde{w}, \tilde{X}_i \rangle)^2$ is minimized.
- ▶ From the solution \tilde{w} reconstruct the vector w (first d coordinates of \tilde{w}) and the intercept b (last coordinate of \tilde{w}).

So without loss of generality from now on we consider the simplified problem that does not involve the intercept b :

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

$$\frac{1}{n} \sum_{i=1}^n (Y_i - \langle w, X_i \rangle)^2 = \frac{1}{n} \|Y - Xw\|^2$$

ML \leadsto Optimization problem

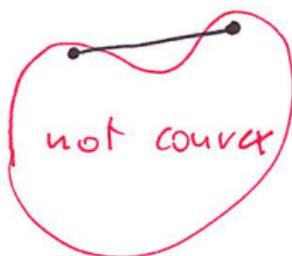
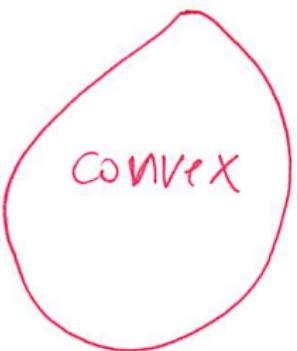
We can see:

- ▶ 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.

Excursion: convex optimization problems

Convex sets:

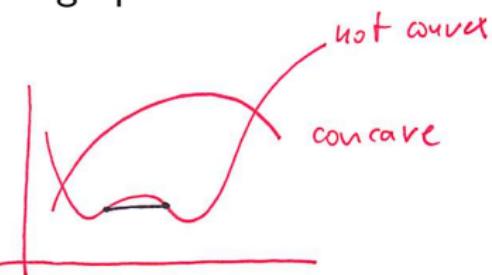
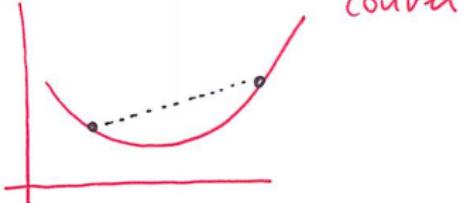
- ▶ A subset S of a vector space is called convex if for all $x, y \in S$ and for all $t \in [0, 1]$ it holds that $tx + (1 - t)y \in S$.
- ▶ In words: for any two points $x, y \in S$, the straight line connecting these two points is contained in the set S .



Excursion: 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.



Examples:

Excursion: convex optimization problems (3)

- ▶ 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.

Excursion: convex optimization problems (4)

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 and g_i ($i = 1, \dots, k$) are all convex.

- ▶ Convex optimization problems have the desirable property that any local minimum is already a global minimum.

Least squares regression is convex

Proposition 8 (Least squares is convex)

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

Proof. Exercise

Solution, part 1

Recap:

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

Solution, part 1 (2)

Theorem 9 (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.

- ▶ We need to 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, part 1 (3)

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

- ▶ Objective function: $Obj(w) := \frac{1}{2} \|Y - Xw\|^2$
- ▶ Derivative: $\frac{\partial Obj(w)}{\partial w} = -X^t(Y - Xw).$
- ▶ Setting this to zero gives the necessary condition
 $X^t Y = (X^t X)w.$
- ▶ We always have $rank(X) = rank(X^t X) = rank(XX^t)$. In particular, under the assumption that X has rank d , the matrix $X^t X$ is of full rank, hence invertible.
- ▶ So we can solve for $w = (X^t X)^{-1} X^t Y$.
- ▶ Now we need to figure out whether this is indeed a minimum. To this end, consider the Hessian matrix that contains all second derivatives: $H(Obj) = \frac{\partial^2 Obj}{\partial w \partial w'} = X^t X$.

Solution, part 1 (4)

- ▶ This matrix is positive definite: obviously all eigenvalues ≥ 0 , and because of the rank condition we have > 0 . So the solution we computed above is indeed a local minimum.



Solution, part 2

Recap:

- ▶ Generalized inverse of a matrix

Solution, part 2 (2)

Theorem 10 (Solution, case $\text{rank}(X) < d$)

Assume that X has $\text{rank} < d$. 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 . This solution is not unique. If w_1, w_2 are two 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!)
- ▶ However, w is not unique: Let w be a solution and v any vector with $Xv = 0$ (exists because X has $\text{rank} < d$). Then $w + v$ is a solution as well.

Solution, part 2 (3)

- XXXX TO DO FOR 2015: more explicit proof of the claim that all solutions do the same to training points XXX ☺

Implementation in Matlab

- ▶ Never compute the solution using the `inv` function!
- ▶ Instead, solve the linear system $X^t Y = (X^t X)w$ for w .
- ▶ In Matlab you can solve the linear system $Ax = b$ by the backslash operator: $A \backslash b$.

Reason: computing the inverse of a matrix is numerically less stable and much more expensive (essentially, computing the inverse of a matrix is as expensive as solving n linear systems $Ax = e_i$).

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 (to the other extrem, we rather 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.

Least squares with linear combination of basis functions

Literature:

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

Using non-linear basis functions

Idea:

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

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

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

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

Examples

Example 1:

- If we want to learn a periodic function, the Φ_i might be the first couple of Fourier functions.

Examples (2)

Example 2:

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

We might consider basis functions as the following ones:

- ▶ Φ_1 counts the number of occurrences of the word “soccer”
- ▶ Φ_1 counts the number of occurrences of the word “team”
- ▶ Φ_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:

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

- ▶ 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 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 $d(d - 1)/2 = \Theta(d^2)$ polynomials of degree two, so it can easily happen that we need more basis functions than we can cope with...

Summary: Linear least squares regression

- ▶ 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.
- ▶ 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.

Ridge regression: least squares with L_2 -regularization

Literature: Hastie/Tibshirnai/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

$$w_{n,\lambda} := \operatorname{argmin}_{w \in \mathbb{R}^D} \frac{1}{n} \sum_{i=1}^n \left(Y_i - \langle w, \Phi_i(x) \rangle \right)^2 + \lambda \cdot \sum_{i=1}^D w_i^2$$

Ridge regression problem (2)

In matrix notation, this problem has the form

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

Solution

Theorem 11 (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:

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

- ▶ 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: $A = V^t D V$ where D is the diagonal matrix with all eigenvalues of A .
- A has full rank if and only if all its eigenvalues are not 0.
- Because of the special form $A := \Phi^t \Phi$, all eigenvalues are ≥ 0 (the matrix is positive semi-definite).
- σ is an eigenvalue of A with eigenvector $v \iff \sigma + \lambda$ is an eigenvalue of $A + \lambda I$. Reason:

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

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



Example (by Matthias Hein)

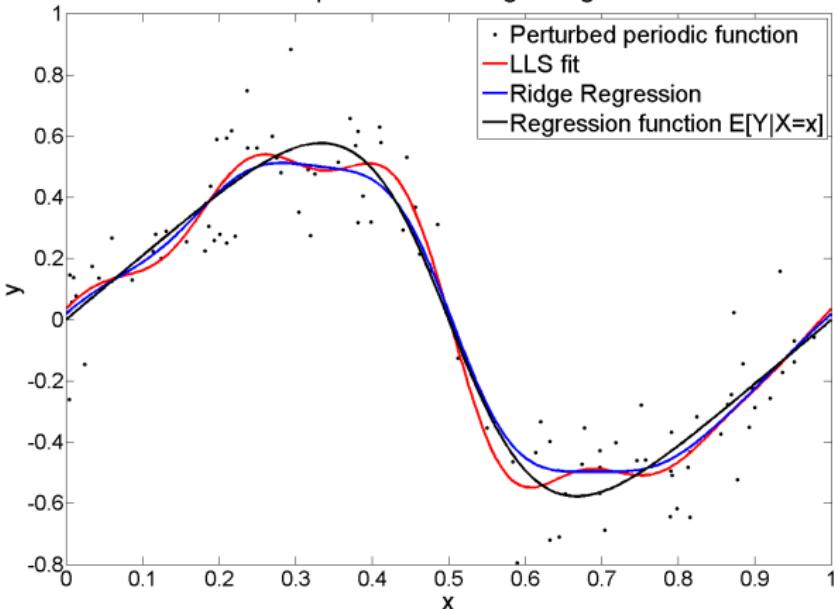
- ▶ True function: periodic function + noise
- ▶ Basis functions Φ : first 10 Fourier basis functions

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

$$f(x) = \sum_{i=1}^{10} w_i \sin(ix)$$

Example (by Matthias Hein) (2)

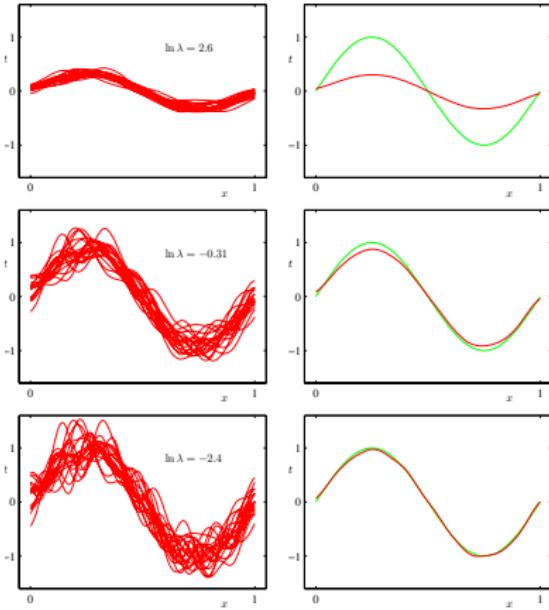
Least Squares and Ridge Regression



Example (from Bishop's book)

Left: results for decreasing amount of regularization

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



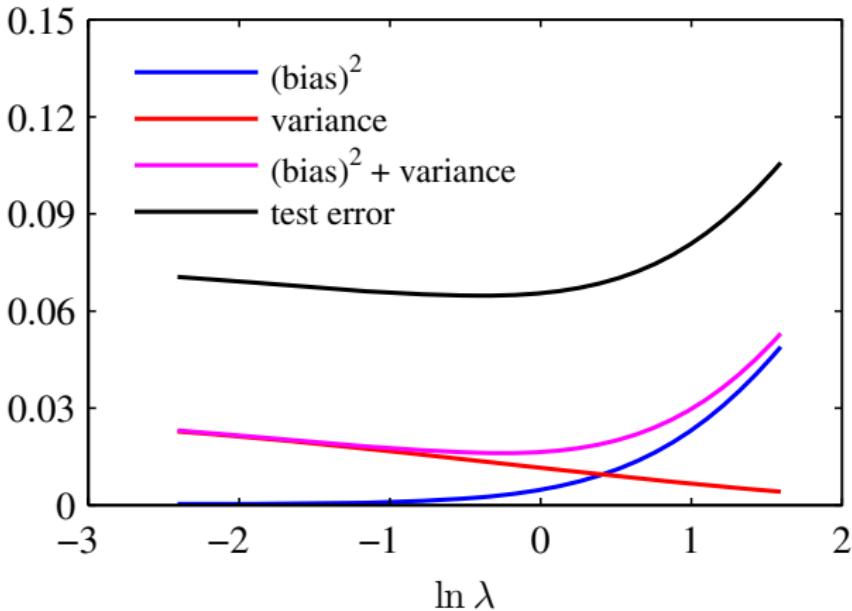
Example (from Bishop's book) (2)

QUESTION: IF THE λ INCREASES, WHAT HAPPENS TO BIAS AND VARIANCE?

Example (from Bishop's book) (3)

Same example, bias-variance decomposition:

Larger regularization constant λ leads to less complex functions:



In practice: normalization

For standard linear least squares regression: if we shift or rescale the individual coordinates of our input data, this directly translates to a shift and rescaling of the parameters w and b (EXERCISE).

However, this is not true for regularized regression!

The reason is that the rescaling and shifting also influences the regularization term in a non-linear way:

In practice: normalization (2)

To see this:

- ▶ Fix a constant α and replace all basis functions Φ_i by $\Psi_i := \alpha \cdot \Phi_i$. Then we get a different solution:
Replacing Φ_i in the formula by Ψ_i/α gives

$$w_{n,\lambda} = \alpha \left(\Psi^t \Psi + \alpha^2 n \lambda \mathbb{1}_D \right)^{-1} \Psi^t Y$$

- ▶ Similarly, if we replace each basis function Φ_i by $\Phi_i + c$ for some $c \in \mathbb{R}$, we get a different solution (EXERCISE).

In practice: normalization (3)

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

1. Centering:

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

In Practice: Choose λ by cross validation

... see the slides towards the end ...

(*) 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 σ_i large: not much difference to non-regularized case:

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

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

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

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

(*) Ridge regression as shrinkage method (3)

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

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

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

(*) Ridge regression as shrinkage method (4)

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

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

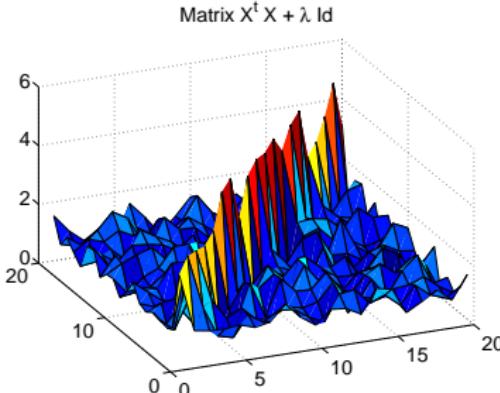
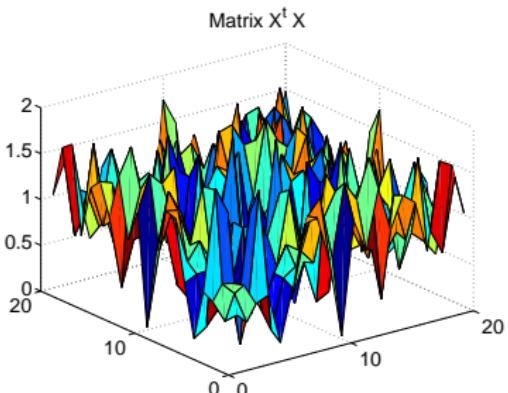
Read it on wikipedia if you are interested ☺

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).



- ▶ Note that the matrix $\Phi^t \Phi + n\lambda Id$ is always of full rank, hence it can be inverted.

History and Terminology (3)

- ▶ The regularization interpretation we described above (adding the penalty term avoids overfitting) is more recent.

Summary: Ridge regression

- ▶ 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 Φ_i .
- ▶ Regularizer: $\Omega(f) = \|w\|^2$
- ▶ 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

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

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

Sparsity

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

A naive regularizer for sparsity

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

A naive regularizer for sparsity (2)

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

Use the regularizer

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

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

However, Ω_0 is a discrete function, so it is not a good idea to try to minimize it.

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)

p-norms (2)

- For $p = 0$, we can define

$$\|w\|_0 := \lim_{p \rightarrow 0} \|w\|_p^p$$

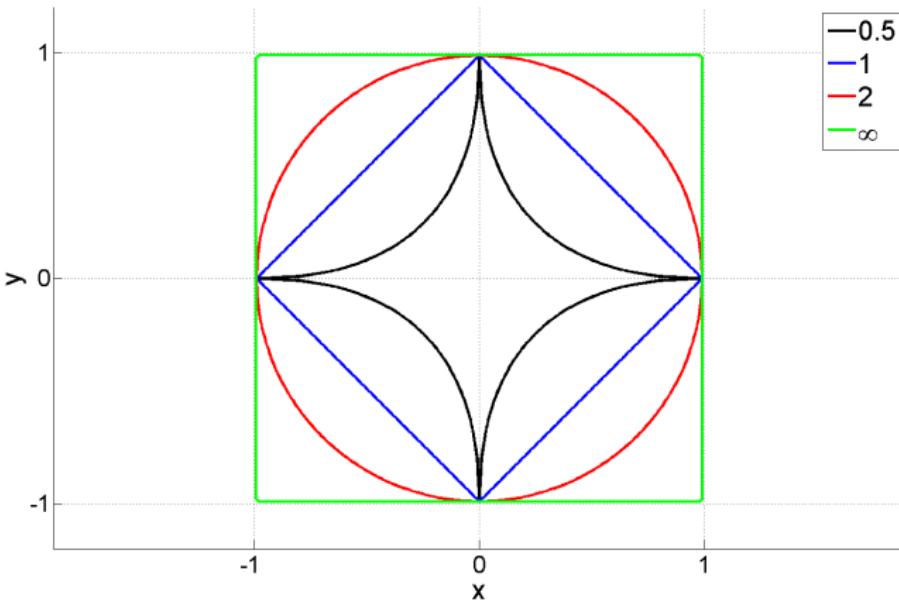
(Note that we take the limit of $\|w\|_p^p$, not of $\|w\|_p$).

This is not a norm either (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: $\Omega_0(f) = \|w\|_0$.

p -norms (3)

Level sets of the p -norms



(Image by Matthias Hein)

Sparsity and the L_1 -norm

We now want to settle for $\|w\|_1$ as a regularizer:

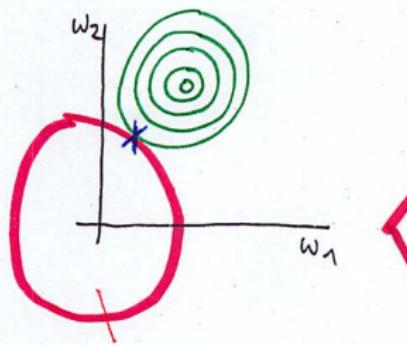
- ▶ It is “as close” to the non-convex regularizer $\|w\|_0$ as possible while still being convex

The next slide shows an illustration for the fact that the 1-norm tends to produce sparse solutions.

Sparsity and the L_1 -norm (2)

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

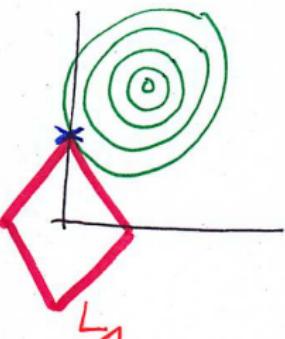
contour lines of the empirical error



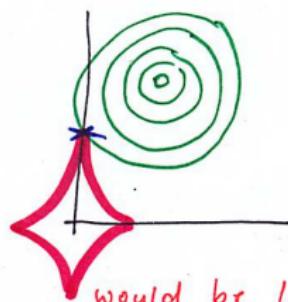
L_2 regularization:

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

$w_1, w_2 \neq 0$, not sparse



$w_1 = 0$, sparse



would be L_p for
 $p < 1$

even sparser, but not
convex any more.

Sparsity and the L_1 -norm (3)

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

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

The Lasso

Consider the following regularization problem:

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

$$w_{n,\lambda} := \operatorname{argmin}_{w \in \mathbb{R}^D} \frac{1}{n} \sum_{i=1}^n \left(Y_i - \langle w, \Phi_i(x) \rangle \right)^2 + \lambda \cdot \sum_{i=1}^D |w_i|$$

The Lasso (2)

In matrix notation, this problem has the form

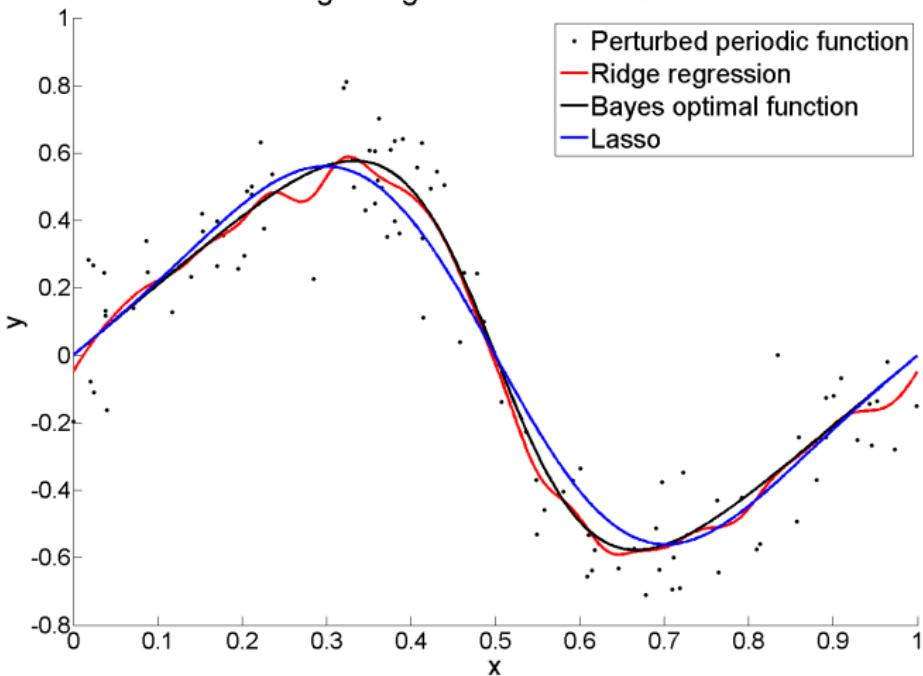
$$w_{n,\lambda} := \operatorname{argmin}_{w \in \mathbb{R}^D} \frac{1}{n} \|Y - \Phi w\|_2^2 + \lambda \|w\|_1.$$

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.

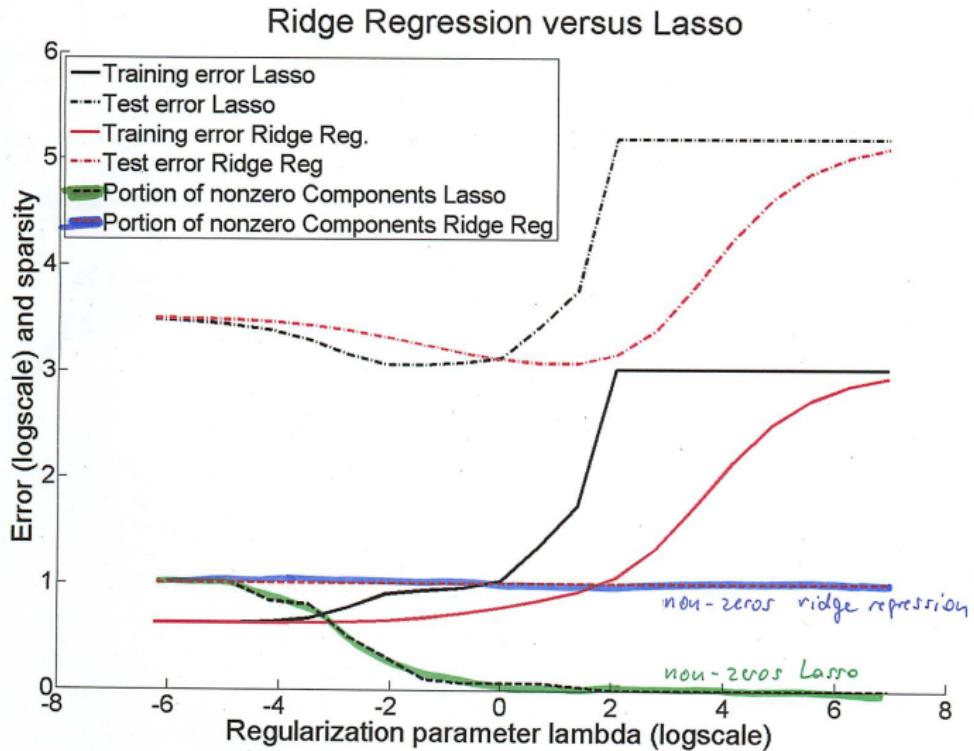
Example

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: $\Omega(f) = \|w\|_1$ to enforce sparsity.
- ▶ Convex optimization problem, no analytic solution, but efficient solvers exist.

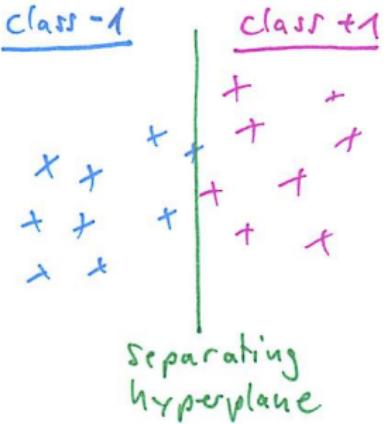
Linear Methods for classification

Intuition and feature representation

Intuition

Given:

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



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

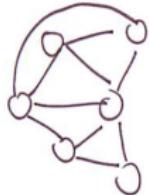
- ▶ Given a string
- ▶ Represent it by counting substrings (can also allow substrings with “gaps” in between)

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

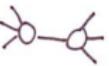
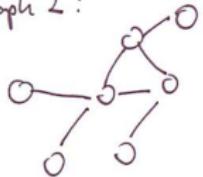
1



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 the learning algorithm as much information as possible, and hope that it is going to pick the information that is helpful for classification.

Feature representation of data (9)

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

We have seen such a mapping when we considered regression with basis functions Φ_1, \dots, Φ_D . Taken together, they build a feature map $\Phi(X) := (\Phi_1(X), \dots, \Phi_D(X))^t : \mathcal{X} \rightarrow \mathbb{R}^D$.

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

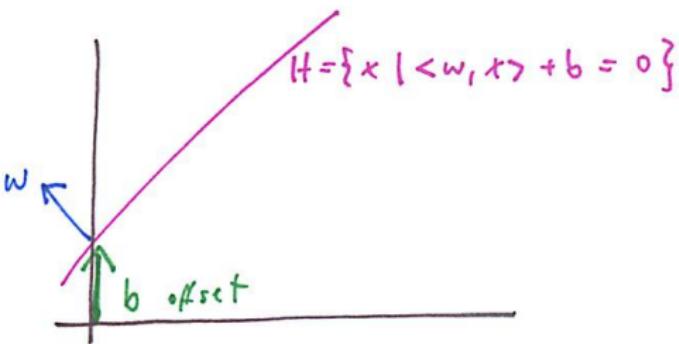
Hyperplanes in \mathbb{R}^d

Now let's come back to 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 a decision function of the form

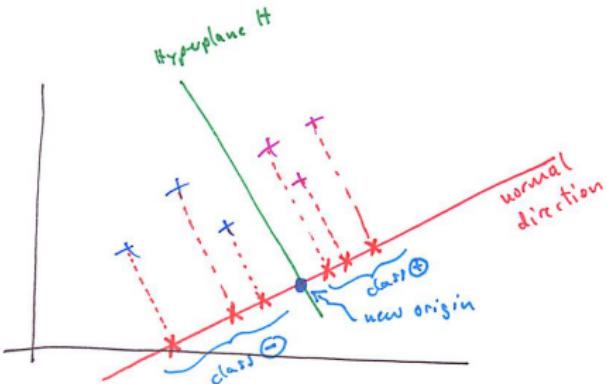
$$f(x) = \text{sign}(\langle w, x \rangle + b) \in \{\pm 1\}$$

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



Crucial question

Now of course the crucial question is: given data, how to choose the separating hyperplane???

We are now going to see three basic approaches to do this:

- ▶ Linear discriminant analysis
- ▶ Logistic regression
- ▶ Linear support vector machines

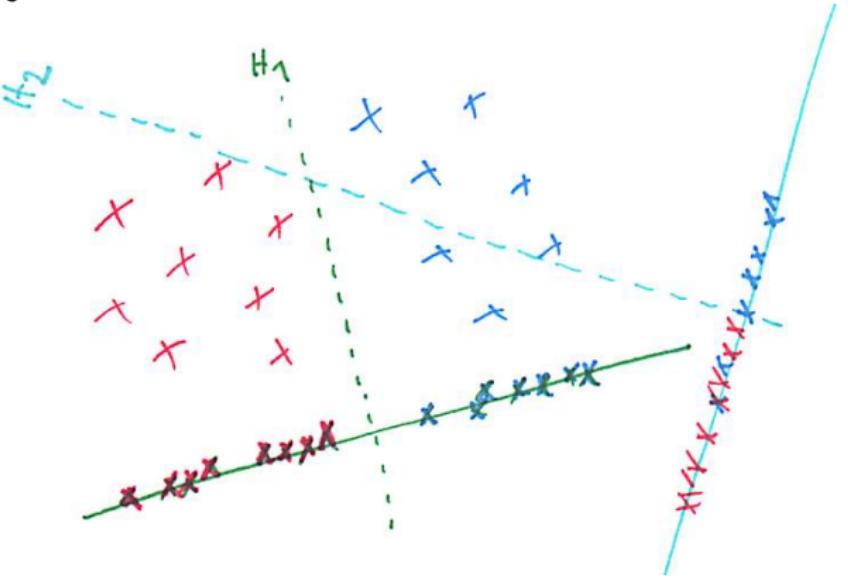
Linear discriminant analysis

Literature:

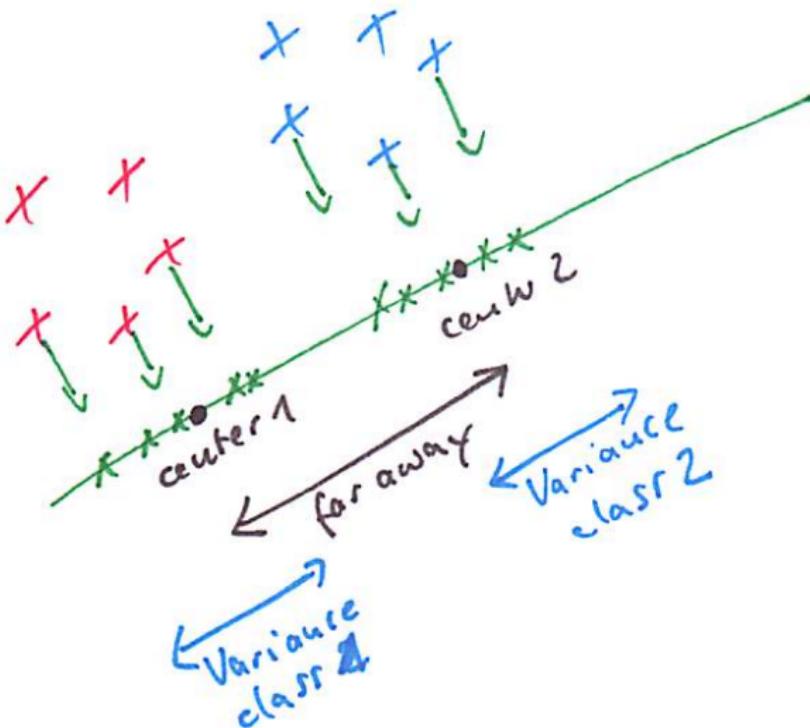
- ▶ Hastie/Tibshirani/Friedman Sec. 4.3
- ▶ Duda / Hart

LDA: Geometric motivation

Different projections: which one is better for classification?



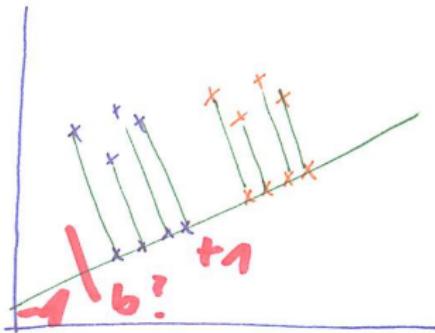
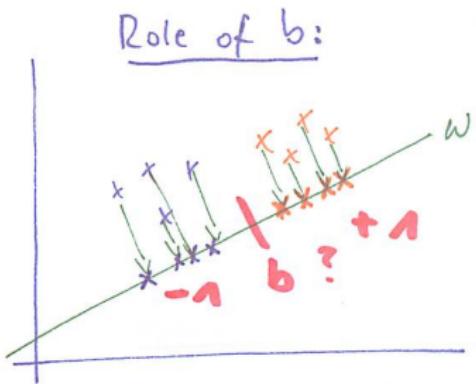
LDA: Geometric motivation (2)



LDA: Geometric motivation (3)

- ▶ Have seen: linear classification amounts to a one-dimensional projection.
- ▶ Here is now an idea to select the direction w on which we want to project. Namely, we try to achieve two things:
 - ▶ The class centers should be as far away from each other as possible
 - ▶ The class variances should be as small as possible.

LDA: Geometric motivation (4)



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 covariance matrix as

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

- ▶ Define the total within-class covariance 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 12 (Solution vector w^* of LDA)

The optimal solution of the problem $w^* := \operatorname{argmax}_{w \in \mathbb{R}^d} J(w)$ is given by

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

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: second motivation, by Bayesian decision theory

We can also derive LDA as a special case of decision theory. The basic argument is as follows:

- ▶ If we know the probability distribution P of the data and we selected a particular loss function, then Bayesian decision theory tells us what the best classifier is.
- ▶ If we make particular assumptions, then we might be able to explicitly compute the form of this optimal classifier.

Let us make the following **assumptions**:

- ▶ Assume that the two classes follow a multivariate normal distribution with the same covariance matrix and equal prior weights.
- ▶ As loss function, we use the 0-1-loss.

LDA: second motivation, by Bayesian decision theory (2)

Then we can argue as follows:

- ▶ According to Bayesian decision theory, we know that under the given assumptions (0-1-loss, equal class weights) the optimal classifier selects according to whether
$$P(Y = 1 \mid X = x) \stackrel{?}{>} P(Y = -1 \mid X = x).$$
- ▶ To decide which one is larger, we consider the log-ratio of both terms: $\log \left(P(Y = 1 \mid X = x) / P(Y = -1 \mid X = x) \right) \stackrel{?}{>} 0.$
- ▶ If we compute this term for the normal distributions, many terms vanish.
- ▶ One can then derive the fact that the decision boundary between the two classes (i.e., the set where both classes have equal posteriors) is a hyperplane, and in fact it is the same hyperplane as the one given by LDA.

LDA: second motivation, by Bayesian decision theory (3)

- Details skipped, see Hastie/Tibshirani/Friedman for details.

LDA: second motivation, by Bayesian decision theory (4)

Insights:

- ▶ Under the given assumptions, LDA will work nicely!
- ▶ We can also suspect that it does not such a good job if the assumptions are not satisfied, see below.

LDA, third 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.
- ▶ As loss function we use the squared loss between the real-valued output of the function and the actual class labels.
- ▶ No assumption on the underlying distributions!

Then we can prove the following nice theorem:

LDA, third motivation: by ERM (2)

Theorem 13 (LDA as ERM)

Consider the following problem of minimizing the least squares loss over the class of affine linear functions:

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

Then $w' \propto w^*$, that is the solutions w' and w^* coincide up to a constant.

Proof: skipped.

LDA, third motivation: by ERM (3)

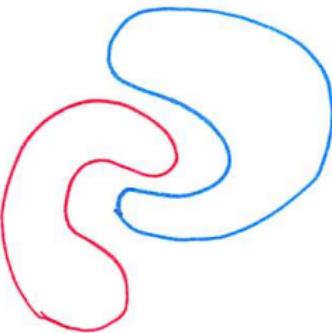
Comments:

- ▶ Important to note here: The least squares loss in this 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).
- ▶ In the theorem, the threshold b is chosen automatically as to minimize the L_2 -loss. However, the rule of minimizing the training error works better in practice.

Reason: the L_2 -error has not so much to do with the 0-1-loss (and our criterion for b tries to optimize the 0-1-loss rather than the L_2 -loss).

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.
- ▶ 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)

- ▶ Projection motivation: project in a direction that separates the classes well
- ▶ ERM motivation: minimize the least squares loss
- ▶ Model-based motivation: Bayes classifier under assumption of normal distributions with equal variances
- ▶ Leads to the Fisher criterion that should be minimized
- ▶ Can compute solution vector w analytically

Logistic regression

Literature: Hastie/Tibshirani/Friedman Section 4.4

Logistic regression, ERM viewpoint

Logistic regression problem as ERM

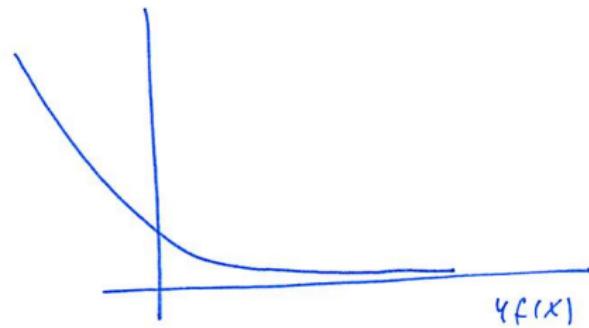
- ▶ Want to solve classification with linear functions:
 - ▶ Given $X_i \in \mathcal{X}$, $Y_i \in \{\pm 1\}$.
 - ▶ $\mathcal{F} = \{\sum_{i=1}^D w_i \Phi_i(x) \mid w \in \mathbb{R}^D\}$.
 - ▶ Use ERM
- ▶ Have seen: LDA corresponds to minimizing the L_2 -loss.
- ▶ Now: use the logistic loss function:

Logistic regression problem as ERM (2)

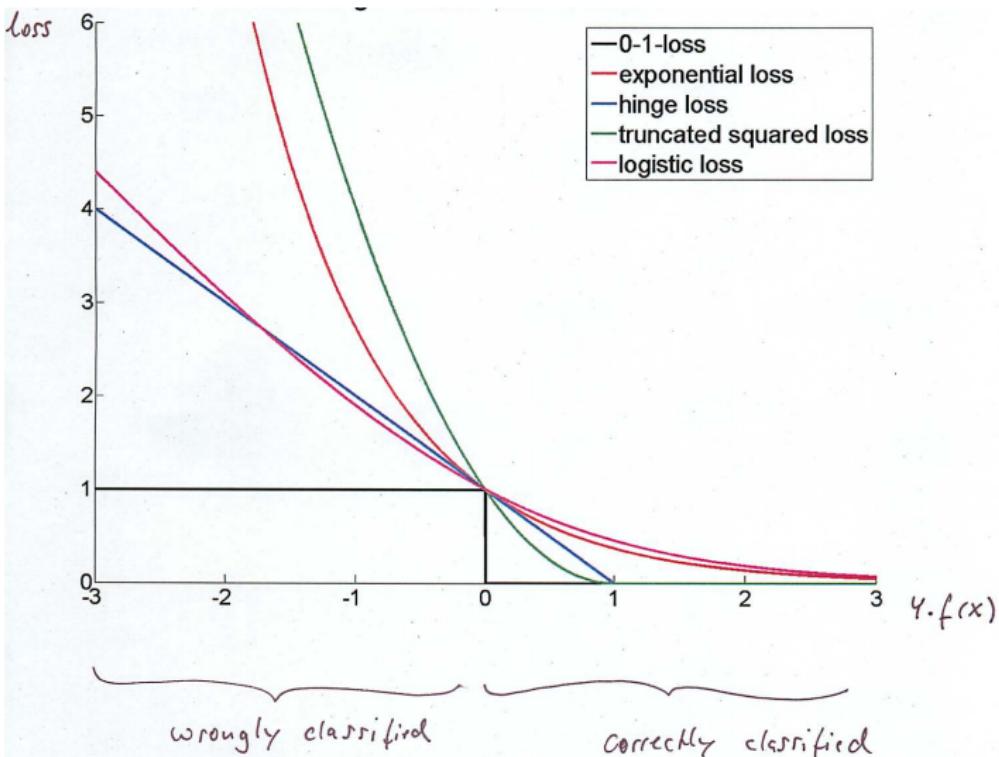
Logistic loss function:

$$\ell(X_i, f(X_i), Y_i) = \sum_{i=1}^n \log(1 + \exp(-Y_i f(X_i)))$$

with $f(X_i) = \langle w, X_i \rangle + b$.



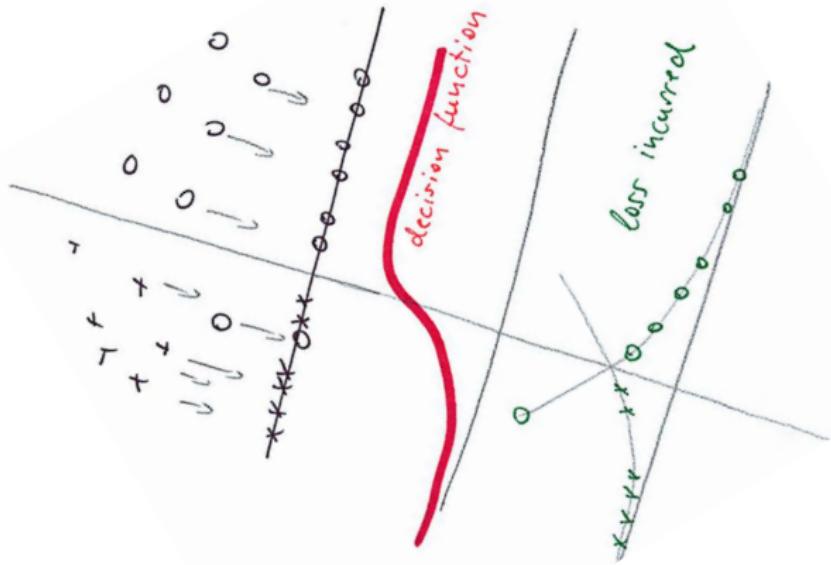
Logistic regression problem as ERM (3)



Logistic regression problem as ERM (4)

We will see in a couple of minutes that logistic regression does the following:

Logistic regression problem as ERM (5)



Logistic regression, Bayesian (model-based) viewpoint

Excursion: Probabilistic interpretation of ERM

- ▶ Bayesian decision theory: 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 $P(f(X_i) = Y_i \mid X = X_i)$ as large as possible.
- ▶ This amounts to selecting $f \in \mathcal{F}$ by

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

Excursion: Probabilistic interpretation of ERM

(2)

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

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

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

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

Excursion: Probabilistic interpretation of ERM

(3)

What does this tell us?

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

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

Note that it does not always work the other way round (we cannot find a probability distribution for any kind of loss function ℓ).

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)

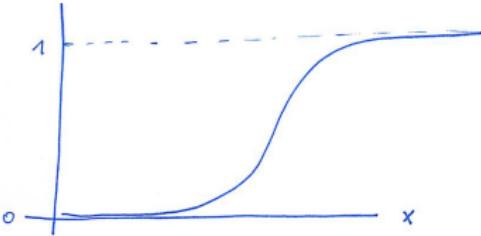
- Given a particular model, writing down the loss function helps to understand the behavior of the classifier: What are the errors that are punished most? So what does the classifier try to avoid at all costs?

The logistic model

- ▶ Assume that we have the following probabilistic model:

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

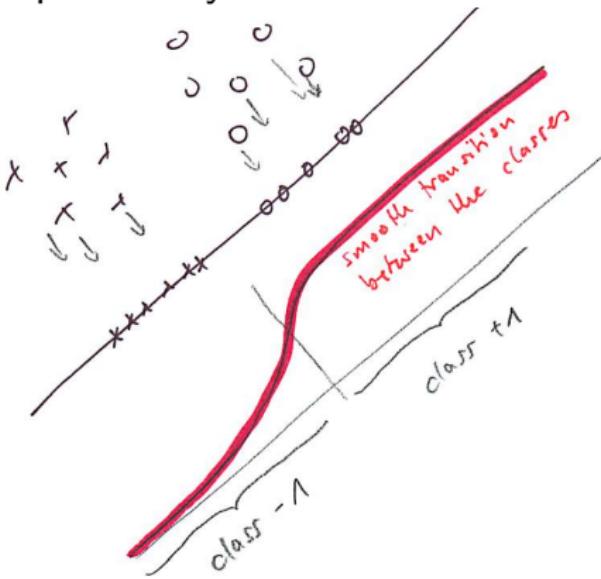
with $f(x) = \langle w, x \rangle + b$. Here w and b are the parameters. The function $1/(1 + \exp(-t))$ is called the **logistic function** and looks as follows:



The logistic model (2)

- ▶ 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 (3)

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 (4)

- With this model, the “log odds” have the following linear form:

$$\log \left(\frac{P(Y = +1 \mid X = x)}{P(Y = -1 \mid X = x)} \right) = \langle w, \Phi(x) \rangle + b =: f(x)$$

- By the argument on the previous slides, this corresponds to minimizing the following loss function:

$$\ell(X_i, f(X_i), Y_i) = \sum_{i=1}^n \log(1 + \exp(-Y_i f(X_i)))$$

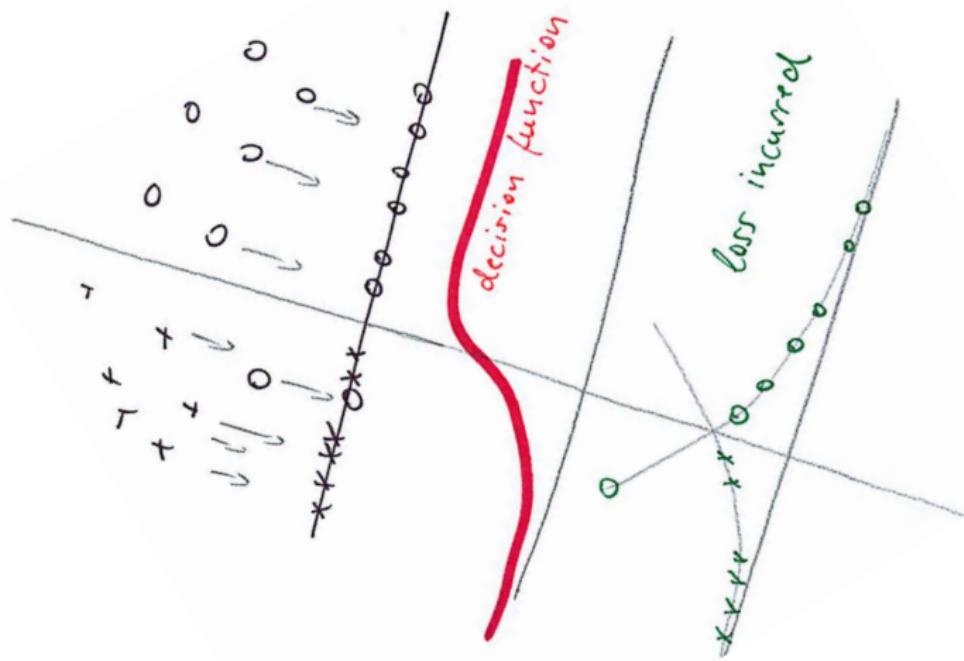
This is the logistic loss function we have seen before.

The logistic model (5)

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



Computing the solution

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

- ▶ Bad news: 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.

Regularized logistic regression

Adding regularization

- ▶ As in linear regression, we now might want to use a regularizer to avoid overfitting.
- ▶ Again we use $\Omega(f) = \|w\|_2^2$ (as in ridge regression) or $\Omega(f) = \|w\|_1$ (as in Lasso).
- ▶ Then the L_2 -regularized logistic regression is the problem to minimize

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

- ▶ This is now again a convex optimization problem in w and 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, eg with L_1 - or L_2 -regularizer
- ▶ Convex optimization problem, no closed form solution.

Linear Support vector machines (Intuition and primal)

Literature:

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

Prelude

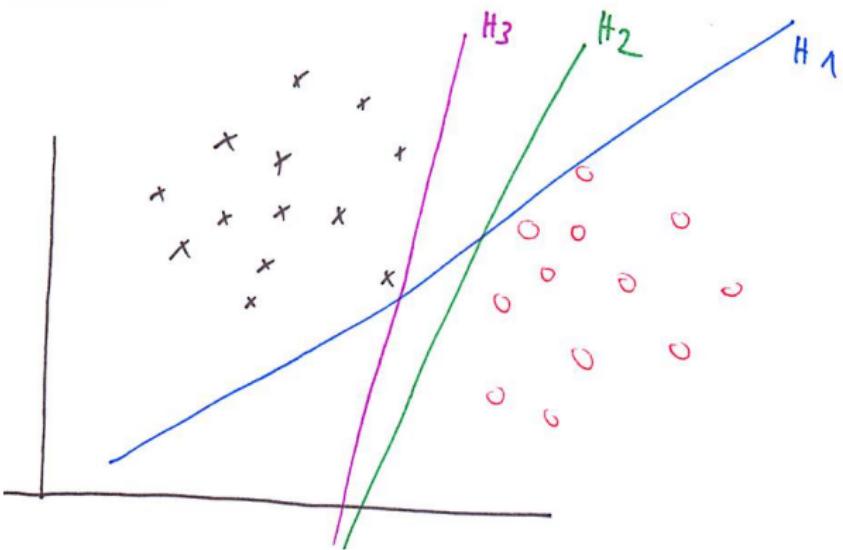
The support vector machine (SVM) is the algorithm that made machine learning its own sub-discipline of computer science, it is the most important machine learning algorithm. It has been published in the late 1990ies (see later for more on history).

It is about the most successful, powerful, and yet easy to understand classification algorithm that exists.

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 \leadsto next week.

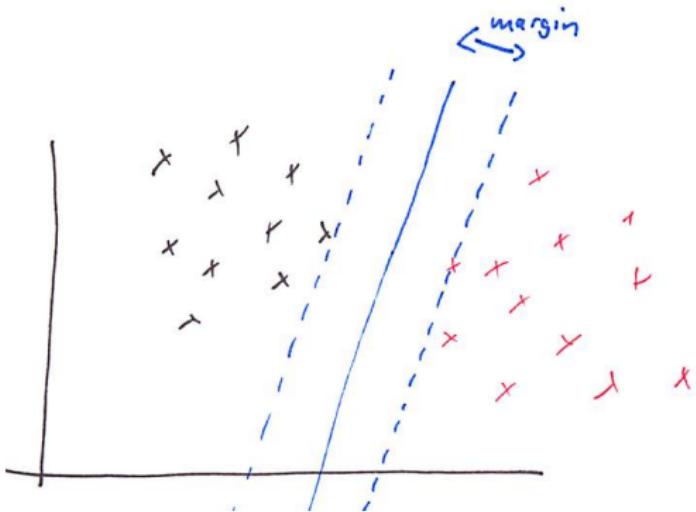
Geometric motivation

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:

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

- We now want to get rid of this degree of freedom and fix the scaling constant.

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

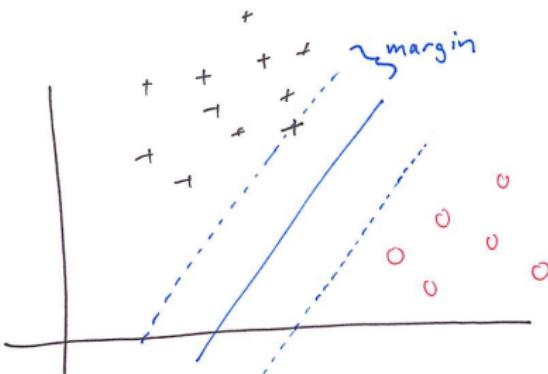
$$\min_{i=1, \dots, n} |\langle w, x_i \rangle + b| = 1$$

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 14 (Margin)

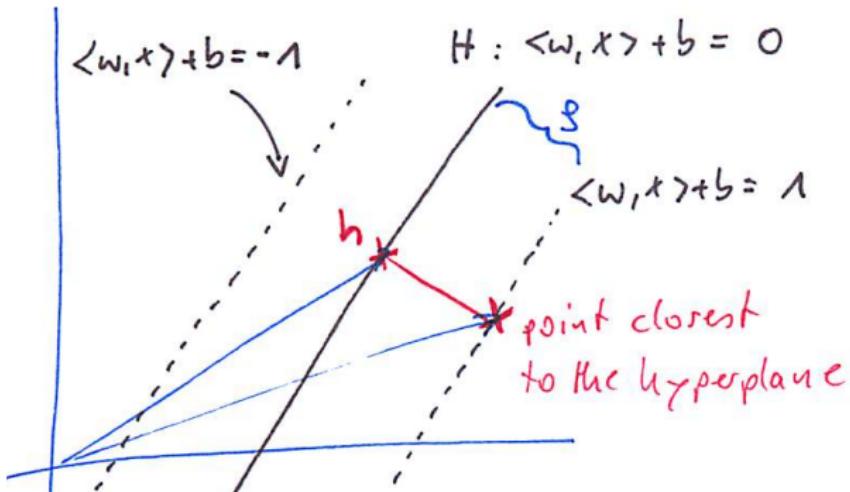
For a hyperplane in canonical representation, the margin ρ can be computed by $\rho = 1/\|w\|$.

First proof.

Observe:

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

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

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

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

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



The Margin (5)

Alternative proof:

By definition, the margin is $\rho = \|X_i - h\|$. 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:

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

Usually, we consider the following equivalent optimization problem:

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

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

Hard margin SVM (2)

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. WHY EXACTLY?

Hard margin SVM (3)

However, big disadvantage:

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

Soft margin SVM

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

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

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

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

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

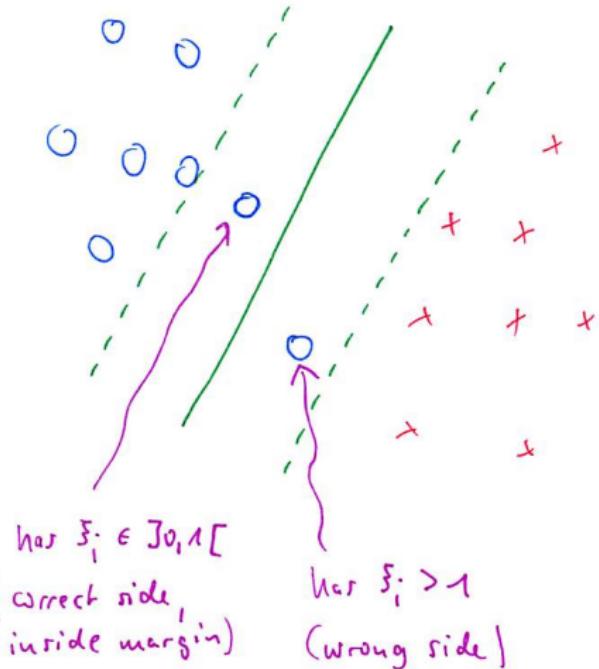
- ▶ Note that this is a convex (quadratic) problem as well.

Soft margin SVM (2)

Interpretation:

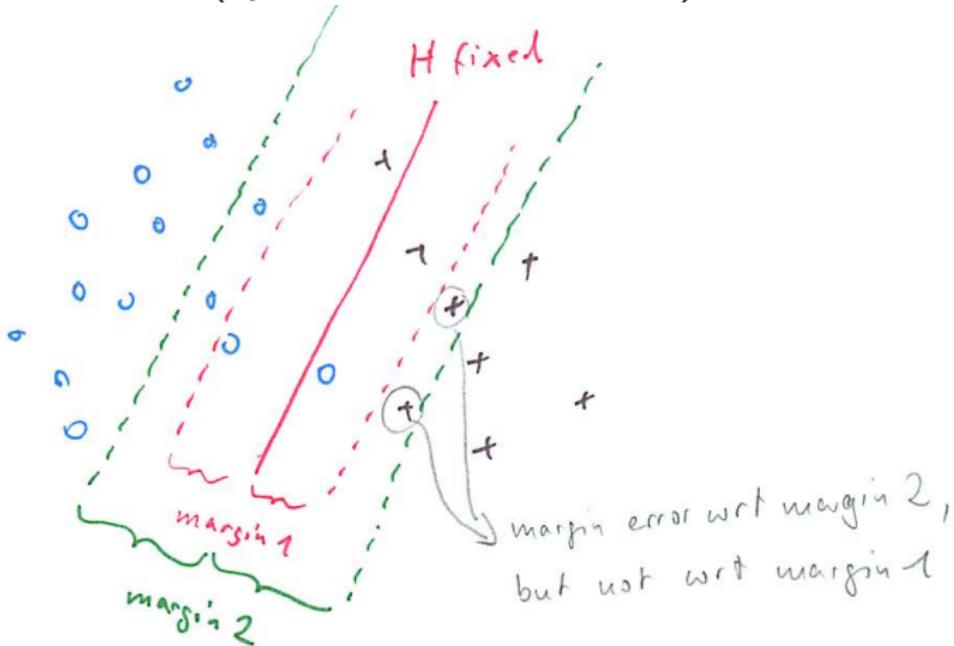
- ▶ If $\xi_i = 0$, then the point X_i is on the correct side of the hyperplane, outside the margin.
- ▶ If $\xi_i \in]0, 1[$, then X_i is still on the correct side of the hyperplane, but inside the margin.
- ▶ If $\xi_i > 1$, then X_i is on the wrong side of the hyperplane.

Soft margin SVM (3)



Soft margin SVM (4)

In particular, we can tradeoff a large margin versus some margin or classification errors (by different choices of C):



SVM as regularized risk minimization

We want to interpret the SVM in the regularization framework:

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

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

SVM as regularized risk minimization (2)

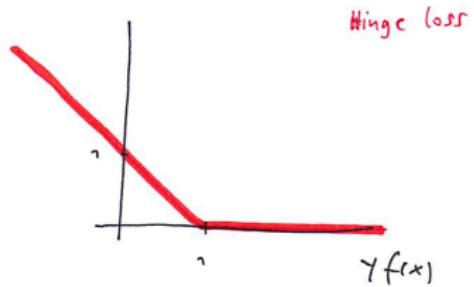
Consider the constraint $Y_i(\langle w, X_i \rangle + b) \geq 1 - \xi_i$. Exploiting $\xi_i \geq 0$ we can rewrite it as follows:

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

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

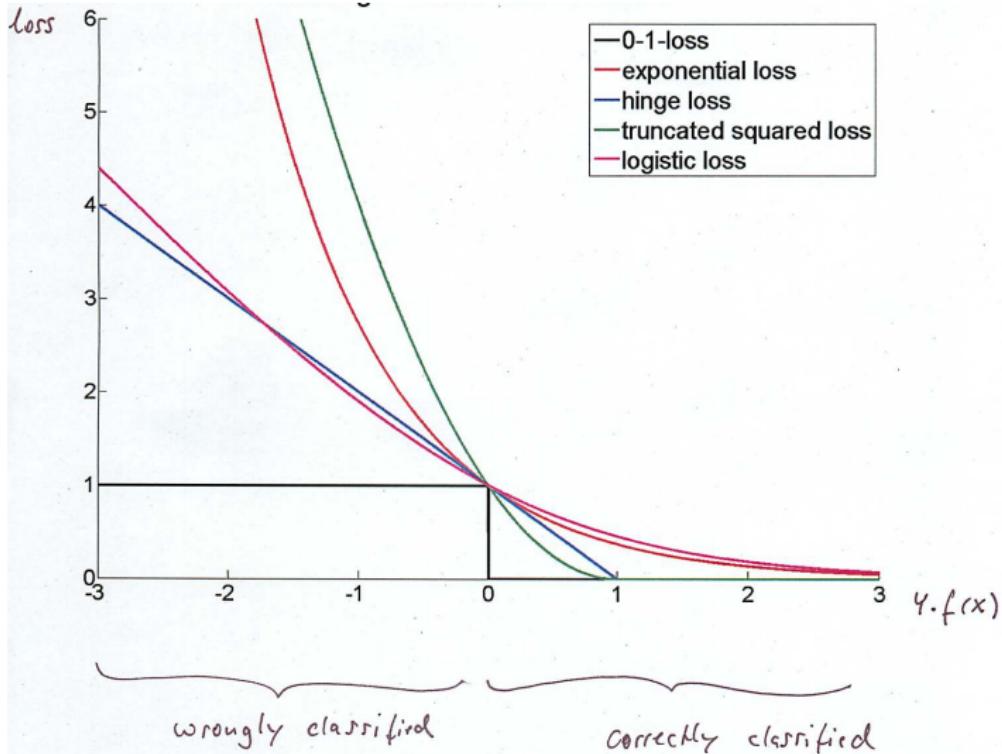
$$\ell(x, y, f(x)) = \max\{0, 1 - yf(x)\}$$

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:

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

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

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

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 to 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. A book that is huge but good to read:
 - ▶ Boyd, S. and L. Vandenberghe. Convex Optimization. Cambridge University Press, 2004.

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.

Lagrange multiplier for equality constraints

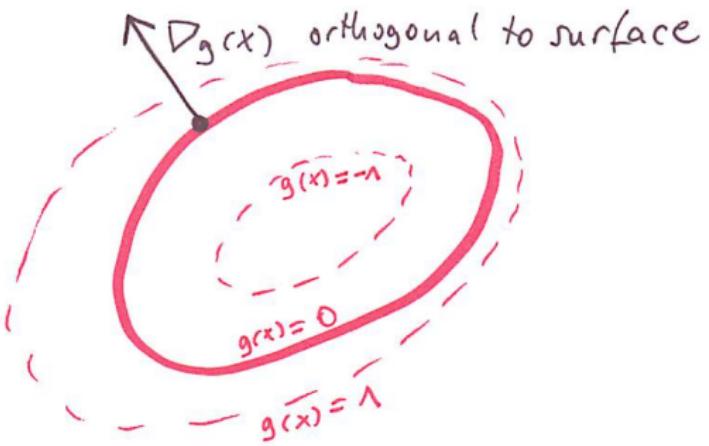
Consider the following convex optimization problem:

$$\begin{aligned} & \text{minimize } f(x) \\ & \text{subject to } g(x) = 0 \end{aligned}$$

where f and g are convex.

Lagrange multiplier for equality constraints (2)

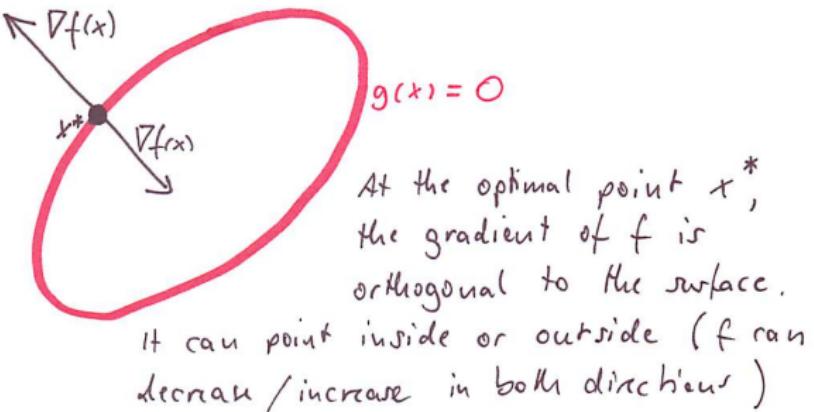
For any point x on the “surface” $\{g(x) = 0\}$ the gradient $\nabla g(x)$ is orthogonal to the surface itself.



Intuition: to increase / decrease $g(x)$, you need to move away from the surface, not walk along the surface.

Lagrange multiplier for equality constraints (3)

Consider the point x^* on the surface such that $f(x)$ is maximized. This point must have the property that $\nabla f(x)$ is orthogonal to the surface.



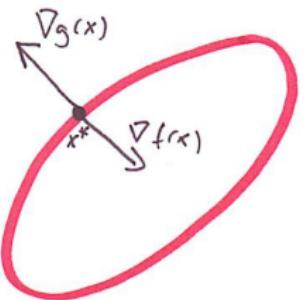
At the optimal point x^* ,
the gradient of f is
orthogonal to the surface.

It can point inside or outside (f can
decrease/increase in both directions)

Intuition: otherwise we could move a little along the surface to increase $f(x)$.

Lagrange multiplier for equality constraints (4)

So at the optimal point, $\nabla g(x)$ and $\nabla f(x)$ are parallel, that is there exists some $\lambda \in \mathbb{R}$ such that $\nabla f(x) + \lambda \nabla g(x) = 0$.



At the optimal point x^* ,
 ∇g and ∇f are parallel to
each other (and can point
either in the same or the
opposite direction).

Lagrange multiplier for equality constraints (5)

We now define the **Lagrangian** function

$$L(x, \lambda) = f(x) + \lambda g(x)$$

Now observe:

- ▶ The condition $\nabla f(x) + \lambda \nabla g(x) = 0$ is equivalent to
 $\nabla_x L(x, \lambda) = 0$
- ▶ The condition $g(x) = 0$ is equivalent to $\nabla_\lambda L(x, \lambda) = 0$.

So to find an optimal point x^* we need to find a saddle point of $L(x, \lambda)$, that is a point such that both $\nabla_x L(x, \lambda)$ and $\nabla_\lambda L(x, \lambda)$ vanish.

Simple example

Consider the problem to maximize $f(x)$ subject to $g(x) = 0$ with

$$f(x_1, x_2) = 1 - x_1^2 - x_2^2$$

$$g(x_1, x_2) = x_1 + x_2 - 1$$

Observe: it is hard to solve this problem by naive methods because it is unclear how to take care of the constraints!

Solution by the Lagrange approach:

Bring it in the standard form:

$$\text{minimize } x_1^2 + x_2^2 - 1$$

$$\text{subject to } x_1 + x_2 - 1 = 0$$

Simple example (2)

The Lagrangian is

$$L(x, \lambda) = \underbrace{x_1^2 + x_2^2 - 1}_{f(x_1, x_2)} + \lambda \underbrace{(x_1 + x_2 - 1)}_{g(x_1, x_2)}$$

Now compute the derivatives and set them to 0:

$$\nabla_{x_1} L = 2x_1 + \lambda \stackrel{!}{=} 0$$

$$\nabla_{x_2} L = 2x_2 + \lambda \stackrel{!}{=} 0$$

$$\nabla_\lambda L = x_1 + x_2 - 1 \stackrel{!}{=} 0$$

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)

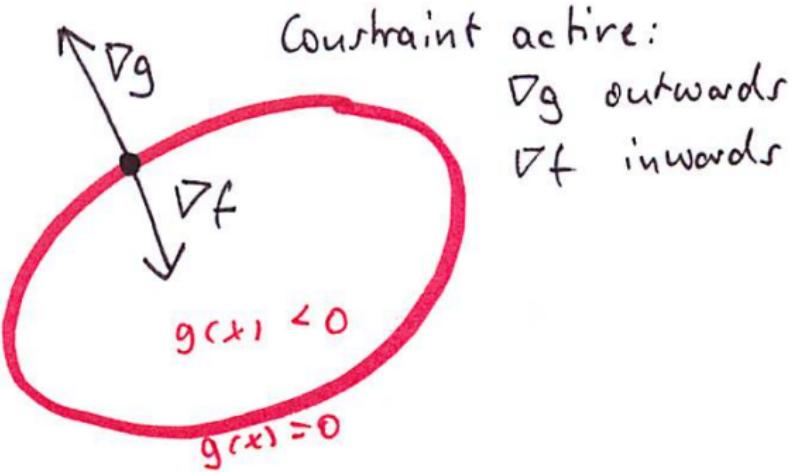
Case 1: Constraint is “active”, that is the optimal point is *on* the surface $g(x) = 0$.

Again ∇f and ∇g are parallel in the optimal point.

But furthermore, the direction of derivatives matters:

- ▶ the derivative of g points outwards (at any point on the surface $g = 0$, because g increases outwards).
- ▶ the derivative of f is directed inwards (otherwise we could decrease the objective by walking inside).

Lagrange multiplier for inequality constraints (3)



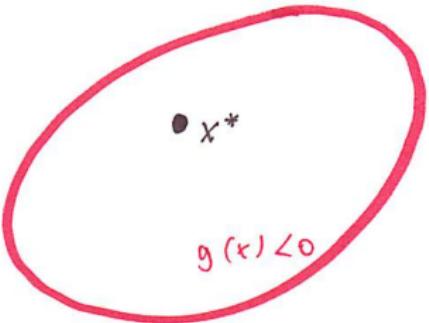
So we have $\nabla f(x) = -\lambda \nabla g(x)$ for some value $\lambda > 0$.

Lagrange multiplier for inequality constraints (4)

Case 2: Constraint is “inactive”, that is the optimal point is not on the surface $g(x) = 0$ but somewhere in the interior.

- ▶ Then we have $\nabla f = 0$ at the solution (otherwise we could decrease the objective value).
- ▶ We do not have any condition on ∇g (it is as if we would not have this constraint).

Constraint inactive. No condition on ∇g



Lagrange multiplier for inequality constraints (5)

We can summarize both cases using the Lagrangian again:

- ▶ 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.

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:

$$L(x, y, \alpha) = -xy + \alpha(2x + 2y - 1)$$

Simple example (2)

Saddle point conditions / derivatives:

$$\partial L / \partial x = -y + 2\alpha \stackrel{!}{=} 0$$

$$\partial L / \partial y = -x + 2\alpha \stackrel{!}{=} 0$$

$$\partial L / \partial \alpha = 2x + 2y - 1 \stackrel{!}{=} 0$$

Solving this system of three equations gives $x = y = 0.25$.

Lagrangian: formal point of view

Lagranigan 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 15 (Dual function is concave)

No matter whether the primal problem is convex or not, the dual problem is always concave in (λ, ν) .

Proof. For fixed x , $L(x, \lambda, \nu)$ is linear in λ and ν and thus convex. The dual function as a pointwise infimum over concave functions is concave as well. ☺

Dual function as lower bound on primal (2)

Proposition 16 (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

Just seen: the dual function provides a lower bound on the primal value. Finding the highest such lower bound is the task of the dual problem:

We define the **dual optimization problem** as

$$\max_{\lambda, \nu} g(\lambda, \nu) \text{ subject to } \lambda_i \geq 0, \nu_j \in \mathbb{R}$$

Denote the solution of this problem by λ^*, ν^* and the corresponding objective value $d^* := g(\lambda^*, \nu^*)$.

Weak duality

Proposition 17 (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 16 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 convex problems that do not satisfy strong duality.
Here is an example:

$$\begin{aligned} & \text{minimize}_{x,y} \exp(-x) \\ & \text{subject to } x_2/y \leq 0 \\ & \quad y \geq 0 \end{aligned}$$

Here one can check that $p^* = 1$ 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 that lead to this solution, if we just know the dual variables that give this solution?

EXERCISE! XXX

Strong duality implies saddle point

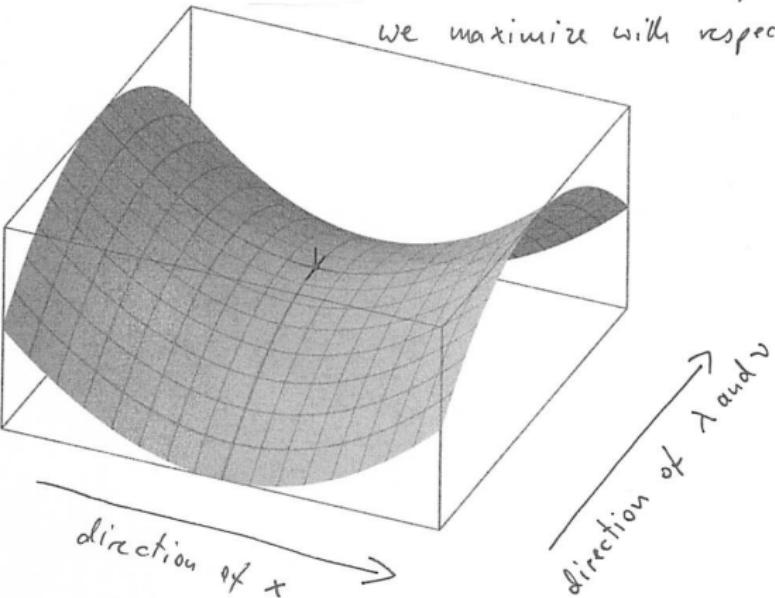
Proposition 18 (Strong duality implies saddle point)

Assume strong duality holds, let x^* the solution of the primal and (λ^*, ν^*) the solution of the dual optimization problem. Then (x^*, λ^*, ν^*) is a saddle point of the Lagrangian.

Strong duality implies saddle point (2)

Lagrangian saddlepoint

we minimize with respect to x ,
we maximize with respect to λ, γ



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

$$f(x^*) = g(\lambda^*, \nu^*) = \inf_x L(x, \lambda^*, \nu^*) \leq L(x^*, \lambda^*, \nu^*) \leq f(x^*)$$

(We had just seen the last inequality as a consequence of x^* being feasible).

- ▶ Because we have the same term on the left and side, we in fact have equality everywhere.
- ▶ So in particular, $\inf_x L(x, \lambda^*, \nu^*) = L(x^*, \lambda^*, \nu^*)$.
- ▶ Then we have to show that (λ^*, ν^*) are maximizers of $L(x^*, \lambda, \nu)$.
 - ▶ This follows from the definition of (λ^*, ν^*) as solutions of $\max_{\lambda, \nu} \min_x L(x, \lambda, \nu)$.

Strong duality implies saddle point (4)

- Taken together we get

$$L(x^*, \lambda, \nu) \leq L(x^*, \lambda^*, \nu^*) \leq L(x, \lambda^*, \nu^*)$$

That is, (x^*, λ^*, ν^*) is a **saddle point** of the Lagrangian:

- It is a minimum for x (with fixed λ^*, ν^*).
- It is a maximum for (λ, ν) (with fixed x^*).



Saddle point always implies primal solution

Proposition 19 (Saddlepoint implies primal solution)

If (x^*, λ^*, ν^*) is a **saddle point** of the Lagrangian, then x^* is always a solution of the primal problem.

Proof. Not very difficult, but we skip it. ☺

Remarks:

- ▶ This proposition always holds (not only under strong duality).
- ▶ this proposition gives sufficient conditions for optimality.
Under additional assumptions (constraint qualifications) it is also a necessary condition.

Why is this whole approach useful?

- ▶ Whenever we have a saddle point of the Lagrangian, we have a solution of our constraint optimization problem. This is great, because otherwise we would not know how to do 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 problem is not convex, at least its dual is convex. 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.

Linear Support vector machines (Dual and Properties)

Literature:

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

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:

$$\underset{\alpha}{\text{maximize}} \quad g(\alpha)$$

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

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

Dual of hard margin SVM (4)

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

In particular,

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

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

Dual of hard margin SVM (5)

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

- ▶ First exploit (*):

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

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

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

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

Dual of hard margin SVM (6)

Dual function:

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

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

(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 α_i .
- ▶ To compute the class label of a test point X , we first need to understand how we can recover the primal variables w and b from the dual variables α . This works as follows.

Dual SVM in practice (2)

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

- ▶ To compute w : directly use (**): $w = \sum_i \alpha_i Y_i X_i$
- ▶ To compute b : Consider j with $\alpha_j > 0$. By the KKT conditions we know that $\alpha_j > 0$ implies that the corresponding constraint is active, that is $Y_j(\langle w, X_j \rangle + b) = 1$. Solving this for b and replacing w by the formula above 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 :

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

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 α_i .

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 next lecture).

Support vectors

Support vector property:

- ▶ KKT conditions tell us: Only Lagrange multipliers α_i that are non-zero at the saddle point correspond to active constraints (the ones that are precisely met). Formally,

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

- ▶ In our context: Only those α_i are non-zero that correspond to points that lie exactly on the margin, inside the margin or on the wrong side of the hyperplane. The corresponding points are called **support vectors**.
- ▶ So the solution can be expressed just by the coefficients of the support vectors.

Support vectors (2)

- ▶ 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 any point sits 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.

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.

History

- ▶ **Vladimir Vapnik** is the “father” of the SVM (and, in fact, the father 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:

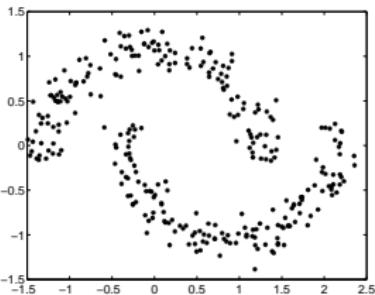
- ▶ Steinwart, Christmann: Support Vector Machines. Springer, 2006.

Intuition

Linear methods — disadvantages

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

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

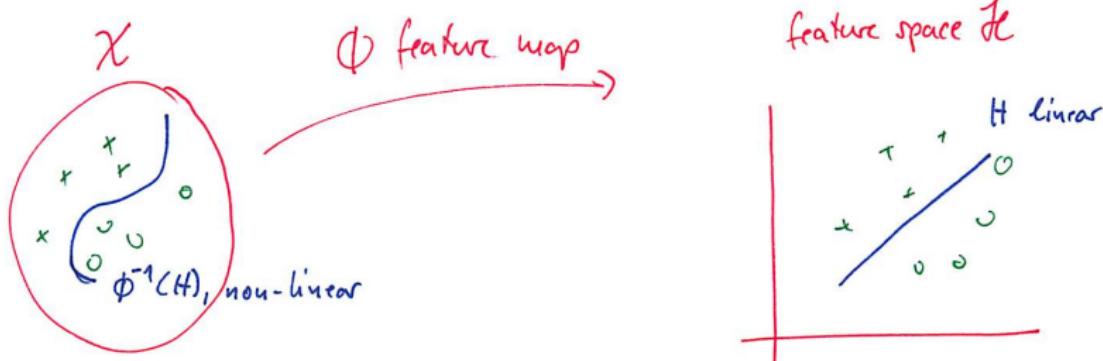


Linear methods — disadvantages (2)

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

Linear methods — disadvantages (3)

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



Starting point: Key observation for SVMs

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

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

Starting point: Key observation for SVMs (2)

Let us be more explicit:

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

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

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

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

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

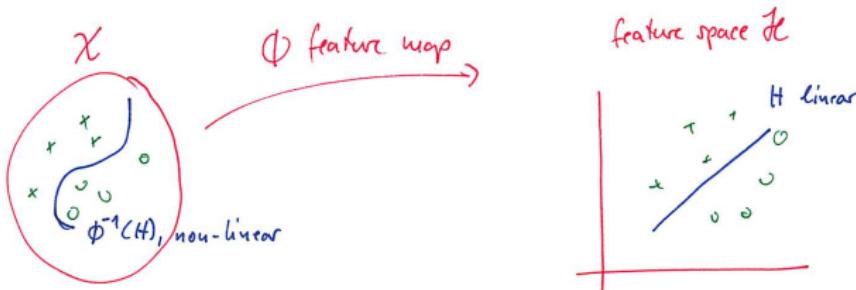
(and the same goes for the function that evaluates the results on test points).

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 Φ
- ▶ In that space, we use a linear method like an SVM
- ▶ Ideally, we never compute the embedding directly.
- ▶ Instead we want to use a “kernel function” to compute

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

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

Kernel methods — the overall picture (2)

The remainder of this lecture now tries 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 Φ the $n \times d$ -matrix that contains the data points $\Phi(X_i)$ as rows.
- ▶ Then the matrix $\Phi \cdot \Phi^t \in \mathbb{R}^{n \times n}$ coincides with the corresponding kernel matrix K with entries $k_{ij} = \langle \Phi(X_i), \Phi(X_j) \rangle = \Phi(X_i)\Phi(X_j)^t$.

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.**

Kernels: First examples

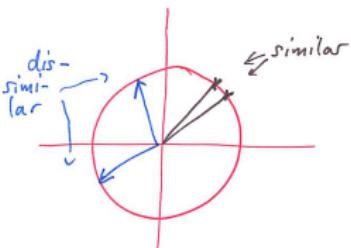
The linear kernel. the trivial kernel on \mathbb{R}^d defined by the standard scalar product:

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

Kernels: First examples (2)

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

Kernels: First examples (3)

- If the data points are normalized, then the cosine can be computed by the scalar product $\langle x, y \rangle$.

Kernels: First examples (4)

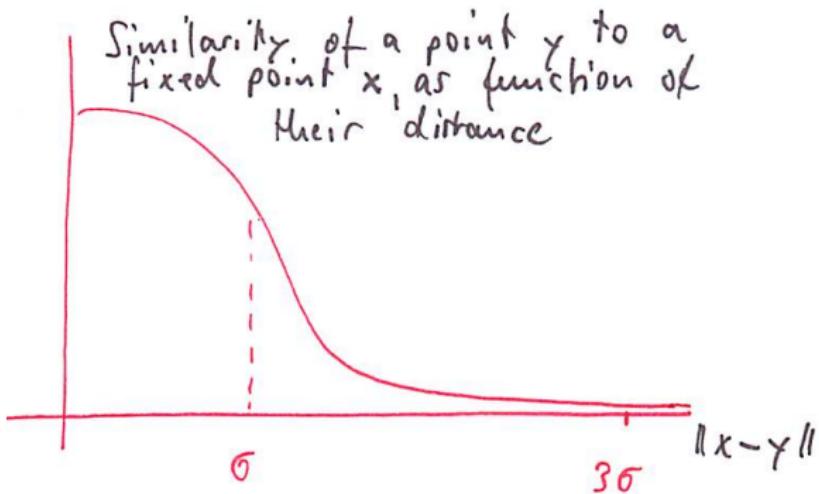
The Gaussian kernel (sometimes called **rbf-kernel** for “radial basis function”):

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

where $\sigma > 0$ is a parameter.

Induced “similarity”: Points are considered “very similar” if they are at distance at most σ , somewhat similar if they are at distance (roughly) at most 3σ , and pretty dissimilar if they are further away than that.

Kernels: First examples (5)



Kernels: First examples (6)

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

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

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

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

Kernels: First examples (7)

Kernel based on feature vectors

- ▶ 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 scalar product 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 ...

Kernels: First examples (8)

String kernels. XXXX

Kernels: First examples (9)

Path-based graph kernels.

- ▶ Consider a directed graph with non-negative edge weights. For a directed path $\pi = v_1, \dots, v_k$ define the weight of the path as $w(\pi) = \pi_{j=1}^{k-1} w(v_j, v_{j+1})$
- ▶ For each pair of vertices v, \tilde{v} consider the set Π_k which consists of all paths from v to \tilde{v} of lengths at most k
- ▶ Now define the kernel function

$$k(v_i, v_j) = \sum_{\pi \in \Pi_k} w(\pi)$$

- ▶ Note that this cannot be interpreted in terms of a 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 very 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, combine, ... 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:

Proposition 20 (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 .

WHICH DIRECTION IS EASY, WHICH ONE IS DIFFICULT?

Kernels do what they are supposed to do (2)

Proof of “ \Leftarrow ”

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

Kernels do what they are supposed to do (3)

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 $\Phi : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ such that

$$k(x, y) = \langle \Phi(x), \Phi(y) \rangle_{\mathcal{H}}$$

for all $x, y \in \mathcal{X}$.

We now introduce the Reproducing Kernel Hilbert Space (RKHS), a scalar product on this space and a corresponding feature mapping Φ . 

Reproducing kernel Hilbert space

As vector space we are going to use a space of functions!

- ▶ Consider a mapping $\Phi : \mathcal{X} \rightarrow \mathbb{R}^{\mathcal{X}}$ (to the space of all real-valued functions on \mathcal{X}), defined as

$$x \mapsto k_x := k(x, \cdot)$$

That is, the point $x \in \mathcal{X}$ is mapped to the function $k_x : \mathcal{X} \rightarrow \mathbb{R}$, $k_x(y) = k(x, y)$.

- ▶ Now consider 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 (2)

- ▶ Define a scalar product on \mathcal{G} as follows:

- ▶ For $g = \sum_i \alpha_i k(x_i, \cdot)$ and $f = \sum_j \beta_j k(y_j, \cdot)$ set

$$\langle f, g \rangle_{\mathcal{G}} := \sum_{i,j} \alpha_i \beta_j k(x_i, y_j)$$

- ▶ Check that this is well-defined (not obvious because there might be several different linear combinations for the same functions).
- ▶ Check that it is indeed a scalar product (crucial ingredient is the fact that k is positive definite.)

Reproducing kernel Hilbert space (3)

- ▶ 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 (4)

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 (5)

For those who know a bit of functional analysis:

- ▶ Let \mathcal{H} be a Hilbert space of function 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 subspace (but this subspace is still pretty large).
- ▶ Later we discuss how to avoid overfitting!

The representer theorem (2)

Theorem 21 (Representer theorem)

Let $\Omega : [0, \infty[\rightarrow \mathbb{R}$ be a strictly monotonically increasing function, L an arbitrary loss function, $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ a kernel on the input space \mathcal{X} and \mathcal{H} the corresponding feature space (RKHS) with norm $\|\cdot\|_{\mathcal{H}}$. Consider a regularized risk minimization problem of the form

$$\underset{f \in \mathcal{H}}{\text{minimize}} R_n(f) + \lambda \Omega(\|f\|_{\mathcal{H}})$$

Then there always exists an optimal solution of the form

$$f^*(x) = \sum_i \alpha_i k(X_i, x).$$

The representer theorem (3)

Proof intuition.

- ▶ Split the space \mathcal{H} into the subspace \mathcal{H}_{data} that is spanned by the data points, and its orthogonal complement \mathcal{H}_{comp} .
- ▶ Similarly, write 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 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.

(*) 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 $f \in C(\mathcal{X})$ and all $\varepsilon > 0$ there exists a function $g \in \mathcal{H}$ such that $\|f - g\|_\infty \leq \varepsilon$.

Intuition: with a universal kernel, we can approximate any function we like, and in particular we can separate any pair of disjoint compact subsets from each other.

Example: The Gaussian kernel on a compact subset \mathcal{X} of \mathbb{R}^d is universal.

The kernel being universal is a necessary requirement if we want to construct learning algorithms that are uniformly Bayes consistent.

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.
- ▶ 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

Kernels — history (2)

- ▶ 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 way to proceed is always similar:

- ▶ Start with a linear algorithm
- ▶ Try to write this algorithm in a way such that the only access to training points 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) 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 α .

SVMs with kernels (3)

Computing labels at new points:

We have already seen above how to compute the label of a test points for known values of α :

$$w = \sum_i \alpha_i Y_i X_i$$

$$b = Y_j - \sum_i Y_i \alpha_i \langle X_i, X_j \rangle \text{ for some } j \text{ such that } \alpha_j > 0.$$

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

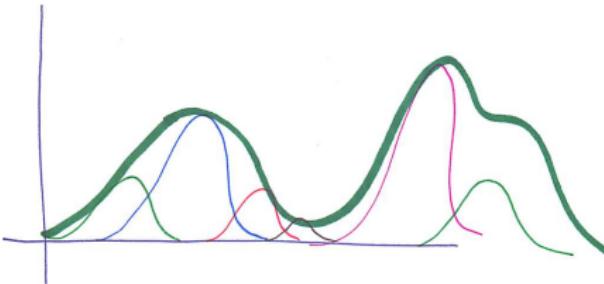
$$f(x) = \sum_i \beta_i k(x, X_i) + b$$

If k is a Gaussian kernel:

$$f(x) = \sum_i \beta_i \exp(-\|x - X_i\|^2 / (2\sigma^2))$$

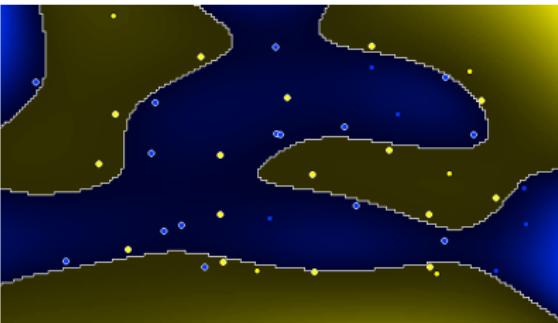
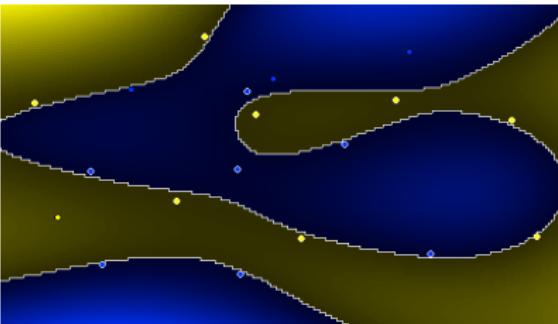
The power of kernels (2)

- 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 “decision surface” in \mathbb{R}^d , no matter how complicated, by an SVM with Gaussian kernel:

The power of kernels (3)



The power of kernels (4)

- ▶ Kernels with this property are called “universal” kernels. One can prove that SVMs with universal kernels are universally consistent in the sense we defined in the very beginning of the lecture, that is they approximate the Bayes risk.
- ▶ Note: if the kernel is universal, the underlying function class is huge (all continuous functions can be approximated). All the more important is that we regularize!

Regularization interpretation

Recall that we interpreted the linear primal SVM problem in terms of regularized risk minimization where the risk was the Hinge loss and we regularized by L_2 -regularizer $\|w\|^2$.

How does it look for the kernelized SVM?

- ▶ The loss function is still the Hinge loss because the part with the variables ξ_i does not change.
- ▶ But the regularizer is now $\|w\|^2$ where w is a vector in the feature space, and the norm is taken in the feature space.
- ▶ By the representer theorem,

$$\|w\|^2 = \left\langle \sum_i \beta_i \Phi(X_i), \sum_j \beta_j \Phi(X_j) \right\rangle = \beta^t K \beta$$

Regularization interpretation (2)

- ▶ It is not so easy to gain intuition about this norm (obviously it depends on the kernel). But at least we can say that the regularization “restricts the size of the function space” (in the sense that there are fewer functions that can be expressed with w with low norm than with high norm).

Regularization interpretation (3)

This regularization interpretation is really important, otherwise the SVM “could not work”:

- ▶ 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. At the time, they were a great invention, 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 is this the case?

- ▶ Convex optimization problem, easy to implement
- ▶ Very few variables to tune (C , and maybe a kernel parameter such as σ in the Gaussian kernel), this can be done by cross validation

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). Not going to be discussed in this lecture.

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 the “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 σ for the Gaussian kernel).

Later in the lecture we will look at more preprocessing steps that you should use (\leadsto non-vanilla-version).

(*) Kernelizing the SVM primal

AS AN EXERCISE: IT IS POSSIBLE TO EXPRESS THE PRIMAL OPTIMIZATION PROBLEM IN TERMS OF KERNELS?

$$\begin{aligned} & \text{minimize}_{w \in \mathcal{H}} \|w\|_{\mathcal{H}}^2 + \frac{C}{n} \sum_{i=1}^n \xi_i \\ & \text{subject to } Y_i \left(\langle w, \Phi(X_i) \rangle_{\mathcal{H}} \right) \geq 1 - \xi_i \quad (i = 1, \dots, n) \end{aligned}$$

(*) 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

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \frac{1}{d} \sum_{i=1}^d (Y_i - \langle \Phi(X_i), w \rangle)^2$$

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

$$\begin{aligned} & \underset{w \in \mathbb{R}^d}{\text{minimize}} \frac{1}{n} \sum_{i=1}^n (Y_i - \langle \Phi(X_i), w \rangle)^2 \\ \iff & \underset{\alpha \in \mathbb{R}^n}{\text{minimize}} \frac{1}{n} \sum_{i=1}^n \left(Y_i - \sum_{j=1}^n \alpha_j \underbrace{\langle \Phi(X_i), \Phi(X_j) \rangle}_{k_{ij}} \right)^2 \end{aligned}$$

Kernelizing least squares (first method via representer theorem) (2)

- In matrix notation:

$$\underset{\alpha \in \mathbb{R}^n}{\text{minimize}} \frac{1}{n} \|Y - K\alpha\|^2$$

- By taking the derivative with respect to α and exploiting that K is pd it is easy to see that the solution is given as $\alpha^* = K^{-1}Y$ (EXERCISE!).
- To evaluate the solution on a new data point x , we need to compute

$$f(x) = \langle \Phi(x), w \rangle = \sum_j \alpha_j \langle \Phi(x), \Phi(X_j) \rangle = \sum_j \alpha_j k(x, X_j)$$

- So we can express the optimization problem, its solution and the evaluation function purely in terms of kernel functions. We have kernelized least squares.

(*) Kernelizing least squares (second method via SVD)

Recap: the kernel matrix is $\Phi\Phi^t$

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

(*) Kernelizing least squares (second method via SVD) (2)

Proposition 22 (Matrix Identities)

For any $n \times d$ -matrix Φ we have

$$(\Phi^t \Phi)^{-1} \Phi^t = \Phi^t (\Phi \Phi^t)^{-1}$$

Proof of the proposition.

- ▶ Let $\Phi = U \Sigma V^t$ the singular value decomposition of Φ .
- ▶ It is straightforward to prove that $(\Phi^t \Phi)^{-1} \Phi^t = V \Sigma^+ U^t$ (have seen this already when we derived least squares).
- ▶ It is even more straightforward to see that $\Phi^t (\Phi \Phi^t)^{-1} = V \Sigma^+ U^t$. 

(*) Kernelizing least squares (second method via SVD) (3)

Using this proposition and the fact that the kernel matrix K is given as $\Phi\Phi^t$ we can rewrite the least squares solution as

$$\mathbf{w}^* = (\Phi^t\Phi)^{-1}\Phi^tY = \Phi^t(\Phi\Phi^t)Y^{-1} = \Phi^tK^{-1}Y$$

Denote $\alpha := K^{-1}Y$. With this notation, the evaluation function is

$$\begin{aligned} f(x) &= \langle w^*, \Phi(x) \rangle = (w^*)^t\Phi(x) \\ &= (\Phi^tK^{-1}Y)^t\Phi(x) = Y^tK^{-1}\Phi^t\Phi(x) \\ &= \alpha^t\Phi\Phi^t(x) \\ &= \sum_{j=1}^n \alpha_j \Phi(X_j)^t\Phi(x) \\ &= \sum_{j=1}^n \alpha_j k(X_j, x) \end{aligned}$$

(*) 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

$$\alpha = (n\lambda I + K)^{-1} Y$$

A subtle difference: Ridge regression vs. kernel ridge regression

- ▶ Given points in space \mathcal{X} . Want to compare:
 - ▶ Ridge regression using basis functions $\Phi_i(x) = k(X_i, x)$
 - ▶ Kernel ridge regression using kernel k and feature map Φ
- ▶ In both cases, we work in the same function space, namely the one spanned by the functions Φ_i .
- ▶ So no matter which function we use, the least squares error is the same in both approaches.
- ▶ **However, the regularizers are different:**
 - ▶ In the standard case we regularize by $\|\alpha\|^2$.
 - ▶ In the kernel case we regularize by $\alpha^t K \alpha$.
- ▶ This is as for linear and kernel SVMs ...

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:

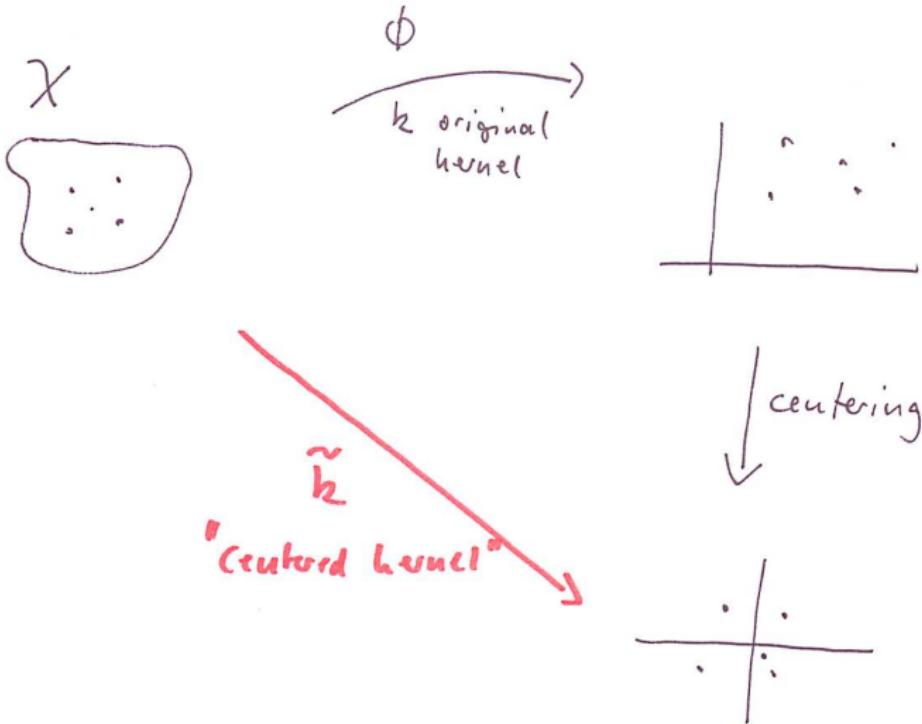
$$\tilde{K} = (K - \mathbb{1}_n K - K \mathbb{1}_n + \mathbb{1}_n K \mathbb{1}_n)$$

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

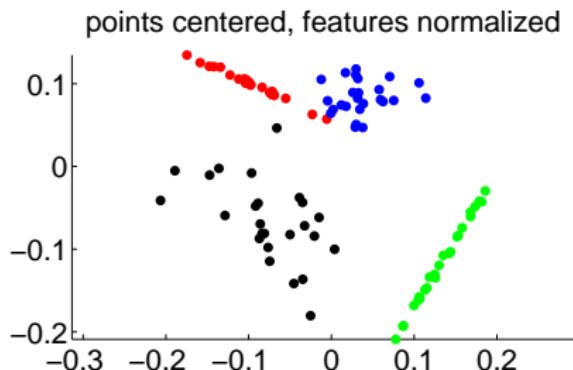
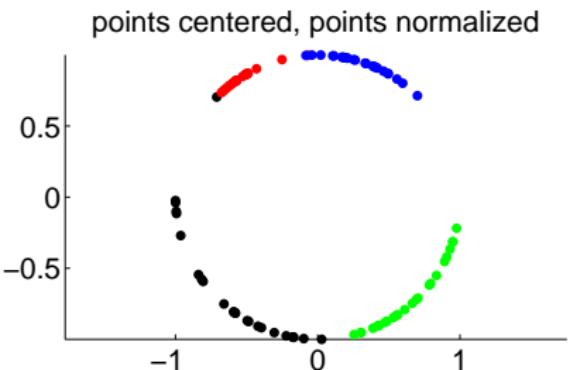
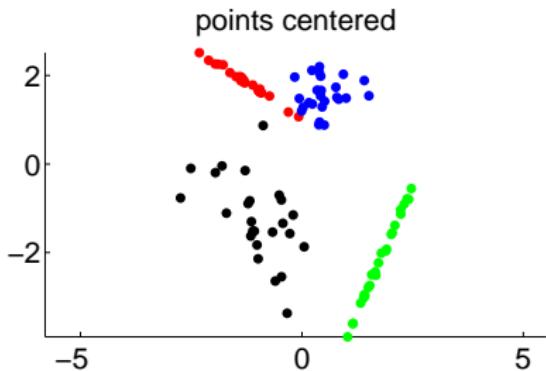
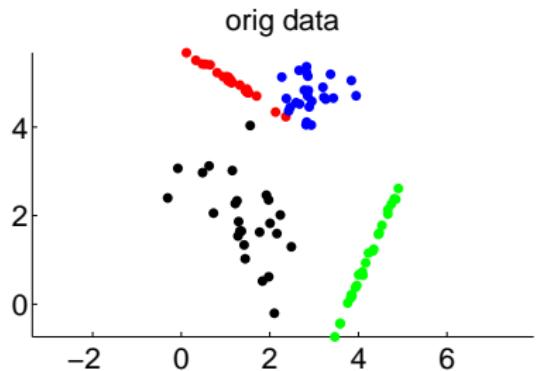
Assume our n data points are in \mathbb{R}^d and we stack them in a data matrix X as usual:

- ▶ 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 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.
 - ▶ In some algorithms it is necessary that the points are normalized to norm 1.
 - ▶ For some other case, it makes sense to do this normalization, but sometimes it also hurts.
 - ▶ If in doubt, use cross-validation to see whether your results improve if you normalize or not.
- ▶ Normalize features:
 - ▶ This just rescales the data, it does not remove information. Typically this never hurts, and often helps.
 - ▶ For kernel methods it is impossible to normalize the features (we don't know the embedding Φ explicitly, in particular we don't know what the features are).

Normalizing in feature space (4)

To normalize the points such that they have unit norm in the feature space:

- ▶ Assume the data points are already centered in feature space.
- ▶ Then define the normalized data point
 $\hat{\Phi}(X) := \Phi(X)/\|\Phi(X)\|.$
- ▶ Observe:

$$\begin{aligned}\langle \hat{\Phi}(X), \hat{\Phi}(Y) \rangle &= \left\langle \frac{\Phi(X)}{\|\Phi(X)\|}, \frac{\Phi(Y)}{\|\Phi(Y)\|} \right\rangle \\ &= \frac{\langle \Phi(X), \Phi(Y) \rangle}{\|\Phi(X)\| \|\Phi(Y)\|} \\ &= \frac{k(x, y)}{\sqrt{k(x, x)k(y, y)}}\end{aligned}$$

Normalizing in feature space (5)

So instead of first normalizing the points and then computing their kernels we can directly compute the kernels for the normalized data points.

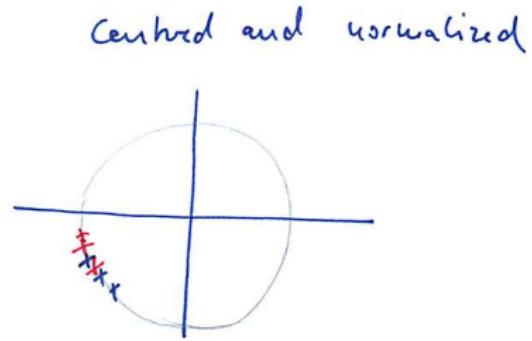
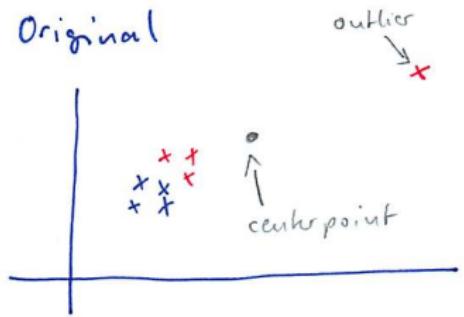
So to normalize the points in feature space, we simply replace the kernel function k by the normalized kernel function

$$\hat{k}(x, y) = \frac{k(x, y)}{\sqrt{k(x, x)k(y, y)}}.$$

(VERIFY THAT THIS IS INDEED A KERNEL)

When it can go wrong

Standardizing the data is a preprocessing step. As any such step, it often helps, but sometimes can go wrong:



Summary: data standardization for kernel methods

- ▶ We first center the data points, this gives us a new kernel matrix \tilde{K} .
- ▶ Then we normalize the data points in the feature space, this gives us yet another kernel matrix \hat{K} .
- ▶ This is then the matrix we work with!

Unsupervised learning

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

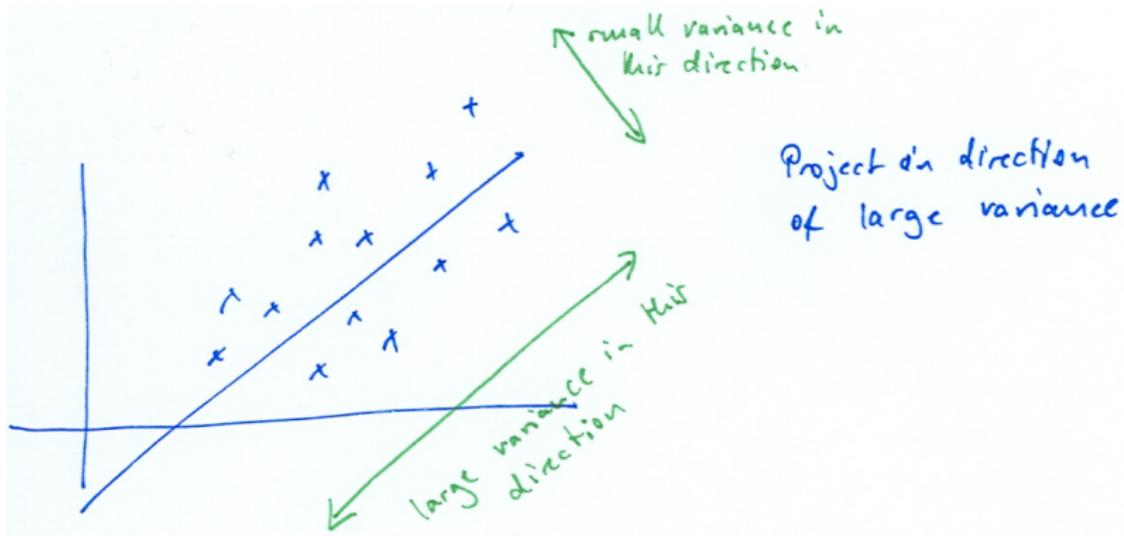
Principle component analysis (PCA)

... 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 $\ell < d$.

Principle component analysis (PCA) (2)



Principle 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

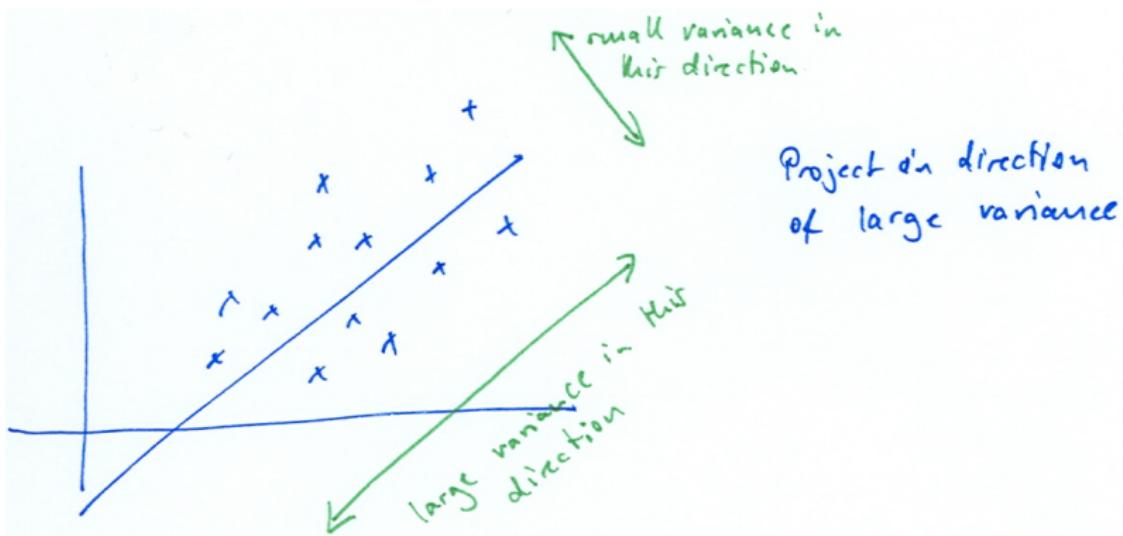
Recap: Projections

... see slides in the appendix ...

Recap: Variance and Covariance

... see slides in the appendix ...

PCA by max variance approach: Idea



Want to find a linear projection on a low-dim space such that the overall variance of the resulting points is as large as possible.

PCA by max variance approach: Idea (2)

Given: data points $x_1, \dots, x_n \in \mathbb{R}^d$, parameter $\ell < d$ (the dimension of the space we want to project to).

Goal: find a projection π_S on an affine subspace S such that the variance of the projected points is maximized: $\max_S \text{Var}_\ell(\pi_S(X))$.

For simplicity, let us assume that the data points are centered:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i = 0$$

If this is not the case, we can center the data points by setting $\tilde{x}_i = x_i - \bar{x}$.

PCA by max variance approach: Case $\ell = 1$

One-dimensional case:

We first of all assume that $\ell = 1$, that is we want to project the data points on a 1-dim space.

Have to solve the following optimization problem:

$$\max_{a \in \mathbb{R}^d, \|a\|=1} \text{Var}(\pi_a(X))$$

$$\iff \max_{a \in \mathbb{R}^d} \sum_{i=1}^n (\pi_a(x_i))^2 \quad \text{subject to } a'a = 1$$

$$\iff \max_{a \in \mathbb{R}^d} \sum_{i=1}^n (a'x_i)^2 \quad \text{subject to } a'a = 1$$

$$\iff \max_{a \in \mathbb{R}^d} \|Xa\|^2 \quad \text{subject to } a'a = 1$$

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 first ℓ eigenvectors of $X'X$.

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_ℓ as the matrix containing the ℓ largest eigenvectors (i.e., the first ℓ columns of V if the eigs in D are ordered decreasingly).
- ▶ Compute the new data points:
 - ▶ View 2: $y_i = V_\ell' \tilde{x}_i \in \mathbb{R}^\ell$
 - ▶ 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

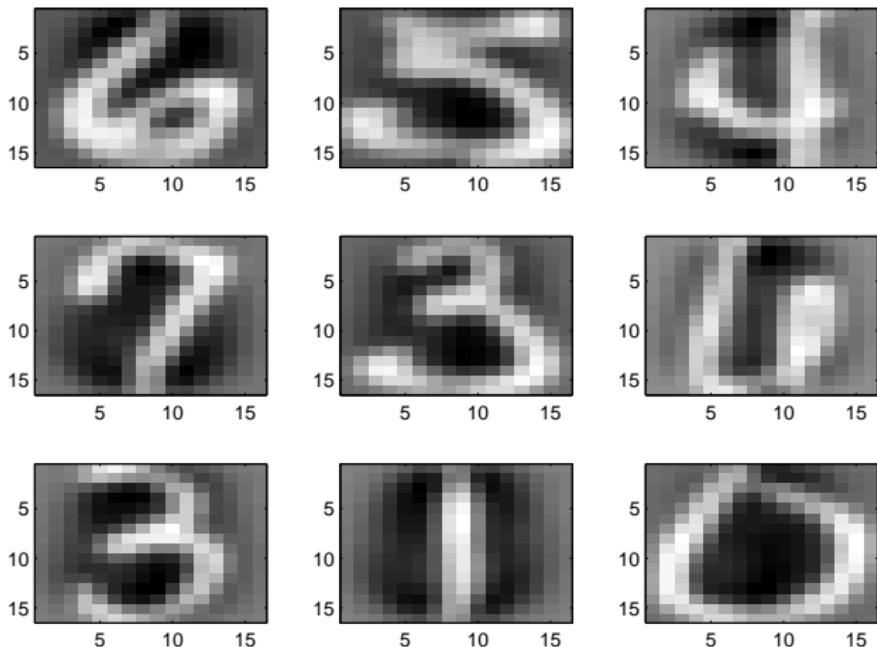
USPS example

USPS handwritten digits, 16×16 greyscale images.

\rightsquigarrow `demo_pca_usps.m`

USPS example (2)

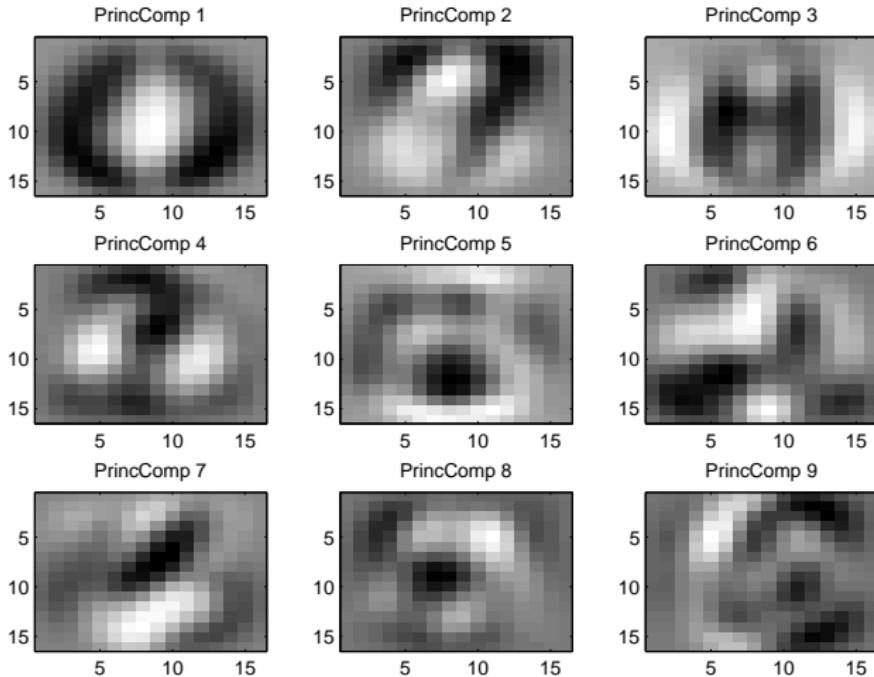
Some digits from the data set



USPS example (3)

The first principle components (computed based on 500 digits):

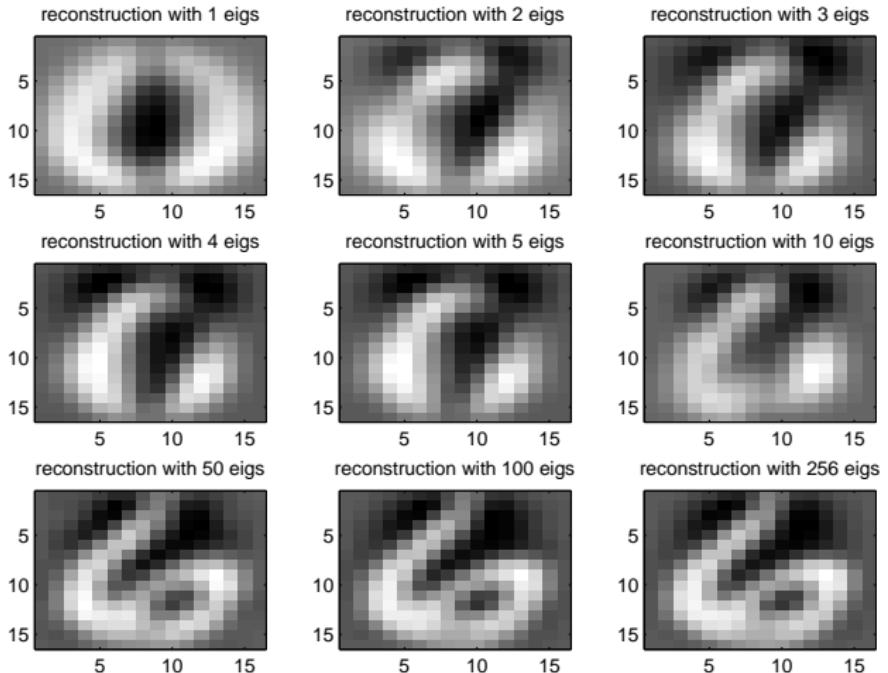
First principal components



USPS example (4)

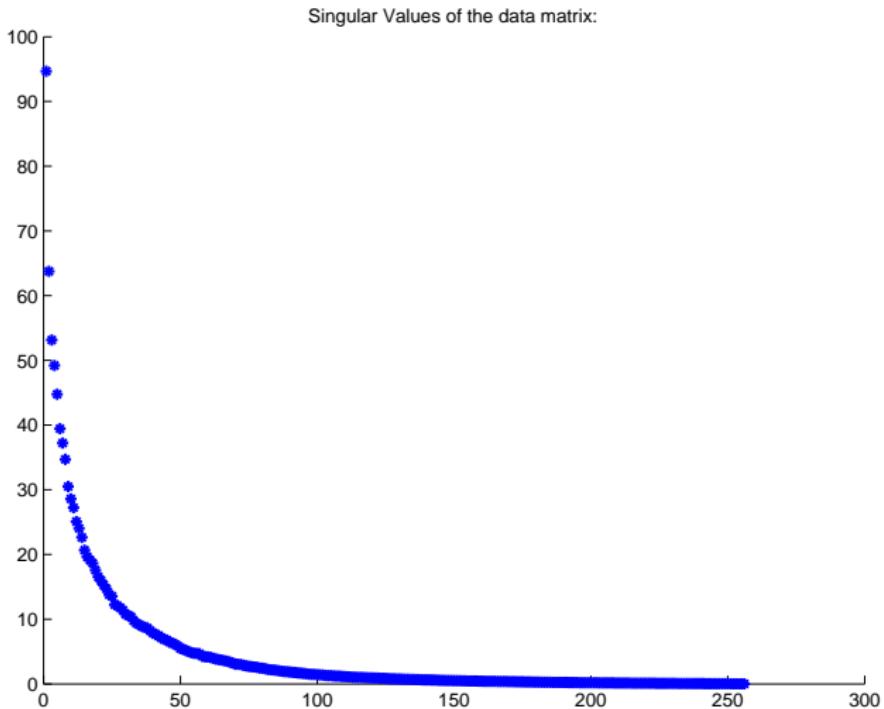
Reconstructing digits:

Reconstructed first digit



USPS example (5)

All eigenvalues:



Eigenfaces

Principal components for a data set of faces:



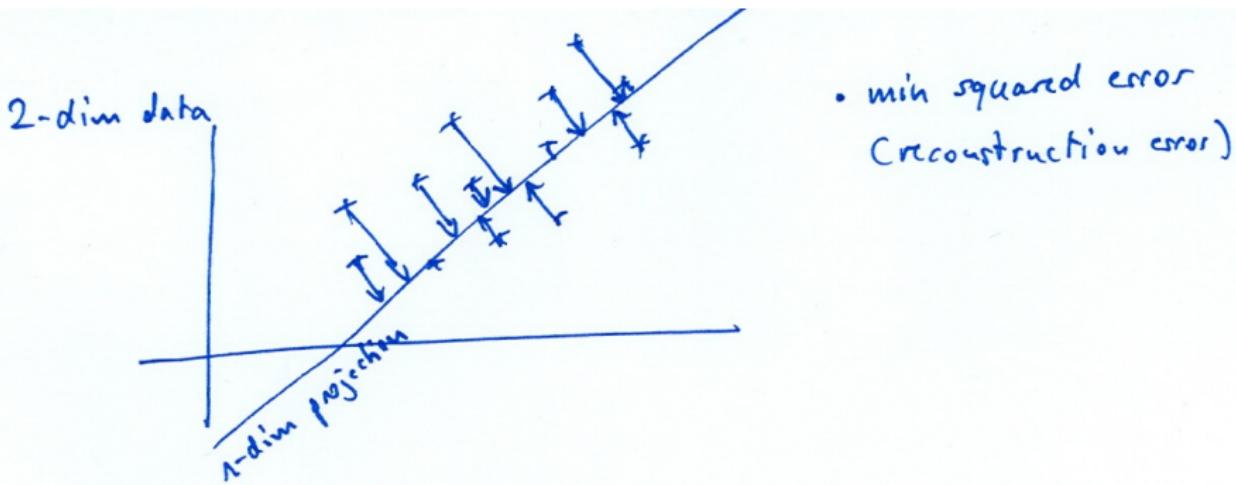
PCA – min squared error approach

Second approach to PCA:

Find a projection π_S on an affine subspace S such that the squared distance between the points and their projections is minimized:

$$\min_S \sum_{i=1}^n \|x_i - \pi_S(x_i)\|^2.$$

PCA – min squared error approach (2)



PCA – min squared error approach (3)

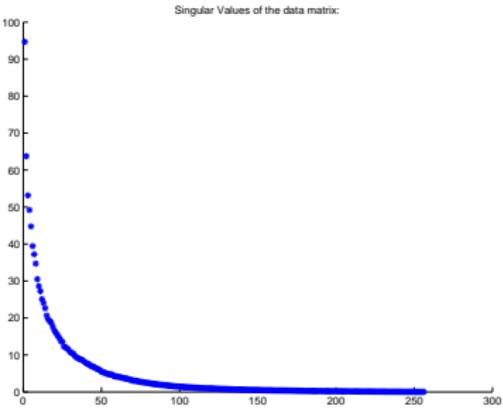
One can prove that this approach leads to exactly the same solution as the one induced by the max-variance criterion.

We skip the derivation ...

Choosing the parameter ℓ

- ▶ Heuristic: look at largest eigenvalues, and take the most “informative” ones. It also can be seen: the reconstruction error is bounded as

$$\sum_i \|x_i - \pi_\ell x_i\|^2 \leq \sum_{k=\ell+1}^n \lambda_k$$



Choosing the parameter ℓ (2)

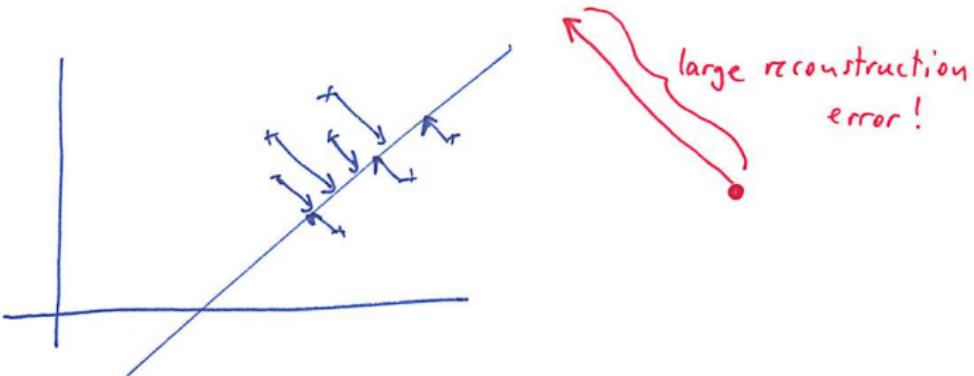
- If PCA is used as a preprocessing step for supervised learning, then use cross validation to set the parameter ℓ !

Note: It is not a priori clear whether it is better to choose ℓ large or small ...

Global!

Keep in mind that PCA optimizes global criteria.

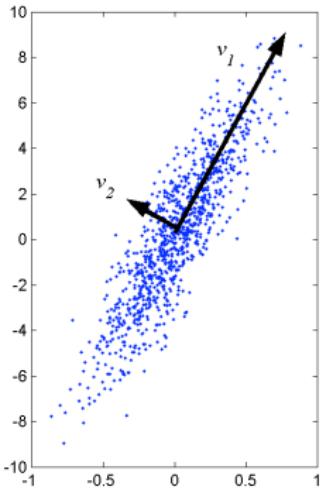
- ▶ No guarantees what happens to individual data points. This is different for some other dimensionality reduction methods (such as random projections and Johnson-Lindenstrauss, in case you have seen this before).



- ▶ If the sample size is small, then outliers can have a large effect on PCA.

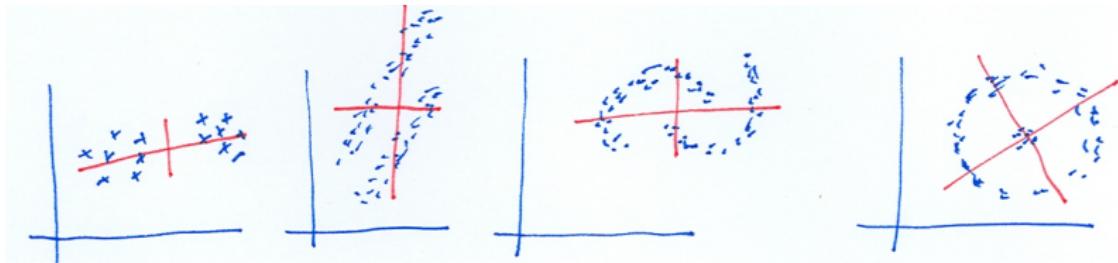
When does it (not) make sense?

- ▶ 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 $C = X'X \in \mathbb{R}^{d \times d}$ and the kernel matrix $K = XX' \in \mathbb{R}^{n \times n}$.

Recap: Covariance matrix vs. kernel matrix

Note:

- ▶ Covariance matrix is $C = X'X$
- ▶ Kernel matrix is $K = XX'$

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 23 (Eigs of K via eigs of C)

Consider a set of points $x_1, \dots, x_n \in \mathbb{R}^d$. Assume that λ and a are an eigenvalue and corresponding eigenvector of the kernel matrix K (with respect to the standard scalar product). Assume that a is not orthogonal to $\text{span}\{x_1, \dots, x_n\}$. Then $v : X'a$ is an eigenvector of C with eigenvalue λ .

Proof.

Expressing the eigs of K by those of C (2)

- $Ka = \lambda a \iff XX'a = \lambda a \implies X'XX'a = \lambda X'a \iff Cv = \lambda v$



Main message: So all eigs (λ, a) of K give rise to eigs (λ, v) of C unless $v = Xa = 0$.

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 24 (Eigs of C via eigs of K)

Assume that the points x_i are centered. Then to compute the ℓ -th eigenvector v of C we can proceed as follows:

- ▶ compute the matrix K
- ▶ compute its ℓ -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:

$$\lambda v = Cv \text{ and } C = 1/n \sum_{i=1}^n x_i x_i' \text{ implies}$$

$$\textcolor{red}{v} = \frac{1}{\lambda} Cv = \frac{1}{\lambda n} \sum_{i=1}^n x_i x_i' v = \frac{1}{\lambda n} \sum_{i=1}^n x_i \langle x_i, v \rangle =: \sum_j a_j x_j$$

Expressing the eigs of C by those of K (3)

Step 2: express eig of C as eig of K :

$$\begin{aligned} Cv \stackrel{!}{=} \lambda v &\iff (\sum_{j=1}^n x_j x'_j) (\sum_{i=1}^n a_i x_i) \stackrel{!}{=} \lambda \sum_{i=1}^n a_i x_i \\ &\iff \sum_{i,j=1}^n x_j x'_j a_i x_i \stackrel{!}{=} \lambda \sum_{i=1}^n a_i x_i \\ &\quad (\text{now multiply with } x'_s \text{ for some } s) \\ &\iff x'_s (\sum_{ij} a_i x_j \langle x_j, x_i \rangle) \stackrel{!}{=} \lambda \sum_i a_i x'_s x_i \\ &\iff \sum_{ij} a_i \langle x_s, x_j \rangle \langle x_j, x_i \rangle \stackrel{!}{=} \lambda \sum_i a_i \langle x_i, x_s \rangle \\ &\iff (K^2 a)_s = \lambda(Ka)_s \end{aligned}$$

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 $1/\sqrt{\lambda}$.



Expressing the projection on eigs of C using K

- ▶ Assume we want to project on eigenvector v of C . Have already seen that we can write $v = \sum_i a_i x_i$. Thus:

$$\pi_v(x_i) = v'x_i = \left\langle \sum_j a_j x_j, x_i \right\rangle = \sum_j a_j \langle x_j, x_i \rangle$$

- ▶ If we want to project on a subspace spanned by ℓ vectors $v_1, \dots, v_\ell \in \mathbb{R}^d$, compute each of the ℓ coordinates by this formula as well.
- ▶ To compute the projections, we only need scalar products ☺

Finally: kernel PCA

Input: Data points X_1, \dots, X_n , parameter ℓ

- ▶ 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 λ_k the corresponding eigenvalue.
- ▶ Define the matrix V_ℓ which has the columns $a_k / \sqrt{\lambda_k}$, $k = 1, \dots, \ell$
- ▶ Compute the low dim representation points $y_i = (y_i^1, \dots, y_i^\ell)'$ with the formula $y_i^\ell = \sum_{j=1}^n v_j^\ell \tilde{K}_{ji}$

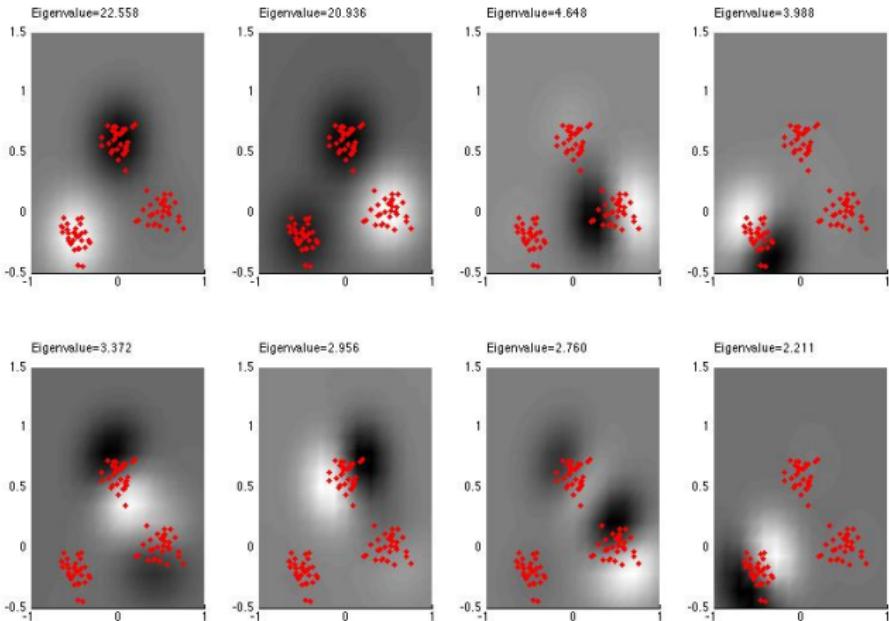
Output: $y_1, \dots, y_n \in \mathbb{R}^\ell$

kPCA toy example: three Gaussians

demo_kernel_pca_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)

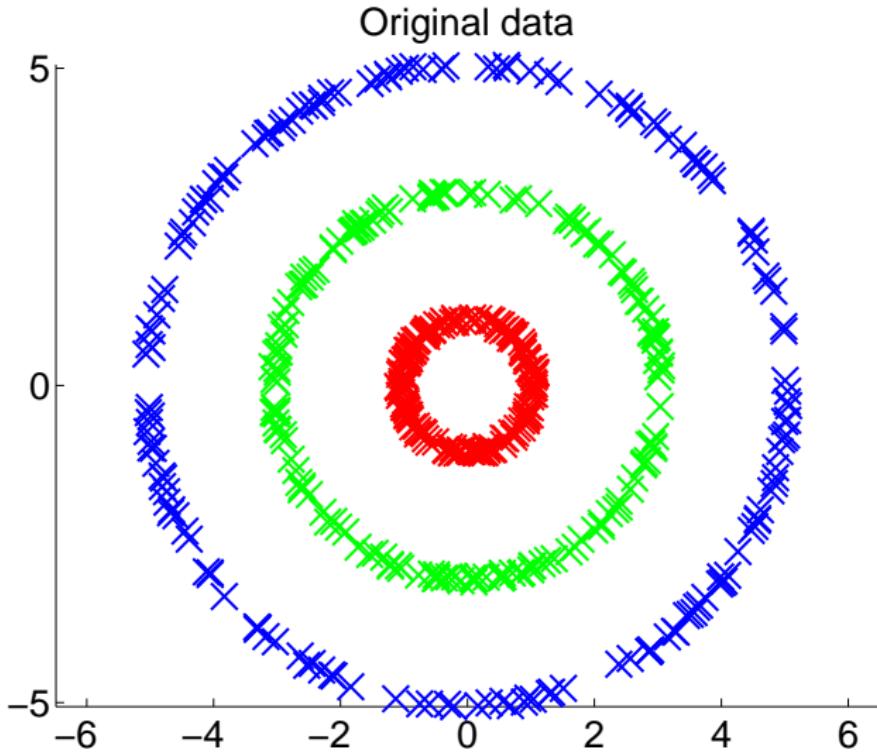


kPCA toy example: rings

`demo_kpca_toy.m`:

Consider the following three-dimensional data set:

kPCA toy example: rings (2)



kPCA toy example: rings (3)

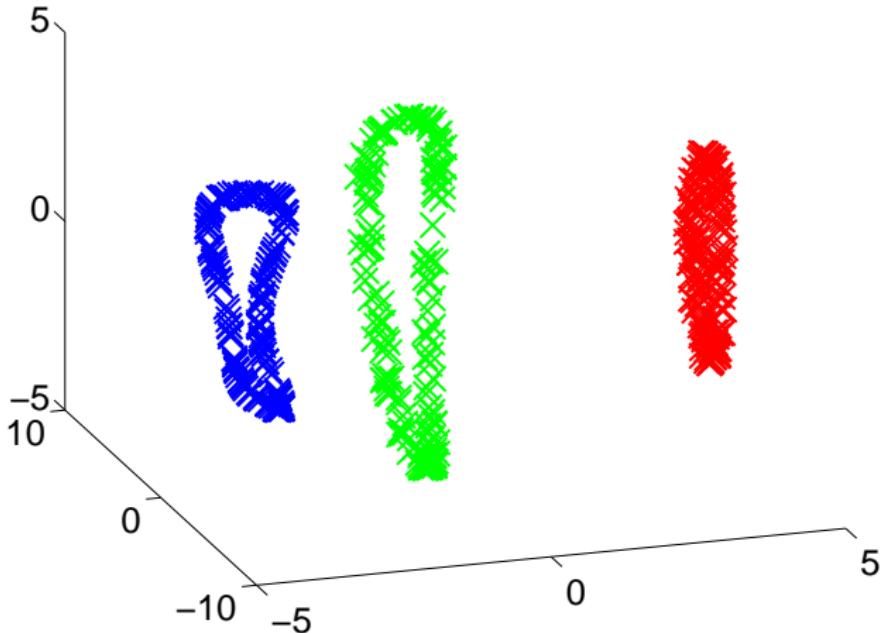
Now apply kPCA:

- ▶ 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 (4)

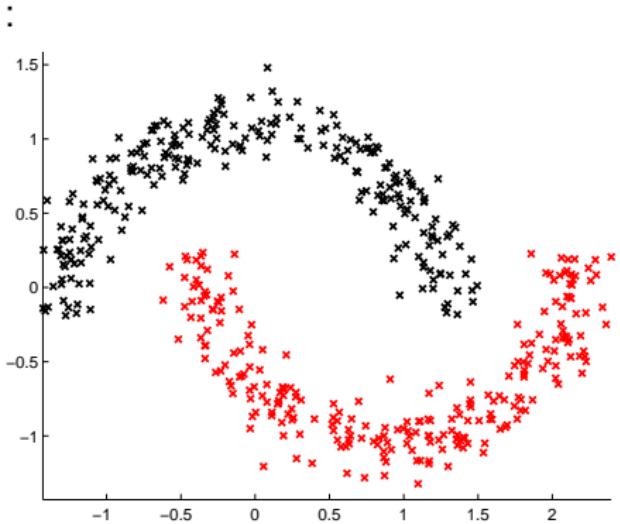
Here is the result:

3 dim kPCA

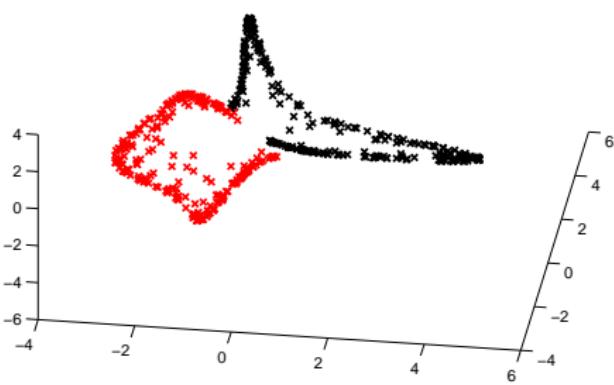


Surprising, isn't it???

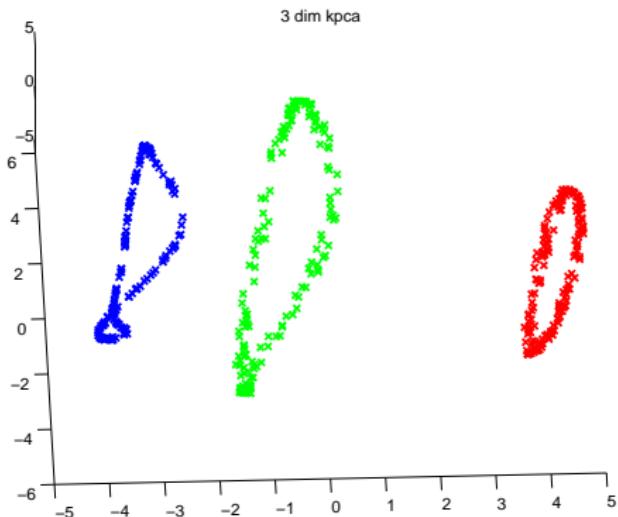
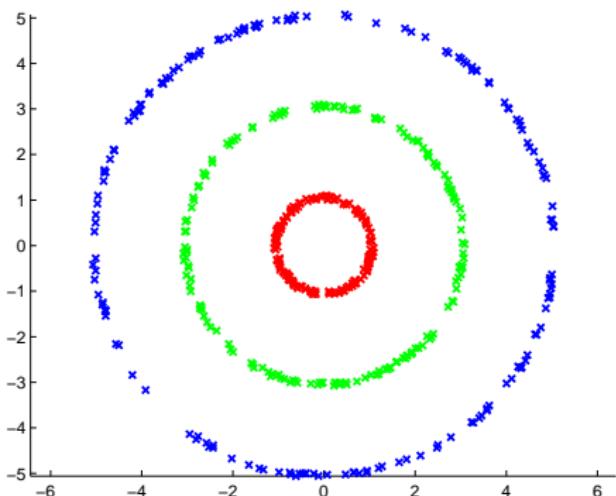
More toy examples



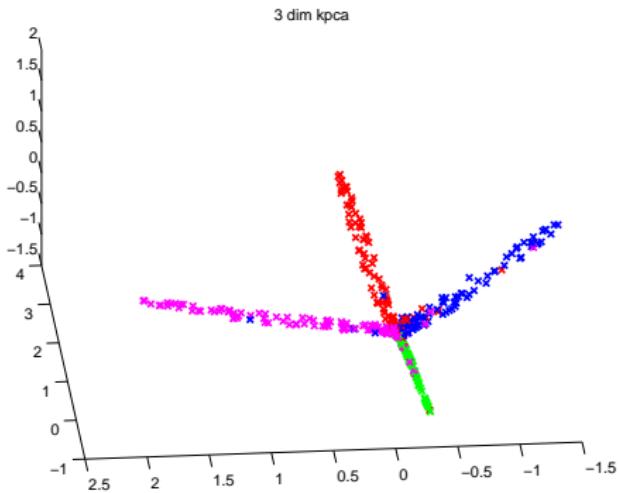
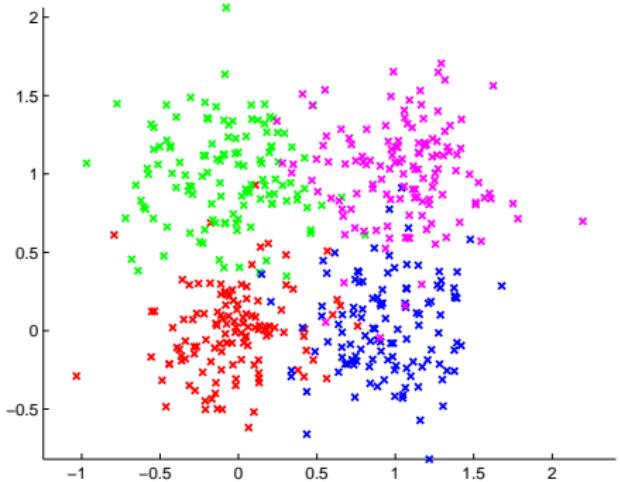
3 dim kpca



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 and Isomap

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.*

Multi-dimensional scaling

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 $\|\Phi(x_i) - \Phi(x_j)\| = d_{ij}$?

WHAT DO YOU THINK?

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 this in this lecture.

Embedding problem (3)

Why might we be interested in such an embedding?

- ▶ Visualization!
- ▶ Many algorithms are just defined for Euclidean data, so if we have abstract data we first need to find a Euclidean representation.
(Note that this is not always a good idea. WHY?)
- ▶ Identify low-dimensional structure, see Isomap below.

MDS in various flavors

- ▶ Classic MDS: we assume that the given distance matrix is Euclidean. For this case we construct an exact embedding.
- ▶ Metric MDS: we are given any distance matrix (might be non-Euclidean). We try to find an embedding that approximately preserves all distance.
 - ▶ 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 .

- ▶ 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 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 (3)

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.
- ▶ So 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).

Classic MDS (4)

Summary: Classic MDS:

- ▶ 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 Λ_d the $d \times d$ diagonal matrix with the first d eigenvalues on the diagonal, and then set $X = V_d\sqrt{\Lambda_d}$.

- ▶ Row i of X then gives the coordinates of the embedded point x_i .

Metric MDS

Metric MDS refers to the problem where the distance matrix D is no longer Euclidean, but we still believe that a good embedding should exist.

- ▶ 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 are two examples for such stress functions. The first minimizes the sum of absolute errors; the second one considers relative errors (WHEN MIGHT THIS BE USEFUL?). Many more stress functions are considered in the literature.

Metric MDS (2)

$$\text{stress}_1(\text{embedding}) = \sum_{i,j=1}^n (d(x_i, x_j) - d_{ij})^2$$

$$\text{stress}_2(\text{embedding}) = \frac{\sum_{ij} \frac{(d(x_i, x_j) - d_{ij})^2}{d_{ij}}}{\sum_{ij} d_{ij}}$$

- ▶ Then try to find an embedding with small loss by a standard non-convex optimization algorithm, say gradient descent.

Metric MDS (3)

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

There even exists a more general version of MDS.

- ▶ Instead of distance values, we are just given “ordinal data”, 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.
- ▶ In particular, it occurs if you have data of the form “film 1 and 2 are more similar than film 1 and film 3” or very unreliable metric data.
- ▶ Our group is also working on some aspects of this problem ☺

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

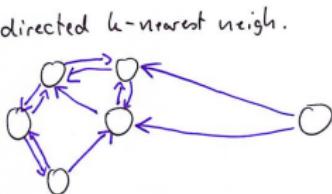
Neighborhood graphs

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) .

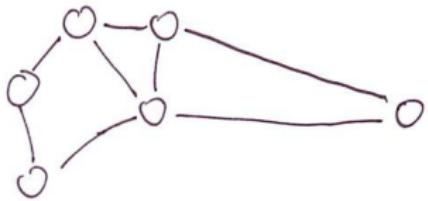


Note that this graph is not symmetric. To make this graph suitable for spectral clustering, we make it directed:

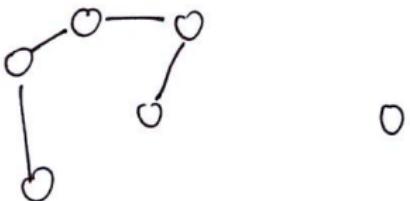
Neighborhood graphs (2)

- ▶ **Standard k -nearest neighbor graph:** put an edge between x_i and x_j if x_i is among the k nearest neighbors of x_j OR vice versa.
- ▶ **Mutual k -nearest neighbor graph:** put an edge between x_i and x_j if x_i is among the k nearest neighbors of x_j AND vice versa.

standard undirected



mutual undirected



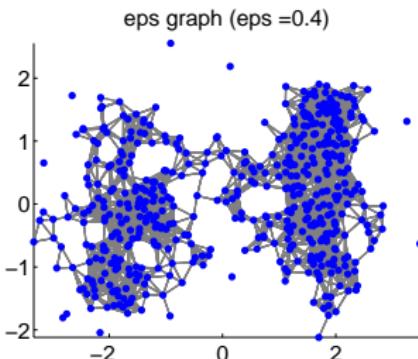
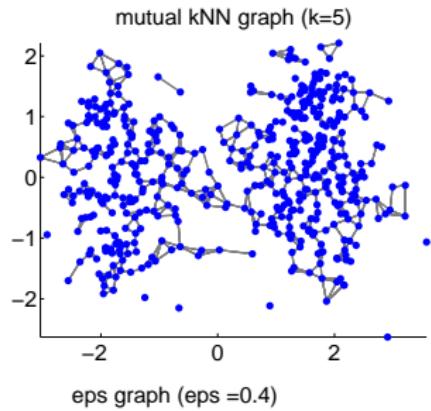
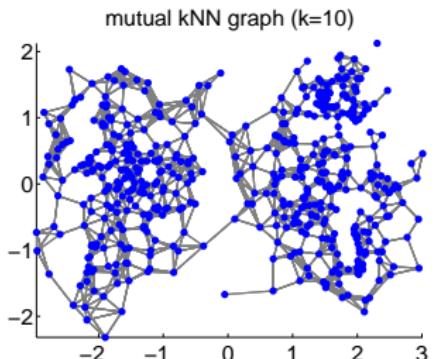
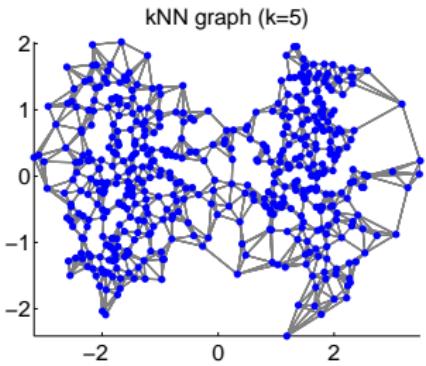
Neighborhood graphs (3)

Alternatively, we can use the ε -graph:

- ▶ Connect each point to all other points that have distance smaller than ε (or similarity larger than some threshold e)

Note: all these neighborhood graphs can be built based on similarities or based on distances.

Neighborhood graphs (4)



Neighborhood graphs (5)

Edge weights:

- ▶ A priori, all the graphs above are unweighted.
- ▶ On kNN graphs, it often makes sense to use **similarities as edge weights**.
(Reason: edges have very diverse “lengths”, and we want to tell this to the algorithm; spectral clustering is allowed to cut long edges more easily than short edges)
- ▶ **Never use distances as weights!** (this destroys the “logic” behind a neighborhood graph: no edge means “far away”, and no edge is the same as edge weight 0 ...)
- ▶ For ε -graphs, edge weights do not make so much sense because all edges are more or less “equally long”. The edge weights then do not carry much extra information.

Neighborhood graphs (6)

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 today)
 - ▶ 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”:

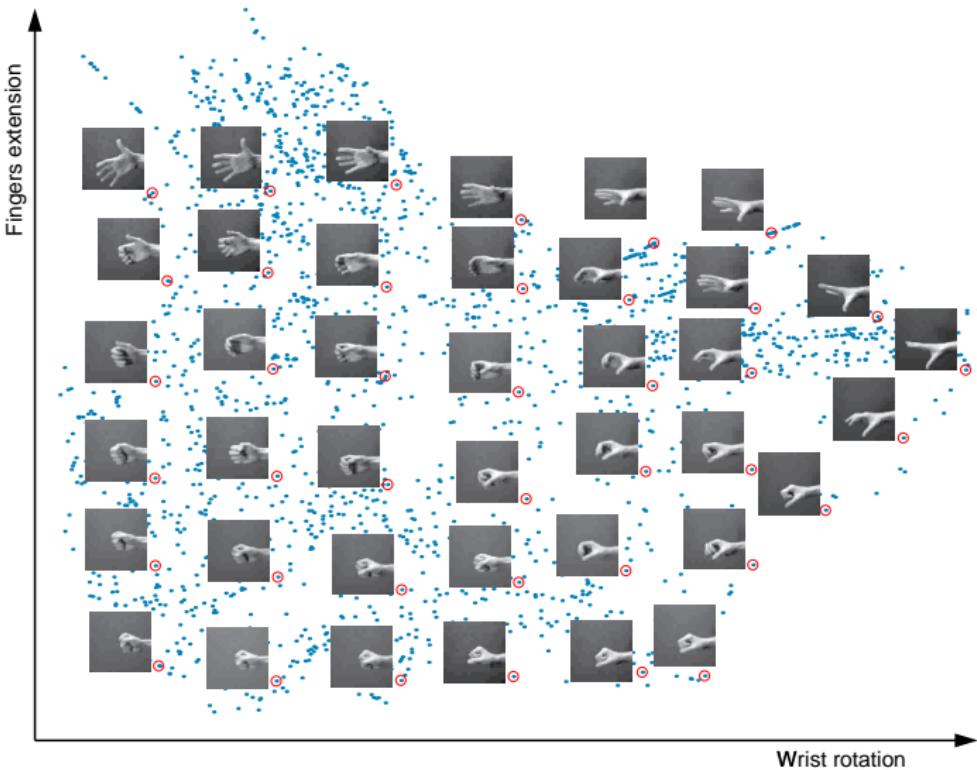
- ▶ Images of a tea pot, taken from all angles. Even though the images live in R^{256} , say, we believe they sit on a manifold corresponding to a circle:



Isomap (2)

- ▶ A phenomenon generates very high-dimensional data, but the “effective number of parameters” is very low

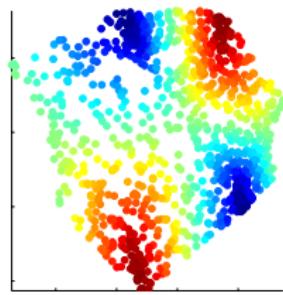
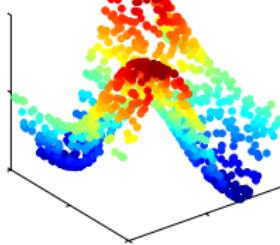
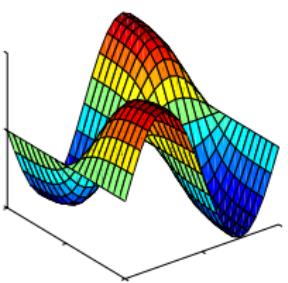
Isomap (3)



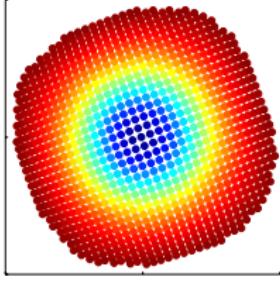
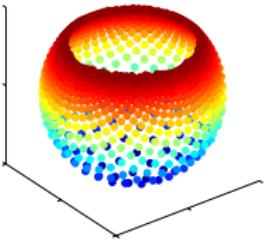
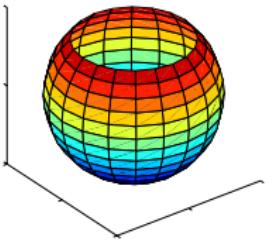
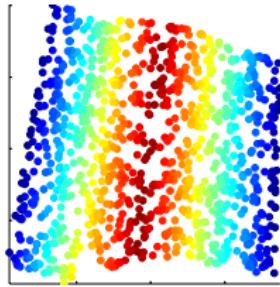
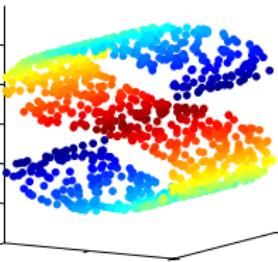
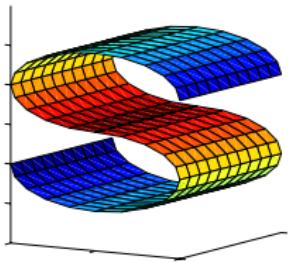
Isomap (4)

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.



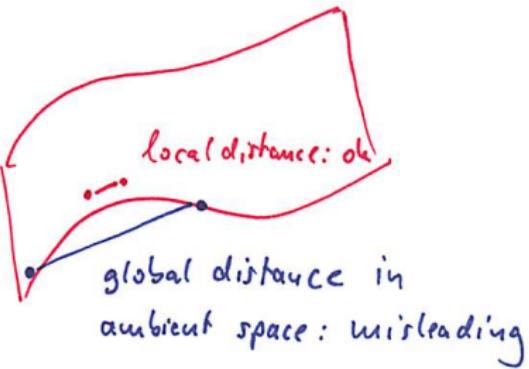
Isomap (5)



The Isomap algorithm

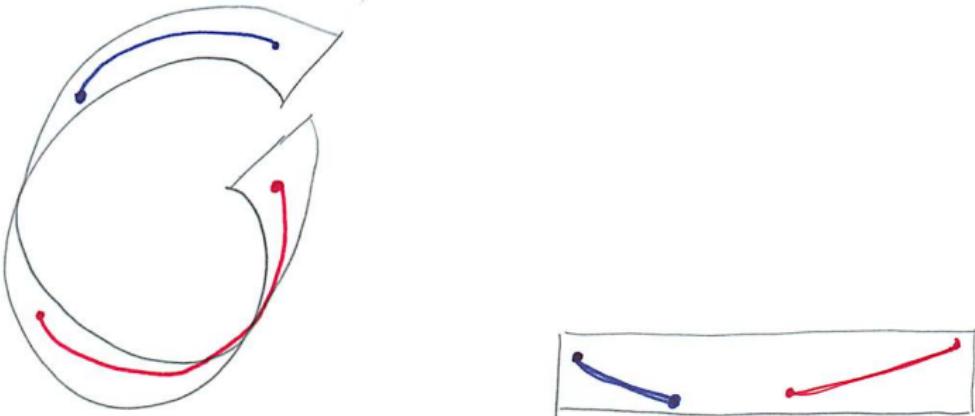
Intuition:

- ▶ In a small local region, Euclidean (extrinsic) distances between points on a manifold approximately coincide with the intrinsic distances. So we want to keep the local distances unchanged.
- ▶ This is no longer the case for large distances. We don't want to use the global Euclidean distances.



The Isomap algorithm (2)

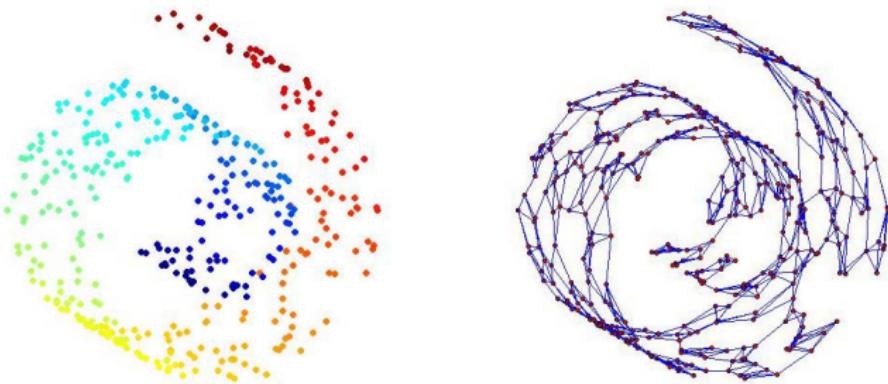
- If we want to “straighten” a manifold, we need to embed it in such a way that the Euclidean distance after embedding corresponds to the geodesic distance on the manifold.



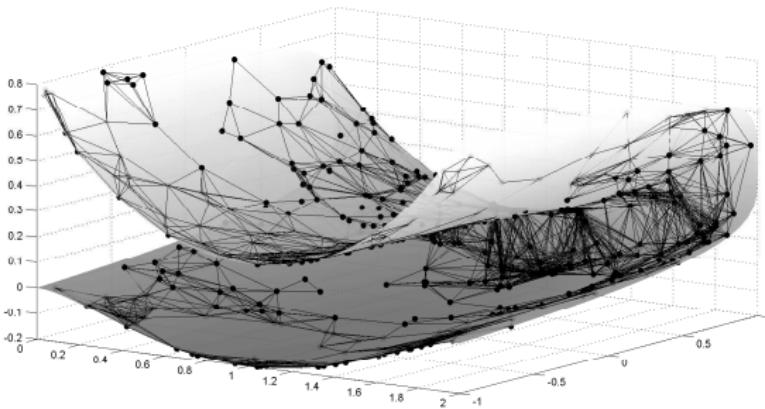
So we would like to discover the geodesic distances in the manifold.

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 .
These are 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.

History

- ▶ Manifold methods became very 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 very 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

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.

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:

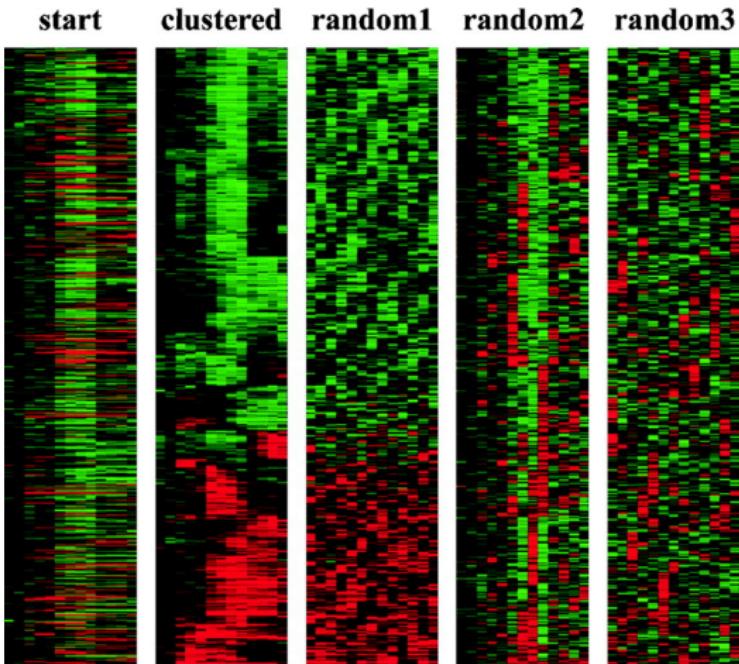
- ▶ Find “genres” of songs
- ▶ Find different “groups” of customers
- ▶ Find two different types of cancer, based on gene expression data
- ▶ Discover proteins that have a similar function
- ▶ ...

Data clustering (3)

Two main reasons to do this:

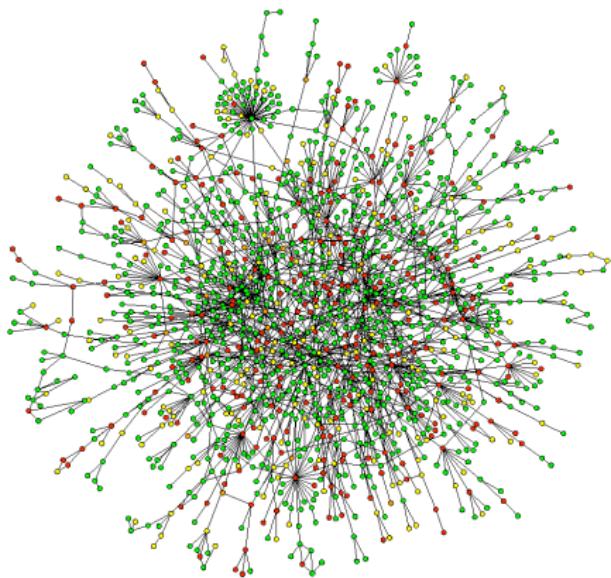
- ▶ Improve your understanding of the data! Exploratory data analysis.
- ▶ Reduce the complexity of the data. Vector quantization.
For example, instead of training on a set of 10^6 books, use 1000 “representative” books.
- ▶ Break your problem into subproblems and treat each cluster individually.

Example: Clustering gene expression data



M. Eisen et al., PNAS, 1998

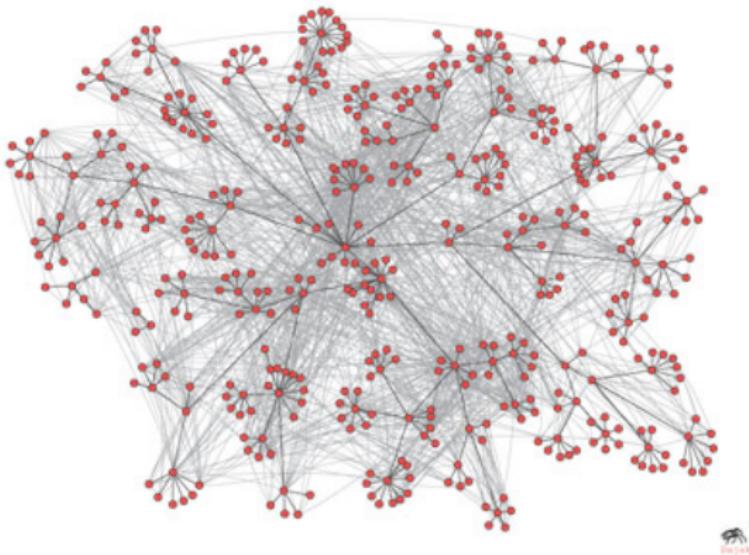
Example: Protein interaction networks



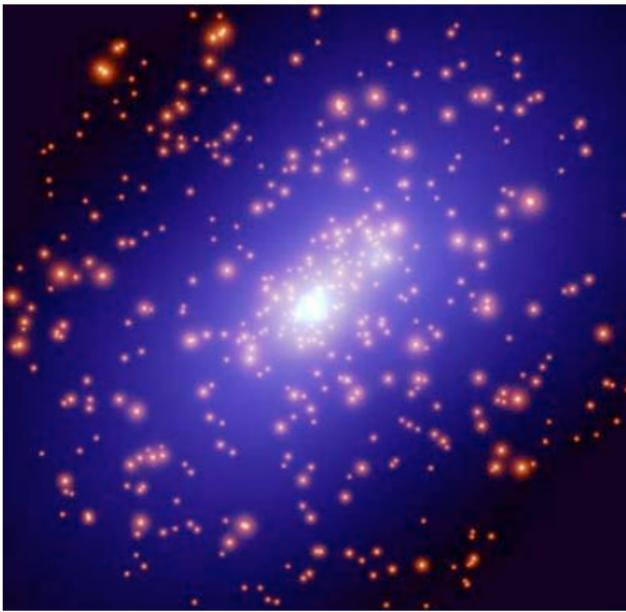
(from <http://www.math.cornell.edu/~durrett/RGD/RGD.html>)

Example: Social networks

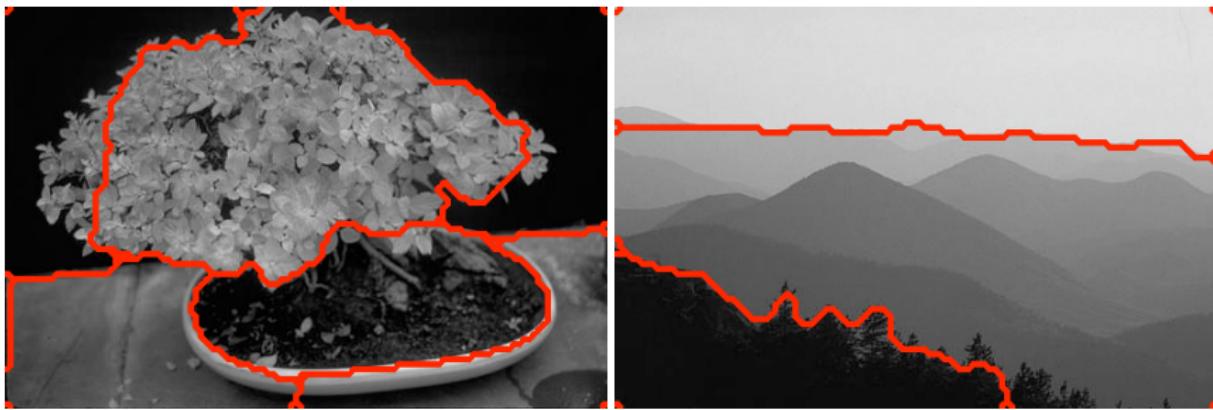
Corporate email communication (Adamic and Adar, 2005)



Example: Clustering galaxies

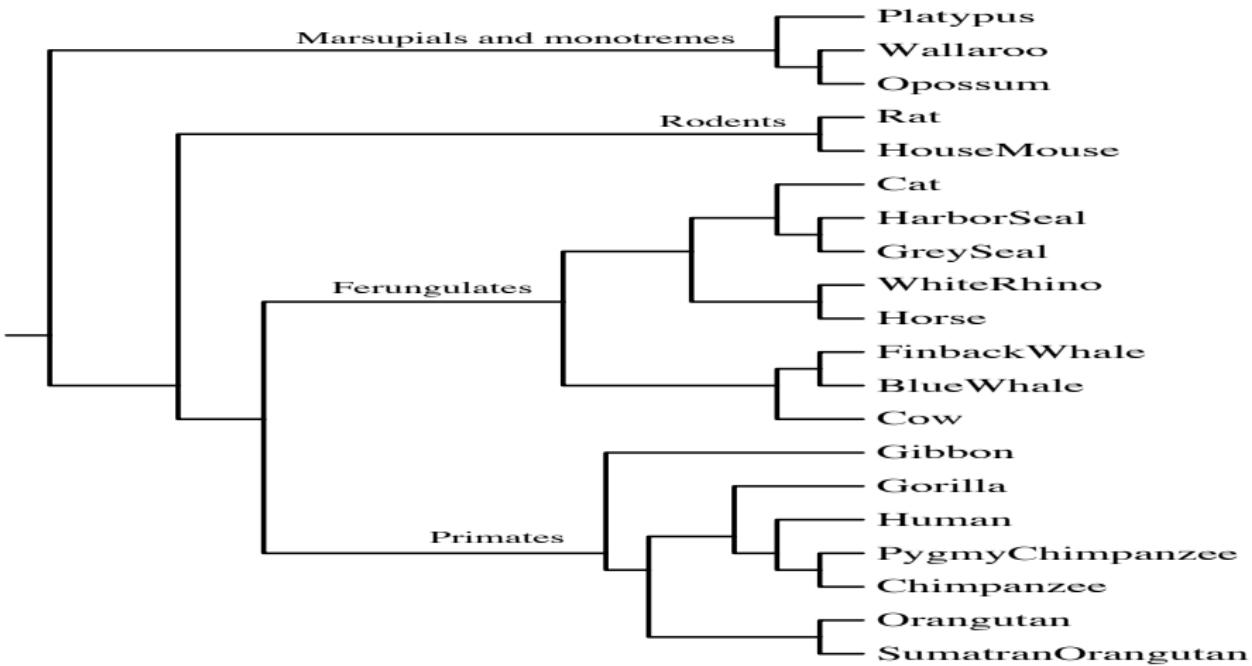


Example: Image segmentation



(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

k -means minimizes within-cluster distances

Proposition 25 (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:

$$\min_{m_1, \dots, m_K \in \mathbb{R}^d} \sum_{k=1}^K \sum_{i \in C_k} \|X_i - m_k\|^2$$

k -means minimizes within-cluster distances (2)

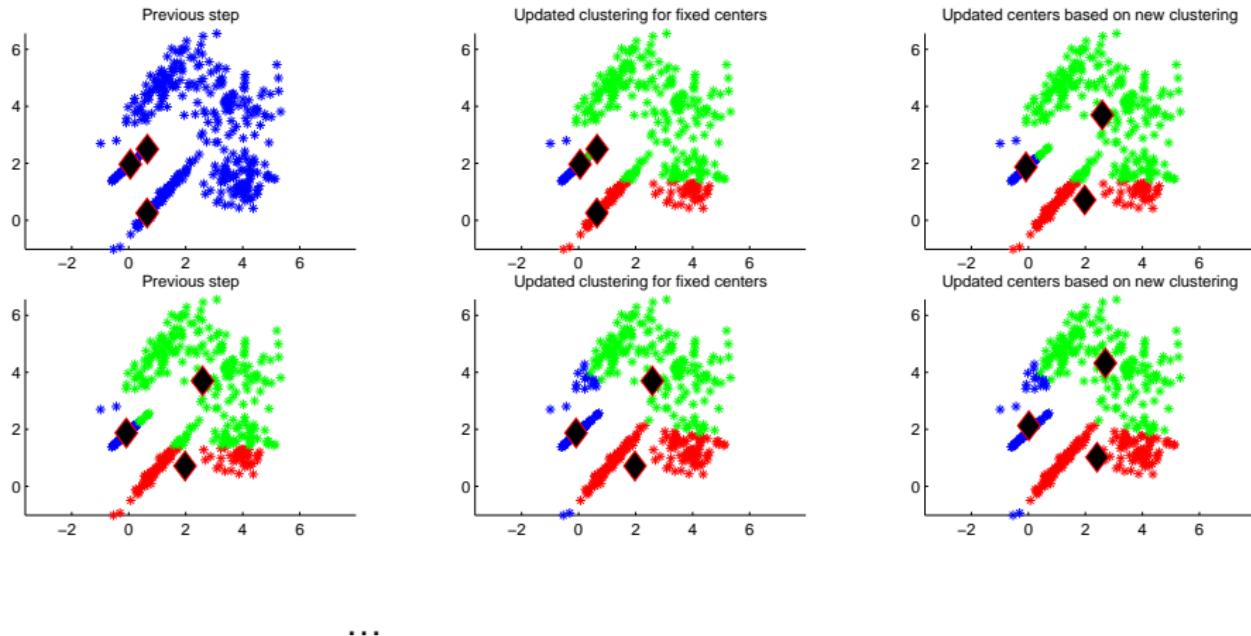
Proof. Elementary, but a bit lengthy, we skip it.

Lloyd's algorithm (k -means algorithm)

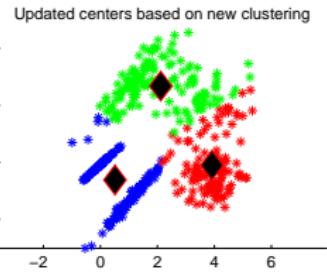
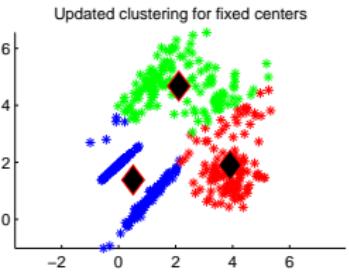
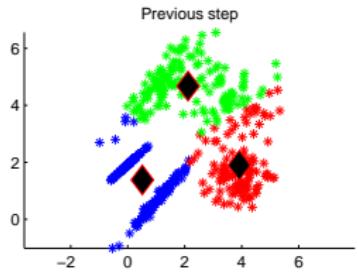
The following heuristic is typically used to find a local optimum of the k -means objective function:

- ▶ 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

$$m_k^{(i+1)} = \frac{1}{|C_k^{(i+1)}|} \sum_{s \in C_k} X_s$$

- 6 **Output:** Clusters C_1, \dots, C_K

Lloyd's algorithm (k -means algorithm) (5)

matlab demo: `demo_kmeans()`

K-means algorithm — Termination

Proposition 26 (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 is decreasing.
- ▶ There are only finitely many partitions we can inspect.
- ▶ So the algorithm has to terminate.



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 enumerate them 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 furthest 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.

K -means algorithm — Solutions can be arbitrarily bad

Proposition 27 (Bad solution possible)

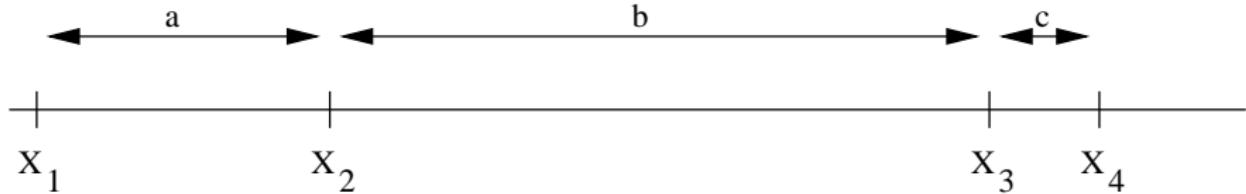
The algorithm ends in a local optimum which can be an arbitrary factor away from the global solution.

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)

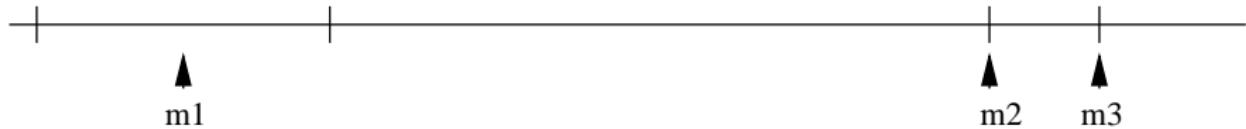
Data set: four points on the real line:



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:

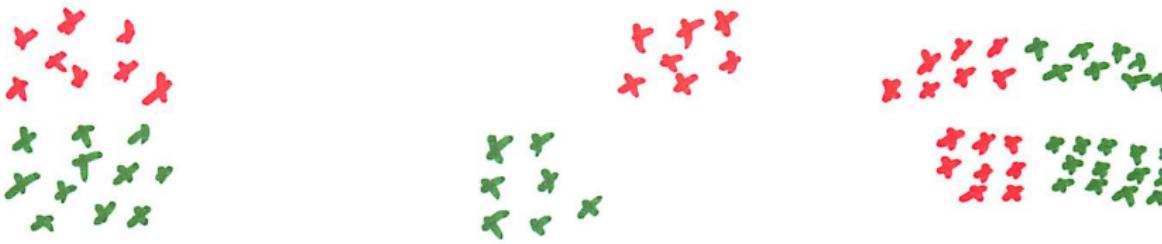
- ▶ Restart many times with different initializations.
- ▶ Swap individual points between clusters.
- ▶ Remove a cluster center, and introduce a completely new center instead.
- ▶ Merge clusters, and additionally introduce a completely new cluster center.
- ▶ Split a cluster in two pieces (preferably, one which has a very bad objective function). Then reduce the number of clusters again, for example by randomly removing one.

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.

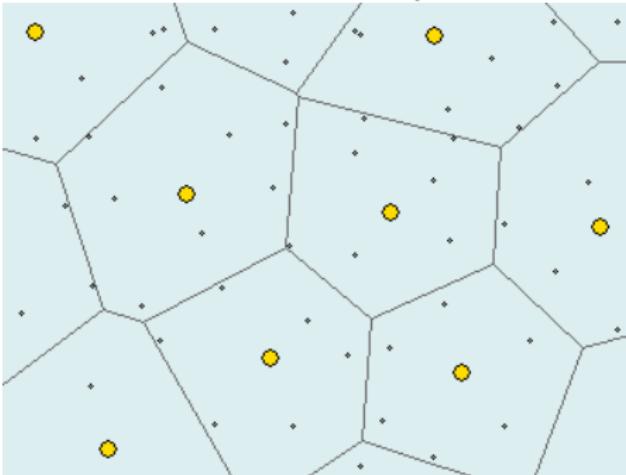
Disadvantages

- ▶ The k -means algorithm always constructs convex clusters!
This is unsuitable for many data sets:



Disadvantages (2)

- ▶ Reason: the clusters form a Voronoi partition of the space:



All cluster boundaries are linear.

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

- ▶ 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.

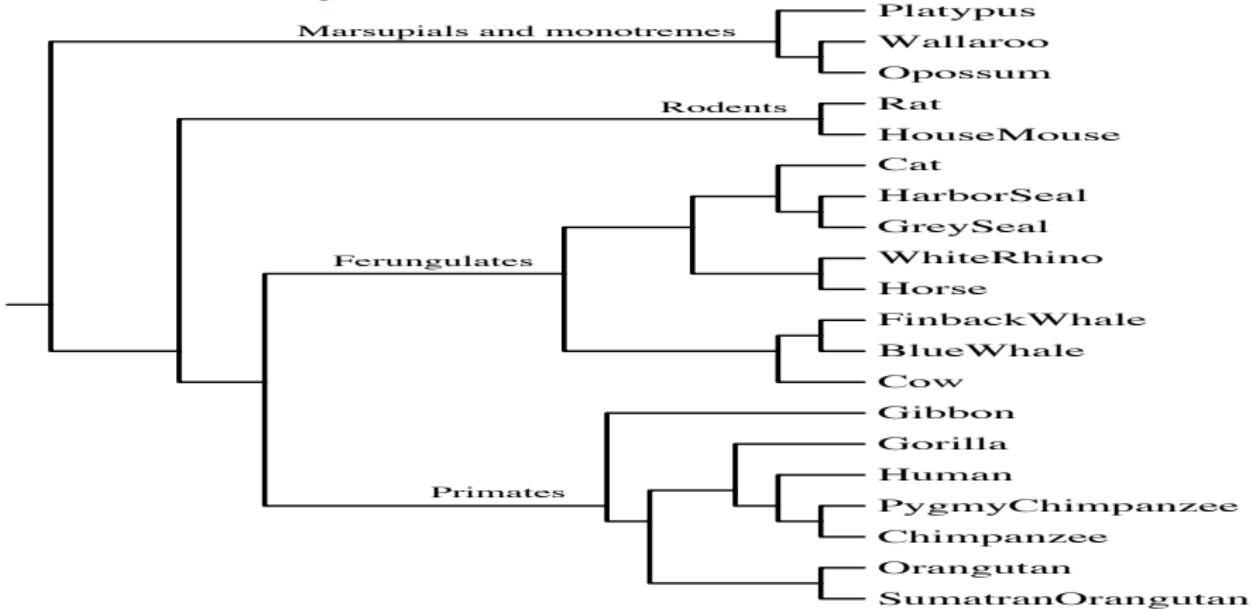
Summary K -means

- ▶ 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

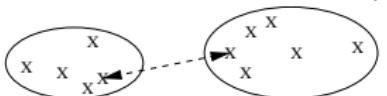
Agglomerative (bottom-up) strategy:

- ▶ 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

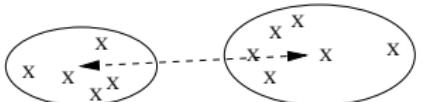
Simple idea (2)

To define which clusters are “closest”:

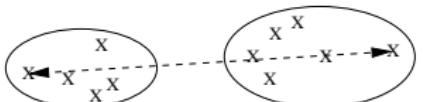
Single linkage: $dist(C, C') = \min_{x \in C, y \in C'} d(x, y)$



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 right” 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). See book repository.
- ▶ D. Spielman: Spectral Graph Theory. See paper repository.

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.

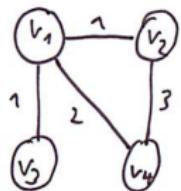
Unnormalized Laplacians

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 28 (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 29 (Simple spectral properties)

For an undirected graph with non-negative edge weights, the graph Laplacian has the following properties:

- ▶ L is symmetric and positive semi-definite.
- ▶ Smallest eigenvalue of L is 0, corresponding eigenvector is $\mathbb{1} := (1, \dots, 1)^t$.
- ▶ Thus eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$.

Unnormalized Laplacians: Spectral properties (2)

Proof.

Symmetry: W is symmetric (graph is undirected), D is symmetric, so L is symmetric.

Positive Semi-Definite: by key proposition:

$$f^t L f = \frac{1}{2} \sum_{ij} w_{ij} (f_i - f_j)^2 \geq 0$$

Smallest Eigenvalue/vector: It is indeed an eigenvector because

$$L\mathbf{1} = D\mathbf{1} - W\mathbf{1} = 0$$

It is the smallest because all eigs are ≥ 0 . □

Unnormalized Laplacians and connected components

Proposition 30 (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.

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_n \end{pmatrix}$$

- For each block (= each connected component), by the case $k = 1$ we know that there is exactly one eigenvector for eigenvalue 0, and it is constant:

$$v_1(L_1) = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \quad v_1(L_2) = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}, \dots$$

Unnormalized Laplacians and connected components (5)

- For the matrix L we then know that there are k eigenvalues 0, each one coming from one of the blocks. Padding the eigenvectors with zeros leads to the cluster indicator vectors:

$$v_1(L) = \begin{pmatrix} 1 \\ \vdots \\ 1 \\ 0 \\ \dots \\ 0 \end{pmatrix}$$

$$v_2(L) = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 1 \\ 0 \\ \dots \\ 0 \end{pmatrix}$$

...



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 31 (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 32 (Simple spectral properties)

Consider an undirected graph with non-negative edge weights.
Then:

1. λ is an eigenvalue of L_{rw} with eigenvector u
 $\iff \lambda$ is an eigenvalue of L_{sym} with eigenvector $w = D^{1/2}u$.
2. λ is an eigenvalue of L_{rw} with eigenvector u
 $\iff \lambda$ and u solve the generalized eigenproblem $Lu = \lambda Du$.
3. 0 is an eigenvalue of L_{rw} with the constant one vector $\mathbb{1}$ as eigenvector. 0 is an eigenvalue of L_{sym} with eigenvector $D^{1/2}\mathbb{1}$.
4. L_{sym} and L_{rw} are positive semi-definite and have n non-negative real-valued eigenvalues $0 = \lambda_1 \leq \dots \leq \lambda_n$.

Normalized Laplacians: First properties (3)

Proof.

Part (1): multiply the eigenvalue equation $L_{\text{sym}}w = \lambda w$ with $D^{-1/2}$ from the left and substitute $u = D^{-1/2}w$.

Part (2): multiply $L_{\text{rw}}u = \lambda u$ with D from the left.

Part (3): Just plug it in the corresponding eigenvalue equations.

Part (4): The statement about L_{sym} follows from the adapted key property, and then the statement about L_{rw} follows from (2). ☺

Normalized Laplacians and connected components

Proposition 33 (Relation between spectrum and clusters)

Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of both L_{rw} and L_{sym} equals the number of connected components A_1, \dots, A_k in the graph. For L_{rw} , the eigenspace of 0 is spanned by the indicator vectors $\mathbb{1}_{A_i}$ of those components. For L_{sym} , the eigenspace of 0 is spanned by the vectors $D^{1/2}\mathbb{1}_{A_i}$.

Proof.

Analogous to the one for the unnormalized case. □

Cheeger constant and isoperimetric problems

Cheeger constant

Let G be an unweighted, undirected graph, $S \subset V$ 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: $\text{cut}(S, \bar{S}) := |\{ \{u, v\} \in E \mid u \in S, v \notin S \}|$.
- ▶ Cheeger constant:

$$h_G(S) := \frac{\text{cut}(S, \bar{S})}{\min\{\text{vol}(S), \text{vol}(\bar{S})\}}$$
$$h_G := \min_{S \subset V} h(S)$$

Isoperimetric problem

In general geometry:

Find a set S with given volume such that its boundary ∂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 λ_2

Theorem 34 (Cheeger inequality for graphs)

Consider a connected, undirected, unweighted graph. Let λ_2 be the second-smallest eigenvalue of L_{sym} . Then

$$\frac{\lambda_2}{2} \leq h_G \leq \sqrt{2\lambda_2}$$

Intuition:

- ▶ The Cheeger constant describes the cluster properties of a graph.
- ▶ The Cheeger constant is controlled by the second eigenvalue.

Proof. Blackboard.



Spectral clustering

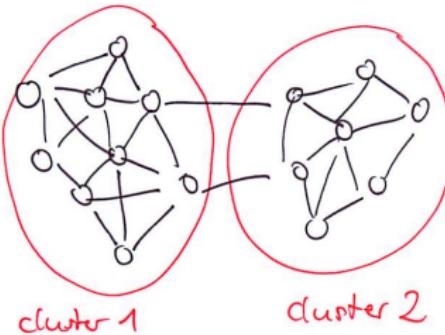
Literature:

- ▶ U. Luxburg. Tutorial on Spectral Clustering, Statistics and Computing, 2007.
- ▶ The later editions of Tibshirani/Hastie/Friedman also contain a chapter on it.

Clustering in graphs

General problem:

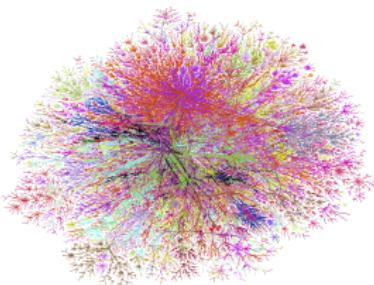
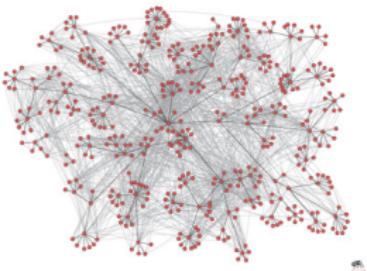
- ▶ Given a graph
- ▶ Want to find “clusters” in the graph:
 - ▶ many connections inside the cluster
 - ▶ few connections between different clusters



Clustering in graphs (2)

Examples:

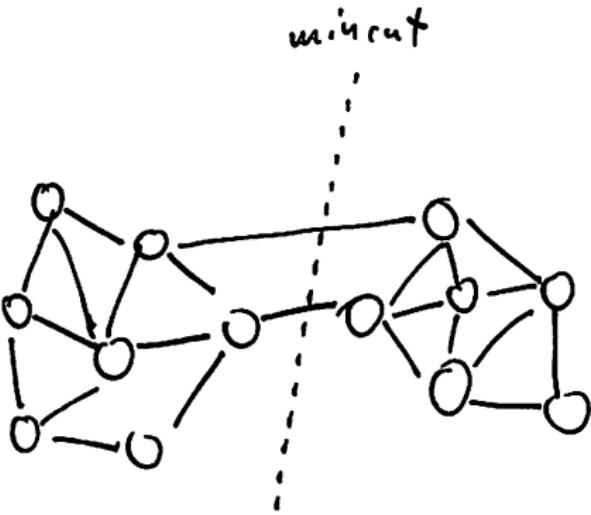
- ▶ Find “communities” in a social network (e.g., to analyze communication patterns in a company; to place targeted ads in facebook)
- ▶ Find groups of jointly acting proteins in a protein-interaction network
- ▶ Find groups of similar films (\leadsto “genres”)
- ▶ Find subgroups of diseases (for more specific medical treatment)



Clustering in graphs (3)

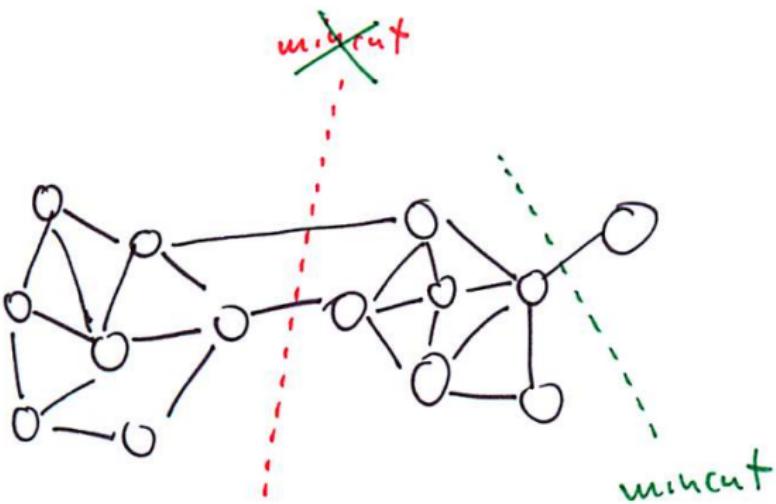
First idea:

- ▶ “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.

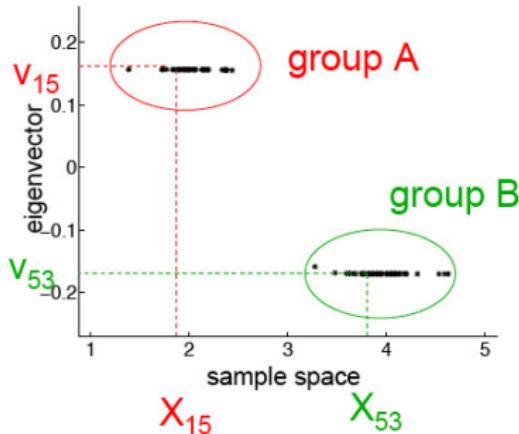
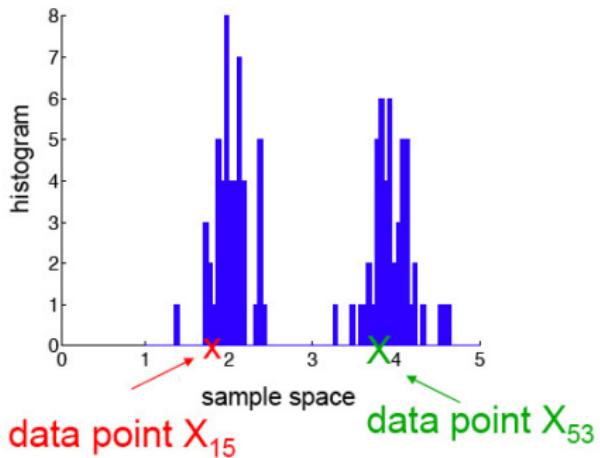
(Unnormalized) Spectral Clustering, case two clusters

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

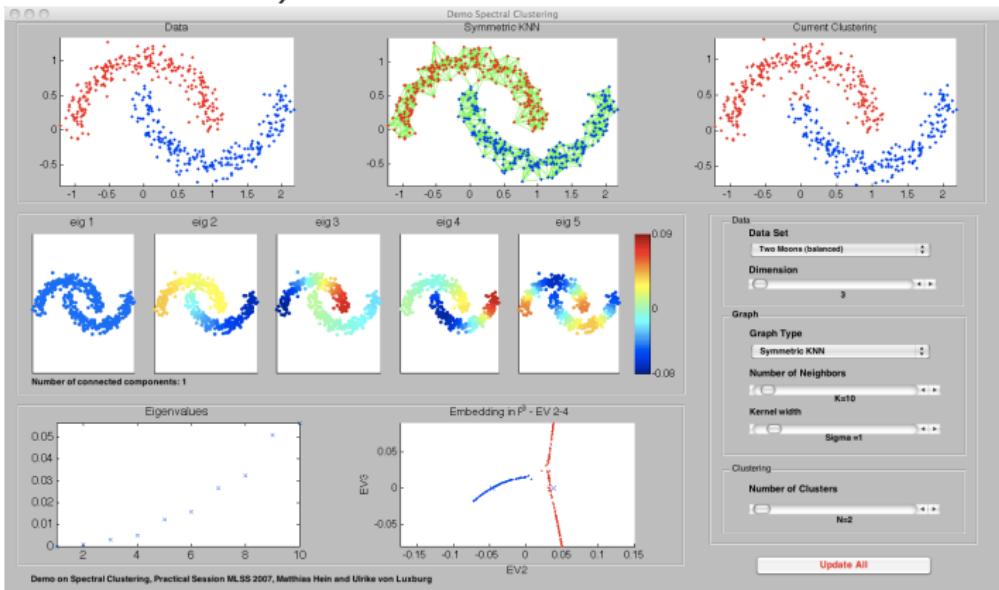
A couple of data points drawn from a mixture of Gaussians on \mathbb{R} .



$$\text{eigenvector } v = (-0.18, -0.18, 0.17, 0.18, \dots, \color{red}{0.17}, \dots -0.18, \dots)$$

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)

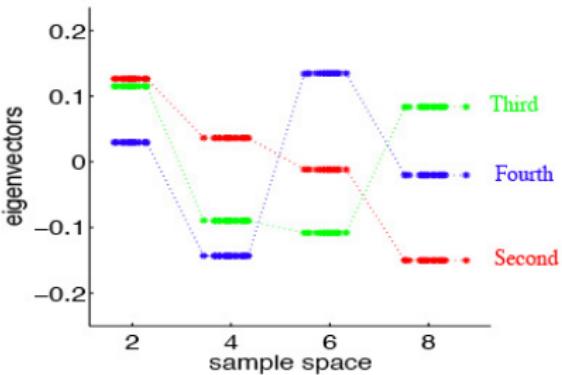
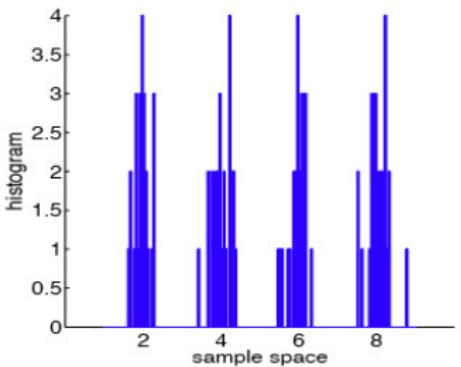
- Note: in the “ideal case” (disconnected clusters) the Y_i are the same for all points in the same cluster.

$$V = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 1 \end{pmatrix}}_{n \times 3} \quad \left. \begin{array}{l} \{ \quad (1 \ 0 \ 0) \\ \{ \quad (0 \ 1 \ 0) \\ \{ \quad (0 \ 0 \ 1) \end{array} \right.$$

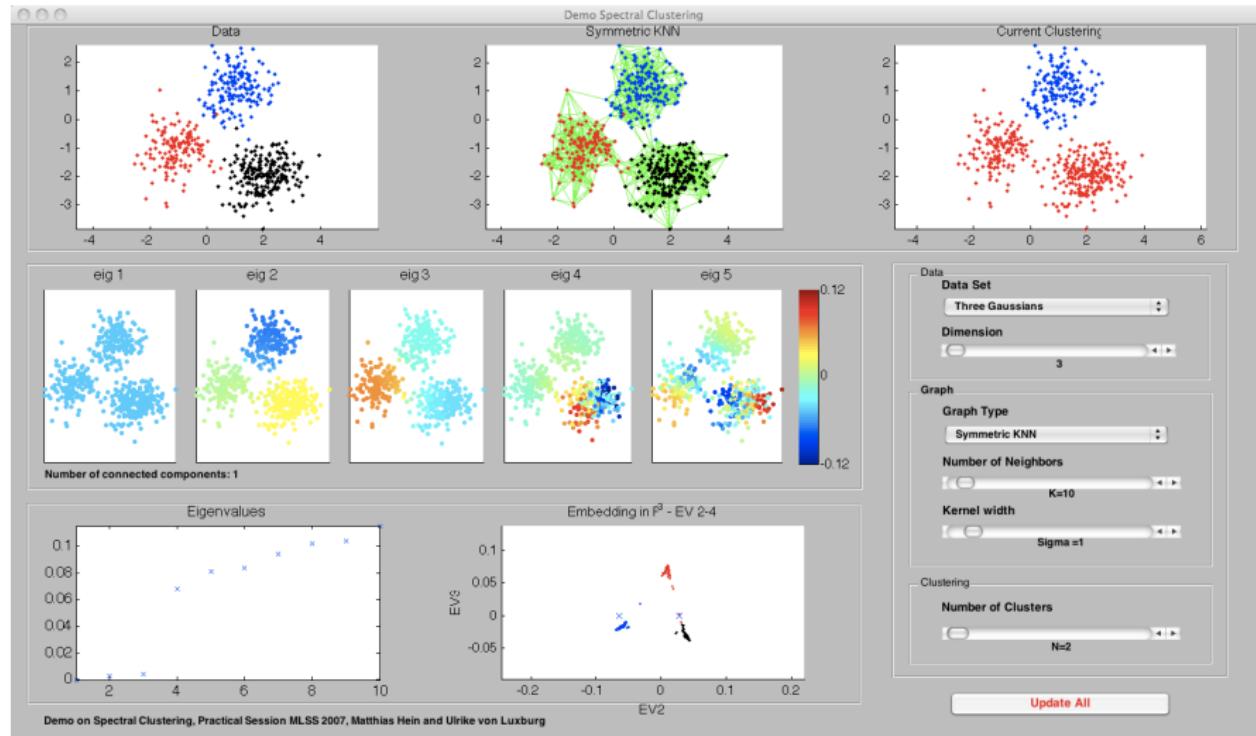
b_2

- 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.

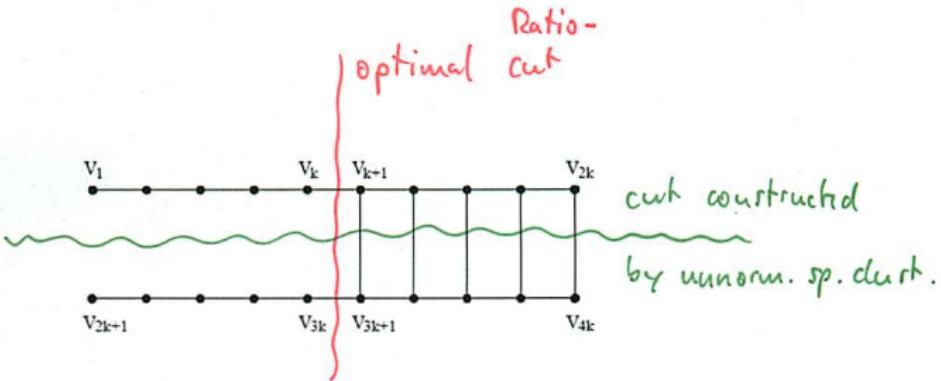
Analysis: Running time

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 graph Laplacian

To solve the problem of minimizing Ncut, we consider the normalized graph Laplacian

$$L_{norm} = D^{-1/2} L D^{-1/2}$$

(where the square root of the diagonal matrix D can be computed entry-wise).

The normalized Laplacian has properties that are very similar to the ones of the unnormalized Laplacian:

- ▶ It is symmetric, positive semi-definite
- ▶ The smallest eigenvalue is 0
- ▶ The number of connected components equals the multiplicity of the eigenvalue 0.

Minimizing normalized cut

NormalizedSpectralClustering(G)

- 1 Input: Weight matrix (or adjacency matrix) W of the graph
- 2 $D :=$ the corresponding degree matrix
- 3 $L_{norm} := D^{-1/2}(D - W)D^{-1/2}$ (the normalized graph Laplacian)
- 4 Compute the second-smallest eigenvector f of L_{norm}
- 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.

The data processing chain: from raw data to machine learning

Preparing the data

Data aquistion: 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)

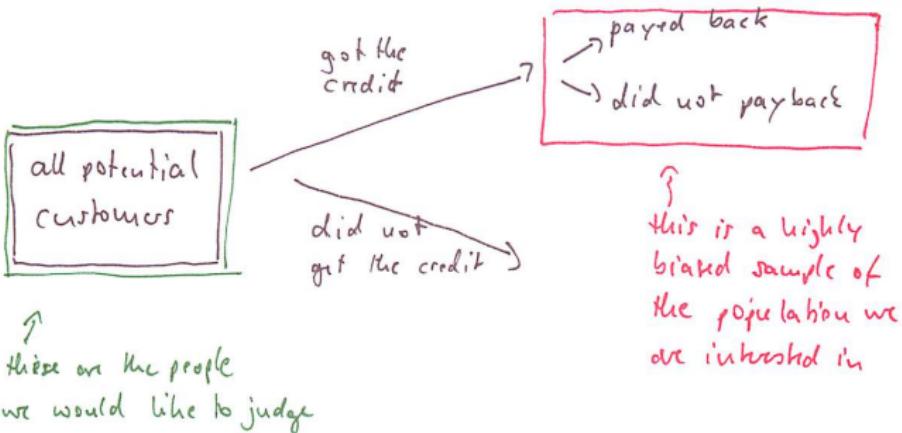
Is your training set “representative” for the test cases: is the distribution of your training inputs roughly the same as the distribution of your test inputs?

- ▶ 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 16×16 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 point 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 for.
 - ▶ If we throw outliers away, we might loose 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 do.

Treating missing values

- ▶ You have to decide what to do with missing values.
- ▶ Note that simply throwing away data points with missing values is not a good idea (values could miss systematically, and the fact that a value is missing might contain information)
- ▶ No standard recipe here ...

Defining features, similarities, distance functions

Defining features, similarities, distance functions

- ▶ 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.

Feature hashing

- ▶ Sometimes the feature vector is very sparse (e.g. in a bag of words approach).
- ▶ Then it might be useful to use a hash function to map the original feature vector to a condensed representation.
- ▶ This is called “feature hashing” .

Constructing features out of existing ones

Sometimes it is useful to even blow up the feature space by introducing conjunctions or disjunctions of features. See the slides about feature selection later

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

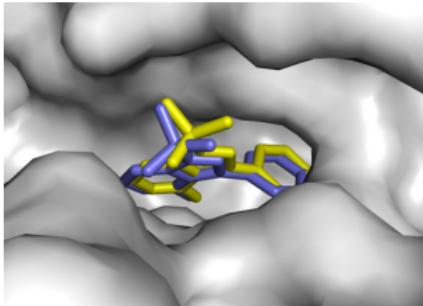
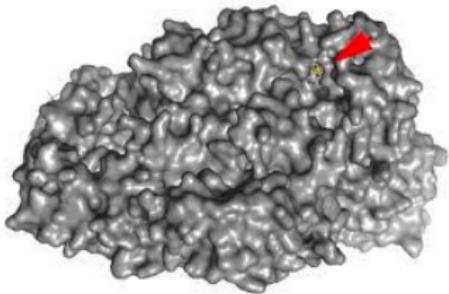
In a kernel approach (or distance or similarity based approach):

- ▶ 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:

- ▶ PCA. This is THE standard algorithm 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.

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 γ is a parameter. High γ 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 γ 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).

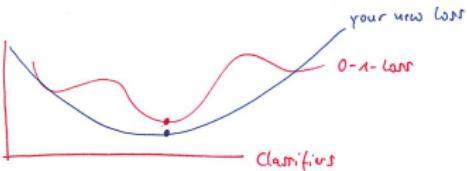
If you want to design such a surrogate loss function, here are some considerations you should take into account (this is not part of the standard ML processing chain, but I wanted to mention the keywords):

- ▶ 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.
- ▶ 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.

- ▶ We train each class against each other class, this then gives $K(K - 1)/2$ classifiers f_{kl} in the end.

Multi-class approaches (2)

- ▶ 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:

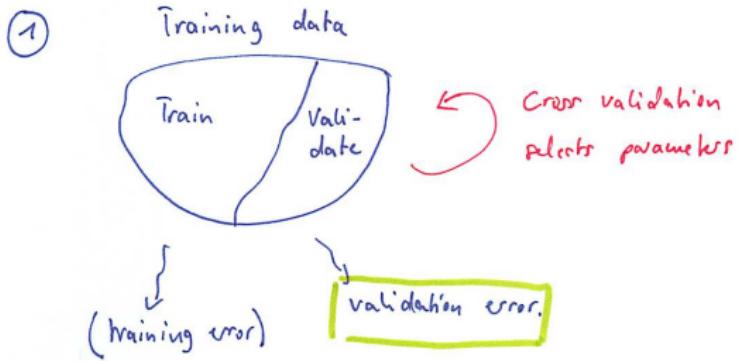
- ▶ Parameter C of the SVM
- ▶ Parameter σ in the Gaussian kernel
- ▶ Number of principle components in PCA
- ▶ But you also might want to figure out whether certain design choices make sense, for example whether it is useful to remove outliers in the beginning or not.

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

K-fold cross validation

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

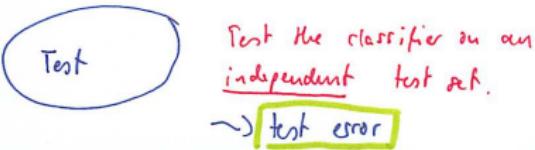
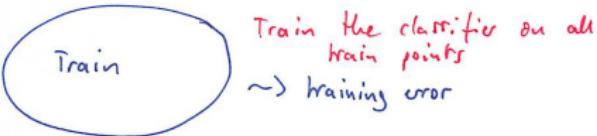
K-fold cross validation (2)



K-fold cross validation (3)

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

- ▶ 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 are not 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

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 loose (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 one, 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).
- ▶ 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

An great overview paper:

Guyon, Elisseeff: Introduction to variable and feature selection.
JMLR, 2003.

A whole edited book:

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
true | positive | negative | |
|------|----------|------------------------------|----------------------|---------------------------------|--|
| true | positive | true positive
tp | false negative
fn | $\Sigma =: P$ positive examples | |
| | negative | false positive
fp | true negatives
tn | | |

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
- ▶ False positive rate: fp / P
- ▶ True negative rate (**specificity**): tn / N
- ▶ False negative rate: fn / N

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):

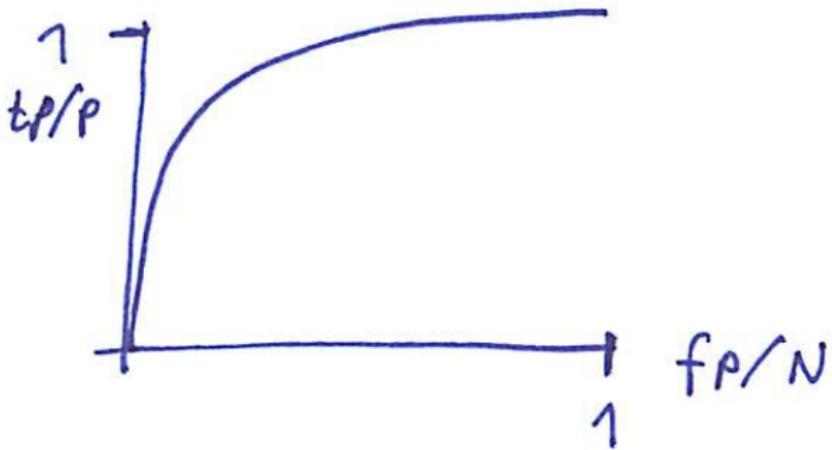
- ▶ **Recall:** tp/P (how many positive examples can we find)
- ▶ **Precision:** $tp/(tp + fp)$ how many of all positively classified examples are indeed correct

ROC and AUC

ROC (=Receiver-operator characteristic) curve:

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

ROC and AUC (2)

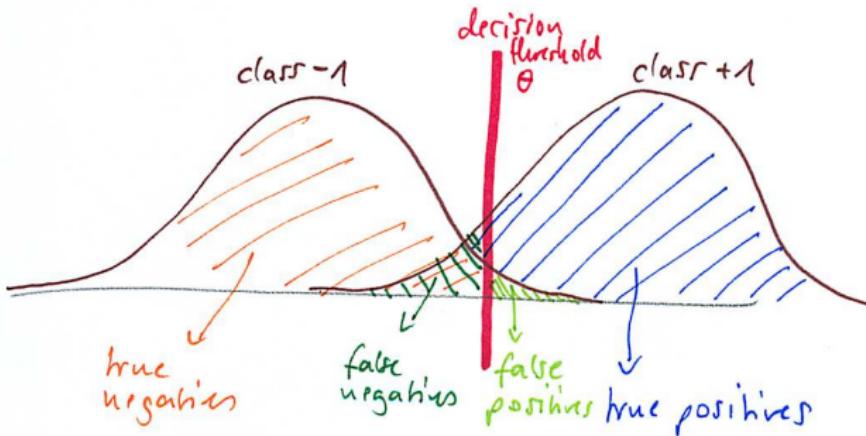


Note: indeed,

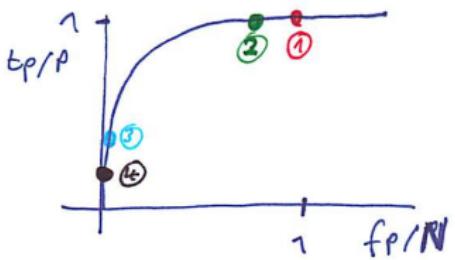
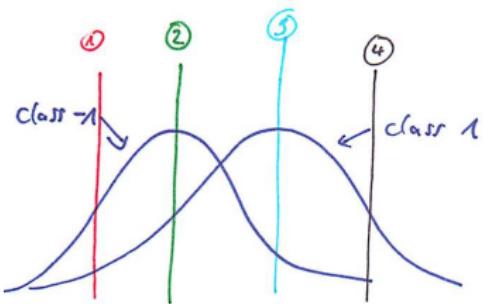
- ▶ $tp/P \in [0, 1]$
- ▶ $fp/N \in [0, 1]$

ROC and AUC (3)

Intuition with normal distributions:

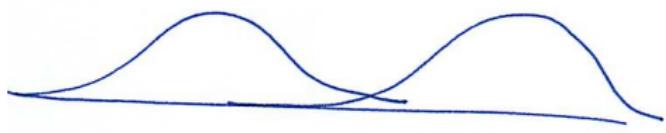


ROC and AUC (4)

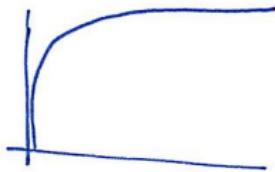


ROC and AUC (5)

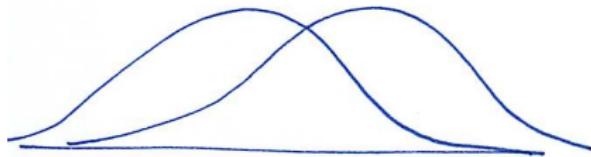
"good" Data



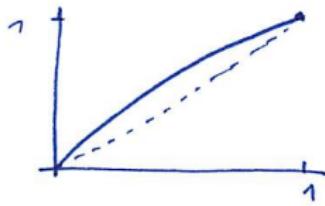
ROC



"bad" Data



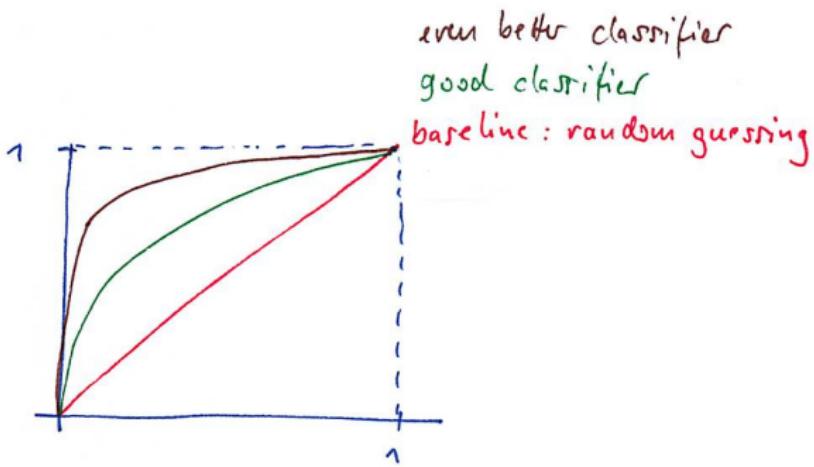
ROC



ROC and AUC (6)

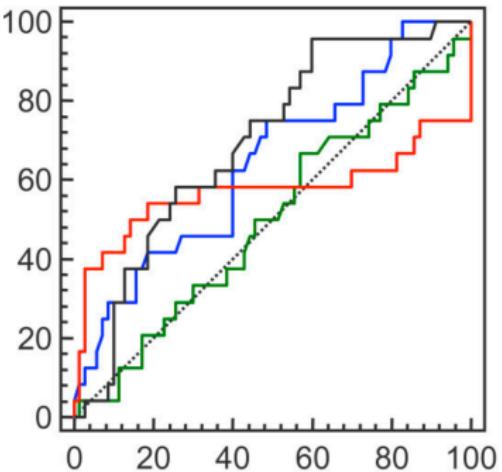
ROC for comparing classifiers:

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



ROC and AUC (7)

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

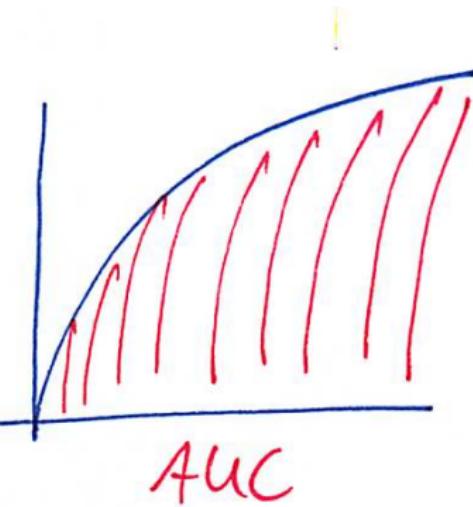


- ▶ In this case, you might still be able to say in what parameter range one classifier is better than the other.
- ▶ Or you might want to use AUC.

ROC and AUC (8)

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 (9)

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

Comparing many classifiers (2)

Table 1: Average classification error rates.

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

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 t-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 plot a histogram of the Bayes errors in the data sets we use, it usually won't 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 work goes into fine-tuning a classifier).
- ▶ Most people won’t believe you if you say that your classifier “consistently outperforms” the other classifiers.

Some critical remarks (4)

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

Some critical remarks (5)

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

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

Some 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).
- ▶ A lot of potential is released in 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)
 - ▶ Usually, such challenges end with a workshop where the best teams discuss their approaches. In particular, they often corroborate / discourage the use of certain approaches.

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, you might not need many of these things.
- ▶ 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:

- ▶ Standardize your data
- ▶ If your data is high-dimensional, try unsupervised dimensionality reduction (PCA) and supervised feature selection.
- ▶ Always use cross-validation to set your parameters / decide on design principles.
- ▶ Consider to incorporate weights in your loss function if your classes are unbalanced.

High-level guidelines and principles

High-level guidelines

Try not to lose 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!

Wrap up

Things we did not talk about

Things that are important but we did not cover

- ▶ Many important algorithms, of course
- ▶ Learning Theory: theoretical justification for why all the algorithms really make sense.
- ▶ Probabilistic approaches (Graphical models, Bayesian learning, Gaussian processes, etc).
- ▶ Bio-inspired approaches (neural networks, genetic algorithms, etc).

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.
- ▶ Nearly all of them get videotaped and are available at videolectures.net.

Videolectures:

- ▶ Most machine learning conferences, workshops, summer schools, etc are videotaped. The videos are online, many of them at videolectures.net.
- ▶ 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:

- ▶ Neural Information Processing Systems (NIPS)
- ▶ International Conference on Machine Learning (ICML)
- ▶ International Conference on Learning Theory (COLT)

The top journals:

- ▶ Journal of machine learning research (JMLR) the flagship journal
- ▶ Machine Learning Journal (MLJ), good as well).

Future lectures in Hamburg

My next classes on ML ...

- ▶ Winter term: lecture on learning theory (2 SWS)
 - ▶ Generalization bounds and how they can be proved.
 - ▶ Tools like concentration inequalities, covering and shattering numbers, VC dimension.
 - ▶ Consistency of important classification algorithms and principles like k -nearest neighbor rules, SVM, regularization principle.
 - ▶ Geometric random graphs, their properties, and the statistical analysis of learning algorithms on random graphs.
- ▶ Always: Oberseminar. Here we mainly read current ML papers or have talks by guests. Everybody welcome.

Reinforcement learning: taught by Norman Hendrich

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

- ▶ $\Omega = \text{space of "elementary events", "sample space"}$.
This space is called "discrete" if it has finitely many elements.
- ▶ "**Space of events**": In the discrete case this is simply the power set $\mathcal{P}(\Omega)$ of Ω , that is all possible subsets of Ω .
(In general it is more complicated, the space of events has to be a " σ -algebra").
- ▶ **Probability measure**: $P : \mathcal{P}(\Omega) \rightarrow [0, 1]$ such that the following three rules are satisfied ("Axioms of 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 Ω can be computed based on the elementary events due to the sigma-additivity.

Example: $P(1, 2, 5) = P(1) + P(2) + P(5) = 3 \cdot 1/6 = 1/2.$

Conditional probabilities

Define the probability of event A under the condition that event B has taken place:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Example with a die: compute the probability $P(\{3\} \mid \text{"uneven"})$.

Solution:

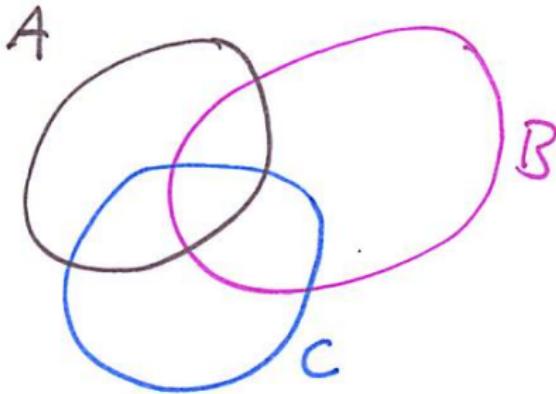
$A = \{3\}$, $B = \{1, 3, 5\}$, $P(A \cap B) = P(\{3\}) = 1/6$, $P(B) = 1/2$,
this implies $P(\{3\} \mid \text{"uneven"}) = (1/6)/(1/2) = 1/3$.

Important formulas

- **Union bound.** Let A_1, \dots, A_k be any events. Then

$$P(A_1 \cup A_2 \cup \dots \cup A_k) \leq \sum_{i=1}^k P(A_i)$$

Intuitive reason:



Important formulas (2)

- ▶ **Formula of total probability.** Let B_1, \dots, B_k be a disjoint decomposition of the probability space, that is all B_i are disjoint and $B_1 \cup \dots \cup B_k = \Omega$. Then:

$$P(A) = \sum_{i=1}^k P(A \cap B_i) = \sum_{i=1}^k P(A \mid B_i)P(B_i)$$

Important formulas (3)

- Bayes' formula:

$$P(B \mid A) = \frac{P(B \cap A)}{P(A)} = \frac{P(A \mid B) \cdot P(B)}{P(A)}$$

Example:

The probability that a woman has breast cancer is 1%. The probability that the disease is detected by a mammography is 80 % (true positive rate). The probability that the test detects the disease although the patient does not have it is 9.6% (false positive rate). If a woman at age 40 is tested as positive, what is the probability that she indeed has breast cancer?

Important formulas (4)

Define the following events:

$A :=$ mammography is positive

$B :=$ woman has breast cancer

Given:

- ▶ $P(B) = 0.01$
- ▶ $P(A \mid B) = 0.80$
- ▶ $P(A \mid \neg B) = 0.096$
- ▶ 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 distribution

- ▶ **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 distribution (2)

- ▶ Poisson distribution $Pois(\lambda)$.

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

The Poisson distribution counts the occurrence of “rare events” in a fixed time interval (like radioactive decay), λ is the intensity parameter.

It has expected value λ and variance λ .

Independence

Two events A, B are called independent if
 $P(A \cap B) = P(A) \cdot P(B)$.

Note that this implies that $P(A \mid B) = P(A)$.

Two random variables $X, Y : \Omega \rightarrow \mathbb{R}$ are called independent if for all events A, B we have that

$$P(X \in A, Y \in B) = P(X \in A) \cdot P(Y \in B).$$

Example:

- ▶ Throw a coin twice. X = result of the first toss, Y = result of the second toss. These two random variables are independent.
- ▶ 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

For a discrete random variable with possible values r_1, \dots, r_n , its variance is defined as

$$\begin{aligned}\text{Var}(X) &= \sum_{i=1}^n (r_i - E(X))^2 \cdot P(X = r_i) \\ &= E((X - E(X))^2)\end{aligned}$$

The variance measures how much the random variable “varies” about its mean.

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: $E(X) = p$.

Variance (2)

- 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:

- 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:

$$\text{std}(X) = \sqrt{\text{Var}(X)}$$

(*) 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:**

$$P(|X - E(X)| \geq t) \leq \frac{\text{Var}(X)}{t^2}$$

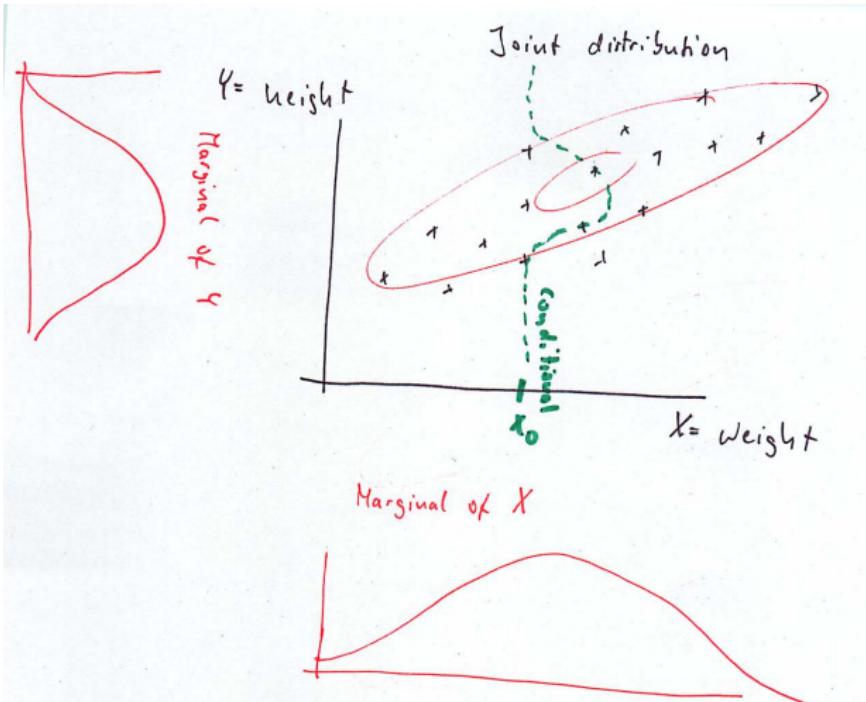
Joint, marginal and product distribution

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** $X \in \mathbb{R}$:

$$\text{Var}(X) = E((X - E(X))^2)$$

$$\text{Cov}(X, Y) = E\left((X - E(X))(Y - E(Y))\right)$$

The 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)

$$\hat{\text{Cov}}(X) = 1/n \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

Note that for variance and covariance, one sometimes normalized 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: $x_i = (x_i^{(1)}, \dots, x_i^{(d)})'$. Define the $n \times d$ -matrix X which contains the points x_i as rows. The overall variance over all d dimensions is the sum of the variances of the individual dimensions:

$$\text{Var}_d(X) = E(\|X - E(X)\|^2)$$

The covariance matrix of X is a $d \times d$ -matrix C which encodes the covariances between the individual dimensions of the distribution:

$$C_{kl} = \text{Cov}(X^{(k)}, X^{(l)})$$

EXAMPLE: SHOE SIZE / HEIGHT / AGE OF A PERSON

Variance and Covariance of Multivariate random variables (4)

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.
- ▶ If the data points are centered, and we define matrix X containing the points as rows, then the empirical covariance matrix \hat{C} coincides with $X'X$.
- ▶ In the following, we are going to drop the “hat” and the words “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)

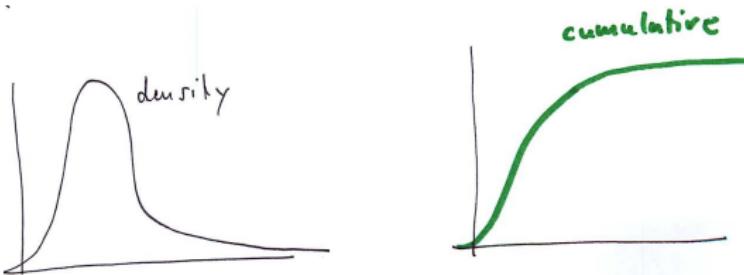
- ▶ Intuitively, a density is something like a “continuous histogram”.
- ▶ Sometimes the density is abbreviated as “pdf” (“probability density function”) in the literature.
- ▶ Density functions are always non-negative and integrate to 1. They don't have to be continuous.
- ▶ Not every random variable 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

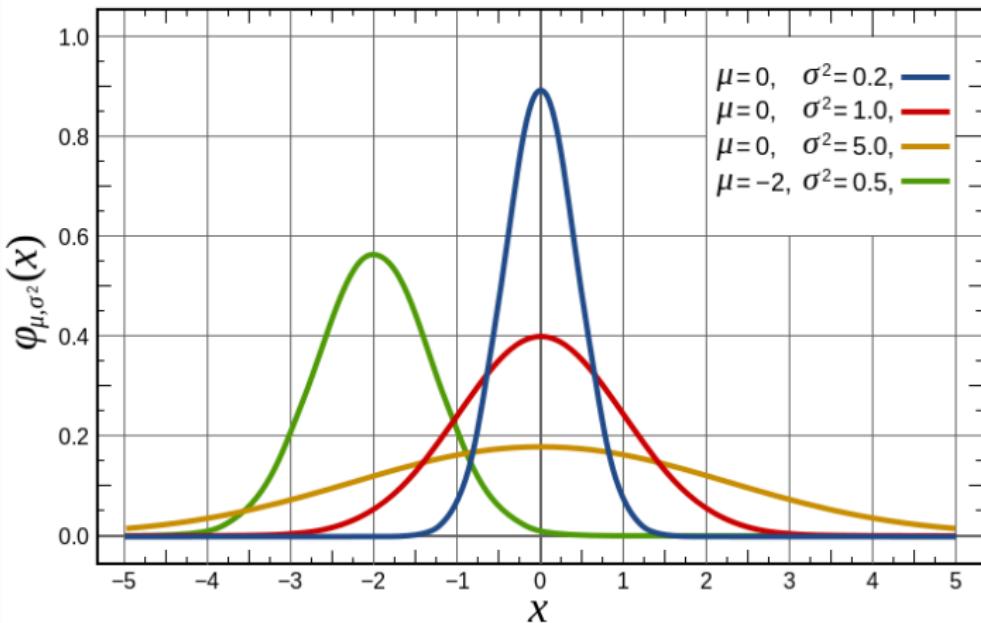
The most important continuous distribution is the normal distribution, abbreviated $N(\mu, \sigma^2)$.

- ▶ It has two parameters: its expectation μ and its variance σ^2 .
- ▶ The density function of $N(\mu, \sigma^2)$ is given as

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

- ▶ The special case of mean 0 and variance 1 is called the “standard normal distribution”.

Normal distribution (2)



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:

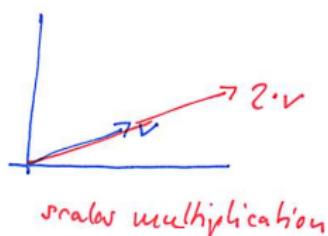
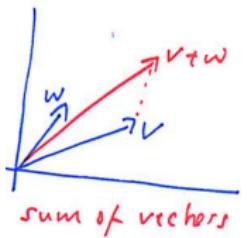
- ▶ We can add and subtract vectors: For $v, w \in V$ we can build $v + w, v - w$
- ▶ We can multiply vectors with scalars: For $v \in V, a \in \mathbb{R}$ we can build av .
- ▶ These operations satisfy all kinds of formal requirements (associativity, commutativity, identity element, inverse element, 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:

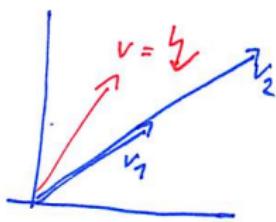
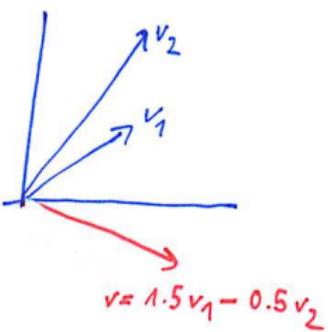
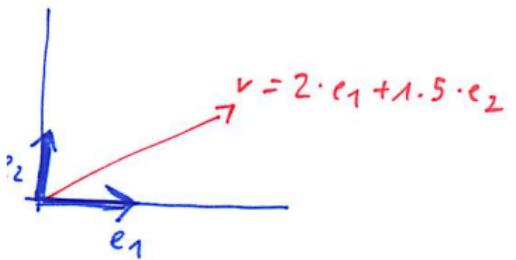
$$\sum_{i=1}^d a_i b_i = 0 \implies a_i = 0 \text{ for all } i = 1, \dots, d.$$

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 .



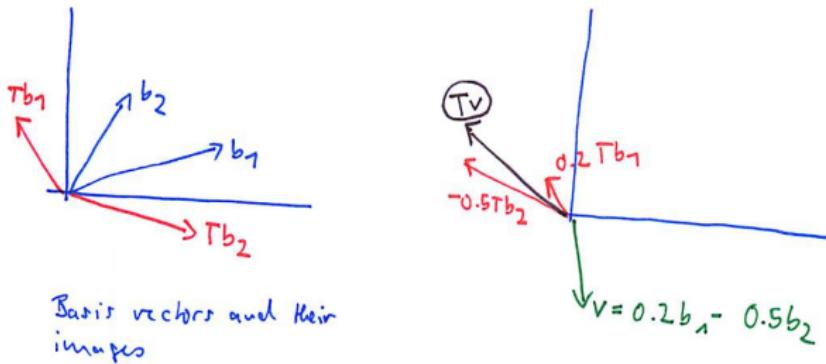
Linear mappings

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 $T(v) = T(\sum_i a_i b_i) = \sum_i a_i T(b_i)$



Matrices

$m \times n$ -matrix A:

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{m1} & & a_{mn} \end{pmatrix}$$

Matrices (2)

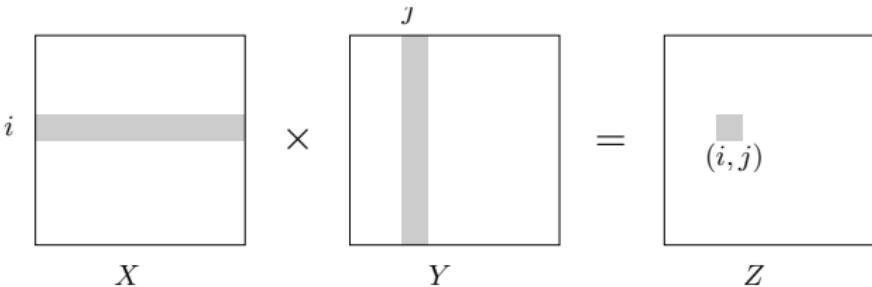
Transpose of a matrix , written as A^t or A' is the matrix where we exchange rows with columns (that is, instead of a_{ij} we have a_{ji}).

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)

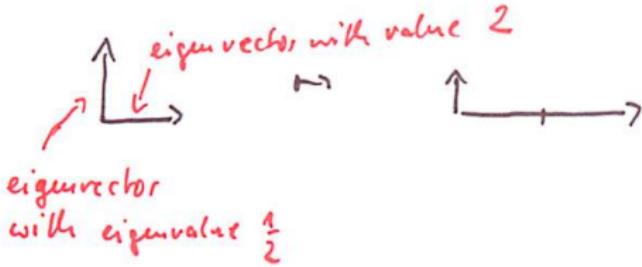
A matrix A is called **orthogonal** if all its columns are orthogonal to each other. It is called **orthonormal** if additionally, all its columns have norm 1.

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 λ if $Av = \lambda v$.

Intuition: in the direction of v , the linear mapping corresponding to A is stretching by factor λ .



Taken together, all eigenvectors with eigenvalue λ form a subspace called the **eigenspace** associated to eigenvalue λ . The dimension of this subspace is called the **geometric multiplicity** of λ .

Eigenvalues and eigenvectors (2)

Just for completeness:

Eigenvectors can also be defined as the roots of the **characteristic polynomial** $f(\lambda) := \det(A - \lambda I) \stackrel{!}{=} 0$. The degree of this polynomial is d (the dimension of the space). The multiplicity of this root is called the **algebraic multiplicity of λ** .

We always have that the algebraic multiplicity is 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.

- ▶ Intuitively, a matrix is diagonalizable if it performs “Strecken und Spiegeln”, but no rotation.

Symmetric matrices are always diagonalizable and have real-valued eigenvalues.

Eigenvectors of different eigenvalues are always perpendicular to each other (only true for symmetric matrices).

Positive Definite Matrices

A symmetric matrix A is called **positive semi-definite** if all its eigenvalues are ≥ 0 . In case of strict inequality it is called **positive definite**.

Being positive definite is equivalent to the condition that $v^t A v > 0$ for all $v \in \mathbb{R}^n \setminus \{0\}$.

(similarly for positive semi-definite with \geq)

(*) 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, $\Sigma \in \mathbb{R}^{n \times d}$ is a diagonal matrix containing the singular values $\sigma_1, \dots, \sigma_d$ on the diagonal, and $V \in \mathbb{R}^{d \times d}$ is an orthogonal matrix.

The columns of U (and V , respectively) are called the left (and right) singular vectors, the entries of Σ are called the singular values.

(*) Singular Value Decomposition (SVD) (2)

There is a close relation between the singular values of Φ and the eigenvalues of the (symmetric!) matrices $\Phi\Phi^t$ and $\Phi^t\Phi$:

- ▶ The left singular vectors of Φ are the eigenvectors of $\Phi\Phi^t$.
- ▶ The right singular vectors of Φ are the eigenvectors of $\Phi^t\Phi$.
- ▶ The non-zero singular values of Φ are the square roots of the non-zero eigenvalues of both $\Phi^t\Phi$ and $\Phi\Phi^t$.

(*) Generalized inverse

Consider a symmetric matrix $A \in \mathbb{R}^{d \times d}$.

- ▶ Let $\lambda_1, \dots, \lambda_d$ the eigenvalues and v_1, \dots, v_d a corresponding set of eigenvectors of A . We can write A in the spectral decomposition as

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

- ▶ In case the matrix has rank d , all its eigenvalues are non-zero. Then we can write the inverse of A as

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

(*) Generalized inverse (2)

- ▶ In case the matrix is not of full rank, it is not invertible.
However, we can define the **Moore-Penrose generalized inverse** as

$$A^+ := \sum_{i:\lambda_i \neq 0} \frac{1}{\lambda_i} v_i v_i^t$$

(intuitively, this is the inverse of the matrix A restricted to the subspace orthogonal to its nullspace).

(*) Generalized inverse (3)

Properties of the generalized inverse:

In general we don't have that $AA^+ = I$ or $A^+A = I$.

But we have the following slightly weaker properties:

- ▶ $AA^+A = A$ and $A^+AA^+ = A^+$
- ▶ $(A^+)^+ = A$
- ▶ A^+A and AA^+ are both symmetric.
- ▶ If A is invertible, then $A^{-1} = A^+$.
- ▶ AA^+ is an orthogonal projection on the $\text{ran}(A)$ (the image of the matrix A), and A^+A is an orthogonal projection on $\text{ran}(A^t)$.

(*) Rayleigh principle

Proposition 35 (Rayleigh principle)

Let $A \in R^{n \times n}$ be a symmetric matrix with eigenvalues $\lambda_1 \geq \dots \geq \lambda_n$ and eigenvectors v_1, \dots, v_n . Then

$$\lambda_1 = \max_{v \in \mathbb{R}^n} \frac{v^t A v}{\|v\|^2} = \max_{v \in \mathbb{R}^n : \|v\|=1} v^t A v.$$

The eigenvector v_1 is the vector for which this maximum is attained. Moreover,

$$\lambda_{k+1} = \max_{v \perp \{v_1, \dots, v_k\} : \|v\|=1} v^t A v.$$

This theorem holds analogously for minimization problems. In this case, the solution is given by the smallest eigenvalue / vector.

(*) Rayleigh principle (2)

Proof intuition.

- ▶ Let λ be any eigenvalue with eigenvector v . Then $v^t A v = v^t (\lambda v) = \lambda$ (because $v^t v = 1$).
- ▶ So among all eigenvectors v_1, \dots, v_n , the eigenvector v_1 leads to the largest value λ_1 .
- ▶ Now consider an arbitrary unit vector $w \in \mathbb{R}^n$. Because A is symmetric, there exists a basis of eigenvectors v_1, \dots, v_n . In particular, there exist coefficients c_i such that $w = \sum_i c_i v_i$ and $\|c\| = 1$.
- ▶ Then $w^t A w = \dots = \sum_{i,j=1}^n c_i c_j v_i^t A v_j$
- ▶ But for $i \neq j$ we get $v_i^t A v_j = v_i^t \lambda v_j = 0$ (because different eigenvectors are perpendicular to each other).
- ▶ so $w^t A w = \sum_i c_i^2 v_i^t A v_i = \sum_i c_i^2 \lambda_i$.

(*) Rayleigh principle (3)

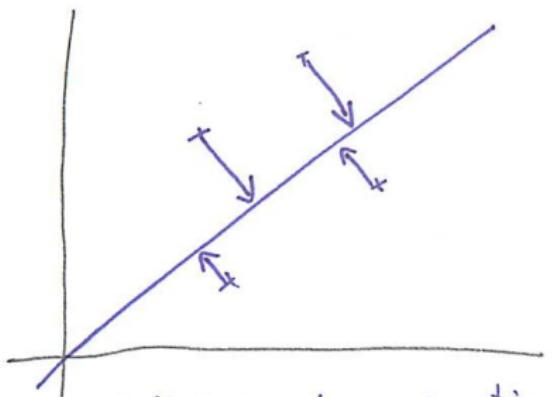
- ▶ Among all c with $\|c\| = 1$, the maximum of this expression is attained for $c_1 = 1, c_2 = \dots = c_n = 0$.



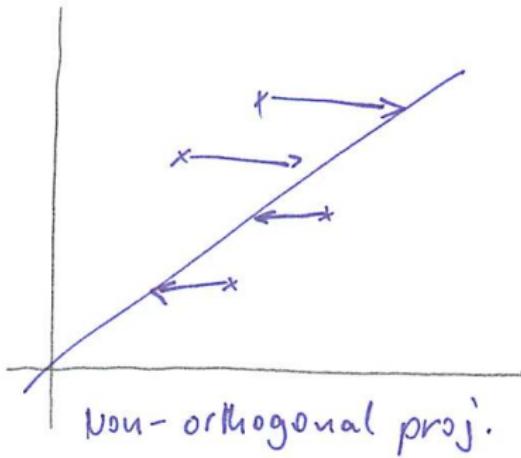
Projections

A linear mapping $\pi : E \rightarrow F$ 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 $\text{nullspace}(\pi) \perp \text{image}(\pi)$.



orthogonal projection



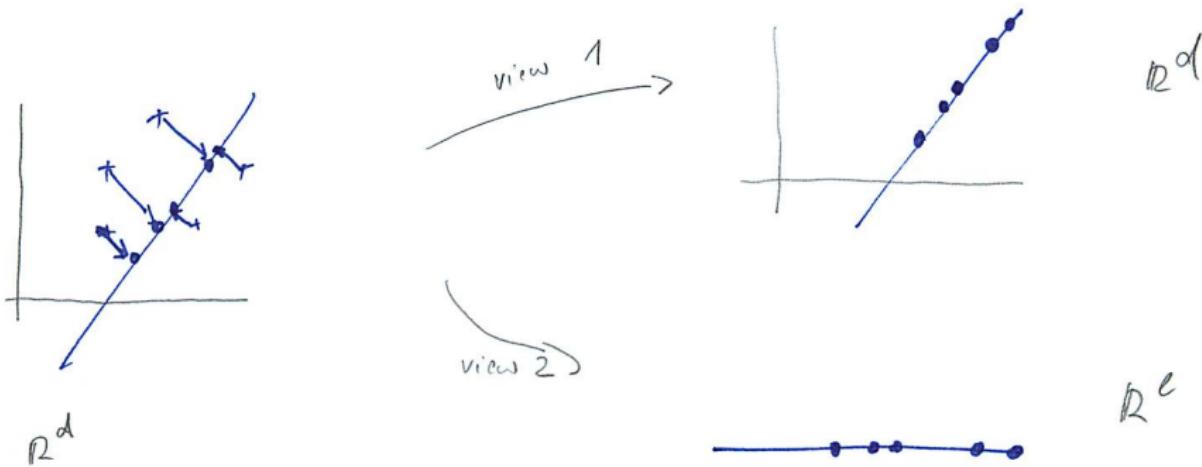
non-orthogonal proj.

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 $\pi : \mathbb{R}^d \rightarrow \mathbb{R}^d$.

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 ℓ -dimensional subspace: Want to project on an ℓ -dim subspace S with ONB v_1, \dots, v_ℓ .

View 2: Define the matrix V with the vectors v_1, \dots, v_ℓ as columns. Then compute the low-dim representation as

$$\pi : \mathbb{R}^d \rightarrow \mathbb{R}^\ell, x \mapsto V'x$$

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 $Tx = P(x - \mu) + \mu$.

