

# Lecture 1: Overview

April 25, 2019

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# Lecture Overview

- 1 Motivation: Why Study Machine Learning?
- 2 Organizational Issues
- 3 Introduction to Supervised Machine Learning
- 4 Other Types of Machine Learning
- 5 The Machine Learning Design Cycle
- 6 Wrapup: Summary, Other Courses, Resources, Preview

# Lecture Overview

- 1 Motivation: Why Study Machine Learning?
- 2 Organizational Issues
- 3 Introduction to Supervised Machine Learning
- 4 Other Types of Machine Learning
- 5 The Machine Learning Design Cycle
- 6 Wrapup: Summary, Other Courses, Resources, Preview

# Where are you already using Machine Learning?

- You're probably using dozens of systems built on machine learning every day ...
- Let's hear some examples from you to get started! 

# Machine Learning is Everywhere

- Spam classification

# Machine Learning is Everywhere

- Spam classification
- Ranking web pages & image search: Google / Bing

# Machine Learning is Everywhere

- Spam classification
- Ranking web pages & image search: Google / Bing
- Face recognition: Facebook, smart phones

# Machine Learning is Everywhere

- Spam classification
- Ranking web pages & image search: Google / Bing
- Face recognition: Facebook, smart phones
- Speech recognition (Siri on the iPhone / Google talk on Android)

# Machine Learning is Everywhere

- Spam classification
- Ranking web pages & image search: Google / Bing
- Face recognition: Facebook, smart phones
- Speech recognition (Siri on the iPhone / Google talk on Android)
- Machine translation (Google translate, Skype auto-translate)

# Machine Learning is Everywhere

- Spam classification
- Ranking web pages & image search: Google / Bing
- Face recognition: Facebook, smart phones
- Speech recognition (Siri on the iPhone / Google talk on Android)
- Machine translation (Google translate, Skype auto-translate)
- Product/song/video recommendations in online stores

# Machine Learning is Everywhere

- Spam classification
- Ranking web pages & image search: Google / Bing
- Face recognition: Facebook, smart phones
- Speech recognition (Siri on the iPhone / Google talk on Android)
- Machine translation (Google translate, Skype auto-translate)
- Product/song/video recommendations in online stores
- Computer game industry: self-configuration based on user actions

# Machine Learning is Everywhere

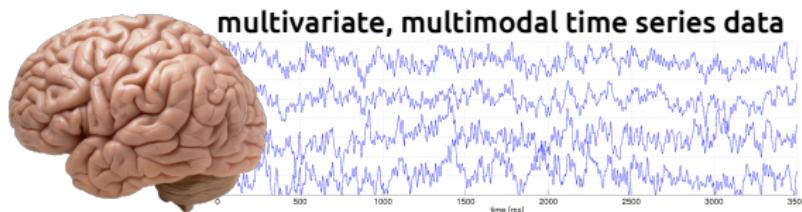
As of now, slightly less common examples:

- Autonomous cars
- Intelligent prostheses (BrainLinks-BrainTools)
- Autonomous helicopter flight
- Learning soccer robots (RoboCup)
- Learning to Design RNA

# Success Story: Decoding of Brain Activity

A grand challenge! Brain signals ...

- are high-dimensional and noisy
- are subject specific
- have distributions that vary over time (fatigue, learning, strategies, ...)



Traditional way to do research:

- do the same experiment for many subjects
- average all the data
- do statistics and publish

**Desired: single-trial decoding to run online applications!**

# Estimate Movie Content from Brain Activity



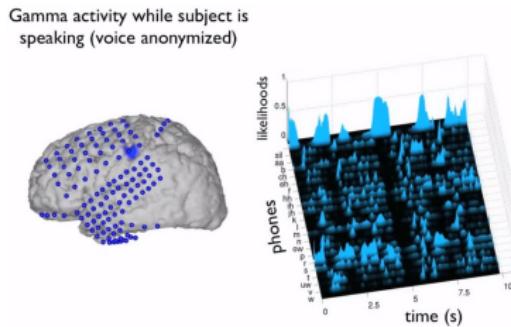
Can you guess a movie's content from brain signals of the observer?

- Show subjects several hours of movies while recording brain signals (fMRI data).
- Train a movie-to-brain activity encoding model (regression).
- Show novel movies and record brain signals.
- Estimate the visual content with the trained model.

[Nishimoto et al., 2011, Current Biology, video from

<https://www.youtube.com/watch?v=nsjDnYxJ0bo>]

# Estimate Spoken Speech from Brain Activity



## Reconstruct spoken speech from brain signals:

- Semi-invasive recordings
  - Train a ECoG-to-phonemes decoder
  - Use speech recognition technique + language models

[Herff et al., Front. Neurosci. 2015, <http://dx.doi.org/10.3389/fnins.2015.00217>]

# Example from our work: Estimate Mental Workload

Using BCI Technology to monitor brain processes in **real time**.

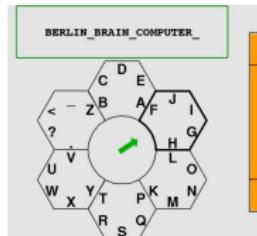
Driver's workload during:

- Driving task only (primary task)
- Driving + audio book listening
- Driving + mental calculation



# Example from our work: Controlling a Brain-Computer Interface

Decode motor imagery classes (left hand, right hand, feet) from EEG signals to control a spelling application:



Today's BCI applications address:

- Communication for motor-impaired patients
- Device control
- Rehabilitation after stroke
- Mental state monitoring (workload, attention, ...)

# Machine Learning and Data Mining

- Many fields have **big data** to be mined
  - Medicine, biology, astronomy, finance, social network studies, . . .

# Machine Learning and Data Mining

- Many fields have **big data** to be mined
  - Medicine, biology, astronomy, finance, social network studies, ...
- We'd like to make inferences from the data
  - “We're drowning in data but starving for knowledge”  
(Futurist John Naisbitt)
  - Machine Learning lets us **generalize** (which is key to knowledge)

# Machine Learning and Artificial Intelligence

- Machine learning is a subfield of AI

# Machine Learning and Artificial Intelligence

- Machine learning is a subfield of AI
- AI used to be focused on **logics**
  - Theorem proving
  - Classical planning

# Machine Learning and Artificial Intelligence

- Machine learning is a subfield of AI
- AI used to be focused on **logics**
  - Theorem proving
  - Classical planning
- Since the 1990s, AI focuses more on **probabilistic reasoning**
  - This is closer to machine learning

# Machine Learning and Artificial Intelligence

- Machine learning is a subfield of AI
- AI used to be focused on **logics**
  - Theorem proving
  - Classical planning
- Since the 1990s, AI focuses more on **probabilistic reasoning**
  - This is closer to machine learning
- One approach to solve AI is to build machine learning systems that **resemble the human brain**
  - Richard Feynman: “what we can’t synthesize, we don’t understand”
  - Deep neural networks are a (small) step in this direction

# Machine Learning as A Different Way of Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Playing Atari games
  - Speech recognition
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience

# Machine Learning as A Different Way of Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Playing Atari games
  - Speech recognition
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience
- If the task changes, we simply re-train

# Machine Learning as A Different Way of Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Playing Atari games
  - Speech recognition
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience
- If the task changes, we simply re-train
- We can construct computer systems that are too complex for us to understand anymore ourselves...
  - E.g., deep neural networks have millions of weights.
  - E.g., AlphaGo, the system that beat world champion Lee Sedol
    - + David Silver, lead author of AlphaGo cannot explain its moves
    - + Paraphrased: "You would have to ask a Go expert."

# Machine Learning is very Sought After

- AI is very popular
  - When university students in a wide range of fields were asked to list a field they would like to study other than their own, the most frequent answer was AI
  - Machine learning is one of the most exciting subfields of AI

# Machine Learning is very Sought After

- AI is very popular
  - When university students in a wide range of fields were asked to list a field they would like to study other than their own, the most frequent answer was AI
  - Machine learning is one of the most exciting subfields of AI
- Predictions by a 2011 McKinsey report (for the US):
  - Better data analysis could save the public sector 250 billion Euro/year
  - There will be a need for 140,000 to 190,000 machine learning experts
  - There will be a need for 1.5 million data-savvy managers

# Lecture Overview

- 1 Motivation: Why Study Machine Learning?
- 2 Organizational Issues
- 3 Introduction to Supervised Machine Learning
- 4 Other Types of Machine Learning
- 5 The Machine Learning Design Cycle
- 6 Wrapup: Summary, Other Courses, Resources, Preview

# Organization: Instructors

## **Michael Tangermann**

Research group leader  
Brain State Decoding  
*Main Point of Contact*



## **Marius Lindauer**

Junior Research Group Leader  
Machine Learning



## **Frank Hutter**

Professor  
Machine Learning



# Organization: Assistants and Tutors



Sebastián  
Castaño



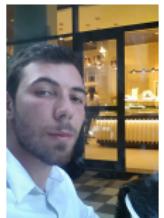
David  
Hübner



Jan  
Sosulski



Andreas  
Meinel



Arber  
Zela



Katharina  
Eggensperger



Simon  
Ging



Yi-Chun  
Lin



Baohe  
Zhang

# Organization: Lectures

- Meeting times:  
Monday & Thursday 16:15-17:52 (S101, room 00 026)
  - Including a 7-minute break roughly in the middle
- Both Monday and Thursday slots might have lecture content and exercise discussion
  - We highly recommend that you attend both lectures & exercise sessions
- All lectures will be recorded & online soon after
  - Thanks to our Hiwis Simon, Yi-Chun & Bahoe

# Organization: Lectures

- Meeting times:  
Monday & Thursday 16:15-17:52 (S101, room 00 026)
  - Including a 7-minute break roughly in the middle
- Both Monday and Thursday slots might have lecture content and exercise discussion
  - We highly recommend that you attend both lectures & exercise sessions
- All lectures will be recorded & online soon after
  - Thanks to our Hiwis Simon, Yi-Chun & Bahoe
- We strive for interaction!
  - Discuss with your neighbor:  (2 minutes)
  - Raise your hands to give answers: 
  - Multiple-choice questions  to be answered with colours on smartphone screens
    - + E.g., "Color Screen" on iPhone, or go to:
    - + <https://ml.informatik.uni-freiburg.de/teaching/colors.html>

## Organization: Assignments

- They are voluntary: we don't want to police you
- You can work in teams of up to 3 members

## Organization: Assignments

- They are voluntary: we don't want to police you
- You can work in teams of up to 3 members
- With 200 students we cannot provide much individual feedback
  - Hiwis will provide some remarks on submissions

# Organization: Assignments

- They are voluntary: we don't want to police you
- You can work in teams of up to 3 members
- With 200 students we cannot provide much individual feedback
  - Hiwis will provide some remarks on submissions
- We'll try to optimize amount learned per time spent with assignments
  - Proposals for improvements are always welcome
- Keeping up with the assignments is the best preparation for the exam

# Organization: Gitlab and ILIAS

- Gitlab Server

- A private Git repository just for you (your team)
- Whatever is committed by the deadline is your submission

- Gitlab Server

- A private Git repository just for you (your team)
- Whatever is committed by the deadline is your submission

- ILIAS

- All lecture materials will be posted in ILIAS  
(incl. assignments, code to get you started on assignments, etc)
- There is a forum for questions of all kinds
  - + Making a first post there is part of Assignment 1
  - + Please answer a question if you know the answer
- There is also a Wiki

## Organization: Amount of Work / ECTS points

- This course yields 6 ECTS points → 180 working hours
- Lectures ( $\approx$  45 hours) + assignments + exam preparation
- Time management options
  - A 6-8 hours per assignment, little exam prep. RECOMMENDED
  - B 4-5 hours per assignment, much exam prep. MINIMUM
  - C 0 hours per assignment, ??? exam prep. VERY BAD IDEA

# Organization: Amount of Work / ECTS points

- This course yields 6 ECTS points → 180 working hours
- Lectures ( $\approx$  45 hours) + assignments + exam preparation
- Time management options
  - A 6-8 hours per assignment, little exam prep. RECOMMENDED
  - B 4-5 hours per assignment, much exam prep. MINIMUM
  - C 0 hours per assignment, ??? exam prep. VERY BAD IDEA
- There is a written exam in the end
  - We'll make a mock exam available about two weeks before
  - We're trying to fix the exam date soon (likely mid August)
    - + Erasmus students: same exam as every one else
  - Doing all the exercise sheets and understanding everything behind them is the perfect preparation for the exam

# Lecture Overview

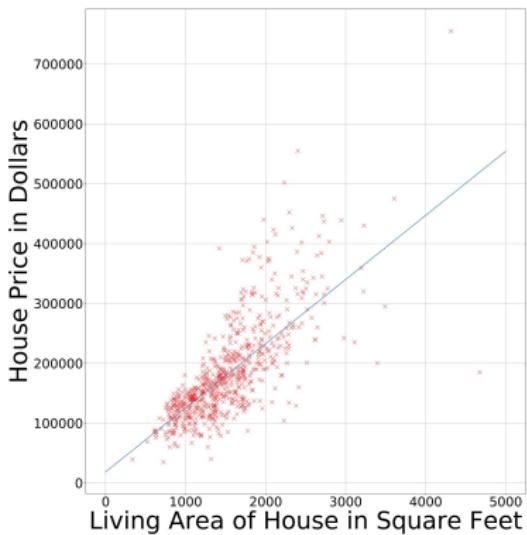
- 1 Motivation: Why Study Machine Learning?
- 2 Organizational Issues
- 3 Introduction to Supervised Machine Learning
- 4 Other Types of Machine Learning
- 5 The Machine Learning Design Cycle
- 6 Wrapup: Summary, Other Courses, Resources, Preview

# Supervised Learning: The Basic Idea

- Use past experience to predict the future
  - Use labelled data points  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$  that we collected in the past
  - to automatically construct a model whose prediction  $\hat{y}_{N+1}$  for a new data point  $\mathbf{x}_{N+1}$  is close to the actual label  $y_{N+1}$ .

# Supervised Learning: The Basic Idea

- Use past experience to predict the future
  - Use labelled data points  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$  that we collected in the past
  - to automatically construct a model whose prediction  $\hat{y}_{N+1}$  for a new data point  $\mathbf{x}_{N+1}$  is close to the actual label  $y_{N+1}$ .
- First example: predicting house price based on size (in square feet)



# Supervised Learning: The Basic Idea

- Use past experience to predict the future
  - Use **labelled data points**  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$  that we collected in the past
  - to automatically construct a model whose prediction  $\hat{y}_{N+1}$  for a new data point  $\mathbf{x}_{N+1}$  is close to the actual label  $y_{N+1}$ .
- Machine learning terminology:
  - Data point  $\mathbf{x}_i$ , often a vector in  $\mathbb{R}^D$
  - Label  $y_i$ 
    - + **Regression:**  $y_i \in \mathbb{R}$
    - + **Classification:**  $y_i$  discrete, e.g.  $y_i \in \{\text{true, false}\}$ , or  $y_i \in \{\text{German, English, Spanish}\}$

# Supervised Learning: The Basic Idea

- Use past experience to predict the future
  - Use **labelled data points**  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$  that we collected in the past
  - to automatically construct a model whose prediction  $\hat{y}_{N+1}$  for a new data point  $\mathbf{x}_{N+1}$  is close to the actual label  $y_{N+1}$ .
- Machine learning terminology:
  - Data point  $\mathbf{x}_i$ , often a vector in  $\mathbb{R}^D$
  - Label  $y_i$ 
    - + **Regression:**  $y_i \in \mathbb{R}$
    - + **Classification:**  $y_i$  discrete, e.g.  $y_i \in \{\text{true, false}\}$ , or  $y_i \in \{\text{German, English, Spanish}\}$
  - Past experience = training set  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$
  - Automatically construct = learn, fit, induce
  - Model = function, hypothesis, classifier/regressor

# Supervised Learning: The Basic Idea

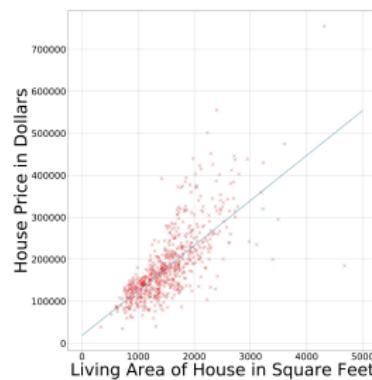
- Use past experience to predict the future
  - Use **labelled data points**  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$  that we collected in the past
  - to automatically construct a model whose prediction  $\hat{y}_{N+1}$  for a new data point  $\mathbf{x}_{N+1}$  is close to the actual label  $y_{N+1}$ .
- Machine learning terminology:
  - Data point  $\mathbf{x}_i$ , often a vector in  $\mathbb{R}^D$
  - Label  $y_i$ 
    - + **Regression:**  $y_i \in \mathbb{R}$
    - + **Classification:**  $y_i$  discrete, e.g.  $y_i \in \{\text{true, false}\}$ , or  $y_i \in \{\text{German, English, Spanish}\}$
  - Past experience = training set  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$
  - Automatically construct = learn, fit, induce
  - Model = function, hypothesis, classifier/regressor
- What is the dimensionality  $D$  in the house price example?
  - ★ D=1
  - ★ D=2
  - ★ D=3
  - ★ D=4

# Supervised Learning: A First Regression Example

## Predicting housing prices

- For many houses in a training set, we know:
  - the square footage
  - the prize they sold for
- We'd like to predict the prize for houses based on their square footage
- One data point: square footage  $x_i$  and its prize  $y_i$  (in US\$)
- This is a regression problem since  $y_i \in \mathbb{R}$

Square footage $x_i$	Prize $y_i$
1.710	208.500
1.262	181.500
1.786	223.500
1.717	140.000
2.198	250.000
1.362	143.000
1.694	307.000
2.090	200.000
1.774	129.900
1.077	118.000
...	...



# Supervised Learning: a Bit More Standard Notation

- We learn a function  $f$  that maps each **data point**  $\mathbf{x}_i$  to a prediction  $\hat{y}_i$  that resembles the true  $y_i$

# Supervised Learning: a Bit More Standard Notation

- We learn a function  $f$  that maps each **data point**  $\mathbf{x}_i$  to a prediction  $\hat{y}_i$  that resembles the true  $y_i$
- In the previous example,  $\mathbf{x}_i$  only had one dimension, but it will typically be a (column) vector with  $D$  dimensions
  - That's why it is bold-faced
  - The dimensions of  $\mathbf{x}_i$  are  $D \times 1$
  - The dimensions of its transpose  $\mathbf{x}_i^T$  are  $1 \times D$

# Supervised Learning: a Bit More Standard Notation

- We learn a function  $f$  that maps each **data point**  $\mathbf{x}_i$  to a prediction  $\hat{y}_i$  that resembles the true  $y_i$
- In the previous example,  $\mathbf{x}_i$  only had one dimension, but it will typically be a (column) vector with  $D$  dimensions
  - That's why it is bold-faced
  - The dimensions of  $\mathbf{x}_i$  are  $D \times 1$
  - The dimensions of its transpose  $\mathbf{x}_i^T$  are  $1 \times D$
- We stack all  $N$  data points on top of each other:

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \dots \\ \mathbf{x}_N^T \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ \dots \\ y_N \end{pmatrix}$$

# Supervised Learning: a Bit More Standard Notation

- We learn a function  $f$  that maps each **data point**  $\mathbf{x}_i$  to a prediction  $\hat{y}_i$  that resembles the true  $y_i$
- In the previous example,  $\mathbf{x}_i$  only had one dimension, but it will typically be a (column) vector with  $D$  dimensions
  - That's why it is bold-faced
  - The dimensions of  $\mathbf{x}_i$  are  $D \times 1$
  - The dimensions of its transpose  $\mathbf{x}_i^T$  are  $1 \times D$
- We stack all  $N$  data points on top of each other:

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_N^T \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}$$

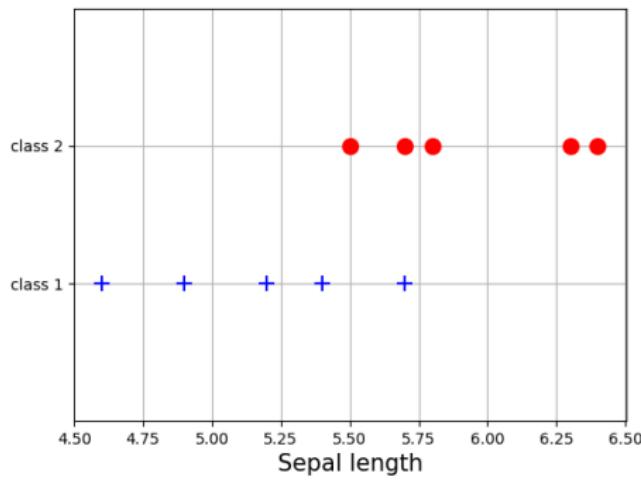
- Thus,  **$\mathbf{X}$**  is a  $N \times D$  matrix;  **$\mathbf{y}$**  is a  $N \times 1$  matrix
  - $\mathbf{y}$  is bold-faced since it's a vector
  - $\mathbf{X}$  is capital and bold-faced since it's a matrix
- **$\mathbf{y}$**  are the 'correct' labels; this is our **supervisory** signal (thus the name)

# Supervised Learning: a Simple Classification Example

- A classical data set from Botany: classifying Iris flowers
  - feature 1: sepal length
  - feature 2: sepal width



$x_{i,1}$	$x_{i,2}$	$y_i$
6.40	2.90	2
5.50	2.50	2
5.20	3.50	1
4.60	3.60	1
5.70	3.80	1
6.30	2.50	2
5.80	2.60	2
4.90	3.10	1
5.70	2.80	2
5.40	3.90	1

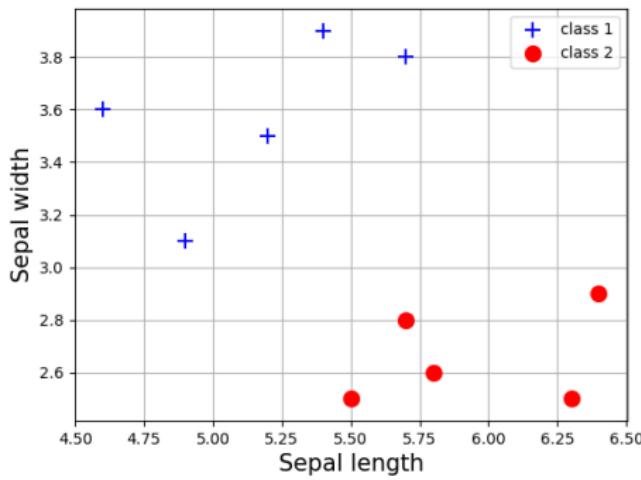


# Supervised Learning: a First Classification Example

- A classical data set from Botany: classifying Iris flowers
  - feature 1: sepal length
  - feature 2: sepal width



$x_{i,1}$	$x_{i,2}$	$y_i$
6.40	2.90	2
5.50	2.50	2
5.20	3.50	1
4.60	3.60	1
5.70	3.80	1
6.30	2.50	2
5.80	2.60	2
4.90	3.10	1
5.70	2.80	2
5.40	3.90	1



# Mini-Quiz in Preparation for Assignment 1: Iris Data

- Classifying Iris flowers (the Iris data set)
  - You will learn a classifier based on 100 labelled Iris flowers
  - Iris flowers are described by 4 characteristics (sepal length/width, petal length/width)
  - There are 3 different classes of Iris flowers

# Mini-Quiz in Preparation for Assignment 1: Iris Data

- Classifying Iris flowers (the Iris data set)
  - You will learn a classifier based on 100 labelled Iris flowers
  - Iris flowers are described by 4 characteristics (sepal length/width, petal length/width)
  - There are 3 different classes of Iris flowers
- The standard supervised learning setup
  - You learn a classifier based on  $\mathbf{X}_{train}$  and  $\mathbf{y}_{train}$
  - You use that classifier to predict  $\mathbf{y}_{test}$  given  $\mathbf{X}_{test}$

# Mini-Quiz in Preparation for Assignment 1: Iris Data

- Classifying Iris flowers (the Iris data set)
  - You will learn a classifier based on 100 labelled Iris flowers
  - Iris flowers are described by 4 characteristics (sepal length/width, petal length/width)
  - There are 3 different classes of Iris flowers
- The standard supervised learning setup
  - You learn a classifier based on  $\mathbf{X}_{train}$  and  $\mathbf{y}_{train}$
  - You use that classifier to predict  $\mathbf{y}_{test}$  given  $\mathbf{X}_{test}$
- What is the correct dimensionality for the matrix  $\mathbf{X}_{train}$ ?
  - ★  $100 \times 3$
  - ★  $3 \times 4$
  - ★  $100 \times 4$
  - ★  $4 \times 100$

# Supervised Learning Example: Credit Risk

- A private customer comes to a bank and applies for a loan
- Should the bank give him the loan?
  - We can build a decision support system for the bank employee

# Supervised Learning Example: Credit Risk

- A private customer comes to a bank and applies for a loan
- Should the bank give him the loan?
  - We can build a decision support system for the bank employee
  - What features could the employee use? 

# Supervised Learning Example: Credit Risk

- A private customer comes to a bank and applies for a loan
- Should the bank give him the loan?
  - We can build a decision support system for the bank employee
  - What features could the employee use? 
  - Features: income, age, credit amount, etc.

# Supervised Learning Example: Credit Risk

- A private customer comes to a bank and applies for a loan
- Should the bank give him the loan?
  - We can build a decision support system for the bank employee
  - What features could the employee use? 
  - Features: income, age, credit amount, etc.
- Ways of casting this as a machine learning problem:
  - 1. Should the bank give the loan?
  - 2. How much interest should the bank charge to break even in expectation?

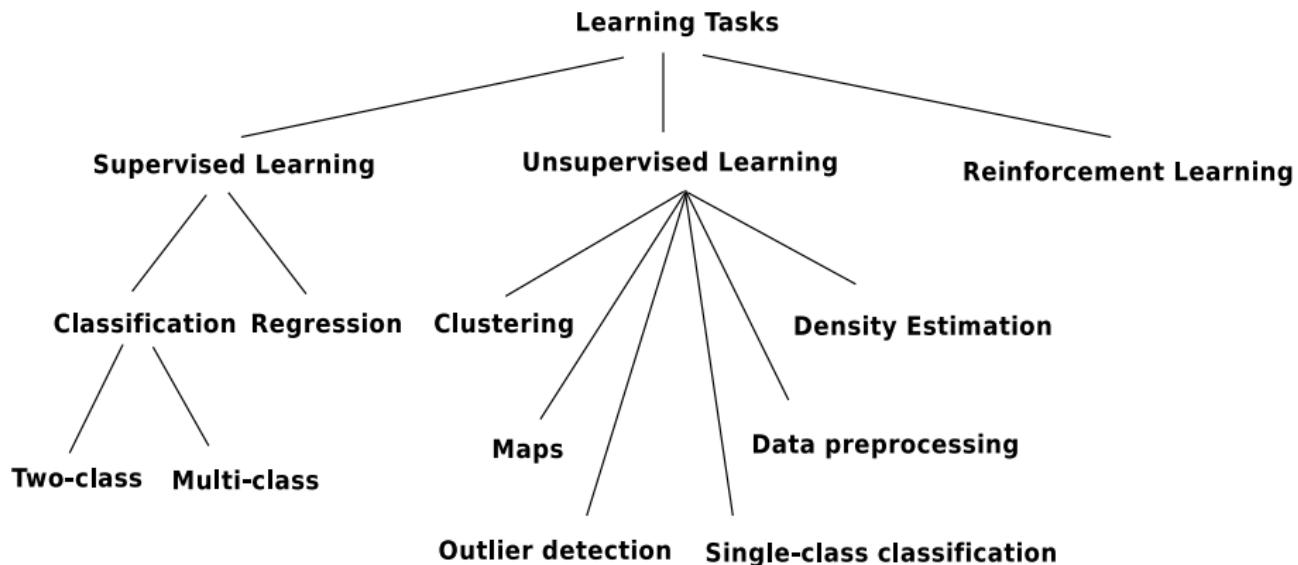
# Supervised Learning Example: Credit Risk

- A private customer comes to a bank and applies for a loan
- Should the bank give him the loan?
  - We can build a decision support system for the bank employee
  - What features could the employee use? 
  - Features: income, age, credit amount, etc.
- Ways of casting this as a machine learning problem:
  - 1. Should the bank give the loan?
  - 2. How much interest should the bank charge to break even in expectation?
- Please raise the color of the correct statement:
  - ★ Both are regression tasks.
  - ★ Both are classification tasks.
  - ★ 1. is regression, 2. is classification.
  - ★ 1. is classification, 2. is regression.

# Lecture Overview

- 1 Motivation: Why Study Machine Learning?
- 2 Organizational Issues
- 3 Introduction to Supervised Machine Learning
- 4 Other Types of Machine Learning
- 5 The Machine Learning Design Cycle
- 6 Wrapup: Summary, Other Courses, Resources, Preview

# Types of Learning Tasks



# Types of learning tasks: Unsupervised Learning

- training data consists of a description of situations/ objects/ ...
- learn what is typical/ similar/ interesting/ strange within the data
- no teacher
- example: **clustering** of things that 'somehow belong together'
  - e.g. cluster customers into those with similar behavior
- example: **novelty detection**
  - detect objects, that are surprising / different from others
  - e.g. find suspicious outliers in radioactivity measurements

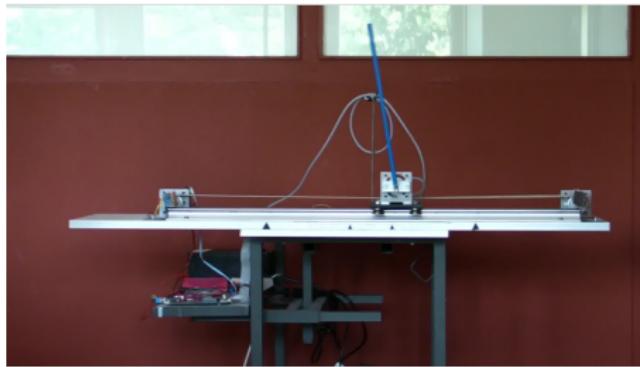
## Types of learning tasks: Reinforcement learning

- training data consist of sequences of situations, actions and reward
- task is: learn a control strategy to maximize the reward
- no teacher
- (famous) example: learn to balance a pole

→ treated in detail in lecture [Reinforcement Learning](#)

# Example from our work: learning dynamics models for optimal control

- task is to swing up and balance a pole on a cart
- no knowledge of system dynamics
- use of regression technique to learn dynamics function from (random) interaction
- use model for planning locally optimal trajectories (model-based RL)



## Example from our work: algorithm runtime prediction

- Task: predict how long an algorithm will run
    - on new inputs
    - with new parameter settings
- regression problem

## Example from our work: algorithm runtime prediction

- Task: predict how long an algorithm will run
    - on new inputs
    - with new parameter settings
- regression problem
- Task: will the algorithm terminate within 1 minute?
- classification problem

## Example from our work: algorithm runtime prediction

- Task: predict how long an algorithm will run
  - on new inputs
  - with new parameter settings
- regression problem
- Task: will the algorithm terminate within 1 minute?
- classification problem
- Are there different types of input instances?
- clustering problem

## Example from our work: algorithm runtime prediction

- Task: predict how long an algorithm will run
  - on new inputs
  - with new parameter settings
- regression problem
- Task: will the algorithm terminate within 1 minute?
- classification problem
- Are there different types of input instances?
- clustering problem
- Can we adapt the algorithm's parameters at runtime to make it better?
- reinforcement learning

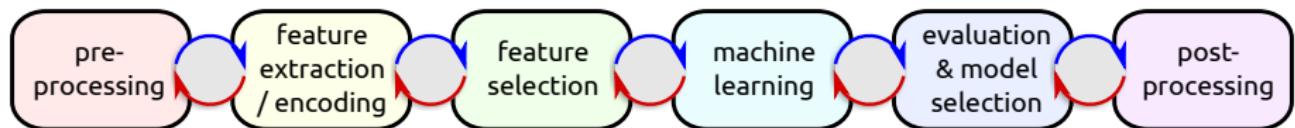
# Lecture Overview

- 1 Motivation: Why Study Machine Learning?
- 2 Organizational Issues
- 3 Introduction to Supervised Machine Learning
- 4 Other Types of Machine Learning
- 5 The Machine Learning Design Cycle
- 6 Wrapup: Summary, Other Courses, Resources, Preview

# Applying machine learning in practice

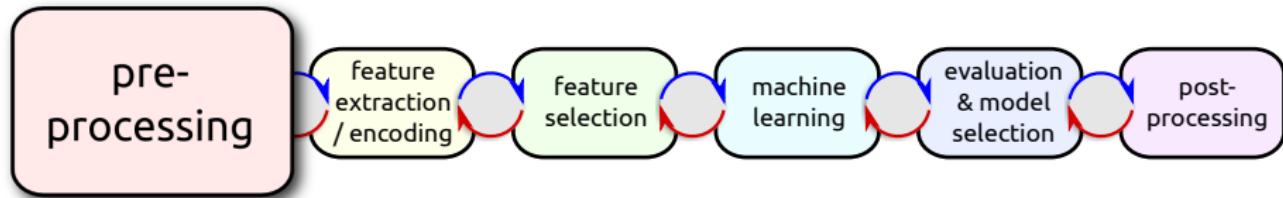
- Let's say we want to apply machine learning to a practical problem
- What do you think are some of the things we need to do? 

# The Machine Learning Design Cycle



- Real ML applications have many different steps, including:
  - Getting the data into the convenient  $(X,y)$  form
  - Casting your problem as an ML problem (supervised, unsupervised, RL)
  - Evaluating your solution & using it in practice

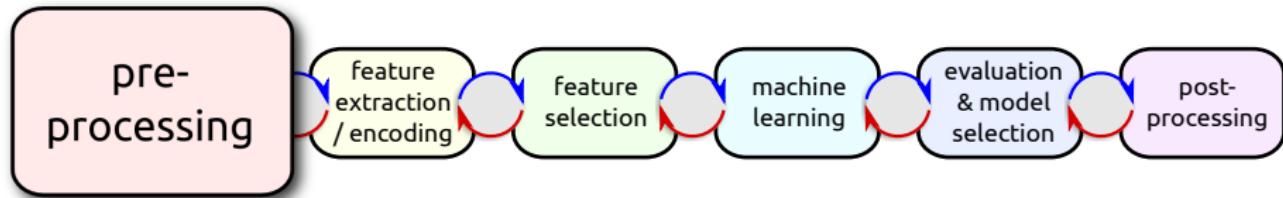
# ML Design Cycle: Preprocessing



Get the data in the first place

- Design sensor systems, design questionnaires
- Pay people for labels
- Get legal approval to use data, etc.

# ML Design Cycle: Preprocessing



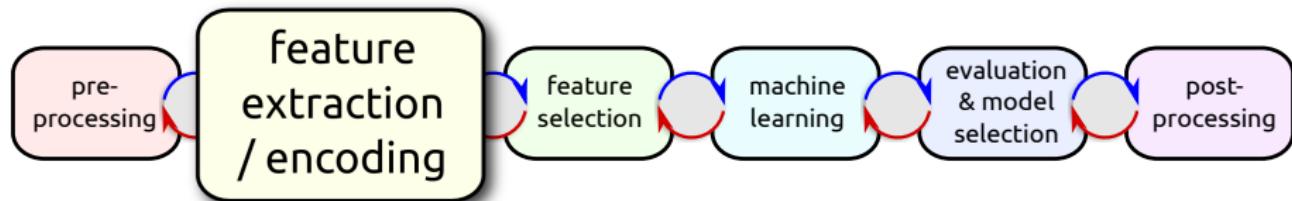
Get the data in the first place

- Design sensor systems, design questionnaires
- Pay people for labels
- Get legal approval to use data, etc.

Clean the data

- Missing values may not be missing at random
- Outliers may be due to errors in data acquisition or carry meaning
- Imbalanced data → e.g., subsample majority class / create additional data from minority class, etc.

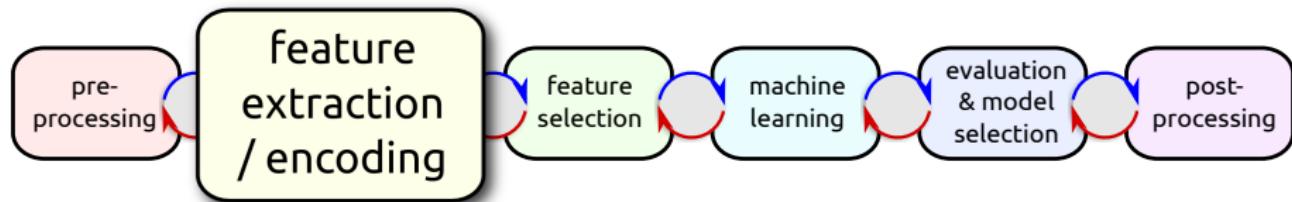
# ML Design Cycle: Feature Extraction & Encoding (1/3)



Extract features: getting a nice  $X$  matrix

- Often unclear what the features should be.
  - E.g., what's a good feature representation of a human cell, a text, a sound, a brainwave, an image?
- Solution 1: use domain knowledge!
- Solution 2: representation learning (e.g., deep neural networks)

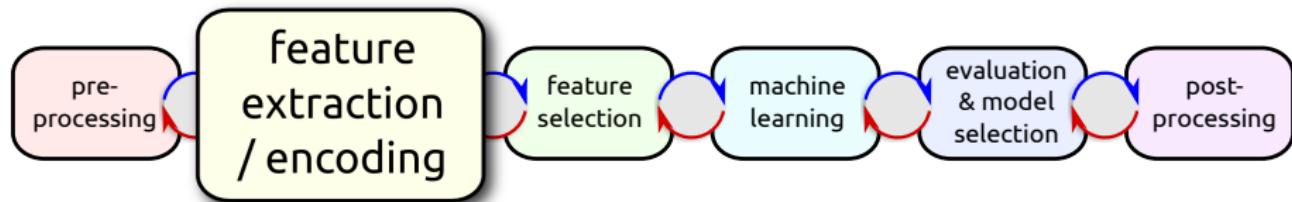
# ML Design Cycle: Feature Extraction & Encoding (2/3)



## Combine features:

- We can use complex operations to combine features
- E.g., body mass index:  $BMI = \text{weight} / (\text{height}^2)$   
(With weight in kg and height in cm)
- Combined features can be more expressive than their components

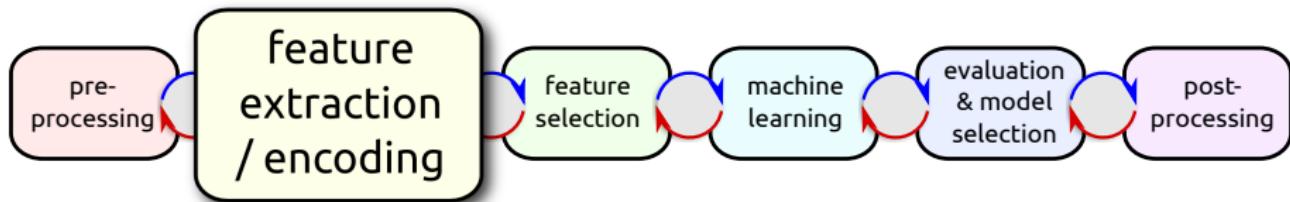
# ML Design Cycle: Feature Extraction & Encoding (3/3)



Encode features: getting to a standard format

- Most prominent example: **categorical features**
  - Features with unordered, finite domain
  - E.g.,  $\{\text{red, green, blue}\}$
  - These could be coded as  $\text{red}=1$ ,  $\text{green}=2$ ,  $\text{blue}=3$
  - But red is not closer to green than it is to blue ...

# ML Design Cycle: Feature Extraction & Encoding (3/3)



Encode features: getting to a standard format

- Most prominent example: **categorical features**
  - Features with unordered, finite domain
  - E.g., {red, green, blue}
  - These could be coded as red=1, green=2, blue=3
  - But red is not closer to green than it is to blue ...
- Standard solution: **one-hot-encoding** (aka 1-in-k encoding)
  - $k$ -ary feature is coded as  $k$  features in  $\{0,1\}$  with exactly one 1
  - E.g., red = [1,0,0]
  - E.g., green = [0,1,0]
  - E.g., blue = [0,0,1]

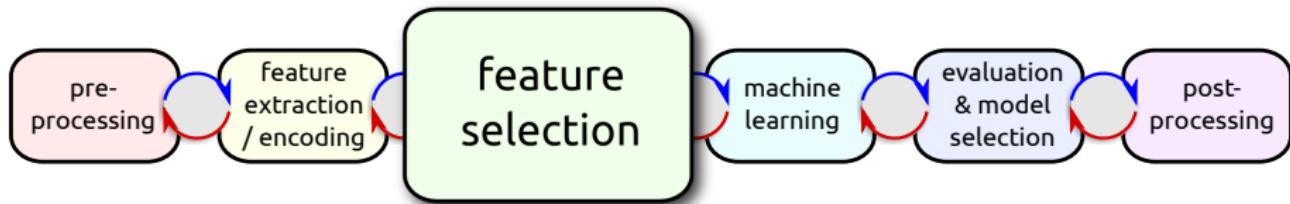
# Mini-Quiz in Preparation for Assignment 1: Iris Data

- Classifying Iris Flowers
  - You will learn a classifier based on 100 labelled Iris flowers
  - Iris flowers are described by 4 characteristics (sepal length/width, petal length/width)
  - There are 3 different classes of Iris flowers
- Let's say we make an additional feature out of the class of flowers (and want to predict whether goats like the flower or not)
  - The flower class is categorical, so we'll use a one-hot encoding

# Mini-Quiz in Preparation for Assignment 1: Iris Data

- Classifying Iris Flowers
  - You will learn a classifier based on 100 labelled Iris flowers
  - Iris flowers are described by 4 characteristics (sepal length/width, petal length/width)
  - There are 3 different classes of Iris flowers
- Let's say we make an additional feature out of the class of flowers (and want to predict whether goats like the flower or not)
  - The flower class is categorical, so we'll use a one-hot encoding
- What is the correct dimensionality for the matrix  $\mathbf{X}_{train}$  now?
  - ★ Still  $100 \times 4$
  - ★  $100 \times 5$
  - ★  $100 \times 6$
  - ★  $100 \times 7$

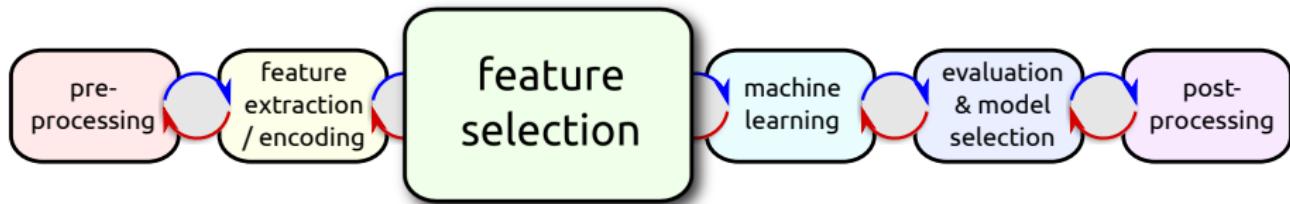
# ML Design Cycle: Feature Selection



Select features: getting rid of distractions

- Sometimes, we have too many features ( $D$  features)
  - Source 1: original data (e.g., 20,000-25,000 human protein-coding genes)
  - Source 2: feature creation (e.g., trying lots of feature combinations)

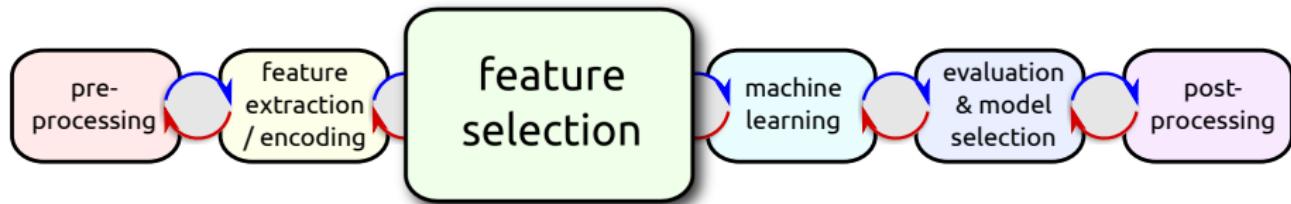
# ML Design Cycle: Feature Selection



Select features: getting rid of distractions

- Sometimes, we have too many features ( $D$  features)
  - Source 1: original data (e.g., 20,000-25,000 human protein-coding genes)
  - Source 2: feature creation (e.g., trying lots of feature combinations)
- We need to select the ones useful for the machine learner
  - Strongly depends on the learning algorithm applied
  - Some algorithms do 'automatic' feature selection, scale to large  $D$

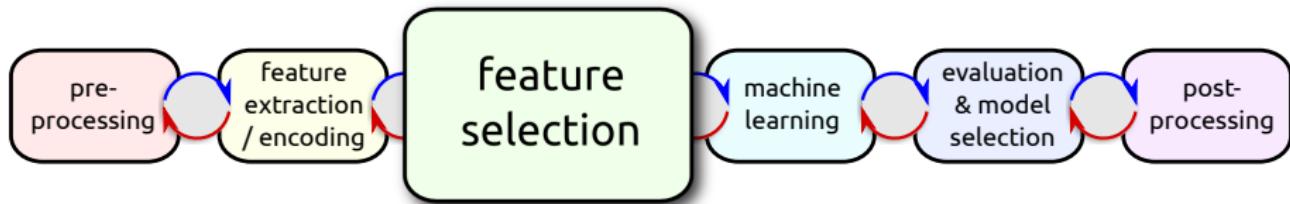
# ML Design Cycle: Feature Selection



Select features: getting rid of distractions

- Sometimes, we have too many features ( $D$  features)
  - Source 1: original data (e.g., 20,000-25,000 human protein-coding genes)
  - Source 2: feature creation (e.g., trying lots of feature combinations)
- We need to select the ones useful for the machine learner
  - Strongly depends on the learning algorithm applied
  - Some algorithms do ‘automatic’ feature selection, scale to large  $D$
  - Some algorithms’ computational complexity scales poorly in  $D$ 
    - + They simply run out of memory when  $D$  is too large

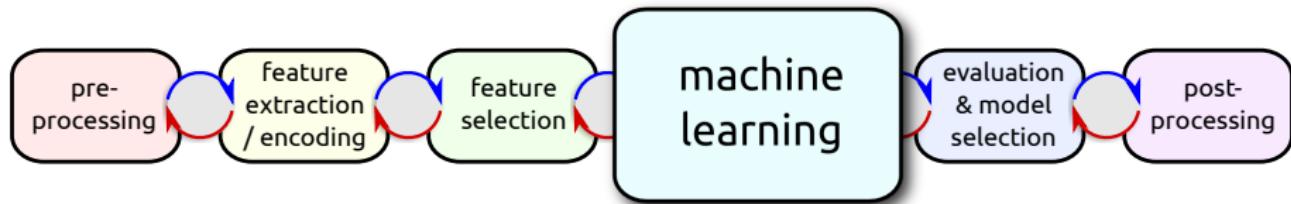
# ML Design Cycle: Feature Selection



Select features: getting rid of distractions

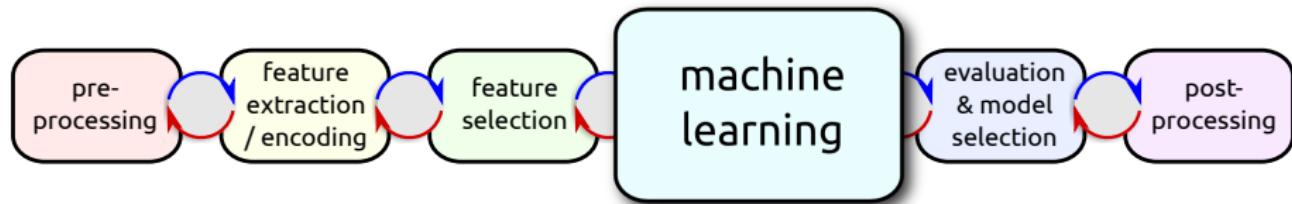
- Sometimes, we have too many features ( $D$  features)
  - Source 1: original data (e.g., 20,000-25,000 human protein-coding genes)
  - Source 2: feature creation (e.g., trying lots of feature combinations)
- We need to select the ones useful for the machine learner
  - Strongly depends on the learning algorithm applied
  - Some algorithms do 'automatic' feature selection, scale to large  $D$
  - Some algorithms' computational complexity scales poorly in  $D$ 
    - + They simply run out of memory when  $D$  is too large
  - Some algorithms' predictive performance scales poorly in  $D$ 
    - + They won't break but simply won't do very well

# ML Design Cycle: 'Machine Learning'



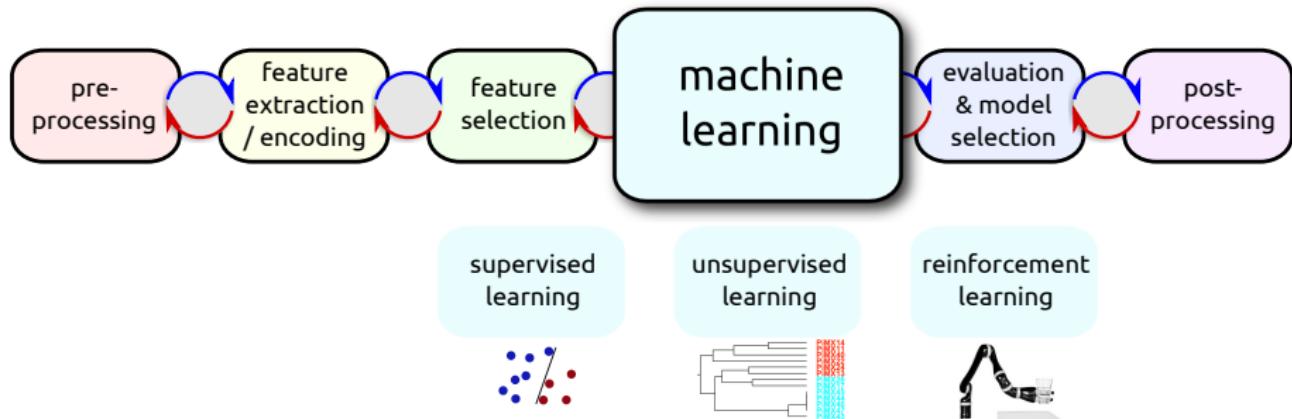
- The 'core' of a machine learning system
- Different algorithms we could use to solve a problem
- This is where 90% of machine learning research happens
  - Computational efficiency
  - Statistical efficiency
  - Provable properties

# ML Design Cycle: 'Machine Learning'



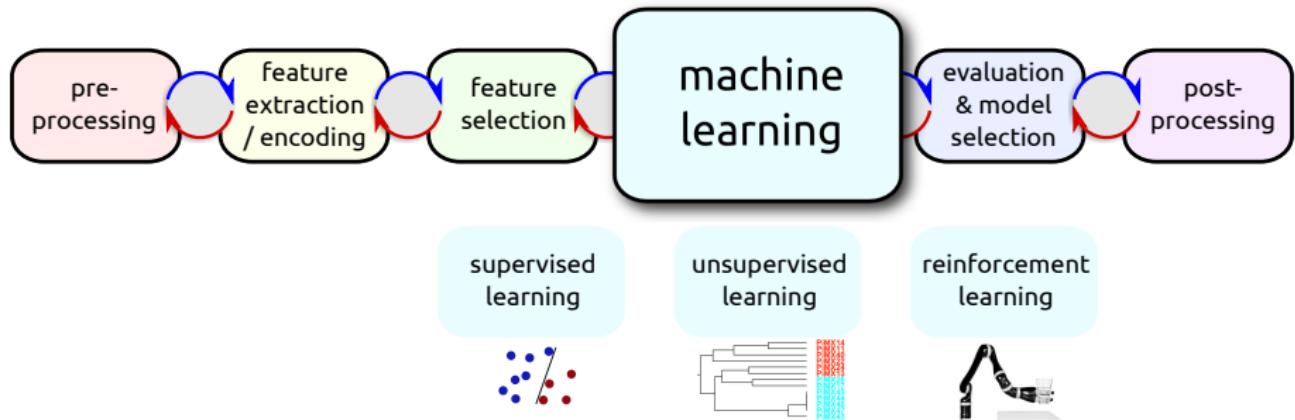
- The 'core' of a machine learning system
- Different algorithms we could use to solve a problem
- This is where 90% of machine learning research happens
  - Computational efficiency
  - Statistical efficiency
  - Provable properties
- In practice, this is not much more important than the other parts
- We'll still focus on this part (since there is the most 'meat')
  - But we connect to the other parts wherever possible
  - In practice, the features are often more important than the algorithm

# ML Design Cycle: 'Machine Learning'



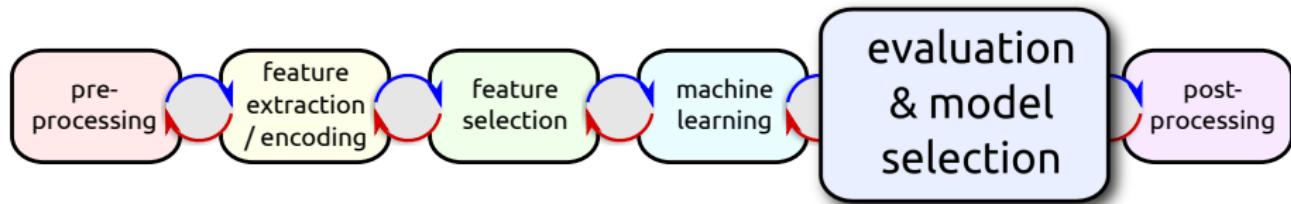
- Sometimes, you can phrase your learning problem in various ways
- Or you can combine various machine learning paradigms
- E.g., AlphaGo:
  - classification to predict expert moves
  - regression to evaluate board positions
  - reinforcement learning by self-play

# ML Design Cycle: 'Machine Learning'



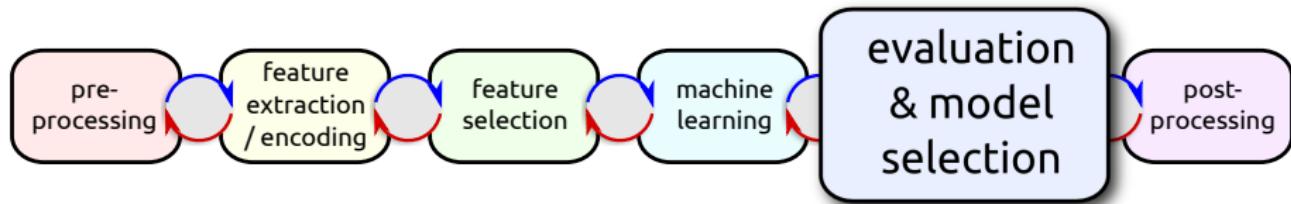
- E.g., learning in infants
  - Lots of unsupervised data
    - + Continuous perception of the environment
    - + Can suffice to learn a representation
  - Very few data points with a supervision signal  $y$ 
    - + E.g., 'Look, that's called a dog!'
    - Infants can learn the class 'dog' from a handful of examples

# ML Design Cycle: Evaluation and Model Selection 1/3



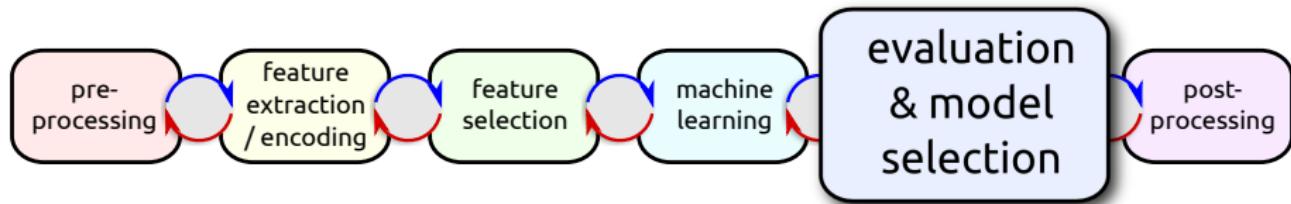
## Evaluation:

- We want our models to **generalize**
  - I.e., perform well on previously unseen data points
- What does it mean to perform well?
  - Precision, recall, false positives, false negatives
  - Speed at training time, speed at test time, memory, accuracy



## Model Selection: Which Algorithm to Use?

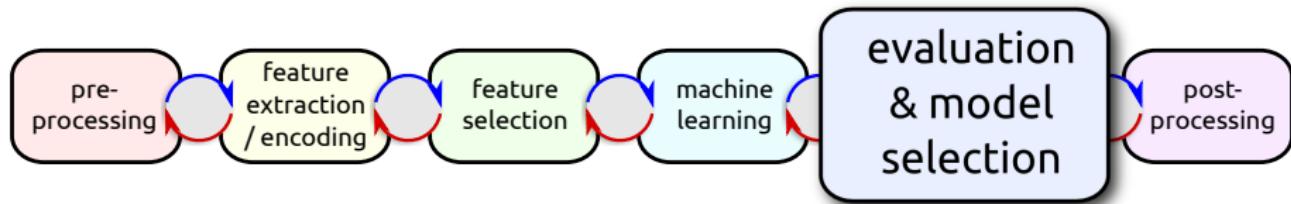
- Computational efficiency, accuracy
- Estimates of generalization performance
- Simplest approach: split dataset into training and validation set
  - Choose algorithm that does best on the validation set
  - Only that algorithm gets deployed and applied to new test data



## Model Selection: Which Algorithm to Use?

- Computational efficiency, accuracy
- Estimates of generalization performance
- Simplest approach: split dataset into training and validation set
  - Choose algorithm that does best on the validation set
  - Only that algorithm gets deployed and applied to new test data
  - To evaluate on a fixed dataset
    - + Split dataset into training/validation/test
    - + Lock away test set for final test
    - + Train on training part, choose best w.r.t. validation part

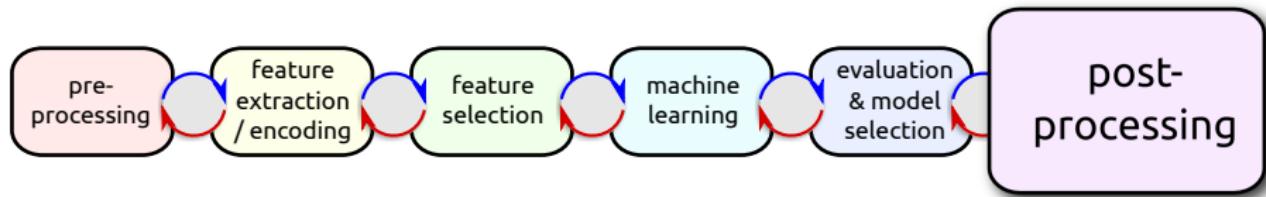
# ML Design Cycle: Evaluation and Model Selection 3/3



Model Selection: How to set free parameters of the algorithm?

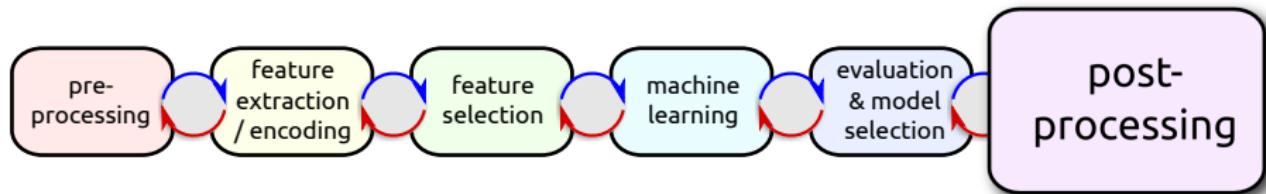
- In machine learning, free tuning parameters are called **hyperparameters**
  - in contrast to internal **model parameters**
- Find the hyperparameter setting with best validation performance
- Exponential number of possible settings

# The Machine Learning Design Cycle



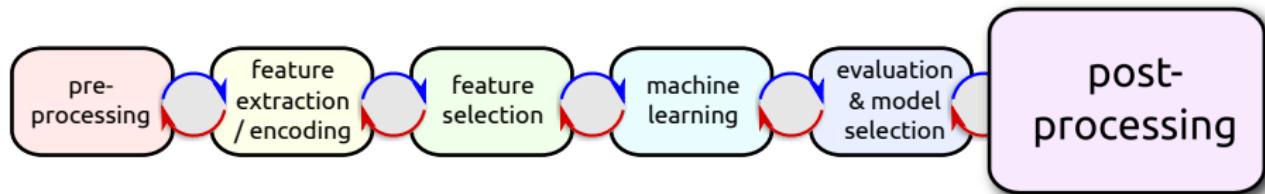
- Ensure **fairness** / avoid algorithmic bias
  - E.g., model should not discriminate based on race, gender, religion, sexual orientation, ...

# The Machine Learning Design Cycle



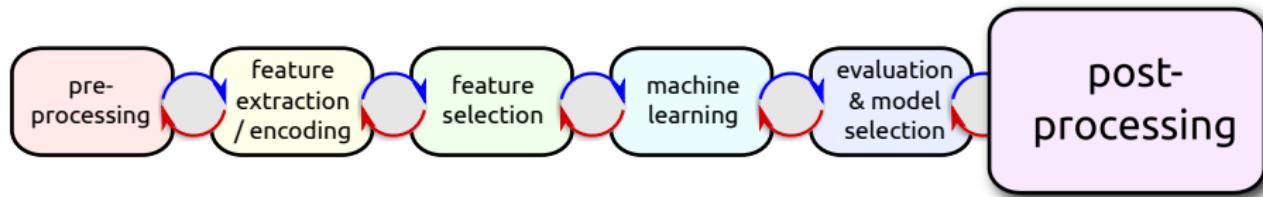
- Ensure **fairness** / avoid algorithmic bias
  - E.g., model should not discriminate based on race, gender, religion, sexual orientation, ...
- Some models don't yield proper probabilities
  - Transform model outputs to probabilities

# The Machine Learning Design Cycle



- Ensure **fairness** / avoid algorithmic bias
  - E.g., model should not discriminate based on race, gender, religion, sexual orientation, ...
- Some models don't yield proper probabilities
  - Transform model outputs to probabilities
- Integrate model predictions with other sources of uncertainty
  - E.g., prediction whether a treatment would help a patient
  - E.g., prediction of side effects
  - E.g., prediction whether the ward has a free bed ...

# The Machine Learning Design Cycle



- Ensure **fairness** / avoid algorithmic bias
  - E.g., model should not discriminate based on race, gender, religion, sexual orientation, ...
- Some models don't yield proper probabilities
  - Transform model outputs to probabilities
- Integrate model predictions with other sources of uncertainty
  - E.g., prediction whether a treatment would help a patient
  - E.g., prediction of side effects
  - E.g., prediction whether the ward has a free bed ...
- Deploy the model we found to achieve something
  - Don't just predict, but **act** based on the model

# Lecture Overview

- 1 Motivation: Why Study Machine Learning?
- 2 Organizational Issues
- 3 Introduction to Supervised Machine Learning
- 4 Other Types of Machine Learning
- 5 The Machine Learning Design Cycle
- 6 Wrapup: Summary, Other Courses, Resources, Preview

# Summary by learning goals

Having heard this lecture, you can now ...

- explain why so many people are interested in machine learning
- give examples for successful machine learning applications
- describe supervised machine learning
- distinguish between classification and regression problems
- distinguish supervised machine learning from other machine learning problems
- describe the parts of the machine learning pipeline

# Most related courses at Uni Freiburg

- Statistical pattern recognition

- By Thomas Brox, [this term](#)
- A probabilistic machine learning course
- Our course is designed to be complementary
- If your main interest is machine learning,  
we strongly encourage you to take this course as well

# Most related courses at Uni Freiburg

- Statistical pattern recognition

- By Thomas Brox, [this term](#)
- A probabilistic machine learning course
- Our course is designed to be complementary
- If your main interest is machine learning,  
we strongly encourage you to take this course as well

- Foundations of Deep Learning

- By Frank Hutter Joschka Boedecker, WS
- More advanced course that builds on this current course

# Most related courses at Uni Freiburg

- Statistical pattern recognition

- By Thomas Brox, [this term](#)
- A probabilistic machine learning course
- Our course is designed to be complementary
- If your main interest is machine learning,  
we strongly encourage you to take this course as well

- Foundations of Deep Learning

- By Frank Hutter Joschka Boedecker, WS
- More advanced course that builds on this current course

- Reinforcement Learning

- By Joschka Boedecker, WS
- More advanced course that builds on this current course

# Most related courses at Uni Freiburg

- Statistical pattern recognition
  - By Thomas Brox, [this term](#)
  - A probabilistic machine learning course
  - Our course is designed to be complementary
  - If your main interest is machine learning,  
we strongly encourage you to take this course as well
- Foundations of Deep Learning
  - By Frank Hutter Joschka Boedecker, WS
  - More advanced course that builds on this current course
- Reinforcement Learning
  - By Joschka Boedecker, WS
  - More advanced course that builds on this current course
- Lab Course Deep Learning
  - By groups Boedecker, Brox, Burgard, Hutter, SS
  - More advanced course that builds on the Deep Learning course

# Other related courses at Uni Freiburg

- Artificial Intelligence

- By groups Burgard, Nebel, Boedecker, Hutter
- This term

# Other related courses at Uni Freiburg

- Artificial Intelligence

- By groups Burgard, Nebel, Boedecker, Hutter
- This term

- Information Retrieval

- By Hannah Bast, WS

# Other related courses at Uni Freiburg

- Artificial Intelligence
  - By groups Burgard, Nebel, Boedecker, Hutter
  - This term
- Information Retrieval
  - By Hannah Bast, WS
- Automated Machine Learning (AutoML)
  - By Marius Lindauer & Frank Hutter, SS
  - More advanced course that builds on this current course & Deep Learning

# We'll use material from several books, including these

- The Elements of Statistical Learning  
by Hastie, Tibshirani and Friedman

- Available online for free: <https://web.stanford.edu/~hastie/ElemStatLearn/>

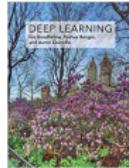


- Pattern Classification by Duda, Hart and Stork



- Deep Learning  
by Goodfellow, Bengio and Courville

- HTML version available online for free:  
<http://www.deeplearningbook.org/>



# Python tutorials

- General:
  - Official tutorial:  
<https://docs.python.org/3/tutorial/index.html>
  - For beginners: [www.learnpython.org/](http://www.learnpython.org/)
  - For programmers:  
[http://stephensugden.com/crash\\_into\\_python/](http://stephensugden.com/crash_into_python/)
  - Many more: <http://docs.python-guide.org/en/latest/intro/learning/>
- Libraries:
  - Numpy: [http://wiki.scipy.org/Tentative\\_NumPy\\_Tutorial](http://wiki.scipy.org/Tentative_NumPy_Tutorial)
  - SciPy: <http://docs.scipy.org/doc/scipy/reference/tutorial/>
  - Matplotlib: <http://matplotlib.org/users/beginner.html>
- Feel free to discuss about these and additional ones on ILIAS ...

## Other resources for the course

We'll also refer to other sources for background throughout. Here is one to start with:

- Khan Academy: <https://www.khanacademy.org/>
  - Useful to review linear algebra, etc
- Again, please post resources you find useful on ILIAS

# Preview of Assignment 1

- Set up infrastructure for the remainder of the course
- Explore a simple classification dataset
- Classify with some manually-derived rules

# Preview of Assignment 1

- Set up infrastructure for the remainder of the course
- Explore a simple classification dataset
- Classify with some manually-derived rules
- Preparation **to do today**: download our virtual machine (VM) from  
<http://ml.informatik.uni-freiburg.de/~zelaa/ml2019/mlss19.ova>
  - The VM has 2GB, so you cannot download it on the fly on Monday
  - Instructions for how to set it up are on  
<http://ml.informatik.uni-freiburg.de/~zelaa/ml2019>, but we'll also cover this on Monday

# Preview of Assignment 1

- Set up infrastructure for the remainder of the course
- Explore a simple classification dataset
- Classify with some manually-derived rules
- Preparation to do today: download our virtual machine (VM) from <http://ml.informatik.uni-freiburg.de/~zelaa/ml2019/mlss19.ova>
  - The VM has 2GB, so you cannot download it on the fly on Monday
  - Instructions for how to set it up are on <http://ml.informatik.uni-freiburg.de/~zelaa/ml2019>, but we'll also cover this on Monday
- On Monday:
  - Bring your (charged!) laptops with the VM image on it!
  - We'll set up the teams
  - Space-permitting, we'll arrange seating so we can walk through & help

# Lecture 2: Linear Classification

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



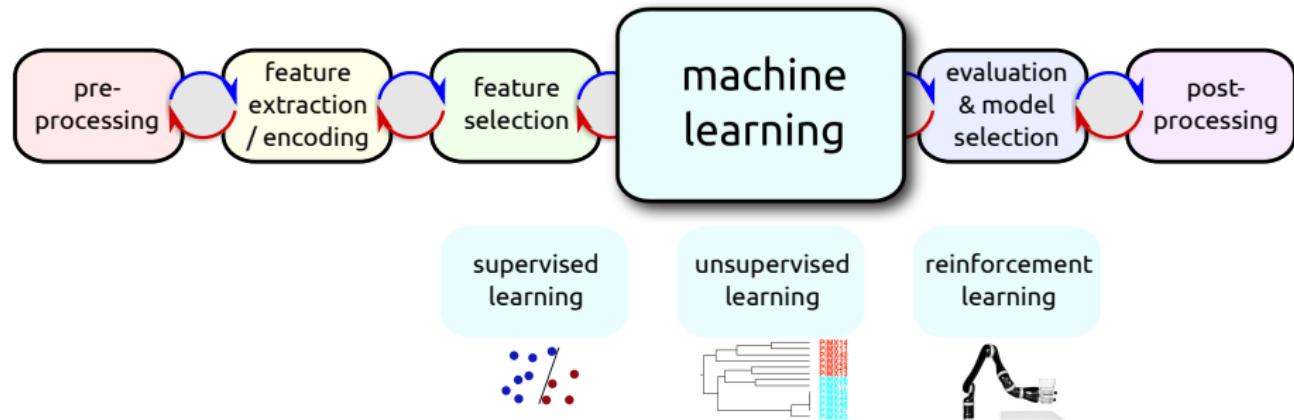
# Lecture Overview

- 1 Brief Recapitulation: Supervised Classification
- 2 Motivation: Linear Discriminant Functions
- 3 LDA Assumptions and Data Scenarios
- 4 Geometric Interpretation
- 5 How to Derive the Parameters...
- 6 Wrapup: Summary, Related Topics, Preview

# Lecture Overview

- 1 Brief Recapitulation: Supervised Classification
- 2 Motivation: Linear Discriminant Functions
- 3 LDA Assumptions and Data Scenarios
- 4 Geometric Interpretation
- 5 How to Derive the Parameters...
- 6 Wrapup: Summary, Related Topics, Preview

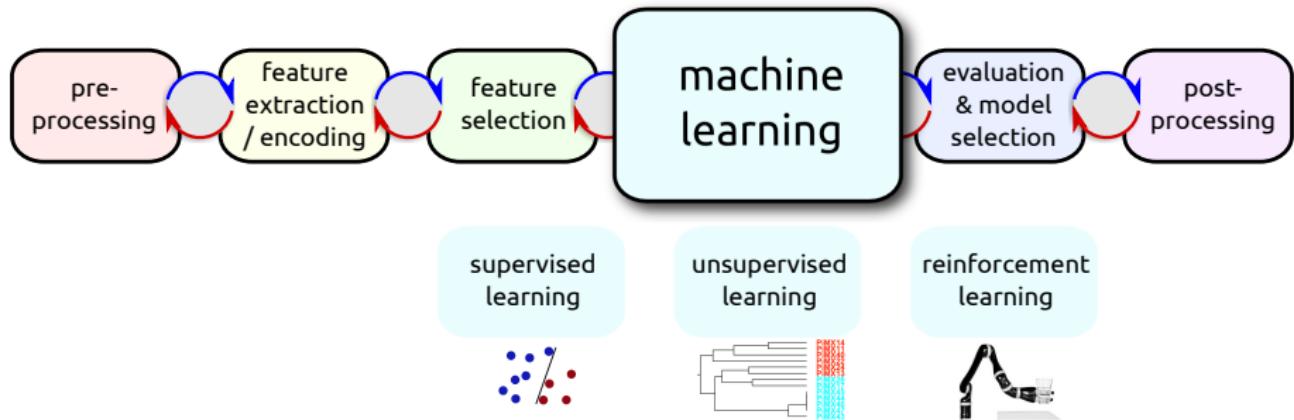
# Reminder: ML Design Cycle



Linear discriminant functions are for **supervised classification**:

- Use past experience to predict the future

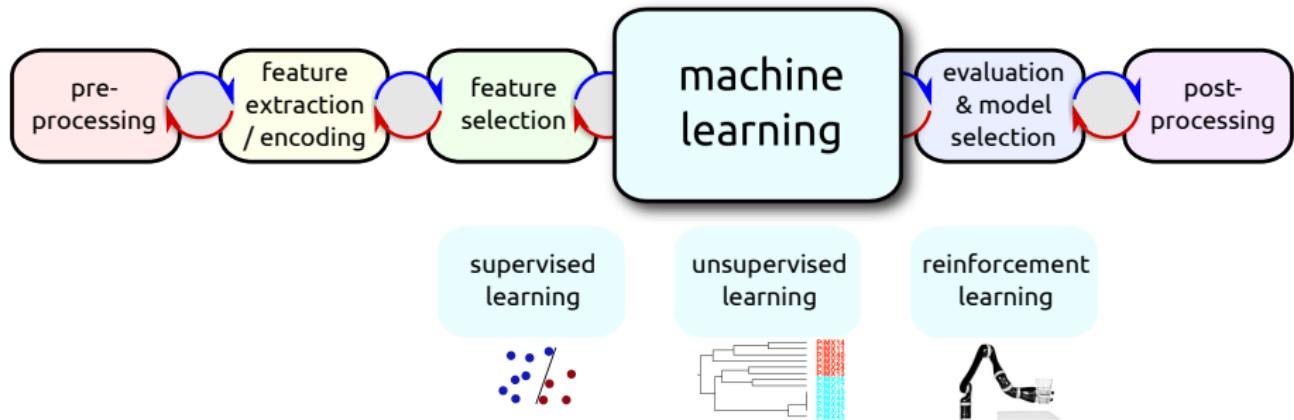
# Reminder: ML Design Cycle



Linear discriminant functions are for **supervised classification**:

- Use past experience to predict the future
- Use **labelled data points**  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$

# Reminder: ML Design Cycle



Linear discriminant functions are for **supervised classification**:

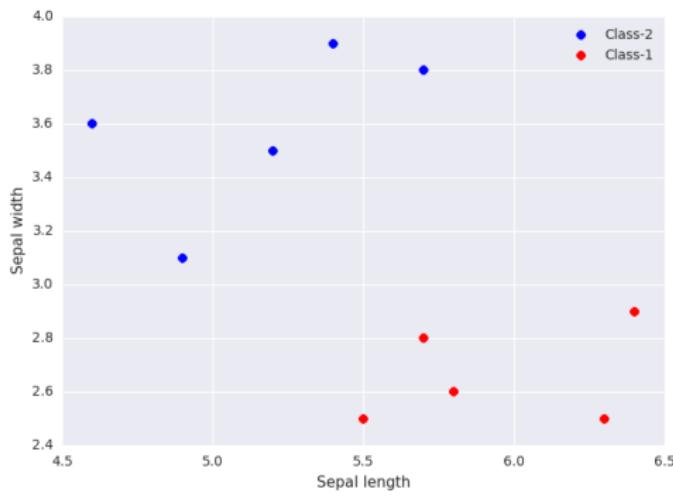
- Use past experience to predict the future
- Use **labelled data points**  $\langle (\mathbf{x}_i, y_i) \rangle_{i=1}^N$
- Train a **model** which can predict the label  $y_{N+1}$  of a new data point  $\mathbf{x}_{N+1}$

# Reminder: A Simple Classification Example

- A classical data set from Botany: classifying Iris flowers
  - feature 1: sepal length
  - feature 2: sepal width



$x_{i,1}$	$x_{i,2}$	$y_i$
6.40	2.90	2
5.50	2.50	2
5.20	3.50	1
4.60	3.60	1
5.70	3.80	1
6.30	2.50	2
5.80	2.60	2
4.90	3.10	1
5.70	2.80	2
5.40	3.90	1



## Reminder: Terminology



- A data point  $\mathbf{x}_i$  is a column vector in  $\mathbb{R}^D$
- A label  $y$  is discrete, e.g.  $y \in \{0,1\}$

We are given a labeled data set of  $N$  examples:

- $\mathbf{X}$  is a  $N \times D$  matrix containing the continuous **feature** values.  
( $\mathbf{X}$  contains one transposed data point  $\mathbf{x}_i^T$  per row.)
- $\mathbf{y}$  is a  $N \times 1$  vector containing the discrete **labels**.

# Classification: The Workflow

(omitting the subscripts " $i$ " for data points for convenience...)

- Learn a **function**  $f(\mathbf{x})$ , which shall separate the two classes as *good* as possible.

# Classification: The Workflow

(omitting the subscripts " $i$ " for data points for convenience...)

- Learn a **function**  $f(\mathbf{x})$ , which shall separate the two classes as *good* as possible.
- Once  $f(\mathbf{x})$  has been learned you can make decisions on a novel input vector  $\mathbf{x}$ :
  - assign  $\mathbf{x}$  to class  $\mathcal{C}_1$  if  $f(\mathbf{x}) \geq 0$
  - assign  $\mathbf{x}$  to class  $\mathcal{C}_2$  otherwise

# Classification: The Workflow

(omitting the subscripts " $i$ " for data points for convenience...)

- Learn a **function**  $f(\mathbf{x})$ , which shall separate the two classes as *good* as possible.
- Once  $f(\mathbf{x})$  has been learned you can make decisions on a novel input vector  $\mathbf{x}$ :
  - assign  $\mathbf{x}$  to class  $\mathcal{C}_1$  if  $f(\mathbf{x}) \geq 0$
  - assign  $\mathbf{x}$  to class  $\mathcal{C}_2$  otherwise

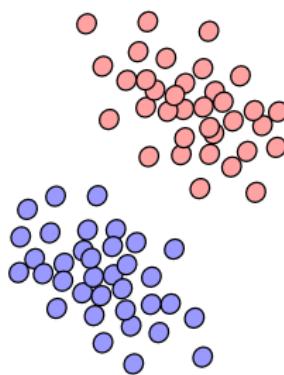
Thus the corresponding **decision boundary**  $H$  is defined by the relation  $f(\mathbf{x}) = 0$ .

# Lecture Overview

- 1 Brief Recapitulation: Supervised Classification
- 2 Motivation: Linear Discriminant Functions
- 3 LDA Assumptions and Data Scenarios
- 4 Geometric Interpretation
- 5 How to Derive the Parameters...
- 6 Wrapup: Summary, Related Topics, Preview

# Linear Discriminant Function: The Basic Idea

Example of two **linearly separable** classes with data points  $\mathbf{x} \in \mathbb{R}^2$ .  
Which is the best decision boundary? Please vote.

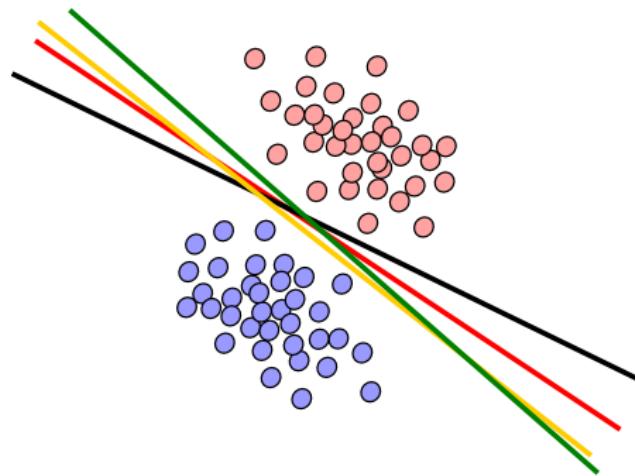


# Linear Discriminant Function: The Basic Idea

Example of two **linearly separable** classes with data points  $x \in \mathbb{R}^2$ .



Which is the best decision boundary? Please vote.



# Linear Discriminant Function: The Basic Idea

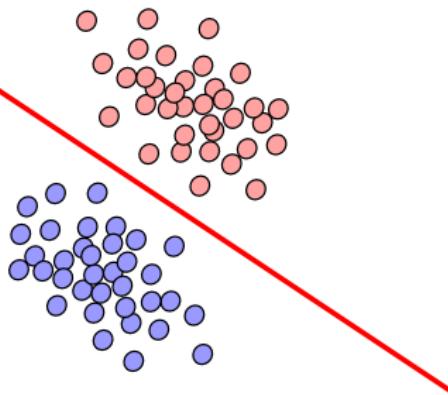
What is your intuition based on?



$$f(x) > 0$$

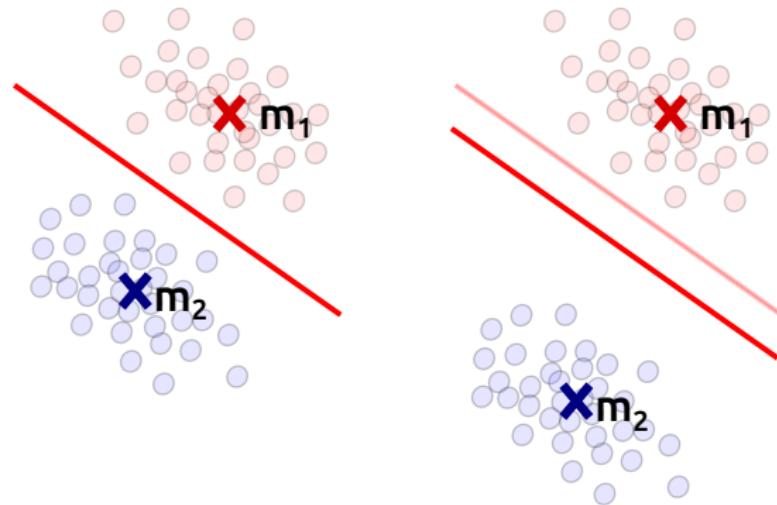
$$f(x) = 0$$

$$f(x) < 0$$



# What influences the Decision Boundary?

Influence of class means  $m_1$  and  $m_2$ :

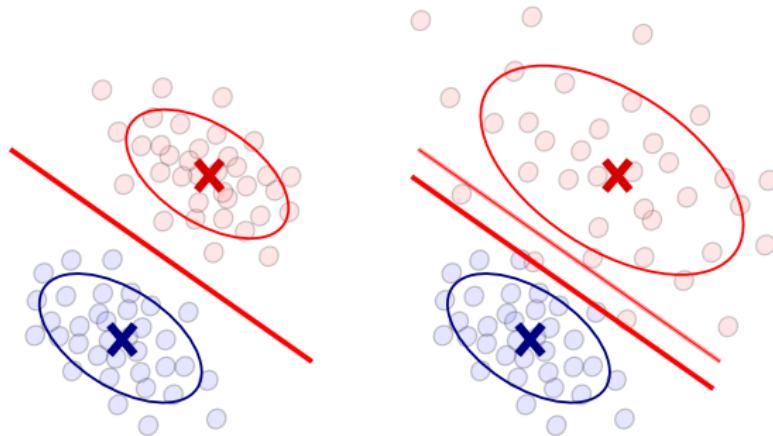


With  $N_k$  points in class  $\mathcal{C}_k$ :

$$\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in \mathcal{C}_1} \mathbf{x}_n \quad \text{and} \quad \mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in \mathcal{C}_2} \mathbf{x}_n$$

# What influences the Decision Boundary?

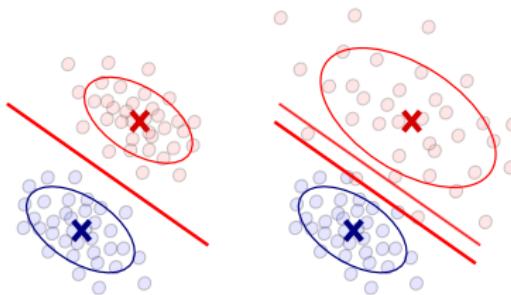
Influence of the size and shape of the distribution:



How can a (normal) distribution be described?



## Reminder: Covariance Matrix

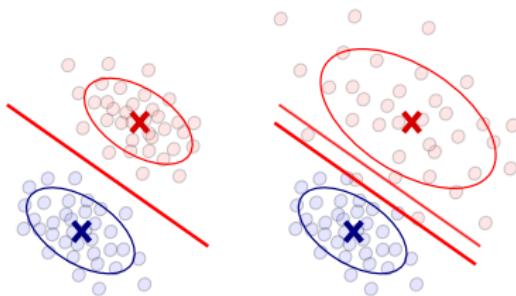


$\mathbf{S}_k$  is the **covariance matrix** of class  $\mathcal{C}_k$ :

$$\mathbf{S}_k = \frac{1}{N_k - 1} \sum_{n \in \mathcal{C}_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T$$

(symmetric and positive semi-definite)

## Reminder: Covariance Matrix



$\mathbf{S}_k$  is the **covariance matrix** of class  $\mathcal{C}_k$ :

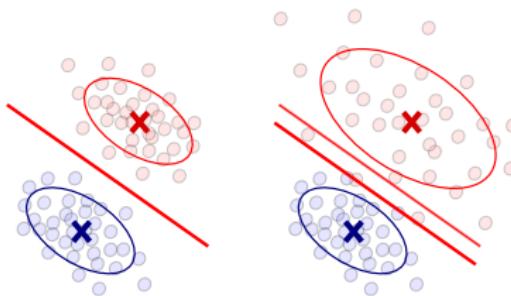
$$\mathbf{S}_k = \frac{1}{N_k - 1} \sum_{n \in \mathcal{C}_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T$$

(symmetric and positive semi-definite)

Assuming equally large classes, the **total within-class covariance matrix** is:

$$\mathbf{S}_W = \frac{1}{2}(\mathbf{S}_1 + \mathbf{S}_2) \quad \text{(for two classes; sometimes factor } \frac{1}{2} \text{ is omitted)}$$

## Reminder: Covariance Matrix



$\mathbf{S}_k$  is the **covariance matrix** of class  $\mathcal{C}_k$ :

$$\mathbf{S}_k = \frac{1}{N_k - 1} \sum_{n \in \mathcal{C}_k} (\mathbf{x}_n - \mathbf{m}_k)(\mathbf{x}_n - \mathbf{m}_k)^T$$

(symmetric and positive semi-definite)

Assuming equally large classes, the **total within-class covariance matrix** is:

$$\mathbf{S}_W = \frac{1}{2}(\mathbf{S}_1 + \mathbf{S}_2) \quad \text{(for two classes; sometimes factor } \frac{1}{2} \text{ is omitted)}$$

$\mathbf{S}_B$  is the **between-class covariance matrix**:

$$\mathbf{S}_B = (\mathbf{m}_2 - \mathbf{m}_1)(\mathbf{m}_2 - \mathbf{m}_1)^T \quad \text{(for two classes)}$$

# Lecture Overview

- 1 Brief Recapitulation: Supervised Classification
- 2 Motivation: Linear Discriminant Functions
- 3 LDA Assumptions and Data Scenarios
- 4 Geometric Interpretation
- 5 How to Derive the Parameters...
- 6 Wrapup: Summary, Related Topics, Preview

# Four Scenarios of Class Distributions

(draw here)

## Assumption About the Data

Linear discriminant functions make an important assumption about the data (otherwise, they may fail or underperform!):

# Assumption About the Data

Linear discriminant functions make an important assumption about the data (otherwise, they may fail or underperform!):

A1: The data distributions of both classes are Gaussian (normally distributed)

- Data of each class  $k$  can be described by a covariance matrix  $\mathbf{S}_k$  (also called: scatter matrix)

# Assumption About the Data

Linear discriminant functions make an important assumption about the data (otherwise, they may fail or underperform!):

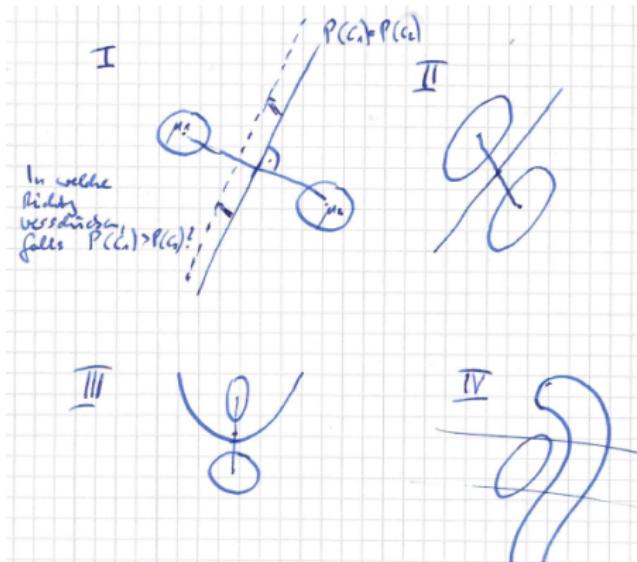
A1: The data distributions of both classes are Gaussian (normally distributed)

- Data of each class  $k$  can be described by a covariance matrix  $\mathbf{S}_k$  (also called: scatter matrix)

A2: The covariance matrices of the classes are equal, i.e.  $\mathbf{S}_k = \mathbf{S}$

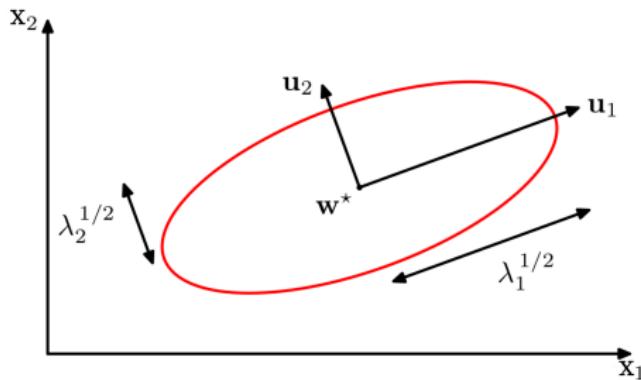
- If you know the covariance matrix of one class, you know also the covariance matrix of every other class.

# Four Scenarios of Class Distributions



(draw here)

## Reminder: Eigenvalue Decomposition



Eigenvalue decomposition of covariance matrix:  $\mathbf{S}\mathbf{u}_i = \lambda_i \mathbf{u}_i$

- requires full rank
- creates novel basis (orthogonal eigenvectors)
- eigenvectors can be sorted according to eigenvalues
- observe the relation between eigenvectors and variance of a normal distribution
- allows for visual interpretation of covariance matrices

# Lecture Overview

- 1 Brief Recapitulation: Supervised Classification
- 2 Motivation: Linear Discriminant Functions
- 3 LDA Assumptions and Data Scenarios
- 4 Geometric Interpretation
- 5 How to Derive the Parameters...
- 6 Wrapup: Summary, Related Topics, Preview

# Linear Discriminant Function

Using the above assumptions and  $k = 2$  classes, we obtain this form of a linear discriminant function:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

- $\mathbf{w}^T$  denotes the **transpose** of  $\mathbf{w}$ .
- $\mathbf{w}^T \mathbf{x}$  is the **inner product** between vectors  $\mathbf{w}$  and  $\mathbf{x}$ .

Terminology:

- $\mathbf{w}$  is called **weight vector**, with  $\mathbf{w} \in \mathbb{R}^{D \times 1}$
- $\mathbf{w}$  *projects* the data point  $\mathbf{x}$  to a scalar

# Linear Discriminant Function

Using the above assumptions and  $k = 2$  classes, we obtain this form of a linear discriminant function:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

- $\mathbf{w}^T$  denotes the **transpose** of  $\mathbf{w}$ .
- $\mathbf{w}^T \mathbf{x}$  is the **inner product** between vectors  $\mathbf{w}$  and  $\mathbf{x}$ .

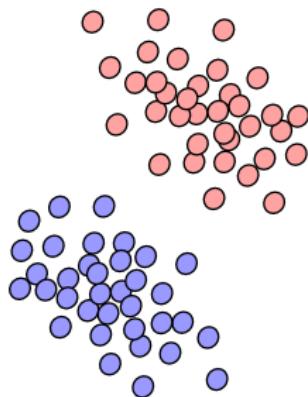
Terminology:

- $\mathbf{w}$  is called **weight vector**, with  $\mathbf{w} \in \mathbb{R}^{D \times 1}$
- $\mathbf{w}$  *projects* the data point  $\mathbf{x}$  to a scalar
- $b$  is called the *bias* or *threshold weight*.
- ( $b$  is also called  $w_0$ )

# Geometric Interpretation I

Using the above assumptions and  $k = 2$  classes, we obtain this form of a linear discriminant function:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$



# Geometric Interpretation I

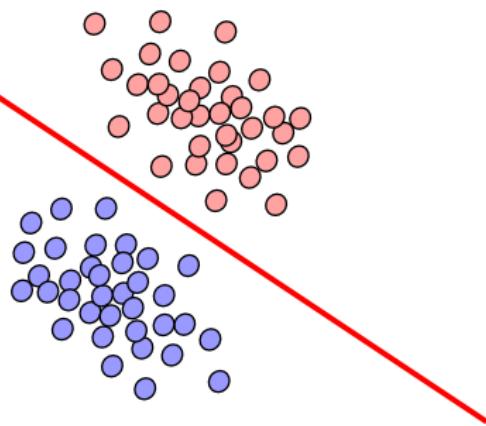
Using the above assumptions and  $k = 2$  classes, we obtain this form of a linear discriminant function:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

$$f(\mathbf{x}) > 0$$

$$f(\mathbf{x}) = 0$$

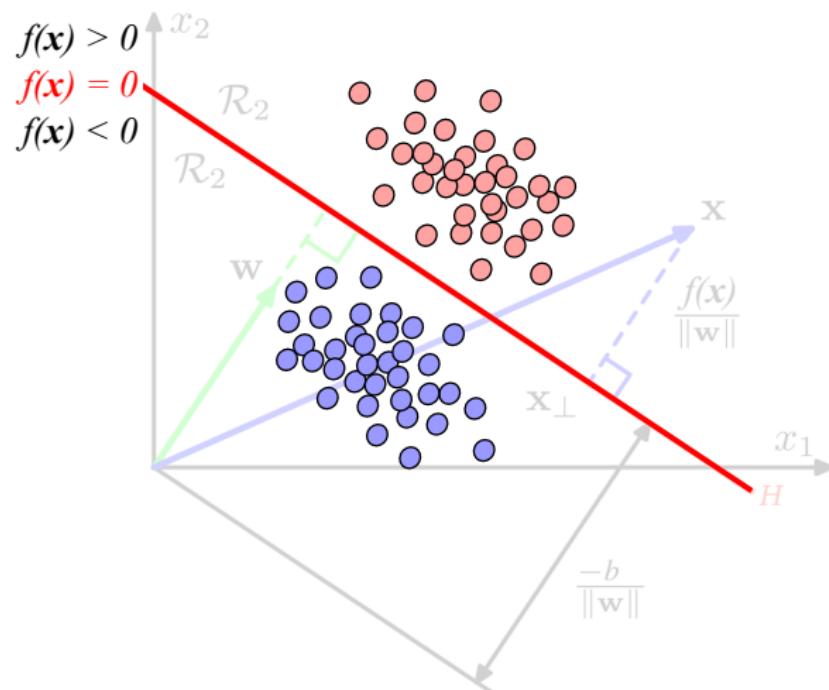
$$f(\mathbf{x}) < 0$$



# Geometric Interpretation I

Using the above assumptions and  $k = 2$  classes, we obtain this form of a linear discriminant function:

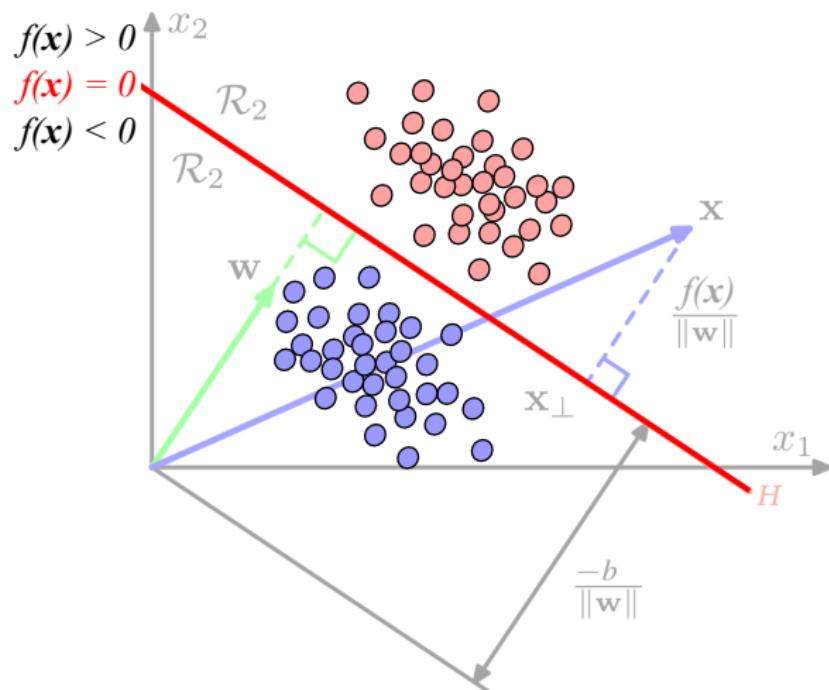
$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$



# Geometric Interpretation I

Using the above assumptions and  $k = 2$  classes, we obtain this form of a linear discriminant function:

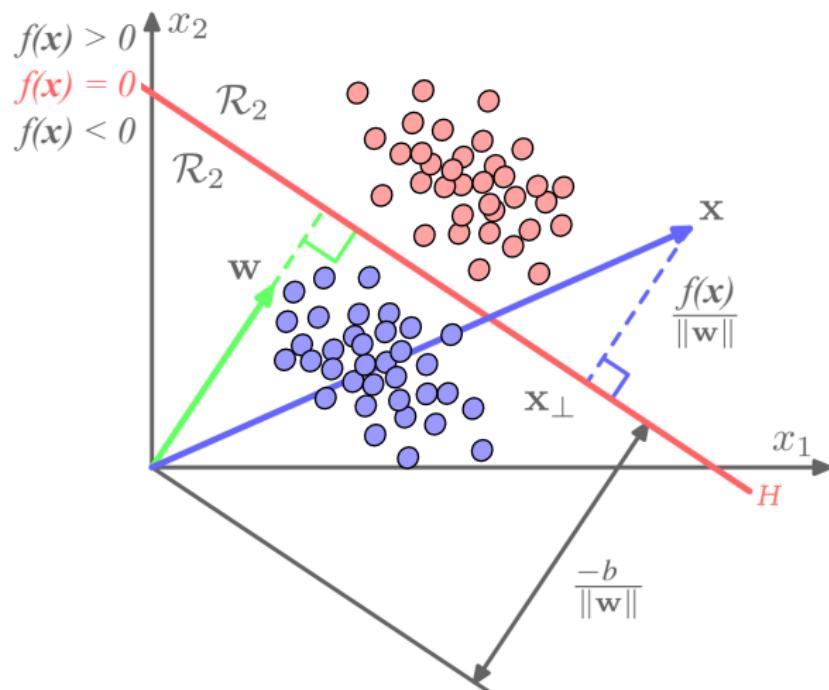
$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$



# Geometric Interpretation I

Using the above assumptions and  $k = 2$  classes, we obtain this form of a linear discriminant function:

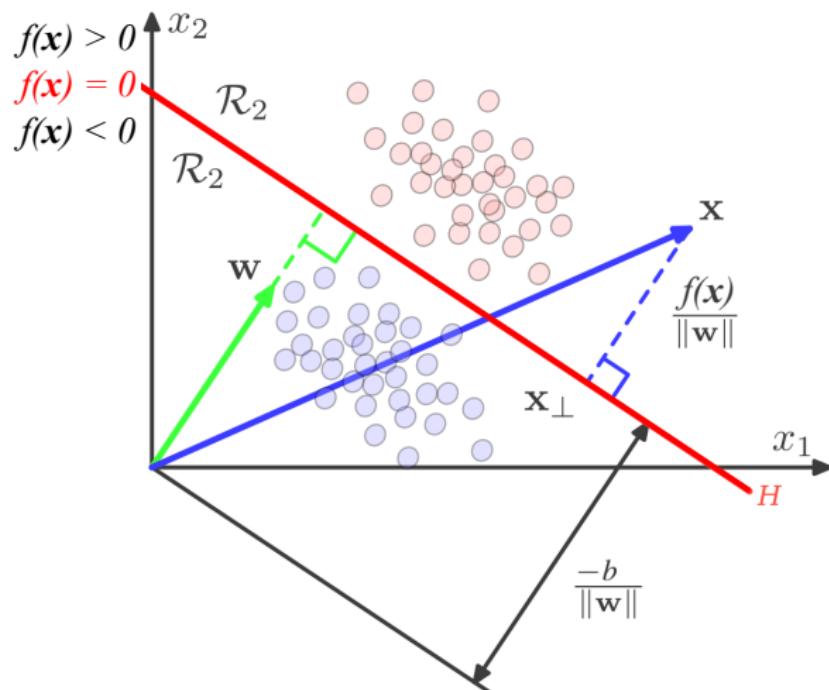
$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$



# Geometric Interpretation I

Using the above assumptions and  $k = 2$  classes, we obtain this form of a linear discriminant function:

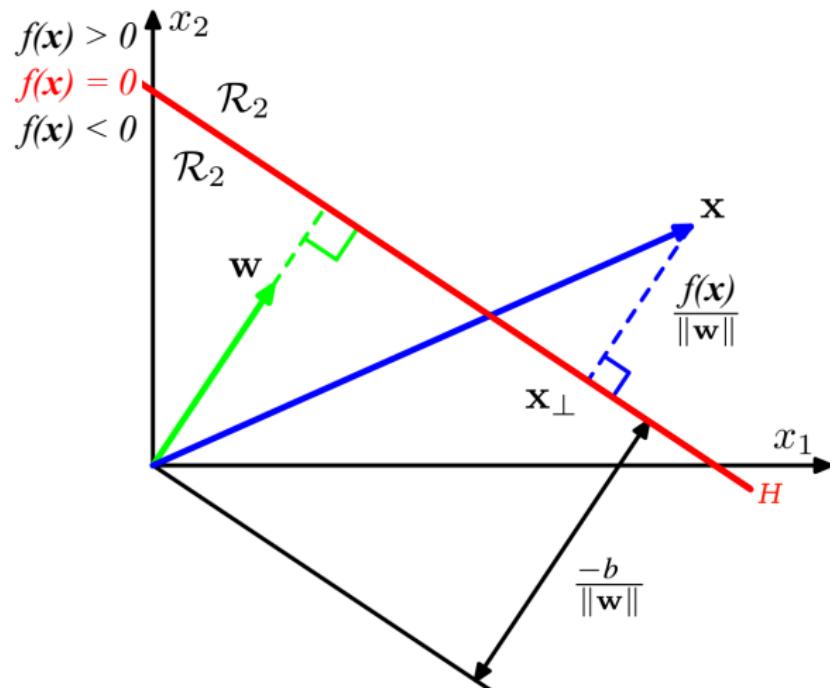
$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$



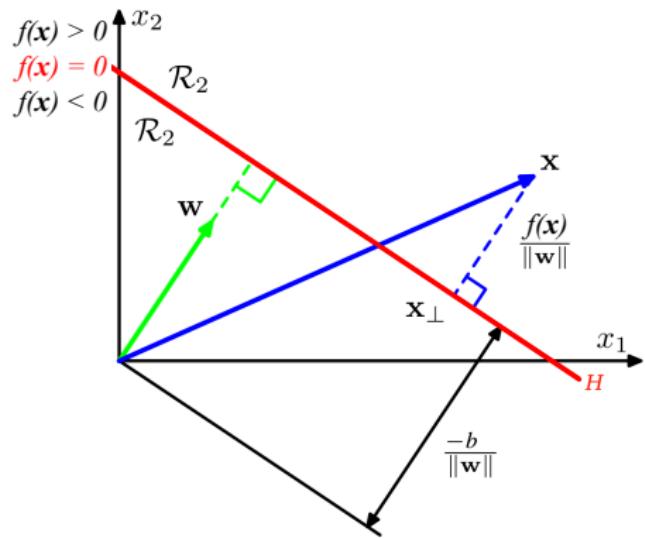
# Geometric Interpretation I

Using the above assumptions and  $k = 2$  classes, we obtain this form of a linear discriminant function:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

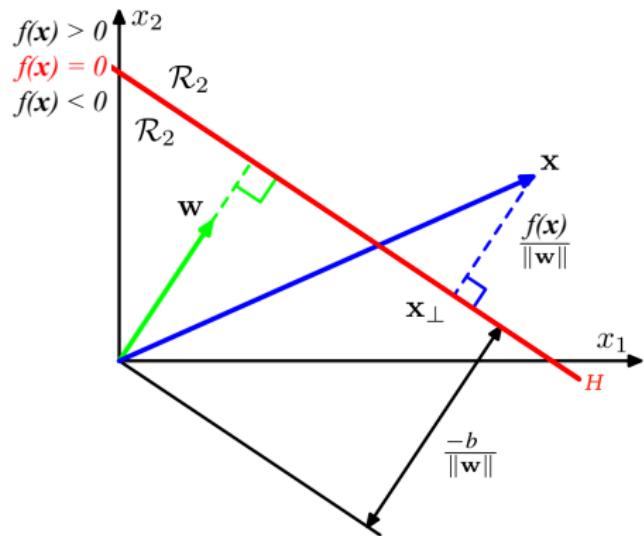


# Geometric Interpretation II



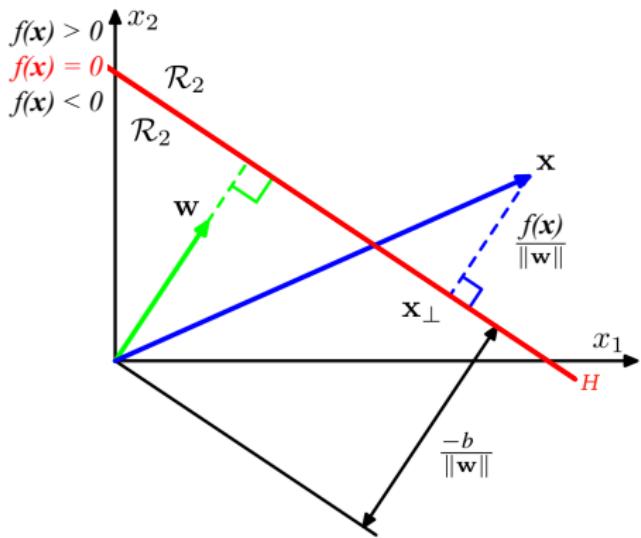
- $\mathbf{w}$  is perpendicular to the decision boundary  $H$
- In general: decision boundary is a  $(D-1)$ -dimensional hyperplane.

# Geometric Interpretation II



- $\mathbf{w}$  is perpendicular to the decision boundary  $H$
- In general: decision boundary is a  $(D-1)$ -dimensional hyperplane.
- Displacement of  $H$  from origin is controlled by bias  $b$ .

# Geometric Interpretation II



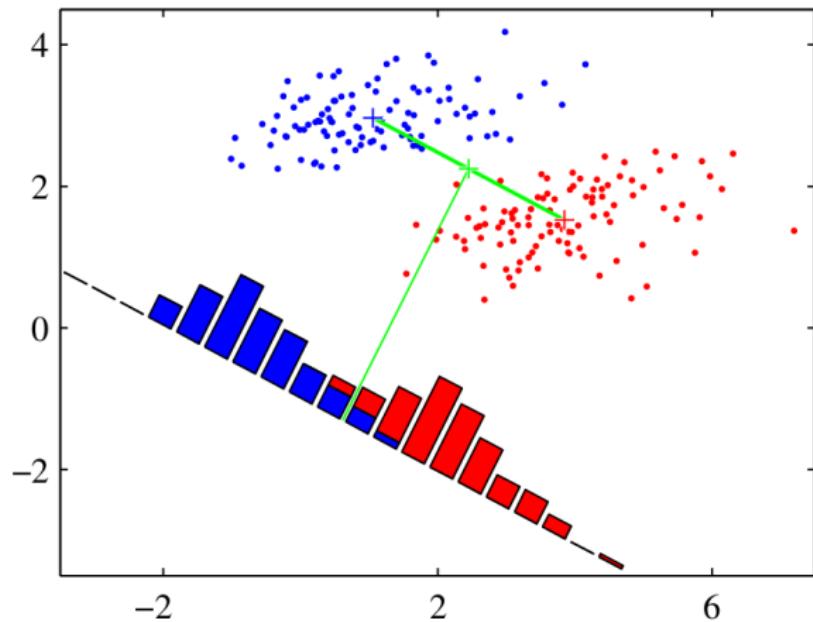
- $\mathbf{w}$  is perpendicular to the decision boundary  $H$
- In general: decision boundary is a  $(D-1)$ -dimensional hyperplane.
- Displacement of  $H$  from origin is controlled by bias  $b$ .
- Signed orthogonal distance of an arbitrary point  $\mathbf{x}$  to  $H$  is determined by  $\frac{f(\mathbf{x})}{\|\mathbf{w}\|}$

# Lecture Overview

- 1 Brief Recapitulation: Supervised Classification
- 2 Motivation: Linear Discriminant Functions
- 3 LDA Assumptions and Data Scenarios
- 4 Geometric Interpretation
- 5 How to Derive the Parameters...
- 6 Wrapup: Summary, Related Topics, Preview

## Two Example Projections

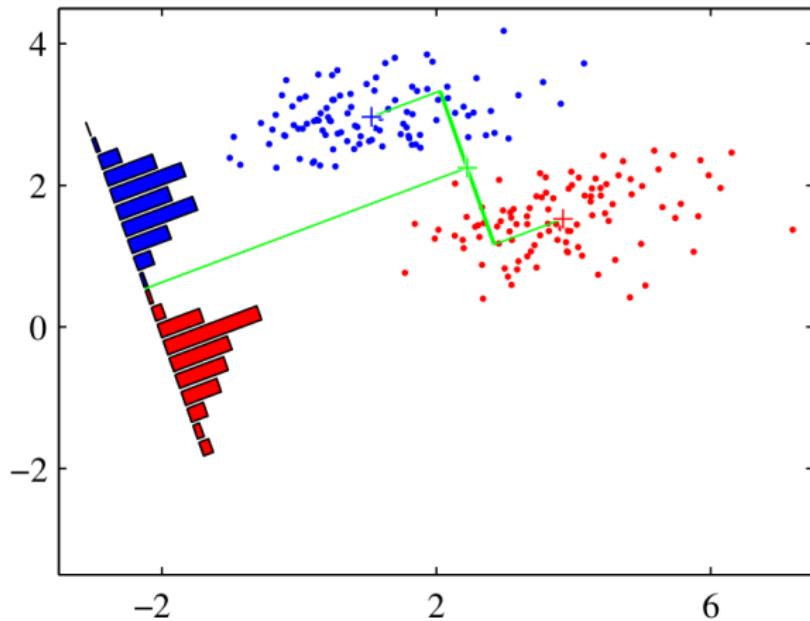
$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$



- Sub-optimal:  
large overlap of  
projected  
distributions

## Two Example Projections

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

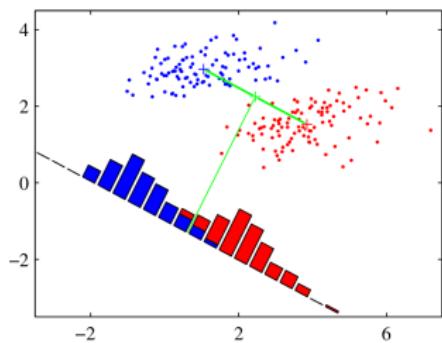


- Pretty good:  
small overlap of  
projected  
distributions

# What is a Good Projection $\mathbf{w}$ ?

Find a  $\mathbf{w}$  such, that the projected data maximizes the [Fisher criterion](#):

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \quad \operatorname{argmax}_{\mathbf{w}} \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$

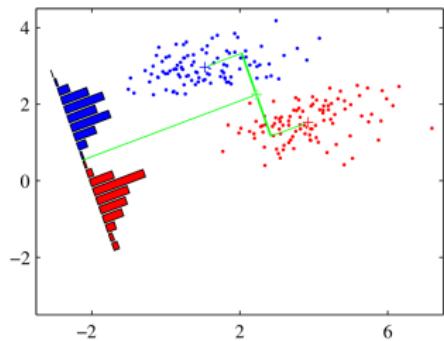


- Projected means  $m_k = \mathbf{w}^T \mathbf{m}_k$  should have large distance:  
Thus maximize  $(m_2 - m_1)^2$
- For the projected data, the within-class variance  $s_k^2$  of every class  $\mathcal{C}_k$  should be small.  
Thus minimize  $(s_1^2 + s_2^2)$

# What is a Good Projection $\mathbf{w}$ ?

Find a  $\mathbf{w}$  such, that the projected data maximizes the [Fisher criterion](#):

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} \quad \operatorname{argmax}_{\mathbf{w}} \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2}$$



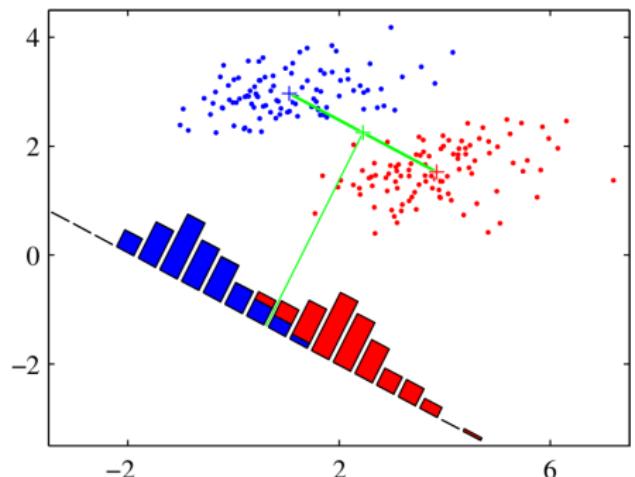
- Projected means  $m_k = \mathbf{w}^T \mathbf{m}_k$  should have large distance:  
Thus maximize  $(m_2 - m_1)^2$
- For the projected data, the within-class variance  $s_k^2$  of every class  $\mathcal{C}_k$  should be small.  
Thus minimize  $(s_1^2 + s_2^2)$

# Fisher Criterion with Explicit $\mathbf{w}$

More useful: Fisher criterion formulated in the original space with explicit projections by  $\mathbf{w}$ .

(See [Bishop, Section 4.1.4] for details on the conversion steps)

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}}$$



- How should the covariance matrices behave? 

## Fisher Criterion with Explicit $\mathbf{w}$

Search the maximum of  $J(\mathbf{w})$  by differentiation with respect to  $\mathbf{w}$ . Make use of the two assumptions [Bishop, Section 4.1.4] to yield:

$$\mathbf{w} = \mathbf{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

and

$$b = -\frac{1}{2}\mathbf{w}(\mathbf{m}_1 + \mathbf{m}_2)$$

- Nice: weight vector  $\mathbf{w}$  and bias  $b$  can be computed **analytically**!  
(Key: within-class covariance matrices are identical and Gaussian.)
- Please observe: the total within-class covariance matrix  $\mathbf{S}_W$  needs to be estimated. (This can be tricky, see assignment 2)



How many free parameters need to be determined for  $\mathbf{S}_W$ ?  
Please vote:  $D$ ,  $N$ ,  $D(D + 1)/2$ ,  $D^2$

# Fisher Criterion with Explicit $w$

Search the maximum of  $J(w)$  by differentiation with respect to  $w$ . Make use of the two assumptions [Bishop, Section 4.1.4] to yield:

$$w = S_W^{-1}(m_2 - m_1)$$

and

$$b = -\frac{1}{2}w(m_1 + m_2)$$

- Nice: weight vector  $w$  and bias  $b$  can be computed **analytically**!  
(Key: within-class covariance matrices are identical and Gaussian.)
- Please observe: the total within-class covariance matrix  $S_W$  needs to be estimated. (This can be tricky, see assignment 2)



How many free parameters need to be determined for  $S_W$ ?

Please vote:  $D$ ,  $N$ ,  $D(D + 1)/2$ ,  $D^2$

- Please discuss with your neighbour for 1 min and vote again.

## Fisher Criterion with Explicit $\mathbf{w}$

Looking for the maximum of  $J(\mathbf{w})$ , differentiation with respect to  $\mathbf{w}$  and use of our assumptions yields [Bishop, Section 4.1.4]

$$\mathbf{w} = \mathbf{S}_W^{-1}(\mathbf{m}_2 - \mathbf{m}_1)$$

and

$$b = -\frac{1}{2}\mathbf{w}(\mathbf{m}_1 + \mathbf{m}_2)$$

Results:

- Con: covariance matrix needs to be **inverted!**  
(Runtime? Stability? Approximations like pseudo-inverse?)

# Typical Use of Linear Discriminant Analysis



(image: courtesy of Robert Bosch)

- As a first shot method.
- When the noise model of your sensors is known.
- When the class means are informative.

# Lecture Overview

- 1 Brief Recapitulation: Supervised Classification
- 2 Motivation: Linear Discriminant Functions
- 3 LDA Assumptions and Data Scenarios
- 4 Geometric Interpretation
- 5 How to Derive the Parameters...
- 6 Wrapup: Summary, Related Topics, Preview

## Summary by learning goals

Having heard this lecture, you can now ...

- describe a classification problem and explain linear separability
- give different criteria for good class separability

# Summary by learning goals

Having heard this lecture, you can now ...

- describe a classification problem and explain linear separability
- give different criteria for good class separability
- formulate the decision function for a linear discriminant
- explain the meaning and characteristics of the weight vector, decision boundary, covariance matrices, the bias etc.

## Summary by learning goals

Having heard this lecture, you can now ...

- describe a classification problem and explain linear separability
- give different criteria for good class separability
- formulate the decision function for a linear discriminant
- explain the meaning and characteristics of the weight vector, decision boundary, covariance matrices, the bias etc.
- explain (different) assumptions for linear discriminant analysis

# Summary by learning goals

Having heard this lecture, you can now ...

- describe a classification problem and explain linear separability
- give different criteria for good class separability
- formulate the decision function for a linear discriminant
- explain the meaning and characteristics of the weight vector, decision boundary, covariance matrices, the bias etc.
- explain (different) assumptions for linear discriminant analysis
- formulate the Fisher criterion in the feature space and the projected space

## Summary by learning goals

Having heard this lecture, you can now . . .

- describe a classification problem and explain linear separability
- give different criteria for good class separability
- formulate the decision function for a linear discriminant
- explain the meaning and characteristics of the weight vector, decision boundary, covariance matrices, the bias etc.
- explain (different) assumptions for linear discriminant analysis
- formulate the Fisher criterion in the feature space and the projected space
- derive a linear discriminant model from given data and apply it to new data

# Related Topics and Further Reading

Linear discriminant functions generalize to multiple classes

- Worst solution: one-against-rest
- Slightly better: one-against-one
- Best option for  $k$  classes: inherent multiclass formulation  
$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$
- [Bishop, Section 4.2.1], [Duda, Hart, Stork, Section 5.2.2]

# Related Topics and Further Reading

Linear discriminant functions generalize to multiple classes

- Worst solution: one-against-rest
- Slightly better: one-against-one
- Best option for  $k$  classes: inherent multiclass formulation  
$$y_k(\mathbf{x}) = \mathbf{w}_k^T \mathbf{x} + w_{k0}$$
- [Bishop, Section 4.2.1], [Duda, Hart, Stork, Section 5.2.2]

Linear discriminant functions under the easier condition  $S_k = \sigma^2 \mathbb{I}$

- Search for nearest class mean
- Decision boundary is perpendicular to  $m_2 - m_1$
- [Duda, Hart, Stork, Section 2.6]

## Related Topics and Further Reading

Linear discriminant functions under the **harder condition**  $S_k = \text{arbitrary}$  normal distribution

- covariance matrices are different for each class
- decision surfaces are *hyperquadratics*
- Decision regions are not necessarily simply connected any more
- [Duda,Hart,Stork, Chapter 2.6]

## Related Topics and Further Reading

Linear discriminant functions under the **harder condition**  $S_k = \text{arbitrary normal distribution}$

- covariance matrices are different for each class
- decision surfaces are *hyperquadratics*
- Decision regions are not necessarily simply connected any more
- [Duda,Hart,Stork, Chapter 2.6]

There are **various LDA formulations** e.g.:

- as a gradient descent problem
- as an eigenvalue problem
- [Duda,Hart,Stork, Section 2.6]
- as an incremental LDA for online learning [Aliyari Ghassabeh et al. (2015), Pattern Recognition 48(6)]

# Related Topics and Further Reading

Bias  $b$  can be removed by using **augmented vectors**:

- slightly enlarging dimensionality:  $D \rightarrow D+1$
- augmented feature vector:  $\mathbf{x} \rightarrow \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$
- augmented weight vector:  $\mathbf{w} \rightarrow \begin{bmatrix} b \\ \mathbf{w} \end{bmatrix}$
- or  $\mathbf{w} \rightarrow \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}$
- decision boundary will pass through origin
- [Duda,Hart,Stork, Chapter 5.3]

# Related Topics and Further Reading

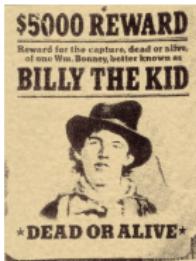
Generalized linear discriminant functions allow for a non-linear feature pre-processing  $\phi$  by

$$f(\mathbf{x}) = \sum_{i=1}^D \mathbf{w}_i \phi(\mathbf{x})$$

but are still linear in  $\phi(\mathbf{x})$

- non-linearity in the features, not the classification method
- powerful, if expert knowledge about features is available
- → relation to pre-processing (see ML design cycle)
- may enlarge the dimensionality
- [Duda,Hart,Stork, Chapter 5.3]

# Hint for the Exam: Overview Sheet for each Method



Typical use case for the method

- supervised / unsupervised / reinforcement learning / dimensionality reduction / ...

Assumptions made by the method about the data, e.g.

- Gaussian data
- equal covariances for each class

Runtime + memory requirements:

- $O(N)$  – data points N
- $O(D)$  – dimensionality D
- at training time
- at recall time (using the model)

Strengths and weaknesses e.g.

- very sensitive to outliers
- can be used for online learning
- difficult to find good hyperparameters
- easy to interpret a trained model

Related methods, improvements

Regularization approaches, number of free parameters...

# Preview of Assignment 2

Objectives of assignment 2:

- Polish up some math concepts
- Get acquainted with covariance matrices (visualize them, analyze their eigenvalue spectrum)

# Preview of Assignment 2

## Objectives of assignment 2:

- Polish up some math concepts
- Get acquainted with covariance matrices (visualize them, analyze their eigenvalue spectrum)
- Implement LDA for a two-class problem, test it
- Experiment with the assumptions of LDA, break it

# Preview of Assignment 2

## Objectives of assignment 2:

- Polish up some math concepts
- Get acquainted with covariance matrices (visualize them, analyze their eigenvalue spectrum)
- Implement LDA for a two-class problem, test it
- Experiment with the assumptions of LDA, break it
- Nonlinear feature preprocessing to realize simple *generalized linear discriminant functions*.

# Lecture 3: Linear Regression

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



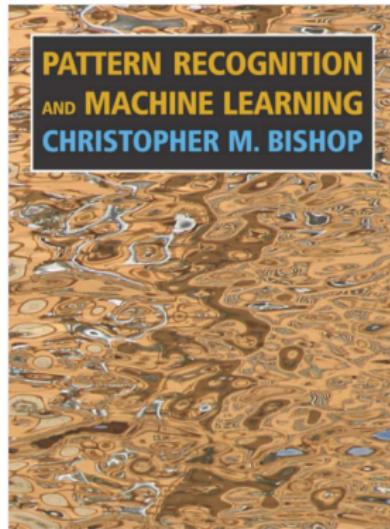
# Lecture Overview

- 1 Brief Recapitulation: Supervised Regression
- 2 The Linear Regression Model
- 3 Assumptions and Data Scenarios
- 4 How to Derive the Parameters...
- 5 Wrapup: Summary, Related Topics, Preview
- 6 Let's Work on Assignment 2

# Lecture Overview

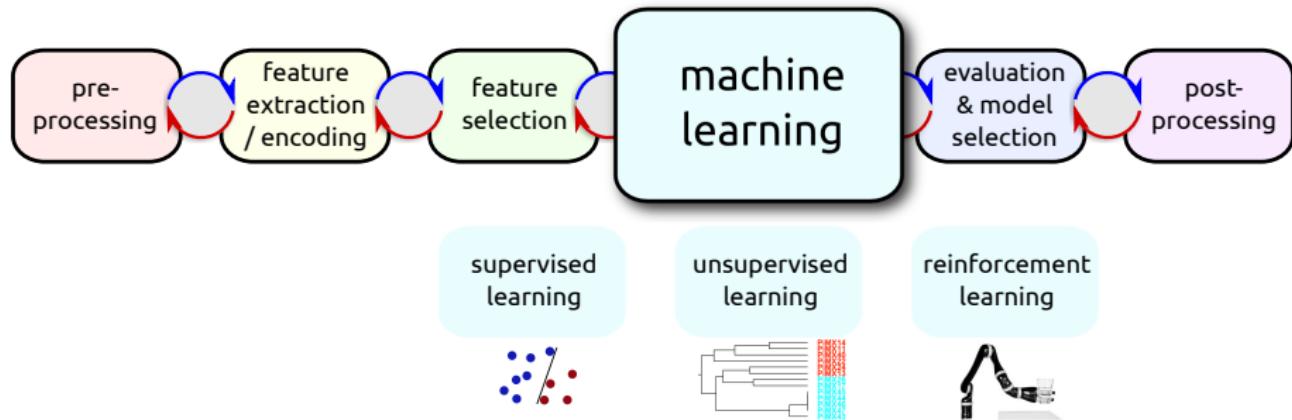
- 1 Brief Recapitulation: Supervised Regression
- 2 The Linear Regression Model
- 3 Assumptions and Data Scenarios
- 4 How to Derive the Parameters...
- 5 Wrapup: Summary, Related Topics, Preview
- 6 Let's Work on Assignment 2

# Additional Literature



This book covers linear regression models (today's lecture) and linear discriminant functions (last lecture) nicely, however, it has a dominantly probabilistic approach...

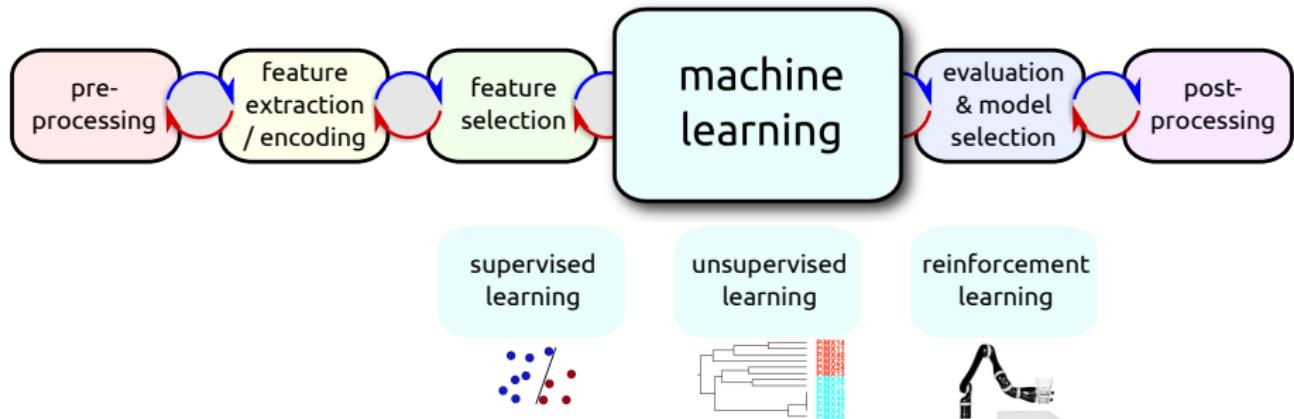
# Reminder: ML Design Cycle



Linear regression models are for **supervised regression**:

- Use past experience to predict the future

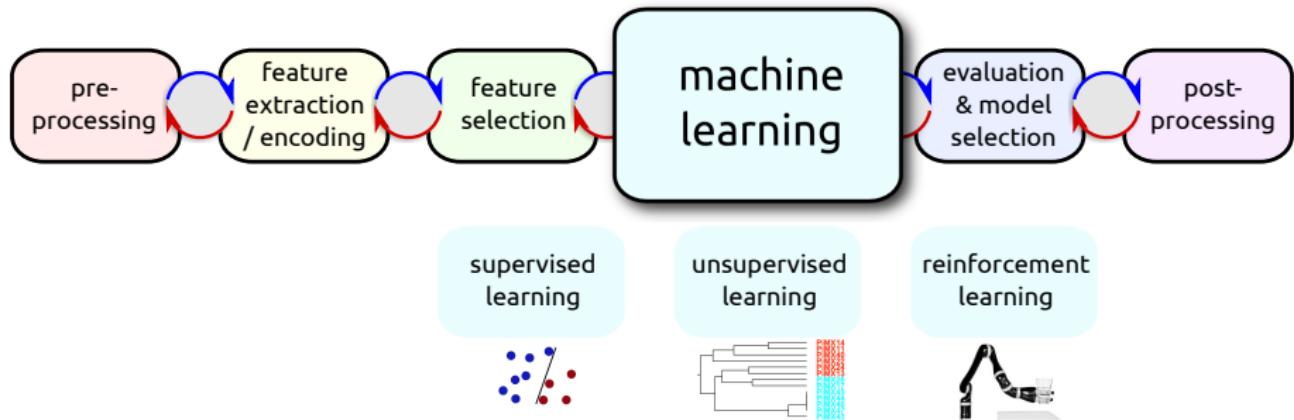
# Reminder: ML Design Cycle



Linear regression models are for **supervised regression**:

- Use past experience to predict the future
- Use **labelled data points**  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$

# Reminder: ML Design Cycle



Linear regression models are for **supervised regression**:

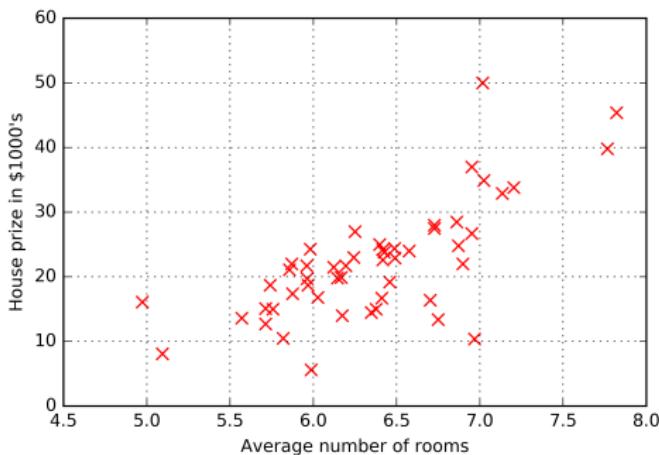
- Use past experience to predict the future
- Use **labelled data points**  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$
- Train a **model** which can predict the label  $y_{N+1}$  of a new data point  $\mathbf{x}_{N+1}$

# Supervised Learning: A Simple Regression Example

## Predicting housing prices

- Let's say we only know the average number of rooms in an area
- And we'd like to predict the prize for a house in that area
- One data point: number of rooms  $x_i$  and its prize  $y_i$  (in 1000's)

avg. # rooms	$y_i$
6.575	24
6.377	21.6
5.57	34.7
5.713	33.4
7.024	36.2
5.963	28.7
5.741	22.9
6.417	27.1
...	...

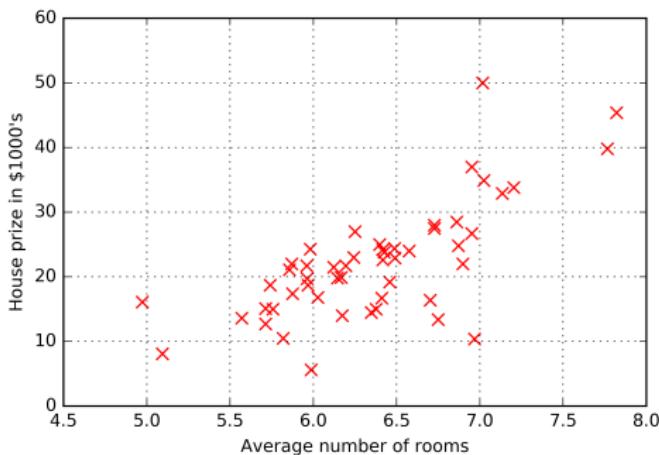


# Supervised Learning: A Simple Regression Example

Predicting housing prices

- Let's say we only know the average number of rooms in an area
- And we'd like to predict the prize for a house in that area
- One data point: number of rooms  $x_i$  and its prize  $y_i$  (in 1000's)

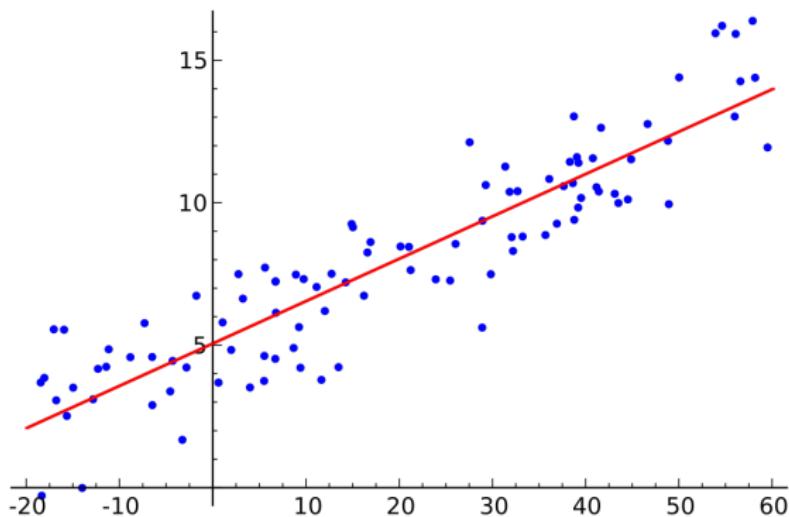
avg. # rooms	$y_i$
6.575	24
6.377	21.6
5.57	34.7
5.713	33.4
7.024	36.2
5.963	28.7
5.741	22.9
6.417	27.1
...	...



Your ideas for other input variables?

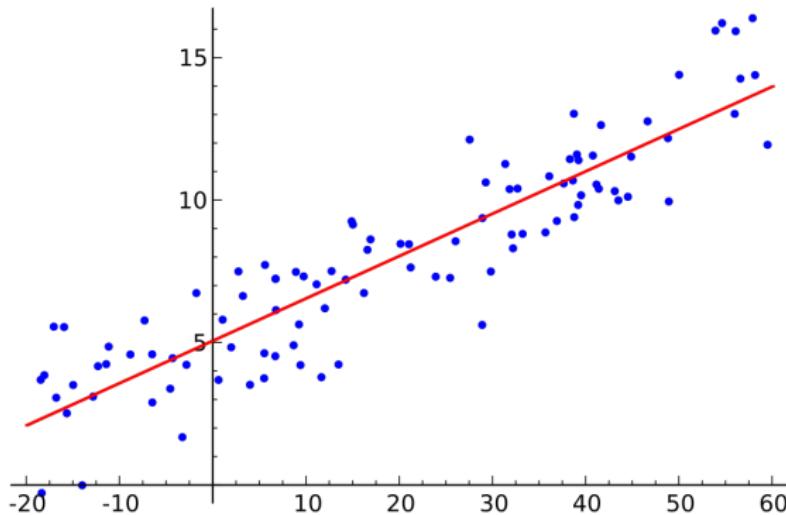


## Reminder: Terminology



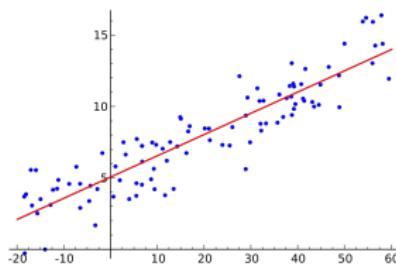
- A data point  $x_i$  is a column vector in  $\mathbb{R}^D$   
In the literature,  $x_i$  are called "regressors", "covariates" or  
"independent" / "explanatory" / "exogenous" / "explanatory" /  
"input" / "predictor" variables

## Reminder: Terminology



- The label  $y$  a scalar value,  $y \in \mathbb{R}$   
In the literature,  $y$  is called "regressand", or "endogenous" / "response" / "criterion" / "dependent" variable.

# Regression Terminology

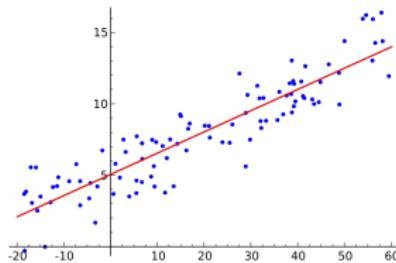


- A data point  $\mathbf{x}_i$  is a column vector in  $\mathbb{R}^D$
- One label  $y_i$  a scalar value,  $y_i \in \mathbb{R}$

We are given a labeled data set of  $N$  examples:

- $\mathbf{X}$  is a  $N \times D$  matrix containing the continuous **feature** values.  
( $\mathbf{X}$  contains one transposed data point  $\mathbf{x}_i^T$  per row.)
- $\mathbf{y}$  is a  $N \times 1$  vector containing the continuous **labels**.
- Beware: Bishop uses  $t_i$  instead of  $y_i$  to denote labels.

# Regression Terminology



- In case  $\mathbf{x}$  is one-dimensional: *simple* linear regression.
- General case: a data point  $\mathbf{x}_i$  is a column vector in  $\mathbb{R}^D$
- One label  $y_i$  a scalar value,  $y_i \in \mathbb{R}$

Remark:

*multivariable* linear regression  
or *multiple* linear regression  
 $y_i \in \mathbb{R}$

$\neq$   
*multivariate* linear regression  
 $y_i \in \mathbb{R}^D$ , with  $D > 1$

# Regression: The Workflow

- Learn a function  $f(\mathbf{x})$ , which shall predict the label  $y$  based on a data point  $\mathbf{x}$  **as good as possible**.

# Regression: The Workflow

- Learn a function  $f(\mathbf{x})$ , which shall predict the label  $y$  based on a data point  $\mathbf{x}$  **as good as possible**.
- Once the function  $f(\mathbf{x})$  has been learned, you can infer the continuous label  $y$  for a novel input vector  $\mathbf{x}$  by simply evaluating the function  $f(\mathbf{x})$ .

# Lecture Overview

- 1 Brief Recapitulation: Supervised Regression
- 2 The Linear Regression Model
- 3 Assumptions and Data Scenarios
- 4 How to Derive the Parameters...
- 5 Wrapup: Summary, Related Topics, Preview
- 6 Let's Work on Assignment 2

# Linear Regression: The Basic Idea

The simplest linear model of regression is:

$$f(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + \dots + w_D x_D$$

where  $w_0$  is a fixed offset, called the *bias*,  
 $\mathbf{w}$  is the weight vector or parameter vector,  
and data point  $\mathbf{x} = (x_1, \dots, x_D)^T$  contains the input variables.

Characteristics:

- Linear function of the weights / parameters  $w_i$
- linear function of the input dimensions / variables  $x_i$
- → **significant limitation** of the model!

# Linear Regression: The Basic Idea

Which relationships between  $x$  and  $y$  can **not** be described using this simple model?



(draw here)

# Linear Regression with Basis Functions

This limitation can partially be removed by considering linear combinations of fixed **nonlinear functions** of the input variables:

$$f(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

where  $\phi_j(\mathbf{x})$  are known as *basis functions*.

Characteristics of this *enhanced* linear regression model:

- $f(\mathbf{x}, \mathbf{w})$  is still a linear function of the weights / parameters  $w_i$
- But:  $f(\mathbf{x}, \mathbf{w})$  is a **nonlinear** function of the input dimensions / variables  $x_i$

# Linear Regression with Basis Functions

This limitation can partially be removed by considering linear combinations of fixed **nonlinear functions** of the input variables:

$$f(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

where  $\phi_j(\mathbf{x})$  are known as *basis functions*.

Characteristics of this *enhanced* linear regression model (cont.):

- Adding basis functions may enlarge the dimensionality

# Linear Regression with Basis Functions

This limitation can partially be removed by considering linear combinations of fixed **nonlinear functions** of the input variables:

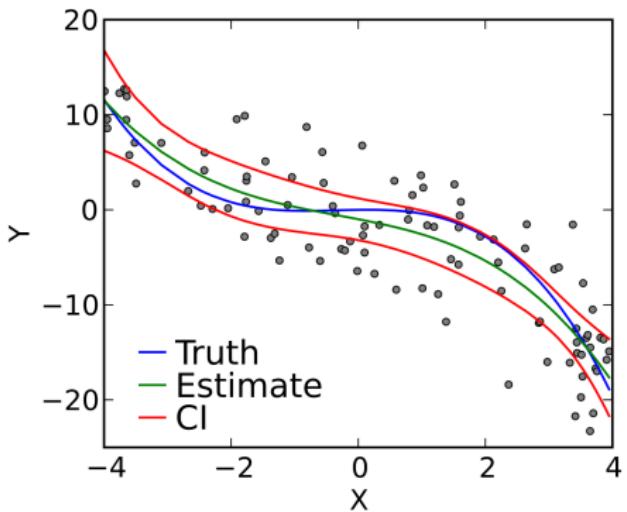
$$f(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

where  $\phi_j(\mathbf{x})$  are known as *basis functions*.

Characteristics of this *enhanced* linear regression model (cont.):

- Adding basis functions may enlarge the dimensionality
- The use of fixed basis functions corresponds to an earlier step in the ML pipeline

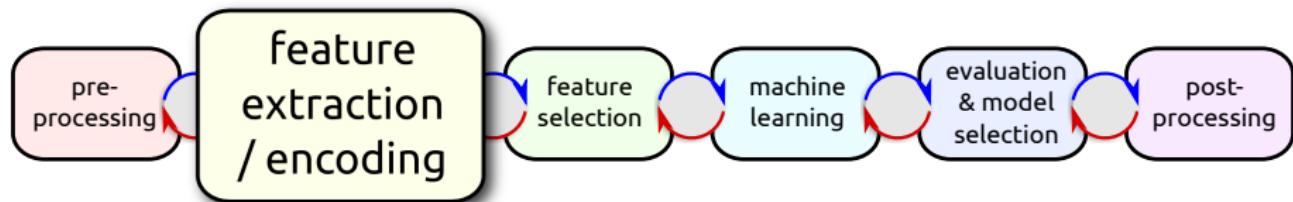
# Caution with Polynomial Regression / Basis Functions!



- Which risks are involved when using a **polynomial** basis function?



# Reminder: ML Design Cycle: Feature Extraction & Encoding



Combine features:

- We can use complex operations to combine features
- Combined features can be more expressive than their components  
*(use expert knowledge!)*

See [Bishop, Section 3.1] for common examples on non-linear basis functions.

# Linear Regression with Basis Functions

$$f(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

Please realize

- The index of  $j$  is now running up to  $M - 1$ , thus the number of free parameters is  $M$
- It is often convenient to define an additional dummy basis function  $\phi_0(\mathbf{x}) = 1$  to get rid of the bias  $w_0$

# Linear Regression with Basis Functions ... and Some Syntactic Sugar

Compare:

$$f(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x})$$

Using one extra basis function  $\phi_0(\mathbf{x}) = 1$  delivers a simpler form:

$$f(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

with  $\mathbf{w} = (w_0, \dots, w_{M-1})^T$  and  $\phi = (\phi_0, \dots, \phi_{M-1})^T$ .

In the literature, this is referred to as "augmented notation", and  $w_0$  as the "intercept".

# Interpretation of the Weight Parameters $w_i$

Please discuss: what is the interpretation of a single weight  $w_i$ ? 

# Interpretation of the Weight Parameters $w_i$

Please discuss: what is the interpretation of a single weight  $w_i$ ?

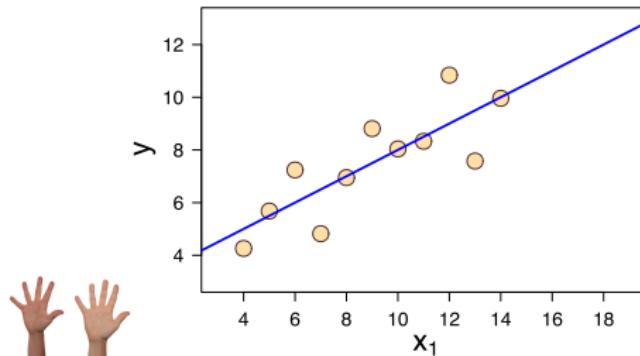
## Importance / Sensitivity:

Interpret a single weight  $w_i$  as a **partial derivative** with respect to the dependent variable  $x_i$ :

- Assume we can keep all other variables fixed – how much would the estimated value for  $\hat{y}$  change, if the value of  $x_1$  would be increased/decreased by a value of 1?

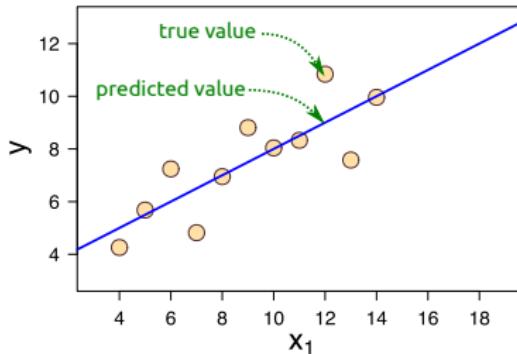
# Interpretation of the Quality of the Model

(omitting the basis functions but including augmented  $x$  and  $w$ ):  
How can we guess the quality of the regression model?



# Interpretation of the Quality of the Model

(omitting the basis functions but including augmented  $\mathbf{x}$  and  $\mathbf{w}$ ):  
How can we guess the quality of the regression model?



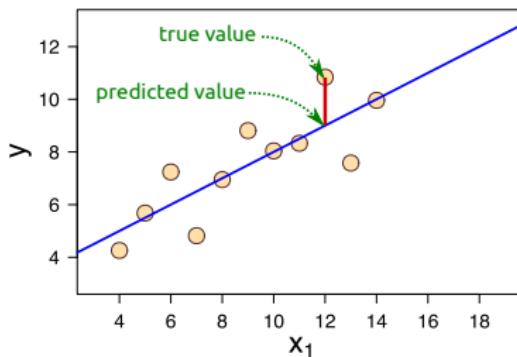
- **Error residuals:** as the model usually is not perfect, an estimated value  $\hat{y}$  may differ from the true  $y$ !
- Thus you may find the model expanded to :

$$f(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^D w_j \mathbf{x}^j + \varepsilon_j = \mathbf{w}^T \mathbf{x} + \varepsilon$$

(with  $\mathbf{x}^j$  denoting the j-th dimension of  $\mathbf{x}$ )

# Interpretation of the Quality of the Model

(omitting the basis functions but including augmented  $\mathbf{x}$  and  $\mathbf{w}$ ):  
How can we guess the quality of the regression model?



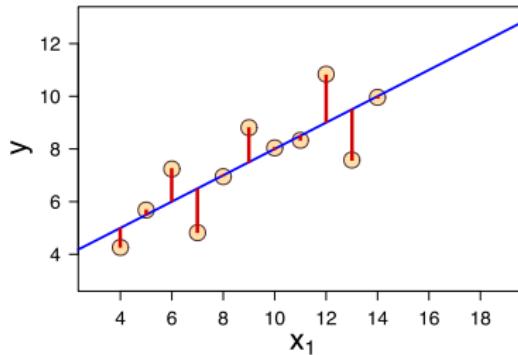
- **Error residuals:** as the model usually is not perfect, an estimated value  $\hat{y}$  may differ from the true  $y$ !
- Thus you may find the model expanded to :

$$f(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^D w_j \mathbf{x}^j + \varepsilon_j = \mathbf{w}^T \mathbf{x} + \varepsilon$$

(with  $\mathbf{x}^j$  denoting the j-th dimension of  $\mathbf{x}$ )

# Interpretation of the Quality of the Model

(omitting the basis functions but including augmented  $\mathbf{x}$  and  $\mathbf{w}$ ):  
How can we guess the quality of the regression model?



- **Error residuals:** as the model usually is not perfect, an estimated value  $\hat{y}$  may differ from the true  $y$ !
- Thus you may find the model expanded to :

$$f(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^D w_j \mathbf{x}^j + \varepsilon_j = \mathbf{w}^T \mathbf{x} + \varepsilon$$

(with  $\mathbf{x}^j$  denoting the j-th dimension of  $\mathbf{x}$ )

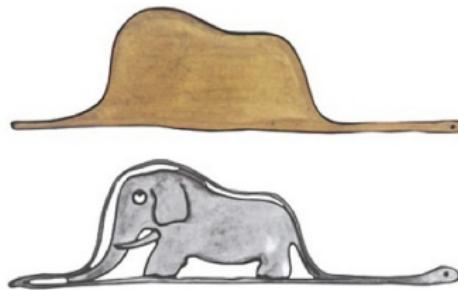
# How Can We Derive the Weight Parameters $w_i$ ?

Weight vector  $w$  can be derived in (at least) two manners:

# How Can We Derive the Weight Parameters $w_i$ ?

Weight vector  $w$  can be derived in (at least) two manners:

Option 1: Guess the entries of  $w$  and try to improve them iteratively  
(gradient descent, other heuristic methods)



Option 2: Determine the weights analytically.

# Lecture Overview

- 1 Brief Recapitulation: Supervised Regression
- 2 The Linear Regression Model
- 3 Assumptions and Data Scenarios
- 4 How to Derive the Parameters...
- 5 Wrapup: Summary, Related Topics, Preview
- 6 Let's Work on Assignment 2

# Assumptions Required for Analytical Solution

$$y_i = w_0 + w_1x_1 + \dots + w_Dx_D + \varepsilon_i$$

To formulate the analytical solution for the linear regression model requires to make **three assumptions** about the data and how it can be fitted. If violated, the model may fail or underperform. For  $i = 1 \dots N$  data points:

# Assumptions Required for Analytical Solution

$$y_i = w_0 + w_1 x_1 + \dots + w_D x_D + \varepsilon_i$$

To formulate the analytical solution for the linear regression model requires to make **three assumptions** about the data and how it can be fitted. If violated, the model may fail or underperform. For  $i = 1 \dots N$  data points:

- **A1:** The expected value of the residual errors is zero:

$$\forall i: E(\varepsilon_i) = 0$$

- **A2:** The residual errors are uncorrelated and share the same variance (across the input range of  $x$ ):

$$\forall i: \text{Var}(\varepsilon_i) = \sigma^2$$

- **A3:** The residual errors follow a normal distribution:

$$\varepsilon_i \sim N(0, \sigma^2)$$

# Assumptions Required for Analytical Solution

$$y_i = w_0 + w_1 x_1 + \dots + w_D x_D + \varepsilon_i$$

To formulate the analytical solution for the linear regression model requires to make **three assumptions** about the data and how it can be fitted. If violated, the model may fail or underperform. For  $i = 1 \dots N$  data points:

- **A1:** The expected value of the residual errors is zero:

$$\forall i: E(\varepsilon_i) = 0$$

- **A2:** The residual errors are uncorrelated and share the same variance (across the input range of  $x$ ):

$$\forall i: \text{Var}(\varepsilon_i) = \sigma^2$$

- **A3:** The residual errors follow a normal distribution:

$$\varepsilon_i \sim N(0, \sigma^2)$$



How would data look like, that violates these assumptions?

(Draw here ...)

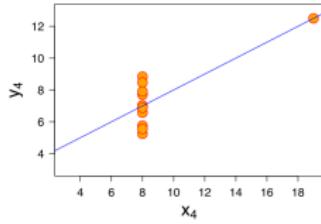
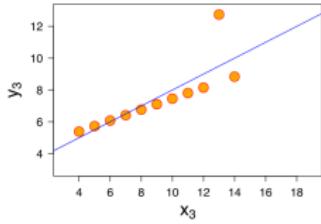
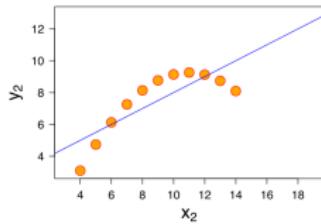
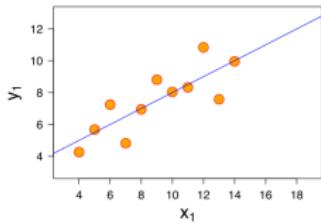
# What influences the Regression Function?

- Offsets
- Heteroscedastic data
- Outliers

# Role of outliers

Four datasets generated by the statistician Francis Anscombe in 1973.  
They demonstrate

- importance of graphing data before analyzing it
- effect of outliers on statistical properties



[[https://en.wikipedia.org/wiki/Anscombe's\\_quartet](https://en.wikipedia.org/wiki/Anscombe's_quartet)]

# Lecture Overview

- 1 Brief Recapitulation: Supervised Regression
- 2 The Linear Regression Model
- 3 Assumptions and Data Scenarios
- 4 How to Derive the Parameters...
- 5 Wrapup: Summary, Related Topics, Preview
- 6 Let's Work on Assignment 2

# Optimization of $w$ via Least Squares Loss Function

Assuming the slightly more elegant formulation for linear regression (in augmented vector notation):

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\varepsilon}$$

we would like to **minimize the squared error** of the model:

$$\operatorname{argmin}_{\mathbf{w}} \|\boldsymbol{\varepsilon}\|^2 = \operatorname{argmin}_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$$

# Optimization of $\mathbf{w}$ via Least Squares Loss Function

$$\operatorname{argmin}_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w})$$

$$\operatorname{argmin}_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = \mathbf{y}^T \mathbf{y} - (\mathbf{X}\mathbf{w})^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\mathbf{w} + (\mathbf{X}\mathbf{w})^T \mathbf{X}\mathbf{w}$$

$$\operatorname{argmin}_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 = -(\mathbf{X}\mathbf{w})^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\mathbf{w} + (\mathbf{X}\mathbf{w})^T \mathbf{X}\mathbf{w}$$

$$= -\mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w}$$

$$= -\mathbf{w}^T \mathbf{X}^T \mathbf{y} - \mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w}$$

$$= -2\mathbf{y}^T \mathbf{X}\mathbf{w} + \mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w}$$

Find optimum by setting the partial derivatives wrt.  $\mathbf{w}_i$  to zero.

Use:  $\frac{\delta}{\delta \mathbf{w}} \mathbf{w}^T \mathbf{A} \mathbf{w} = \mathbf{w}^T (\mathbf{A} + \mathbf{A}^T)$

$$0 = -2\mathbf{y}^T \mathbf{X} + \mathbf{w}^T (\mathbf{X}^T \mathbf{X} + (\mathbf{X}^T \mathbf{X})^T)$$

# Optimization of $\mathbf{w}$ via Least Squares Loss Function

$$\begin{aligned} 0 &= -2\mathbf{y}^T \mathbf{X} + \mathbf{w}^T (\mathbf{X}^T \mathbf{X} + (\mathbf{X}^T \mathbf{X})^T) \\ &= -2\mathbf{y}^T \mathbf{X} + 2\mathbf{w}^T \mathbf{X}^T \mathbf{X} \end{aligned}$$

$$\begin{aligned} &\Leftrightarrow \mathbf{y}^T \mathbf{X} = (\mathbf{w}^T \mathbf{X}^T) \mathbf{X} \\ &\Leftrightarrow \mathbf{X}^T \mathbf{y} = \mathbf{X}^T \mathbf{X} \mathbf{w} \\ &\Leftrightarrow (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \mathbf{w} \end{aligned}$$

# Optimization of $\mathbf{w}$

This delivers the analytical solution:

$$(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \mathbf{w}$$

- What is the costly part of the model training?



# Optimization of $\mathbf{w}$

This delivers the analytical solution:

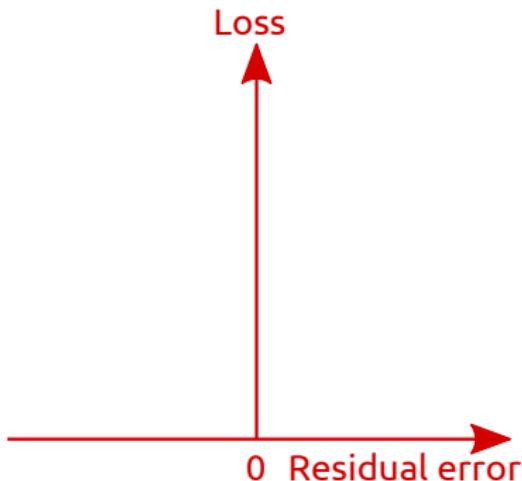
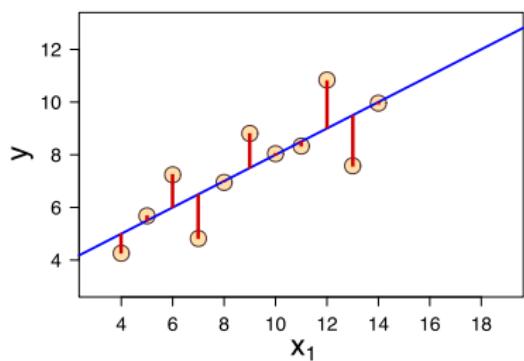
$$(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} = \mathbf{w}$$

- What is the costly part of the model training?  
→ Calculation of inverse  $\in \mathbb{R}^{D \times D}$ 
  - Gauss-Jordan elimination procedure:  $O(D^3)$
  - Coppersmith-Winograd:  $O(D^{2.37\dots})$
  - For large dimensions: stochastic gradient descend  
(nice: the error function is convex!)



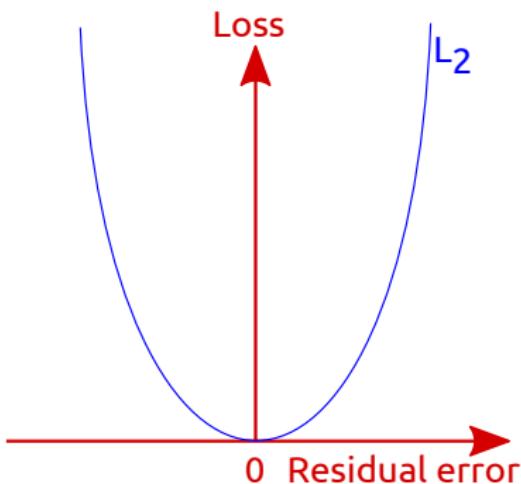
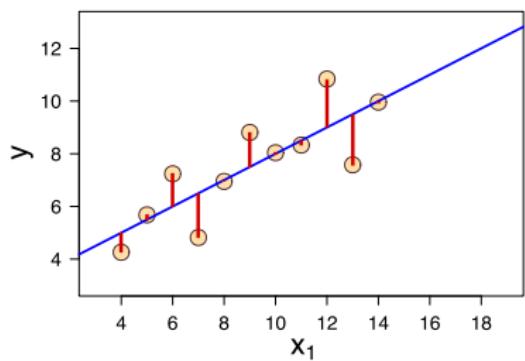
# Alternative Loss Functions

Besides squared loss ( $L_2$ ), other loss functions are possible. They have different pros and cons.



# Alternative Loss Functions

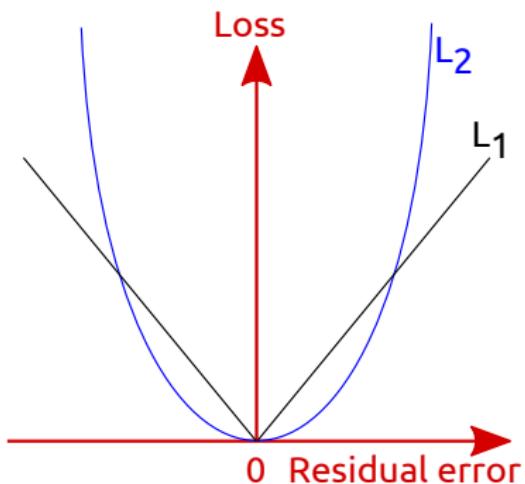
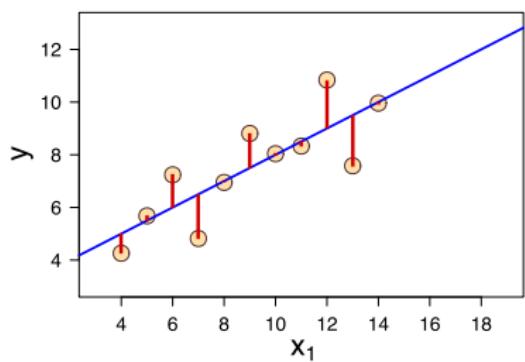
Besides squared loss ( $L_2$ ), other loss functions are possible. They have different pros and cons.



- In which situations would you expect absolute loss ( $L_1$ ) to be preferable compared to squared loss ( $L_2$ )?

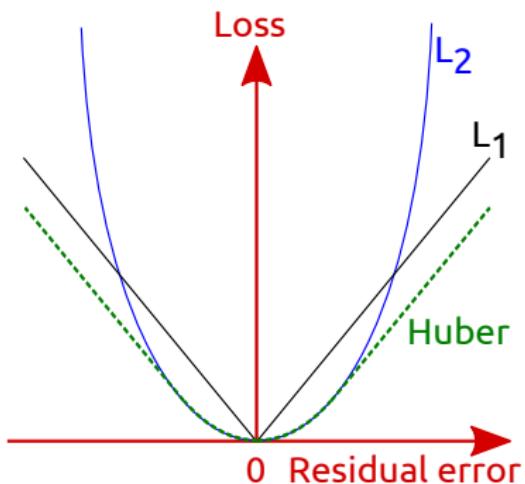
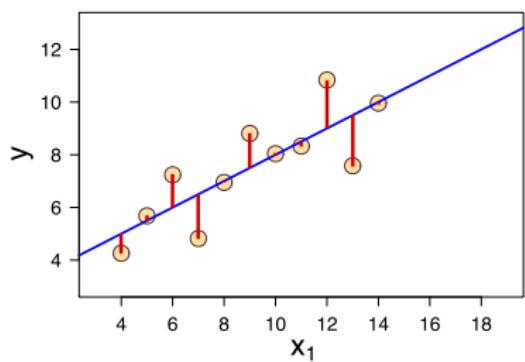
# Alternative Loss Functions

Besides squared loss ( $L_2$ ), other loss functions are possible. They have different pros and cons.



# Alternative Loss Functions

Besides squared loss ( $L_2$ ), other loss functions are possible. They have different pros and cons.



# Regularization with Penalty Terms

Loss functions can be combined with **penalties** for large weight vectors  $\mathbf{w}$ . So-called **penalty** terms are utilized to **regularize** the optimization problems.

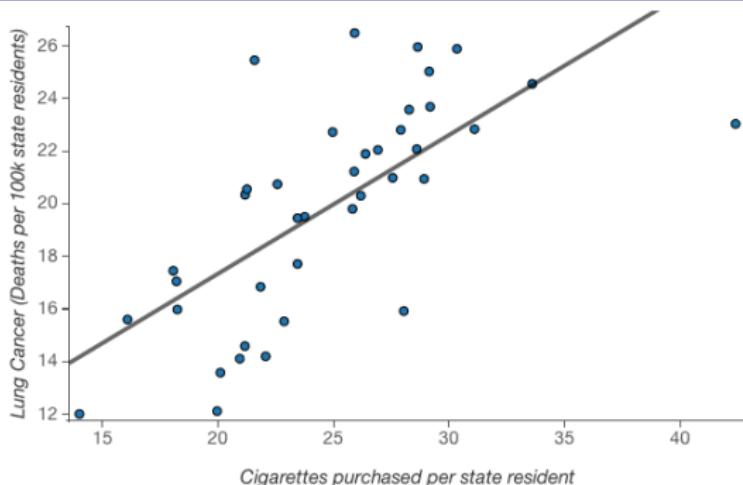
(Regularization may limit overfitting!). Famous regression models:

- **Ridge regression:** quadratic loss on residuals with  $L_2$  norm penalty on the weights. Analytic formulation for  $\mathbf{w}$ . Strong influence of outliers.
- **Lasso:** quadratic loss on residuals with  $L_1$  norm penalty on the weights. No analytic solution, but sparse in  $\mathbf{w}$ . Reduced influence by outliers.

For most-used combinations, their pros and cons, see e.g.

[\[http://www.cs.cornell.edu/courses/cs4780/2015fa/web/lecturenotes/lecturenote10.html\]](http://www.cs.cornell.edu/courses/cs4780/2015fa/web/lecturenotes/lecturenote10.html)

# Typical Use of Linear Regression



- To predict unknown values  $y_i$  for novel input variables  $x_i$
- Estimate the influence of a single input variable or several variables (e.g. in medicine), i.e. estimating the strength of the **correlation** between  $x_i$  and  $y$ . **BUT: does not allow for causal interpretation!**
- Visualization to understand a novel dataset: linear or non-linear relationships? Outliers? Distribution of residual errors?

# Lecture Overview

- 1 Brief Recapitulation: Supervised Regression
- 2 The Linear Regression Model
- 3 Assumptions and Data Scenarios
- 4 How to Derive the Parameters...
- 5 Wrapup: Summary, Related Topics, Preview
- 6 Let's Work on Assignment 2

# Summary by learning goals

Having heard this lecture, you can now ...

- describe a regression problem
- explain pros and cons of using non-linear basis functions
- formulate the regression function (in augmented and non-augmented form)
- explain the meaning / interpretation of the weight vector
- explain (different) assumptions for linear regression models and effects that violations may have
- formulate the optimization criterion for ordinary least-square regression
- describe pros and cons of different loss functions and regularizations
- derive a regression model from given data and apply it to new data

# Organizational Issues

Exam-related:

- Date of the exam: **August 22nd, 2pm**

# Organizational Issues

Exam-related:

- Date of the exam: **August 22nd, 2pm**
- The overview sheets for the different algorithms are meant to help you to prepare the exam, but will not be allowed to bring into the exam.

# Organizational Issues

Exam-related:

- Date of the exam: **August 22nd, 2pm**
- The overview sheets for the different algorithms are meant to help you to prepare the exam, but will not be allowed to bring into the exam.

Assignment-related:

-  Let's discuss: why do we offer assignments at all, if they are not obligatory?

# Lecture Overview

- ➊ Brief Recapitulation: Supervised Regression
- ➋ The Linear Regression Model
- ➌ Assumptions and Data Scenarios
- ➍ How to Derive the Parameters...
- ➎ Wrapup: Summary, Related Topics, Preview
- ➏ Let's Work on Assignment 2

“...”

# Lecture 4: Logistic Regression

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# Lecture Overview

- 1 Motivation
- 2 The Model
- 3 How to Derive the Parameters...
- 4 Wrapup: Summary, Related Topics, Preview

# Lecture Overview

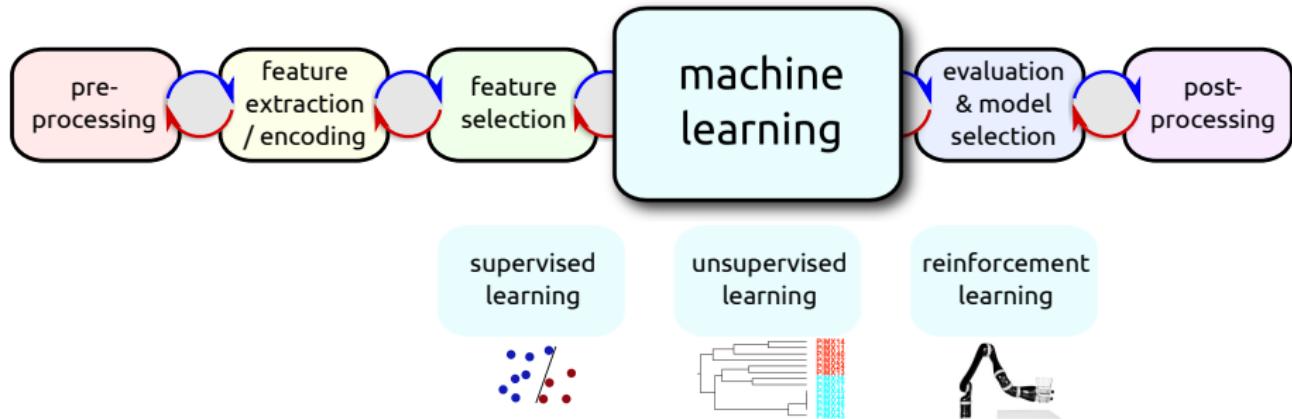
1 Motivation

2 The Model

3 How to Derive the Parameters...

4 Wrapup: Summary, Related Topics, Preview

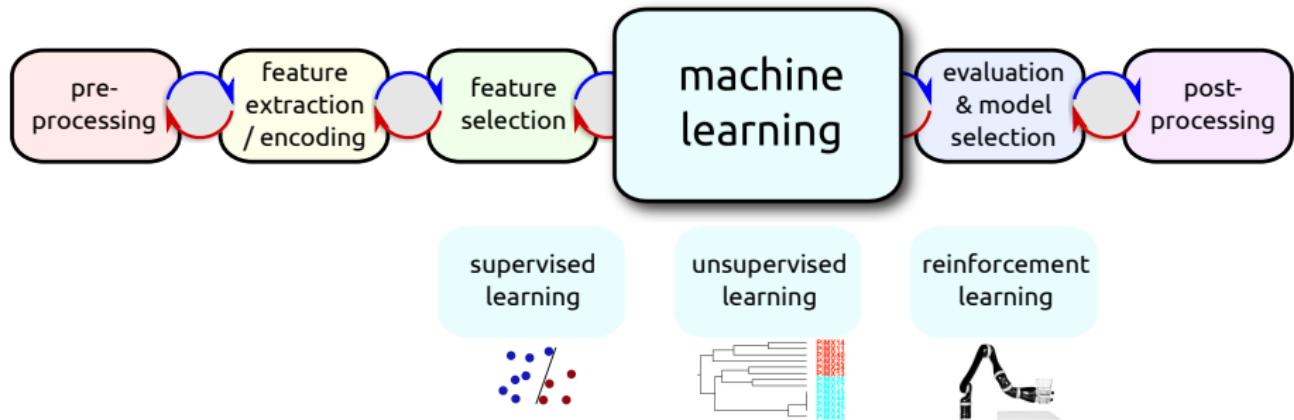
# ML Design Cycle



Today's topic is **classification**:

- Use past experience to predict the future

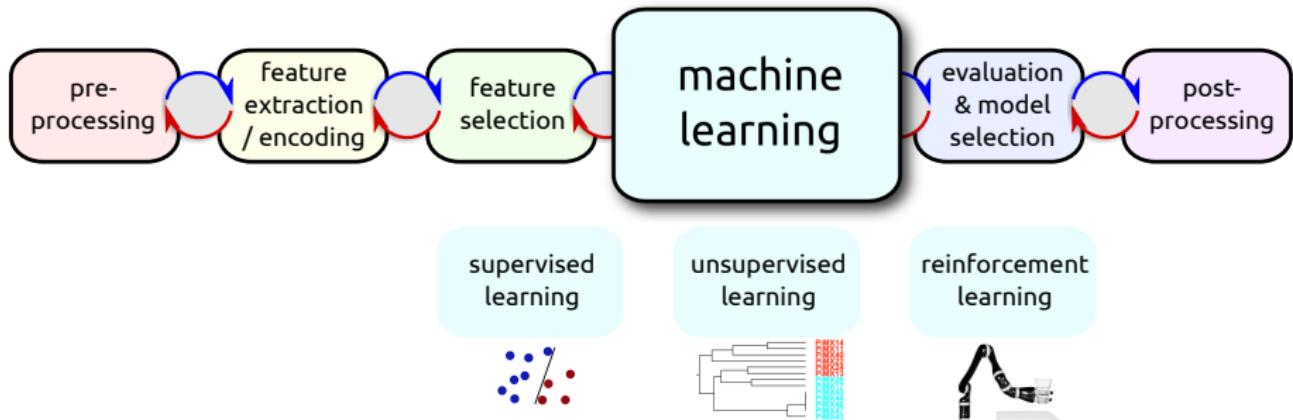
# ML Design Cycle



Today's topic is **classification**:

- Use past experience to predict the future
- Use labelled data points  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$

# ML Design Cycle



Today's topic is **classification**:

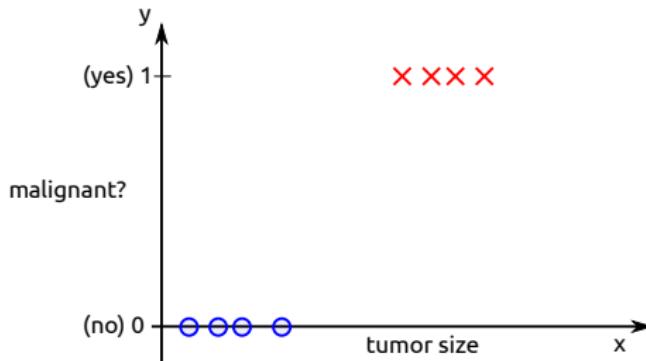
- Use past experience to predict the future
- Use labelled data points  $\langle (\mathbf{x}_i, y_i) \rangle_{i=1}^N$
- Train a **model** which can predict the label  $y_{N+1}$  of a new data point  $\mathbf{x}_{N+1}$

# Simple Example: Tumor Classification

Is the tumor benign or malignant?

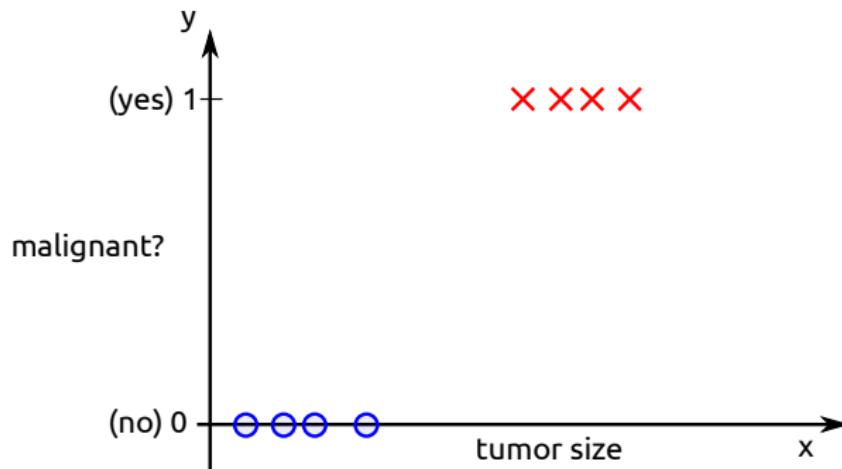
Labels  $y \in \{0, 1\}$ , with

- $y = 0$ : "Negative class" (e.g. benign, not malignant)
- $y = 1$ : "Positive class" (e.g. malignant)



Try fitting a (augmented) model  $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$  as in linear regression...

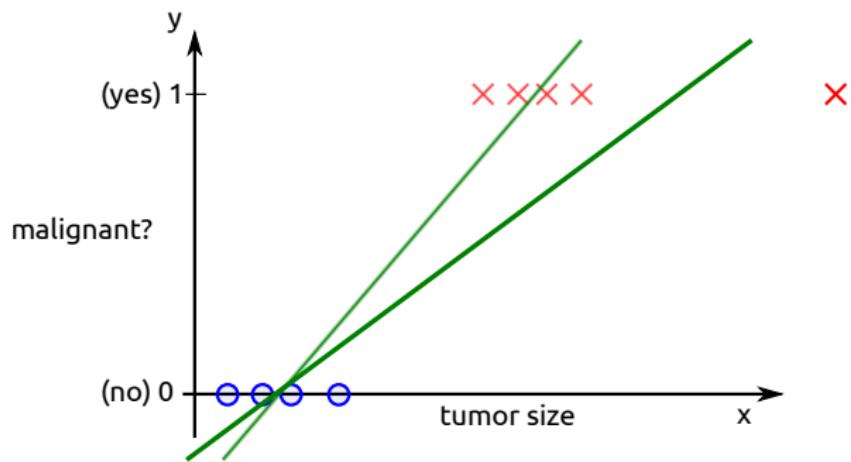
## Simple Example: Tumor Classification



Threshold the classifier output  $h_w(x)$  at 0.5:

- If  $h_w(x) > 0.5$ , predict  $y = 1$ : (malignant)
- If  $h_w(x) \leq 0.5$ , predict  $y = 0$ : (not malignant)

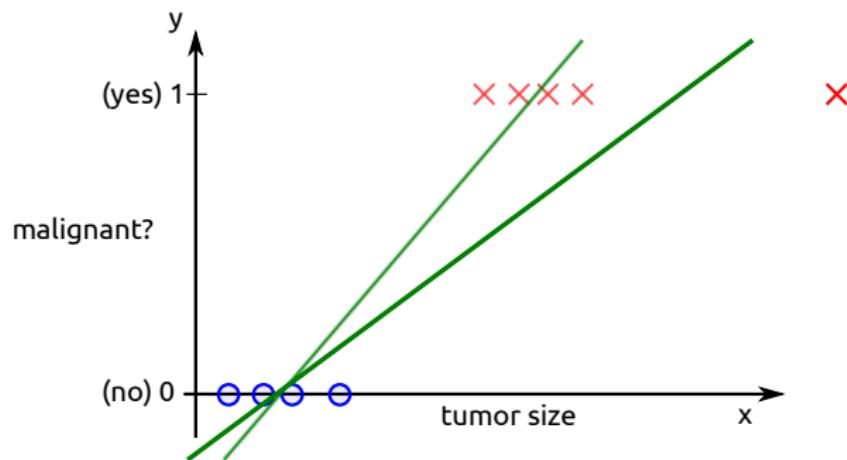
## Simple Example: Tumor Classification



Outliers? Houston, we have a problem...

- For classification case we should only accept  $y = 0$  or  $y = 1$ .
- However, linear regression model  $h_w(x)$  can generate  $y > 1$  or  $y < 0$ .

## Simple Example: Tumor Classification



Outliers? Houston, we have a problem...

- For classification case we should only accept  $y = 0$  or  $y = 1$ .
- However, linear regression model  $h_w(x)$  can generate  $y > 1$  or  $y < 0$ .

Solution: **logistic regression**, which bounds the output to  $0 \leq h_w(x) \leq 1$

# Lecture Overview

1 Motivation

2 The Model

3 How to Derive the Parameters...

4 Wrapup: Summary, Related Topics, Preview

# Logistic Regression Model

We want:  $0 \leq h_{\mathbf{w}}(\mathbf{x}) \leq 1$

Standard linear regression  
(using the inner product) delivers:  
$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

# Logistic Regression Model

We want:  $0 \leq h_{\mathbf{w}}(\mathbf{x}) \leq 1$

Standard linear regression  
(using the inner product) delivers:  
 $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$

A subtle change introduces non-linearity:  
 $h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w}^T \mathbf{x})$  with  $g(z) = \frac{1}{1+e^{-z}}$

# Logistic Regression Model

We want:  $0 \leq h_{\mathbf{w}}(\mathbf{x}) \leq 1$

Standard linear regression  
(using the inner product) delivers:  
 $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$

A subtle change introduces non-linearity:  
 $h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w}^T \mathbf{x})$  with  $g(z) = \frac{1}{1+e^{-z}}$

$$\Rightarrow h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T \mathbf{x}}}$$

# Logistic Regression Model

We want:  $0 \leq h_{\mathbf{w}}(\mathbf{x}) \leq 1$

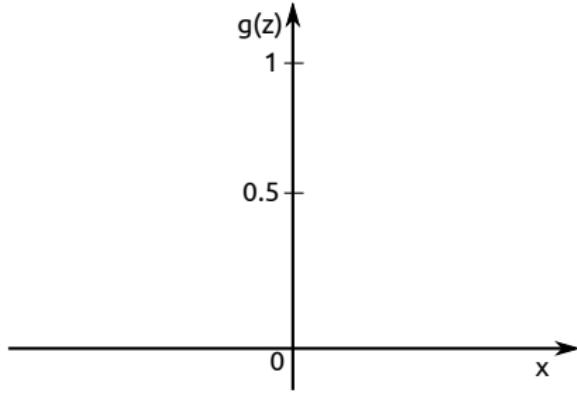
Standard linear regression  
(using the inner product) delivers:  
 $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$

$g(z)$  is called **sigmoid function**  
or **logistic function**

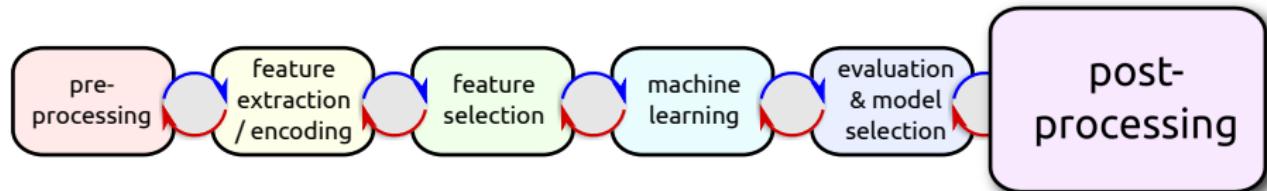
A subtle change introduces non-linearity:

$$h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w}^T \mathbf{x}) \text{ with } g(z) = \frac{1}{1+e^{-z}}$$

$$\Rightarrow h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T \mathbf{x}}}$$



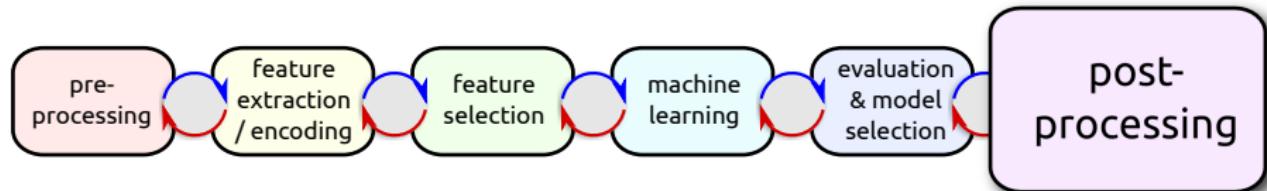
# Role of Non-linear Function $g$



Please observe:

- The use of function  $g(\mathbf{x})$  has similarity to a post-processing of a linear method

# Role of Non-linear Function $g$



Please observe:

- The use of function  $g(\mathbf{x})$  has similarity to a post-processing of a linear method
- You can actually apply a post processing to linear classifier outputs by
  - → **logistic calibration** (making distribution assumptions)
  - → **isotonic calibration** (no assumptions)

# Interpretation of Model Output

$h_{\mathbf{w}}(\mathbf{x}) =$  estimated probability, that  $y = 1$  on input  $\mathbf{x}$

Example with tumor size:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{bmatrix} = \begin{bmatrix} 1 \\ tumorSize \end{bmatrix}$$

# Interpretation of Model Output

$h_{\mathbf{w}}(\mathbf{x}) =$  estimated probability, that  $y = 1$  on input  $\mathbf{x}$

Example with tumor size:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{bmatrix} = \begin{bmatrix} 1 \\ tumorSize \end{bmatrix}$$

How can we interpret the model output  $h_{\mathbf{w}}(\mathbf{x}) = 0.7$ ?

→ We would infer that with a probability of 70% the patient's tumor is malignant.

# Interpretation of Model Output

$h_{\mathbf{w}}(\mathbf{x}) \approx$  estimated probability, that  $y = 1$

More formally, we work with a *hypothesis*:

# Interpretation of Model Output

$h_{\mathbf{w}}(\mathbf{x}) \approx$  estimated probability, that  $y = 1$

More formally, we work with a *hypothesis*:

$$h_{\mathbf{w}}(\mathbf{x}) = P(y = 1 | \mathbf{x}; \mathbf{w})$$

"probability that  $y = 1$ ,  
given that the input is  $\mathbf{x}$  and  
the model is parameterized by  $\mathbf{w}$ "

# Interpretation of Model Output

$h_{\mathbf{w}}(\mathbf{x}) \approx$  estimated probability, that  $y = 1$

More formally, we work with a *hypothesis*:

$$h_{\mathbf{w}}(\mathbf{x}) = P(y = 1|\mathbf{x}; \mathbf{w})$$

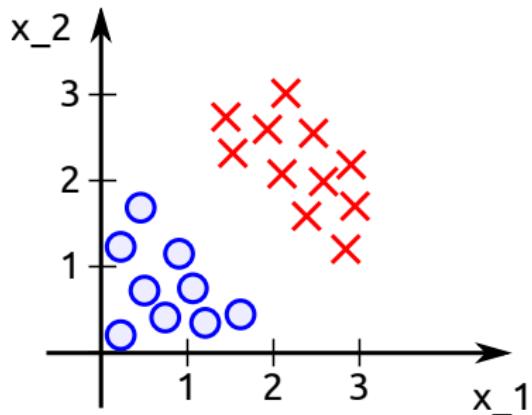
"probability that  $y = 1$ ,  
given that the input is  $\mathbf{x}$  and  
the model is parameterized by  $\mathbf{w}$ "

The actual labels are still discrete  
( $y = 0$  or  $y = 1$ ), but the  
probabilities need to add to one:

$$\begin{aligned}P(y = 0|\mathbf{x}; \mathbf{w}) + P(y = 1|\mathbf{x}; \mathbf{w}) &= 1 \\P(y = 0|\mathbf{x}; \mathbf{w}) &= 1 - P(y = 1|\mathbf{x}; \mathbf{w})\end{aligned}$$

# Example in $\mathbb{R}^2$

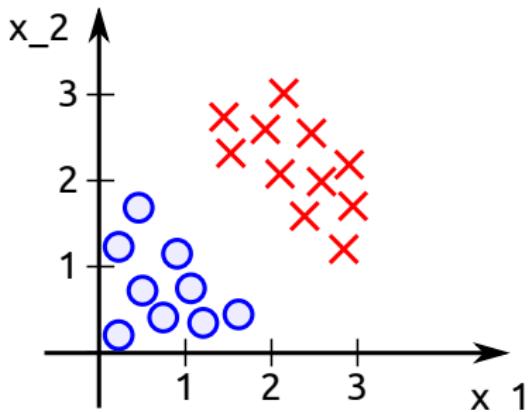
Where is the decision boundary?



$$h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w}_0 + \mathbf{w}_1 \mathbf{x}_1 + \mathbf{w}_2 \mathbf{x}_2)$$

# Example in $\mathbb{R}^2$

Where is the decision boundary?



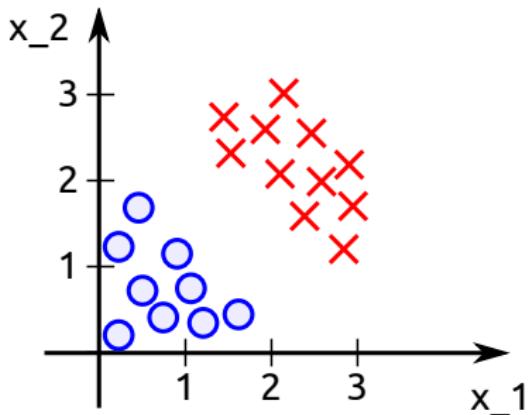
$$h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w}_0 + \mathbf{w}_1 \mathbf{x}_1 + \mathbf{w}_2 \mathbf{x}_2)$$

$$\mathbf{w} = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

Predict " $y = 1$ " if  $-3 + \mathbf{x}_1 + \mathbf{x}_2 \geq 0$

# Example in $\mathbb{R}^2$

Where is the decision boundary?



$$h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w}_0 + \mathbf{w}_1 \mathbf{x}_1 + \mathbf{w}_2 \mathbf{x}_2)$$

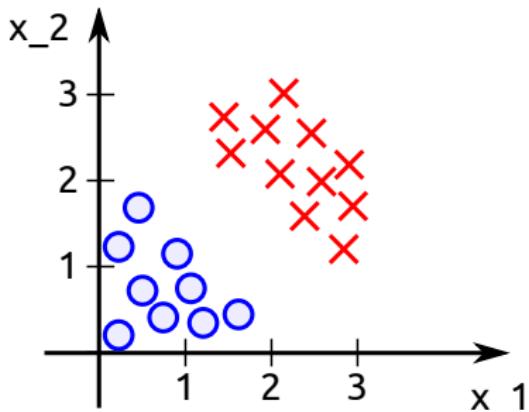
$$\mathbf{w} = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

Predict " $y = 1$ " if  $-3 + \mathbf{x}_1 + \mathbf{x}_2 \geq 0$

$$\mathbf{x}_1 + \mathbf{x}_2 = 3 \Leftrightarrow h_{\mathbf{w}}(\mathbf{x}) = 0.5$$

# Example in $\mathbb{R}^2$

Where is the decision boundary?



$$h_{\mathbf{w}}(\mathbf{x}) = g(\mathbf{w}_0 + \mathbf{w}_1 \mathbf{x}_1 + \mathbf{w}_2 \mathbf{x}_2)$$

$$\mathbf{w} = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

Predict " $y = 1$ " if  $-3 + x_1 + x_2 \geq 0$

$$x_1 + x_2 = 3 \Leftrightarrow h_{\mathbf{w}}(\mathbf{x}) = 0.5$$

Remark: as in linear models, the use of additional dimensions and **basis functions** allows for non-linear decision boundaries!

# Lecture Overview

1 Motivation

2 The Model

3 How to Derive the Parameters...

4 Wrapup: Summary, Related Topics, Preview

# How to Choose the Weights / Parameters $w$ ?

We assume labelled training data points  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$

# How to Choose the Weights / Parameters $w$ ?

We assume labelled training data points  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$

- If we use the augmented notation – what is the dimensionality of the data?



# How to Choose the Weights / Parameters $w$ ?

We assume labelled training data points  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$

- If we use the augmented notation – what is the dimensionality of the data? 
- Answer:  $\mathbf{x} \in \mathbb{R}^{D+1}$  with first dimension  $x_0 = 1$

# How to Choose the Weights / Parameters $w$ ?

We assume labelled training data points  $\langle(\mathbf{x}_i, y_i)\rangle_{i=1}^N$

- If we use the augmented notation – what is the dimensionality of the data? 
- Answer:  $\mathbf{x} \in \mathbb{R}^{D+1}$  with first dimension  $x_0 = 1$

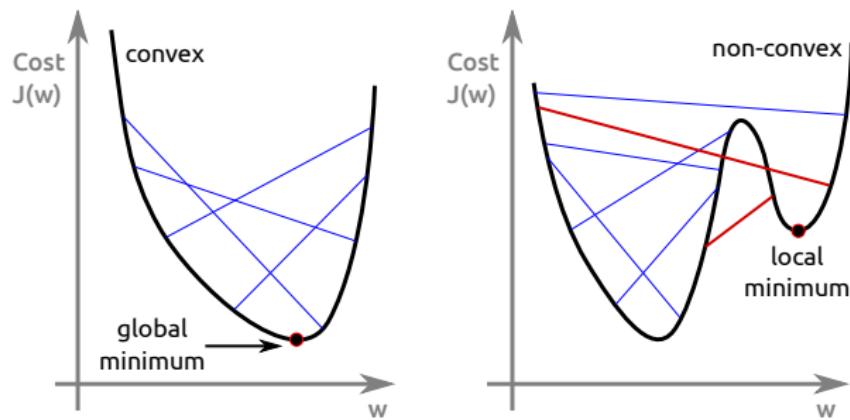
Our model is:

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$

How can we choose the parameters  $\mathbf{w}$ ?

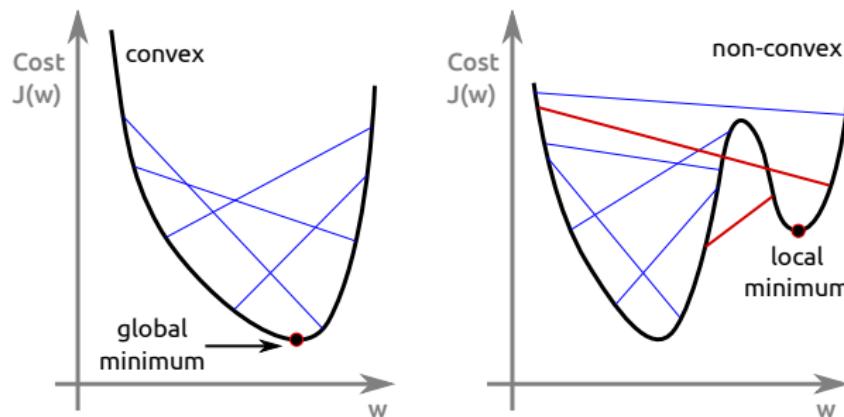
# Reminder: Loss Function Punishes Wrong Predictions

- To tune the weight vector  $w$ , we should check the costs / losses generated by data points, which have not been fitted perfectly (and thus show non-zero residuals).



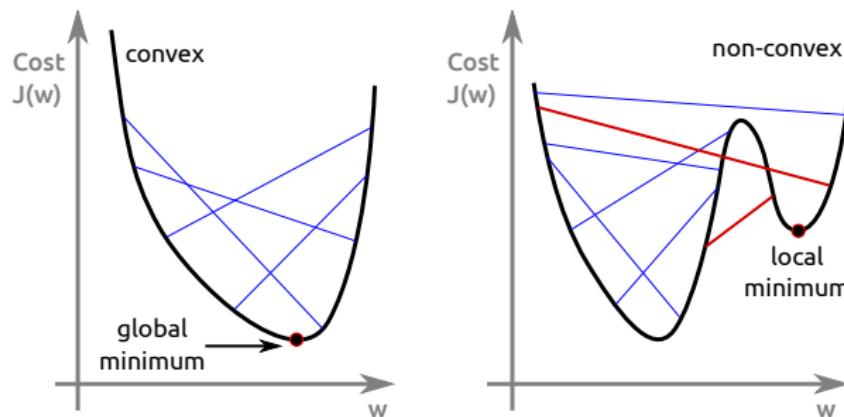
# Reminder: Loss Function Punishes Wrong Predictions

- To tune the weight vector  $w$ , we should check the costs / losses generated by data points, which have not been fitted perfectly (and thus show non-zero residuals).
- Loss for data point  $(x, y)$  depends on the choice of  $w$ :  $Cost(h_w, y)$ .



# Reminder: Loss Function Punishes Wrong Predictions

- To tune the weight vector  $w$ , we should check the costs / losses generated by data points, which have not been fitted perfectly (and thus show non-zero residuals).
- Loss for data point  $(x, y)$  depends on the choice of  $w$ :  $Cost(h_w, y)$ .
- Parameters  $w_i$  can be optimized easier, if the loss function is convex and (continuously) differentiable.



# Quadratic Loss is Unfavorable for Logistic Regression

Unfortunately, a quadratic loss function (as e.g. in LDA)

$$J(\mathbf{w}) = ||h_{\mathbf{w}}(\mathbf{x}), y||^2$$

would lead to a non-convex optimization problem with local minima.

- What may cause this trouble?



# Quadratic Loss is Unfavorable for Logistic Regression

Unfortunately, a quadratic loss function (as e.g. in LDA)

$$J(\mathbf{w}) = ||h_{\mathbf{w}}(\mathbf{x}), y||^2$$

would lead to a non-convex optimization problem with local minima.



- What may cause this trouble?
- Answer: the sigmoid function in  $h_{\mathbf{w}}(\mathbf{x})$

# Quadratic Loss is Unfavorable for Logistic Regression

Unfortunately, a quadratic loss function (as e.g. in LDA)

$$J(\mathbf{w}) = \|\mathbf{h}_{\mathbf{w}}(\mathbf{x}), y\|^2$$

would lead to a non-convex optimization problem with local minima.

- What may cause this trouble?
- Answer: the sigmoid function in  $\mathbf{h}_{\mathbf{w}}(\mathbf{x})$



- How can we derive better loss functions?



# Quadratic Loss is Unfavorable for Logistic Regression

Unfortunately, a quadratic loss function (as e.g. in LDA)

$$J(\mathbf{w}) = \|\mathbf{h}_{\mathbf{w}}(\mathbf{x}), y\|^2$$

would lead to a non-convex optimization problem with local minima.

- What may cause this trouble?



- Answer: the sigmoid function in  $\mathbf{h}_{\mathbf{w}}(\mathbf{x})$

- How can we derive better loss functions?



- Answer: talk to domain experts!

# Adapted Loss Function of Logistic Regression (I)

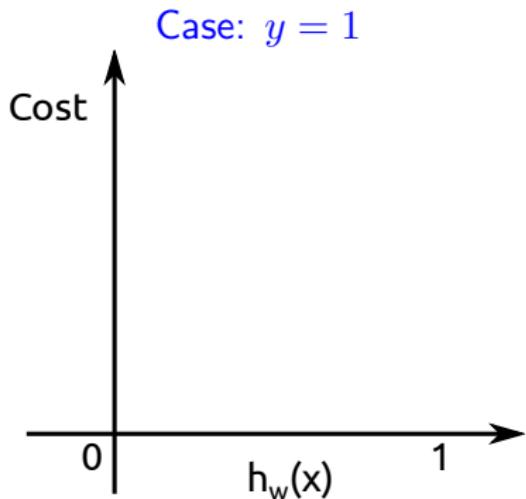
Proposed loss function (specifically adapted to logistic regression):

$$J(h_{\mathbf{w}}(\mathbf{x}), y) = \begin{cases} -\log(h_{\mathbf{w}}(\mathbf{x})) & \text{for } y = 1 \text{ (malignant)} \\ -\log(1 - h_{\mathbf{w}}(\mathbf{x})) & \text{for } y = 0 \end{cases}$$

# Adapted Loss Function of Logistic Regression (I)

Proposed loss function (specifically adapted to logistic regression):

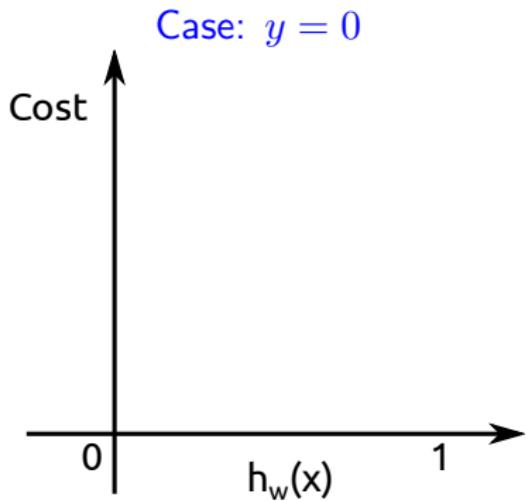
$$J(h_{\mathbf{w}}(\mathbf{x}), y) = \begin{cases} -\log(h_{\mathbf{w}}(\mathbf{x})) & \text{for } y = 1 \text{ (malignant)} \\ -\log(1 - h_{\mathbf{w}}(\mathbf{x})) & \text{for } y = 0 \end{cases}$$



# Adapted Loss Function of Logistic Regression (I)

Proposed cost function for the logistic regression model:

$$Cost(h_{\mathbf{w}}(\mathbf{x}), y) = \begin{cases} -\log(h_{\mathbf{w}}(\mathbf{x})) & \text{for } y = 1 \\ -\log(1 - h_{\mathbf{w}}(\mathbf{x})) & \text{for } y = 0 \end{cases}$$



## Adapted Loss Function for Logistic Regression (II)

The case distinction can be avoided using this formulation  
(check by setting  $y = 0$  and  $y = 1$ ):

$$J(h_{\mathbf{w}}(\mathbf{x}), y) = -y \log(h_{\mathbf{w}}(\mathbf{x})) - ((1 - y) \log(1 - h_{\mathbf{w}}(\mathbf{x})))$$

- Nice property:  $J$  is convex.
- Not so nice property: There is no analytic solution (but gradient descent works well).

Literature:

- Big parts of this section are based on the lectures by Andrew Ng (see youtube channel) on logistic regression.
- For a good reading on logistic calibration of linear classifiers, see Section 7.4 (Obtaining probabilities from linear classifiers) in Peter Flach: Machine Learning (Cambridge Univ. Press, 2012)

## For Toolbox Use: Check Definition of Labels

For the assignment, you have used / will use the scikit-learn toolbox.  
Please note, that this toolbox uses labels  $y \in \{-1, 1\}$ . By reformulation,  
this leads to the following alternative loss function for logistic regression:

$$J(\mathbf{w}) = \sum_{i=1}^N \log(\exp(-y_i(\mathbf{w}^T \mathbf{x}_i)) + 1)$$

Again, check by setting the labels to fixed values and compare the both formulations:

- Case 1:  $y = 0$  (our cost function / Andrew Ng) and  $y = -1$  (scikit-learn)
- Case 1:  $y = 1$  (our cost function / Andrew Ng) and  $y = 1$  (scikit-learn)

# Lecture Overview

- 1 Motivation
- 2 The Model
- 3 How to Derive the Parameters...
- 4 Wrapup: Summary, Related Topics, Preview

# Summary by learning goals

Having heard this lecture, you can now ...

- explain, why probability outputs instead of class values may be desired
- describe the logistic function
- formulate the logistic regression hypothesis
- formulate the optimization criterion and explain, how parameters are determined
- explain, how the output of logistic regression is interpreted
- (assignments) formulate pros and cons of LDA and logistic regression
- (assignments) compare assumptions and the effects of their violation for the two classification approaches

# Lecture 5: Linear Subspace Projections: Principal Component Analysis

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# Lecture Overview

- 1 Motivation
- 2 The PCA Transformation
- 3 Wrapup: Related Topics, Summary, Preview

# Lecture Overview

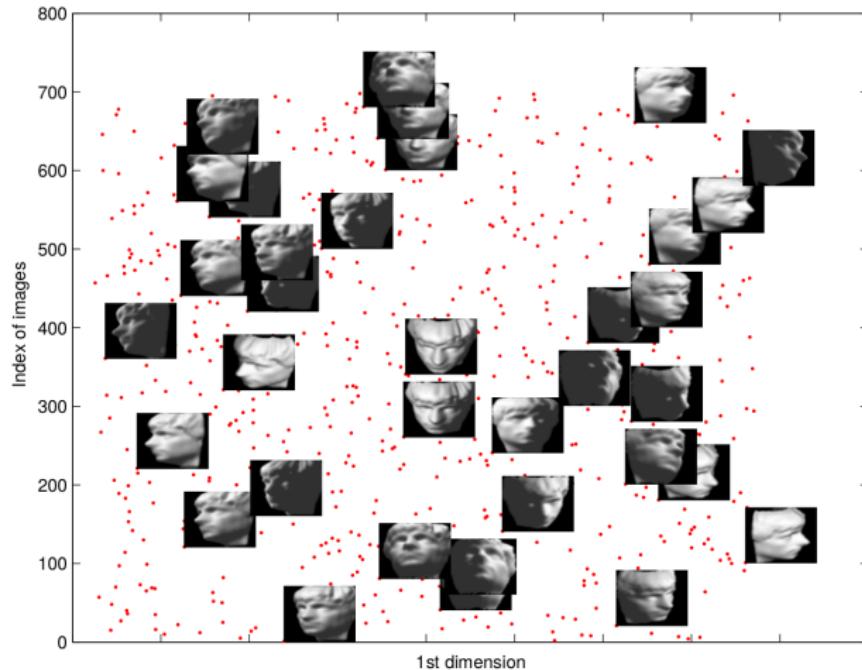
1 Motivation

2 The PCA Transformation

3 Wrapup: Related Topics, Summary, Preview

# Example: Images of Faces

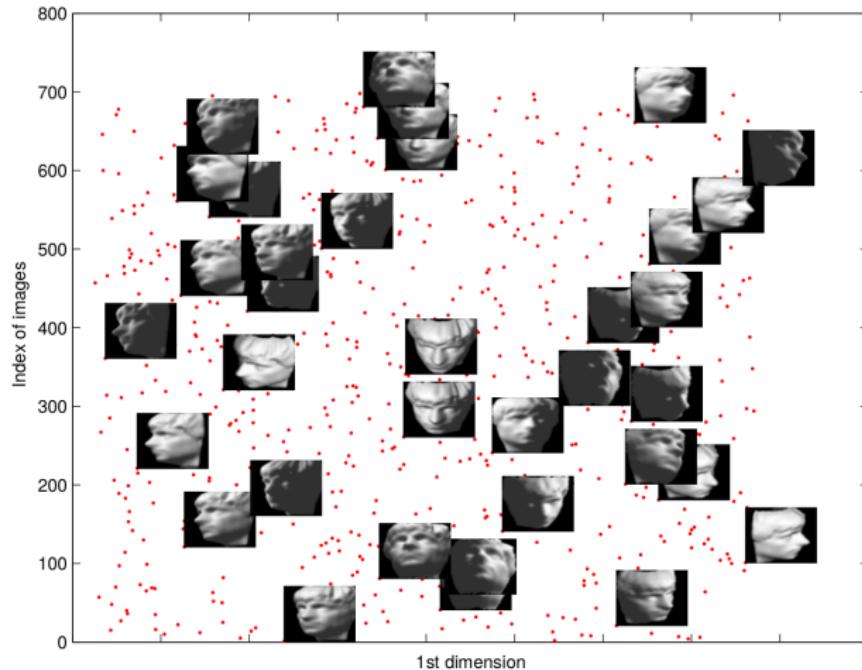
(Plots by Ali Ghodsi, "Dimensionality Reduction, A Short Tutorial", 2006)



Varying pose and and lighting conditions, 698 images (64x64 pixels) of the same face were generated. Dimensionality  $D = 4096$

# Example: Images of Faces

(Plots by Ali Ghodsi, "Dimensionality Reduction, A Short Tutorial", 2006)



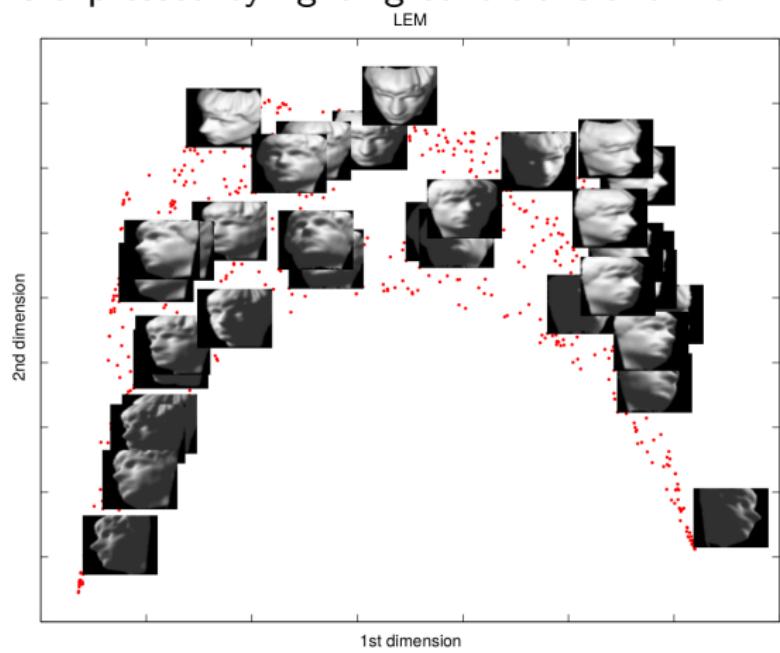
Varying pose and and lighting conditions, 698 images (64x64 pixels) of the same face were generated. Dimensionality  $D = 4096$  **really?**

## Example: Images of Faces

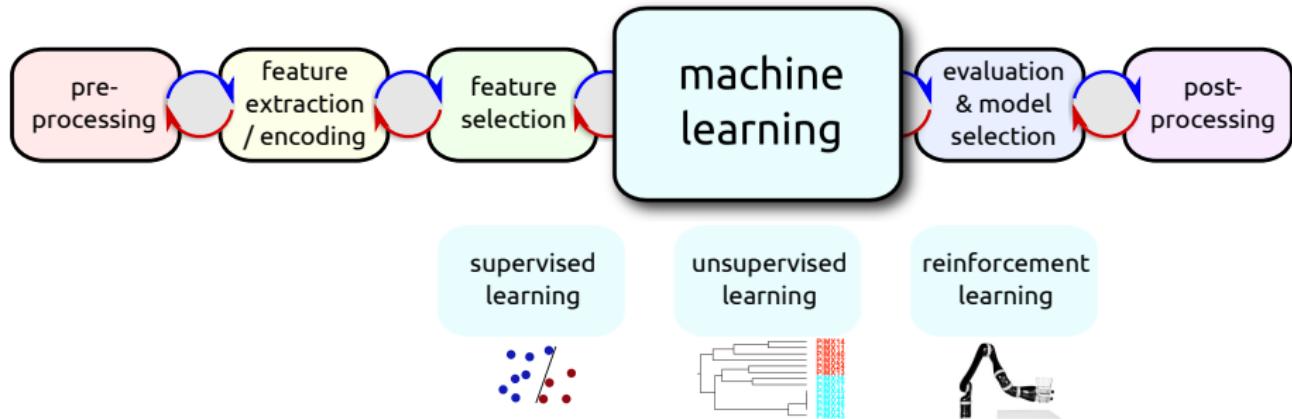
→ As we know, how this data has been generated, we can expect that all of the images live on an embedded, **smaller-dimensional manifold**, which is expressed by lighting conditions and viewing angle.

## Example: Images of Faces

→ As we know, how this data has been generated, we can expect that all of the images live on an embedded, **smaller-dimensional manifold**, which is expressed by lighting conditions and viewing angle.



# ML Design Cycle



Today's topic is how to project data to a useful subspace with **unsupervised** principal component analysis (PCA):

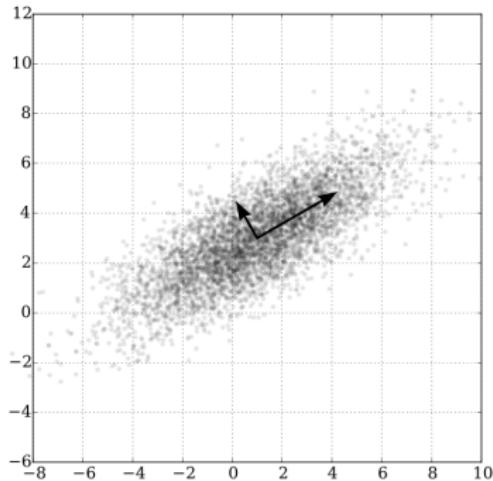
- Labels are not required

# Typical Application of PCA: Dimensionality Reduction

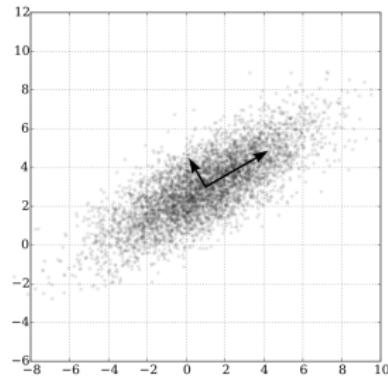
Given:

- N high dimensional data points  $\mathbf{x}_i \in \mathbb{R}^D$  with  $i = 1 \dots N$ .
- Data is collected in matrix  $\mathbf{X} \in \mathbb{R}^{N \times D}$

Scatter plot:



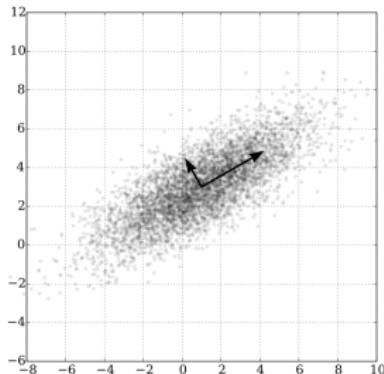
# Typical Application of PCA: Dimensionality Reduction



Let's assume, that

- the input dimensions are correlated

# Typical Application of PCA: Dimensionality Reduction



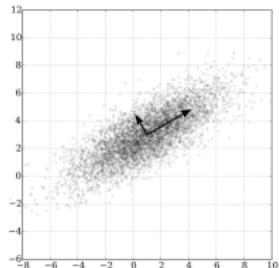
Let's assume, that

- the input dimensions are correlated
- some later method in the pipeline is *extremely* slow on high dimensional data...



What do you propose to do?

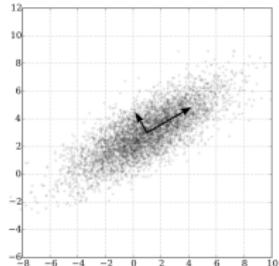
# Simple Example: Dimensionality Reduction



Key ideas:

- Let's try to determine a **subspace**  $\mathbb{R}^M$  of  $\mathbb{R}^D$ , with  $M < D$ .
- The subspace should contain the **relevant** part of our data.

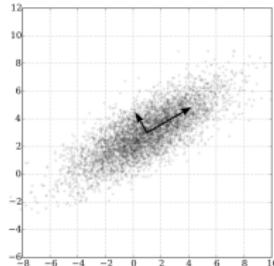
# Simple Example: Dimensionality Reduction



Key ideas:

- Let's try to determine a **subspace**  $\mathbb{R}^M$  of  $\mathbb{R}^D$ , with  $M < D$ .
- The subspace should contain the **relevant** part of our data.
- Let's choose the new dimensions of this projected subspace such, that the data is uncorrelated.
- The subspace can be defined by a projection.

# Simple Example: Dimensionality Reduction



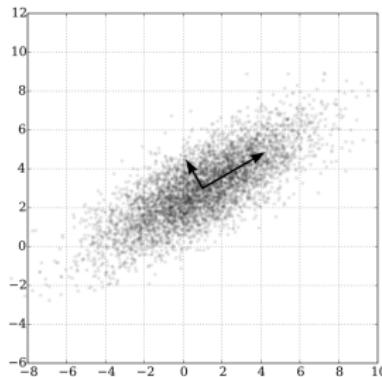
Key ideas:

- Let's try to determine a **subspace**  $\mathbb{R}^M$  of  $\mathbb{R}^D$ , with  $M < D$ .
- The subspace should contain the **relevant** part of our data.
- Let's choose the new dimensions of this projected subspace such, that the data is uncorrelated.
- The subspace can be defined by a projection.



You have seen projections before - in which context?

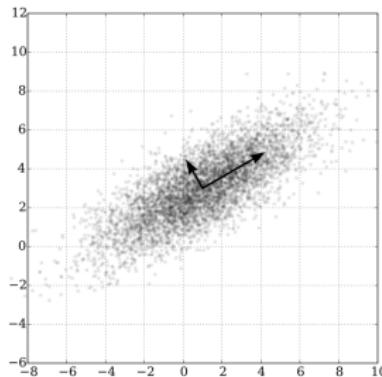
# Simple Example: Dimensionality Reduction



Principal component analysis (PCA) can be applied in such a scenario:

- PCA determines a **linear** subspace
- PCA makes the (somewhat strong!) assumption, that **relevance is expressed by variance!**

# Simple Example: Dimensionality Reduction

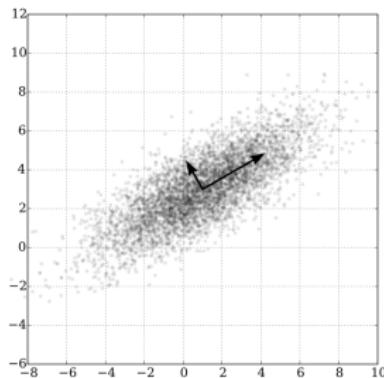


Principal component analysis (PCA) can be applied in such a scenario:

- PCA determines a **linear** subspace
- PCA makes the (somewhat strong!) assumption, that **relevance is expressed by variance!** (**Problematic?** 

Tangermann, Hutter & Lindauer

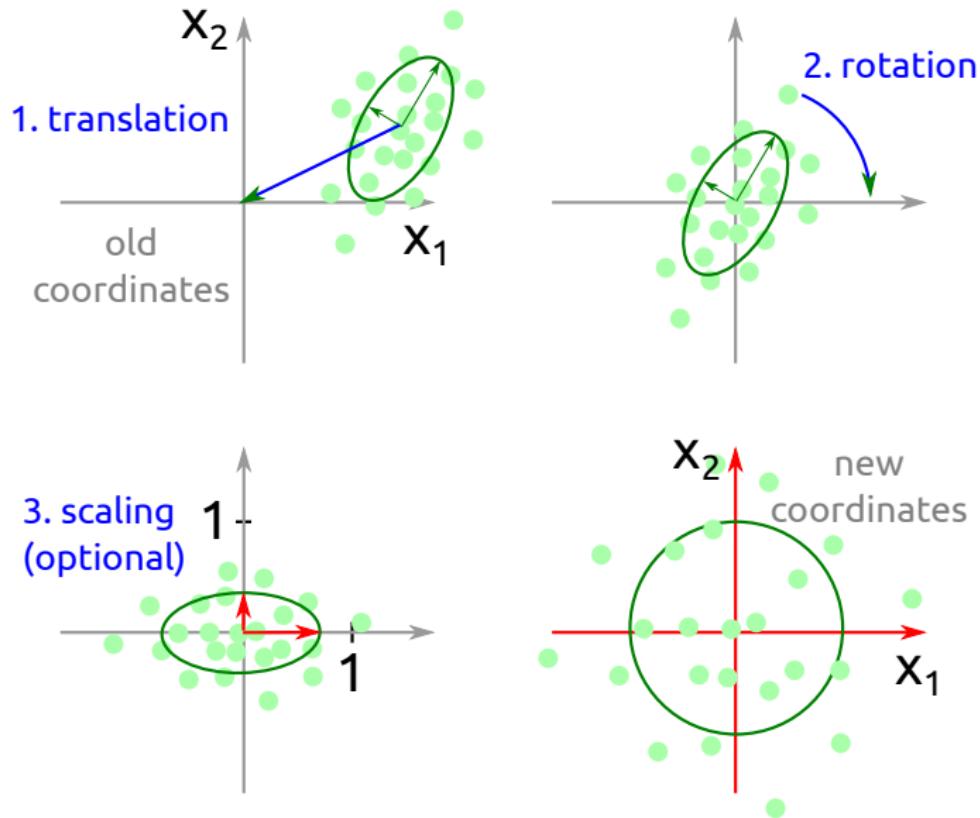
# Simple Example: Dimensionality Reduction



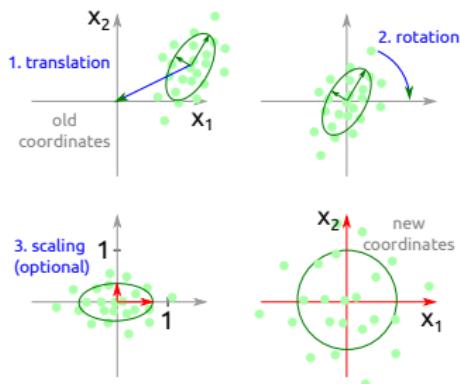
Principal component analysis (PCA) can be applied in such a scenario:

- PCA determines a **linear** subspace
- PCA makes the (somewhat strong!) assumption, that **relevance is expressed by variance!** (**Problematic?** )
- → Can we reduce our data to the subspace with highest variance?

# Intuition of PCA



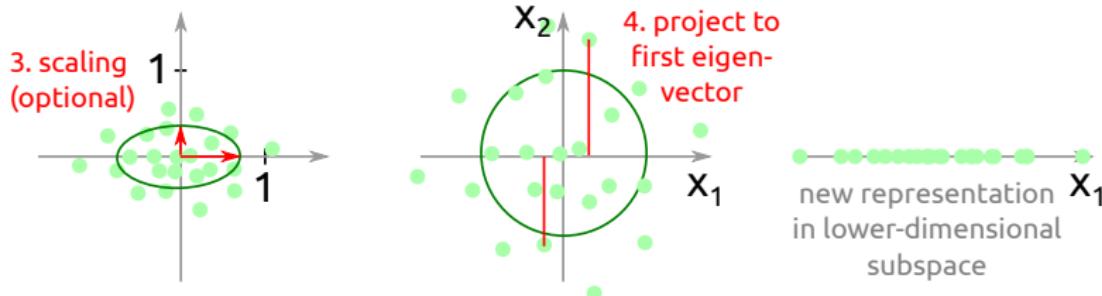
# Intuition of PCA



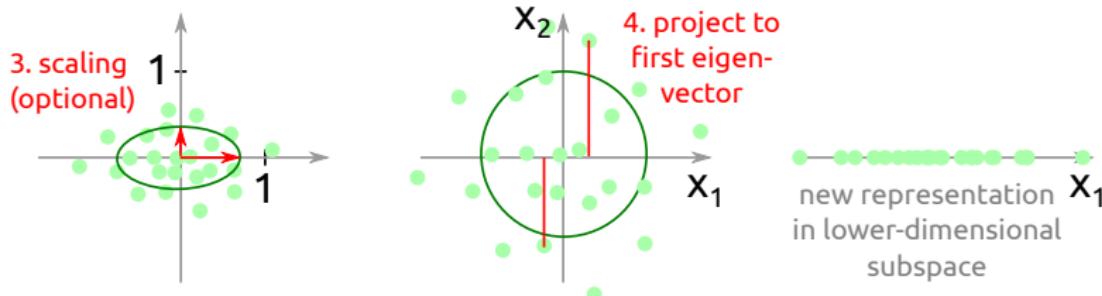
Principal component analysis (PCA) performs the following steps:

- Translation of data  $\mathbf{X}$  to the origin
- Rotation, such that eigenvectors of  $\mathbf{X}$  form the new axes
- (optional:) Scale the projected data according to the eigenvalues

# Intuition of PCA for Dimensionality Reduction



# Intuition of PCA for Dimensionality Reduction



Dimensionality reduction with PCA:

- Project data only to the first  $M$  eigenvectors  
(sorted by strongest eigenvalues)

👉 Assume we have found a lower-dimensional representation with PCA.

Will we be able to **save measuring time** in the future, as we don't need to measure all variables?

# Lecture Overview

1 Motivation

2 The PCA Transformation

3 Wrapup: Related Topics, Summary, Preview

# Derive Projections by Maximizing the Variance

There are many ways to formulate and solve the PCA problem.  
Let's follow the idea to **maximize the variance!**

## Derive Projections by Maximizing the Variance

There are many ways to formulate and solve the PCA problem.  
Let's follow the idea to **maximize the variance!**

Consider a projection to a 1-dimensional subspace ( $M = 1$ ), defined by the direction of a D-dimensional unit vector  $\mathbf{u}_1$ .

# Derive Projections by Maximizing the Variance

There are many ways to formulate and solve the PCA problem.  
Let's follow the idea to **maximize the variance!**

Consider a projection to a 1-dimensional subspace ( $M = 1$ ), defined by the direction of a D-dimensional unit vector  $\mathbf{u}_1$ .

- Unit vector characteristic:  $\mathbf{u}_1^T \mathbf{u}_1 = 1$

## Derive Projections by Maximizing the Variance

There are many ways to formulate and solve the PCA problem.  
Let's follow the idea to **maximize the variance!**

Consider a projection to a 1-dimensional subspace ( $M = 1$ ), defined by the direction of a D-dimensional unit vector  $\mathbf{u}_1$ .

- Unit vector characteristic:  $\mathbf{u}_1^T \mathbf{u}_1 = 1$
- Then each data point can be projected to a scalar value via  $\mathbf{u}_1^T \mathbf{x}$ .

# Derive Projections by Maximizing the Variance

There are many ways to formulate and solve the PCA problem.  
Let's follow the idea to **maximize the variance!**

Consider a projection to a 1-dimensional subspace ( $M = 1$ ), defined by the direction of a D-dimensional unit vector  $\mathbf{u}_1$ .

- Unit vector characteristic:  $\mathbf{u}_1^T \mathbf{u}_1 = 1$
- Then each data point can be projected to a scalar value via  $\mathbf{u}_1^T \mathbf{x}$ .

The mean of projected data is thus given by:  $\mathbf{u}_1^T \bar{\mathbf{x}}$ , with  $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$ .

# Derive Projections by Maximizing the Variance

There are many ways to formulate and solve the PCA problem.  
Let's follow the idea to **maximize the variance!**

Consider a projection to a 1-dimensional subspace ( $M = 1$ ), defined by the direction of a D-dimensional unit vector  $\mathbf{u}_1$ .

- Unit vector characteristic:  $\mathbf{u}_1^T \mathbf{u}_1 = 1$
- Then each data point can be projected to a scalar value via  $\mathbf{u}_1^T \mathbf{x}$ .

The mean of projected data is thus given by:  $\mathbf{u}_1^T \bar{\mathbf{x}}$ , with  $\bar{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n$ .

We want  $\mathbf{u}_1$  to maximize the variance of the projected data:

$$\operatorname{argmax}_{\mathbf{u}_1} \sum_{n=1}^N \{\mathbf{u}_1^T \mathbf{x}_n - \mathbf{u}_1^T \bar{\mathbf{x}}\}^2 = \operatorname{argmax}_{\mathbf{u}_1} \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$$

with  $\mathbf{S}$  being the data covariance matrix.

# The Principal Component Transformation

Attention: maximizing  $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$  with respect to  $\mathbf{u}_1$  requires a constraint, otherwise  $\mathbf{u}_1$  would simply grow to infinity...:

# The Principal Component Transformation

Attention: maximizing  $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$  with respect to  $\mathbf{u}_1$  requires a constraint, otherwise  $\mathbf{u}_1$  would simply grow to infinity...:

- Conveniently, we chose the constraint  $\mathbf{u}_1^T \mathbf{u}_1 = 1$
- We can enforce it by introducing a so-called **Lagrange multiplier**  $\lambda_1$

# The Principal Component Transformation

Attention: maximizing  $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$  with respect to  $\mathbf{u}_1$  requires a constraint, otherwise  $\mathbf{u}_1$  would simply grow to infinity...:

- Conveniently, we chose the constraint  $\mathbf{u}_1^T \mathbf{u}_1 = 1$
- We can enforce it by introducing a so-called **Lagrange multiplier**  $\lambda_1$

This leads to the following maximization problem:

$$\underset{\mathbf{u}_1, \lambda_1}{\operatorname{argmax}} \quad \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)$$

# The Principal Component Transformation

Attention: maximizing  $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$  with respect to  $\mathbf{u}_1$  requires a constraint, otherwise  $\mathbf{u}_1$  would simply grow to infinity...:

- Conveniently, we chose the constraint  $\mathbf{u}_1^T \mathbf{u}_1 = 1$
- We can enforce it by introducing a so-called **Lagrange multiplier**  $\lambda_1$

This leads to the following maximization problem:

$$\underset{\mathbf{u}_1, \lambda_1}{\operatorname{argmax}} \quad \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)$$

Setting the derivative with respect to  $\lambda_1$  to zero, we obtain

$$1 = \mathbf{u}_1^T \mathbf{u}_1$$

# The Principal Component Transformation

Attention: maximizing  $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1$  with respect to  $\mathbf{u}_1$  requires a constraint, otherwise  $\mathbf{u}_1$  would simply grow to infinity...:

- Conveniently, we chose the constraint  $\mathbf{u}_1^T \mathbf{u}_1 = 1$
- We can enforce it by introducing a so-called **Lagrange multiplier**  $\lambda_1$

This leads to the following maximization problem:

$$\underset{\mathbf{u}_1, \lambda_1}{\operatorname{argmax}} \quad \mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 + \lambda_1 (1 - \mathbf{u}_1^T \mathbf{u}_1)$$

Setting the derivative with respect to  $\lambda_1$  to zero, we obtain

$$1 = \mathbf{u}_1^T \mathbf{u}_1$$

Setting the derivative with respect to  $\mathbf{u}_1$  to zero, we obtain

$$0 = \mathbf{S} \mathbf{u}_1 - \lambda_1 \mathbf{u}_1$$

# The Principal Component Transformation

Let's re-write this to

$$\mathbf{S}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

# The Principal Component Transformation

Let's re-write this to

$$\mathbf{S}\mathbf{u}_1 = \lambda_1\mathbf{u}_1$$

This means, that the **variance is maximal**, if  $\mathbf{u}_1$  is an eigenvector of the covariance matrix  $\mathbf{S}$ .

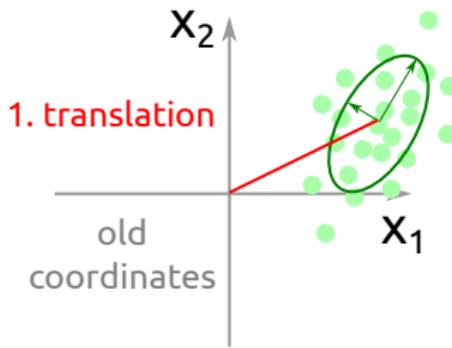
# The Principal Component Transformation

Let's re-write this to

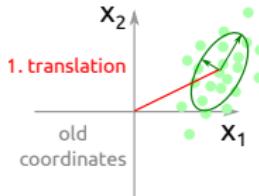
$$\mathbf{S}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

This means, that the **variance is maximal**, if  $\mathbf{u}_1$  is an eigenvector of the covariance matrix  $\mathbf{S}$ .

This meets our intuition, compare



# The Principal Component Transformation



$$\mathbf{S}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1$$

Multiplying with  $\mathbf{u}_1^T$  from the left and making use of  $\mathbf{u}_1^T \mathbf{u}_1 = 1$ , variance is given by:

$$\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 = \lambda_1$$

Observe:

- Variance is maximized, if we set  $\mathbf{u}_1$  equal to the eigenvector having the largest eigenvalue  $\lambda_1$ .

# Obtaining More Than One Projection Direction

We have seen how to obtain the first projection direction via  $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 = \lambda_1$ .



Any idea how to find the next one(s)?

# Obtaining More Than One Projection Direction

We have seen how to obtain the first projection direction via  $\mathbf{u}_1^T \mathbf{S} \mathbf{u}_1 = \lambda_1$ .

 Any idea how to find the next one(s)?

Further projection directions can be obtained by iterating the procedure, making sure that the following eigenvector is **orthogonal** to the ones obtained so far.

- Delivers a set of  $M$  eigenvalues  $\mathbf{u}_1, \dots, \mathbf{u}_M$
- Eigenvalues can be sorted according to the eigenvalues  $\lambda_1, \dots, \lambda_m$

Comment:

consider the **spectrum of eigenvalues**  
to determine a suitable value of  $M$

# Lecture Overview

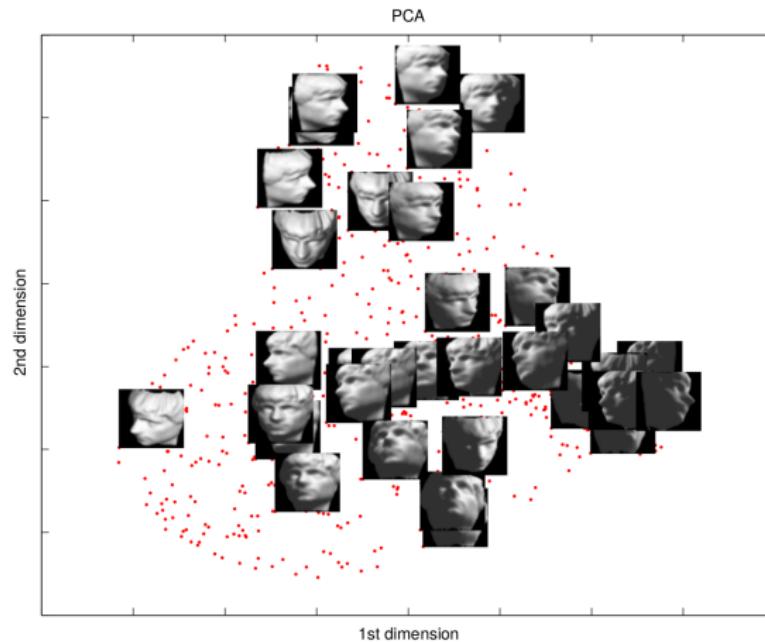
- 1 Motivation
- 2 The PCA Transformation
- 3 Wrapup: Related Topics, Summary, Preview

# Typical Use of Principal Component Analysis

- Data compression / dimensionality reduction (if you think, that variance matters!)

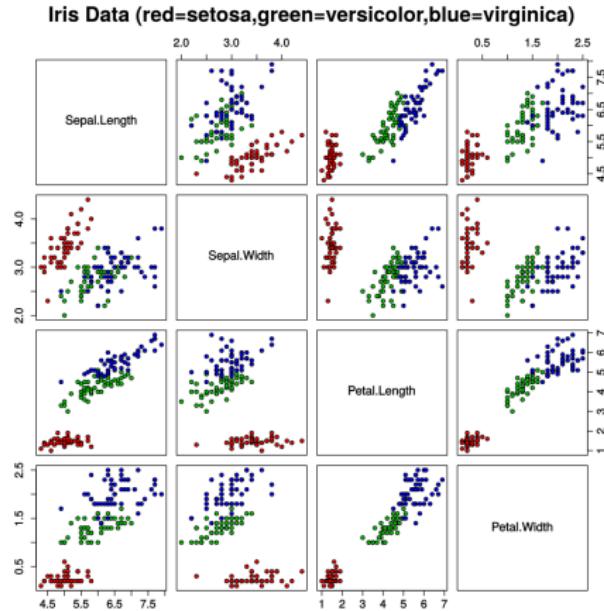
# Typical Use of Principal Component Analysis

- Data compression / dimensionality reduction (if you think, that variance matters!)



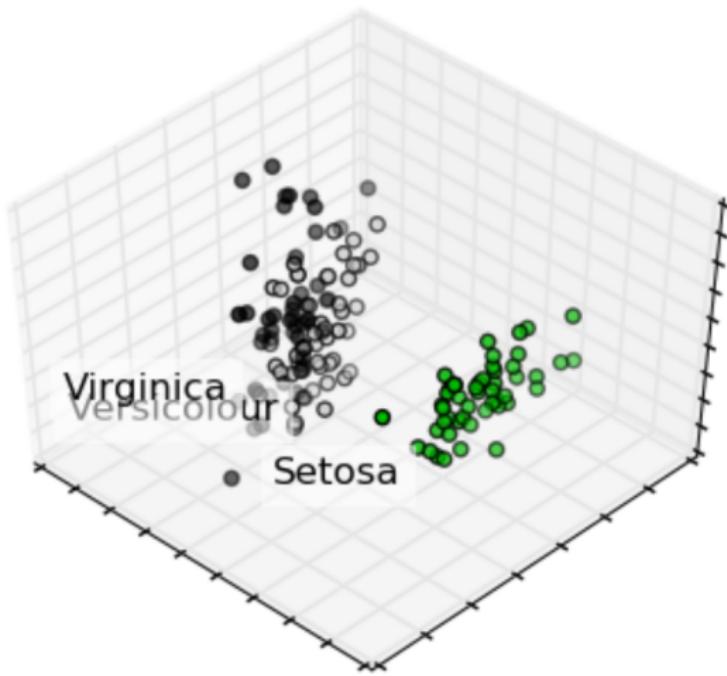
# Typical Use of Principal Component Analysis

- Data compression / dimensionality reduction
- Data visualization using a projection onto  $M = 2$  or  $M = 3$



# Typical Use of Principal Component Analysis

- Data compression / dimensionality reduction
- Data visualization using a projection onto  $M = 2$  or  $M = 3$



# Alternative Names and Formulations for PCA

Depending on the context, principal component analysis is also referred to as

- Linear algebra: singular value decomposition SVD
- Linear algebra: eigenvalue decomposition EVD
- Image processing, control theory: Hotelling transform
- Karhunen-Loëve transform
- ...

# Alternative Names and Formulations for PCA

Depending on the context, principal component analysis is also referred to as

- Linear algebra: singular value decomposition SVD
- Linear algebra: eigenvalue decomposition EVD
- Image processing, control theory: Hotelling transform
- Karhunen-Loëve transform
- ...

Comments:

- There are many algorithmic approaches to derive projection directions (Raleigh coefficient, SVD, Bayesian PCA, iterative vs. analytical, ...)
- Using the covariance matrix has disadvantages, if dimensionality  $D$  is large.

## Further Reading for PCA

- Section 12.1 of Bishop's book was mostly used for these slides
- Wikipedia.org on PCA for a great top-down overview!

# Related Subspace Methods

- Whitening / sphereing: transform data to zero mean and unit covariance as a common preprocessing step
- Factor analysis FA (incorporate domain-specific assumptions)
- Canonical correlation analysis CCA (relate two data sources to a common subspace which maximizes cross-covariance)
- Kernel-PCA (non-linear extension of PCA)

# Summary by learning goals

Having heard this lecture, you can now . . .

- explain, what PCA is doing
- explain, how the novel basis vectors are obtained
- program an iterative version of the algorithm
- formulate assumptions made by PCA (e.g. what happens, if we forget the translation to the origin?)
- name typical use cases of PCA
- (assignments) explain the role and benefits of whitening
- (assignments) explain typical pitfalls related to the use of PCA

# Lecture 6: Linear Subspace Projections: Independent Component Analysis

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



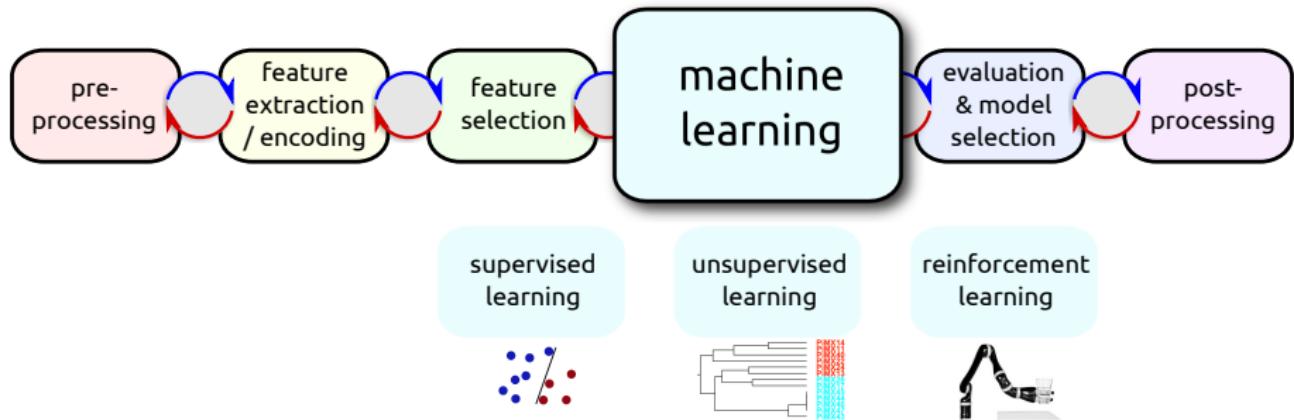
# Lecture Overview

- ① Motivation
- ② The Model
- ③ Estimating Model Parameters
- ④ Practical Issues when Using ICA
- ⑤ Wrapup: Summary, Related Topics, Preview

# Lecture Overview

- 1 Motivation
- 2 The Model
- 3 Estimating Model Parameters
- 4 Practical Issues when Using ICA
- 5 Wrapup: Summary, Related Topics, Preview

# ML Design Cycle



Today's topic is how to separate mixed data sources into *subspaces* or *components* making use of the **unsupervised** independent component analysis (ICA):

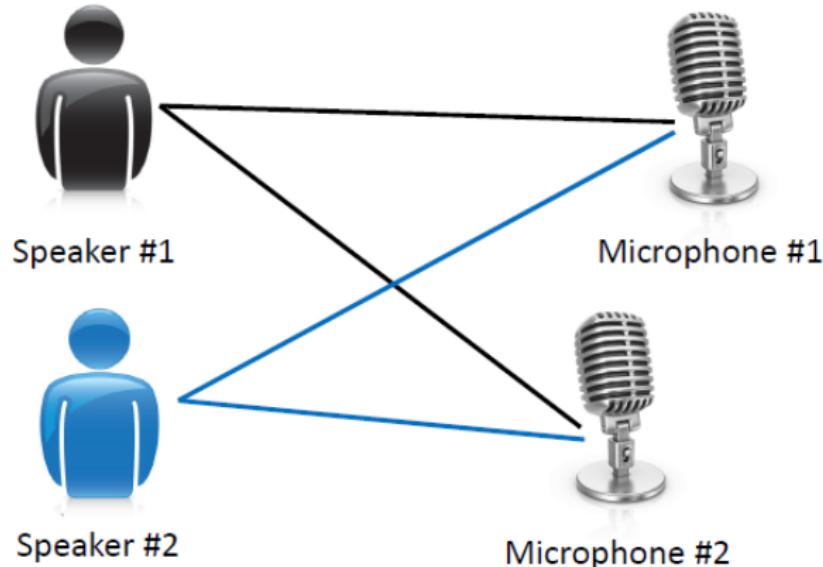
- Labels are not required

# Motivation: Cocktail Party Problem



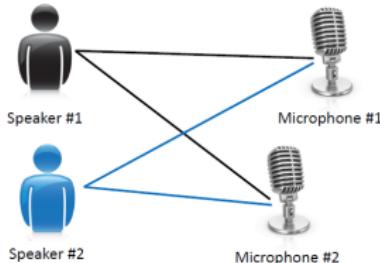
Using a number of microphones -  
can we separate the single speakers (sources)?  
For a computer: very hard problem due to reverberation!

# Motivation: Cocktail Party Problem



Assumptions: no reverberation,  
number of speakers and microphones is matched.

# Assumptions for ICA



Demo: [http://cnl.salk.edu/~tewon/Blind/blind\\_audio.html](http://cnl.salk.edu/~tewon/Blind/blind_audio.html)

Demo: [http://d-kitamura.net/en/demo\\_rank1\\_en.htm](http://d-kitamura.net/en/demo_rank1_en.htm)

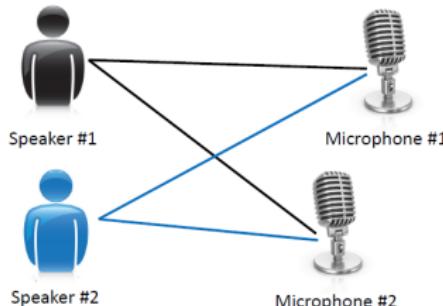
Demo: <http://paris.cs.illinois.edu/demos/index.html>

Demo: [https://cnl.salk.edu/~tewon/Blind/blind\\_audio.html](https://cnl.salk.edu/~tewon/Blind/blind_audio.html)

General problem setting:

- $N$  **independent** sound sources  $s_j$ , which may be active simultaneously.
- $N$  sensors (microphones) are spread in the room.  
They capture different **mixtures**  $x_i$  of the sources.
- Sources  $s_j(t)$  and observed mixtures  $x_i(t)$  are time series signals.  
(with  $t = \{1, \dots, T\}$ , but this is usually omitted)

# Assumptions for ICA



Helpful prerequisites:

- The observed signals are mean-free.
- The observed signals have been whitened.

Strict assumptions for ICA:

- The sources mix **linearly** into the observations.
- At least  $n-1$  of the sources have a **non-Gaussian** distribution.
- The sources are **statistically independent** at each time point  $t$ .

# Key Observations for ICA

From the assumptions:

- The sources mix **linearly** into the observations.
- At least  $n-1$  of the sources have a **non-Gaussian** distribution.
- The sources are **statistically independent** at each time point  $t$ .

... and from the central limit theorem we can expect,

- that any **mixture of sources is more Gaussian than ( $n-1$ -many of the) the original sources.**

How can we make use of this?



# Key Observations for ICA

From the assumptions:

- The sources mix **linearly** into the observations.
- At least  $n-1$  of the sources have a **non-Gaussian** distribution.
- The sources are **statistically independent** at each time point  $t$ .

... and from the central limit theorem we can expect,

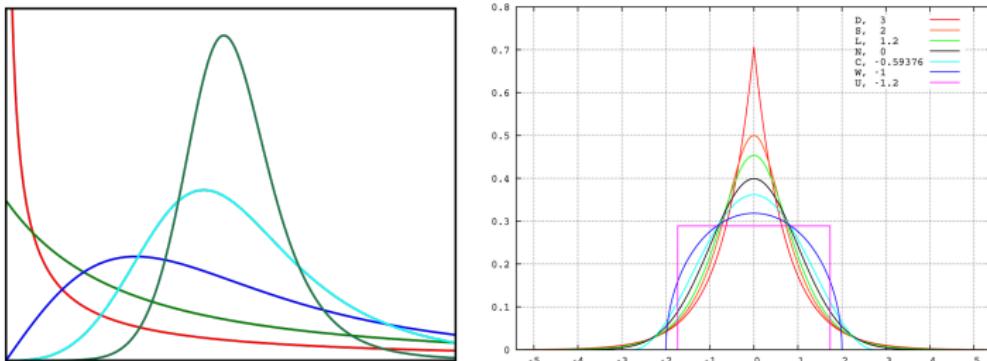
- that any **mixture of sources is more Gaussian than ( $n-1$ -many of the) the original sources.**

How can we make use of this?



- Conversely: the independent sources will be less Gaussian than their mixture.
- For undoing the mixing, we should search for components, which are as non-Gaussian as possible!

# Reminder: Non-Gaussian Distributions



- Distributions can be described by their moments (mean, variance, skewness, **kurtosis**)
- Gaussian distributions have kurtosis of zero, while **non-Gaussian distributions have non-zero kurtosis!**
- Kurtosis of  $y$  is defined by  $kurt(y) = E\{y^4\} - 3(E\{y^2\})^2$
- Assumption of non-Gaussian distributions is often met by real data

[<https://en.wikipedia.org/wiki/Kurtosis>],

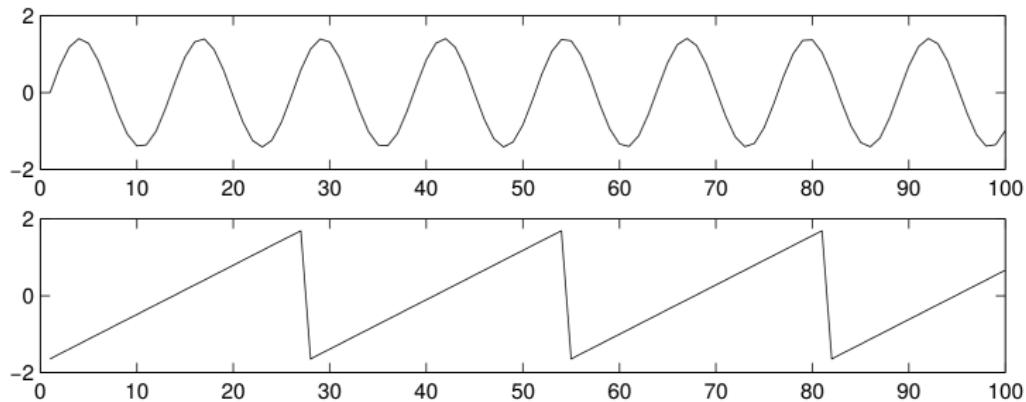
[[https://en.wikipedia.org/wiki/Moment\\_\(mathematics\)](https://en.wikipedia.org/wiki/Moment_(mathematics))]

# The Simplest Cocktail Party Problem

Task:

- Try to recover (unmix) the unknown original sources  $s_j(t)$  based on the recorded /observed mixtures  $x_i(t)$ .

Let's assume optimal conditions (no reverberation), just two sources and two microphones. These are the time series of the [original sources](#):

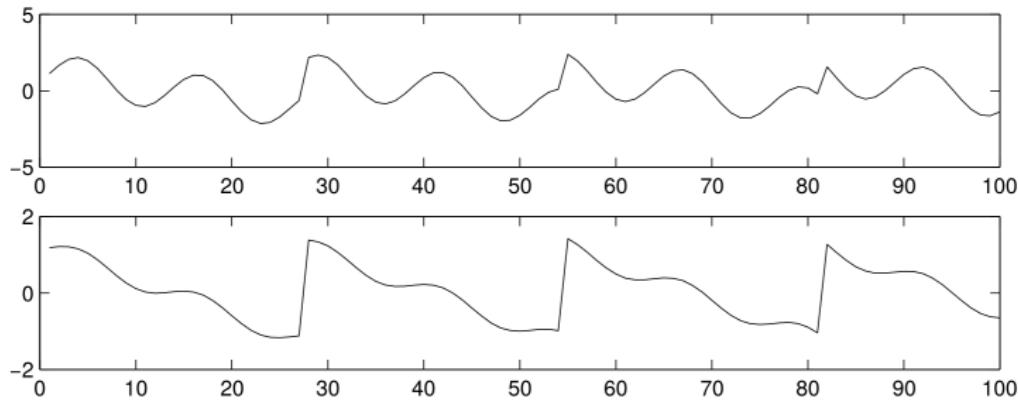


# The Simplest Cocktail Party Problem

Task:

- Try to recover (unmix) the unknown original sources  $s_j(t)$  based on the recorded /observed mixtures  $x_i(t)$ .

Let's assume optimal conditions (no reverberation), just two sources and two microphones. These are the **mixed signals**:



# The Simplest Cocktail Party Problem

Task:

- Try to recover (unmix) the unknown original sources  $s_j(t)$  based on the recorded /observed mixtures  $x_i(t)$ .

Let's assume optimal conditions (no reverberation, neither sources nor sensor move), just two sources and two microphones.

If we knew **know the mixture coefficients**, we could express the relationship between sources and observed microphone signals:

$$\begin{aligned}x_1(t) &= a_{11}s_1(t) + a_{12}s_2(t) \\x_2(t) &= a_{21}s_1(t) + a_{22}s_2(t)\end{aligned}$$

# The Simplest Cocktail Party Problem

Task:

- Try to recover (unmix) the unknown original sources  $s_j(t)$  based on the recorded /observed mixtures  $x_i(t)$ .

Let's assume optimal conditions (no reverberation, neither sources nor sensor move), just two sources and two microphones.

If we knew know the mixture coefficients, we could express the relationship between sources and observed microphone signals:

$$\begin{aligned}x_1(t) &= a_{11}s_1(t) + a_{12}s_2(t) \\x_2(t) &= a_{21}s_1(t) + a_{22}s_2(t)\end{aligned}$$

- If  $x_i$  and  $a_{ij}$  were provided - how could we recover the sources  $s_j$ ?



# Lecture Overview

- 1 Motivation
- 2 The Model
- 3 Estimating Model Parameters
- 4 Practical Issues when Using ICA
- 5 Wrapup: Summary, Related Topics, Preview

# How we get to an ICA Model

As we only know the mixed sensor signals  $x_i$  but not the mixing weights  $a_{ij}$ , solving the equations for  $s_j$  is impossible.

→ We need additional information!

## How we get to an ICA Model

As we only know the mixed sensor signals  $x_i$  but not the mixing weights  $a_{ij}$ , solving the equations for  $s_j$  is impossible.

→ We need additional information!

Use the assumptions! We expected the sources  $s_i$  to have (mostly) non-Gaussian distributions and that they are statistically independent at all time points  $t$ .

# How we get to an ICA Model

As we only know the mixed sensor signals  $x_i$  but not the mixing weights  $a_{ij}$ , solving the equations for  $s_j$  is impossible.

→ We need **additional information!**

**Use the assumptions!** We expected the sources  $s_i$  to have (mostly) non-Gaussian distributions and that they are statistically independent at all time points  $t$ .

Thus we could try to **extremize the kurtosis** of estimated sources (while ensuring their independence) to solve the ICA problem.

**Compare: what is minimized or maximized to solve PCA?**



# The ICA Model

For  $N$  sources and  $N$  sensors, the **mixing** or **forward model** can conveniently be written as (omitting time  $t$  for convenience):

$$\mathbf{x} = \mathbf{As}$$

$\mathbf{A} \in \mathbb{R}^{N \times N}$  is the so called mixing matrix and is assumed to be unknown (as are the sources  $\mathbf{s}_1, \dots, \mathbf{s}_N$ ).

The model is *generative*, as it describes, how the observed data  $\mathbf{x}$  is generated by a mixing process of the underlying sources  $\mathbf{s}$ .

If we would be able to estimate  $\mathbf{A}$ , then its inverse  $\mathbf{W}$  would tell us, how to obtain the hidden components based on our observed signals:

$$\mathbf{s} = \mathbf{Wx}$$

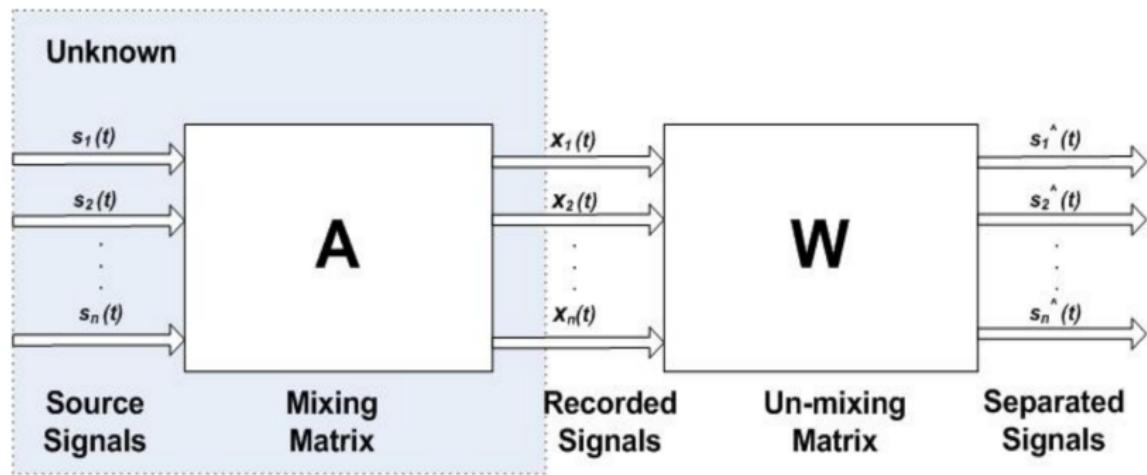
# The ICA Model

If we would be able to estimate  $\mathbf{A}$ , then its inverse  $\mathbf{W}$  would tell us, how to obtain the hidden components based on our observed signals:

$$\mathbf{s} = \mathbf{Wx}$$

This is sometimes called the [backward model](#), with  $\mathbf{W}$  being the unmixing matrix.

# Example of Forward and Backward Model



How can we estimate **A** and **W**?

# Lecture Overview

- 1 Motivation
- 2 The Model
- 3 Estimating Model Parameters
- 4 Practical Issues when Using ICA
- 5 Wrapup: Summary, Related Topics, Preview

# Estimation of Model Parameters

Simplest approach: optimize the vector  $\mathbf{w}$  (one of the columns of  $\mathbf{W}$ ), which is used to project to column vector  $\mathbf{x}$  onto an estimated source  $\hat{s}$ :

$$\hat{s} = \mathbf{w}^T \mathbf{x}$$

To optimize  $\mathbf{w}$ , loop until convergence:

- Initialize weight vector  $\mathbf{w}$
- Determine direction, in which kurtosis of  $\hat{s}$ 
  - grows most strongly (for positive kurtosis) or
  - decreases most strongly (for negative kurtosis)
- Run a step with a gradient descent method to get improved vector  $\mathbf{w}$

# Estimation of Model Parameters

Remarks:

- Iterative approach can be expanded to multivariate case
- Once  $\mathbf{W}$  is estimated,  $\mathbf{A}$  is available, too.

# Estimation of Model Parameters

Remarks:

- Iterative approach can be expanded to multivariate case
- Once  $\mathbf{W}$  is estimated,  $\mathbf{A}$  is available, too.
- Unfortunately kurtosis is hard to estimate in a robust way, thus other measures of non-Gaussianity are preferred in practical implementations:
  - negentropy (to be maximized)
  - mutual information (to be minimized)
  - likelihood (to be maximized)
  - ...

# Estimation of Model Parameters

Remarks:

- Iterative approach can be expanded to multivariate case
- Once  $\mathbf{W}$  is estimated,  $\mathbf{A}$  is available, too.
- Unfortunately kurtosis is hard to estimate in a robust way, thus other measures of non-Gaussianity are preferred in practical implementations:
  - negentropy (to be maximized)
  - mutual information (to be minimized)
  - likelihood (to be maximized)
  - ...
- Popular implementation of ICA (available in most toolboxes): [FastICA](#)

# Lecture Overview

- 1 Motivation
- 2 The Model
- 3 Estimating Model Parameters
- 4 Practical Issues when Using ICA
- 5 Wrapup: Summary, Related Topics, Preview

# Ambiguity of ICA Solution

The ICA problem is **under-determined**, as we only know sensor time series  $\mathbf{X}$ .

Though solutions can be found by making strong assumptions, they are ambiguous:

- We cannot determine the **variances** (energies) of the independent components.
- We cannot determine the **signs** of the independent components.
- We cannot determine the **order** of the independent components.

# Typical Pitfalls for ICA on Real Data

Your data is **noisy**, and repeated runs of ICA on data of successive experimental sessions will deliver slightly different components. However, you would like to compare sessions...

How can you deal with this problem?



# Typical Pitfalls for ICA on Real Data

Your data is **noisy**, and repeated runs of ICA on data of successive experimental sessions will deliver slightly different components. However, you would like to compare sessions...

How can you deal with this problem?



Helpful strategies:

- Try to match the components obtained from repeated runs of the ICA. (difficult! Non-Euclidian space...)
- Train the unmixing matrix based on data of one session and apply it to data of another session.

## Typical Pitfalls for ICA on Real Data

The number of independent sources and the number of your sensors may not match.

- Case 1: You have more sources than sensors – cp. natural cocktail party problem with a human listener (two sensors!) and more than 2 speakers (very bad for standard ICA!). What is the effect?

# Typical Pitfalls for ICA on Real Data

The number of independent sources and the number of your sensors may not match.

- Case 1: You have more sources than sensors – cp. natural cocktail party problem with a human listener (two sensors!) and more than 2 speakers (very bad for standard ICA!). What is the effect?
- Case 2: you have less sources than sensors



Please discuss with your neighbours: what may be the result in case 2?

# Typical Pitfalls for ICA on Real Data

The number of independent sources and the number of your sensors may not match.

- Case 1: You have more sources than sensors – cp. natural cocktail party problem with a human listener (two sensors!) and more than 2 speakers (very bad for standard ICA!). What is the effect?
- Case 2: you have less sources than sensors

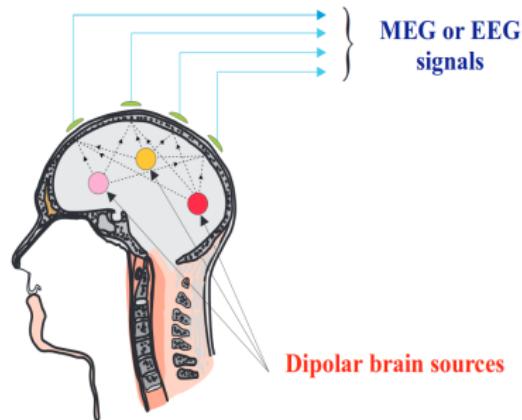


Please discuss with your neighbours: what may be the result in case 2?

As data is usually noisy, ICA will start to [create arbitrary splits](#) of meaningful components. Potential solutions are:

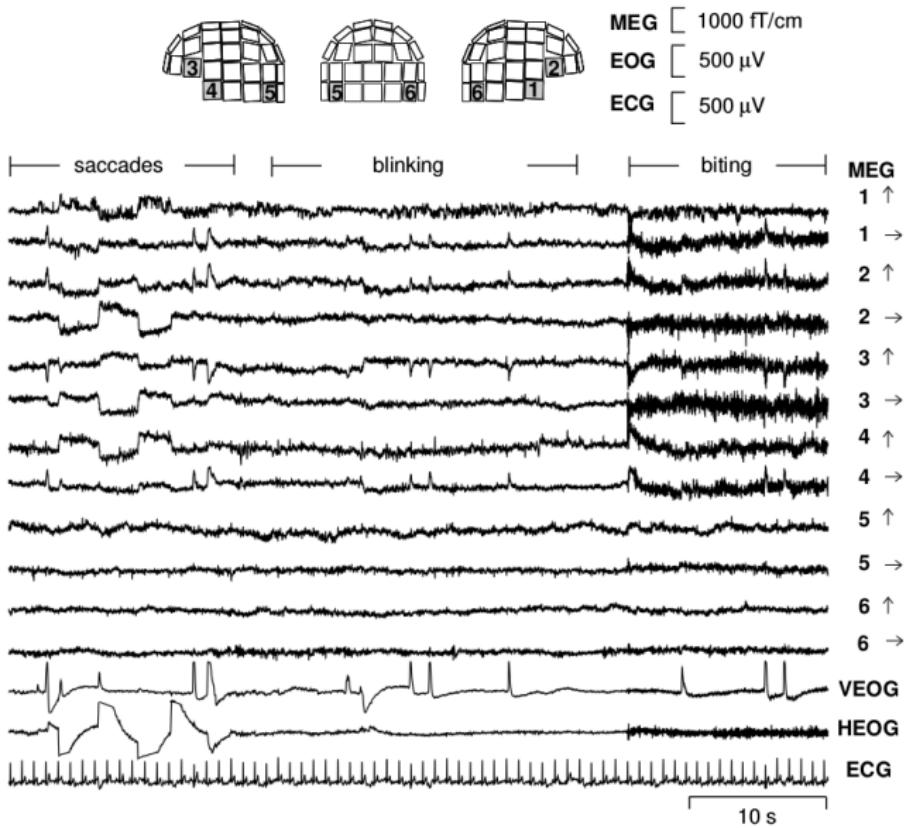
- Try to merge corresponding subspaces post-hoc
- Reduce the dimensionality of your data prior to applying ICA. PCA may do a good job...

# Application Fields for ICA



- Neuroscience: interpretation of brain signals and their sources, separation of neural signals from artifacts
- Hearing aid research
- Prediction of stock market prices
- Telecommunications
- Geology
- Radioastronomy
- Image denoising

# Typical ICA Workflow on (Neuro-)Physiological Data

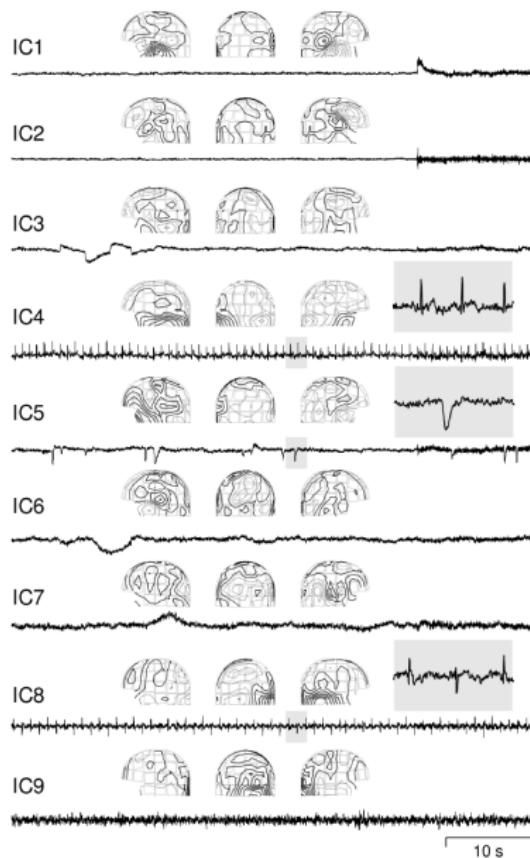


# Typical Workflow for Independent Component Analysis

Perform the following steps:

- **Unmix** the recorded data to obtain the original sources
- **Inspect** the components visually into desired and undesired sources  
(remark: automated approaches exist to classify neural- from non-neural sources, e.g. [MARA toolbox by Winkler et al., J. Neural Eng. 2014])

# Typical Workflow of Independent Component Analysis



# Typical Workflow for Independent Component Analysis

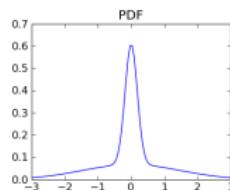
Perform the following steps:

- **Unmix** the recorded data to obtain the original sources
- **Inspect** the components visually into desired and undesired sources  
(remark: automated approaches exist to classify neural- from non-neural sources, e.g. [MARA toolbox by Winkler et al., J. Neural Eng. 2014])
- Option 1: Use desired sources only, continue to work in the lower-dimensional ICA-space
- Option 2: **Reconstruct** the sensor data in the original space using only the desired sources

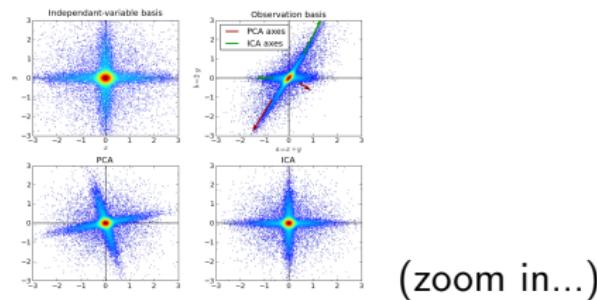
Attention: Option 2 leads to reconstructed data,  
which may **not have full rank** any more!

# Comparison with PCA

Let's choose two variables, which are independent and are sampled from this non-Gaussian probability density function:

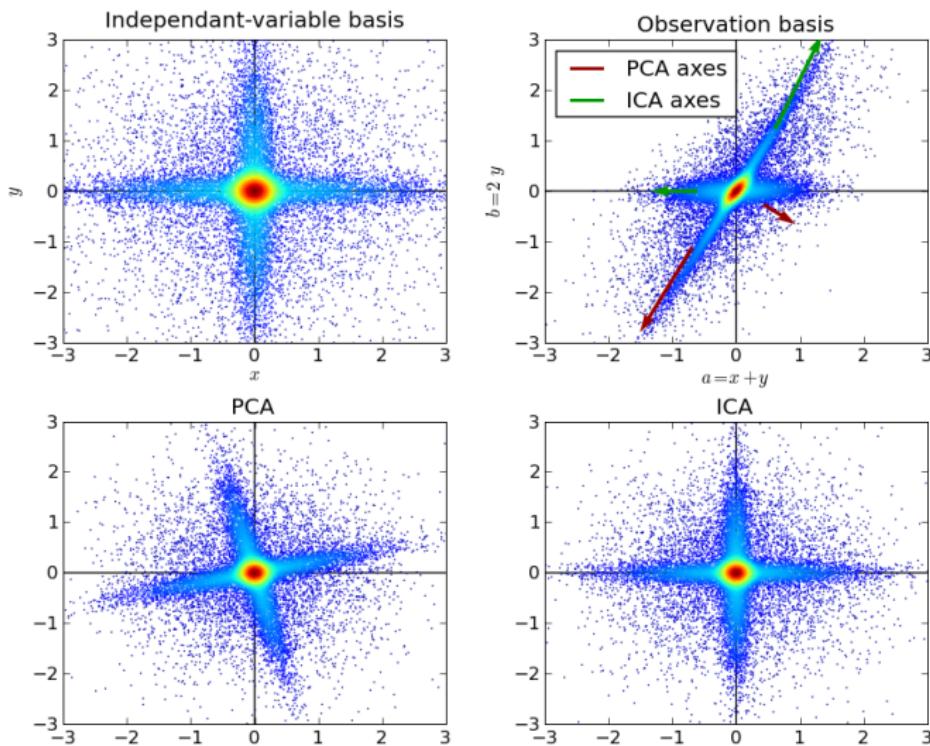


In some situations, PCA can not recover the original sources, while ICA is able to separate them:

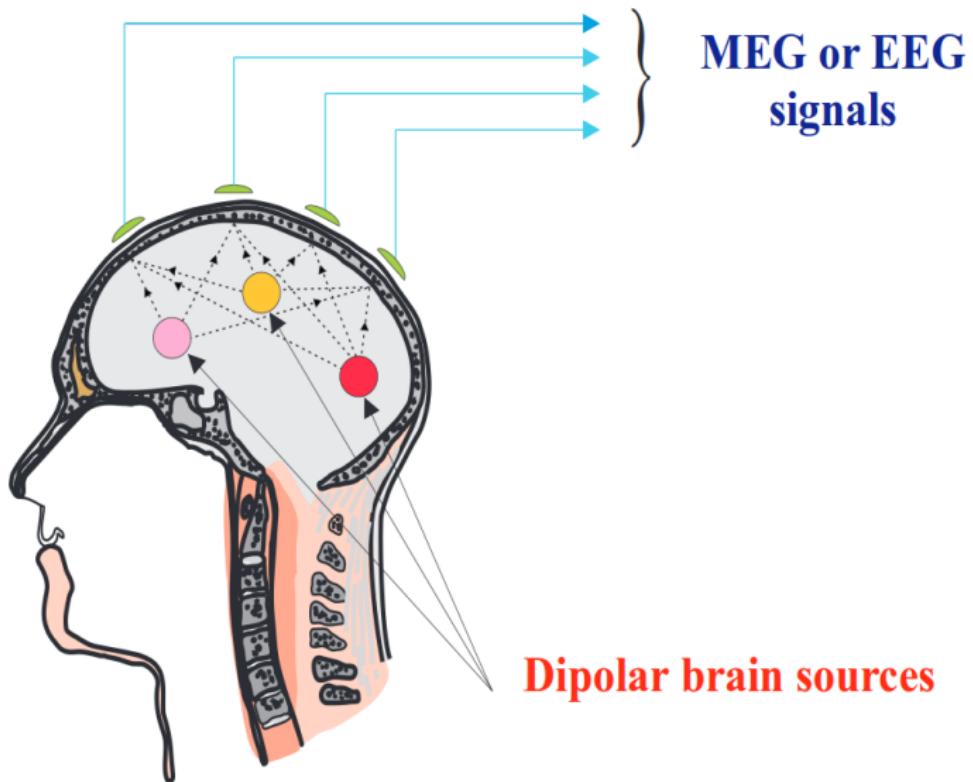


# Comparison with PCA

While PCA can not recover the two sources, ICA is able to separate them:



# Application Fields for ICA



# Lecture Overview

- 1 Motivation
- 2 The Model
- 3 Estimating Model Parameters
- 4 Practical Issues when Using ICA
- 5 Wrapup: Summary, Related Topics, Preview

# Pros and Cons of ICA

- Con: Components have arbitrary sign, arbitrary order and amplitude.  
→ Finding the **matching components** e.g. for two experimental sessions is not trivial.

# Pros and Cons of ICA

- Con: Components have arbitrary sign, arbitrary order and amplitude.  
→ Finding the **matching components** e.g. for two experimental sessions is not trivial.
- Pro: ICA is a linear method - once trained, it can be **applied extremely fast** for online systems

# Pros and Cons of ICA

- Con: Components have arbitrary sign, arbitrary order and amplitude.  
→ Finding the **matching components** e.g. for two experimental sessions is not trivial.
- Pro: ICA is a linear method - once trained, it can be **applied extremely fast** for online systems
- Pro: Being a linear method, independent components can be **visualized**. Experts can judge the quality of the unmixing.

# Pros and Cons of ICA

- Con: Components have arbitrary sign, arbitrary order and amplitude.  
→ Finding the **matching components** e.g. for two experimental sessions is not trivial.
- Pro: ICA is a linear method - once trained, it can be **applied extremely fast** for online systems
- Pro: Being a linear method, independent components can be **visualized**. Experts can judge the quality of the unmixing.
- Pro: Many variants of ICA exist. They use different assumptions on what "statistical independence" means. Thus for most unmixing problems (specific forms of noise, exploit temporal correlations within sources, perform ICA across data of several subjects etc.), you will probably find a variant, which can deal with your data.

## Related Subspace Methods

- Whitening / sphereing: transform data to zero mean and unit covariance (preprocessing step for ICA)
- Factor analysis FA (incorporate domain-specific assumptions)
- Canonical correlation analysis CCA (relate two data sources to a common subspace which maximizes cross-covariance)
- Kernel-ICA (non-linear extension of ICA)
- Blind Source Separation (BSS) - actually ICA is a special case of BSS.

# Ressources on ICA

- Great collection of demos, easy and detailed papers on ICA on the webpage of Aapo Hyvärinen's group in Finland:  
[<http://research.ics.aalto.fi/ica/>].  
(The lecture was mostly based on his great tutorial paper!)
- Cool applications of ICA and advanced algorithms for blind source separation on website of Paris Smaragdis: [<http://paris.cs.illinois.edu/>]
- Wikipedia.org on ICA for a great top-down overview!
- Tutorial provided for the EEGLab matlab toolbox by the group of Scott Makeig

# Summary by learning goals

Having heard this lecture, you can now ...

- Formulate the type of problems, that can be solved by ICA (and which can not)
- Formulate the assumptions made by ICA
- Formulate and solve the optimization problem for ICA
- Explain the difference between ICA and PCA

# Lecture 7: Algorithm Independent Principles

Model Class Complexity, Bias-Variance Analysis, Validation Protocols

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# Lecture Overview

- 1 Complexity of Hypothesis Class
- 2 Bias - Variance Analysis
- 3 Validation Protocols
- 4 Summary

# Lecture Overview

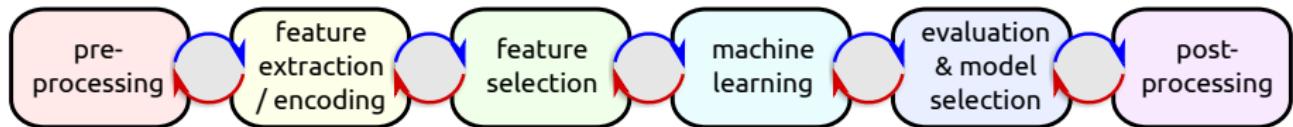
1 Complexity of Hypothesis Class

2 Bias - Variance Analysis

3 Validation Protocols

4 Summary

# Reminder: ML Design Cycle



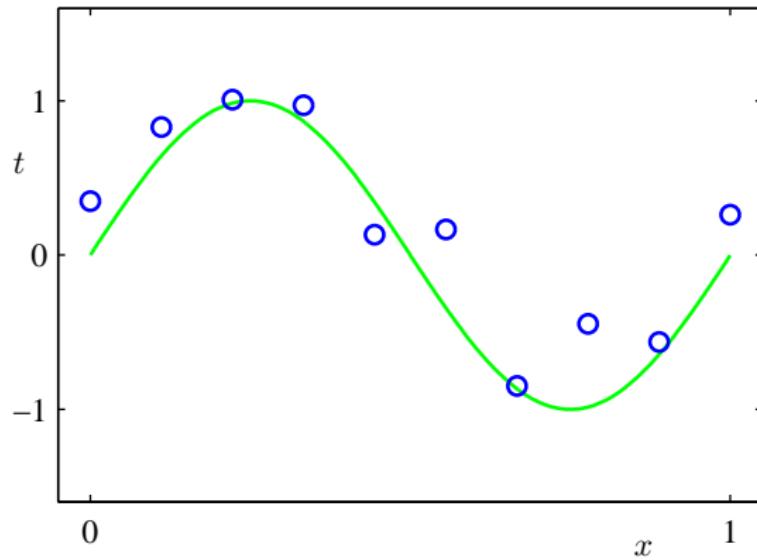
Today, we'll talk about **algorithm-independent principles** that concern several of the steps in the design cycle.

# Notation

- $L(\cdot, \cdot)$  - loss function
- $h(\mathbf{x}; \mathbf{w})$  - our hypothesis with parameters  $\mathbf{w}$  for input  $\mathbf{x}$   
(our model, was called  $f$  in previous lectures)
- $\hat{h}$  - estimated optimal hypothesis
- $h^*$  - optimal (generating) hypothesis
- $\mathcal{H}$  - hypothesis set under consideration

## Example: Polynomial Curve Fitting

Consider a curve fitting example where we are given a labeled data set of  $N$  examples  $\langle(x_i, y_i)\rangle_{i=1}^N$ . Labels are generated from the target function  $\sin(2\pi x)$  plus a bit of Gaussian noise.



# Example: Polynomial Curve Fitting

We will fit a polynomial regression model of the form:

$$h(x, \mathbf{w}) = w_0 + w_1 x^1 + w_2 x^2 + \dots + w_M x^M$$

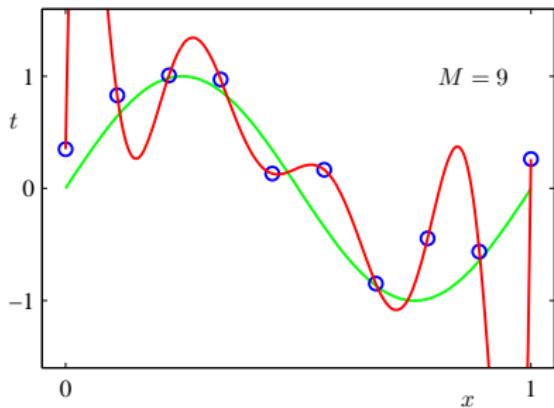
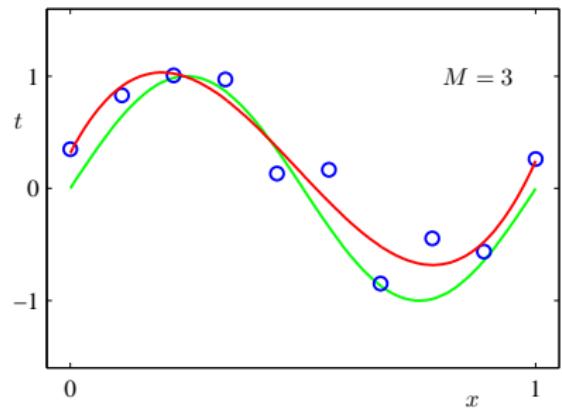
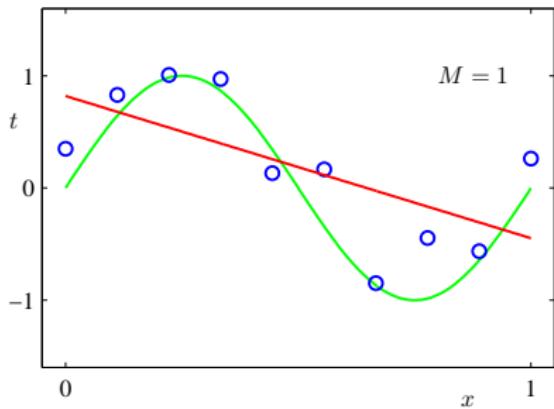
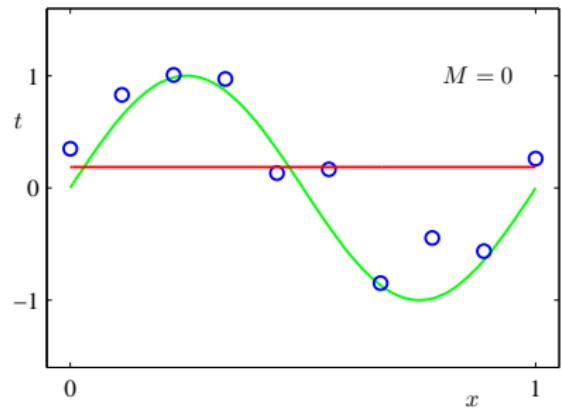
Or, using features  $\phi_j(x) = x^j$ , in basis function notation:

$$h(x, \mathbf{w}) = \sum_{j=0}^M w_j \phi_j = \mathbf{w}^T \phi(x)$$

How should we choose  $M$ ?



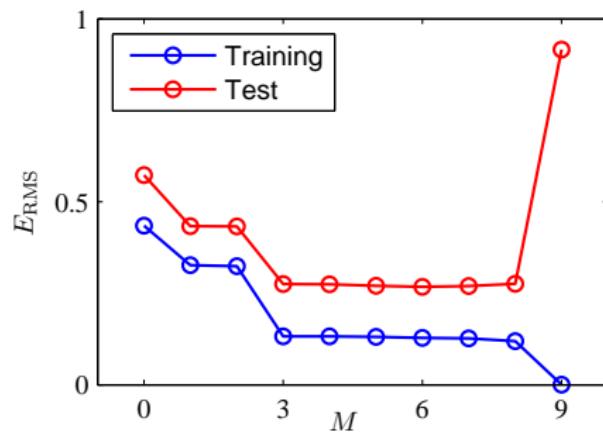
# How to choose $M$ ?



# Generalization Performance

We care about **good generalization**, i.e. accurate predictions for new data.

- Quantify dependence of generalization performance on  $M$  on separate test set



- Overfitting:** *Fitting the data more than is warranted*
- Problem: Flexible hypothesis class starts to fit the noise to reduce training error

# Approximation-Generalization Tradeoff

- More complex hypothesis class  $\rightsquigarrow$   
more flexibility to approximate target function
- Less complex hypothesis class  $\rightsquigarrow$   
more likely to generalize well on new data

Questions:

- How to quantify this tradeoff?
- What is the correct hypothesis class  $\mathcal{H}$  to choose from?

# Loss Function and Risk Minimization

In general, when taking a decision based on our learned model, we strive to minimize wrong decisions as measured by a user-defined **loss function** (e.g. squared error loss for regression).

The *expected* loss is also called **risk**, defined as:

$$R(h) = \mathbb{E}_{p(x,y)}[L(h(x), y)] = \int L(h(x), y)p(x, y)dxdy$$

This is the quantity we would ultimately like to minimize. The **optimal hypothesis** would then be:

$$h^* \in \arg \min_{h \in \mathcal{H}} R(h)$$

However,  $p(x, y)$  is generally unknown.

## Loss Function and Risk Minimization - contd

We approximate the risk with sampled data ([empirical risk](#)):

$$R_{\text{emp}}(h) = \frac{1}{N} \sum_{i=1}^N L(h(x_i), y_i)$$

The [estimated optimal hypothesis](#) is then:

$$\hat{h} \in \arg \min_{h \in \mathcal{H}} R_{\text{emp}}(h)$$

It depends, of course, on the available dataset:

$$\mathcal{D} = \{\mathbf{X}, \mathbf{y}\} = \langle (\mathbf{x}_i, y_i) \rangle_{i=1}^N \sim p(x, y)$$

Question:

- Can we estimate the error we make by this approximation?

# Lecture Overview

1 Complexity of Hypothesis Class

2 Bias - Variance Analysis

3 Validation Protocols

4 Summary

# Risk Decomposition

Decomposing the risk for regression with the squared error function:

$$L(h(x), y) = (\hat{h}(x) - y)^2$$

The risk can be decomposed into (somewhat lengthy derivation, see [Bishop, Section 1.5.5 and 3.2]):

$$\begin{aligned} R(h) &= \mathbb{E}_{p(x,y)}[L] \\ &= \int (\hat{h}(x) - h^*(x))^2 p(x) dx + \int (h^*(x) - y)^2 p(x, y) dx dy \end{aligned}$$

- First term can be minimized by choosing the right hypothesis.
- Second term is the noise in the data, independent of our hypothesis.

# Average Performance and Average Hypothesis

We focus on the first integrand and see what happens when we fit the hypothesis depending on datasets  $\hat{h}(x) = \hat{h}(x, \mathcal{D})$  trying to minimize the empirical risk  $R_{\text{emp}}$ .

- For different data sets, we obtain different prediction functions  $\hat{h}(x, \mathcal{D})$ .
- Results in different values of the loss; e.g. for data set  $\mathcal{D}$ , consider  $(\hat{h}(x, \mathcal{D}) - h^*(x))^2$ .

Suppose now, we had a large number of data sets  $\mathcal{D}$  of size  $N$ , each drawn independently from  $p(x, y)$ .

→ get a better performance estimate of the learning algorithm by considering its **average performance** over data sets, i.e. evaluating  $\mathbb{E}_{\mathcal{D}}[(\hat{h}(x, \mathcal{D}) - h^*(x))^2]$ .

## Average Performance and Average Hypothesis - contd

Define **average hypothesis**:  $\bar{h}(x) = \mathbb{E}_{\mathcal{D}}[\hat{h}(x, \mathcal{D})]$ . Now use this in the average performance:

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}[\{\hat{h}(\mathbf{x}; \mathcal{D}) - h^*(\mathbf{x})\}^2] &= \mathbb{E}_{\mathcal{D}}[\{\hat{h}(\mathbf{x}; \mathcal{D}) - \bar{h}(\mathbf{x}) + \bar{h}(\mathbf{x}) - h^*(\mathbf{x})\}^2] \\ &= \mathbb{E}_{\mathcal{D}}[\{\hat{h}(\mathbf{x}; \mathcal{D}) - \bar{h}(\mathbf{x})\}^2 + \{\bar{h}(\mathbf{x}) - h^*(\mathbf{x})\}^2 \\ &\quad + 2\{\hat{h}(\mathbf{x}; \mathcal{D}) - \bar{h}(\mathbf{x})\}\{\bar{h}(\mathbf{x}) - h^*(\mathbf{x})\}] \\ &= \underbrace{\{\bar{h}(\mathbf{x}) - h^*(\mathbf{x})\}^2}_{\text{(bias)}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[\{\hat{h}(\mathbf{x}; \mathcal{D}) - \bar{h}(\mathbf{x})\}^2]}_{\text{variance}}\end{aligned}$$

# Bias and Variance

Overall, we get:

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

where

$$(\text{bias})^2 = \int \{\bar{h}(x) - h^*(x)\}^2 p(x) dx$$

$$\text{variance} = \int \mathbb{E}_{\mathcal{D}}[\{\hat{h}(\mathbf{x}; \mathcal{D}) - \bar{h}(\mathbf{x})\}^2] p(x) dx$$

$$\text{noise} = \int \{h^*(\mathbf{x}) - y\}^2 p(x, y) dx dy$$

where  $\bar{h}(x) = \mathbb{E}_{\mathcal{D}}[\hat{h}(x, \mathcal{D})]$  is the average hypothesis in expectation over many datasets drawn from  $\mathcal{D}$

→ Bias-Variance decomposition lets us judge model class complexity by the data resources available for training

# Bias - Variance: Examples

a)

b)

c)

d)

$$g(x) = \text{fixed}$$

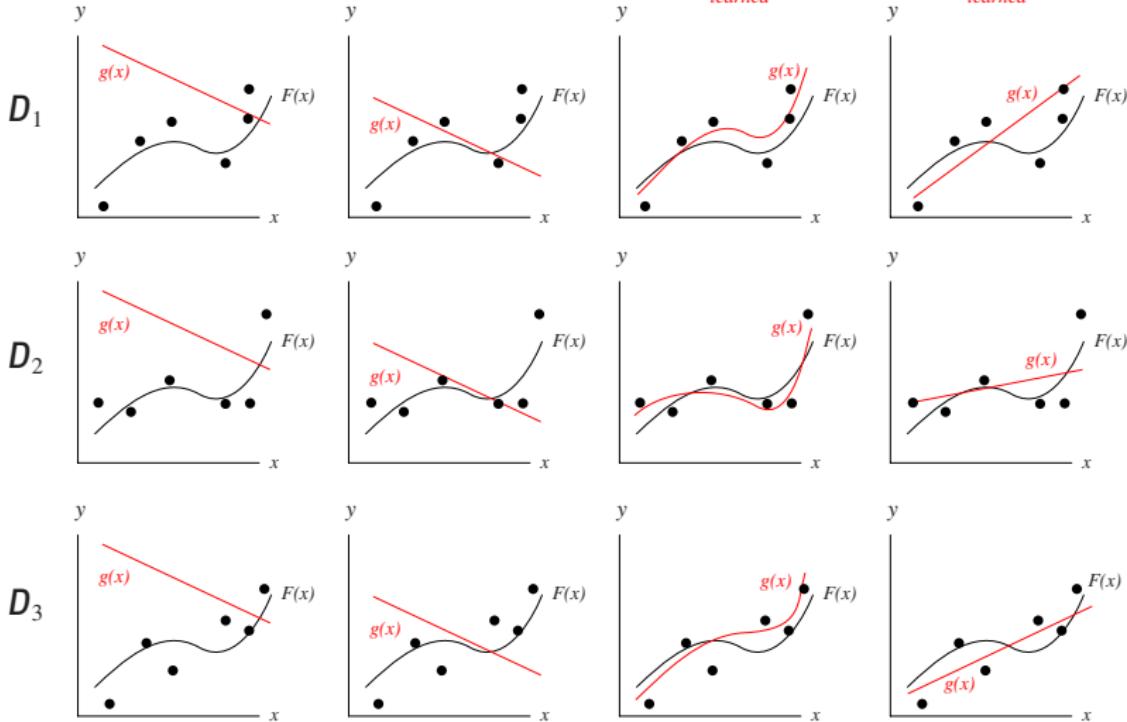
$$g(x) = \text{fixed}$$

$$g(x) = a_0 + a_1x + a_2x^2 + a_3x^3$$

learned

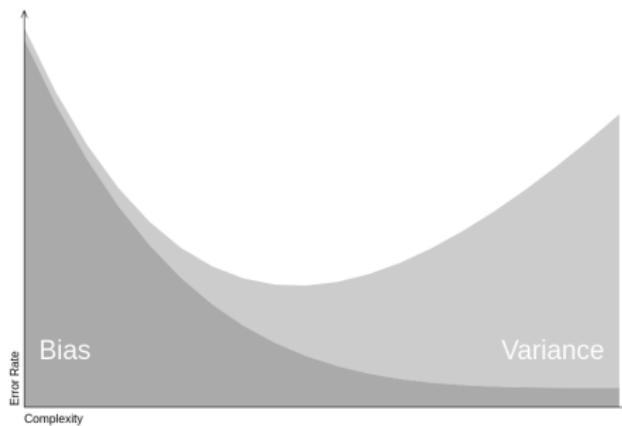
$$g(x) = a_0 + a_1x$$

learned



# Approximation-Generalization Tradeoff – Revisited

- More complex models → more flexibility to approximate target function (**low bias, high variance**)
- Less complex models → more likely to generalize well on new data (**high bias, low variance**)



Balancing bias and variance gives **optimal predictive capability**.

# Remarks on Bias-Variance-Analysis

Bias-Variance-Analysis is not very useful in practice, because ...

- ① requires the ground truth for the bias and noise terms
- ② splitting the data into many datasets to estimate the variance

In practice, we need something feasible!

# Lecture Overview

1 Complexity of Hypothesis Class

2 Bias - Variance Analysis

3 Validation Protocols

4 Summary

# Validation: Hold-out

- Validation is the process to check the performance of a learned function on independent validation data.
- Hold-out validation:
  - split the available data from the underlying distribution  $p(x, y)$  into two disjoint subsets: training set  $\mathcal{D}_{\text{train}}$  and validation set  $\mathcal{D}_{\text{validation}}$
  - train your model only on the training set
  - evaluate empirical risk of the learned hypothesis on the validation set
- Remark: Since  $\mathcal{D}_{\text{validation}}$  is often used to optimize hyperparameters, it's crucial to also keep yet another separate test set  $\mathcal{D}_{\text{test}}$  to test final performance.

## Remarks: Hold-out

- There is no clear answer on how many percent of the data should go into  $\mathcal{D}_{\text{train}}$  and how many into  $\mathcal{D}_{\text{validation}}$
- Two problematic cases:
  - $\mathcal{D}_{\text{validation}}$  is too small  $\rightsquigarrow$  your loss estimate will be poor
  - $\mathcal{D}_{\text{train}}$  is too small  $\rightsquigarrow$  your estimated loss will be too pessimistic because you used less data for training than you actually would use in practice
- if your  $\mathcal{D}$  is quite small, different hold-out splits might give your different results
- **Recommendation:** Only use hold-out validation if you have a large dataset where you can afford to put some data to the side and if a different split of your data will likely lead to similar results

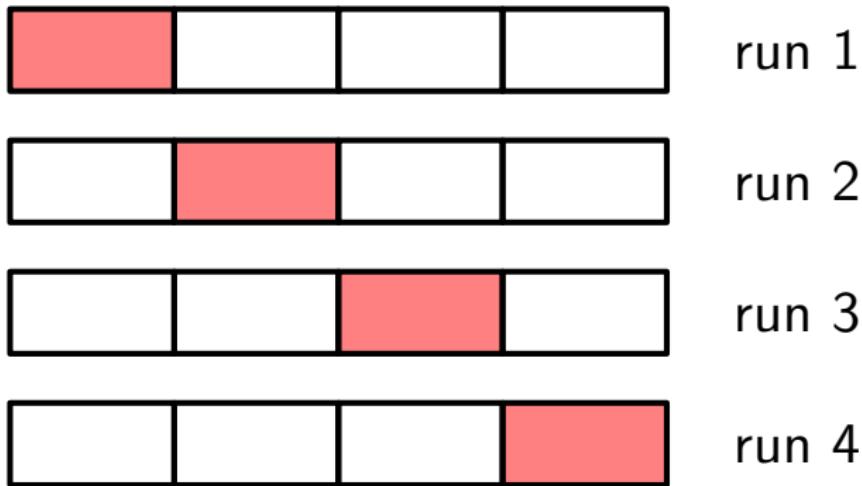
# k-fold Cross-Validation

**Require:**  $\mathcal{D}$  with  $N$  data points,  $2 \leq k \leq N$

- 1: split  $\mathcal{D}$  into  $k$  disjoint subsets of (roughly) equal size:  $\mathcal{D}_1, \dots, \mathcal{D}_k$
- 2: **for**  $i = 1$  to  $k$  **do**
- 3:   (re-) initialize hypothesis parameters
- 4:   train hypothesis on set  $\mathcal{D}_1 \cup \dots \cup \mathcal{D}_{i-1} \cup \mathcal{D}_{i+1} \cup \dots \cup \mathcal{D}_k$
- 5:   calculate empirical risk  $R_{\text{emp}}^i$  on  $\mathcal{D}_i$
- 6: **end for**
- 7: **return**  $\frac{1}{k} \sum_{i=1}^k R_{\text{emp}}^i$

- Advantage: each hypothesis is learned on most data points ( $\frac{k-1}{k}N$ ) and every data point is used at some point for validation
- Disadvantage: hypothesis has to be learned  $k$  times
- Recommendation: quite common to set  $k$  to 10

# Cross-Validation - Illustration



# Cross-Validation - Example

- Example:  $\mathcal{D} = \{z_1, \dots, z_{17}\}$ , 3-fold crossvalidation
- Split training set randomly:

$$\mathcal{D}_1 = \{z_1, z_2, z_8, z_{13}, z_{14}, z_{15}\}$$

$$\mathcal{D}_2 = \{z_4, z_5, z_{10}, z_{11}, z_{12}, z_{17}\}$$

$$\mathcal{D}_3 = \{z_3, z_6, z_7, z_9, z_{16}\}$$

- Train the model three times:
  1. use  $\mathcal{D}_2 \cup \mathcal{D}_3$  for training,  $\mathcal{D}_1$  for validation: empirical risk  $R_{\text{emp}}^1$
  2. use  $\mathcal{D}_1 \cup \mathcal{D}_3$  for training,  $\mathcal{D}_2$  for validation: empirical risk  $R_{\text{emp}}^2$
  3. use  $\mathcal{D}_1 \cup \mathcal{D}_2$  for training,  $\mathcal{D}_3$  for validation: empirical risk  $R_{\text{emp}}^3$
- Calculate cross-validation loss (average empirical risk over folds):

$$R_{\text{cv}} = \frac{R_{\text{emp}}^1 + R_{\text{emp}}^2 + R_{\text{emp}}^3}{3}$$

## Leave-One-Out (LOO)

- The extreme case of cross-validation is to set  $k$  to  $|\mathcal{D}|$ 
  - ↝ in each iteration, only a single point is used for validation
- Super expensive!
  - you can only use it on very small datasets
- Nevertheless, it is a good estimate of your loss since you use  $|\mathcal{D}| - 1$  points for training
- (the method is also known as jackknifing )

# Out-of-Bootstrap Validation

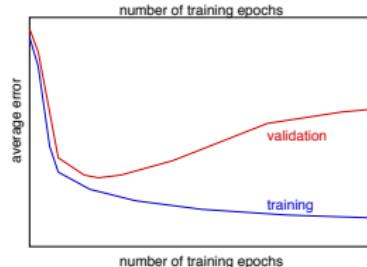
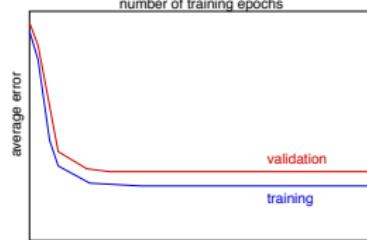
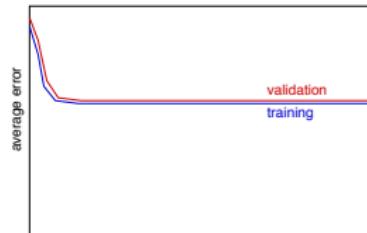
- cross-validation has the problem that the splits are not independent, and thus the loss estimates are not independent
  - ⇝ you cannot apply some tests on these statistics (e.g., statistical hypothesis tests)
- Out-of-Bootstrap Validation:
  - ① Bootstrap sample (with replacement!) from  $\mathcal{D}$
  - ② All samples not being in the training set are put into the test set
- by repeating the above procedure, we get independent estimates of the loss
- Problem:  
since we sample with replacement, we change the training distribution

# Overfitting / Underfitting: Training and Validation error behavior

Example of underfitting:  
validation and training error remain  
large  
(often too large bias in hypothesis)

Example of successful learning:  
validation error and training error  
monotonically decrease

Example of overfitting:  
validation error increases while  
training error decreases  
(often too large variance in  
hypothesis)



# Lecture Overview

1 Complexity of Hypothesis Class

2 Bias - Variance Analysis

3 Validation Protocols

4 Summary

## Summary by learning goals

Having heard this lecture, you can now ...

- describe what **overfitting** means and why it happens
- explain the **approximation-generalization tradeoff**
- quantify the approximation-generalization tradeoff using the **bias-variance analysis**
- explain different **validation policies** for estimating your empirical loss

# Lecture 8: Algorithm Independent Principles - II

## Validation, Regularization, General Issues, Model Selection

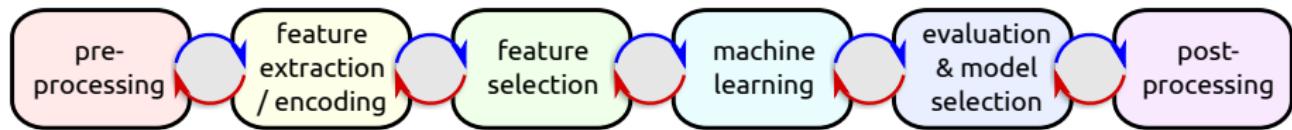
Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# The Big Picture



# Lecture Overview

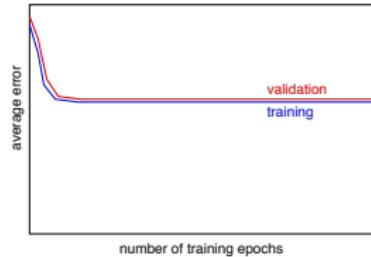
- ① Validation (cont'd)
- ② Regularization
- ③ General Considerations
- ④ Model Selection and Feature Selection
- ⑤ Wrapup

# Lecture Overview

- 1 Validation (cont'd)
- 2 Regularization
- 3 General Considerations
- 4 Model Selection and Feature Selection
- 5 Wrapup

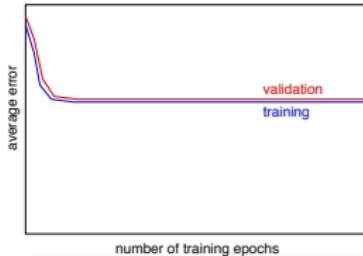
# Recap: Overfitting / Underfitting

Example of underfitting:  
validation and training error remain  
large

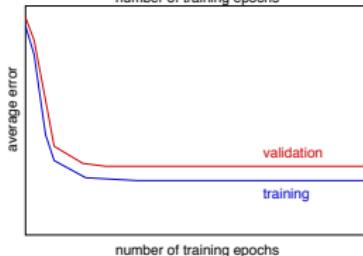


# Recap: Overfitting / Underfitting

Example of underfitting:  
validation and training error remain  
large

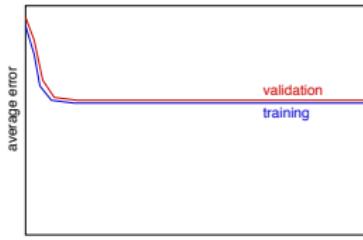


Example of successful learning:  
validation error and training error  
monotonically decrease  
~~ good generalization

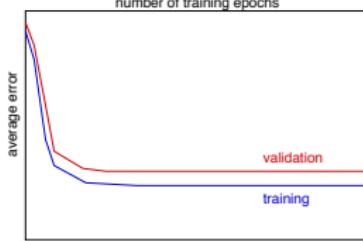


# Recap: Overfitting / Underfitting

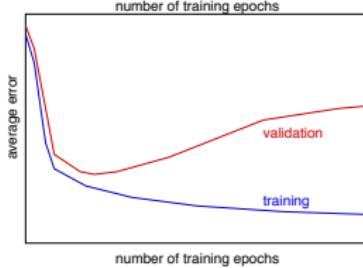
Example of underfitting:  
validation and training error remain  
large



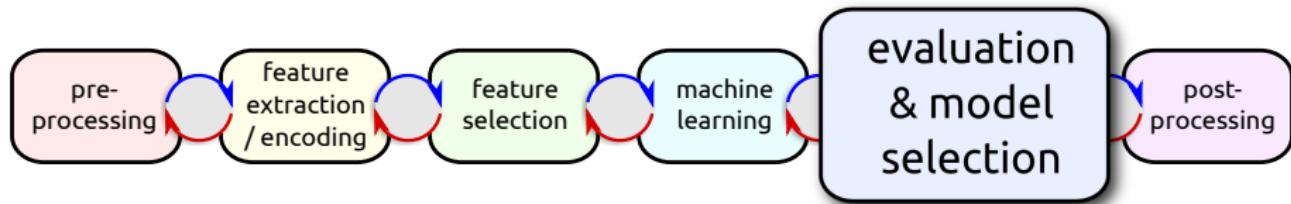
Example of successful learning:  
validation error and training error  
monotonically decrease  
~~ good generalization



Example of overfitting:  
validation error increases while  
training error decreases



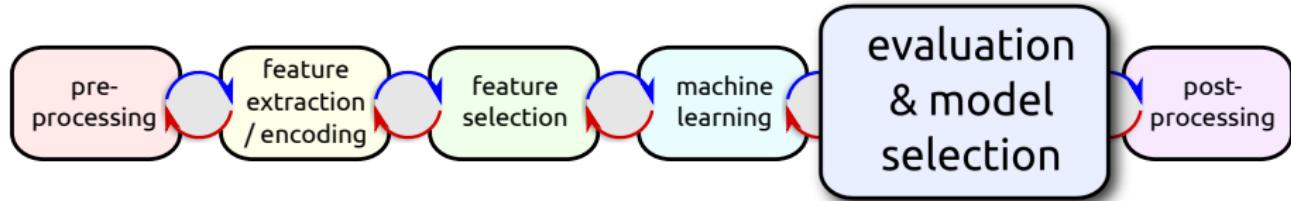
# Model Selection 1/2



## Evaluation:

- We want our models to **generalize**
  - I.e., perform well on previously unseen data points
- What does it mean to perform well?
  - See metrics covered later today
  - Speed at training time, speed at test time, memory, accuracy, ...

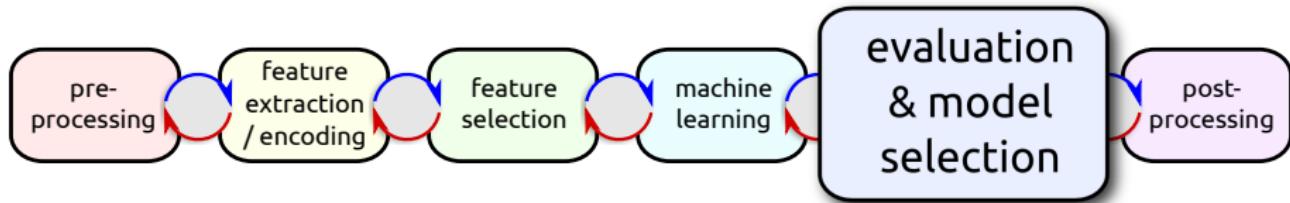
## Model Selection 2/2



To obtain estimates of generalization performance using a fixed dataset

- Split dataset into **training set** and **test set**
- **Lock away test set for final assessment**

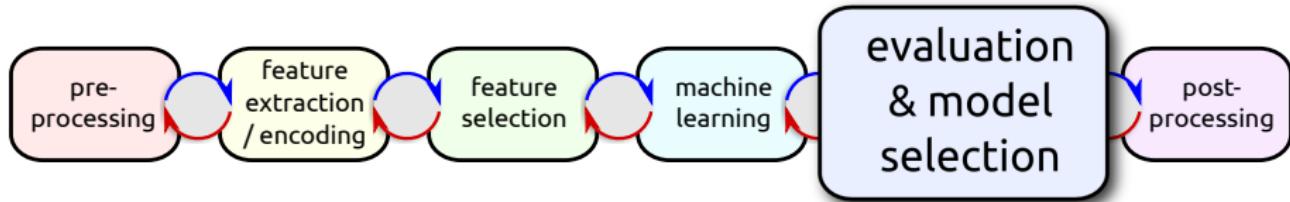
## Model Selection 2/2



To obtain estimates of generalization performance using a fixed dataset

- Split dataset into **training set** and **test set**
- **Lock away test set** for final assessment
- You can do anything you want on the training set
- E.g., split it further into training and validation
  - Train different models on training set
  - Pick the one with best performance on validation set

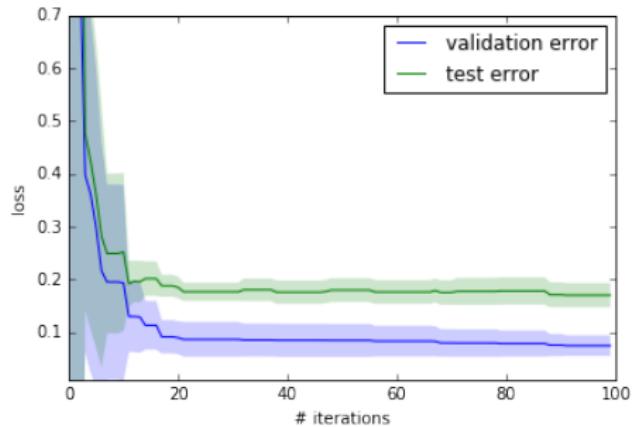
## Model Selection 2/2



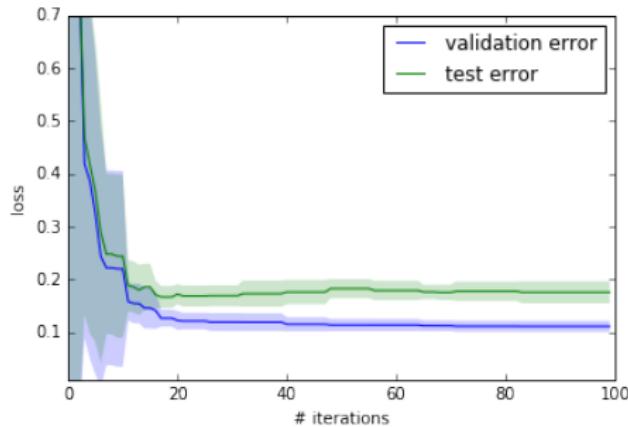
To obtain estimates of generalization performance using a fixed dataset

- Split dataset into **training set** and **test set**
- **Lock away test set** for final assessment
- You can do anything you want on the training set
- E.g., split it further into training and validation
  - Train different models on training set
  - Pick the one with best performance on validation set
- E.g., split it further into cross-validation (CV) folds and pick the model with best CV performance
  - CV performance is not an unbiased estimate of test performance
  - But better estimate than a single split into training/valid

# Cross Validation Can Still Overfit 1/2



Single training-validation split (90% - 10%)



10-fold cross-validation

- Validation performance from single training-validation split is **overconfident** (overly optimistic)
- CV performance is still overconfident, but not as much
- In one of the next assignments, you will basically create these plots

## Cross Validation Can Still Overfit 2/2

- To overfit least, when and how would you choose between different preprocessors (or feature selectors, data normalizers, etc) when you use  $k$ -fold cross-validation?
  - ★ Once: in the beginning, on all data
  - ★ Once: on all the training data
  - ★  $k$  times: on the training split of each CV fold

## Cross Validation Can Still Overfit 2/2

- To overfit least, when and how would you choose between different preprocessors (or feature selectors, data normalizers, etc) when you use  $k$ -fold cross-validation?
  - ★ Once: in the beginning, on all data
  - ★ Once: on all the training data
  - ★  $k$  times: on the training split of each CV fold
- Typical method `trainModel`
  - Normalizes data, drops unimportant features, etc
  - Then builds model on preprocessed data
  - Saves both these data transformations and the model

## Cross Validation Can Still Overfit 2/2

- To overfit least, when and how would you choose between different preprocessors (or feature selectors, data normalizers, etc) when you use  $k$ -fold cross-validation?
  - ★ Once: in the beginning, on all data
  - ★ Once: on all the training data
  - ★  $k$  times: on the training split of each CV fold
- Typical method `trainModel`
  - Normalizes data, drops unimportant features, etc
  - Then builds model on preprocessed data
  - Saves both these data transformations and the model
- Typical method `applyModel`
  - First apply the transformations, then the model
  - Routine used for both the validation set and the test set alike

# Stratified Cross Validation

- E.g., only 10 positive data points, 90 negative ones
- How many positive data points would standard 10-fold cross-validation put into each fold?
  - ★ 1
  - ★ 10
  - ★ Between 0 and 10, depends on the random split into folds

# Stratified Cross Validation

- E.g., only 10 positive data points, 90 negative ones
- How many positive data points would standard 10-fold cross-validation put into each fold?
  - ★ 1
  - ★ 10
  - ★ Between 0 and 10, depends on the random split into folds

Stratified cross-validation would put exactly 1 positive and 9 negative data points into each fold

# Lecture Overview

- 1 Validation (cont'd)
- 2 Regularization
- 3 General Considerations
- 4 Model Selection and Feature Selection
- 5 Wrapup

# Regularization

Approaches to improve generalization

Which approaches do you know / can you think of?



# Regularization - Overview

Approaches to improve generalization

# Regularization - Overview

Approaches to improve generalization

Preference for small  
parameter values

- Early stopping
- Shrinkage methods

# Regularization - Overview

Approaches to improve generalization

Preference for small parameter values

- Early stopping
- Shrinkage methods

Better task description

- More training data
- Filter training data
- Use more / less / other input features

# Regularization - Overview

Approaches to improve generalization

Preference for small parameter values

- Early stopping
- Shrinkage methods

Better task description

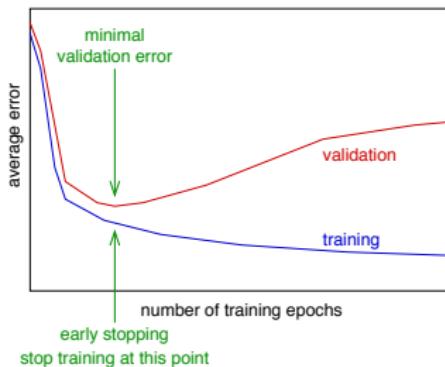
- More training data
- Filter training data
- Use more / less / other input features

Ensemble techniques

- Bagging

# Regularization Techniques - Early Stopping

- Stop learning when error on validation set has reached its minimum
- Often, training is already stopped after a few epochs
  - typically combined with small initial weights
- Simple, popular heuristic
- Needs perpetual observation of the validation error



# Regularization Techniques - Shrinkage Methods

- Extend the loss function with extra term (penalty) to control overfitting

$$L(\mathbf{w}) = L_D(\mathbf{w}) + \lambda L_W(\mathbf{w})$$

# Regularization Techniques - Shrinkage Methods

- Extend the loss function with extra term (penalty) to control overfitting

$$L(\mathbf{w}) = L_D(\mathbf{w}) + \lambda L_W(\mathbf{w})$$

- Common example: sum-of-squares loss function with L2 regularization

$$L(\mathbf{w}) = \underbrace{\frac{1}{2} \sum_{n=1}^N \{y_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2}_{L_D(\mathbf{w})} + \underbrace{\frac{\lambda}{2} \mathbf{w}^T \mathbf{w}}_{\lambda L_W(\mathbf{w})}$$

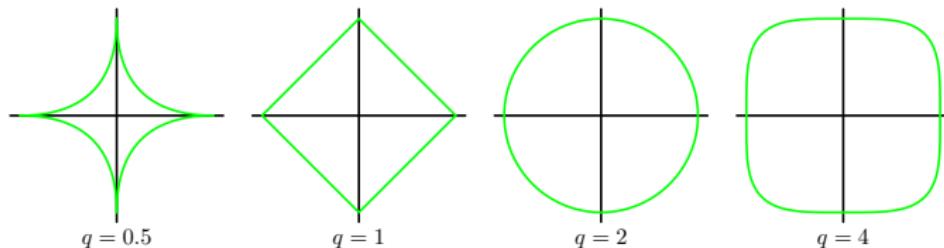
- Allows for closed-form solution:  $\mathbf{w} = (\lambda \mathbf{I} + \Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$ . This is called **ridge-regression** or **Tikhonov regularization** in the literature.

# Regularization Techniques - Shrinkage Methods

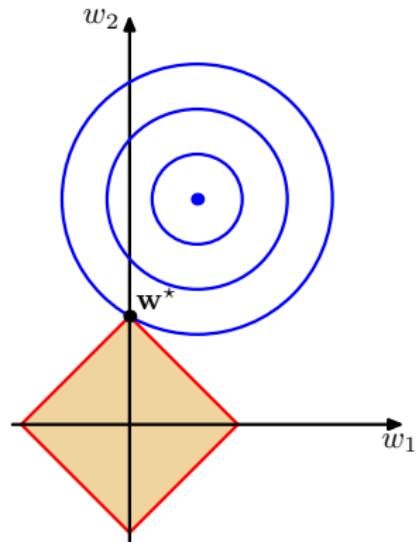
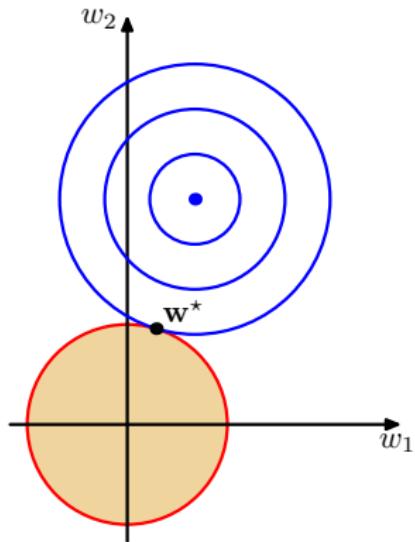
- More general form:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q$$

- Shrinkage**: encourages weights to shrink towards zero
- Case  $q = 2$ : L2 regularizer as before
- Case  $q = 1$ : L1 regularizer as before (**lasso** in the literature)  
→ sparse solutions



# Regularization techniques - Shrinkage Methods



# Regularization Techniques - Shrinkage Methods

$$L(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q$$

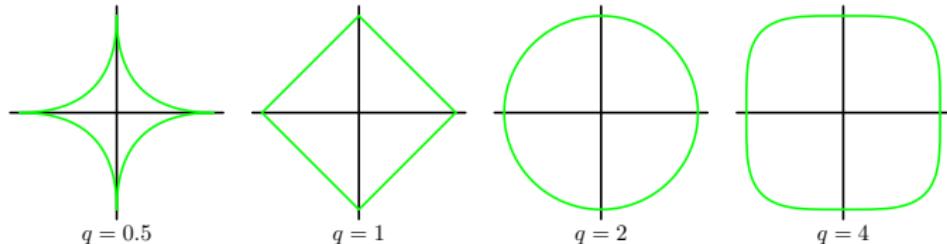
- Which of these values of  $q$  encourages sparsity the most?
  - ★  $q=0.5$
  - ★  $q=1$
  - ★  $q=2$
  - ★  $q=4$

# Regularization Techniques - Shrinkage Methods

$$L(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q$$

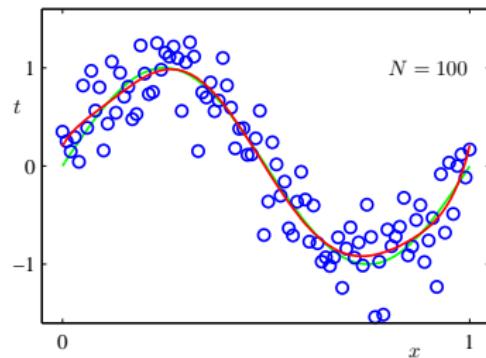
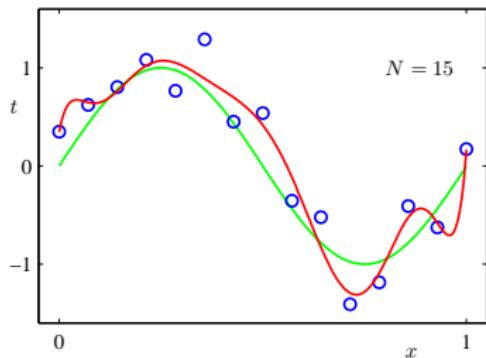
- Which of these values of  $q$  encourages sparsity the most?

- ★  $q=0.5$
- ★  $q=1$
- ★  $q=2$
- ★  $q=4$



# Regularization Techniques - More Data

- Data analysts' fundamental slogan: *there's no data like more data!*
- Try to get more data; if not possible directly, think about related sources of similar data
- Although trivial, one of the most important techniques to improve ML models



# Regularization Techniques - Data Augmentation

- Data augmentation is a common strategy for creating additional training data
- Example: computer vision
  - Translation, Scaling, Reflection, Rotation, Stretching



# Regularization Techniques - Data Augmentation

- Data augmentation is a common strategy for creating additional training data
- Example: computer vision
  - Translation, Scaling, Reflection, Rotation, Stretching



# Regularization Techniques - Data Augmentation

- Data augmentation is a common strategy for creating additional training data
- Example: computer vision
  - Translation, Scaling, Reflection, Rotation, Stretching



# Regularization Techniques - Data Augmentation

- Data augmentation is a common strategy for creating additional training data
- Example: computer vision
  - Translation, Scaling, Reflection, Rotation, Stretching



# Regularization Techniques - Data Augmentation

- Data augmentation is a common strategy for creating additional training data
- Example: computer vision
  - Translation, Scaling, Reflection, Rotation, Stretching



# Regularization Techniques - Data Augmentation

- Data augmentation is a common strategy for creating additional training data
- Example: computer vision
  - Translation, Scaling, Reflection, Rotation, Stretching



- Hypothesis learns representation **invariant** to the perturbations
  - Often yields very substantial improvements
  - However: 100x augmentation can slow down training 100x
  - Diminishing returns: rarely more than 100x data augmentation

# Regularization Techniques - Data Augmentation

- Data augmentation is a common strategy for creating additional training data
- Example: computer vision
  - Translation, Scaling, Reflection, Rotation, Stretching



- Hypothesis learns representation **invariant** to the perturbations
  - Often yields very substantial improvements
  - However: 100x augmentation can slow down training 100x
  - Diminishing returns: rarely more than 100x data augmentation
- What data do we apply these augmentations to?
  - ★ All data
  - ★ Only the test data
  - ★ Only the validation data
  - ★ Only the training data

# Regularization Techniques - Data Augmentation

- We only want to be invariant to **certain degrees** of transformations
  - Like the human visual system; not these:



# Regularization Techniques - Data Augmentation

- We only want to be invariant to **certain degrees** of transformations
  - Like the human visual system; not these:



# Regularization Techniques - Data Augmentation

- We only want to be invariant to **certain degrees** of transformations
  - Like the human visual system; not these:



- **Hyperparameters:** how much of each perturbation to apply?
  - Usually specify a distribution to sample from
  - E.g., zero-mean 5-dimensional Gaussian with degrees of translation, scaling, reflection, rotation, stretching to apply to original data
  - Best hyperparameter setting: most helpful invariants

# Regularization Techniques - Data Augmentation

- We only want to be invariant to **certain degrees** of transformations
  - Like the human visual system; not these:



- **Hyperparameters:** how much of each perturbation to apply?
  - Usually specify a distribution to sample from
  - E.g., zero-mean 5-dimensional Gaussian with degrees of translation, scaling, reflection, rotation, stretching to apply to original data
  - Best hyperparameter setting: most helpful invariants
- In computer vision, it is not hard to find useful image perturbations
- What could perturbations be in other applications? Domain-specific!
  - Could even lead to domain-specific insights about important invariants

# Filtering Training Data

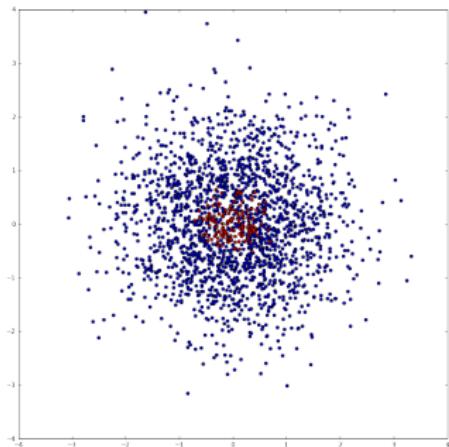
- Some training data points are much more helpful to learn the target concept than others
- **Filtering** means reducing the training set to the really important points that help adjusting the classification boundary/regression curve
- Techniques: oversampling, subsampling, outlier rejection, jittering
- Frequent problem: imbalanced data in classification

# Balancing Data

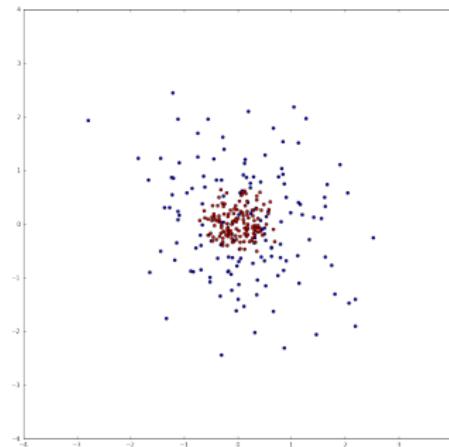
- E.g., 93% negative data, 7% positive
  - You could trivially get 93% accuracy
  - But you may want also want high recall

# Balancing Data

- E.g., 93% negative data, 7% positive
  - You could trivially get 93% accuracy
  - But you may want also want high recall



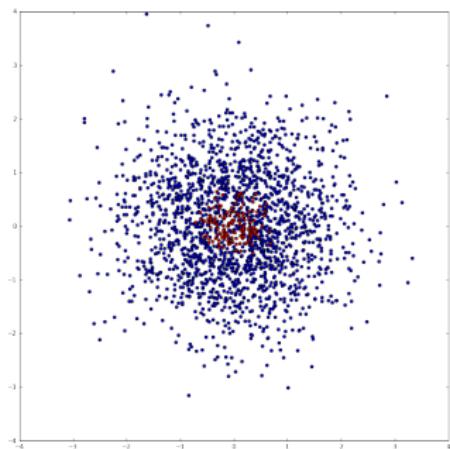
Original data



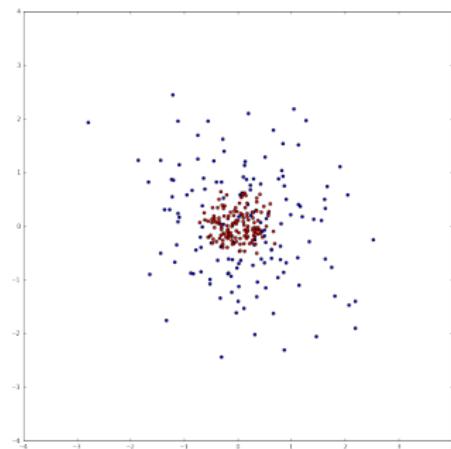
Subsampled majority class

# Balancing Data

- E.g., 93% negative data, 7% positive
  - You could trivially get 93% accuracy
  - But you may want also want high recall



Original data



Subsampled majority class

- Solution 1: If your classifier supports it, weight your data
- Solution 2: Subsample the majority class
- Solution 3: Generate new data points of minority class

# Regularization Techniques - Input Features

- Removing features may reduce overfitting by helping the model avoid fitting pseudo relationships
  - Extreme: think of a feature that is just random noise
- Dimensionality reduction is related: PCA, ICA

# Regularization Techniques - Input Features

- Removing features may reduce overfitting by helping the model avoid fitting pseudo relationships
  - Extreme: think of a feature that is just random noise
- Dimensionality reduction is related: PCA, ICA
- Of course, adding features can also help improve performance
  - If they are related to the desired output
  - Very often, domain experts can come up with useful additional features
  - Can also add non-linear transformations of features

# Regularization Techniques - Committee / Ensemble Approaches

- If you ask one expert, the expert may fail
- Ask a committee of experts: the majority has a better chance to be right

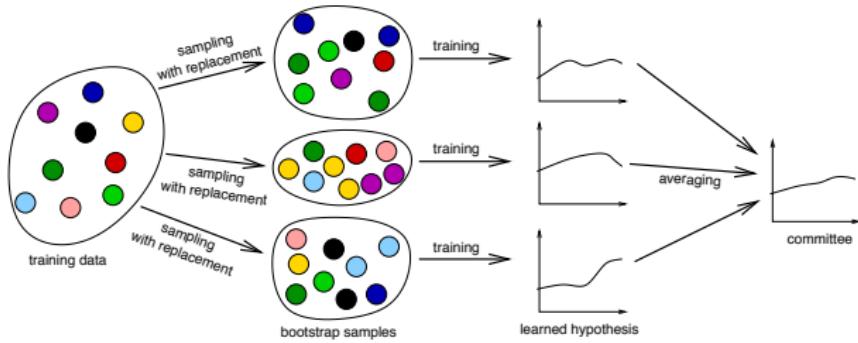
# Regularization Techniques - Committee / Ensemble Approaches

- If you ask one expert, the expert may fail
- Ask a committee of experts: the majority has a better chance to be right
- Premise: experts are experienced and diverse



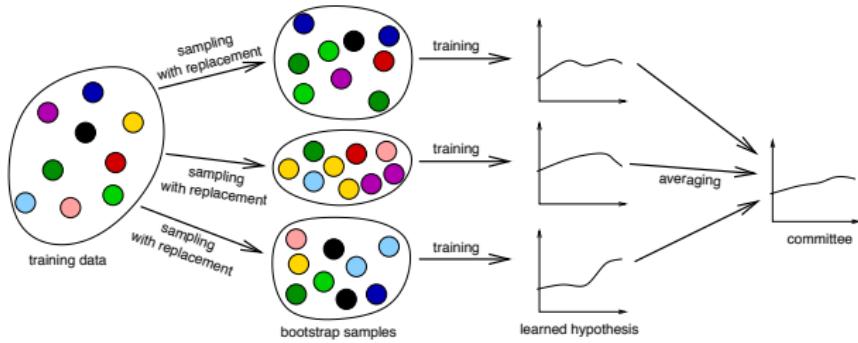
# Committees - Bagging [Breimann, 1996]

- Train several models on bootstrap samples of the training data
  - Data is drawn randomly with replacements  $\Rightarrow$  some data points may occur twice or more, others don't occur at all
- Average the output of all trained models



# Committees - Bagging [Breimann, 1996]

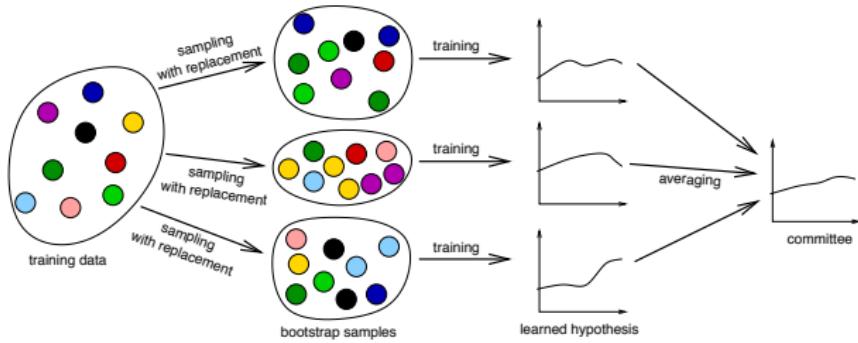
- Train several models on bootstrap samples of the training data
  - Data is drawn randomly with replacements  $\Rightarrow$  some data points may occur twice or more, others don't occur at all
- Average the output of all trained models



- Single members of the committee might produce a higher test-set error; but the committee error can still improve

# Committees - Bagging [Breimann, 1996]

- Train several models on bootstrap samples of the training data
  - Data is drawn randomly with replacements  $\Rightarrow$  some data points may occur twice or more, others don't occur at all
- Average the output of all trained models



- Single members of the committee might produce a higher test-set error; but the committee error can still improve
- We'll cover committee methods / ensemble methods in more detail in the module on tree-based methods

# Lecture Overview

- 1 Validation (cont'd)
- 2 Regularization
- 3 General Considerations
- 4 Model Selection and Feature Selection
- 5 Wrapup

# Metrics of Success 1/3

- In different applications, different metrics of success apply
- Example metrics for classification
  - E.g., 0/1 loss, higher loss for false positives than false negatives, etc
  - E.g., AUC, F1, precision/recall, ... (see next slide)

# Metrics of Success 1/3

- In different applications, different metrics of success apply
- Example metrics for classification
  - E.g., 0/1 loss, higher loss for false positives than false negatives, etc
  - E.g., AUC, F1, precision/recall, ... (see next slide)
- Example metrics for regression
  - E.g., RMSE (root mean squared error)
  - E.g., MAE (mean absolute error)
  - Log-likelihood:  $\log(P(\text{observations} \mid \text{model}))$

# Metrics of Success 1/3

- In different applications, different metrics of success apply
- Example metrics for classification
  - E.g., 0/1 loss, higher loss for false positives than false negatives, etc
  - E.g., AUC, F1, precision/recall, ... (see next slide)
- Example metrics for regression
  - E.g., RMSE (root mean squared error)
  - E.g., MAE (mean absolute error)
  - Log-likelihood:  $\log(P(\text{observations} \mid \text{model}))$
- Internal loss function being optimized often differs from external metric of success
  - E.g., internally, to fit the model, we may need a differentiable **surrogate loss** function, such as cross entropy
  - E.g., **0/1 loss is not differentiable**

## Metrics of Success 2/3

- Binary classification: detecting a signal (e.g., disease)
  - True positive rate, sensitivity or recall:  
percentage of positive data points that are classified as positive  
→ want this to be high

## Metrics of Success 2/3

- Binary classification: detecting a signal (e.g., disease)
  - True positive rate, sensitivity or recall:  
percentage of positive data points that are classified as positive  
→ want this to be high
  - False positive rate:  
percentage of negative data points that are classified as positive  
→ want this to be low

# Metrics of Success 2/3

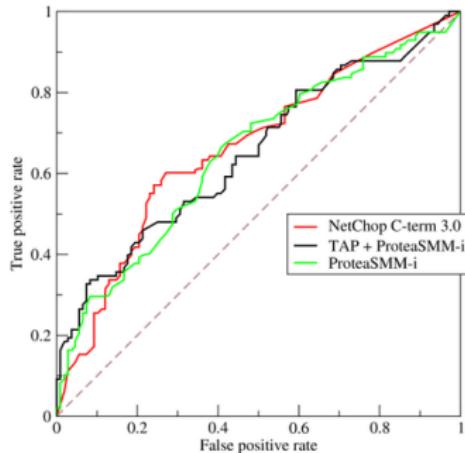
- Binary classification: detecting a signal (e.g., disease)
  - True positive rate, sensitivity or recall:  
percentage of positive data points that are classified as positive  
→ want this to be high
  - False positive rate:  
percentage of negative data points that are classified as positive  
→ want this to be low
  - Precision: percentage of positively-classified data points that are truly positive  
→ want this to be high

# Metrics of Success 2/3

- Binary classification: detecting a signal (e.g., disease)
  - True positive rate, sensitivity or recall:  
percentage of positive data points that are classified as positive  
→ want this to be high
  - False positive rate:  
percentage of negative data points that are classified as positive  
→ want this to be low
  - Precision: percentage of positively-classified data points that are truly positive  
→ want this to be high
  - F-measure or F<sub>1</sub> score:  $F = 2 \cdot (\text{precision} \cdot \text{recall}) / (\text{precision} + \text{recall})$

# Metrics of Success 3/3

- Receiver operating characteristic (ROC curve)
  - Plots true positive rate vs. false positive rate
  - At various threshold settings of a classifier
  - Single number for a classifier: Area Under the ROC Curve (AUC)

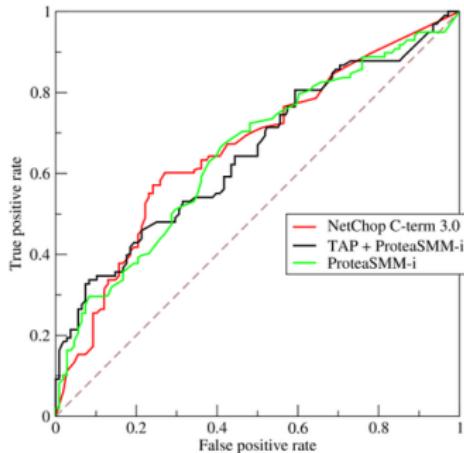


ROC curve of three predictors of peptide cleaving in the proteasome.

Source: [https://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](https://en.wikipedia.org/wiki/Receiver_operating_characteristic)

# Metrics of Success 3/3

- Receiver operating characteristic (ROC curve)
  - Plots true positive rate vs. false positive rate
  - At various threshold settings of a classifier
  - Single number for a classifier: Area Under the ROC Curve (AUC)



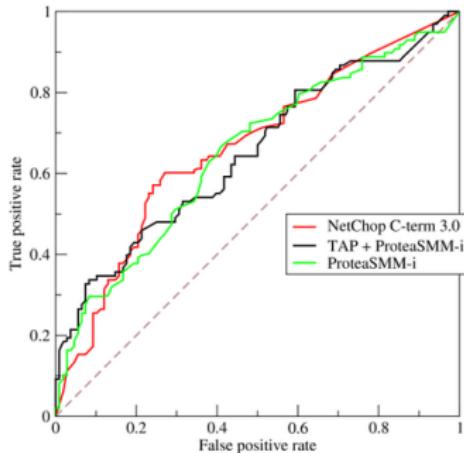
ROC curve of three predictors of peptide cleaving in the proteasome.

Source: [https://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](https://en.wikipedia.org/wiki/Receiver_operating_characteristic)

- With which algorithm can you trivially achieve the diagonal? 

# Metrics of Success 3/3

- Receiver operating characteristic (ROC curve)
  - Plots true positive rate vs. false positive rate
  - At various threshold settings of a classifier
  - Single number for a classifier: Area Under the ROC Curve (AUC)



ROC curve of three predictors of peptide cleaving in the proteasome.

Source: [https://en.wikipedia.org/wiki/Receiver\\_operating\\_characteristic](https://en.wikipedia.org/wiki/Receiver_operating_characteristic)

- With which algorithm can you trivially achieve the diagonal? 
- Another, similar type of curve: precision-recall curve

# Metrics of Success: Take Home Message

- There are several metrics of success
- The correct metric depends on your application
- For example, for spam classification you should not only consider accuracy
  - The predictor should not classify an important message as spam
  - similar cases in medicine

# Metrics of Success: Take Home Message

- There are several metrics of success
- The correct metric depends on your application
- For example, for spam classification you should not only consider accuracy
  - The predictor should not classify an important message as spam
  - similar cases in medicine
- In case of doubt, you should consider several metrics

# The i.i.d. Assumption

- So far, we've assumed an underlying distribution  $p(\mathbf{x}, y)$
- More precisely, we made the **standard assumption in supervised learning**: data points  $p(\mathbf{x}, y)$  are **independently and identically distributed** (i.i.d.)

# The i.i.d. Assumption

- So far, we've assumed an underlying distribution  $p(x, y)$
- More precisely, we made the **standard assumption in supervised learning**: data points  $p(x, y)$  are **independently and identically distributed (i.i.d.)**
  - **Independent**: learning the label A of one data point doesn't tell you anything about the label B of another data point:  
$$P(x \geq A) = P(x \geq A \mid y \geq B), \text{ for all } x \text{ and } y$$

# The i.i.d. Assumption

- So far, we've assumed an underlying distribution  $p(x, y)$
- More precisely, we made the **standard assumption in supervised learning**: data points  $p(x, y)$  are **independently and identically distributed (i.i.d.)**
  - **Independent**: learning the label A of one data point doesn't tell you anything about the label B of another data point:  
$$P(x \geq A) = P(x \geq A \mid y \geq B), \text{ for all } x \text{ and } y$$
  - **Identically distributed**: data points (including their labels) are drawn from the same probability distribution:  
$$P(x \geq A) = P(x \geq B), \text{ for all } x$$

# The i.i.d. Assumption

- So far, we've assumed an underlying distribution  $p(x, y)$
- More precisely, we made the **standard assumption in supervised learning**: data points  $p(x, y)$  are **independently and identically distributed (i.i.d.)**
  - **Independent**: learning the label A of one data point doesn't tell you anything about the label B of another data point:  
$$P(x \geq A) = P(x \geq A \mid y \geq B), \text{ for all } x \text{ and } y$$
  - **Identically distributed**: data points (including their labels) are drawn from the same probability distribution:  
$$P(x \geq A) = P(x \geq B), \text{ for all } x$$
- In practice, especially independence can be broken in many ways
- Example: weather prediction: one data point per day
  - ★ Data points from consecutive days are independent.
  - ★ Data points from consecutive days are dependent.

# Validation for the Non-i.i.d. Setting

- We want to obtain an **unbiased estimate** of the performance that we would achieve in practice (on a hold-out private test set)

# Validation for the Non-i.i.d. Setting

- We want to obtain an **unbiased estimate** of the performance that we would achieve in practice (on a hold-out private test set)
- Example: weather forecast
  - To predict tomorrow's weather, we cannot use data from the future
  - Standard option: when predicting for validation data point at time  $t$ , use training set up to  $t - 1$

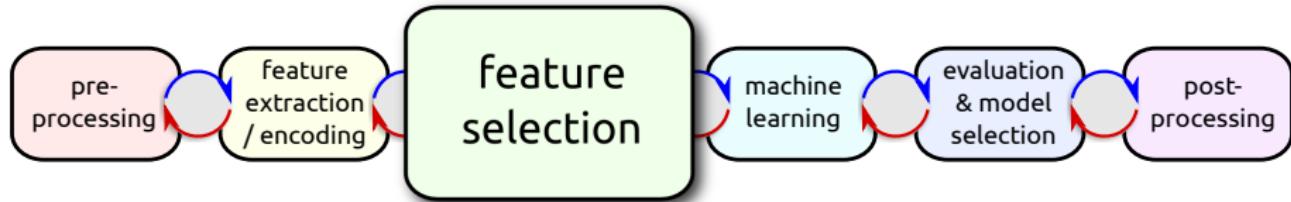
# Validation for the Non-i.i.d. Setting

- We want to obtain an **unbiased estimate** of the performance that we would achieve in practice (on a hold-out private test set)
- Example: weather forecast
  - To predict tomorrow's weather, we cannot use data from the future
  - Standard option: when predicting for validation data point at time  $t$ , use training set up to  $t - 1$
- Example: 50 data points measured for each of 100 patients
- What would the right 10-fold cross-validation protocol be to get an unbiased estimate of how well we'll predict on a **new patient**?
  - ★ Split data randomly into 10 folds of 500 data points each
  - ★ In each fold: use 5 data points of each of the 100 patients
  - ★ In each fold: use all 50 data points of 10 patients each

# Lecture Overview

- 1 Validation (cont'd)
- 2 Regularization
- 3 General Considerations
- 4 Model Selection and Feature Selection
- 5 Wrapup

# Feature Selection



- Feature selection: pick a subset of features that performs best
- Exactly the same problem of generalization as for model selection
- Special mechanisms exist to evaluate feature importance, etc
  - Here a quick preview

# Feature Selection: Forward Selection

## Forward Selection

- Build up your feature set step by step
- Iterate:
  - ① Evaluate the predictor performance by adding each of the unused features
  - ② Add the feature with the highest performance improvements

# Feature Selection: Forward Selection

## Forward Selection

- Build up your feature set step by step
- Iterate:
  - ① Evaluate the predictor performance by adding each of the unused features
  - ② Add the feature with the highest performance improvements
- ↝ small feature set, but dependencies between features are maybe missed

# Feature Selection: Forward Selection

## Forward Selection

- Build up your feature set step by step
  - Iterate:
    - ① Evaluate the predictor performance by adding each of the unused features
    - ② Add the feature with the highest performance improvements
- ↝ small feature set, but dependencies between features are maybe missed

Example:

Iteration	selected	$f_1$	$f_2$	$f_3$	$f_4$
1st	{}	20.0	17.7	<b>30.2</b>	20.1

# Feature Selection: Forward Selection

## Forward Selection

- Build up your feature set step by step
- Iterate:
  - ① Evaluate the predictor performance by adding each of the unused features
  - ② Add the feature with the highest performance improvements
- ↝ small feature set, but dependencies between features are maybe missed

Example:

Iteration	selected	$f_1$	$f_2$	$f_3$	$f_4$
1st	{}	20.0	17.7	<b>30.2</b>	20.1
2nd	{ $f_3$ }	5.1	<b>8.2</b>	—	4.9

# Feature Selection: Forward Selection

## Forward Selection

- Build up your feature set step by step
  - Iterate:
    - ① Evaluate the predictor performance by adding each of the unused features
    - ② Add the feature with the highest performance improvements
- ↝ small feature set, but dependencies between features are maybe missed

Example:

Iteration	selected	$f_1$	$f_2$	$f_3$	$f_4$
1st	{}	20.0	17.7	<b>30.2</b>	20.1
2nd	{ $f_3$ }	5.1	<b>8.2</b>	—	4.9
3rd	{ $f_3, f_2$ }	<b>0.8</b>	—	—	0.5

# Feature Selection: Forward Selection

## Forward Selection

- Build up your feature set step by step
  - Iterate:
    - ① Evaluate the predictor performance by adding each of the unused features
    - ② Add the feature with the highest performance improvements
- ↝ small feature set, but dependencies between features are maybe missed

Example:

Iteration	selected	$f_1$	$f_2$	$f_3$	$f_4$
1st	{}	20.0	17.7	<b>30.2</b>	20.1
2nd	{ $f_3$ }	5.1	<b>8.2</b>	–	4.9
3rd	{ $f_3, f_2$ }	<b>0.8</b>	–	–	0.5
4th	{ $f_3, f_2, f_1$ }	–	–	–	–0.2

# Feature Selection: Forward Selection

## Forward Selection

- Build up your feature set step by step
  - Iterate:
    - ① Evaluate the predictor performance by adding each of the unused features
    - ② Add the feature with the highest performance improvements
- ~~ small feature set, but dependencies between features are maybe missed

Example:

Iteration	selected	$f_1$	$f_2$	$f_3$	$f_4$
1st	{}	20.0	17.7	<b>30.2</b>	20.1
2nd	{ $f_3$ }	5.1	<b>8.2</b>	—	4.9
3rd	{ $f_3, f_2$ }	<b>0.8</b>	—	—	0.5
4th	{ $f_3, f_2, f_1$ }	—	—	—	-0.2

~~ we could also consider to stop after the second iteration, if the improvement by adding  $f_1$  is considered too small.

# Feature Selection: Backward Elimination

## Backward Elimination

- Reduce your feature set step by step
  - Iterate:
    - ① Evaluate the predictor performance by removing each of the considered features
    - ② Remove the feature with the smallest performance loss
- ~~ "larger" feature set, but dependencies between features are preserved

# Feature Selection: Backward Elimination

## Backward Elimination

- Reduce your feature set step by step
  - Iterate:
    - ① Evaluate the predictor performance by removing each of the considered features
    - ② Remove the feature with the smallest performance loss
- ↝ "larger" feature set, but dependencies between features are preserved

Example:

Iteration	selected	$f_1$	$f_2$	$f_3$	$f_4$
1st	$\{f_1, f_2, f_3, f_4\}$	0.8	0.0	-10.0	0.2

# Feature Selection: Backward Elimination

## Backward Elimination

- Reduce your feature set step by step
  - Iterate:
    - ① Evaluate the predictor performance by removing each of the considered features
    - ② Remove the feature with the smallest performance loss
- ↝ "larger" feature set, but dependencies between features are preserved

Example:

Iteration	selected	$f_1$	$f_2$	$f_3$	$f_4$
1st	$\{f_1, f_2, f_3, f_4\}$	<b>0.8</b>	0.0	-10.0	0.2
2nd	$\{f_2, f_3, f_4\}$	-	<b>-0.5</b>	-12.3	-2.3

# Feature Selection: Backward Elimination

## Backward Elimination

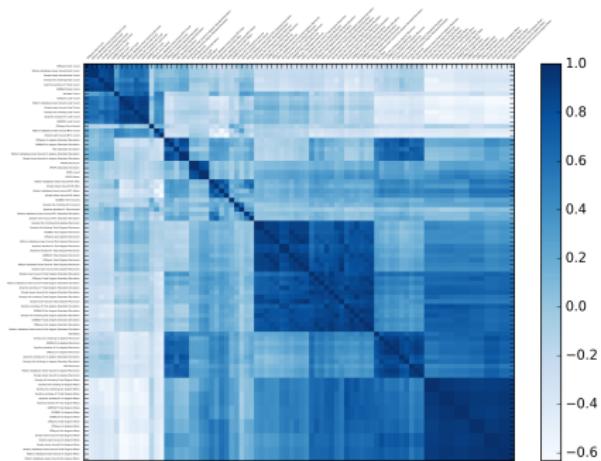
- Reduce your feature set step by step
  - Iterate:
    - ① Evaluate the predictor performance by removing each of the considered features
    - ② Remove the feature with the smallest performance loss
- ~~ "larger" feature set, but dependencies between features are preserved

Example:

Iteration	selected	$f_1$	$f_2$	$f_3$	$f_4$
1st	$\{f_1, f_2, f_3, f_4\}$	<b>0.8</b>	0.0	-10.0	0.2
2nd	$\{f_2, f_3, f_4\}$	-	<b>-0.5</b>	-12.3	-2.3
3rd	$\{f_3, f_4\}$	-	-	-18.3	-8.5

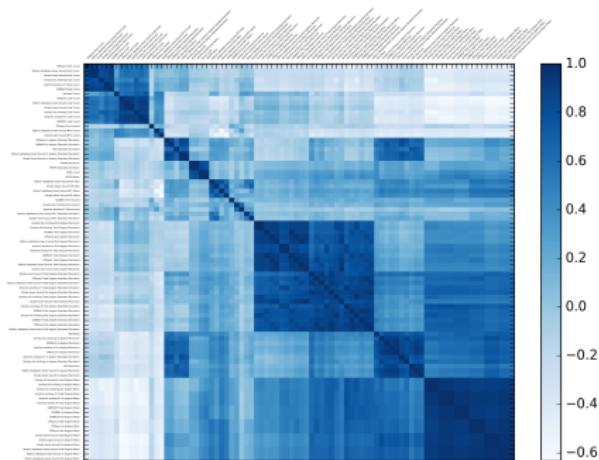
~~ Forward selection and backward elimination often do not recover the same set of features.

# Feature Selection: Correlation Analysis



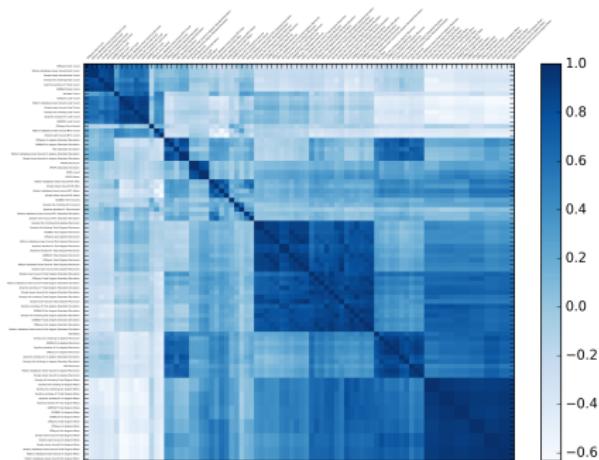
- highly correlated features often hurt the training process more than they help

# Feature Selection: Correlation Analysis



- highly correlated features often hurt the training process more than they help
- you should consider to remove correlated features all but one
  - a PCA is implicitly doing something similar

## Feature Selection: Correlation Analysis



- highly correlated features often hurt the training process more than they help
  - you should consider to remove correlated features all but one
    - a PCA is implicitly doing something similar
  - Pearson correlation coefficient

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X\sigma_Y}$$

# Lecture Overview

- 1 Validation (cont'd)
- 2 Regularization
- 3 General Considerations
- 4 Model Selection and Feature Selection
- 5 Wrapup

# Summary by learning goals

Having heard this lecture, you can now ...

- Explain how to check performance using **cross-validation**
- Identify **over- and underfitting** based on learning curves
- Explain different **regularization approaches**
- **Select features** (using basic approaches)
- Explain the **i.i.d. assumption** and where it can break down
- Handle **imbalanced data**
- Choose the right **metric of success** for a new application
- Explain the **appropriate cross-validation splits** for several settings

# Lecture 9: Algorithm Independent Principles - III

## Hyperparameter Optimization, AutoML

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# Lecture Overview

- 1 Hyperparameter Optimization
- 2 Automated Machine Learning (AutoML)
- 3 Grey-Box AutoML

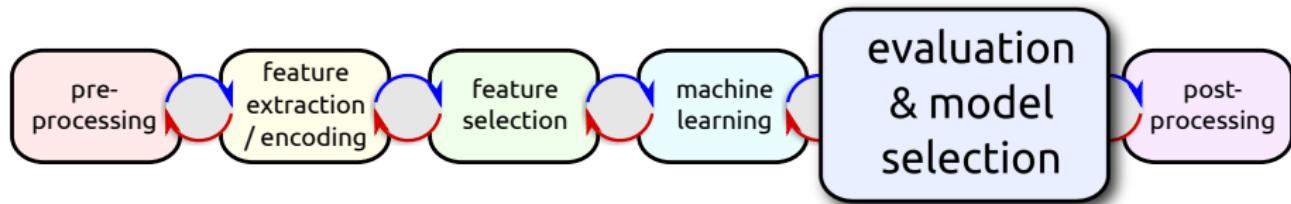
# Lecture Overview

1 Hyperparameter Optimization

2 Automated Machine Learning (AutoML)

3 Grey-Box AutoML

# Hyperparameter Optimization in the ML Design Cycle



Part of model selection: how to set free algorithm hyperparameters?

- Hyperparameter optimization is essentially the same problem as model selection, except:
- Many more possible settings to choose from (infinite/exponential)

# Parameters vs. Hyperparameters

- Most machine learning algorithms internally optimize parameters
  - E.g., weights in linear regression
  - E.g., split points and leaf predictions in decision trees
  - E.g., deep learning: millions of network weights
- Standard ML approach:  $\text{min. training loss} + \lambda \times \text{regularization loss}$ 
  - Using standard gradient-based optimizers
- Hyperparameters: decisions left to the algorithm designer
  - How to set  $\lambda$ ?
  - How complex a model to use?
  - How many layers/which structure of deep networks to use?
  - How to set the options of the gradient-based optimizer?

# SVM Documentation



Home Installation Documentation Examples

Google Custom Search



Previous [sklearn.svm.SVC](#) Next [sklearn.svm.SVR](#) Up [API Reference](#)

scikit-learn v0.21.1  
Other versions

Please cite us if you use  
the software.

[sklearn.svm.SVC](#)  
Examples using  
[sklearn.svm.SVC](#)

## sklearn.svm.SVC

```
class sklearn.svm.SVC (C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True,  
probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1,  
decision_function_shape='ovr', random_state=None)
```

[source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using

`sklearn.linear_model.LinearSVC` or `sklearn.linear_model.SGDClassifier` instead, possibly after a `sklearn.kernel_approximation.Nystroem` transformer.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: [Kernel functions](#).

Read more in the [User Guide](#).

**Parameters:** `C : float, optional (default=1.0)`

Penalty parameter C of the error term.

`kernel : string, optional (default='rbf')`

Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n\_samples, n\_samples).

`degree : int, optional (default=3)`

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

`gamma : float, optional (default='auto')`

# Hyperparameter Optimization Method 1: Random Search

- Select configurations uniformly at random (completely uninformed)
- Global search, won't get stuck in a local region
- Parallelizes nicely and is at least better than grid search:

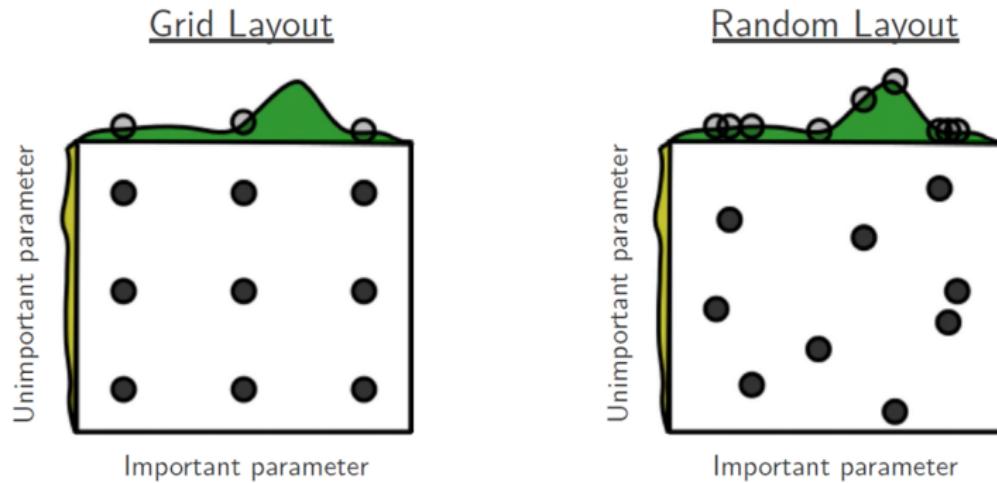


Image source: [Bergstra et al, JMLR 2012]

# Hyperparameter Optimization Method 2: Local Search

(also sometimes jokingly called “Graduate Student Descent”)

---

Start with some configuration  $\theta$

**repeat**

  | Modify a single hyperparameter

  | **if** *results on benchmark set improve* **then**

    | keep new configuration

**until** *no more improvement possible (or “good enough”)*

---

~~ Manually-executed **first-improvement local search**

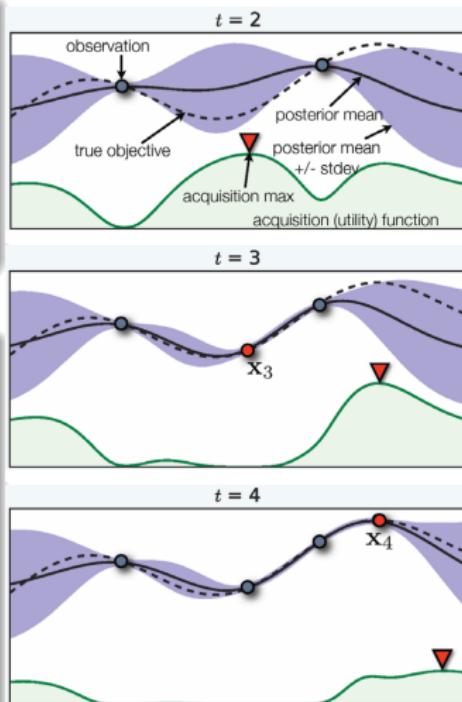
# Hyperparameter Opt. Method 3: Bayesian Optimization

## General approach

- Fit a probabilistic model to the collected function samples  $\langle \theta, f(\theta) \rangle$
- Use the model to guide optimization, trading off exploration vs exploitation

Popular approach in the statistics literature since [Mockus, 1978]

- Efficient in # function evaluations
- Works when objective is nonconvex, noisy, has unknown derivatives, etc
- Recent convergence results  
[Srinivas et al, 2010; Bull 2011; de Freitas et al, 2012; Kawaguchi et al, 2015]



# Bayesian Optimization Algorithm

---

**Algorithm 1:** Bayesian Optimization (BO)

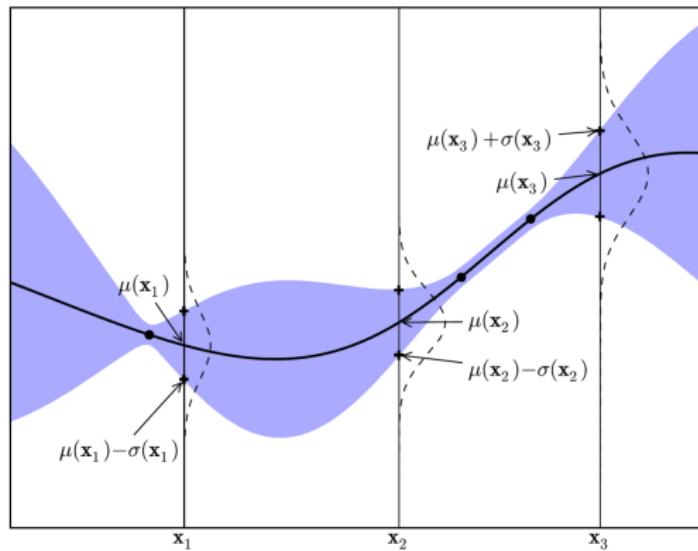
---

**Input** : Search Space  $\mathcal{X}$ , black box function  $f$ , acquisition function  $\alpha$ , maximal number of function evaluations  $m$

- 1  $\mathcal{D}_0 \leftarrow \text{initial\_design}(\mathcal{X})$ ;
  - 2 **for**  $n = 1, 2, \dots m - |\mathcal{D}_0|$  **do**
  - 3      $\hat{m} : \theta \mapsto y \leftarrow \text{fit predictive model on } \mathcal{D}_{n-1}$ ;
  - 4     select  $x_n$  by optimizing  $x_n \in \arg \max_{x \in \mathcal{X}} \alpha(x; \mathcal{D}_{n-1}, \hat{m})$ ;
  - 5     Query  $y_n := f(x_n)$ ;
  - 6     Add observation to data  $\mathcal{D}_n := \mathcal{D}_{n-1} \cup \{\langle x_n, y_n \rangle\}$ ;
  - 7 **return** Best  $x$  according to  $\mathcal{D}_m$
-

# Acquisition Function: Upper Confidence Bound

- $\text{UCB}(\theta) = \mu(\theta) + \kappa_t \sigma(\theta)$ , with exploration parameter  $\kappa$ 
  - for maximizing  $f$ !



- Which point would we pick next with UCB and  $\kappa = 1$ ?
- $\kappa_t$  increases over time

# Range Transformations of Hyperparameters

- Several hyperparameters naturally lay on a **logarithmic scale**
  - E.g., learning rate between  $10^{-5}$  and  $10^0$
- Transform these to a **linear scale**
  - E.g., instead work with  $\log_{10}(\text{lr})$ , with values between -5 and 0
- Good rule of thumb: transform the space to the one you would want to sample from uniformly
  - E.g., uniform sampling from  $[10^{-5}, 10^0]$  would have 90% samples greater than  $10^{-1}$

# Conditional Hyperparameters

- Some hyperparameters  $h$  are only active if other hyperparameters  $p$  take certain values
  - E.g., weight initialization for layer  $k$  is only active if the hyperparameter number of layers is at least  $k$
- We call these hyperparameters **conditional** with **parents**  $p$
- You can always ignore such conditionality, but exploiting it can dramatically simplify the problem

# Lecture Overview

- 1 Hyperparameter Optimization
- 2 Automated Machine Learning (AutoML)
- 3 Grey-Box AutoML

# Automated Machine Learning

Machine Learning is very successful in many applications.

- But it still requires human machine learning experts to
  - Preprocess the data
  - Select / engineer features
  - Select a model family
  - Optimize hyperparameters
  - Construct ensembles
  - ...
- **AutoML**: taking the human expert out of the loop
- **Deep learning** helps to automatically learn features
  - But it is even more sensitive to hyperparameters

# Automated Machine Learning

The AutoML approach introduced by Auto-WEKA [Thornton et al, 2013]

- Expose the choices in a machine learning framework
  - Algorithms, hyperparameters, preprocessors, ...
  - Highly conditional hyperparameter space
  - Combined algorithm selection and hyper-parameter optimization  
→ defined in detail on the next slide
- Optimize CV performance using Bayesian optimization
- ↝ Obtain a true push-button solution for machine learning

Extended in Auto-sklearn [Feurer et al, 2015]

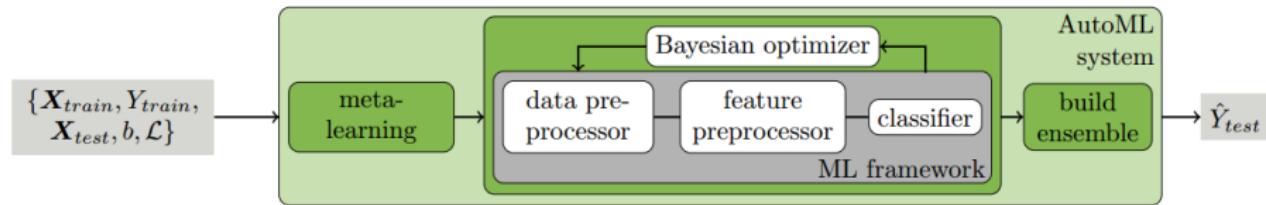
# CASH Problem [Thornton et al. 2013]

Let

- $\mathcal{A} = A^{(1)}, \dots, A^{(J)}$  be a set of algorithms,
- the hyper-parameters of each algorithm  $A^{(j)}$  have domain  $\Lambda^{(j)}$ ,
- $D_{train} = \{(x_1, y_1), \dots, (x_n, y_n)\}$  be a training set which is split into  $K$  cross-validation folds  $\{D_{valid}^{(1)}, \dots, D_{valid}^{(K)}\}$  and  $D_{train}^{(i)} = D_{train} - D_{valid}^{(i)}$  for  $i = 1, \dots, K$ ,
- $\mathcal{L}(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$  denote the loss of  $A_\lambda^{(j)}$  trained on  $D_{train}^{(i)}$  and evaluated on  $D_{valid}^{(i)}$ .

The *CASH* problem is to find the joint algorithm and hyper-parameter setting that minimizes this loss across the  $K$  folds:

$$A^*, \lambda^* \in \arg \min_{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}} \frac{1}{K} \sum_{i=1}^K \mathcal{L}(A_\lambda^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$$



- Winner of ChaLearn Automatic Machine Learning Challenge
  - Performed better than 150 teams of human experts
  - Searches over  $\approx 15$  classifiers & preprocessors (110 hyperparameters)
- Trivial to use even for novices in machine learning:

```
import autosklearn.classification as cls
automl = cls.AutoSklearnClassifier()
automl.fit(X_train, y_train)
y_hat = automl.predict(X_test)
```

Available online: <https://github.com/automl/auto-sklearn>

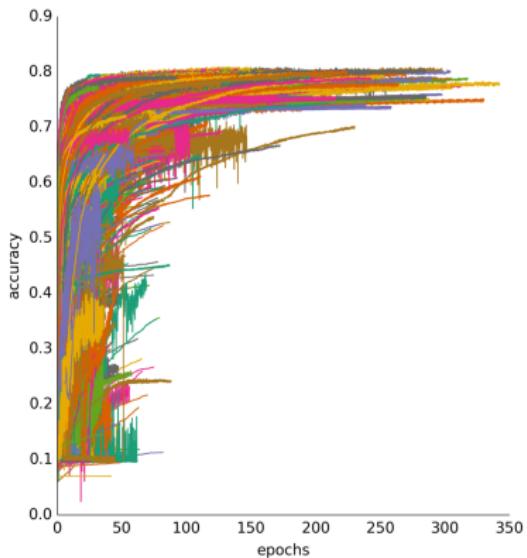
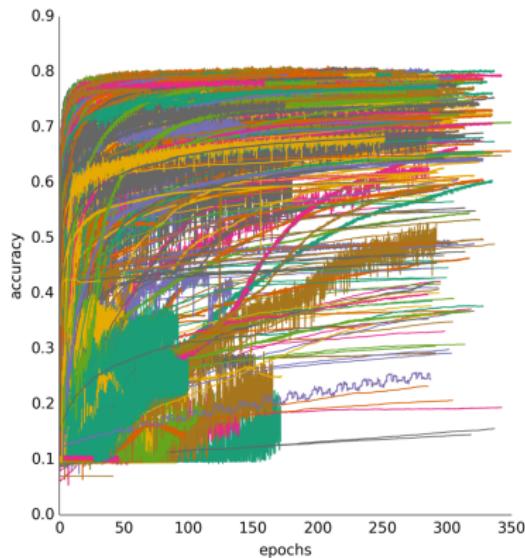
# Lecture Overview

- ① Hyperparameter Optimization
- ② Automated Machine Learning (AutoML)
- ③ Grey-Box AutoML

# Speeding up Hyperparameter Optimization Methods

- If we use  $k$ -fold cross-validation
  - We can reject poor hyperparameter settings after few folds
- If we use iterative ML algorithms
  - We can stop poor runs early
- If runs on smaller datasets are faster (almost always)
  - We can quickly select decent models based on data subsets

# Learning Curve Tuning

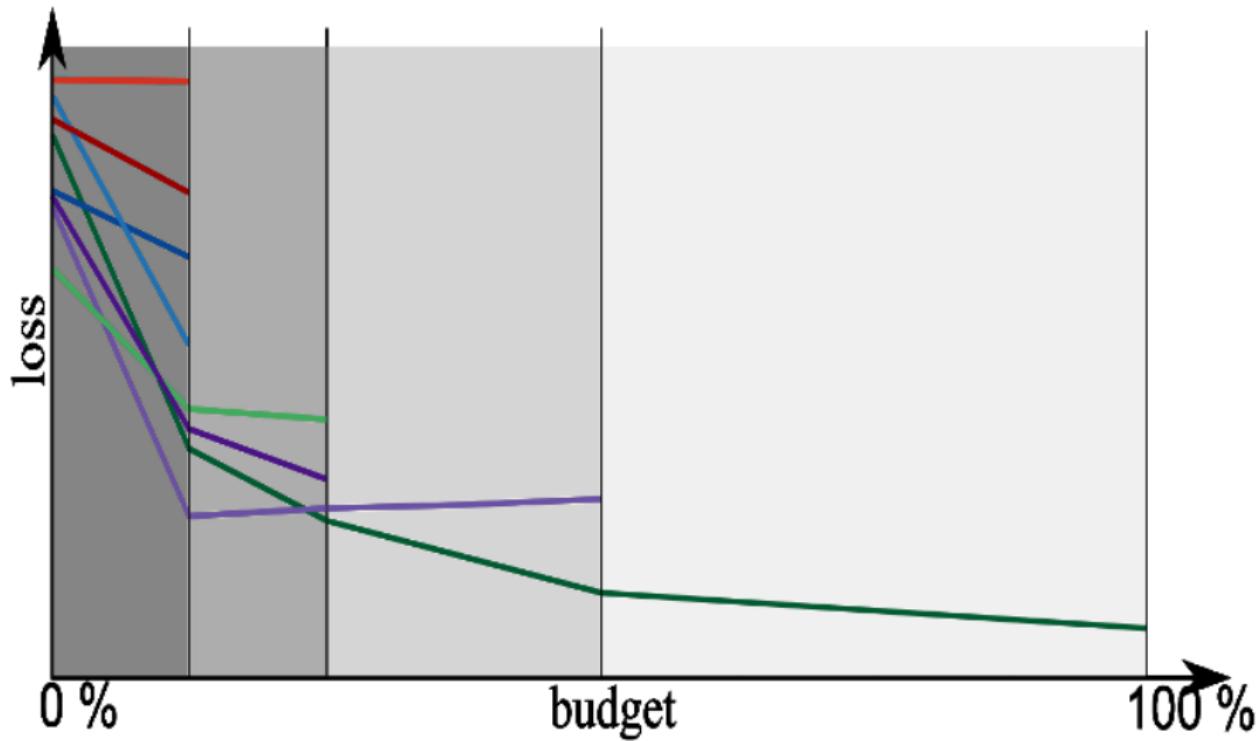


↝ We only want to train the best learning curves until the end

## Successive Halving

- Ideas:
  - Invest only resources in promising configurations
    - ⇝ aggressive dropping of poor configurations
  - Model-free — (more or less assumption free)
- Algorithm Outline:
  - Input:  $n$  (randomly sampled) configurations and budget  $B$
  - ① Run remaining configurations with some resource allocation (depending on  $B$ )
  - ② Sort configurations by cost (e.g., validation loss)
  - ③ Throw away lower half of configurations
  - ④ Repeat
- Resource allocation can correspond to
  - partial learning curves
  - subset of training data

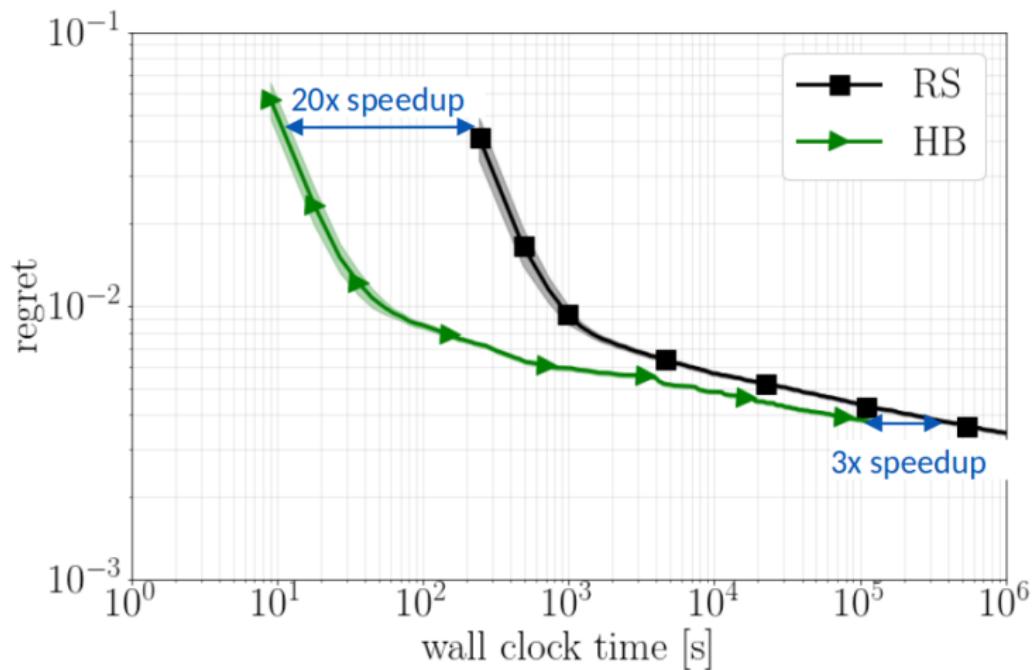
## Successive Halving: Illustration



## Hyperband

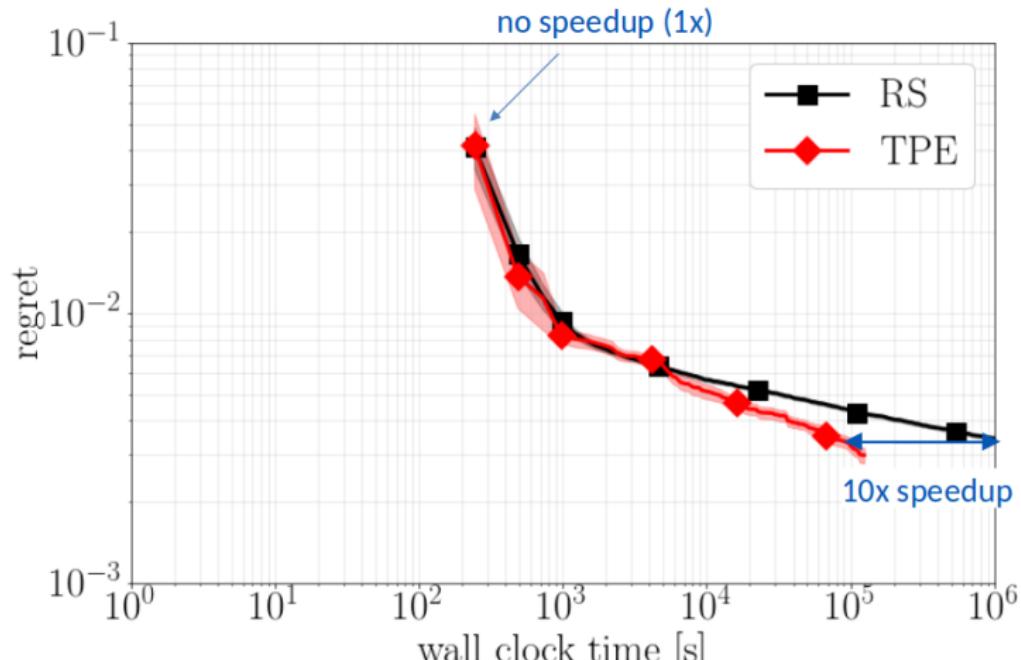
- Issue of successive halving (for a fixed  $B$ ):  
Do you want to run many configurations with aggressive rejection?  
Or: Do you want to run few configurations with non-aggressive rejection?
- Ideas:
  - Add an outer loop to try different trade-offs between #configurations and budget
  - Add further parameter: proportion of configurations discarded in each round of successive halving
- Starts with many configurations that gets aggressively rejected
- In later iterations, few configurations with more budget each
- Returns: configuration with the smallest intermediate loss seen so far.

# Random Search vs. Hyperband



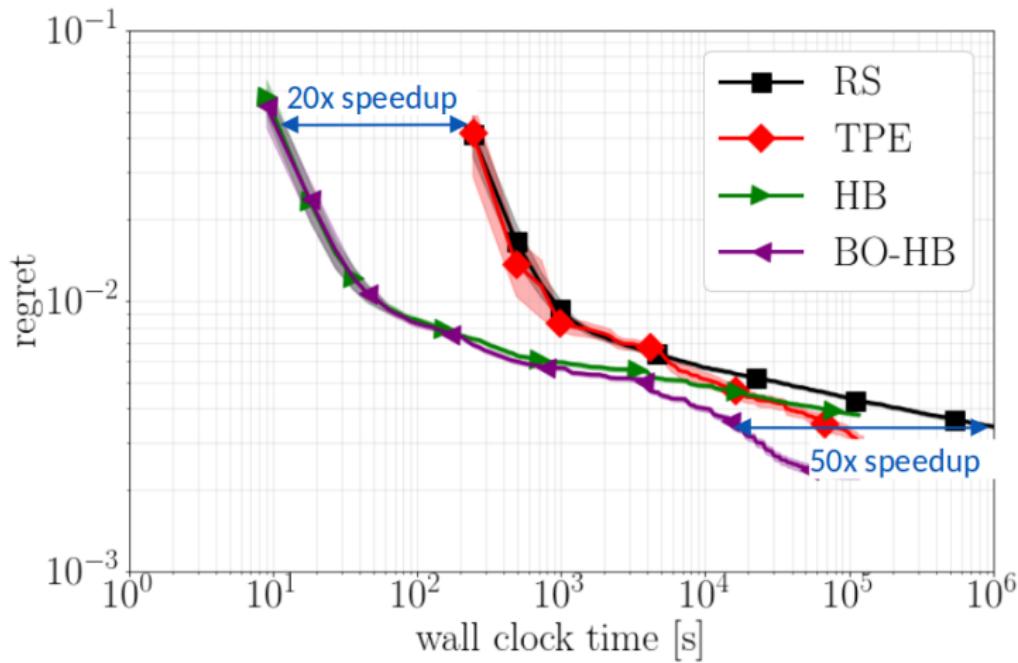
~~ Hyperband performs well early on

# Random Search vs. Bayesian Optimization



~~ Bayesian optimization performs well later on

# Random Search vs. Bayesian Optimization vs. Hyperband



~~ A combination of Bayesian Optimization and Hyperband performs well overall

# BOHB: a Robust & Efficient Framework for Hyperparameter Optimization

- BOHB: Bayesian Optimization & HyperBand
  - Bayesian optimization for selecting configurations
  - Hyperband for implementing the speedup techniques, using the concept of a budget (size of data subsets, #epochs, #CV folds, etc)
  - Currently one of the best available approach for robust & efficient hyperparameter optimization
- You can use this for optimizing your own hyperparameters  
<https://github.com/automl/HpBandSter>

# Summary by learning goals

Having heard this lecture, you can now ...

- Describe the difference between **parameters** and **hyperparameters**
- Give some **examples of hyperparameters** of various algorithms
- Describe the ideas behind **random search** and **Bayesian optimization**
- Explain methods to speedup hyperparameter optimization by using **grey-box approaches**

# Further Reading

- Random search: [Bergstra et al, 2012]
- Bayesian optimization: [Mockus, 1978]; [Brochu et al, 2010]
- BOHB: [Falkner et al, 2019]

# Lecture 10: Kernel Methods (Part I)

## Top-Down Introduction to the Kernel Trick and the Support Vector Machine

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# Lecture Overview

- 1 Mechanical Understanding of Support
- 2 Introduction Kernel Algorithms
- 3 Classification Revisited
- 4 Dot Product (and what you can do with it)
- 5 From Input Space to Feature Space
- 6 Statistical Learning Theory
- 7 Large Margin Principle: Optimal Hyperplane Classifier
- 8 Support Vector Machine (SVM)

# Lecture Overview

- 1 Mechanical Understanding of Support
- 2 Introduction Kernel Algorithms
- 3 Classification Revisited
- 4 Dot Product (and what you can do with it)
- 5 From Input Space to Feature Space
- 6 Statistical Learning Theory
- 7 Large Margin Principle: Optimal Hyperplane Classifier
- 8 Support Vector Machine (SVM)

(use the stick)

# Lecture Overview

- 1 Mechanical Understanding of Support
- 2 Introduction Kernel Algorithms
- 3 Classification Revisited
- 4 Dot Product (and what you can do with it)
- 5 From Input Space to Feature Space
- 6 Statistical Learning Theory
- 7 Large Margin Principle: Optimal Hyperplane Classifier
- 8 Support Vector Machine (SVM)

# Kernel Algorithms

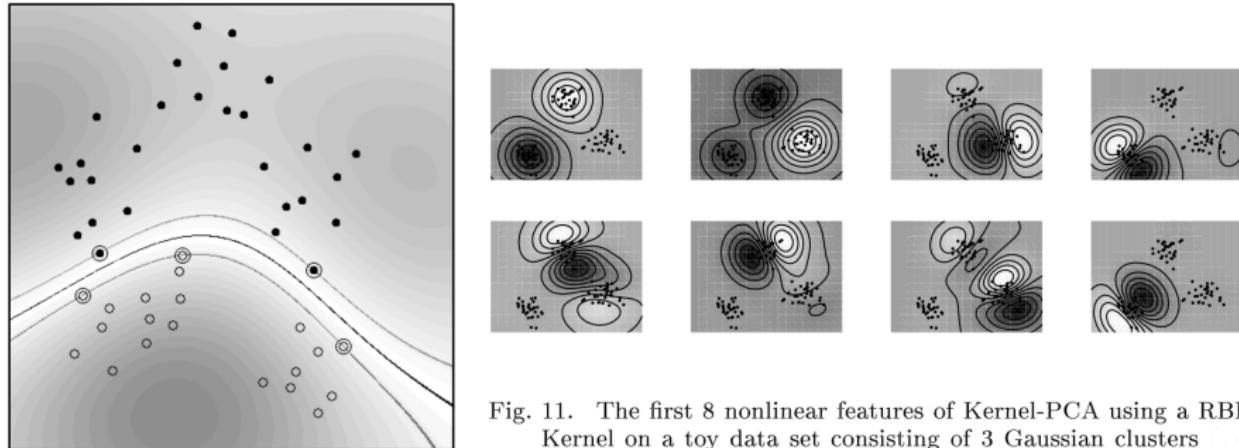


Fig. 11. The first 8 nonlinear features of Kernel-PCA using a RBF Kernel on a toy data set consisting of 3 Gaussian clusters

Kernel algorithms open up new possibilities for non-linear algorithms.  
Examples:

- support vector machine (SVM)
- kernel principal component analysis (kPCA)
- Gaussian processes

# Kernel Algorithms

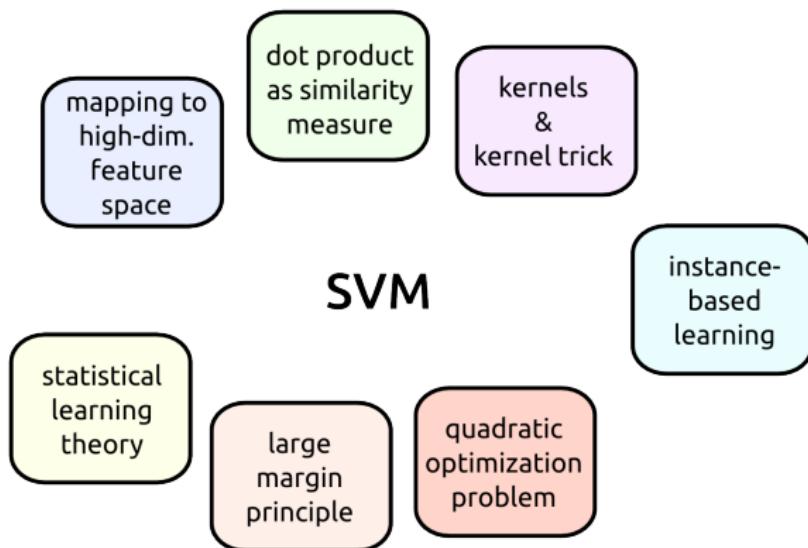
Kernel methods have rocked the world of machine learning for many application areas, e.g.

- Optical character recognition (OCR), handwritten character recognition
- Image detection
- Bioinformatics (DNA data)
- Biomedical imaging data
- ...

The screenshot shows a news article from BBC News. The headline reads: "Computers are able to diagnose Alzheimer's disease faster and more accurately than experts". Below the headline, it says "Medical Research News" and "Published: Sunday, 24-Feb-2008". There are links for "Printer Friendly" and "Email to a Friend". The main text of the article states: "Computers are able to diagnose Alzheimer's disease faster and more accurately than experts, according to research published in the journal Brain." It continues: "The findings may help ensure that patients are diagnosed earlier, increasing treatment options. According to the Alzheimer's Research Trust, there are over 700,000 people currently living in the UK with dementia, of which Alzheimer's disease, a neurodegenerative disease, is the most common form. Alzheimer's is caused by the build up in the brain of plaques and neurofibrillary tangles (tangles of brain tissue filaments), leading the brain to atrophy. Definitive diagnosis is usually only..." The page has a red and black color scheme with some blurred text at the top.

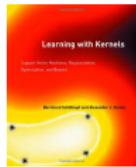
# Concepts Underlying Kernel Methods

Today's lecture introduces SVM — however, a number of concepts need to be introduced, in order to get there...



# Literature / Materials

- Book by Bernhard Schölkopf and Alexander Smola:  
Learning with Kernels (MIT Press)



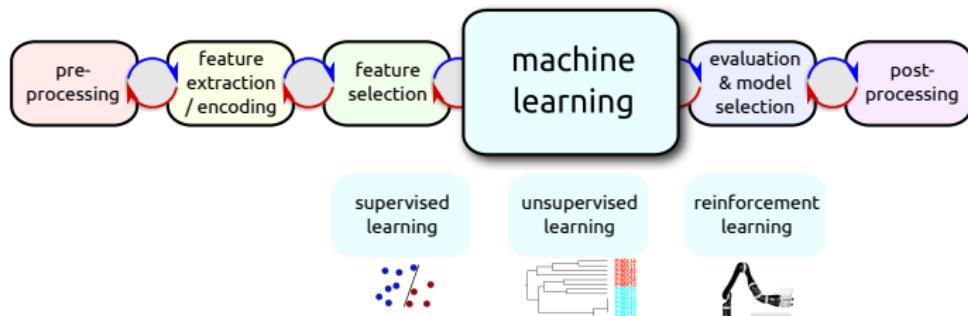
(mostly Sec. 1)

- Tutorial by Hearst et al. "Support vector machines", IEEE Intelligent Systems, 1989.
- Slides by Bernhard Schölkopf: "Introduction to Machine Learning", held at Machine Learning Summer School 2011 in Bordeaux
- Slides by Statnikov et al. "A Gentle Introduction to Support Vector Machines in Biomedicine", held at AMIA 2009.
- Müller et al.: "An Introduction to Kernel-Based Learning Algorithms", IEEE Trans. Neural Networks 2001
- [www.kernel-machines.org](http://www.kernel-machines.org)

# Lecture Overview

- 1 Mechanical Understanding of Support
- 2 Introduction Kernel Algorithms
- 3 Classification Revisited
- 4 Dot Product (and what you can do with it)
- 5 From Input Space to Feature Space
- 6 Statistical Learning Theory
- 7 Large Margin Principle: Optimal Hyperplane Classifier
- 8 Support Vector Machine (SVM)

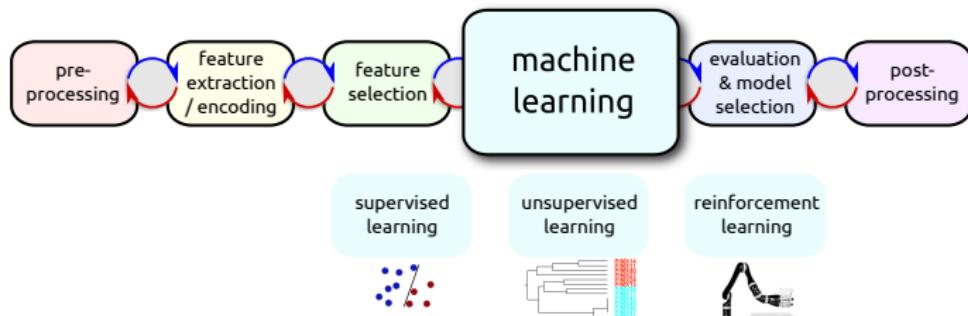
# Support Vector Machine within the ML Design Cycle



Context of **supervised classification**:

- Use past experience to predict the future
- Typical training data are vectors:  
m-many N-dimensional patterns  $\mathbf{x}_i$  and class labels  $y_i$ :  
 $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m) \in \mathbb{R}^N \times \{\pm 1\}$

# Support Vector Machine within the ML Design Cycle



Context of **supervised classification**:

- Use past experience to predict the future
- Typical training data are vectors:  
m-many N-dimensional patterns  $\mathbf{x}_i$  and class labels  $y_i$ :  
 $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m) \in \mathbb{R}^N \times \{\pm 1\}$
- However, patterns could be from an **arbitrary set  $\mathcal{X}$** :  
 $(x_1, y_1), \dots, (x_m, y_m) \in \mathcal{X} \times \{\pm 1\}$





## Need for Similarity Measure

Learning task in vector spaces:

- estimate a function (model)  $f : \mathbb{R}^N \rightarrow \{\pm 1\}$ , such that  $f$  will correctly classify new examples  $(\mathbf{x}, y)$ , i.e.  $f(\mathbf{x}) = y$



## Need for Similarity Measure

Learning task in vector spaces:

- estimate a function (model)  $f : \mathbb{R}^N \rightarrow \{\pm 1\}$ , such that  $f$  will correctly classify new examples  $(\mathbf{x}, y)$ , i.e.  $f(\mathbf{x}) = y$

If patterns  $x_i$  are from an arbitrary set  $\mathcal{X}$ , then we need some **similarity measure** in this space!



## Need for Similarity Measure

Learning task in vector spaces:

- estimate a function (model)  $f : \mathbb{R}^N \rightarrow \{\pm 1\}$ , such that  $f$  will correctly classify new examples  $(\mathbf{x}, y)$ , i.e.  $f(\mathbf{x}) = y$

If patterns  $x_i$  are from an arbitrary set  $\mathcal{X}$ , then we need some **similarity measure** in this space!

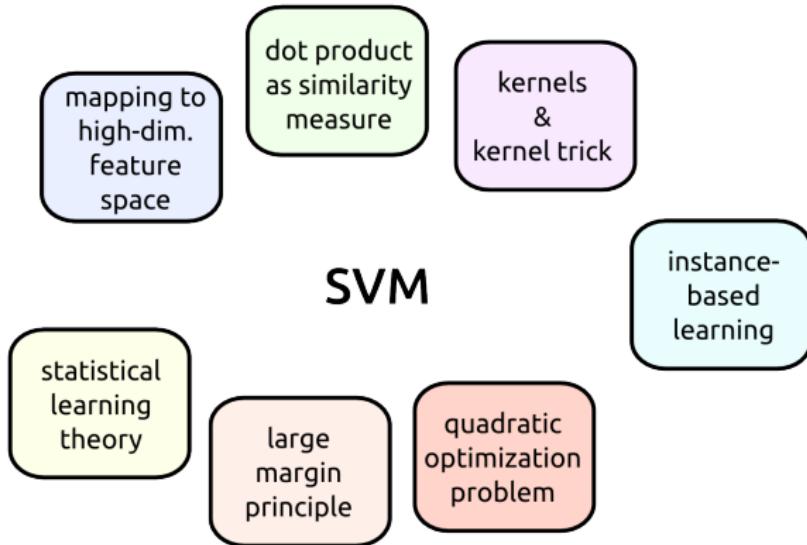
Convenient similarity measures should:

- ... compare two patterns  $x, x'$
- ... deliver a real number describing their similarity:  
 $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$   
 $(x, x') \rightarrow k(x, x')$
- ... be symmetric, i.e.  $k(x, x') = k(x', x)$  for all  $x, x' \in \mathcal{X}$
- This similarity function  $k$  is called a kernel.

# Lecture Overview

- ① Mechanical Understanding of Support
- ② Introduction Kernel Algorithms
- ③ Classification Revisited
- ④ Dot Product (and what you can do with it)
- ⑤ From Input Space to Feature Space
- ⑥ Statistical Learning Theory
- ⑦ Large Margin Principle: Optimal Hyperplane Classifier
- ⑧ Support Vector Machine (SVM)

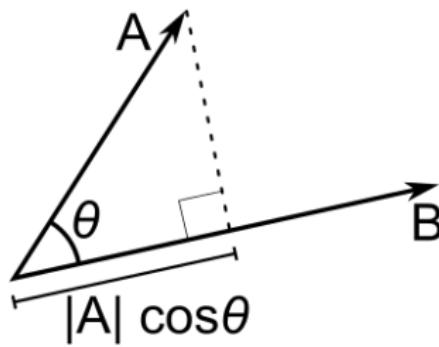
# Concept: Similarity Measure



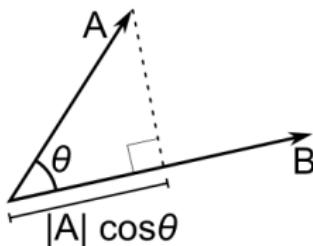
# Dot Product as a Convenient Similarity Measure

- Synonyms of dot product: inner product, scalar product.
- Canonical dot product is calculated between two vectors  $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^N$
- Definition of dot product:  $\langle \mathbf{x}, \mathbf{x}' \rangle := \sum_{i=1}^N [\mathbf{x}]_i [\mathbf{x}']_i$   
with  $[\mathbf{x}]_i$  being the  $i$ th entry / dimension of the vector  $\mathbf{x}$ .
- Remark: sometimes written as  $(\mathbf{x} \cdot \mathbf{x}')$

Geometrical interpretation:  $\langle \mathbf{A}, \mathbf{B} \rangle = \|\mathbf{A}\| \|\mathbf{B}\| \cos(\phi)$



# Dot Product as a Convenient Similarity Measure



Dot products are very handy to describe **similarity** in terms of

- angle
- projected length
- length / norm of vector  $\mathbf{x}$  as:  $\|\mathbf{x}\| = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$
- distance between two points (the length of the difference vector)

Thus any geometric construction can be carried out, which can be formulated in terms of angles, lengths and distances!

# How Far Do We Get with Angles, Lengths, Distances?

Can we define a simple classification algorithm just based on dot products?



Please vote: **yes** / **no**

# How Far Do We Get with Angles, Lengths, Distances?

Can we define a simple classification algorithm just based on dot products?



Please vote: **yes** / **no**

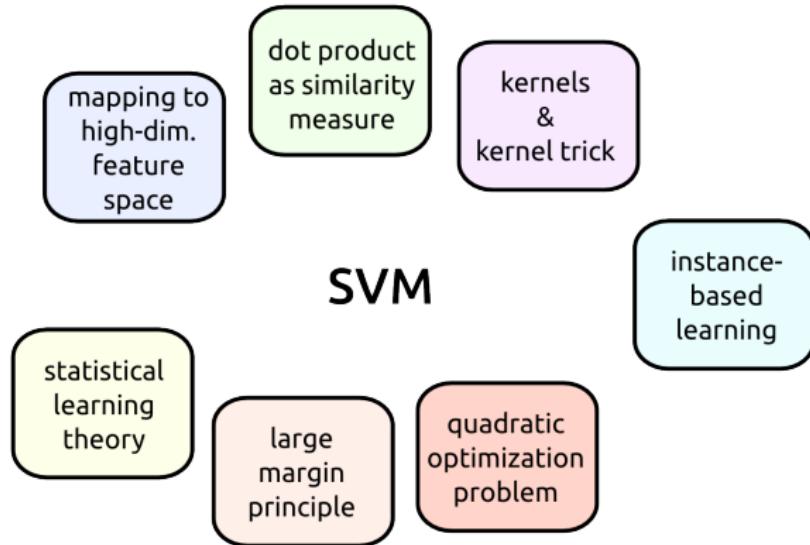
(For an example read "Learning with Kernels", Section 1.2)

- Decision function and the offset  $b$  can be expressed based on dot products of the training data points
- All computations necessary to obtain the class label for a novel test data point  $x$  involve only dot product operations between  $x$  and some training data points  $x_i$ .

# Lecture Overview

- 1 Mechanical Understanding of Support
- 2 Introduction Kernel Algorithms
- 3 Classification Revisited
- 4 Dot Product (and what you can do with it)
- 5 From Input Space to Feature Space
- 6 Statistical Learning Theory
- 7 Large Margin Principle: Optimal Hyperplane Classifier
- 8 Support Vector Machine (SVM)

# Concept: Mapping to High-Dimensional Feature Space



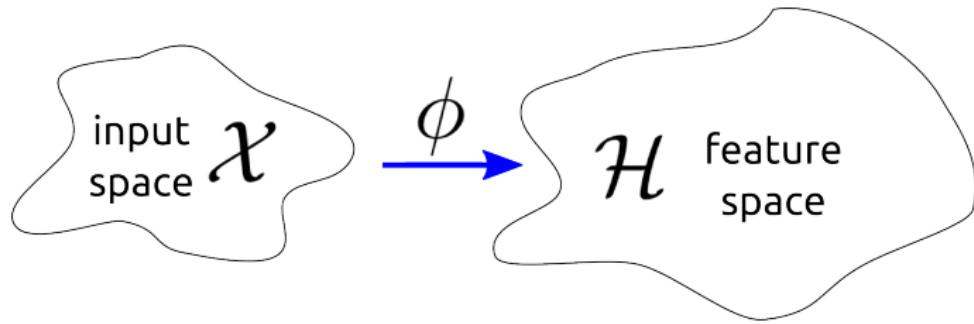
# (Non-Linear) Mapping to Feature Space

Please observe:

- The dot product as a similarity measure is OK if patterns  $\mathbf{x}_i$  are vectors.
- In general, data points  $x_i$  are from the set  $\mathcal{X}$  and may need to be mapped to some dot product space first!

Use mapping:

$$\phi : \mathcal{X} \rightarrow \mathcal{H}$$
$$x \rightarrow \mathbf{x} := \phi(x)$$



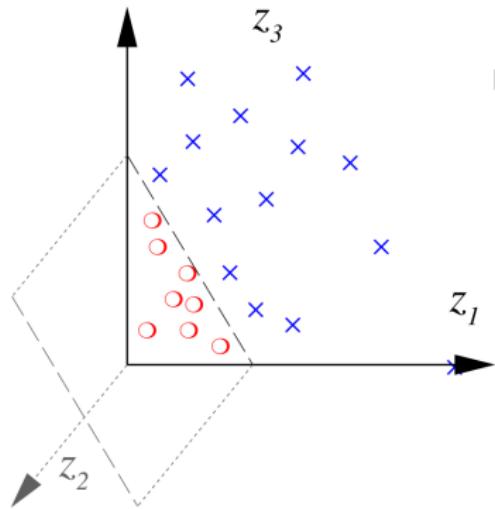
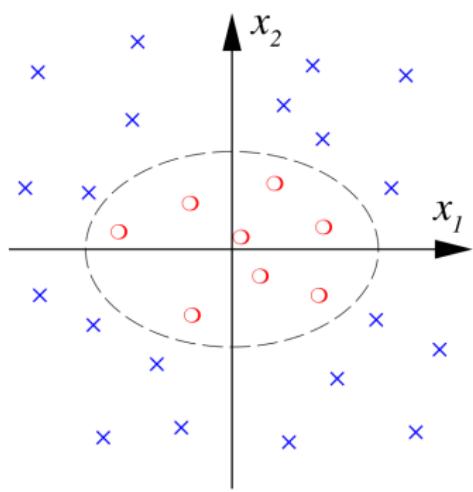
# (Non-Linear) Mapping to Feature Space

Who remembers this mapping?



$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

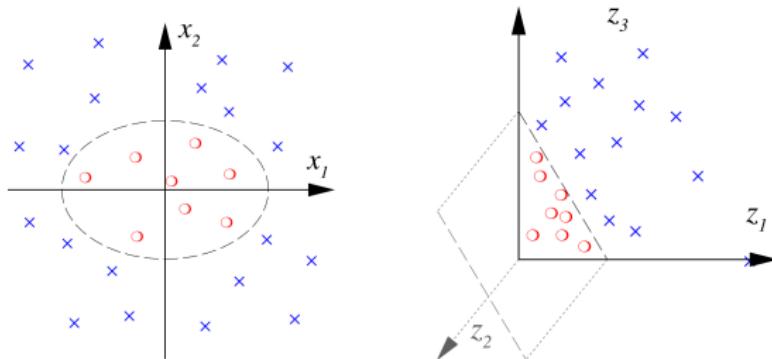
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$



# (Non-Linear) Mapping to Feature Space

Even if the input space is a dot product space already (patterns  $x_i$  are vectors), we may still want to apply a (non-linear) mapping into a feature space.

$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

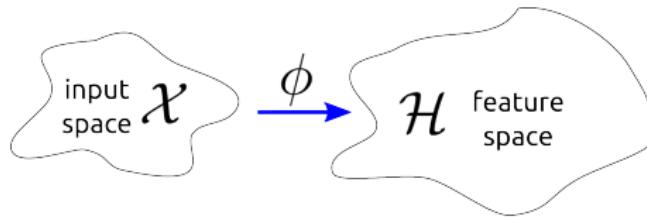


- Feature space  $\mathcal{H}$  may have higher dimensionality
- Separability of data points may be easier in  $\mathcal{H}$  compared to  $\mathcal{X}$

# (Non-Linear) Mapping to Feature Space

Advantages of applying the mapping:

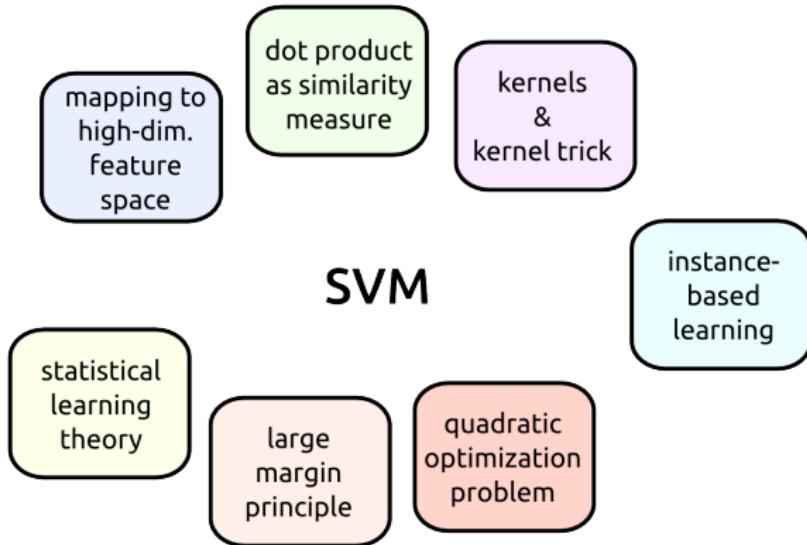
- Freedom to choose the mapping  $\phi$  opens a wide choice of similarity measures and learning algorithms!
- A suitable mapping  $\phi$  will improve the representation into one that is more suitable to solve the given problem.
- $\phi$  can define a similarity measure using the dot product in  $\mathcal{H}$ ,  
 $k(x, x') := \langle \mathbf{x}, \mathbf{x}' \rangle = \langle \phi(x), \phi(x') \rangle$
- It allows us to deal with patterns geometrically (we can apply linear algebra + analytic geometry)



# Lecture Overview

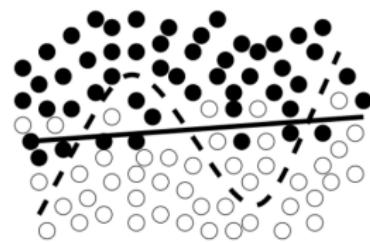
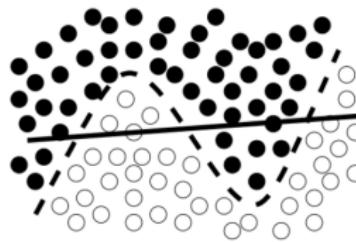
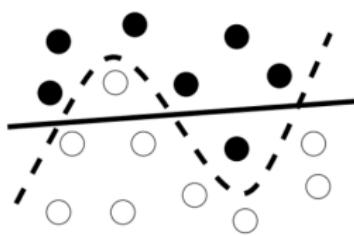
- 1 Mechanical Understanding of Support
- 2 Introduction Kernel Algorithms
- 3 Classification Revisited
- 4 Dot Product (and what you can do with it)
- 5 From Input Space to Feature Space
- 6 Statistical Learning Theory
- 7 Large Margin Principle: Optimal Hyperplane Classifier
- 8 Support Vector Machine (SVM)

# Concept: Statistical Learning Theory



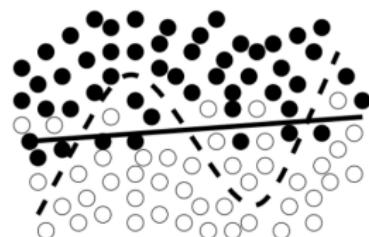
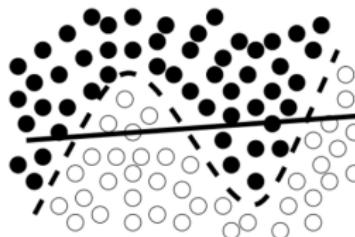
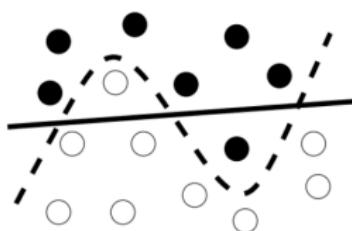
# Overfitting Dilemma

Which function (dashed or solid) fits the data on the left better?



# Overfitting Dilemma

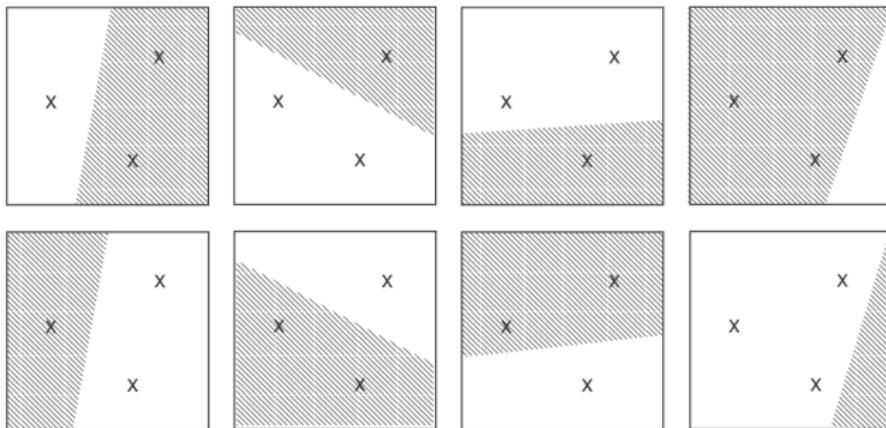
Which function (dashed or solid) fits the data on the left better?



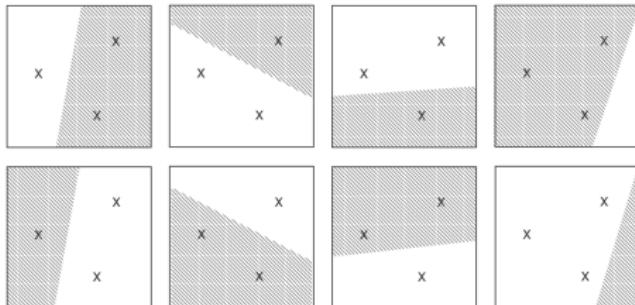
Please observe:

- In supervised learning, the choice of the **function class / hypothesis class** is relevant for good generalization of the trained model to novel data.
- Very powerful function classes (high "capacity") tend to overfit the training data.

# Estimating the Capacity of a Function Class



# Estimating the Capacity of a Function Class



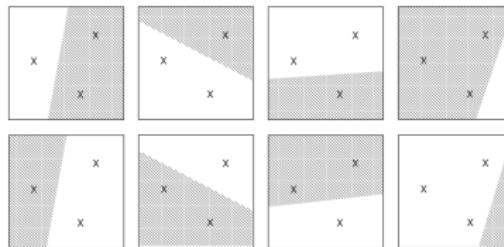
Observe:

- Every function of the class separates patterns in a certain way, thus creating a labelling.
- With labels in  $\{\pm 1\}$ , there are at most  $2^m$  different labellings for  $m$  patterns.
- A rich (high capacity) function class may be able to realize all  $2^m$  separations, thus completely shatter the  $m$  points.
- A poorer function class may not be able to shatter all  $m$  points.

# Definition VC-Dimension

Vapnik & Chervonenkis proposed **VC-dimension**  $h$  to estimate the capacity of a function class:

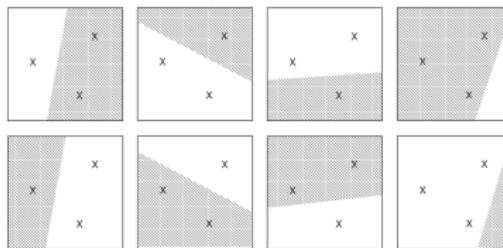
VC-dimension is defined as the largest number  $m$ , such that there exists a set of  $m$  data points which the function class can shatter (and  $\infty$  if no such  $m$  exists).



# Definition VC-Dimension

Vapnik & Chervonenkis proposed **VC-dimension**  $h$  to estimate the capacity of a function class:

VC-dimension is defined as the largest number  $m$ , such that there exists a set of  $m$  data points which the function class can shatter (and  $\infty$  if no such  $m$  exists).



What is the VC-dimension of the function class of linear functions in  $\mathbb{R}^2$ ?    2, 3, 4,  $\infty$

# Definition VC-Dimension

Vapnik & Chervonenkis proposed **VC-dimension**  $h$  to estimate the capacity of a function class:

VC-dimension is defined as the largest number  $m$ , such that there exists a set of  $m$  data points which the function class can shatter (and  $\infty$  if no such  $m$  exists).



What is the VC-dimension of the function class of linear functions in  $\mathbb{R}^2$ ?    2, 3, 4,  $\infty$



What is the VC-dimension of the function class of linear functions (2D-hyperplanes) in  $\mathbb{R}^3$ ?    2, 3, 4,  $\infty$

# Bound the Risk!

A low so-called empirical risk / average training error with zero-one loss:

$$R_{emp}[f] = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} |f(x_i) - y_i|$$

does not imply a small test error ("risk") on novel test examples drawn from the underlying data distribution  $P(x, y)$ :

$$R[f] = \int \frac{1}{2} |f(x) - y| dP(x, y)$$

# Bound the Risk!

A low so-called empirical risk / average training error with zero-one loss:

$$R_{emp}[f] = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} |f(x_i) - y_i|$$

does not imply a small test error ("risk") on novel test examples drawn from the underlying data distribution  $P(x, y)$ :

$$R[f] = \int \frac{1}{2} |f(x) - y| dP(x, y)$$

With (some) high probability, the VC-bound

$$R[f] \leq R_{emp}[f] + \phi(h, m, \delta)$$

holds, with the capacity term  $\phi$  describing the capacity of the function class ( $h$  being the VC-dimension).

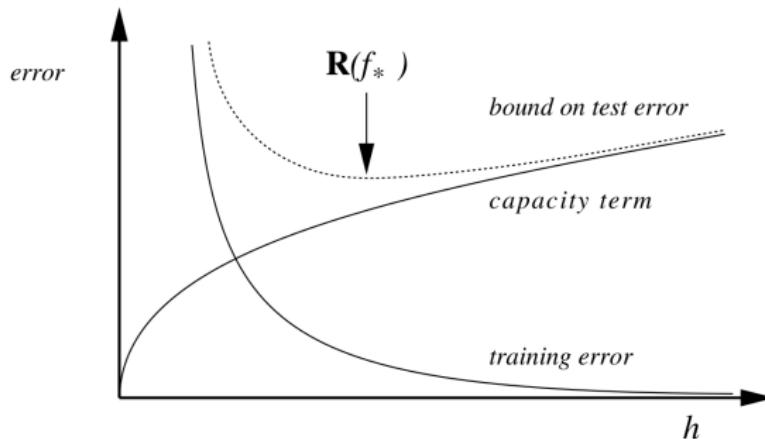
# Bound the Risk!

Remark: Large VC-dimension  $h$  increases  $\phi$ .

$$R[f] \leq R_{\text{emp}}[f] + \sqrt{\frac{h \left( \log \frac{2m}{h} + 1 \right) - \log(\delta/4)}{m}}$$

(For details refer to "Learning with Kernels", Sec. 1.3)

# Bound the Risk!



Implications:

- Large VC-dimension  $h$  of a function class implies a small empirical risk, but enlarges overall risk!
- A practitioner should prefer function classes of lower capacity, if they perform equally well on the training data, or **restrict** its capacity.

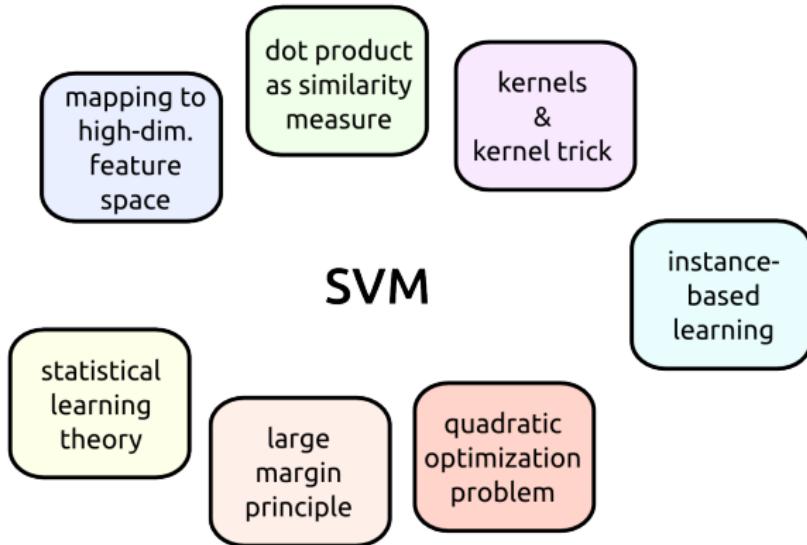
How could such a restriction be carried out?



# Lecture Overview

- 1 Mechanical Understanding of Support
- 2 Introduction Kernel Algorithms
- 3 Classification Revisited
- 4 Dot Product (and what you can do with it)
- 5 From Input Space to Feature Space
- 6 Statistical Learning Theory
- 7 Large Margin Principle: Optimal Hyperplane Classifier
- 8 Support Vector Machine (SVM)

# Concept: Large Margin Principle



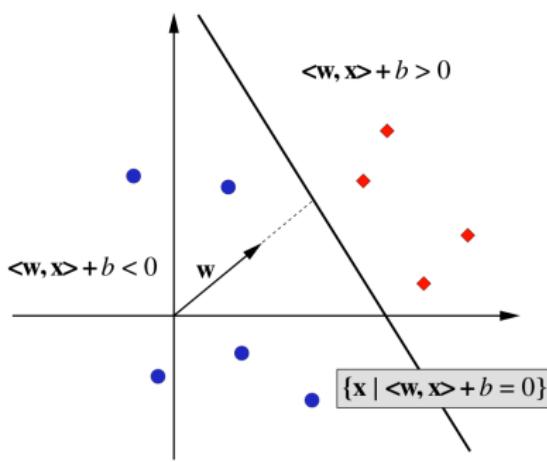
# Hyperplane Classifiers

Let's consider the function class of hyperplanes in a dot product space  $\mathcal{H}$ :

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$$

where  $\mathbf{w}, \mathbf{x} \in \mathcal{H}, b \in \mathbb{R}$ , equipped with the decision function

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



# Find the **Optimal** Hyperplane Classifier



For linear separable problems – which criterion could be used to define "*optimal*"?

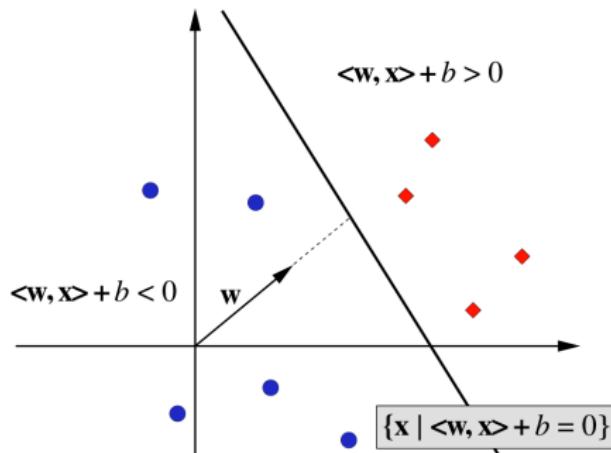
# Find the Optimal Hyperplane Classifier

For linear separable problems – which criterion could be used to define "optimal"?

Vapnik et al. proposed to use the unique hyperplane with the **maximum margin of separation** between any training point and the hyperplane.

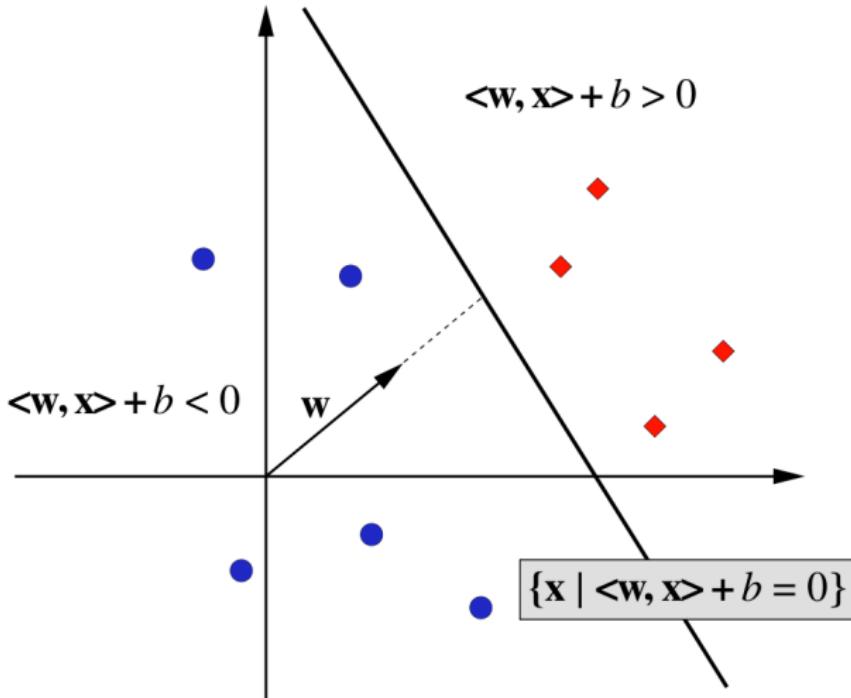
It is the solution of:

$$\operatorname{argmax}_{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}} \min \{ \|\mathbf{x} - \mathbf{x}_i\| \mid \mathbf{x} \in \mathcal{H}, \langle \mathbf{w}, \mathbf{x} \rangle + b = 0, i = 1, \dots, m \}$$

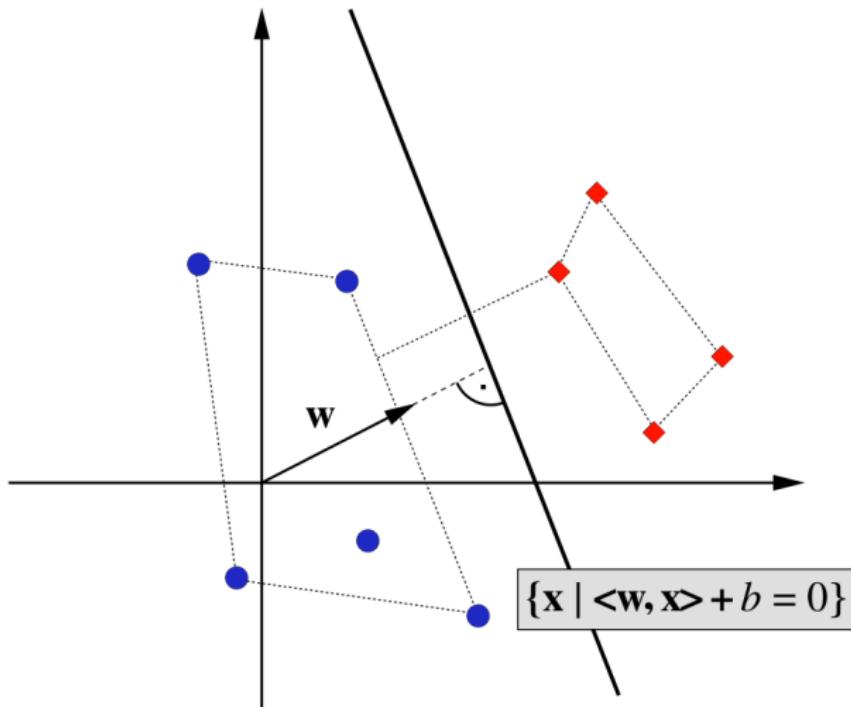


# Find the Optimal Hyperplane Classifier

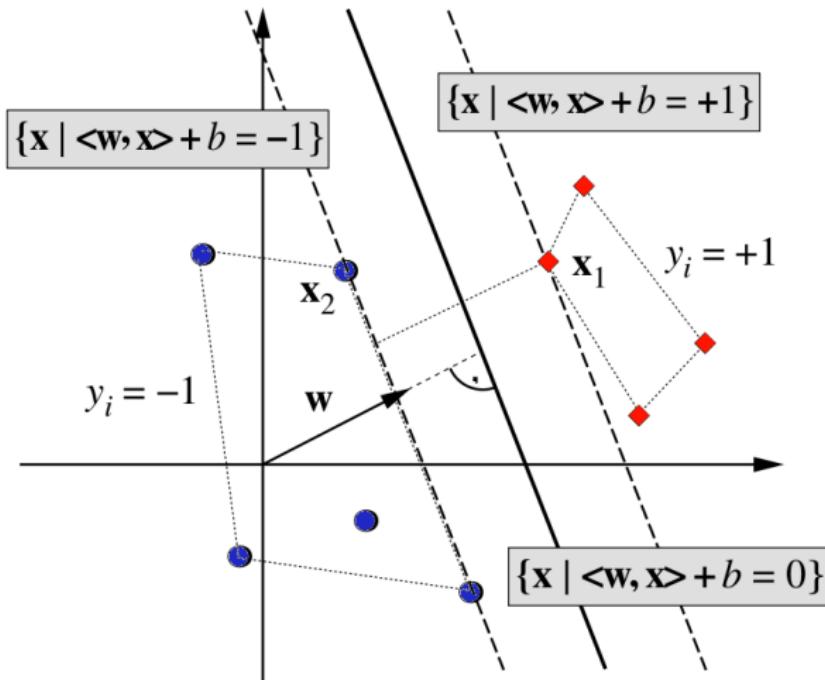
$$\underset{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}}{\operatorname{argmax}} \min\{\|\mathbf{x} - \mathbf{x}_i\| \mid \mathbf{x} \in \mathcal{H}, \langle \mathbf{w}, \mathbf{x} \rangle + b = 0, i = 1, \dots, m\}$$



# Minimal $w$ Implies Large Margin Implies **Low Capacity**

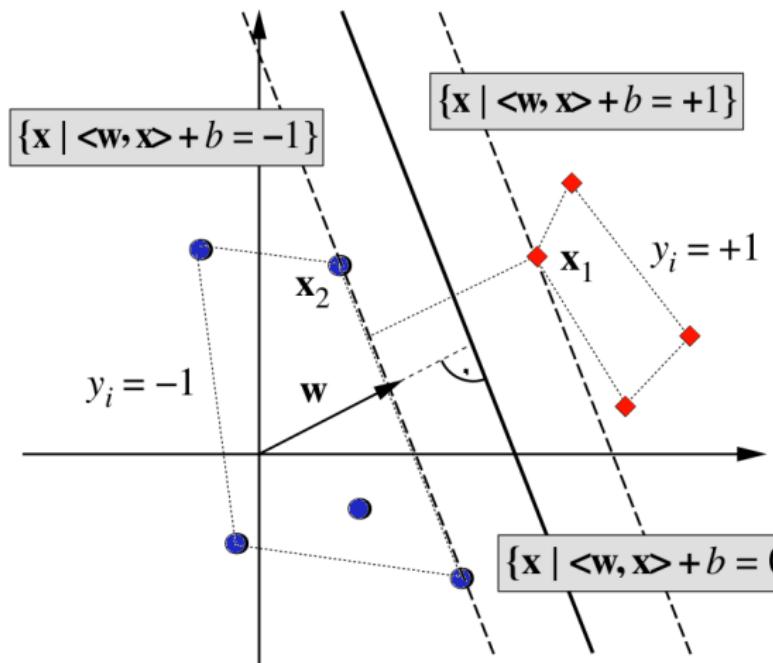


# Minimal $w$ Implies Large Margin Implies **Low Capacity**



Observe:

# Minimal $w$ Implies Large Margin Implies **Low Capacity**



Observe:

Note:

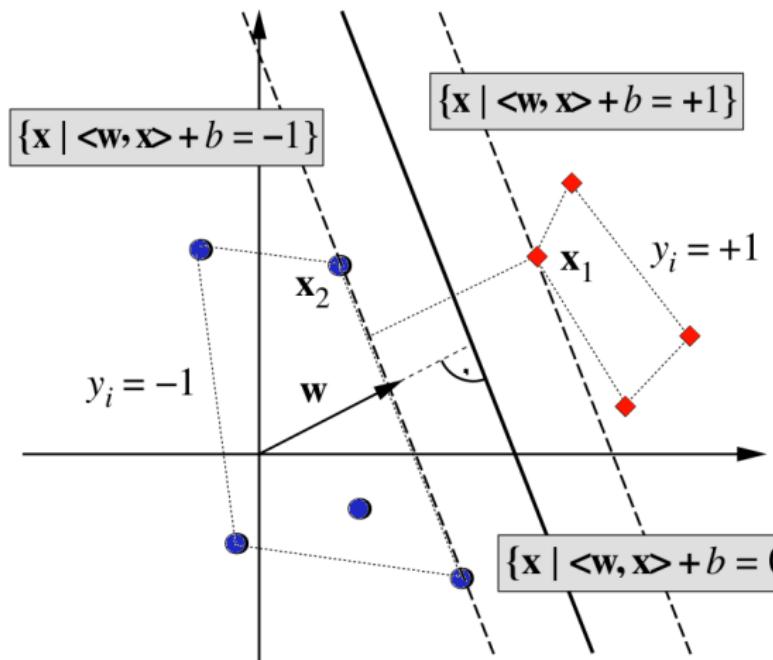
$$\langle w, x_1 \rangle + b = +1$$

$$\langle w, x_2 \rangle + b = -1$$

$$\Rightarrow \langle w, (x_1 - x_2) \rangle = 2$$

$$\Rightarrow \left\langle \frac{w}{\|w\|}, (x_1 - x_2) \right\rangle = \frac{2}{\|w\|}$$

# Minimal $w$ Implies Large Margin Implies **Low Capacity**



Note:

$$\begin{aligned} & \langle w, x_1 \rangle + b = +1 \\ & \langle w, x_2 \rangle + b = -1 \end{aligned}$$

$$\Rightarrow \langle w, (x_1 - x_2) \rangle = 2$$

$$\Rightarrow \left\langle \frac{w}{\|w\|}, (x_1 - x_2) \right\rangle = \frac{2}{\|w\|}$$

Observe:

- ① The optimal hyperplane has the largest margin.
- ② The largest margin is defined by the shortest normal vector  $w$  (... which still leads to correct classification).

# The Constrained Optimization Problem

The optimal hyperplane classifier leads to the following constrained optimization problem:

$$\operatorname{argmin}_{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}} \tau(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2$$

subject to

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \quad \text{for all } i = 1, \dots, m.$$

- Function  $\tau$  is called the **objective function**
- Constraints are **inequality constraints**, which ensure, that the labels will be correct
- The " $\geq 1$ " on the right hand side of the constraints effectively fix the scaling of  $\mathbf{w}$ .

# Solving the Constrained Optimization Problem

$$\underset{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}}{\operatorname{argmin}} \tau(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2$$

subject to

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \quad \text{for all } i = 1, \dots, m.$$

Constrained optimization problems can be solved by introducing Lagrange multipliers  $\alpha_i \in \mathbb{R}$ ,  $\alpha_i \geq 0$ , and by **minimizing** the Lagrangian:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^m \alpha_i (y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1)$$

Approach:

- Using the partial derivatives, minimize the Lagrangian  $L$  with respect to the **primal variables**  $w$  and  $b$  and maximize it with respect to the **dual variables**  $\alpha_i$ . Effectively, this finds a solution in a saddle point.
- Desired: Eliminate the primal variables by substituting  $w$  and  $b$ .

# Solving the Dual Optimization Problem

In practice, the resulting **dual** optimization problem is solved:

$$\underset{\alpha \in \mathbb{R}^m}{\operatorname{argmax}} \quad W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

subject to  $\alpha_i \geq 0$  for all  $i = 1, \dots, m$  and  $\sum_{i=1}^m \alpha_i y_i = 0$

# Solving the Dual Optimization Problem

In practice, the resulting **dual** optimization problem is solved:

$$\underset{\alpha \in \mathbb{R}^m}{\operatorname{argmax}} \quad W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle$$

subject to  $\alpha_i \geq 0$  for all  $i = 1, \dots, m$  and  $\sum_{i=1}^m \alpha_i y_i = 0$

Once the optimization problem is solved, the **hyperplane decision function** for a novel data point  $\mathbf{x}$  in the feature space  $\mathcal{H}$  can be written as:

$$f(\mathbf{x}) = \operatorname{sgn} \left( \sum_{i=1}^m \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right)$$

(Interpretation on black board)

# Solving the Dual Optimization Problem

Hyperplane decision function:

$$f(\mathbf{x}) = \operatorname{sgn} \left( \sum_{i=1}^m \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right)$$

Observe:

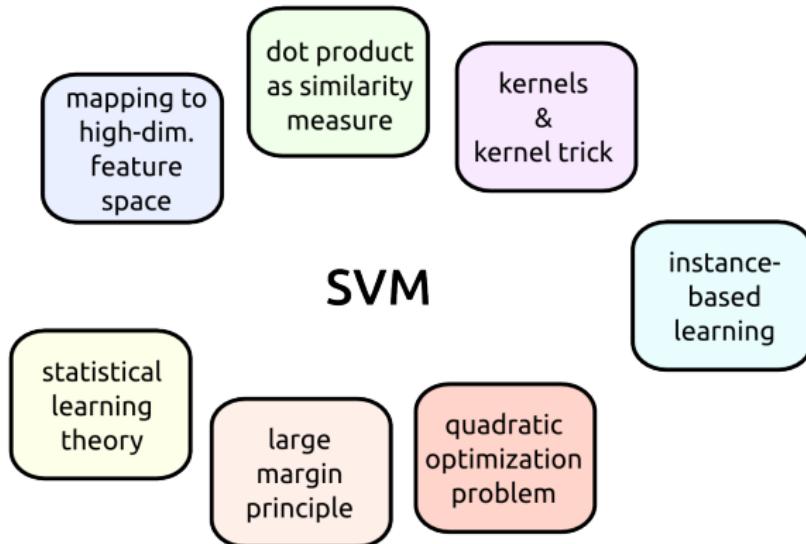
- The solution can be expressed completely in terms of dot products of training data points  $\mathbf{x}_i \rightarrow \text{"instance based learning"}$  (cp. to k-Nearest Neighbour algorithm)
- In practical problems, only a subset of the Lagrange multipliers  $\alpha_i$  are active and influence the decision hyperplane; corresponding training data points  $\mathbf{x}_i$  are called **support vectors**.
- Other training data points have  $\alpha_i = 0$ . They do not define the shape of the hyperplane (see mechanical analogy from the beginning of the lecture).

# Lecture Overview

- 1 Mechanical Understanding of Support
- 2 Introduction Kernel Algorithms
- 3 Classification Revisited
- 4 Dot Product (and what you can do with it)
- 5 From Input Space to Feature Space
- 6 Statistical Learning Theory
- 7 Large Margin Principle: Optimal Hyperplane Classifier
- 8 Support Vector Machine (SVM)

# Concepts towards SVM

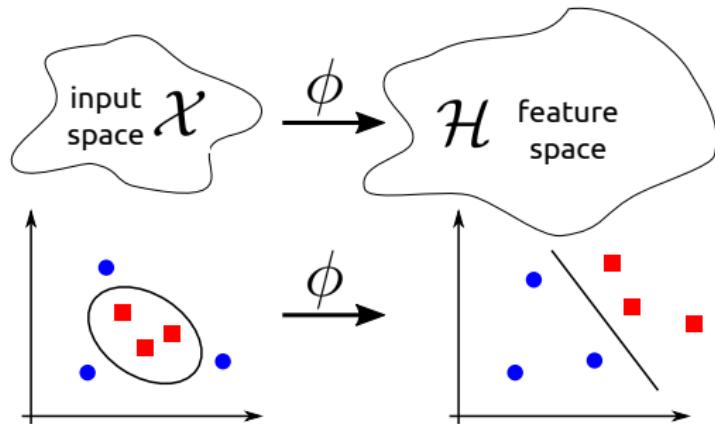
Most tools are ready now to describe support vector machines (SVMs).



To arrive at an SVM, they simply need to be put to work together.

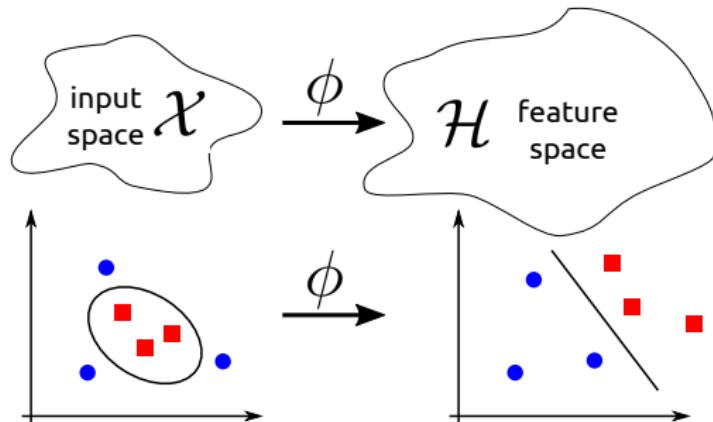
# Support Vector Machine

- SVMs implement the large margin principle and thus share the optimization problem of optimal hyperplane classifiers.
- They make use of nonlinear mappings  $\phi$ :



# Support Vector Machine

- SVMs implement the large margin principle and thus share the optimization problem of optimal hyperplane classifiers.
- They make use of nonlinear mappings  $\phi$ :



- Can we avoid performing the mapping  $\phi$  explicitly?
- Can we avoid calculating dot products in feature space explicitly?  
(YES: kernel trick)

# Kernel Trick

The kernel trick for SVM and other kernel methods is a **substitution**. It consists of the following ingredients:

- All computations can be formulated in a dot product space (think of it as the feature space  $\mathcal{H}$ )
- All computations can be executed as dot product operations in  $\mathcal{H}$ .
- To express formulas in terms of the input patterns in  $\mathcal{X}$ , we can make use of a kernel function  $k$ .
- This kernel function expresses the dot product of bold face feature vectors  $\mathbf{x}, \mathbf{x}'$  in terms of the kernel  $k$  **evaluated on input patterns  $x, x'$** !

$$k(x, x') := \langle \mathbf{x}, \mathbf{x}' \rangle$$

# Kernel Trick

The kernel trick for SVM and other kernel methods is a **substitution**. It consists of the following ingredients:

- All computations can be formulated in a dot product space (think of it as the feature space  $\mathcal{H}$ )
- All computations can be executed as dot product operations in  $\mathcal{H}$ .
- To express formulas in terms of the input patterns in  $\mathcal{X}$ , we can make use of a kernel function  $k$ .
- This kernel function expresses the dot product of bold face feature vectors  $\mathbf{x}, \mathbf{x}'$  in terms of the kernel  $k$  **evaluated on input patterns  $x, x'$** !

$$k(x, x') := \langle \mathbf{x}, \mathbf{x}' \rangle$$

→ The kernel trick replaces the mapping  $\phi$  and following dot product operations by a (simple) calculation in the input space!

# Examples of Kernels

Polynomial kernel (hyperparameter:  $d$ ):

$$k(x, x') = \langle \mathbf{x}, \mathbf{x}' \rangle^d$$

Gaussian radial basis function kernels (parameter  $\sigma > 0$ ):

$$k(x, x') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

Sigmoid kernel (parameter  $k > 0$  and parameter  $\Theta < 0$ ):

$$k(x, x') = \tanh(k \langle \mathbf{x}, \mathbf{x}' \rangle + \Theta)$$

# Decision Function for SVM

Reminder: **hyperplane decision function** for linear (separable) case in  $\mathcal{H}$ :

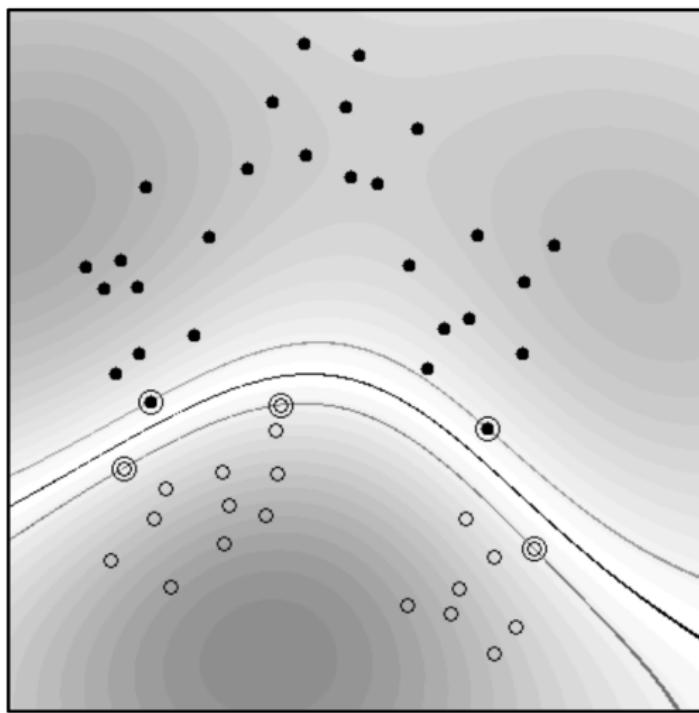
$$f(\mathbf{x}) = \operatorname{sgn} \left( \sum_{i=1}^m \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right)$$

Decision function for SVM including the kernel trick:

$$f(\mathbf{x}) = \operatorname{sgn} \left( \sum_{i=1}^m \alpha_i y_i \langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle + b \right) = \operatorname{sgn} \left( \sum_{i=1}^m \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i) + b \right)$$

→ Computations are carried out in the input space!

# Example of SVM Classifier with Radial Basis Function Kernel



# Why is the Kernel Trick Useful?

- Computations in the input space usually are preferred compared to an explicit mapping into a high dimensional (and potentially even infinite-dimensional feature space)!
- Every linear algorithm, which can be expressed by dot product operations, can be "kernelized", thus leading to a non-linear version of the algorithm. Example: kernel-PCA (see Sec. 1.7 and Sec. 14 of "Learning with Kernels")

# SVM – Pros and Cons

- Pro: SVM does not make (strong) assumptions about distributions
- Pro: High dimensionality of input data does not hurt much
- Pro: Very robust performance, even when used as black box!
- Pro: Inspecting the selected support vectors may help to understand the problem better
- Pro: SVM formulation can easily be expanded to non-separable cases in the feature space (next lecture):
  - Inclusion of slack variables  $\xi_i$ , which allow for a *slight* violation of the margin
  - Penalty for slack variables must be determined at training time by a regularization parameter  $C$  or  $\nu$  (different formulations exist)
- Pro: Very few hyperparameters to tune.
- Pro: Formulation for support vector *regression* and for *one-class SVM* (for outlier detection) both are easy to obtain from the SVM classification formulation.

## SVM – Pros and Cons

- Con: Kernel computations get expensive with many training data points! (Runtime for training? Runtime for recall?). Chunking methods like sequential minimal optimization (SMO) avoid calculating the full kernel matrix effectively, though...
- Con: Formerly outstanding performance in benchmarks has been overrun by Deep Neural Networks (at least if training data set is huge).
- Con: compared to truly linear parametric methods, interpretation of a trained SVM is harder.

## Wrap-Up: Summary by Learning Goals

Having heard this lecture and done the assignment, you will be able to ...

- describe the concepts underlying kernel methods
- understand, why it is necessary to restrict the capacity of a learning machine
- explain (non-linear) mapping to a feature space and the kernel trick
- formulate the optimization problem for optimal large-margin hyperplane classifiers
- explain the meaning of instant-based learning vs. model-based learning
- explain the role of support vectors and their interpretation with respect to the Lagrange multipliers

# Preview of Assignment for Kernel Methods

- Definition of kernels, check definitions on kernel candidates
- Get acquainted with SVM and its hyperparameters
- Get acquainted with k-PCA by using implementations of scikit-learn

# Lecture 11: Kernel Methods (Part II)

## SVM revisited

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# Lecture Overview

- 1 Motivation: Shortcomings of Linear Models
- 2 Optimization Problem of the Optimal Hyperplane Classifier
- 3 Support Vector Machine (SVM)
- 4 A Bit More On Kernels

# Lecture Overview

- 1 Motivation: Shortcomings of Linear Models
- 2 Optimization Problem of the Optimal Hyperplane Classifier
- 3 Support Vector Machine (SVM)
- 4 A Bit More On Kernels

# Shortcomings of Linear Models (I)

- Linear models (e.g. LDA for classification, OLS regression) make strong **assumptions about data distributions**.
- Linear models require data points (objects) to be vectors from **Euclidian vector space** in order to calculate distances, similarities etc.
- Linear models intrinsically can not deal with **non-linear problems**

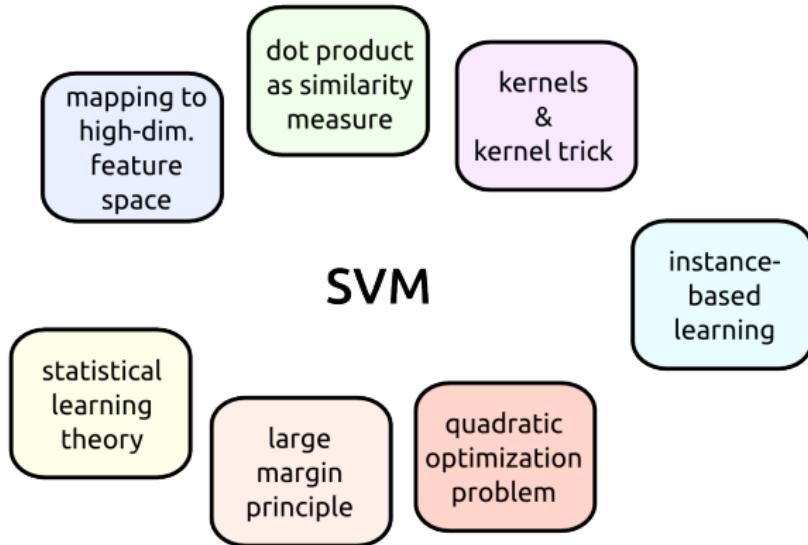
## Shortcomings of Linear Models (II)

Using domain knowledge, non-linear feature pre-processing can add helpful extra dimensions, leading to (potentially very) high dimensionality.

This is problematic:

- Large dimensionality but limited data → **numerical and computational burden**.
- **Overfitting** may become a problem!  
(how could we control for it?)

# SVM Mitigates Shortcomings of Linear Methods



# Lecture Overview

- 1 Motivation: Shortcomings of Linear Models
- 2 Optimization Problem of the Optimal Hyperplane Classifier
- 3 Support Vector Machine (SVM)
- 4 A Bit More On Kernels

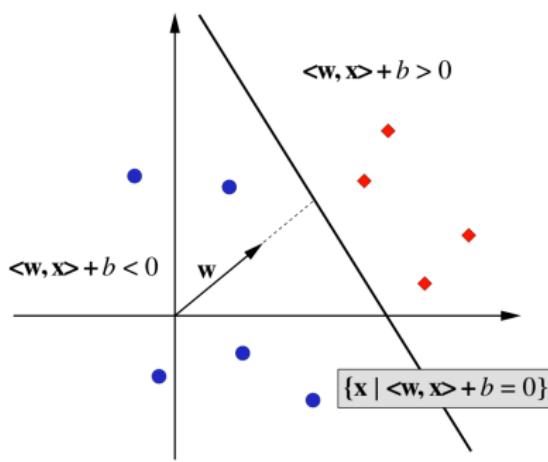
## Reminder: Hyperplane Classifiers (Linearly Separable Case)

Let's consider the function class of hyperplanes in a dot product space  $\mathcal{H}$ :

$$\langle \mathbf{w}, \mathbf{x} \rangle + b = 0$$

where  $\mathbf{w}, \mathbf{x} \in \mathcal{H}, b \in \mathbb{R}$ , and the decision function is

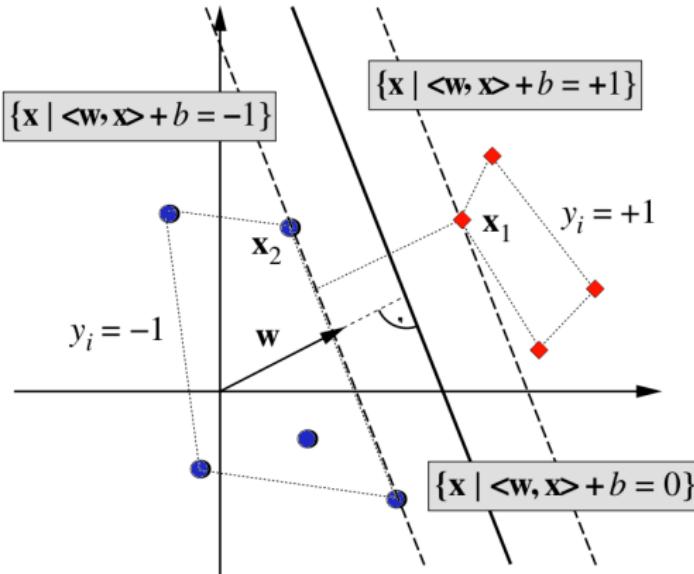
$$f(\mathbf{x}) = \text{sgn} (\langle \mathbf{w}, \mathbf{x} \rangle + b)$$



# Reminder: Find the **Optimal** Hyperplane Classifier

(For linearly separable problem:) Vapnik et al. proposed to use the unique hyperplane with the **maximum margin of separation** between any training point  $\mathbf{x}_i$  and the hyperplane:

$$\underset{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}}{\operatorname{argmax}} \min \{ \|\mathbf{x} - \mathbf{x}_i\| \mid \mathbf{x} \in \mathcal{H}, \langle \mathbf{w}, \mathbf{x} \rangle + b = 0, i = 1, \dots, m \}$$



## Reminder: The Constrained Optimization Problem

Vapnik's formulation of the optimal hyperplane classifier (separable case) led to the following constrained optimization problem:

$$\operatorname{argmin}_{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}} \tau(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2$$

subject to

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \quad \text{for all } i = 1, \dots, m.$$

- Function  $\tau$  is called the **objective function**
- Constraints are **inequality constraints**, which ensure, that the labels will be correct
- The " $\geq 1$ " on the right hand side of the constraints effectively fix the scaling of  $\mathbf{w}$ .

# Solving the Constrained Optimization Problem

$$\operatorname{argmin}_{\mathbf{w} \in \mathcal{H}, b \in \mathbb{R}} \tau(\mathbf{w}) = \frac{1}{2} \|\mathbf{w}\|^2$$

subject to

$$y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \quad \text{for all } i = 1, \dots, m.$$

Constrained optimization problems can be solved by introducing Lagrange multipliers  $\alpha_i \in \mathbb{R}$ ,  $\alpha_i \geq 0$ , and by **minimizing** the Lagrangian:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^m \alpha_i (y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1)$$

## Brief Excursion: Lagrangian Multipliers

Lagrange multipliers are a concept used in optimization.

- For a very nice explanation of Lagrange multipliers, please see [https://en.wikipedia.org/wiki/Lagrange\\_multiplier](https://en.wikipedia.org/wiki/Lagrange_multiplier)
- The simplest case (with equality constraints) and the calculations for example 1 are rather intuitive.
- If you like to follow a career in machine learning, then it may be worth while taking a full course on optimization.  
(→ Prof. Diehl, online lecture from winter semester 2015)

# The Lagrangian of the Constrained Optimization Problem

$$\text{minimize } L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^m \alpha_i (y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1)$$

Observe:

- Constraints have been included into the second part of the Lagrangian
- What do we need to maximize/minimize in order to minimize  $L$ ?



# The Lagrangian of the Constrained Optimization Problem

$$\text{minimize } L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^m \alpha_i (y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1)$$

Observe:

- Constraints have been included into the second part of the Lagrangian
- What do we need to maximize/minimize in order to minimize  $L$ ?

Answer:

- Minimize the Lagrangian  $L$  with respect to the **primal variables  $\mathbf{w}$  and  $b$**
- Maximize  $L$  with respect to the **dual variables  $\alpha_i$ .**

(Effectively, this finds a solution in a saddle point.)

# The Lagrangian of the Constrained Optimization Problem

$$\text{minimize } L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^m \alpha_i (y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1)$$

Observe:

- Constraints have been included into the second part of the Lagrangian
- What do we need to maximize/minimize in order to minimize  $L$ ?

Answer:

- Minimize the Lagrangian  $L$  with respect to the **primal variables  $\mathbf{w}$  and  $b$**
- Maximize  $L$  with respect to the **dual variables  $\alpha_i$ .**

(Effectively, this finds a solution in a saddle point.)

How would you go ahead with this problem?



# Solving the Constrained Optimization Problem

$$\text{minimize } L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^m \alpha_i (y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1)$$

Minimize the Lagrangian  $L$  with respect to the **primal variables  $\mathbf{w}$  and  $b$**   
→ the partial derivatives of  $L$  with respect to the primal variables must vanish. Thus set:

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0 \quad \text{and} \quad \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0$$

This leads to:

$$\sum_{i=1}^m \alpha_i y_i = 0 \quad \text{and} \quad \mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i$$

Observe: The solution vector  $\mathbf{w}$  has an **expansion** in terms of a subset of the training patterns.

# From Primal to Dual Problem

Eliminate the primal variables  $w$  and  $b$  from the optimization problem by substituting

$$\sum_{i=1}^m \alpha_i y_i = 0 \quad \text{and} \quad w = \sum_{i=1}^m \alpha_i y_i x_i$$

into the Lagrangian.

In most practical applications of SVMs, this resulting **dual** optimization problem is preferred for solving:

$$\underset{\alpha \in \mathbb{R}^m}{\operatorname{argmax}} \quad W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle$$

subject to  $\alpha_i \geq 0$  for all  $i = 1, \dots, m$  and  $\sum_{i=1}^m \alpha_i y_i = 0$ .

For a discussion of solvers for quadratic programs, see contribution by John Platt in the paper by Hearst et al., IEEE Intelligent Systems, 1995.

# Advantage of a Dual Representation

Making use of  $\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i$  once more, the **hyperplane decision function** for a novel data point  $\mathbf{x}$  can be written as:

$$f(\mathbf{x}) = \operatorname{sgn} \left( \sum_{i=1}^m \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + b \right)$$

Observe:

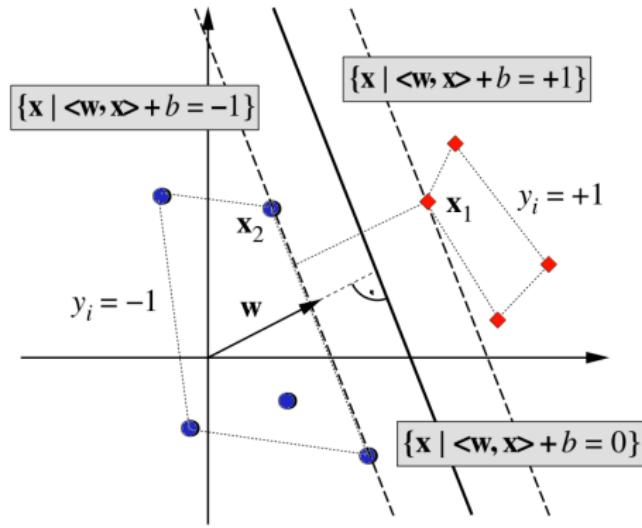
- In dual formulation, the solution can be expressed completely in terms of dot products of training data points ("**instance based learning**"), cp. to e.g. k-Nearest Neighbour algorithm.
- In practical problems, only a subset of the Lagrange multipliers  $\alpha_i$  are active (i.e. non-zero) and influence the decision hyperplane; corresponding training data points  $\mathbf{x}_i$  are called **support vectors**.
- Other training data points have  $\alpha_i = 0$ . They do not define the shape of the hyperplane (see mechanical analogy from the beginning of the lecture).

# Support Vectors are on the Margin

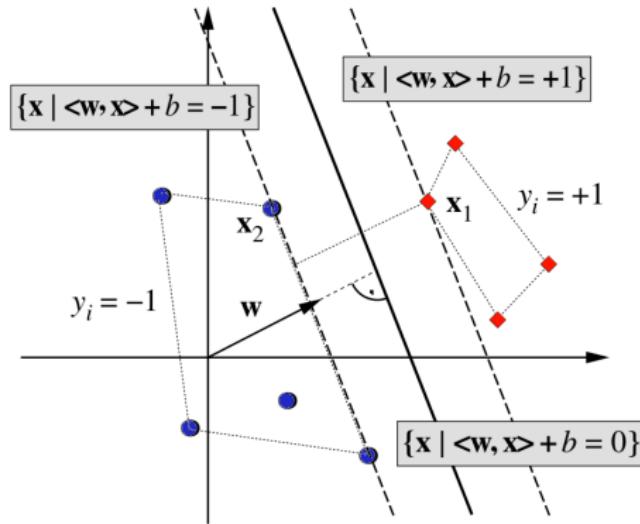
By the Karush-Kuhn-Tucker (KKT) conditions (see "Learning with Kernels", Chap. 6)

$$\alpha_i[y_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) - 1] = 0 \quad \text{for all } i = 1, \dots, m$$

the SVs of a separable training data set lie on the margin:



# Ratio of Support Vectors is Informative



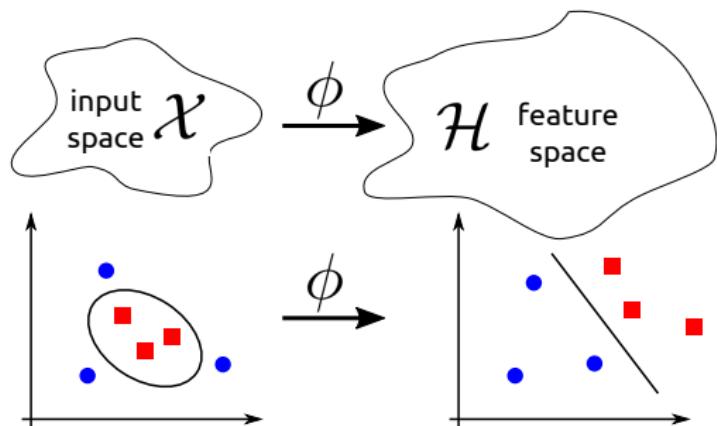
Practical aspect of SVs: It is interesting, which ratio of the training patterns end up as SVs, in order to estimate the classification performance on unseen data. **Why?**  
(Hint: Drawing + consider a LOO cross-validation.)

# Lecture Overview

- 1 Motivation: Shortcomings of Linear Models
- 2 Optimization Problem of the Optimal Hyperplane Classifier
- 3 Support Vector Machine (SVM)
- 4 A Bit More On Kernels

# Reminder Support Vector Machine

- SVMs implement the large margin principle and thus share the optimization problem of optimal hyperplane classifiers.
- They make use of nonlinear mappings:



- ... but don't compute the dot product in feature space explicitly!  
(kernel trick)

# Going Non-Linear: Decision Function for SVM

Decision function for SVM including the kernel trick:

$$f(\mathbf{x}) = \operatorname{sgn} \left( \sum_{i=1}^m \alpha_i y_i \langle \phi(x), \phi(x_i) \rangle + b \right) = \operatorname{sgn} \left( \sum_{i=1}^m \alpha_i y_i k(x, x_i) + b \right)$$

# Going Non-Linear: Decision Function for SVM

Decision function for SVM including the kernel trick:

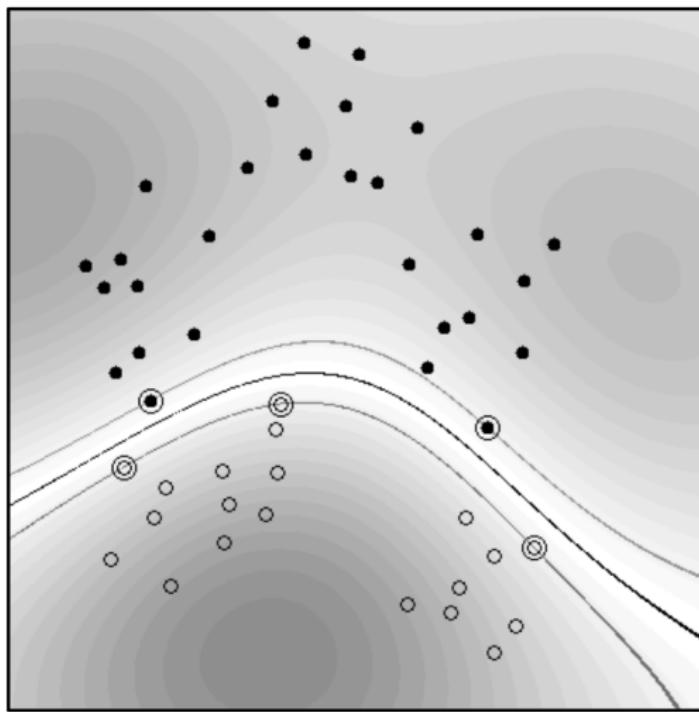
$$f(\mathbf{x}) = \operatorname{sgn} \left( \sum_{i=1}^m \alpha_i y_i \langle \phi(x), \phi(x_i) \rangle + b \right) = \operatorname{sgn} \left( \sum_{i=1}^m \alpha_i y_i \mathbf{k}(\mathbf{x}, \mathbf{x}_i) + b \right)$$

This decision function for the **non-linear** SVM can again be derived by optimizing the  $\alpha_i$  in a quadratic program. (Blue color indicates the changes compared to the linear hyperplane quadratic program!):

$$\underset{\boldsymbol{\alpha} \in \mathbb{R}^m}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j \mathbf{k}(\mathbf{x}_i, \mathbf{x}_j)$$

subject to  $\alpha_i \geq 0$  for all  $i = 1, \dots, m$  and  $\sum_{i=1}^m \alpha_i y_i = 0$

# Example of SVM Classifier with Radial Basis Function Kernel



# Lecture Overview

- 1 Motivation: Shortcomings of Linear Models
- 2 Optimization Problem of the Optimal Hyperplane Classifier
- 3 Support Vector Machine (SVM)
- 4 A Bit More On Kernels

## Recap: Examples of Kernels

*Assumption: Input space already has a vectorial representation.  
( $\phi$  is identity.)*

Polynomial kernel (hyperparameter:  $d$ ):

$$k(x, x') = \langle x, x' \rangle^d$$

Gaussian radial basis function kernels (parameter  $\sigma > 0$ ):

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

Sigmoid kernel (parameter  $k > 0$  and parameter  $\Theta < 0$ ):

$$k(x, x') = \tanh(k \langle x, x' \rangle + \Theta)$$

# RBF Kernel Revisited

The RBF kernel is often mentioned as an example, how the SVM can *implicitly* project data to an **infinite-dimensional** feature space.

Intuition:

- RBF (Gaussian) kernel

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

is a simple modification of this kernel:

$$k'(x, x') = \exp\left(-\frac{\langle x, x' \rangle}{\sigma^2}\right)$$

- Use power series expansion:

$$k'(x, x') = \sum_{n=0}^{+\infty} \frac{\langle x, x' \rangle^n}{\sigma^2 n!}$$



- Does this numerator look familiar to you?

# RBF Kernel Revisited

$$k'(x, x') = \sum_{n=0}^{+\infty} \frac{\langle x, x' \rangle^n}{\sigma^2 n!}$$

- Numerator contains a **polynomial kernel of degree n**.
- → RBF kernel can be seen as a combination of all polynomial kernels of degree  $n \geq 0$ .
- Please note: RBF kernel maps to this infinite-dimensional feature space only *implicitly*.

# Kernel Trick Revisited (I)

Why can we calculate something **implicitly** in a high-dimensional feature space?

Intuition with an example:

Given we have a mapping  $\phi$ , mapping from  $\mathbb{R}^3$  to  $\mathbb{R}^9$ :

$$\phi(x_1, x_2, x_3) = (x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3,)$$

-   $\langle \mathbf{x}, \mathbf{x}' \rangle$

How many operations are necessary to perform the mapping to  $\mathbb{R}^9$  and to calculate the dot product?

# Kernel Trick Revisited (I)

Why can we calculate something **implicitly** in a high-dimensional feature space?

Intuition with an example:

Given we have a mapping  $\phi$ , mapping from  $\mathbb{R}^3$  to  $\mathbb{R}^9$ :

$$\phi(x_1, x_2, x_3) = (x_1x_1, x_1x_2, x_1x_3, x_2x_1, x_2x_2, x_2x_3, x_3x_1, x_3x_2, x_3x_3, )$$

-   $\langle \mathbf{x}, \mathbf{x}' \rangle$

How many operations are necessary to perform the mapping to  $\mathbb{R}^9$  and to calculate the dot product?

- 9+9 multiplications for the mapping, 9 multiplications and 8 additions for the dot product → **35** operations.

## Kernel Trick Revisited (II)

Same situation,  $x, x' \in \mathbb{R}^3$

- Now consider using the polynomial kernel function of degree 2 instead of explicit mapping using  $\phi$ :  
 $k(x, x') = \langle x, x' \rangle^2$
-  How many operations are necessary now?

## Kernel Trick Revisited (II)

Same situation,  $x, x' \in \mathbb{R}^3$

- Now consider using the polynomial kernel function of degree 2 instead of explicit mapping using  $\phi$ :  
 $k(x, x') = \langle x, x' \rangle^2$
-  How many operations are necessary now?
- 3 multiplications and two additions for the dot product, one multiplication for the squaring  $\rightarrow \mathbf{6}$  operations.

This shows, how kernels help to save computation time, **if dot products ONLY** are required in the high-dimensional space (but nothing else).

## Wrap-Up: Summary by Learning Goals

Having heard this lecture and working on the SVM assignment, you will be able to:

- formulate the optimization problems for
  - optimal large-margin hyperplane classifiers
  - SVM
  - soft-margin SVM
- obtain the Lagrange formulations
- explain how to get from the primal to the dual formulations
- Give examples, why the kernel trick is useful.

# Lecture 12: Kernel Methods (Part III)

## Soft-margin SVM, SVR, kPCA

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# Lecture Overview

1 Soft-Margin SVM

2 Support Vector Regression at a Glance

3 Kernel Principal Component Analysis at a Glance

4 Kernel Methods: The Bigger Picture

# SVM and Non-Separable Data

Separable hyperplane may not exist in the input space  $\mathcal{X}$  for practical data. Why?



# SVM and Non-Separable Data

Separable hyperplane may not exist in the input space  $\mathcal{X}$  for practical data. Why?

Possible reasons:

- high noise level may cause an overlap between classes.
- training data set contains mislabelled data points (e.g. web forms)

In these cases, it may be desirable not to enforce full separability in feature space  $\mathcal{H}$  but instead to allow for *small / few* violations of the margin constraints during the SVM training!

# Soft-Margin SVM

Implementation of *acceptable* violations in a soft-margin SVM in three steps:

- ① introduce one so-called slack variable  $\xi_i \geq 0$  per training pattern.
- ② utilize the slack variables to relax the constraints for the optimization problem to:  
 $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \quad \text{for all } i = 1, \dots, m.$
- ③ Limit the sum of the slacks by an upper bound on the training error in the objective function by minimizing:

$$\tau(\mathbf{w}, \xi) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i$$

What is the role of  $C$ ?



# Soft-Margin SVM

Implementation of *acceptable* violations in a soft-margin SVM in three steps:

- ① introduce one so-called slack variable  $\xi_i \geq 0$  per training pattern.
- ② utilize the slack variables to relax the constraints for the optimization problem to:  
 $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i \quad \text{for all } i = 1, \dots, m.$
- ③ Limit the sum of the slacks by an upper bound on the training error in the objective function by minimizing:

$$\tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i$$

What is the role of  $C$ ? The constant  $C$  determines a trade-off between enlarging the margin and minimizing the training error. [Blackboard]

# Quadratic Program for Soft-Margin SVM

Incorporating the kernel trick and reformulating using Lagrange multipliers, the quadratic program for soft-margin SVMs looks very similar to that of the hard-margin SVM:

$$\underset{\alpha \in \mathbb{R}^m}{\text{maximize}} \quad W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

subject to  $0 \leq \alpha_i \leq C$  for all  $i = 1, \dots, m$  and  $\sum_{i=1}^m \alpha_i y_i = 0$

Comments:

- Constant  $C$  is a regularization hyperparameter: optimize it for each data set.
- $C$  limits the influence of a single  $\alpha_i$ , thus not allowing a single SV to "push" upon the decision hyperplane too strongly.

# Quadratic Program for Soft-Margin SVM

$$\underset{\alpha \in \mathbb{R}^m}{\text{maximize}} \quad W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

subject to  $0 \leq \alpha_i \leq C$  for all  $i = 1, \dots, m$  and  $\sum_{i=1}^m \alpha_i y_i = 0$

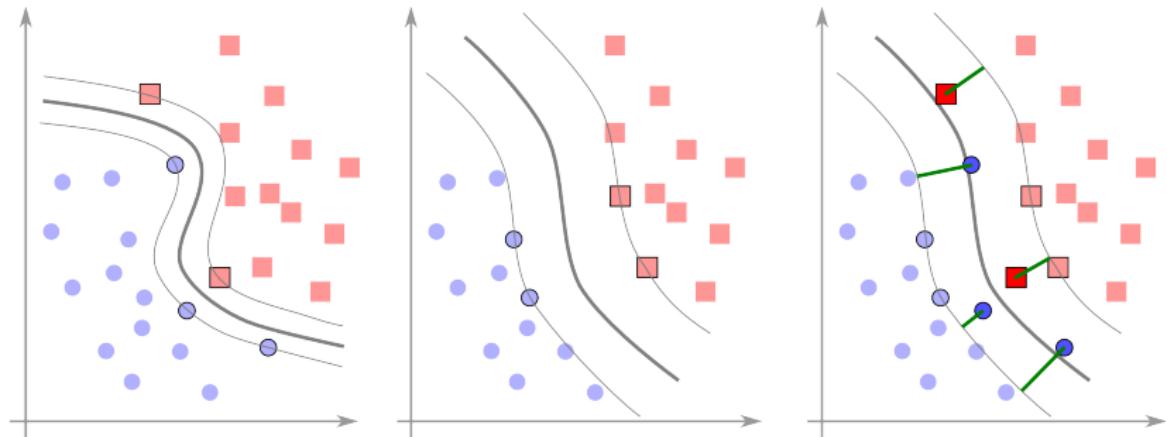
Comments (continued):

- Scalar  $b$  can be computed exploiting that for all SVs  $x_i$  with  $\alpha_i < C$  the corresponding slack variable  $\xi_i$  is zero.
- Intuition: scalar  $b$  shifts the decision hyperplane such, that SVs with zero slack lie on  $\pm 1$  lines.
- A parametrization via  $\nu$  (with  $0 < \nu \leq 1$ ) is an alternative to using  $C$  and easier to handle.
- $\nu$  lower-bounds (relative to the number of training data points) the fraction of training patterns which can become SVs and thus can have non-zero slacks and it upper-bounds the fraction of margin errors (see section 7.5 of "Learning with Kernels").

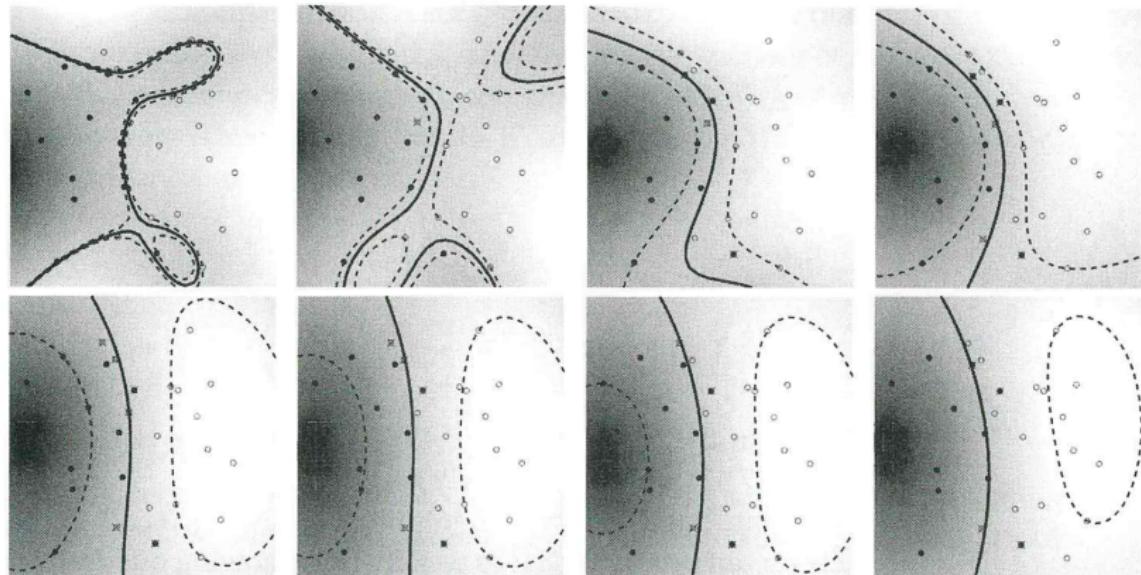
# Soft-Margin vs. Hard-Margin SVM for Separable Data

Even, if a (complicated) separable hyperplane exists, it may be a good idea to try soft-margin SVM in order to avoid overfitting:

- get a solution with a function class of lower capacity
- enlarge the margin ( $\rightarrow$  smoother hyperplane)
- reduce  $R[f]$

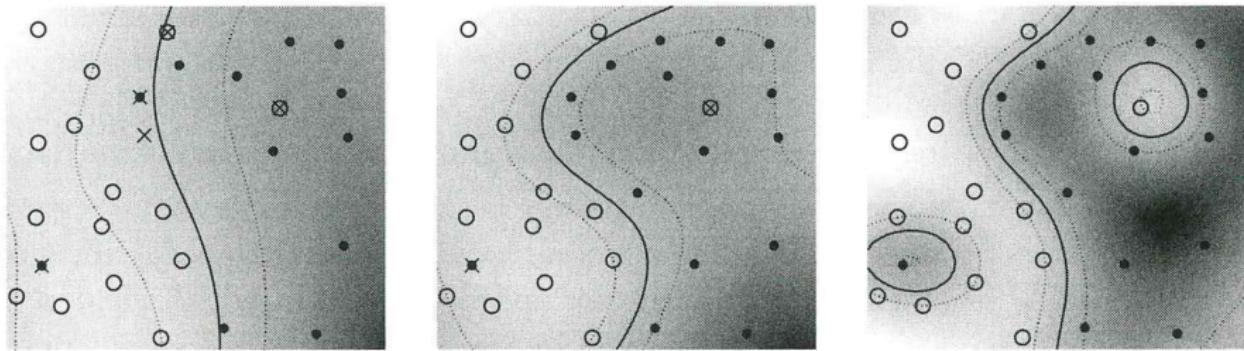


# Effect of $\nu$ -Regularization on SVM for Classification



**Figure 7.9** Toy problem (task: separate circles from disks) solved using  $\nu$ -SV classification, with parameter values ranging from  $\nu = 0.1$  (top left) to  $\nu = 0.8$  (bottom right). The larger we make  $\nu$ , the more points are allowed to lie inside the margin (depicted by dotted lines). Results are shown for a Gaussian kernel,  $k(x, x') = \exp(-\|x - x'\|^2)$ .

# Kernel Parameters Can Also Regularize!



**Figure 7.10** 2D toy example of a binary classification problem solved using a soft margin SVC. In all cases, a Gaussian kernel (7.27) is used. From left to right, we decrease the kernel width. Note that for a large width, the decision boundary is almost linear, and the data set cannot be separated without error (see text). Solid lines represent decision boundaries; dotted lines depict the edge of the margin (where (7.34) becomes an equality with  $\xi_i = 0$ ).

# Lecture Overview

1 Soft-Margin SVM

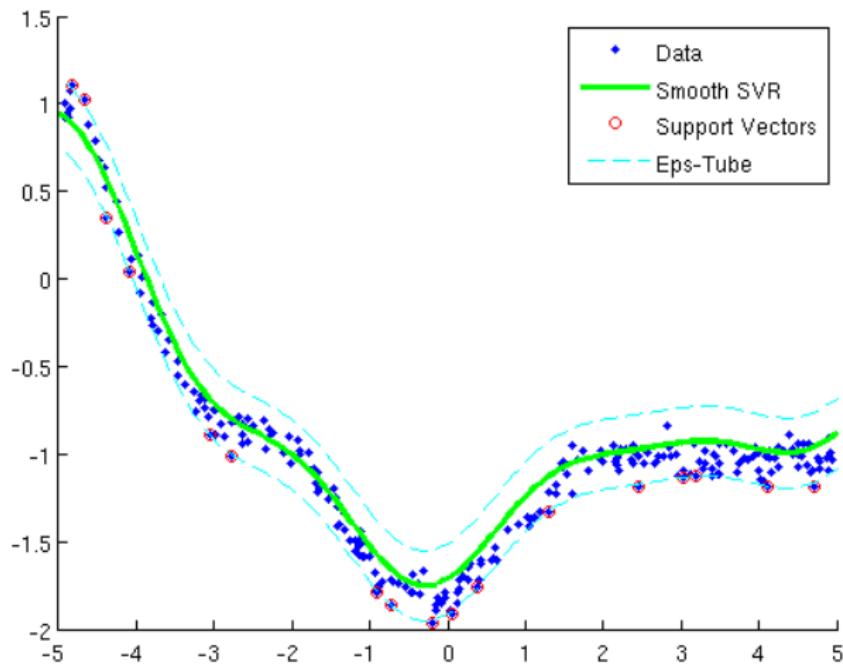
2 Support Vector Regression at a Glance

3 Kernel Principal Component Analysis at a Glance

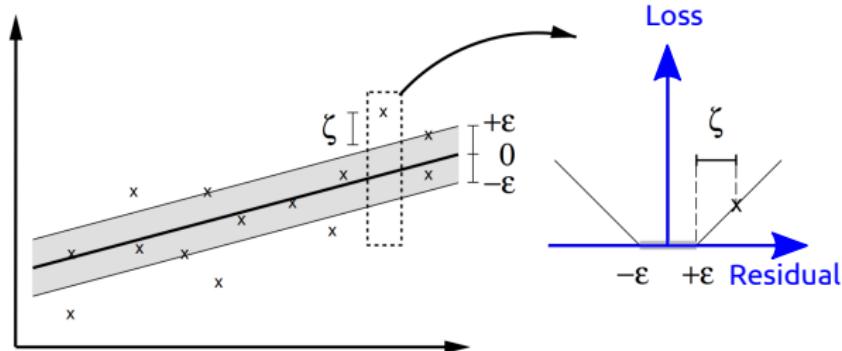
4 Kernel Methods: The Bigger Picture

# SV Method for Regression Problems

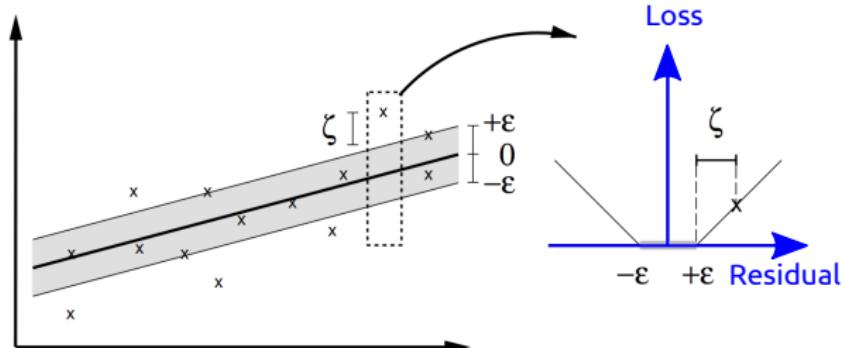
Moving from class labels  $y_i \in \{\pm 1\}$  to continuous labels, a support vector regression model can estimate continuous non-linear functions.



# SVM → SVR: Margin → Tube



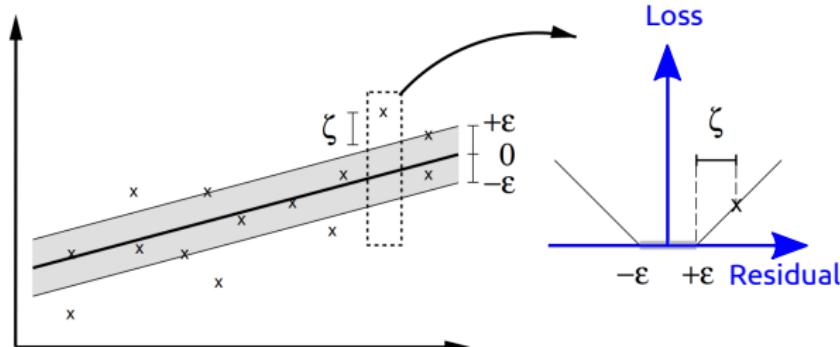
# SVM → SVR: Margin → Tube



Key ingredients for support vector regression (SVR):

- Kernel trick (of course...)
- Use the  $\epsilon$ -insensitive loss function to obtain a smooth regression.  
(regularizing similar to a large margin).

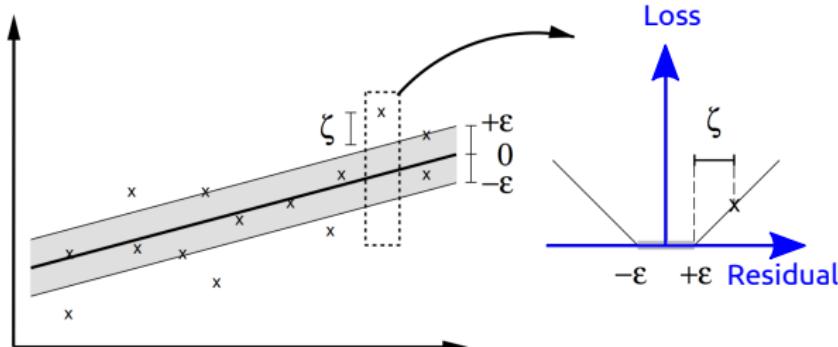
# SVM → SVR: Margin → Tube



Key ingredients for support vector regression (SVR):

- Kernel trick (of course...)
- Use the  $\epsilon$ -insensitive loss function to obtain a smooth regression. (regularizing similar to a large margin).
- Introduce (two types) of slack variables  $\xi_i$  ("xi"), which allow for violations of the  $\epsilon$  tube. Limit their influence by regularization with a constant  $C$ .
- Choose  $C, \epsilon \geq 0$  as hyperparameters e.g. via cross-validation.

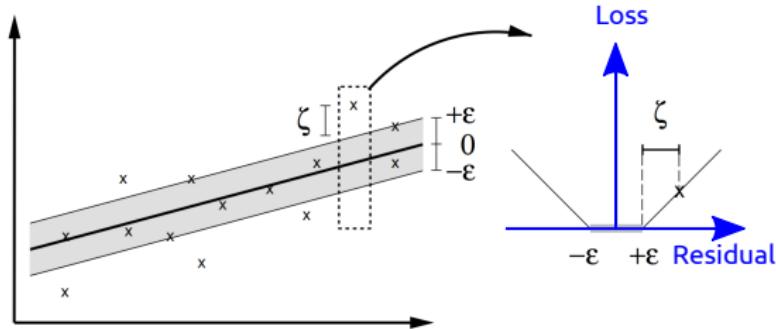
# SVM → SVR: Margin → Tube



Key ingredients for support vector regression (SVR):

- Kernel trick (of course...)
- Use the  $\epsilon$ -insensitive loss function to obtain a smooth regression. (regularizing similar to a large margin).
- Introduce (two types) of slack variables  $\xi_i$  ("xi"), which allow for violations of the  $\epsilon$  tube. Limit their influence by regularization with a constant  $C$ .
- Choose  $C, \epsilon \geq 0$  as hyperparameters e.g. via cross-validation.
- Formulation via Lagrange multipliers, solve quadratic problem.

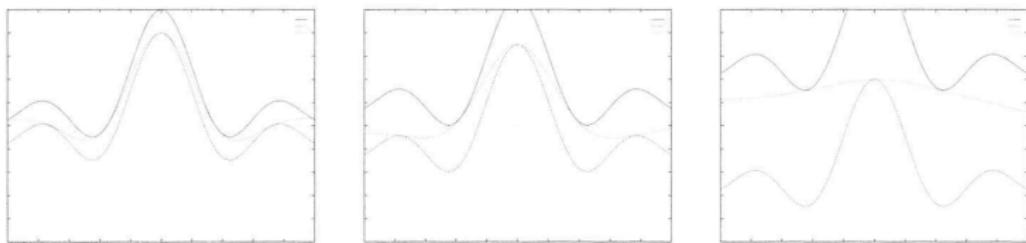
# Support Vector Regression (SVR)



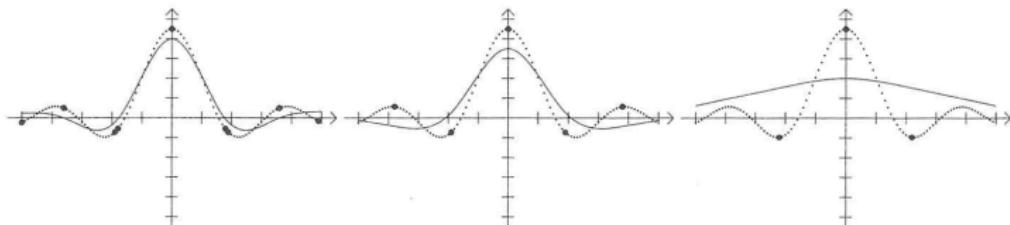
Further reading:

- <http://www.svms.org/regression/SmSc98.pdf>
- Learning with Kernels, Sec. 9

# Effect Of Epsilon Upon Smoothness of Regression Function



**Figure 9.3** From top to bottom: approximation of the function  $\text{sinc } x$  with precisions  $\varepsilon = 0.1, 0.2$ , and  $0.5$ . The solid top and dashed bottom lines indicate the size of the  $\varepsilon$ -tube, here drawn around the target function  $\text{sinc } x$ . The dotted line between them is the regression function.



**Figure 9.4** Left to right: regression (solid line), data points (small dots) and SVs (big dots) for an approximation of  $\text{sinc } x$  (dotted line) with  $\varepsilon = 0.1, 0.2$ , and  $0.5$ . Note the decrease in the number of SVs.

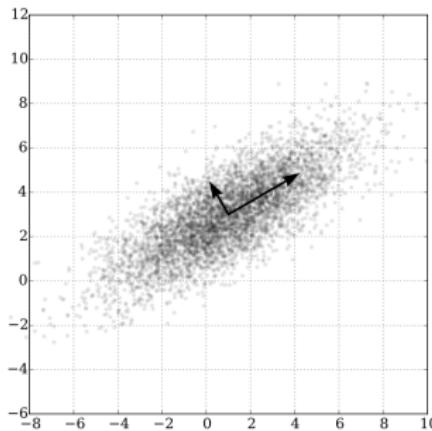
# Lecture Overview

- 1 Soft-Margin SVM
- 2 Support Vector Regression at a Glance
- 3 Kernel Principal Component Analysis at a Glance
- 4 Kernel Methods: The Bigger Picture

# Reminder Linear Principal Component Analysis

Linear principal component analysis (PCA)...

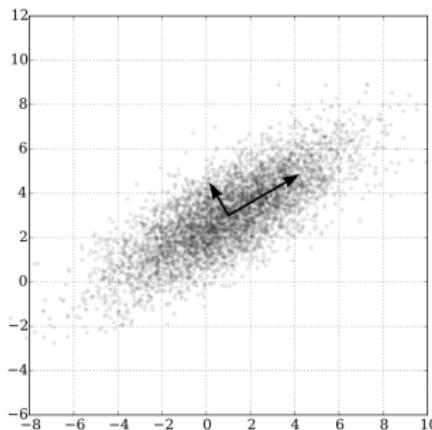
- ... obtains directions of largest variance.
- Directions are obtained as solutions (eigenvectors) of an eigenvalue problem.
- Corresponding eigenvalues deliver a sorting of the eigenvectors.



# Reminder Linear Principal Component Analysis

Linear principal component analysis (PCA)...

- ... obtains directions of largest variance.
- Directions are obtained as solutions (eigenvectors) of an eigenvalue problem.
- Corresponding eigenvalues deliver a sorting of the eigenvectors.

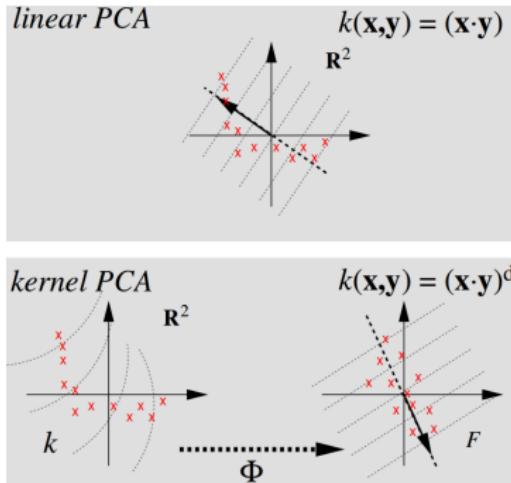


Role of dimensionality? Role of data set size?

# Kernel PCA (kPCA)

Key ideas of kPCA:

- formulate all calculations of linear PCA (eigenvalue problem!) via dot products of the input space  $\mathcal{X}$ .
- obtain a non-linear version of linear PCA by mappings  $\phi$  from input space  $\mathcal{X}$  to a high-dimensional feature space  $\mathcal{H}$ !
- use the kernel trick, thus compute everything in input space  $\mathcal{X}$ !



# Kernel PCA (kPCA)

kPCA is a non-linear feature extractor. It computes  $n$  nonlinear feature functions

$$f_1(x) = \sum_{i=1}^m \alpha_i^1 k(x_i, x), \quad \dots, \quad f_n(x) = \sum_{i=1}^m \alpha_i^n k(x_i, x)$$

where  $\alpha_i^n$  are (up to a normalizing constant) the components of the  $n$ th eigenvector of the kernel matrix  $K_{ij} := (k(x_i, x_j))$ , with  $n = 1, \dots, m$

Observations:

- All mathematical and statistical properties of PCA carry over to kPCA (with the modification, that they become statements about a set of data patterns in  $\mathcal{H}$  instead of vectors in  $\mathcal{X}$ ).
- The number of feature functions is determined by the number of patterns (NOT by dimensionality of input space!)
- Computation of the Gram matrix  $K_{ij} := k(x_i, x_j)$  is expensive for large number of patterns  $m$  ( $\rightarrow$  iterative solutions!).

# Example of kPCA

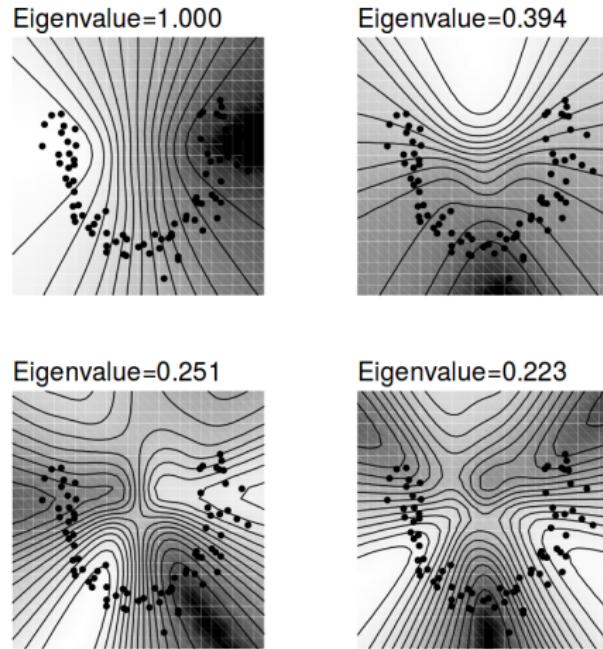


Fig. 10. The first 4 nonlinear features of Kernel-PCA using a sigmoidal Kernel on the data set from Figure 9. The Kernel-PCA components capture the nonlinear structure in the data, e.g. the first feature (upper left) is better adapted to the curvature of the data than the respective linear feature from Figure 9 (figure

# How is kPCA Applied?

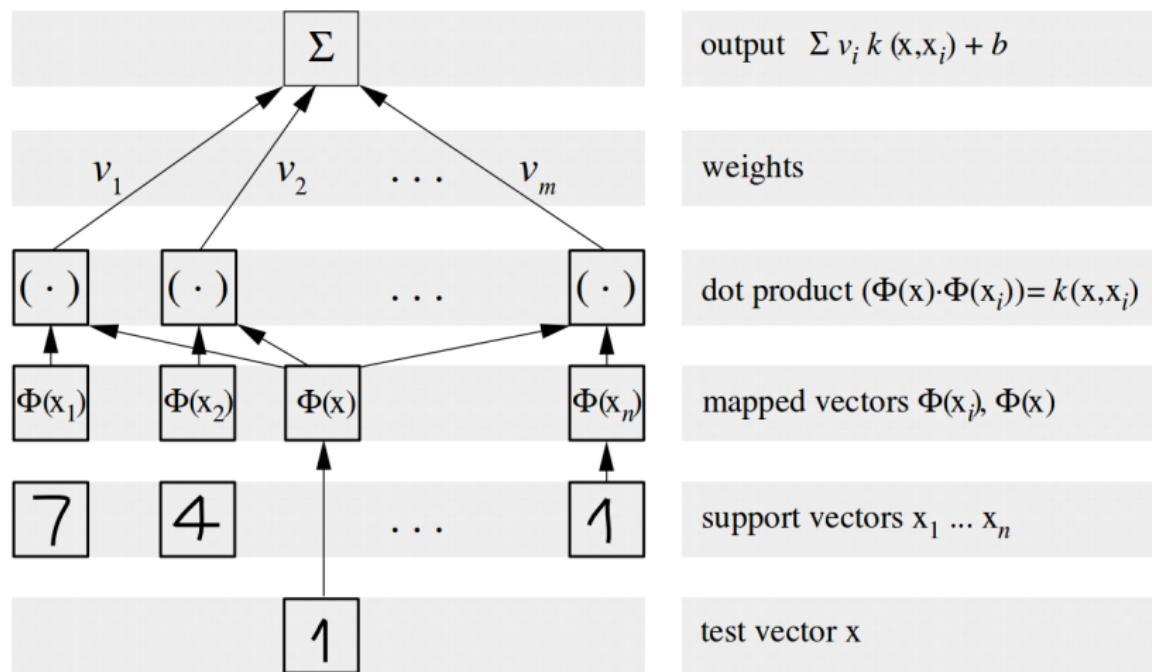
- As **preprocessing** step for non-kernel methods (e.g. clustering).
- To derive **introspection** into an SVM solution (posthoc visualization of variance isolines of feature functions in  $\mathcal{X}$  using the SVM hyperparameters).
- To understand, why a hard classification problem is actually hard (see "relevant dimensionality estimation" (RDE) method by Braun et al., JMLR 2008):
  - ① Reason 1: the problem is intrinsically complex / high dimensional
  - ② Reason 2: the problem is intrinsically simple, but data is very noisy

Further reading on kPCA: Sec. 14 of "Learning with Kernels".

# Lecture Overview

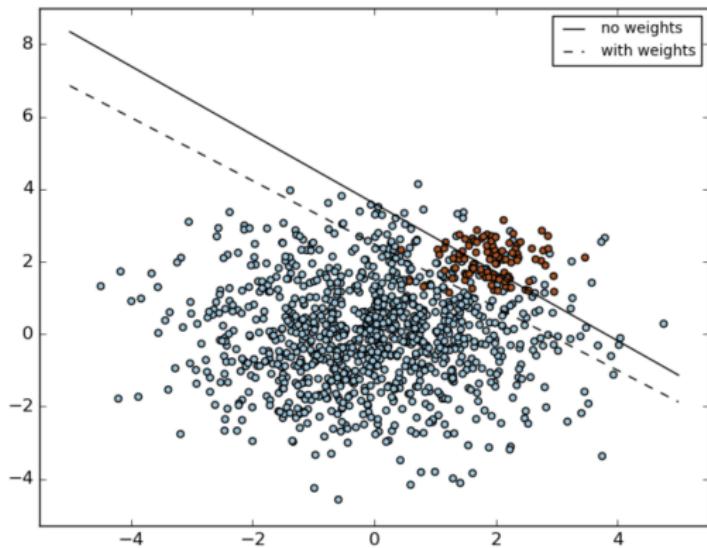
- 1 Soft-Margin SVM
- 2 Support Vector Regression at a Glance
- 3 Kernel Principal Component Analysis at a Glance
- 4 Kernel Methods: The Bigger Picture

# Support Vector Models Can Be Expressed as Neural Networks



# Dealing With Unbalanced Classes in SVM Training

Some classification problems have very unbalanced numbers of training patterns. By weighting the influence of misclassifications (via the slack variables!) separately per class[[[ , very unbalanced problems become tractable.



# Wrap-Up: Summary by Learning Goals

Having heard this lecture and done the last assignment,  
you will be able to:

- formulate the optimization problems for
  - optimal large-margin hyperplane classifiers
  - (hard margin) SVM
  - soft-margin SVM
- obtain the Lagrange formulations
- explain how to get from the primal to the dual formulations
- motivate the use of slack variables, interpret support vectors
- explain (top-down) the idea of support vector regression and of kernel PCA

# Lecture 13: Decision Trees and Random Forests

Machine Learning, Summer Term 2019

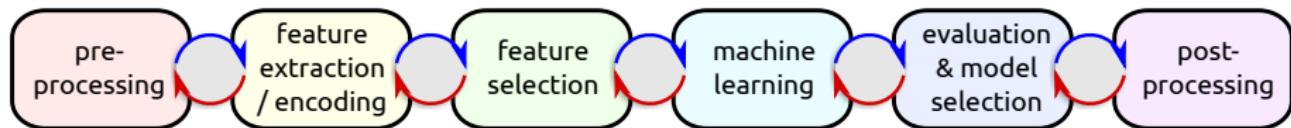
July 1, 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# The Big Picture



- Lecture 1: overview
- Lecture 2-6: linear methods
- Lecture 7-9: algorithm-independent principles
- Lectures 10-15: nonlinear methods
  - Lecture 10-12: kernel-based methods
  - Lectures 13-14: tree-based methods and ensembles
  - Lecture 15: neural networks

# Lecture Overview

## 1 Decision and Regression Trees

- Regression Trees
- Classification Trees (= Decision Trees)

## 2 Bagging

## 3 Random Forests

# Lecture Overview

## 1 Decision and Regression Trees

- Regression Trees
- Classification Trees (= Decision Trees)

## 2 Bagging

## 3 Random Forests

# Motivation for Trees and Forests

For many applications, random forests are the **best off-the-shelf model**

# Motivation for Trees and Forests

For many applications, random forests are the **best off-the-shelf model**

- Trees: easy to **interpret**

# Motivation for Trees and Forests

For many applications, random forests are the **best off-the-shelf model**

- Trees: easy to **interpret**
- Directly handle **categorical features**

# Motivation for Trees and Forests

For many applications, random forests are the **best off-the-shelf model**

- Trees: easy to **interpret**
- Directly handle **categorical features**
- Scalable to **many data points** (they are fast)

# Motivation for Trees and Forests

For many applications, random forests are the **best off-the-shelf model**

- Trees: easy to **interpret**
- Directly handle **categorical features**
- Scalable to **many data points** (they are fast)
- Scalable to **many features** (they are automated feature selectors)

# Motivation for Trees and Forests

For many applications, random forests are the **best off-the-shelf model**

- Trees: easy to **interpret**
- Directly handle **categorical features**
- Scalable to **many data points** (they are fast)
- Scalable to **many features** (they are automated feature selectors)
- Random forests: **robust** performance even for **small datasets**

# Motivation for Trees and Forests

For many applications, random forests are the **best off-the-shelf model**

- Trees: easy to **interpret**
- Directly handle **categorical features**
- Scalable to **many data points** (they are fast)
- Scalable to **many features** (they are automated feature selectors)
- Random forests: **robust** performance even for **small datasets**
- Random forests: **robust** to their **hyperparameter settings**
  - In contrast to, e.g., SVMs or neural networks

# Trees and Forests are Extremely Popular Models



## Classification and Regression Trees

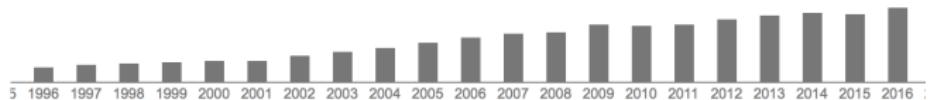
Leo Breiman 1928-2005

Authors Leo Breiman, Jerome H Friedman, Richard A Olshen, Charles J Stone

Publication date 1999/5

Publisher CRC Press, New York

Total citations [Cited by 34504](#)



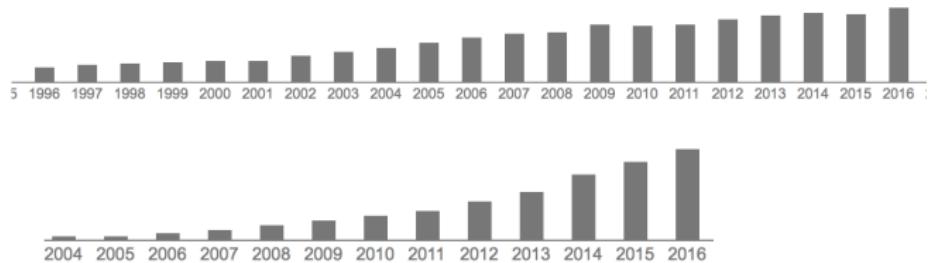
# Trees and Forests are Extremely Popular Models



## Classification and Regression Trees

Leo Breiman 1928-2005

Authors	Leo Breiman, Jerome H Friedman, Richard A Olshen, Charles J Stone
Publication date	1999/5
Publisher	CRC Press, New York
Total citations	<a href="#">Cited by 34504</a>



Scholar articles

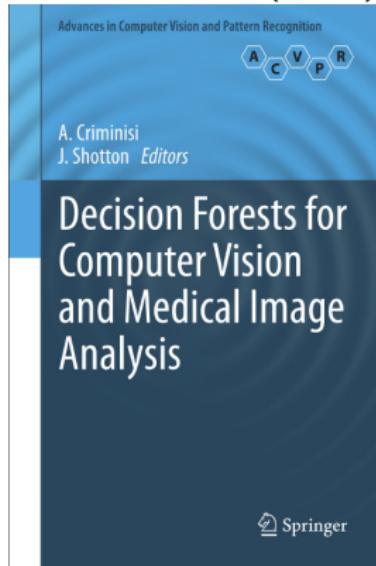
[Random forests](#)

L Breiman - Machine learning, 2001

[Cited by 29053](#) - [Related articles](#) - All 68 versions

# Acknowledgement

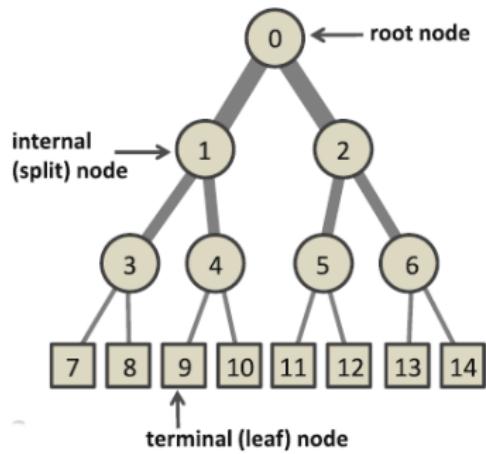
Most visualizations in this lecture are taken from this excellent book by Criminisi et. al (2013):



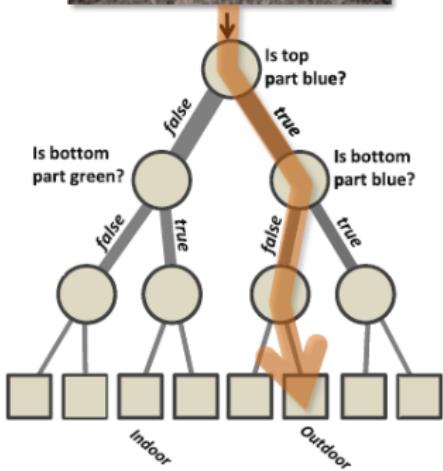
PDF of entire book available from university machines:  
<http://link.springer.com/book/10.1007/978-1-4471-4929-3>

# Decision and Regression Trees – General Idea

A general tree structure



A decision tree



# Lecture Overview

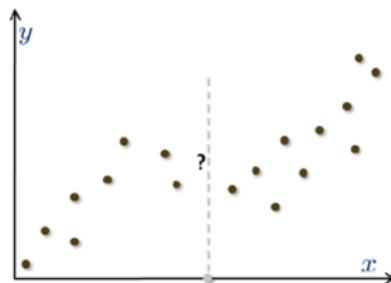
## 1 Decision and Regression Trees

- Regression Trees
- Classification Trees (= Decision Trees)

## 2 Bagging

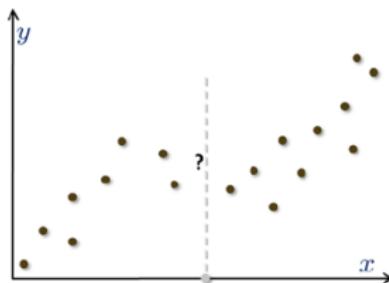
## 3 Random Forests

# Regression Trees

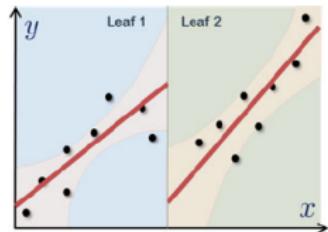


Idea: fit simple model to subset of the data

# Regression Trees

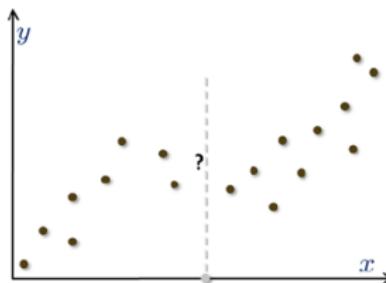


Idea: fit simple model to subset of the data

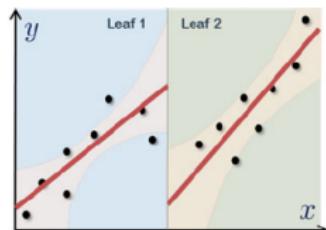


prob. model

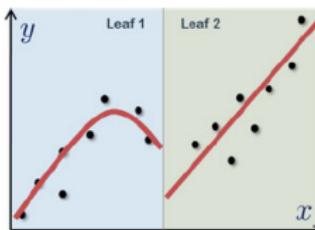
# Regression Trees



Idea: fit simple model to subset of the data

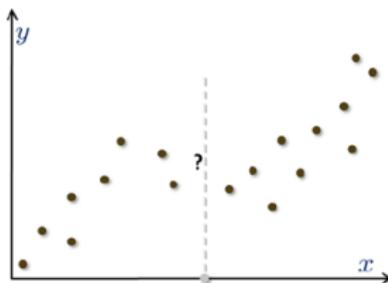


probab. model

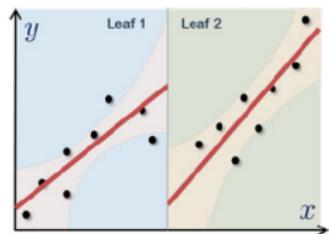


polynomial

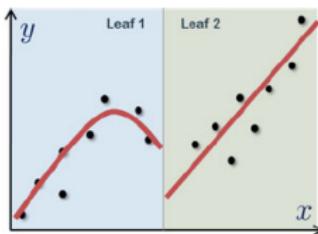
# Regression Trees



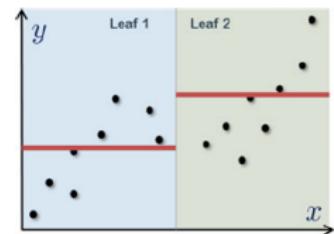
Idea: fit simple model to subset of the data



probab. model

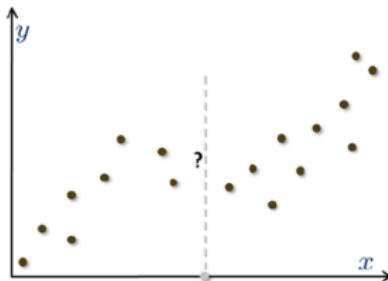


polynomial

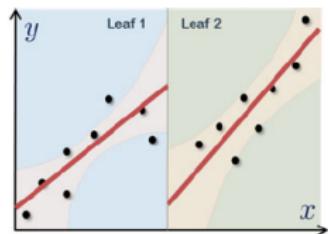


constant (standard)

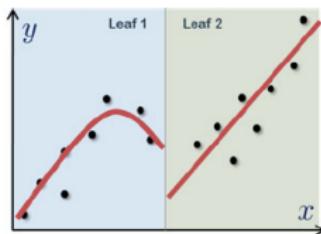
# Regression Trees



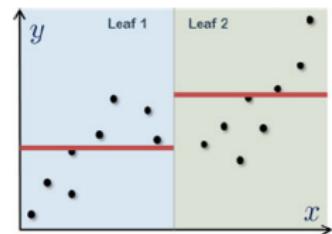
Idea: fit simple model to subset of the data



prob. model



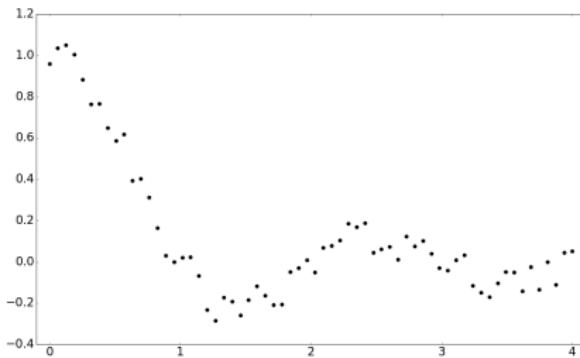
polynomial



constant (standard)

We will only cover the standard case of constant leaf predictions

# How to Split the Data

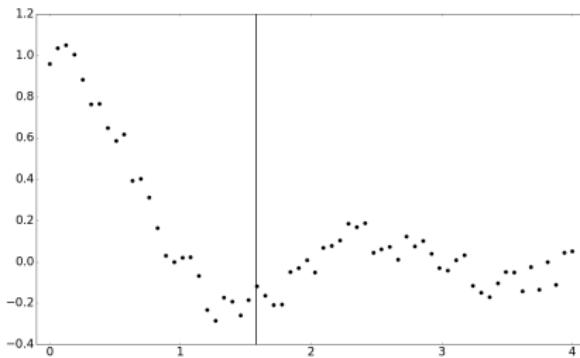


- Constant models in the leafs:  $\hat{f}_{left}$  and  $\hat{f}_{right}$
- Greedily minimize sum of squared errors in the two children

$$x_{\text{split}} \in \arg \min_x \left( \sum_{x_i \leq x} (y_i - \hat{f}_{left})^2 + \sum_{x_i > x} (y_i - \hat{f}_{right})^2 \right)$$

- Recursively split subsets

# How to Split the Data

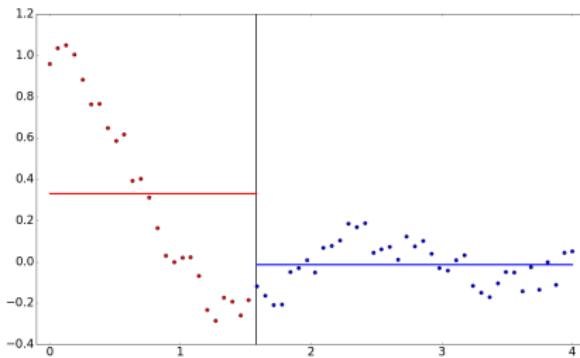


- Constant models in the leafs:  $\hat{f}_{left}$  and  $\hat{f}_{right}$
- Greedily minimize sum of squared errors in the two children

$$x_{\text{split}} \in \arg \min_x \left( \sum_{x_i \leq x} (y_i - \hat{f}_{left})^2 + \sum_{x_i > x} (y_i - \hat{f}_{right})^2 \right)$$

- Recursively split subsets

# How to Split the Data

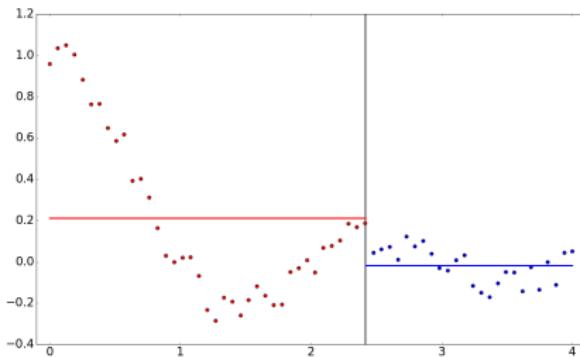


- Constant models in the leafs:  $\hat{f}_{left}$  and  $\hat{f}_{right}$
- Greedily minimize sum of squared errors in the two children

$$x_{\text{split}} \in \arg \min_x \left( \sum_{x_i \leq x} (y_i - \hat{f}_{left})^2 + \sum_{x_i > x} (y_i - \hat{f}_{right})^2 \right)$$

- Recursively split subsets

# How to Split the Data

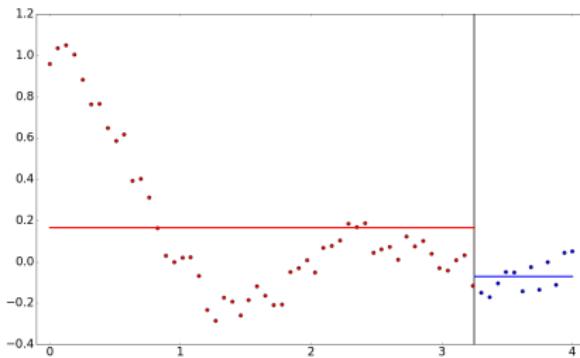


- Constant models in the leafs:  $\hat{f}_{left}$  and  $\hat{f}_{right}$
- Greedily minimize sum of squared errors in the two children

$$x_{\text{split}} \in \arg \min_x \left( \sum_{x_i \leq x} (y_i - \hat{f}_{left})^2 + \sum_{x_i > x} (y_i - \hat{f}_{right})^2 \right)$$

- Recursively split subsets

# How to Split the Data

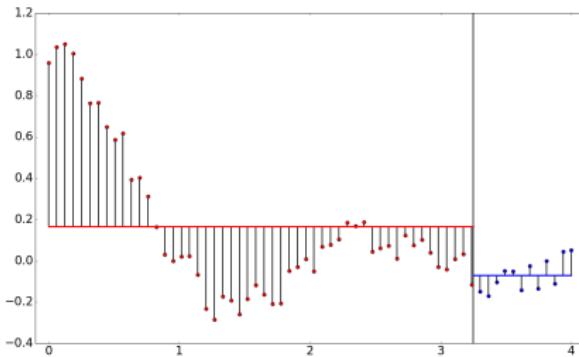


- Constant models in the leafs:  $\hat{f}_{left}$  and  $\hat{f}_{right}$
- Greedily minimize sum of squared errors in the two children

$$x_{\text{split}} \in \arg \min_x \left( \sum_{x_i \leq x} (y_i - \hat{f}_{left})^2 + \sum_{x_i > x} (y_i - \hat{f}_{right})^2 \right)$$

- Recursively split subsets

# How to Split the Data

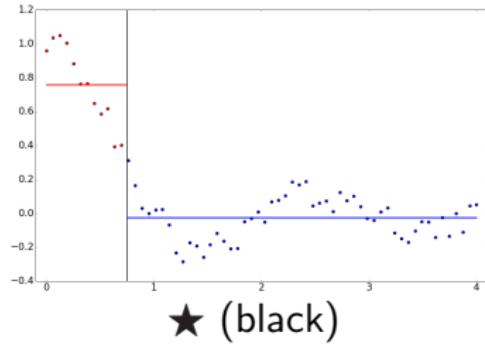


- Constant models in the leafs:  $\hat{f}_{left}$  and  $\hat{f}_{right}$
- Greedily minimize sum of squared errors in the two children

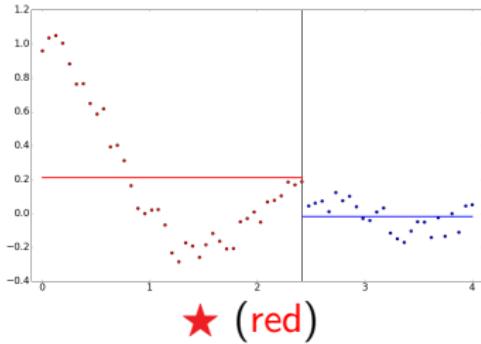
$$x_{\text{split}} \in \arg \min_x \left( \sum_{x_i \leq x} (y_i - \hat{f}_{left})^2 + \sum_{x_i > x} (y_i - \hat{f}_{right})^2 \right)$$

- Recursively split subsets

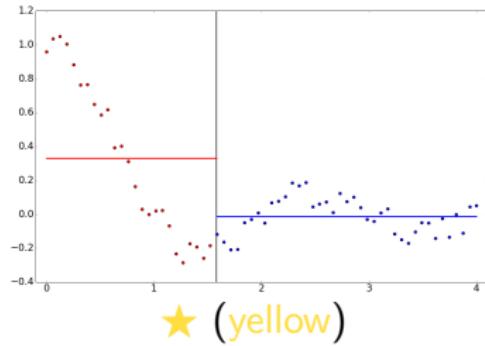
# Let's vote: Which of These Splits is the Best?



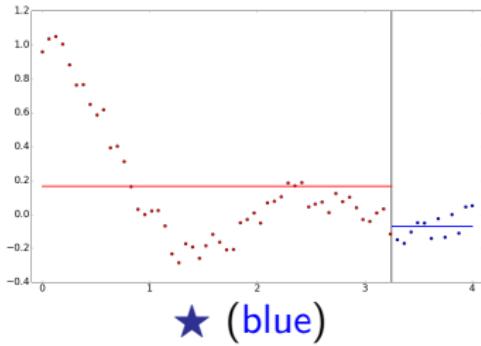
★ (black)



★ (red)

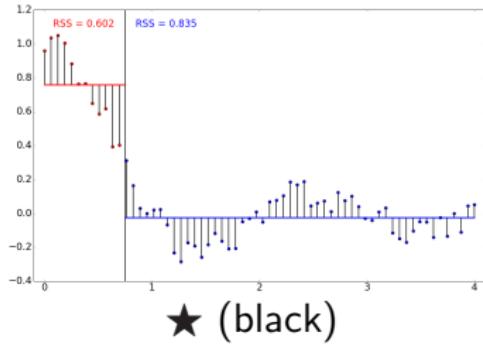


★ (yellow)

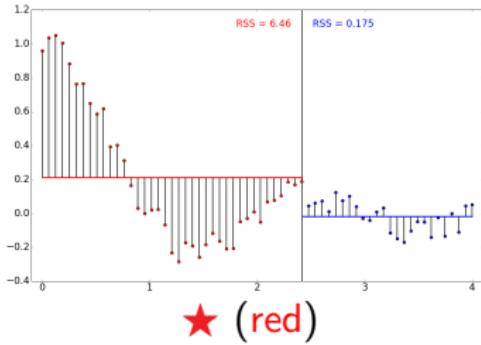


★ (blue)

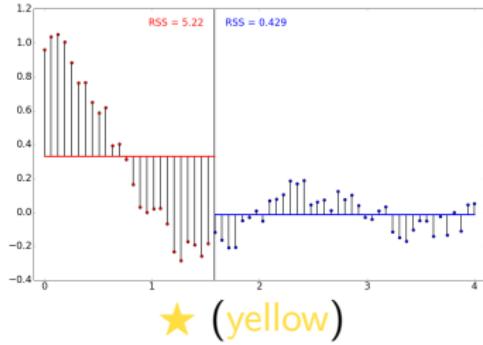
# Let's vote: Which of These Splits is the Best?



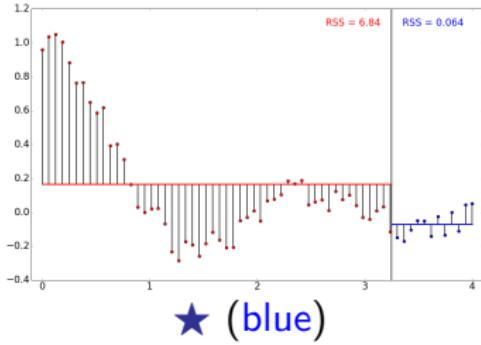
★ (black)



★ (red)



★ (yellow)



★ (blue)

# CART algorithm [Breiman et. al 1984]

## CART = Classification And Regression Trees

- Input:  $\mathbf{X}$ ,  $\mathbf{y}$  (and hyperparameters `max_depth`, `min_leaf`)
- Check whether data should be split further; otherwise return leaf node
- Find best split value for each feature
- Choose best combination of split feature and split value
- Split data into left and right accordingly:  $(\mathbf{X}_l, \mathbf{y}_l)$  and  $(\mathbf{X}_r, \mathbf{y}_r)$
- Save split feature and value, and pointer to two new subtrees to be built recursively:  
$$(\text{CART}(\mathbf{X}_l, \mathbf{y}_l, \text{max\_depth}-1, \text{min\_leaf}),$$
$$\text{CART}(\mathbf{X}_r, \mathbf{y}_r, \text{max\_depth}-1, \text{min\_leaf}))$$

# Visualization of CART Algorithm For Regression: Training

feature1	feature2	feature3	target
false	2	red	3.7
false	2.5	blue	20
true	5.5	red	2.1
false	5.5	blue	25
false	5	red	1.2
true	4.5	green	19
true	4	blue	12
true	3.5	green	17

# Visualization of CART Algorithm For Regression: Training

feature1	feature2	feature3	target
false	2	red	3.7
false	2.5	blue	20
true	5.5	red	2.1
false	5.5	blue	25
false	5	red	1.2
true	4.5	green	19
true	4	blue	12
true	3.5	green	17

$feature_3 \in \{red\}$

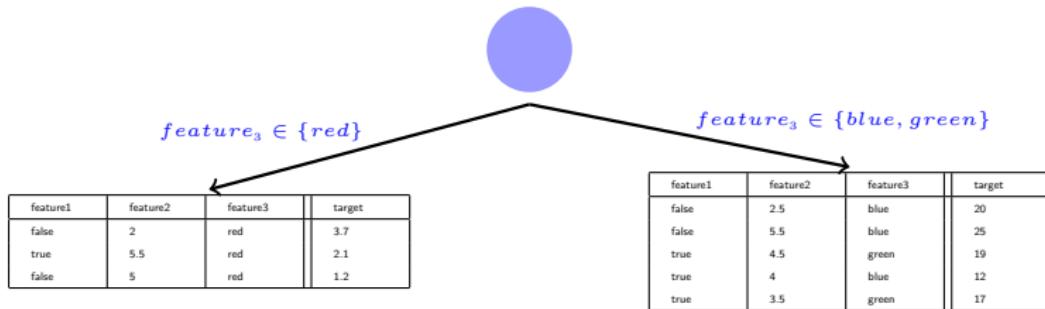
$feature_3 \in \{blue, green\}$

feature1	feature2	feature3	target
false	2	red	3.7
true	5.5	red	2.1
false	5	red	1.2

feature1	feature2	feature3	target
false	2.5	blue	20
false	5.5	blue	25
true	4.5	green	19
true	4	blue	12
true	3.5	green	17

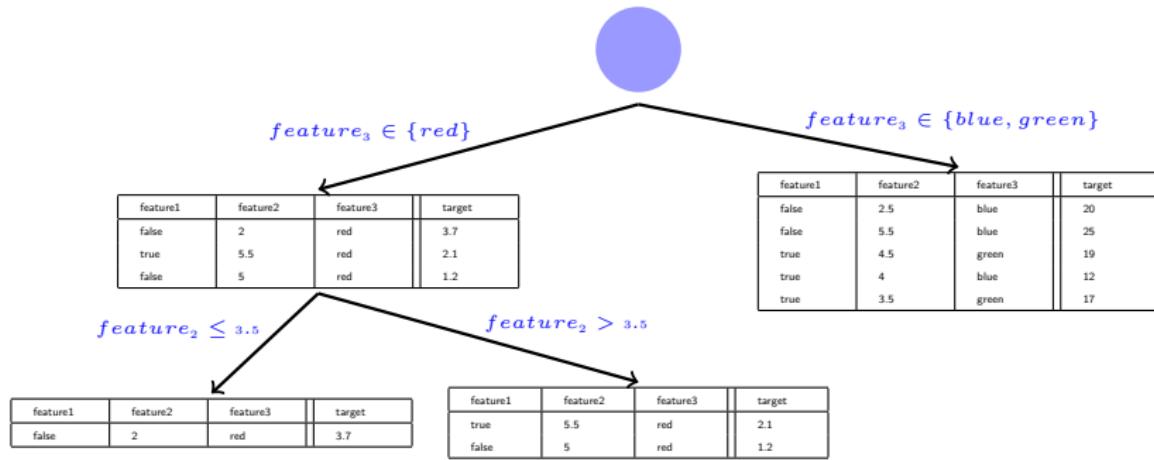
# Visualization of CART Algorithm For Regression: Training

- In each internal node: only store split criterion used



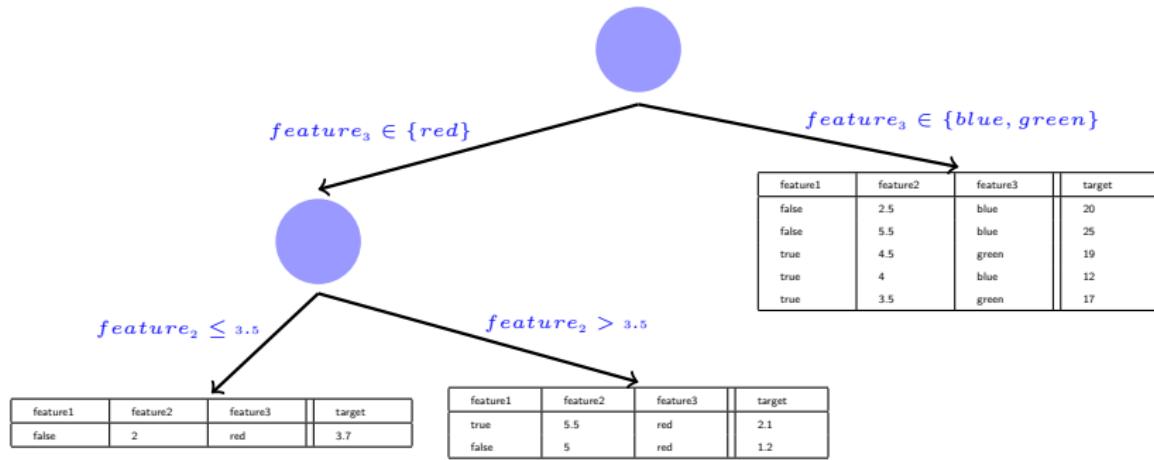
# Visualization of CART Algorithm For Regression: Training

- In each internal node: only store split criterion used



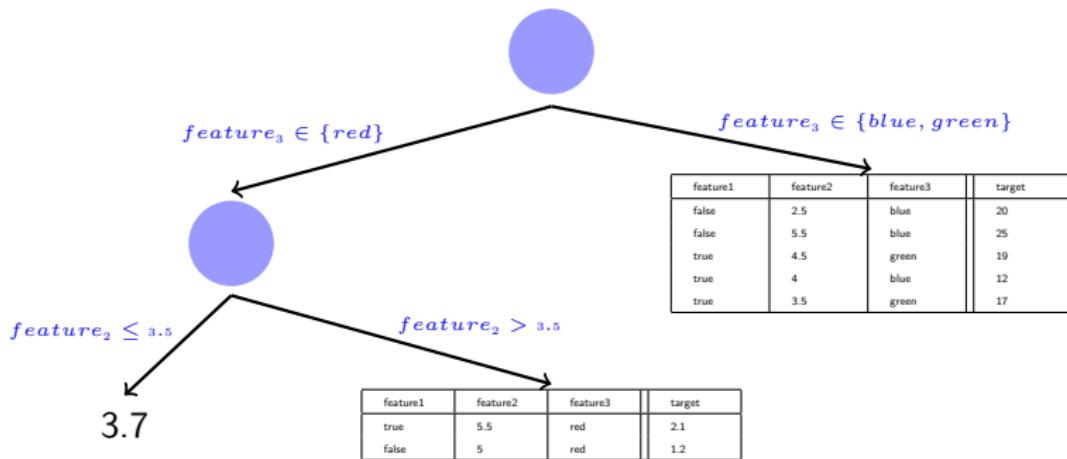
# Visualization of CART Algorithm For Regression: Training

- In each internal node: only store split criterion used



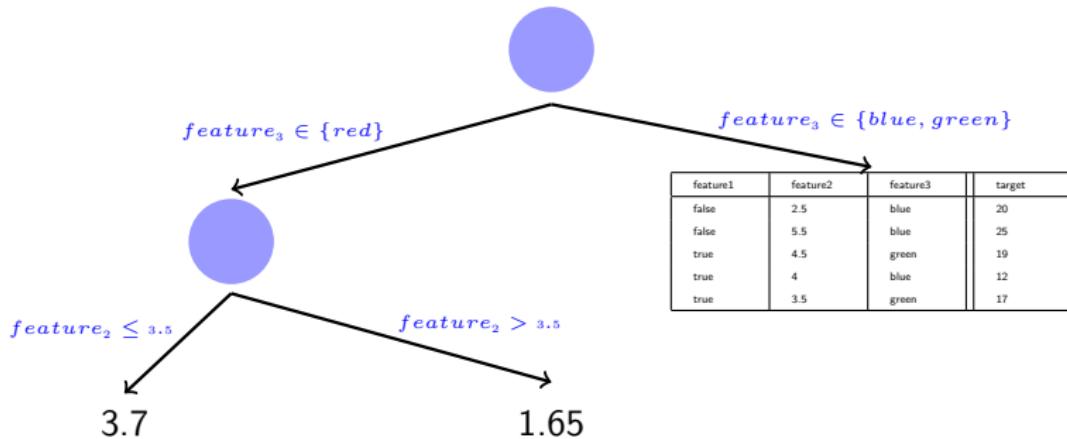
# Visualization of CART Algorithm For Regression: Training

- In each internal node: only store split criterion used
- In each leaf: store mean of targets



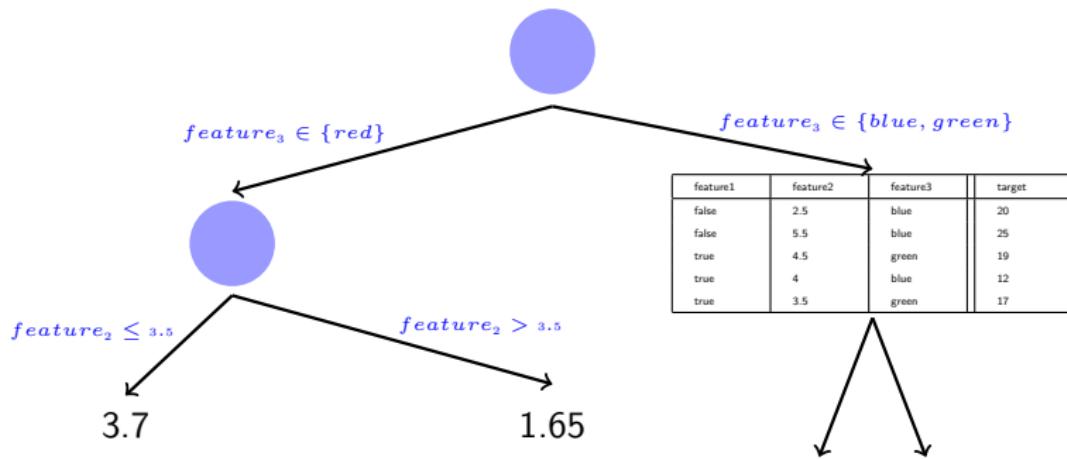
# Visualization of CART Algorithm For Regression: Training

- In each internal node: only store split criterion used
- In each leaf: store mean of targets



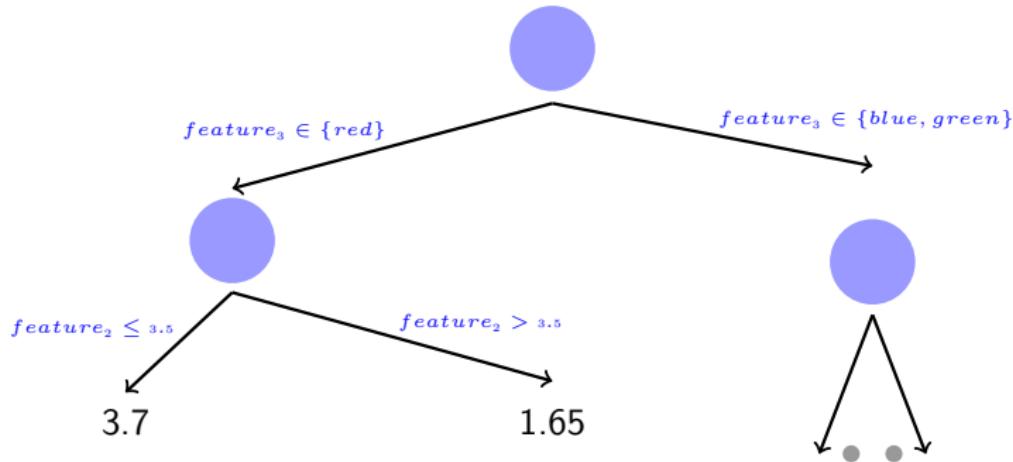
# Visualization of CART Algorithm For Regression: Training

- In each internal node: only store split criterion used
- In each leaf: store mean of targets



# Visualization of CART Algorithm For Regression: Training

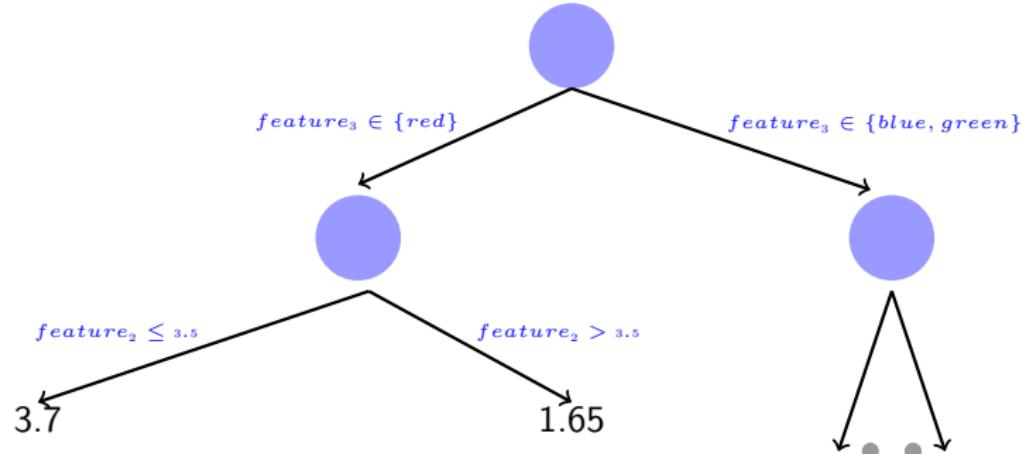
- In each internal node: only store split criterion used
- In each leaf: store mean of targets



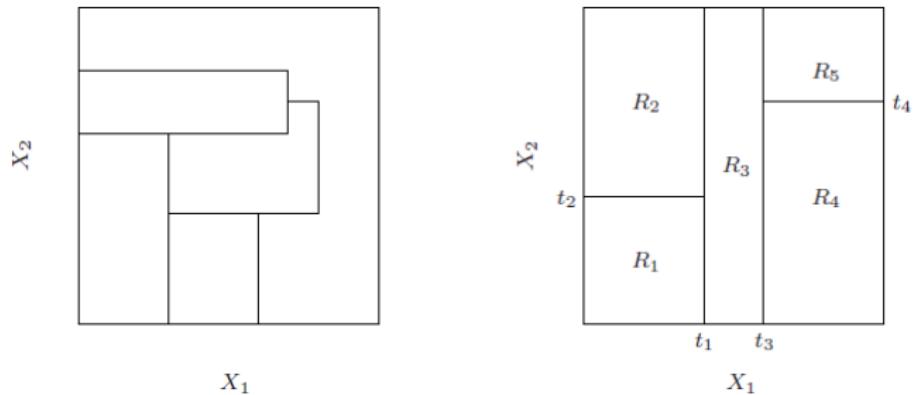
# Visualization of CART Algorithm For Regression: Prediction for New Inputs

E.g  $x_{n+1} = (\text{true}, 4.7, \text{red})$

- Walk down tree, return mean target stored in leaf  $\Rightarrow 1.65$

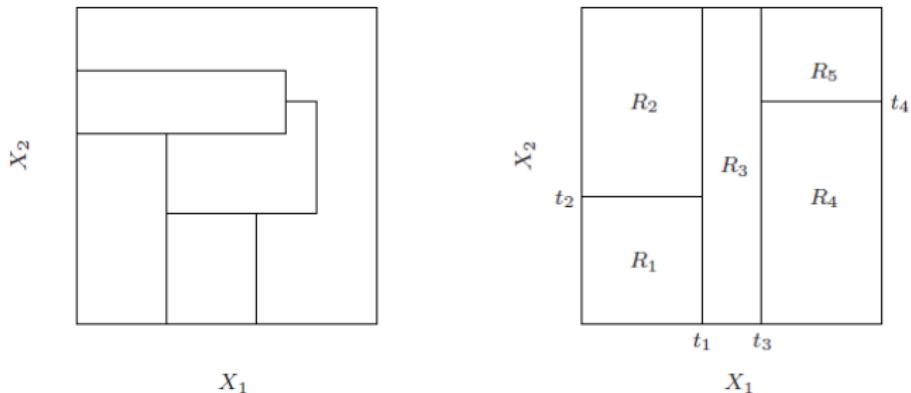


# Hierarchichal Binary Splits



- Hierarchical splits (as on the right side) are easy to represent
  - Splits as in the left figure would be harder

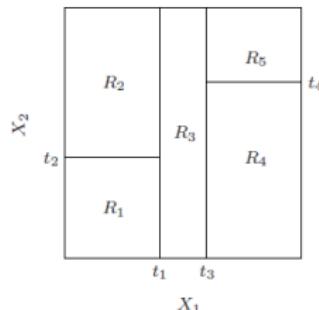
# Hierarchichal Binary Splits



- Hierarchical splits (as on the right side) are easy to represent
  - Splits as in the left figure would be harder
- We could also use  $k$ -ary splits
  - But every  $k$ -ary split can be seen as a sequence of binary splits
  - Binary splits are faster and often yield better predictions

# Formal Notation for Tree Predictions (1/2)

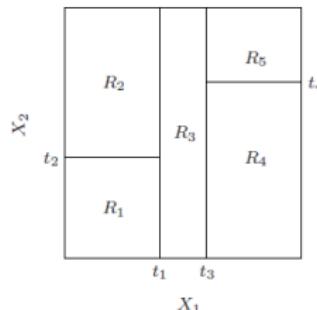
After the intuitive treatment so far, we now write down tree predictions formally.



- Trees partition the input space  $\mathcal{X}$  into regions  $R_1, \dots, R_J$  associated with their leaves
- Each leaf  $j$  has a simple model; here the constant  $\gamma_j$

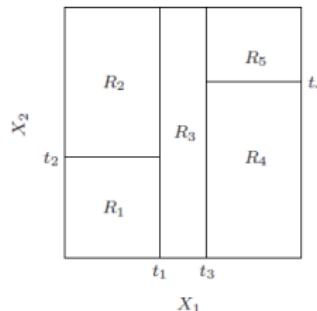
## Formal Notation for Tree Predictions (1/2)

After the intuitive treatment so far, we now write down tree predictions formally.



- Trees partition the input space  $\mathcal{X}$  into regions  $R_1, \dots, R_J$  associated with their leaves
- Each leaf  $j$  has a simple model; here the constant  $\gamma_j$
- Mathematically, a decision/regression tree  $T$  with constant leaf predictions is fully specified by  $\langle R_1, \dots, R_J, \gamma_1, \dots, \gamma_J \rangle$

## Formal Notation for Tree Predictions (2/2)



### Tree Prediction

The prediction of a decision/regression tree with parameters  $\Theta = \langle R_1, \dots, R_J, \gamma_1, \dots, \gamma_J \rangle$  is

$$T(x_i, \Theta) = \sum_{j=1}^J \gamma_j \mathbb{I}(x \in R_j)$$

Here and throughout,  $\mathbb{I}$  is the indicator function

$$\mathbb{I}(a) := \begin{cases} 1 & , \text{ if } a = \text{true} \\ 0 & , \text{ otherwise} \end{cases}$$

# Lecture Overview

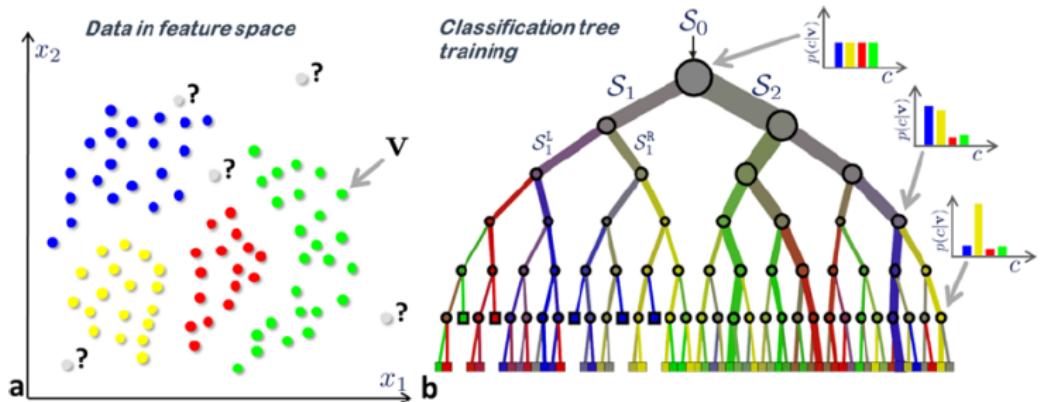
## 1 Decision and Regression Trees

- Regression Trees
- Classification Trees (= Decision Trees)

## 2 Bagging

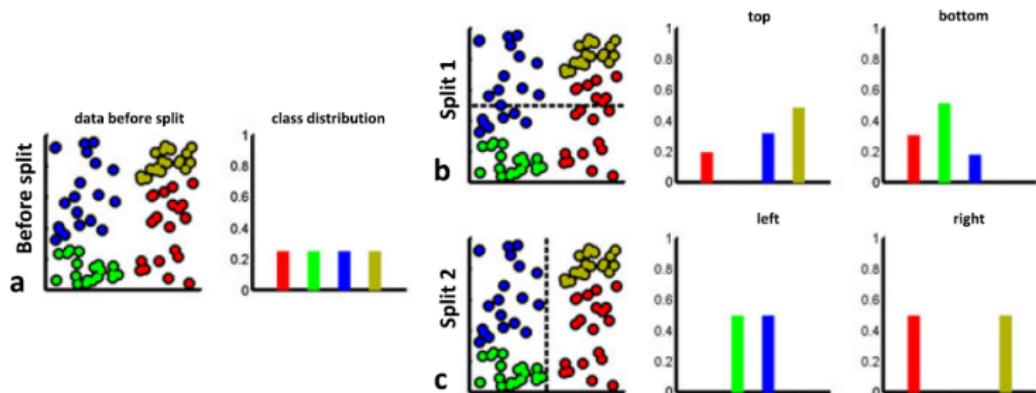
## 3 Random Forests

# Classification Trees (= Decision Trees)



- Conceptually the same as for regression
- Leaf model: majority vote; **probability of data in the leaf**
- Split criterion: Gini index; variance reduction; **information gain**

# Classification Splitting Criteria



Intuitively, which of these splits is better?

- ★ Split 1 (into top and bottom)
- ★ Split 2 (into left and right)

# Information Entropy

## Definition: Entropy

The **entropy** of a discrete random variable  $V$  with  $K$  possible outcomes  $v_k$  of respective probability  $p(v_k)$  (for  $k = 1, \dots, K$ ) is

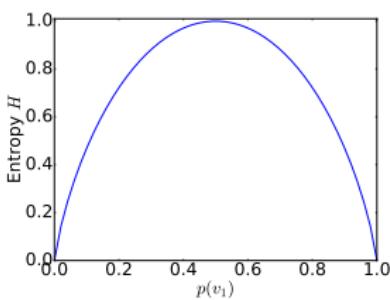
$$H(V) = - \sum_{k=1}^K p(v_k) \log_2 p(v_k)$$

# Information Entropy

## Definition: Entropy

The **entropy** of a discrete random variable  $V$  with  $K$  possible outcomes  $v_k$  of respective probability  $p(v_k)$  (for  $k = 1, \dots, K$ ) is

$$H(V) = - \sum_{k=1}^K p(v_k) \log_2 p(v_k)$$



Examples for a Boolean random variable  $V$  with outcomes  $v_1$  and  $v_2$ :

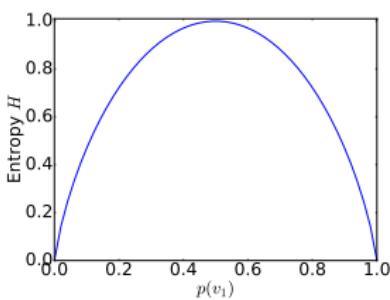
- $p(v_1) = 0.5, p(v_2) = 0.5$ . Then,  
 $H(V) =$

# Information Entropy

## Definition: Entropy

The **entropy** of a discrete random variable  $V$  with  $K$  possible outcomes  $v_k$  of respective probability  $p(v_k)$  (for  $k = 1, \dots, K$ ) is

$$H(V) = - \sum_{k=1}^K p(v_k) \log_2 p(v_k)$$



Examples for a Boolean random variable  $V$  with outcomes  $v_1$  and  $v_2$ :

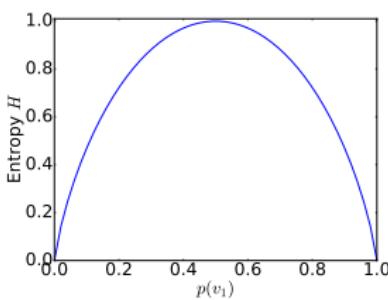
- $p(v_1) = 0.5, p(v_2) = 0.5$ . Then,  
$$H(V) = -0.5 \log_2(0.5) - 0.5 \log_2(0.5) = 1$$

# Information Entropy

## Definition: Entropy

The **entropy** of a discrete random variable  $V$  with  $K$  possible outcomes  $v_k$  of respective probability  $p(v_k)$  (for  $k = 1, \dots, K$ ) is

$$H(V) = - \sum_{k=1}^K p(v_k) \log_2 p(v_k)$$



Examples for a Boolean random variable  $V$  with outcomes  $v_1$  and  $v_2$ :

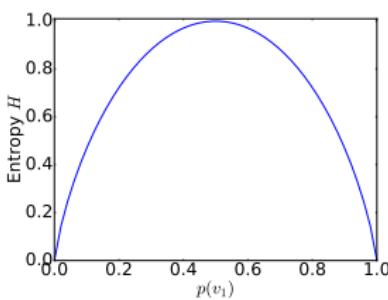
- $p(v_1) = 0.5, p(v_2) = 0.5$ . Then,  
$$H(V) = -0.5 \log_2(0.5) - 0.5 \log_2(0.5) = 1$$
- $p(v_1) = 0, p(v_2) = 1$ . Then,  
$$H(V) =$$

# Information Entropy

## Definition: Entropy

The **entropy** of a discrete random variable  $V$  with  $K$  possible outcomes  $v_k$  of respective probability  $p(v_k)$  (for  $k = 1, \dots, K$ ) is

$$H(V) = - \sum_{k=1}^K p(v_k) \log_2 p(v_k)$$



Examples for a Boolean random variable  $V$  with outcomes  $v_1$  and  $v_2$ :

- $p(v_1) = 0.5, p(v_2) = 0.5$ . Then,  
$$H(V) = -0.5 \log_2(0.5) - 0.5 \log_2(0.5) = 1$$
- $p(v_1) = 0, p(v_2) = 1$ . Then,  
$$H(V) = -\lim_{x \rightarrow 0} x \log_2(x) - 1 \log_2(1) = 0$$

# Prominent Splitting Criterion: Information Gain

## Definition: Information Gain

Let  $V$  denote a random variable with the empirical class distribution  $p$  of the  $N$  data points at the current node.

# Prominent Splitting Criterion: Information Gain

## Definition: Information Gain

Let  $V$  denote a random variable with the empirical class distribution  $p$  of the  $N$  data points at the current node. Let a split  $s$  result in two child nodes with  $N_l$  and  $N_r$  data points and empirical class distributions  $p_l$  and  $p_r$ , and let  $V_l$  and  $V_r$  denote random variables with these probability distributions, respectively.

# Prominent Splitting Criterion: Information Gain

## Definition: Information Gain

Let  $V$  denote a random variable with the empirical class distribution  $p$  of the  $N$  data points at the current node. Let a split  $s$  result in two child nodes with  $N_l$  and  $N_r$  data points and empirical class distributions  $p_l$  and  $p_r$ , and let  $V_l$  and  $V_r$  denote random variables with these probability distributions, respectively. Then, the **information gain** achieved by split  $s$  is:

$$I = N \cdot H(V) - N_l \cdot H(V_l) - N_r \cdot H(V_r)$$

# Prominent Splitting Criterion: Information Gain

## Definition: Information Gain

Let  $V$  denote a random variable with the empirical class distribution  $p$  of the  $N$  data points at the current node. Let a split  $s$  result in two child nodes with  $N_l$  and  $N_r$  data points and empirical class distributions  $p_l$  and  $p_r$ , and let  $V_l$  and  $V_r$  denote random variables with these probability distributions, respectively. Then, the **information gain** achieved by split  $s$  is:

$$\begin{aligned} I &= N \cdot H(V) - N_l \cdot H(V_l) - N_r \cdot H(V_r) \\ &= -N \sum_{k=1}^K p(v_k) \log_2(p(v_k)) + N_l \sum_{k=1}^K p_l(v_k) \log_2(p_l(v_k)) + N_r \sum_{k=1}^K p_r(v_k) \log_2(p_r(v_k)) \end{aligned}$$

# Prominent Splitting Criterion: Information Gain

## Definition: Information Gain

Let  $V$  denote a random variable with the empirical class distribution  $p$  of the  $N$  data points at the current node. Let a split  $s$  result in two child nodes with  $N_l$  and  $N_r$  data points and empirical class distributions  $p_l$  and  $p_r$ , and let  $V_l$  and  $V_r$  denote random variables with these probability distributions, respectively. Then, the **information gain** achieved by split  $s$  is:

$$\begin{aligned} I &= N \cdot H(V) - N_l \cdot H(V_l) - N_r \cdot H(V_r) \\ &= -N \sum_{k=1}^K p(v_k) \log_2(p(v_k)) + N_l \sum_{k=1}^K p_l(v_k) \log_2(p_l(v_k)) + N_r \sum_{k=1}^K p_r(v_k) \log_2(p_r(v_k)) \end{aligned}$$

We want to maximize this information gain.

# Information Gain: Example

$$I = N \cdot H(V) - N_l \cdot H(V_l) - N_r \cdot H(V_r)$$
$$= -N \sum_{k=1}^K p(v_k) \log_2(p(v_k)) + N_l \sum_{k=1}^K p_l(v_k) \log_2(p_l(v_k)) + N_r \sum_{k=1}^K p_r(v_k) \log_2(p_r(v_k))$$

- Example: Splitting 8 data points ( $4 \times \text{false}$ ,  $4 \times \text{true}$ ).
  - Split 1: left:  $4 \times \text{true}$ ; right:  $4 \times \text{false}$ 
    - $N \cdot H(V) =$
    - $N_l \cdot H(V_l) =$
    - $N_r \cdot H(V_r) =$
  - Split 2: left:  $2 \times \text{true}$ ,  $2 \times \text{false}$ ; right:  $2 \times \text{true}$ ,  $2 \times \text{false}$ 
    - $N \cdot H(V) =$
    - $N_l \cdot H(V_l) =$
    - $N_r \cdot H(V_r) =$

# Information Gain: Example

$$I = N \cdot H(V) - N_l \cdot H(V_l) - N_r \cdot H(V_r)$$
$$= -N \sum_{k=1}^K p(v_k) \log_2(p(v_k)) + N_l \sum_{k=1}^K p_l(v_k) \log_2(p_l(v_k)) + N_r \sum_{k=1}^K p_r(v_k) \log_2(p_r(v_k))$$

- Example: Splitting 8 data points ( $4 \times \text{false}$ ,  $4 \times \text{true}$ ).
  - Split 1: left:  $4 \times \text{true}$ ; right:  $4 \times \text{false}$ 
    - $N \cdot H(V) = 8 \cdot 1 = 8$
    - $N_l \cdot H(V_l) =$
    - $N_r \cdot H(V_r) =$
  - Split 2: left:  $2 \times \text{true}$ ,  $2 \times \text{false}$ ; right:  $2 \times \text{true}$ ,  $2 \times \text{false}$ 
    - $N \cdot H(V) =$
    - $N_l \cdot H(V_l) =$
    - $N_r \cdot H(V_r) =$

# Information Gain: Example

$$I = N \cdot H(V) - N_l \cdot H(V_l) - N_r \cdot H(V_r)$$
$$= -N \sum_{k=1}^K p(v_k) \log_2(p(v_k)) + N_l \sum_{k=1}^K p_l(v_k) \log_2(p_l(v_k)) + N_r \sum_{k=1}^K p_r(v_k) \log_2(p_r(v_k))$$

- Example: Splitting 8 data points ( $4 \times \text{false}$ ,  $4 \times \text{true}$ ).
  - Split 1: left:  $4 \times \text{true}$ ; right:  $4 \times \text{false}$ 
    - $N \cdot H(V) = 8 \cdot 1 = 8$
    - $N_l \cdot H(V_l) = 4 \cdot 0 = 0$
    - $N_r \cdot H(V_r) =$
  - Split 2: left:  $2 \times \text{true}$ ,  $2 \times \text{false}$ ; right:  $2 \times \text{true}$ ,  $2 \times \text{false}$ 
    - $N \cdot H(V) =$
    - $N_l \cdot H(V_l) =$
    - $N_r \cdot H(V_r) =$

# Information Gain: Example

$$I = N \cdot H(V) - N_l \cdot H(V_l) - N_r \cdot H(V_r)$$
$$= -N \sum_{k=1}^K p(v_k) \log_2(p(v_k)) + N_l \sum_{k=1}^K p_l(v_k) \log_2(p_l(v_k)) + N_r \sum_{k=1}^K p_r(v_k) \log_2(p_r(v_k))$$

- Example: Splitting 8 data points ( $4 \times \text{false}$ ,  $4 \times \text{true}$ ).
  - Split 1: left:  $4 \times \text{true}$ ; right:  $4 \times \text{false}$ 
    - $N \cdot H(V) = 8 \cdot 1 = 8$
    - $N_l \cdot H(V_l) = 4 \cdot 0 = 0$
    - $N_r \cdot H(V_r) = 4 \cdot 0 = 0$
  - Split 2: left:  $2 \times \text{true}$ ,  $2 \times \text{false}$ ; right:  $2 \times \text{true}$ ,  $2 \times \text{false}$ 
    - $N \cdot H(V) =$
    - $N_l \cdot H(V_l) =$
    - $N_r \cdot H(V_r) =$

# Information Gain: Example

$$I = N \cdot H(V) - N_l \cdot H(V_l) - N_r \cdot H(V_r)$$
$$= -N \sum_{k=1}^K p(v_k) \log_2(p(v_k)) + N_l \sum_{k=1}^K p_l(v_k) \log_2(p_l(v_k)) + N_r \sum_{k=1}^K p_r(v_k) \log_2(p_r(v_k))$$

- Example: Splitting 8 data points ( $4 \times \text{false}$ ,  $4 \times \text{true}$ ).
  - Split 1: left:  $4 \times \text{true}$ ; right:  $4 \times \text{false}$ 
    - $N \cdot H(V) = 8 \cdot 1 = 8$
    - $N_l \cdot H(V_l) = 4 \cdot 0 = 0$
    - $N_r \cdot H(V_r) = 4 \cdot 0 = 0$
  - Split 2: left:  $2 \times \text{true}$ ,  $2 \times \text{false}$ ; right:  $2 \times \text{true}$ ,  $2 \times \text{false}$ 
    - $N \cdot H(V) = 8 \cdot 1 = 8$
    - $N_l \cdot H(V_l) =$
    - $N_r \cdot H(V_r) =$

# Information Gain: Example

$$I = N \cdot H(V) - N_l \cdot H(V_l) - N_r \cdot H(V_r)$$
$$= -N \sum_{k=1}^K p(v_k) \log_2(p(v_k)) + N_l \sum_{k=1}^K p_l(v_k) \log_2(p_l(v_k)) + N_r \sum_{k=1}^K p_r(v_k) \log_2(p_r(v_k))$$

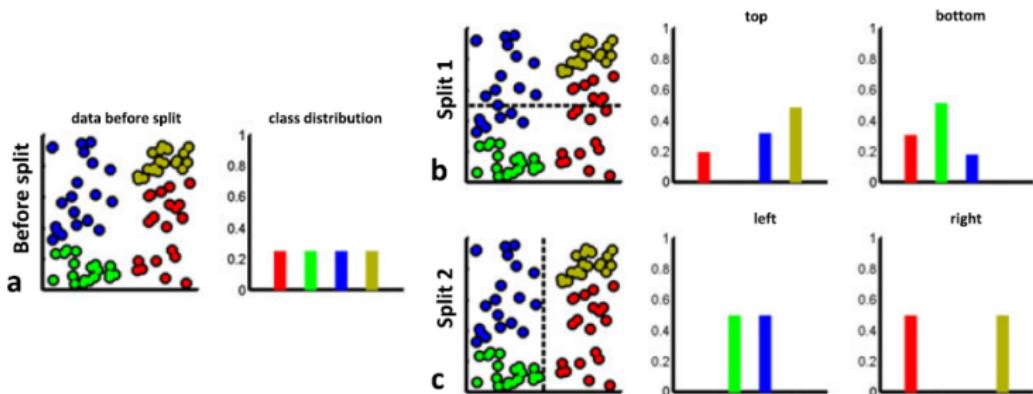
- Example: Splitting 8 data points ( $4 \times \text{false}$ ,  $4 \times \text{true}$ ).
  - Split 1: left:  $4 \times \text{true}$ ; right:  $4 \times \text{false}$ 
    - $N \cdot H(V) = 8 \cdot 1 = 8$
    - $N_l \cdot H(V_l) = 4 \cdot 0 = 0$
    - $N_r \cdot H(V_r) = 4 \cdot 0 = 0$
  - Split 2: left:  $2 \times \text{true}$ ,  $2 \times \text{false}$ ; right:  $2 \times \text{true}$ ,  $2 \times \text{false}$ 
    - $N \cdot H(V) = 8 \cdot 1 = 8$
    - $N_l \cdot H(V_l) = 4 \cdot 1 = 4$
    - $N_r \cdot H(V_r) =$

# Information Gain: Example

$$I = N \cdot H(V) - N_l \cdot H(V_l) - N_r \cdot H(V_r)$$
$$= -N \sum_{k=1}^K p(v_k) \log_2(p(v_k)) + N_l \sum_{k=1}^K p_l(v_k) \log_2(p_l(v_k)) + N_r \sum_{k=1}^K p_r(v_k) \log_2(p_r(v_k))$$

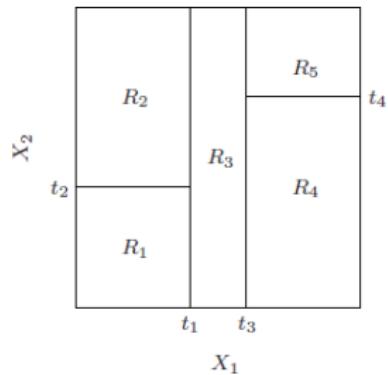
- Example: Splitting 8 data points ( $4 \times \text{false}$ ,  $4 \times \text{true}$ ).
  - Split 1: left:  $4 \times \text{true}$ ; right:  $4 \times \text{false}$ 
    - $N \cdot H(V) = 8 \cdot 1 = 8$
    - $N_l \cdot H(V_l) = 4 \cdot 0 = 0$
    - $N_r \cdot H(V_r) = 4 \cdot 0 = 0$
  - Split 2: left:  $2 \times \text{true}$ ,  $2 \times \text{false}$ ; right:  $2 \times \text{true}$ ,  $2 \times \text{false}$ 
    - $N \cdot H(V) = 8 \cdot 1 = 8$
    - $N_l \cdot H(V_l) = 4 \cdot 1 = 4$
    - $N_r \cdot H(V_r) = 4 \cdot 1 = 4$

# Information Gain: Example



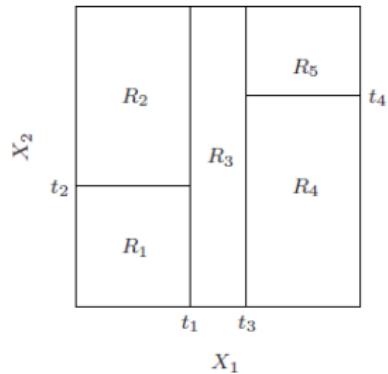
- We previously said split 2 (into left and right) is intuitively better than split 1 (into top and bottom)
- Information gain quantifies this

# Splits Along Several Dimensions

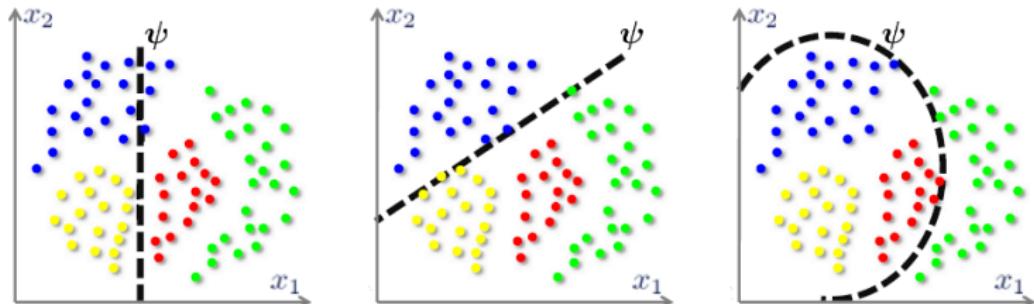


Axis-aligned split are standard, but more complex splits are possible

# Splits Along Several Dimensions



Axis-aligned split are standard, but more complex splits are possible



# Pros and Cons of Decision and Regression Trees

- + Flexible framework with exchangeable components:  
splitting criterion, leaf model, type of split
- + Interpretability
- + Handle categorical input values natively
- + Handle unimportant features well
- + Scalable for large datasets
- Tend to overfit
- Deterministic, i.e. not suitable for some ensemble methods

# Hyperparameters of Decision and Regression Trees

Regression and decision trees have several hyperparameters:

- Minimum number of samples in a leaf (`min_leaf`)
- Maximal depth of the tree (`max_depth`)
- Total number of nodes
- Leaf model (weak learner; here constant)
- Split criterion

Assignment 8 explores some aspects of their influence on the predictive quality.

# Bias and Variance of Trees

- Facts about tree-based models
  - Trees are very expressive models
  - When you slightly change the data, you might get a very different tree

# Bias and Variance of Trees

- Facts about tree-based models
  - Trees are very expressive models
  - When you slightly change the data, you might get a very different tree
- Using these facts, please choose the right answer:
  - ★ Trees are a high-variance model.
  - ★ Trees are a low-variance model.

# Bias and Variance of Trees

- Facts about tree-based models
  - Trees are very expressive models
  - When you slightly change the data, you might get a very different tree
- Using these facts, please choose the right answer:
  - ★ Trees are a high-variance model.
  - ★ Trees are a low-variance model.
- Using these facts, please choose the right answer:
  - ★ Trees are a high-bias model.
  - ★ Trees are a low-bias model.

## Reminder: Bias and Variance

$$\mathbb{E}_{\mathcal{D}}[\{\hat{h}(\mathbf{x}; \mathcal{D}) - h^*(\mathbf{x})\}^2] = \underbrace{\{\bar{h}(\mathbf{x}) - h^*(\mathbf{x})\}^2}_{\text{(bias)}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[\{\hat{h}(\mathbf{x}; \mathcal{D}) - \bar{h}(\mathbf{x})\}^2]}_{\text{variance}}$$

expected squared deviation from best prediction =  $(\text{bias})^2 + \text{variance}$

- $\hat{h}(\mathbf{x}; \mathcal{D})$  is the prediction of the model that fits data  $\mathcal{D}$  best
- $h^*(\mathbf{x})$  is the true best (unknown) prediction
- $\bar{h}(\mathbf{x}) = \mathbb{E}_{\mathcal{D}}[\hat{h}(\mathbf{x}; \mathcal{D})]$  is the average model prediction  
(average of models trained on data sets  $\mathcal{D}$  from a data distribution)

# Computational Complexity of Regression and Decision Trees

- $N$  data points of dimensionality  $D$ , axis-aligned splits
- Finding the best split value for a given feature (pre-sorted):  $O(N)$ 
  - Amortized analysis:  $O(N)$  to initialize all data points to the left child,  $O(1)$  for moving one data point from left to right at a time

# Computational Complexity of Regression and Decision Trees

- $N$  data points of dimensionality  $D$ , axis-aligned splits
- Finding the best split value for a given feature (pre-sorted):  $O(N)$ 
  - Amortized analysis:  $O(N)$  to initialize all data points to the left child,  $O(1)$  for moving one data point from left to right at a time
- Finding best split point at root:  $O(DN)$
- Let  $T(N)$  denote the time we pay for a complete subtree with  $N$  data points; this has a part due to the cost at the root and parts due to the 2 smaller child subtrees

# Computational Complexity of Regression and Decision Trees

- $N$  data points of dimensionality  $D$ , axis-aligned splits
- Finding the best split value for a given feature (pre-sorted):  $O(N)$ 
  - Amortized analysis:  $O(N)$  to initialize all data points to the left child,  $O(1)$  for moving one data point from left to right at a time
- Finding best split point at root:  $O(DN)$
- Let  $T(N)$  denote the time we pay for a complete subtree with  $N$  data points; this has a part due to the cost at the root and parts due to the 2 smaller child subtrees
- Best case: balanced trees
  - Fitting:  $T(N) = O(DN) + 2T(N/2)$

# Computational Complexity of Regression and Decision Trees

- $N$  data points of dimensionality  $D$ , axis-aligned splits
- Finding the best split value for a given feature (pre-sorted):  $O(N)$ 
  - Amortized analysis:  $O(N)$  to initialize all data points to the left child,  $O(1)$  for moving one data point from left to right at a time
- Finding best split point at root:  $O(DN)$
- Let  $T(N)$  denote the time we pay for a complete subtree with  $N$  data points; this has a part due to the cost at the root and parts due to the 2 smaller child subtrees
- Best case: balanced trees
  - Fitting:  $T(N) = O(DN) + 2T(N/2)$
  - This leads to  $O(DN \log N)$  since we pay  $O(D \cdot N)$  at each of  $\log N$  levels

# Computational Complexity of Regression and Decision Trees

- $N$  data points of dimensionality  $D$ , axis-aligned splits
- Finding the best split value for a given feature (pre-sorted):  $O(N)$ 
  - Amortized analysis:  $O(N)$  to initialize all data points to the left child,  $O(1)$  for moving one data point from left to right at a time
- Finding best split point at root:  $O(DN)$
- Let  $T(N)$  denote the time we pay for a complete subtree with  $N$  data points; this has a part due to the cost at the root and parts due to the 2 smaller child subtrees
- Best case: balanced trees
  - Fitting:  $T(N) = O(DN) + 2T(N/2)$ 
    - This leads to  $O(DN \log N)$  since we pay  $O(D \cdot N)$  at each of  $\log N$  levels
  - Prediction:  $O(\log N)$

# Computational Complexity of Regression and Decision Trees

- $N$  data points of dimensionality  $D$ , axis-aligned splits
- Finding the best split value for a given feature (pre-sorted):  $O(N)$ 
  - Amortized analysis:  $O(N)$  to initialize all data points left to the left child,  $O(1)$  for moving one data point from left to right at a time
- Finding best split point at root:  $O(DN)$
- Let  $T(N)$  denote the time we pay for a complete subtree with  $N$  data points; this has a part due to the cost at the root and parts due to the 2 smaller child subtrees
- Worst case: splitting off one data point at a time
  - Fitting:  $T(N) = O(DN) + T(N - 1)$ ; this leads to  $O(DN^2)$
  - Prediction:  $O(N)$

# Lecture Overview

## 1 Decision and Regression Trees

- Regression Trees
- Classification Trees (= Decision Trees)

## 2 Bagging

## 3 Random Forests

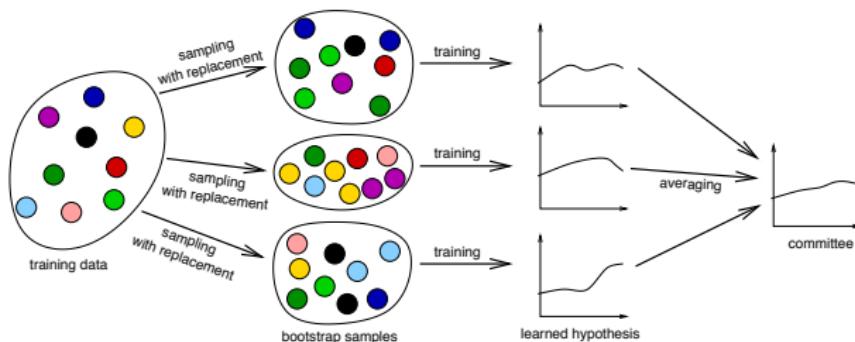
# Bagging is a Committee / Ensemble Approach

- A single expert may fail
- A committee of experts is more likely to get it right (if experts are experienced and diverse)



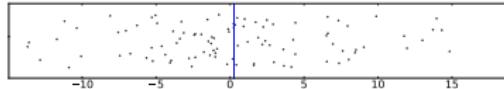
# Bagging [Breiman, 1996]

- Train  $N$  models on **bootstrap samples** of the training data
- For each model, data is drawn randomly with replacements
- **Average** output of all models (**bagging = bootstrap aggregation**)

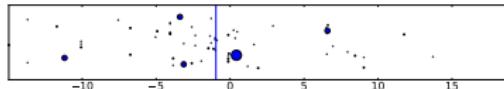
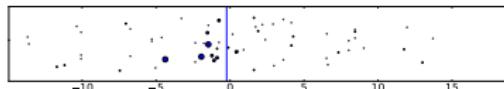
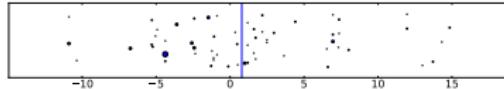


# Bootstrapping

original data

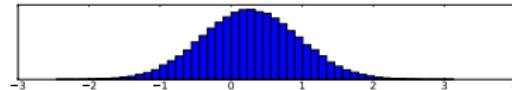


resampled data



⋮

histogram of the means: bootstrap distribution of the mean



# Why Does Bagging Work?

- Bagging
  - Train  $N$  models on **bootstrap samples** of the training data
  - For each model, data is drawn randomly with replacements
  - Average output of all models (**bagging = bootstrap aggregation**)

Discuss with your neighbor first.  (2 minutes)

- Voting question 1: A bagged estimator has

- ★ higher variance
- ★ lower variance

than the individual models it bags.

- Voting question 2: A bagged estimator has

- ★ higher bias
- ★ lower bias

than the individual models it bags.

## Reminder: Bias and Variance

$$\mathbb{E}_{\mathcal{D}}[\{\hat{h}(\mathbf{x}; \mathcal{D}) - h^*(\mathbf{x})\}^2] = \underbrace{\{\bar{h}(\mathbf{x}) - h^*(\mathbf{x})\}^2}_{\text{(bias)}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}[\{\hat{h}(\mathbf{x}; \mathcal{D}) - \bar{h}(\mathbf{x})\}^2]}_{\text{variance}}$$

expected squared deviation from best prediction =  $(\text{bias})^2 + \text{variance}$

- $\hat{h}(\mathbf{x}; \mathcal{D})$  is the prediction of the model that fits data  $\mathcal{D}$  best
- $h^*(\mathbf{x})$  is the true best (unknown) prediction
- $\bar{h}(\mathbf{x}) = \mathbb{E}_{\mathcal{D}}[\hat{h}(\mathbf{x}; \mathcal{D})]$  is the average model prediction  
(average of models trained on data sets  $\mathcal{D}$  from a data distribution)

# Lecture Overview

## 1 Decision and Regression Trees

- Regression Trees
- Classification Trees (= Decision Trees)

## 2 Bagging

## 3 Random Forests

# Why Does Bagging Work Well for Trees?

- Theoretical result [Breiman 2001]:
  - Ensemble error depends on (1) **strength** of individual models and (2) the degree to which the models' errors are **uncorrelated**.

# Why Does Bagging Work Well for Trees?

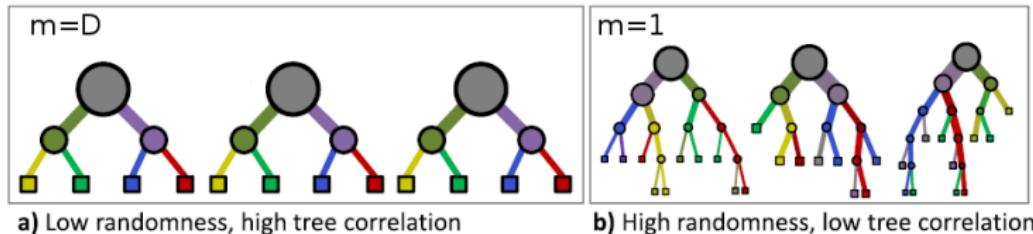
- Theoretical result [Breiman 2001]:
  - Ensemble error depends on (1) **strength** of individual models and (2) the degree to which the models' errors are **uncorrelated**.
- We thus want:
  - Individual models that still work (quite) well
  - Quite different models

# Why Does Bagging Work Well for Trees?

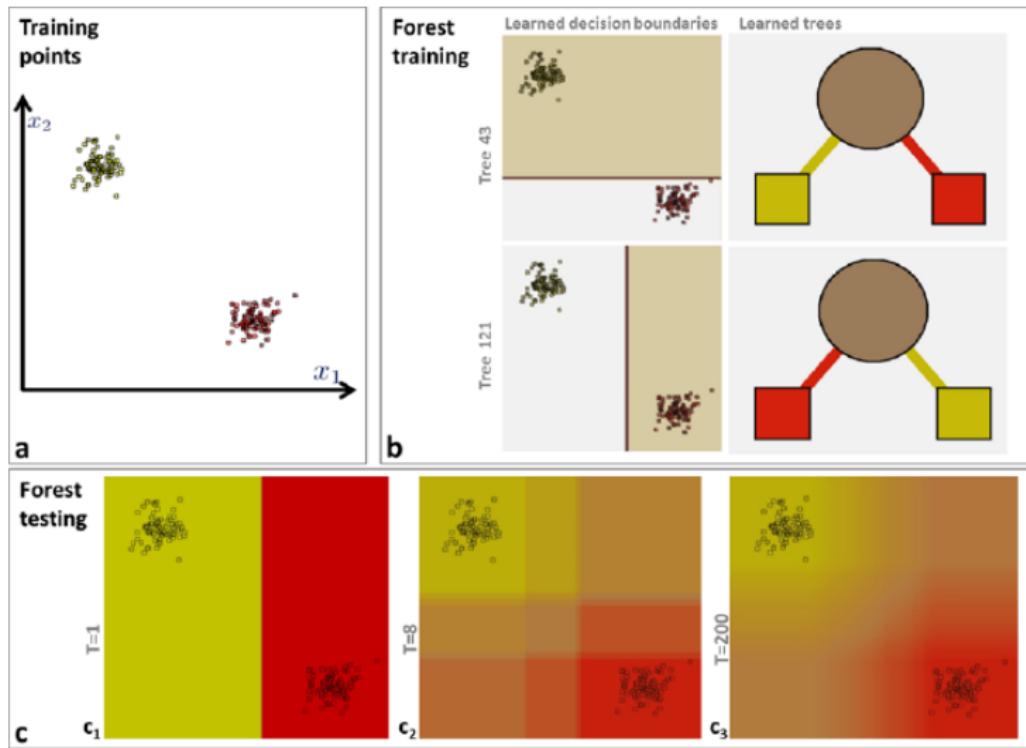
- Theoretical result [Breiman 2001]:
  - Ensemble error depends on (1) **strength** of individual models and (2) the degree to which the models' errors are **uncorrelated**.
- We thus want:
  - Individual models that still work (quite) well
  - Quite different models
- **Randomized trees** give us these properties. We can randomize in many ways:
  - best split using a random subset of  $m \leq D$  features
  - splitting using best out of a fixed number of random splits
  - training on bootstrap-samples of the data

# Why Does Bagging Work Well for Trees?

- Theoretical result [Breiman 2001]:
  - Ensemble error depends on (1) **strength** of individual models and (2) the degree to which the models' errors are **uncorrelated**.
- We thus want:
  - Individual models that still work (quite) well
  - Quite different models
- Randomized trees give us these properties. We can randomize in many ways:
  - best split using a random subset of  $m \leq D$  features
  - splitting using best out of a fixed number of random splits
  - training on bootstrap-samples of the data



# Visualization of Random Forests and Their Predictions



# Algorithm to Grow a Random Forest

---

**Algorithm 15.1** *Random Forest for Regression or Classification.*

---

1. For  $b = 1$  to  $B$ :
  - (a) Draw a bootstrap sample  $\mathbf{Z}^*$  of size  $N$  from the training data.
  - (b) Grow a random-forest tree  $T_b$  to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size  $n_{min}$  is reached.
    - i. Select  $m$  variables at random from the  $p$  variables.
    - ii. Pick the best variable/split-point among the  $m$ .
    - iii. Split the node into two daughter nodes.
2. Output the ensemble of trees  $\{T_b\}_1^B$ .

To make a prediction at a new point  $x$ :

$$\text{Regression: } \hat{f}_{\text{rf}}^B(x) = \frac{1}{B} \sum_{b=1}^B T_b(x).$$

*Classification:* Let  $\hat{C}_b(x)$  be the class prediction of the  $b$ th random-forest tree. Then  $\hat{C}_{\text{rf}}^B(x) = \text{majority vote } \{\hat{C}_b(x)\}_1^B$ .

---

from [Hastie, Tibshirani and Friedman]

# Computational Complexity of Random Forests

- $N$  data points of dimensionality  $D$ , axis-aligned splits
- Picking  $m$  variables at random at each split;  $B$  trees

# Computational Complexity of Random Forests

- $N$  data points of dimensionality  $D$ , axis-aligned splits
- Picking  $m$  variables at random at each split;  $B$  trees
- Best case: balanced trees
  - Fitting:  $O(BmN \log N)$

# Computational Complexity of Random Forests

- $N$  data points of dimensionality  $D$ , axis-aligned splits
- Picking  $m$  variables at random at each split;  $B$  trees
- Best case: balanced trees
  - Fitting:  $O(BmN \log N)$
  - Prediction:  $O(B \log N)$

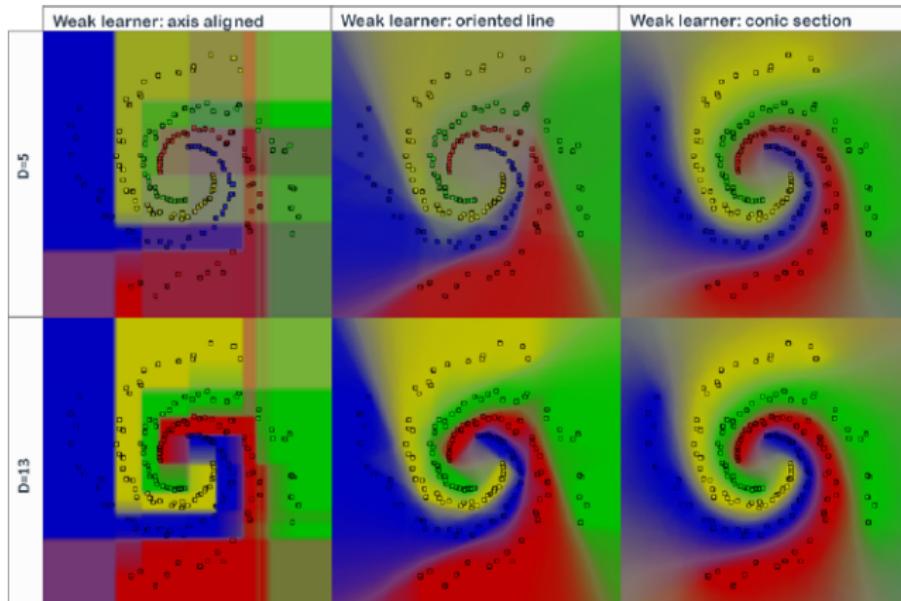
# Computational Complexity of Random Forests

- $N$  data points of dimensionality  $D$ , axis-aligned splits
- Picking  $m$  variables at random at each split;  $B$  trees
- Best case: balanced trees
  - Fitting:  $O(BmN \log N)$
  - Prediction:  $O(B \log N)$
- Worst case: splitting off one data point at a time
  - Fitting:  $O(BmN^2)$
  - Prediction:  $O(BN)$

# Some Ways of Ensembling Trees

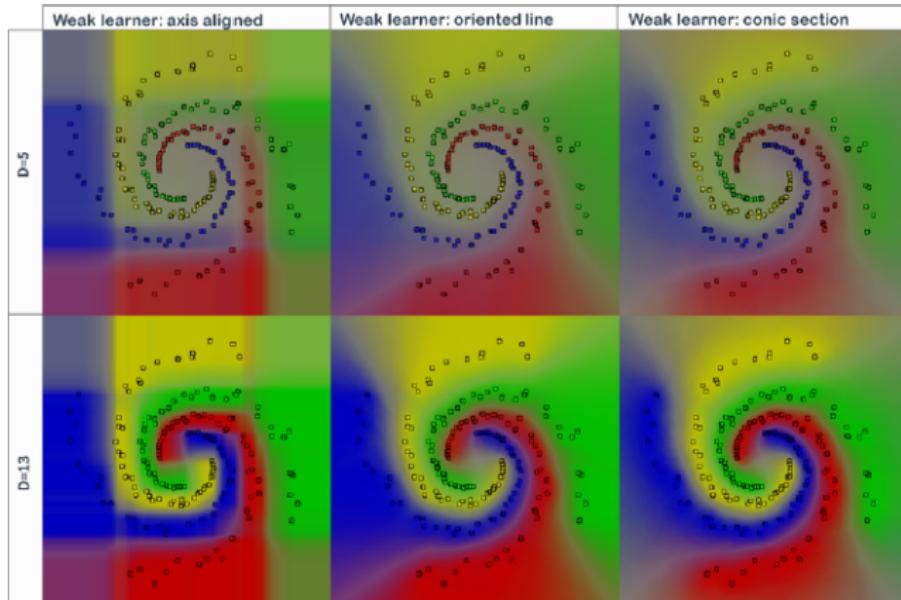
- Decision Trees + Bagging = **Bagged trees**
- Decision Trees + random feature subsets + Bagging = **Random forest**
- Decision Trees with almost random splits + Bagging = **ExtraTrees**
  - Extremely randomized trees

# Examples: Influence of split randomization



- 400 trees with `max_depth` 5 (top) and 13 (bottom)
- Different split types: axis aligned (left), oriented lines (middle), and conic sections (right)
- Splitting: best out of 500 random proposed splits for each splits

## Examples: Influence of split randomization



- 400 trees with `max_depth` 5 (top) and 13 (bottom)
- Different split types: axis aligned (left), oriented lines (middle), and conic sections (right)
- Splitting: best out of 5 random proposed splits for each splits

# Advantages of Bootstrapping

- Can help detect **outliers**
- Decorrelation of the trees in the ensemble
- **Out-of-bag error:**
  - not every data point is used to fit every single tree
  - in fact, almost 37% are not used in each tree (see assignment)
  - predict unused points for each tree to obtain  
**unbiased estimate of the generalization error**

# Pros and Cons of Random Forests (and co.)

- + All pros from decision trees remain (except interpretability)
- + Better generalization
- + Out-of-bag error with little overhead
- + Scalability to large data sets and high dimensions
- + Require little tuning
- Relatively weak performance for smooth functions without noise

## Summary by learning goals

Having heard this lecture, you can now ...

- determine **good splits** in regression and classification trees
- describe the steps of the **CART algorithm**
- describe the steps of the **random forest algorithm**
- formally describe **entropy** and **information gain**
- derive the **complexity** of decision trees and random forests
- explain why **bootstrapping** works well **for trees**
- describe some ways to **randomize trees** and their effects

# Further Reading

- Criminisi et. al (2013)

- Chapter 3: Introduction: The Abstract Forest Model
- Chapter 4: Classification Forests
- Chapter 5: Regression Forests
- PDF of entire book available from university machines: <http://link.springer.com/book/10.1007/978-1-4471-4929-3>



- Hastie, Tibshirani and Friedman

- Section 9.2: Tree-Based Methods
- Chapter 15: Random Forests

# Preview of Assignment 8

In assignment 8, you will . . .

- implement a simple regression tree and forest
- study hyperparameter influence on toy data
- calculate entropy and information gain by hand
- build a small decision tree by hand

# Lecture 14: Boosting

Machine Learning, Summer Term 2019

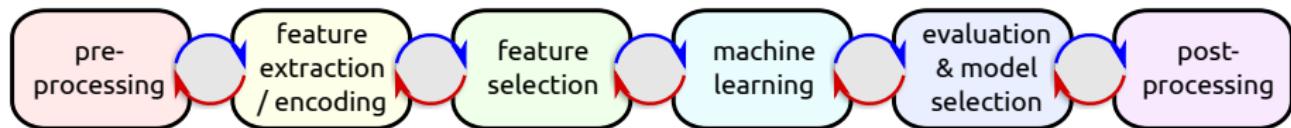
July 4, 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# The Big Picture



- Lecture 1: overview
- Lecture 2-6: linear methods
- Lecture 7-9: algorithm-independent principles
- Lectures 10-15: nonlinear methods
  - Lecture 10-12: kernel-based methods
  - Lectures 13-14: tree-based methods and ensembles
  - Lecture 15: neural networks

# Motivation for Boosting

Winning Methods of 10 Kaggle competitions in 2016 with non-image input  
(Competitions with images: deep neural networks dominate)

Almost all winners were large ensembles prominently using boosting:

Competition	Winners
Expedia Hotel Recommendations	1st: XGB
Santander Customer Satisfaction	3rd: XGB, RF, AdaBoost & others
Home Depot Product Search Relevance	1st and 3rd: XGB and RF
BNP Paribas Cardif Claims Management	1st and 2nd: XGB, RF & others
March Machine Learning Mania 2016	1st: RF and logarithmic regression
Telstra Network Disruptions	1st: sklearn, XGB, NN
Prudential Life Insurance Assessment	Top 3 used XGB (2nd and 3rd also others)
Airbnb New User Bookings	2nd: XGB; 3rd: XGB, NN, RF, ET
Homesite Quote Conversion	1st and 2nd: XGB & others; 3rd: others

XGB = Extreme Gradient Boosting

Places not listed: choice of method not released

# Motivation for Boosting

- Similar to random forests, boosting has often been called the **best off-the-shelf model** (e.g., by Leo Breiman, inventor of random forests)

# Motivation for Boosting

- Similar to random forests, boosting has often been called the **best off-the-shelf model** (e.g., by Leo Breiman, inventor of random forests)
- Similar advantages as random forests (we'll boost trees)
  - Trees: easy to **interpret**
  - Directly handle **categorical features**
  - Scalable to **many data points** (can be fast)
  - Scalable to **many features** (automated feature selectors)
  - **Robust** performance even for **small datasets**

# Lecture Overview

1 Introduction to Boosting

2 AdaBoost

3 Gradient Boosting

# Lecture Overview

1 Introduction to Boosting

2 AdaBoost

3 Gradient Boosting

# Boosting combines weak learners

- A **weak learner** is a learning algorithm that does at least slightly better than random (e.g., strictly better than 50% error for binary classification)
  - E.g., email spam filter: occurrence of 'buy now' would already let us classify better than random
  - Often, these simple rules already yield reasonable classification performance

# Boosting combines weak learners

- A **weak learner** is a learning algorithm that does at least slightly better than random (e.g., strictly better than 50% error for binary classification)
  - E.g., email spam filter: occurrence of 'buy now' would already let us classify better than random
  - Often, these simple rules already yield reasonable classification performance
- **Boosting** combines many weak learners into a highly accurate decision model.

# Boosting combines weak learners

- A **weak learner** is a learning algorithm that does at least slightly better than random (e.g., strictly better than 50% error for binary classification)
  - E.g., email spam filter: occurrence of 'buy now' would already let us classify better than random
  - Often, these simple rules already yield reasonable classification performance
- **Boosting** combines many weak learners into a highly accurate decision model.
- In this lecture, we use decision trees as our class of weak learners (even single-level trees, 'stumps', are sometimes used)

## Boosting: Relationship to Bagging

- Like bagging, boosting is a committee / method that combines many weak learners (submodels) into a highly accurate decision model.

## Boosting: Relationship to Bagging

- Like bagging, boosting is a committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels **sequentially**, with the  $m$ -th submodel trained to fix the mistakes of the first  $m - 1$  submodels

# Boosting: Relationship to Bagging

- Like bagging, boosting is a committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels **sequentially**, with the  $m$ -th submodel trained to fix the mistakes of the first  $m - 1$  submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging

## Boosting: Relationship to Bagging

- Like bagging, boosting is a committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels **sequentially**, with the  $m$ -th submodel trained to fix the mistakes of the first  $m - 1$  submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally give **different weights** to its submodels

# Boosting: Relationship to Bagging

- Like bagging, boosting is a committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels **sequentially**, with the  $m$ -th submodel trained to fix the mistakes of the first  $m - 1$  submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally give **different weights** to its submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging

# Boosting: Relationship to Bagging

- Like bagging, boosting is a committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels **sequentially**, with the  $m$ -th submodel trained to fix the mistakes of the first  $m - 1$  submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally give **different weights** to its submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally **bootstrap** samples and use **random feature subsets**

# Boosting: Relationship to Bagging

- Like bagging, boosting is a committee / method that combines many weak learners (submodels) into a highly accurate decision model.
- Boosting trains submodels **sequentially**, with the  $m$ -th submodel trained to fix the mistakes of the first  $m - 1$  submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally give **different weights** to its submodels
  - ★ This is exactly like bagging
  - ★ This is different to bagging
- Boosting can optionally **bootstrap** samples and use **random feature subsets**
  - ★ This is exactly like bagging
  - ★ This is different to bagging

## Two approaches to boosting

*Boosting: Submodels are trained sequentially, with the  $m$ -th submodel trained to fix the **mistakes** of the first  $m - 1$  submodels*

In this lecture, we will consider two algorithms that implement boosting:

# Two approaches to boosting

*Boosting: Submodels are trained sequentially, with the  $m$ -th submodel trained to fix the **mistakes** of the first  $m - 1$  submodels*

In this lecture, we will consider two algorithms that implement boosting:

- **Adaboost**, where mistakes are identified by weightings on more “difficult” data points
  - Each submodel also gets a **different weight**, based on how “good” it is

# Two approaches to boosting

*Boosting: Submodels are trained sequentially, with the  $m$ -th submodel trained to fix the **mistakes** of the first  $m - 1$  submodels*

In this lecture, we will consider two algorithms that implement boosting:

- **Adaboost**, where mistakes are identified by weightings on more “difficult” data points
  - Each submodel also gets a **different weight**, based on how “good” it is
- **Gradient Boosting**, where mistakes are identified by the gradient of our loss function
  - Each submodel gets the **same weight**

# Lecture Overview

1 Introduction to Boosting

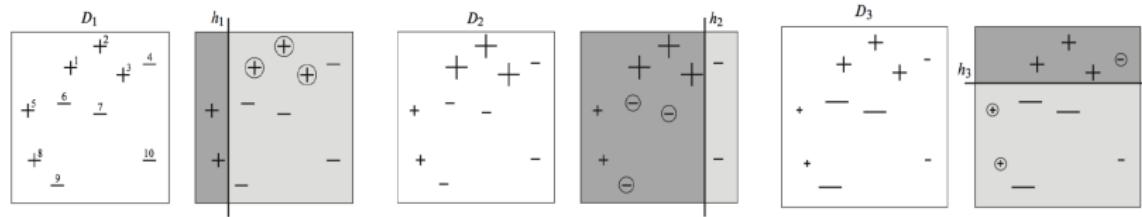
2 AdaBoost

3 Gradient Boosting

# AdaBoost [Freund and Schapire, 1995]

Basic idea: iteratively construct submodels  $G_m$  that fix previous errors

- We'll have a weight  $w_i$  for each data point  $i$ 
  - $w_i$  measures how hard data point  $x_i$  is to predict
  - Start with uniform weights  $w_i^{(1)}$ , adapt across iterations  $m$ :  $w_i^{(m)}$
  - In each iteration  $m$ , exponentially increase weights of data points  $x_i$  misclassified by  $G_m$ :



Figure

from [Schapire and Freund, 2012]

# AdaBoost [Freund and Schapire, 1995]

Basic idea: iteratively construct submodels  $G_m$  that fix previous errors

- We'll have a weight  $w_i$  for each data point  $i$ 
  - $w_i$  measures how hard data point  $x_i$  is to predict
  - Start with uniform weights  $w_i^{(1)}$ , adapt across iterations  $m$ :  $w_i^{(m)}$
  - In each iteration  $m$ ,  
exponentially increase weights of data points  $x_i$  misclassified by  $G_m$ :

$$w_i^{(m+1)} := \begin{cases} w_i^{(m)} \times \exp(\alpha_m) & , \text{ if } y_i \neq G_m(x_i) \\ w_i^{(m)} & , \text{ otherwise} \end{cases}$$

# AdaBoost [Freund and Schapire, 1995]

Basic idea: iteratively construct submodels  $G_m$  that fix previous errors

- We'll have a weight  $w_i$  for each data point  $i$ 
  - $w_i$  measures how hard data point  $x_i$  is to predict
  - Start with uniform weights  $w_i^{(1)}$ , adapt across iterations  $m$ :  $w_i^{(m)}$
  - In each iteration  $m$ ,  
exponentially increase weights of data points  $x_i$  misclassified by  $G_m$ :

$$\begin{aligned} w_i^{(m+1)} &:= \begin{cases} w_i^{(m)} \times \exp(\alpha_m) & , \text{ if } y_i \neq G_m(x_i) \\ w_i^{(m)} & , \text{ otherwise} \end{cases} \\ &= w_i^{(m)} \exp(\alpha_m \mathbb{I}(y_i \neq G_m(x_i))) \end{aligned}$$

# AdaBoost [Freund and Schapire, 1995]

Basic idea: iteratively construct submodels  $G_m$  that fix previous errors

- We'll have a weight  $w_i$  for each data point  $i$ 
  - $w_i$  measures how hard data point  $x_i$  is to predict
  - Start with uniform weights  $w_i^{(1)}$ , adapt across iterations  $m$ :  $w_i^{(m)}$
  - In each iteration  $m$ ,  
exponentially increase weights of data points  $x_i$  misclassified by  $G_m$ :

$$\begin{aligned} w_i^{(m+1)} &:= \begin{cases} w_i^{(m)} \times \exp(\alpha_m) & , \text{ if } y_i \neq G_m(x_i) \\ w_i^{(m)} & , \text{ otherwise} \end{cases} \\ &= w_i^{(m)} \exp(\alpha_m \mathbb{I}(y_i \neq G_m(x_i))) \end{aligned}$$

- We'll also compute a weight  $\alpha_m$  for each submodel  $G_m$ 
  - This depends on how good the model is
  - We'll use these weights  $\alpha_m$  for a weighted majority vote in the end

## AdaBoost [Freund and Schapire, 1995]

- Binary classification problem; class labels +1 and -1

## AdaBoost [Freund and Schapire, 1995]

- Binary classification problem; class labels +1 and -1
- Final model  $G(x)$  combines individual submodels  $G_1, \dots, G_M$  through a **weighted majority vote** with weights  $\alpha_1, \dots, \alpha_M$ :

$$G(x) = \text{sign} \left( \sum_{m=1}^M \alpha_m G_m(x) \right)$$

# AdaBoost [Freund and Schapire, 1995]

- Binary classification problem; class labels +1 and -1
- Final model  $G(x)$  combines individual submodels  $G_1, \dots, G_M$  through a **weighted majority vote** with weights  $\alpha_1, \dots, \alpha_M$ :

$$G(x) = \text{sign} \left( \sum_{m=1}^M \alpha_m G_m(x) \right)$$

- Submodels  $G_m$  are weighted depending on their error  $err_m$ :

$$\alpha_m := \log \frac{1 - err_m}{err_m}$$

- E.g.,  $err_m = 0.1 \rightarrow \alpha_m = 2.197$
- E.g.,  $err_m = 0.4 \rightarrow \alpha_m = 0.41$
- E.g.,  $err_m = 0.5 \rightarrow \alpha_m = 0$

# AdaBoost [Freund and Schapire, 1995]

- Binary classification problem; class labels +1 and -1
- Final model  $G(x)$  combines individual submodels  $G_1, \dots, G_M$  through a **weighted majority vote** with weights  $\alpha_1, \dots, \alpha_M$ :

$$G(x) = \text{sign} \left( \sum_{m=1}^M \alpha_m G_m(x) \right)$$

- Submodels  $G_m$  are weighted depending on their error  $err_m$ :

$$\alpha_m := \log \frac{1 - err_m}{err_m}$$

- E.g.,  $err_m = 0.1 \rightarrow \alpha_m = 2.197$

- E.g.,  $err_m = 0.4 \rightarrow \alpha_m = 0.41$

- E.g.,  $err_m = 0.5 \rightarrow \alpha_m = 0$

- This choice of  $\alpha_m$  can be shown to minimize an upper bound of the final hypothesis error (see [Schapire, 2003] for details)

# AdaBoost [Freund and Schapire, 1995]

- The (unweighted) **training error rate** of a submodel  $G_m$  is

$$\overline{err} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(y_i \neq G_m(x_i)).$$

## AdaBoost [Freund and Schapire, 1995]

- The (unweighted) **training error rate** of a submodel  $G_m$  is

$$\overline{err} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(y_i \neq G_m(x_i)).$$

- The **weighted training error rate** of submodel  $G_m$  is

$$err_m = \frac{\sum_{i=1}^N w_i \mathbb{I}(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$$

## AdaBoost [Freund and Schapire, 1995]

- The (unweighted) **training error rate** of a submodel  $G_m$  is

$$\overline{err} = \frac{1}{N} \sum_{i=1}^N \mathbb{I}(y_i \neq G_m(x_i)).$$

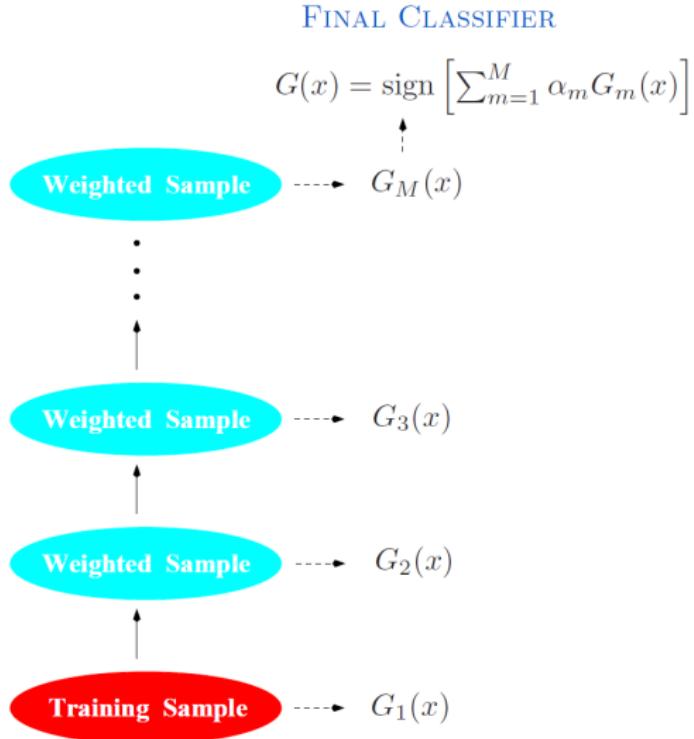
- The **weighted training error rate** of submodel  $G_m$  is

$$err_m = \frac{\sum_{i=1}^N w_i \mathbb{I}(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}$$

- As mentioned, these weighted training error rates are used for computing the model weights:

$$\alpha_m := \log \frac{1 - err_m}{err_m}$$

# AdaBoost [Freund and Schapire, 1995]

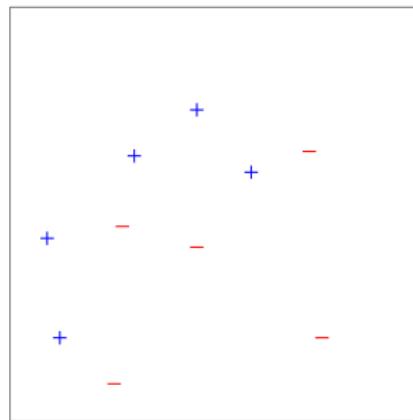


from [Hastie, Tibshirani and Friedman]

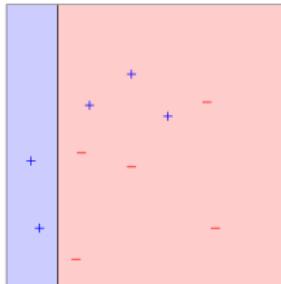
# AdaBoost Example (step $m = 1$ )

Example taken from [Schapire, 2003]

Model class: simple axis-aligned splits (decision stumps)



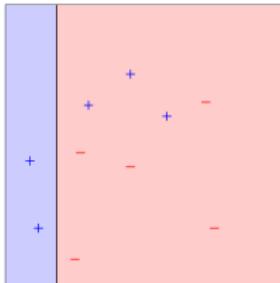
# AdaBoost Example (step $m = 1$ )



How large is the error  $err_1$  of this first model?

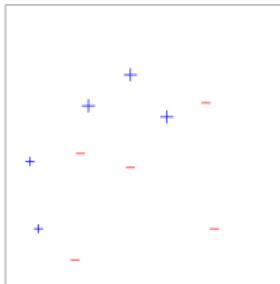
- ★ 0.3      ★ 0.2      ★ 0.5      ★ 0.7

# AdaBoost Example (step $m = 1$ )



How large is the error  $err_1$  of this first model?

- ★ 0.3      ★ 0.2      ★ 0.5      ★ 0.7



## AdaBoost Example (step $m = 1$ details)

- Model error and model weight:

$$err_1 = \sum_{i=1}^N w_i^{(1)} \mathbb{I}(G_1(x_i) \neq y_i) = \frac{1}{10} \times 3 = 0.3$$

$$\alpha_1 = \log \frac{1 - err_1}{err_1} = \log \frac{1 - 0.3}{0.3} \approx 0.847$$

# AdaBoost Example (step $m = 1$ details)

- Model error and model weight:

$$err_1 = \sum_{i=1}^N w_i^{(1)} \mathbb{I}(G_1(x_i) \neq y_i) = \frac{1}{10} \times 3 = 0.3$$

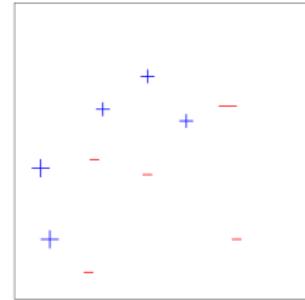
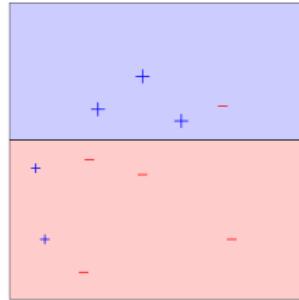
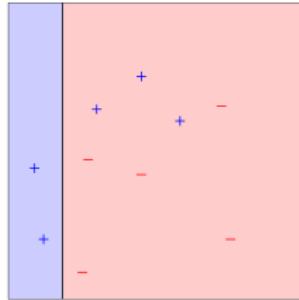
$$\alpha_1 = \log \frac{1 - err_1}{err_1} = \log \frac{1 - 0.3}{0.3} \approx 0.847$$

- Weight adaptation for data points:

$$w_i^{(m+1)} = w_i^{(m)} \exp(\alpha_m \mathbb{I}(y_i \neq G_m(x_i)))$$

- Misclassified data point:  $w_i^{(2)} \leftarrow w_i^{(1)} \exp(\alpha_1) \approx 0.1 \exp(0.847) \approx 0.233$
- Correctly classified data points:  $w_i^{(2)} \leftarrow w_i^{(1)} = 0.1$

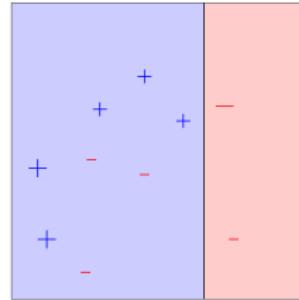
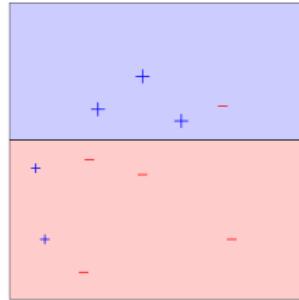
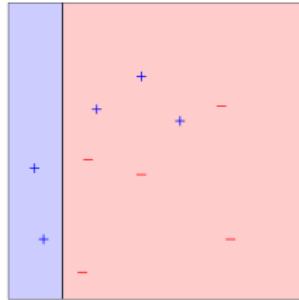
# AdaBoost Example (step $m = 2$ )



$$err_2 = \frac{\sum_{i=1}^N w_i^{(2)} \mathbb{I}(G_m(x_i) \neq y_i)}{\sum_{i=1}^N w_i^{(2)}} \approx \frac{0.1 + 0.1 + 0.1}{1.4} \approx 0.21$$

$$\alpha_2 = \log \frac{1 - err_2}{err_2} \approx \log \frac{1 - 0.21}{0.21} \approx 1.3$$

# AdaBoost Example (step $m = 3$ )

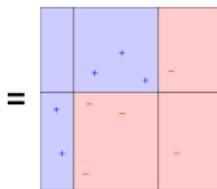
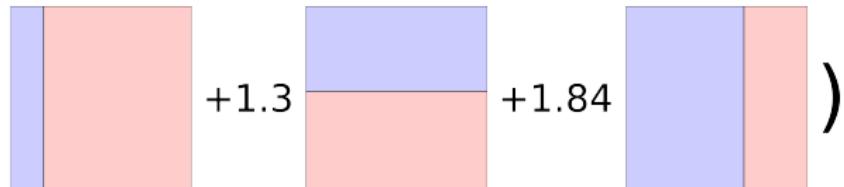


$$err_3 \approx 0.14, \alpha_3 \approx 1.84$$

# AdaBoost Example

Final classifier:

$$G = \text{sign} \left( +0.84 \right)$$



# AdaBoost – Complete Algorithm

---

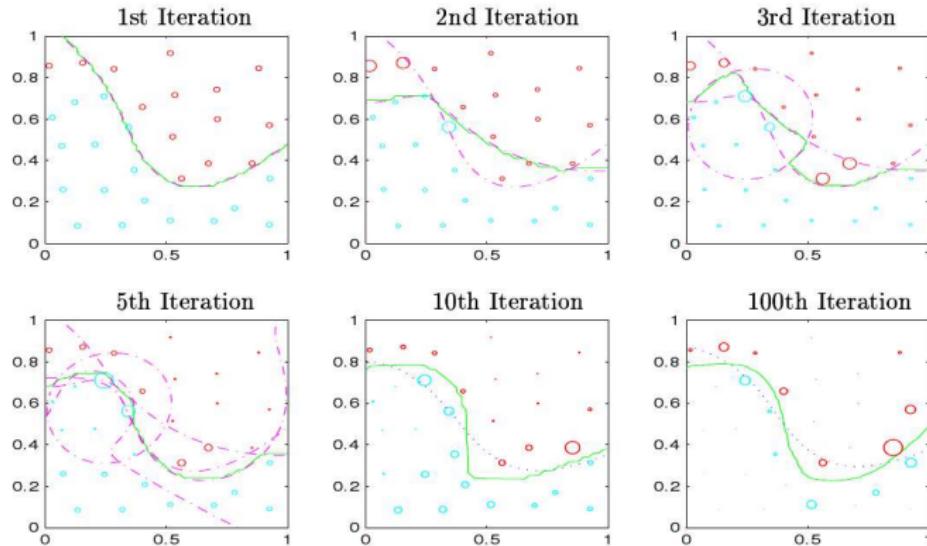
**Algorithm 10.1** *AdaBoost.M1.*

---

1. Initialize the observation weights  $w_i = 1/N, i = 1, 2, \dots, N$ .
  2. For  $m = 1$  to  $M$ :
    - (a) Fit a classifier  $G_m(x)$  to the training data using weights  $w_i$ .
    - (b) Compute
$$\text{err}_m = \frac{\sum_{i=1}^N w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^N w_i}.$$
    - (c) Compute  $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$ .
    - (d) Set  $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))], i = 1, 2, \dots, N$ .
  3. Output  $G(x) = \text{sign} \left[ \sum_{m=1}^M \alpha_m G_m(x) \right]$ .
- 

from [Hastie, Tibshirani and Friedman]

# AdaBoost: Focus on Hardest Data Points



Taken from [Meir, Raetsch, 2003]: AdaBoost on a 2D toy data set: color indicates the label, diameter is proportional to the weight of the example. Purple dashed lines: decision boundaries of the single classifiers (up to 5th iteration). Solid green line: decision boundary of the combined classifier. In the last two plots the decision line of Bagging is plotted for a comparison.

- ~ Main problem of AdaBoost: if the data is very noisy, it can overfit badly / it is very sensitive to outliers

# Lecture Overview

1 Introduction to Boosting

2 AdaBoost

3 Gradient Boosting

## Break: Teaching evaluations

- Teaching evaluations are running
  - Evaluations are our only formal reward
  - We redesigned much of the course, so feedback is very valuable
- Let's take a 10-minute break to do the evaluation right now

# An alternative formulation of boosting

- With AdaBoost, we took the sign of a weighted majority vote of  $M$  submodels, for binary classification:

$$G(x) = \text{sign} \left( \sum_{m=1}^M \alpha_m G_m(x) \right)$$

- Let's now generalise to a real-valued prediction, and remove the weights on each submodel:

$$G(x) = \sum_{m=1}^M G_m(x)$$

- Again, we will fit these  $M$  submodels sequentially.

## An alternative formulation of boosting

Suppose that  $M - 1$  submodels have already been fitted, and our task is to fit the  $M$ th submodel.

$$G(x) = \sum_{m=1}^{M-1} G_m(x) + \textcolor{blue}{G_M}(x)$$

## An alternative formulation of boosting

Suppose that  $M - 1$  submodels have already been fitted, and our task is to fit the  $M$ th submodel.

$$G(x) = \sum_{m=1}^{M-1} G_m(x) + \textcolor{blue}{G_M}(x)$$

As always, our objective is to minimise the loss over all data points:

$$\sum_{i=1}^N L(y_i, \sum_{m=1}^{M-1} G_m(x_i) + \textcolor{blue}{G_M}(x_i))$$

## An alternative formulation of boosting

Suppose that  $M - 1$  submodels have already been fitted, and our task is to fit the  $M$ th submodel.

$$G(x) = \sum_{m=1}^{M-1} G_m(x) + \textcolor{blue}{G_M}(x)$$

As always, our objective is to minimise the loss over all data points:

$$\sum_{i=1}^N L(y_i, \sum_{m=1}^{M-1} G_m(x_i) + \textcolor{blue}{G_M}(x_i))$$

Question: What method have we seen for iteratively minimising the loss of a model's predictions?

- ★ PCA   ★ ICA   ★ Talking to domain experts   ★ Gradient descent

# Gradient Boosting: Idea

- We can use **any** differentiable loss function  $L$ , and find the gradient  $g_i$  of our model's predictions (with  $M$  submodels) with respect to our predictions for each data point  $i$ :

$$g_i = \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)}$$

# Gradient Boosting: Idea

- We can use **any** differentiable loss function  $L$ , and find the gradient  $g_i$  of our model's predictions (with  $M$  submodels) with respect to our predictions for each data point  $i$ :

$$g_i = \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)}$$

- We then fit our new weak learner  $G_{M+1}$  with parameters  $\theta_{M+1}$  using a **gradient descent step**  $-\nu \cdot g$  as our target variable  $y$ .
  - $\nu$  is our gradient descent step size, or learning rate. In the context of gradient boosting,  $\nu$  is also known as **shrinkage**.

$$\theta_{M+1} = \arg \min_{\theta} \sum_{i=1}^N L(-\nu \cdot g_i, G(x_i; \theta))$$

# Gradient Boosting Algorithm (Simplified)

1. Initialize  $G = \text{best constant prediction for } y$ .
2. For  $m = 1$  to  $M$ 
  - (a) For  $i = 1$  to  $N$ :  $g_i = \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)}$
  - (b) Fit weak learner  $G_m$  to data  $\langle x_i, -\nu \cdot g_i \rangle_{i=1}^N$
  - (c) Update  $G(x) \leftarrow G(x) + G_m(x)$
3. Output final prediction  $G(x)$

# Gradient Boosting: Plugging in decision trees

1. Initialize  $G = \text{best constant prediction for } y.$
2. For  $m = 1$  to  $M$ 
  - (a) For  $i = 1$  to  $N$ :  $g_i = \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)}$
  - (b) Fit decision tree  $G_m$  to data  $\langle x_i, -\nu \cdot g_i \rangle_{i=1}^N$
  - (c) Update  $G(x) \leftarrow G(x) + G_m(x)$
3. Output final prediction  $G(x)$

Each of these lines is basically one line in Python (using a tree building library)

# Gradient Boosting: Plugging in Squared Error Loss

- Plugging in squared error loss to our gradient function, we get a familiar result:

$$\begin{aligned}g_i &= \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)} \\&= \frac{\partial \frac{1}{2}(y_i - G(x_i))^2}{\partial G(x_i)} \\&= -(y_i - G(x_i))\end{aligned}$$

# Gradient Boosting: Plugging in Squared Error Loss

- Plugging in squared error loss to our gradient function, we get a familiar result:

$$\begin{aligned}g_i &= \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)} \\&= \frac{\partial \frac{1}{2}(y_i - G(x_i))^2}{\partial G(x_i)} \\&= -(y_i - G(x_i))\end{aligned}$$

- Doing a step w.r.t. the negative gradient: Fit a submodel with the previous model's residuals  $y_i - G(x_i)$  as targets.

- Review: why do we use the *negative* gradient?



# Gradient Tree Boosting with L2 Error

1. Initialize  $G = \text{best constant prediction for } y.$
  2. For  $m = 1$  to  $M$ 
    - (a) For  $i = 1$  to  $N$ :  $r_i = y_i - G(x_i)$
    - (b) Fit decision tree  $G_m$  to data  $\langle x_i, -\nu \cdot r_i \rangle_{i=1}^N$
    - (c) Update  $G(x) \leftarrow G(x) + G_m(x)$
  3. Output final prediction  $G(x)$
- 
- Computationally very simple!
  - This is related to **forward stagewise additive modelling**, which fits an additive model by greedily fitting one component at a time

# Gradient Boosting: Plugging in Absolute Error

- Plugging in **absolute error loss** to our gradient function, we get another effective algorithm:

$$\begin{aligned}g_i &= \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)} \\&= \frac{\partial |y_i - G(x_i)|}{\partial G(x_i)} \\&= \text{sign}(y_i - G(x_i))\end{aligned}$$

# Gradient Boosting: Plugging in Absolute Error

- Plugging in **absolute error loss** to our gradient function, we get another effective algorithm:

$$\begin{aligned}g_i &= \frac{\partial L(y_i, G(x_i))}{\partial G(x_i)} \\&= \frac{\partial |y_i - G(x_i)|}{\partial G(x_i)} \\&= \text{sign}(y_i - G(x_i))\end{aligned}$$

- Review: when would you expect this to perform better than squared error loss?



# The link between Gradient Boosting and Adaboost

- What if we plug in the exponential loss function

$$L(y_i, G(x_i)) = \exp(-y_i \times G(x_i))?$$

# The link between Gradient Boosting and Adaboost

- What if we plug in the exponential loss function  
 $L(y_i, G(x_i)) = \exp(-y_i \times G(x_i))$ ?
- It turns out that we recover the Adaboost algorithm!
  - The optimal  $G_m$  minimizes weighted error:

$$G_m = \arg \min_G \sum_{i=1}^N w_i^{(m)} \mathbb{I}(y_i \neq G(x_i))$$

- Reintroduce a model weight  $\alpha_m$  as in Adaboost:

$$\alpha_m = \log \frac{1 - err_m}{err_m}$$

- The weight update rule is as in Adaboost:

$$w_i^{(m+1)} \propto w_i^{(m)} \cdot \exp(\alpha_m \mathbb{I}(y_i \neq G_m(x_i)))$$

- (More info on ILIAS; these details will not be examinable)

# Regularisation for Tree Complexity

- Recall our formal definition of a tree  $G(x_i; \theta)$ , where:

$$\theta = \langle \underbrace{\{R_1, \dots, R_J\}}_{\text{leaf regions}}, \underbrace{\{w_1, \dots, w_J\}}_{\text{leaf scores}} \rangle$$

# Regularisation for Tree Complexity

- Recall our formal definition of a tree  $G(x_i; \theta)$ , where:

$$\theta = \underbrace{\langle \{R_1, \dots, R_J\}, \{w_1, \dots, w_J\} \rangle}_{\text{leaf regions}}_{\text{leaf scores}}$$

- We can impose a complexity penalty  $\Omega$  on each submodel  $G_m$ 
  - hyperparameters  $\gamma$  and  $\lambda$  penalise tree complexity  $J$  and leaf score magnitude  $\|\mathbf{w}_m\|$  respectively:

$$\Omega(G_m) = \gamma J_m + \frac{1}{2} \lambda \|\mathbf{w}_m\|_2^2$$

# Regularisation for Tree Complexity

- Recall our formal definition of a tree  $G(x_i; \theta)$ , where:

$$\theta = \underbrace{\{R_1, \dots, R_J\}}_{\text{leaf regions}}, \underbrace{\{w_1, \dots, w_J\}}_{\text{leaf scores}}$$

- We can impose a complexity penalty  $\Omega$  on each submodel  $G_m$ 
  - hyperparameters  $\gamma$  and  $\lambda$  penalise tree complexity  $J$  and leaf score magnitude  $\|\mathbf{w}_m\|$  respectively:

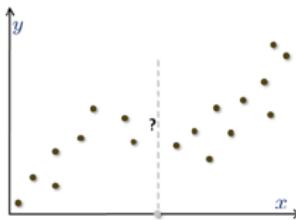
$$\Omega(G_m) = \gamma J_m + \frac{1}{2} \lambda \|\mathbf{w}_m\|_2^2$$

- Our method for fitting each submodel then becomes:

$$\theta_m = \arg \min_{\theta} \left[ \sum_{i=1}^N L(-\nu \cdot g_i, G(x_i; \theta)) + \Omega(G_m) \right]$$

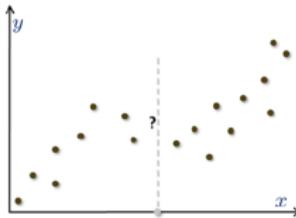
# Regularisation for Tree Complexity

- Suppose we have a prior belief that:
  - there should be **few** divisions along the axes of our input space . . .
  - . . . but that these divisions should separate **highly** different samples



# Regularisation for Tree Complexity

- Suppose we have a prior belief that:
  - there should be **few** divisions along the axes of our input space . . .
  - . . . but that these divisions should separate **highly** different samples



- What would be an appropriate hyperparameter setting?

$$\Omega(G_m) = \gamma J_m + \frac{1}{2} \lambda \|\mathbf{w}_m\|_2^2$$

- ★ High  $\gamma$ , High  $\lambda$    ★ High  $\gamma$ , Low  $\lambda$   
★ Low  $\gamma$ , High  $\lambda$    ★ Low  $\gamma$ , Low  $\lambda$

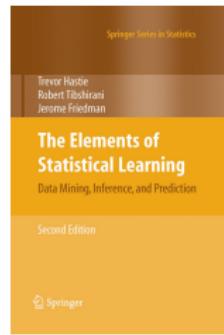
# Summary by learning goals

Having heard this lecture, you can now ...

- explain the principles of boosting, and how it differs from bagging
- describe the steps of the AdaBoost algorithm
- describe the steps of the gradient boosting approach
- Explain the relationship between gradient boosting, gradient descent, and Adaboost
- Describe the role of shrinkage and regularisation in gradient boosting

# Further Reading

- Main source: Hastie, Tibshirani and Friedman
  - Chapter 10: Boosting and Additive Trees
- Wikipedia article on boosting
- Specialized literature
  - Schapire: A boosting tutorial  
[www.cs.princeton.edu/~schapire](http://www.cs.princeton.edu/~schapire)
  - Meir and Raetsch, 2003: An introduction to boosting
  - Raetsch: Tutorial at MLSS 2003



# Preview of Assignment 9

In the 9<sup>th</sup> assignment, you will

- Implement the AdaBoost algorithm
- Implement the gradient boosting algorithm

# Derivation of AdaBoost as a Special Case of Forward Stagewise Additive Modelling

Matthias Feurer

Katharina Eggensperger

Frank Hutter

July 3, 2018

This explanation is based on Section 10.4 (“Exponential Loss and AdaBoost”) from:

- [1] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, Springer, 2nd edition, 2005, <http://statweb.stanford.edu/~tibs/ElemStatLearn/>.

We highly recommend Chapter 10 of Hastie et al. We only wrote this note to fill in the gory details for the derivation in Section 10.4.

AdaBoost combines several weak learners through a weighted majority vote to produce the final prediction (equation 10.1):

$$G(x) = \text{sign}\left(\sum_{m=1}^M \alpha_m G_m(x)\right) \quad (1)$$

Boosting is a special way to combine weak learners (basis functions) in an additive way. Section 10.2 provides other examples of this technique, for example neural networks, wavelets and decision trees.

Such additive models could optimally be fit by minimizing an averaged loss function over the training data (10.4):

$$\min_{\{\beta_m, \gamma_m\}_1^M} \sum_{i=1}^N \mathbb{L}(y_i, \sum_{m=1}^M \beta_m b(x_i; \gamma_m)) \quad (2)$$

However, since solving this combined minimization problem is often prohibitive, we resort to a technique called *Forward Stagewise Additive Modeling*. At each iteration  $m$ , this solves for the optimal next weak learner  $b(x, \gamma_m)$  and the corresponding coefficient  $\beta_m$  (without changing the previous weak learners and coefficients) and adds the current expansion  $f_{m-1}(x)$  to produce  $f_m(x)$ .

We will show that AdaBoost is a special case of forward stagewise additive modeling using the following exponential loss function (10.8):

$$\mathbb{L}(y, f(x)) = \exp(-y \times f(x)) \quad (3)$$

In each iteration, forward stagewise additive modeling must solve the following minimization problem:

$$(\beta_m, G_m) = \arg \min_{\beta, G} \sum_{i=1}^N \exp(-y_i(f_{m-1}(x_i) + \beta G(x_i))) \quad (4)$$

$$= \arg \min_{\beta, G} \sum_{i=1}^N \exp(-y_i f_{m-1}(x_i) + (-y_i)\beta G(x_i)) \quad (5)$$

$$= \arg \min_{\beta, G} \sum_{i=1}^N \exp(-y_i f_{m-1}(x_i)) \cdot \exp(-y_i \beta G(x_i)) \quad (6)$$

Because  $y_i$  and  $f_{m-1}$  neither depend on  $\beta$  nor  $G(x)$ , we can write the term  $\exp(-y_i f_{m-1}(x_i))$  as a weight, which is applied to each individual observation:

$$w_i^{(m)} = \exp(-y_i f_{m-1}(x_i)).$$

This leads us to the following equation (10.9):

$$(\beta_m, G_m) = \arg \min_{\beta_m, G} \sum_{i=1}^N w_i^{(m)} \exp(-\beta y_i G(x_i)) \quad (7)$$

Weights depend on  $f_{(m-1)}$ , and so the individual weight values change with each iteration  $m$ . We can solve this equations in two steps:

1. find a model for  $G_m(x)$ .
2. find a value for  $\beta_m$ .

We first show that the optimal  $G_m(x)$  does not depend on  $\beta$ . With  $\beta > 0$ , we split the right-hand side of the above equation into two terms and then simplify:

$$G_m = \arg \min_G \sum_{y_i=G(x_i)} w_i^{(m)} \exp(-y_i \beta G(x_i)) + \sum_{y \neq G(x_i)} w_i^{(m)} \exp(-y_i \beta G(x_i)) \quad (8)$$

$$= \arg \min_G \sum_{y_i=G(x_i)} w_i^{(m)} \exp(-\beta) + \sum_{y \neq G(x_i)} w_i^{(m)} \exp(\beta) \quad (9)$$

$$= \arg \min_G \exp(-\beta) \sum_{y_i=G(x_i)} w_i^{(m)} + \exp(\beta) \sum_{y \neq G(x_i)} w_i^{(m)} \quad (10)$$

In the first sum  $y_i$  and  $G(x_i)$  are equal, therefore  $\exp(-y_i\beta G(x_i)) = \exp(-\beta)$ . In the second sum they differ, changing the sign in the exponent:  $\exp(-y_i\beta G(x_i)) = \exp(\beta)$ . Hastie et al. rewrite this in equation 10.11 in a way which shows that only the weighted errors influence the choice:

$$G_m = \arg \min_G \left( (\exp(\beta) - \exp(-\beta)) \cdot \sum_{i=1}^N w_i^{(m)} \mathbb{I}(y_i \neq G(x_i)) + \exp(-\beta) \cdot \sum_{i=1}^N w_i^{(m)} \right) \quad (11)$$

Observe that the first sum resembles the second sum of the upper equation, but is multiplied with a lower factor. The second sum serves as a correction factor in the case of a wrong classification, or is equal to the first sum in the positive case.

Now that we have a strategy for finding  $G_m$ , we turn to finding  $\beta_m$ . Setting the partial derivative of 10.9 with respect to  $\beta$  to zero and solving for  $\beta$ , we obtain:

$$\frac{\partial(\sum_{i=1}^N w_i^{(m)} \exp(-\beta y_i G(x_i)))}{\partial \beta} = 0 \quad (12)$$

$$\Rightarrow \frac{\partial(\exp(-\beta) \sum_{y=G(x)} w_i^{(m)} + \exp(\beta) \sum_{y \neq G(x)} w_i^{(m)})}{\partial \beta} = 0 \quad (13)$$

$$\Rightarrow -\exp(-\beta) \sum_{y=G(x)} w_i^{(m)} + \exp(\beta) \sum_{y \neq G(x)} w_i^{(m)} = 0 \quad (14)$$

$$\Rightarrow \exp(\beta) \sum_{y \neq G(x)} w_i^{(m)} = \exp(-\beta) \sum_{y=G(x)} w_i^{(m)}$$

$$\Rightarrow \log(\exp(\beta) \sum_{y \neq G(x)} w_i^{(m)}) = \log(\exp(-\beta) \sum_{y=G(x)} w_i^{(m)})$$

$$\Rightarrow \log(\exp(\beta)) + \log(\sum_{y \neq G(x)} w_i^{(m)}) = \log(\exp(-\beta)) + \log(\sum_{y=G(x)} w_i^{(m)})$$

$$\Rightarrow \beta + \log(\sum_{y \neq G(x)} w_i^{(m)}) = -\beta + \log(\sum_{y=G(x)} w_i^{(m)})$$

$$\Rightarrow 2\beta = \log(\sum_{y=G(x)} w_i^{(m)}) - \log(\sum_{y \neq G(x)} w_i^{(m)})$$

$$\Rightarrow \beta = \frac{1}{2} \log \frac{\sum_{y=G(x)} w_i^{(m)}}{\sum_{y \neq G(x)} w_i^{(m)}}$$

Now that we have an expression for  $\beta$ , we want to reformulate it, depending only on the weighted classification errors:

$$\Rightarrow \beta = \frac{1}{2} \log \frac{\sum_{y=G(x)} w_i^{(m)} + \sum_{y \neq G(x)} w_i^{(m)} - \sum_{y \neq G(x)} w_i^{(m)}}{\sum_{y \neq G(x)} w_i^{(m)}} \quad (15)$$

$$\Rightarrow \beta = \frac{1}{2} \log \frac{\sum_{i=1}^N w_i^{(m)} - \sum_{y \neq G(x)} w_i^{(m)}}{\sum_{y \neq G(x)} w_i^{(m)}} \quad (16)$$

$$\Rightarrow \beta = \frac{1}{2} \log \frac{(\sum_{i=1}^N w_i^{(m)} - \sum_{y \neq G(x)} w_i^{(m)}) \cdot \frac{1}{\sum_{i=1}^N w_i^{(m)}}}{\sum_{y \neq G(x)} w_i^{(m)} \cdot \frac{1}{\sum_{i=1}^N w_i^{(m)}}} \quad (17)$$

(18)

By defining  $\epsilon = \frac{\sum_{y \neq G(x)} w_i^{(m)}}{\sum_{i=1}^N w_i^{(m)}}$ , we get that  $\beta_m$  ( $\beta$  at step  $m$ ) is:

$$\beta_m = \frac{1}{2} \log \frac{1 - \epsilon}{\epsilon} \quad (19)$$

The last missing piece is the weight update rule. By using the update rule of forward stagewise additive modelling:

$$f_m(x) = m_{(m-1)}(x) + \beta_m G_m(x) \quad | - y \quad (20)$$

$$-y f_m(x) = -y m_{(m-1)}(x) + -y \beta_m G_m(x) \quad | \exp() \quad (21)$$

$$\exp(-y f_m(x)) = \exp(-y m_{(m-1)}(x) + (-y \beta_m G_m(x))) \quad (22)$$

$$\exp(-y f_m(x)) = \exp(-y m_{(m-1)}(x)) \cdot \exp(-y \beta_m G_m(x)) \quad (23)$$

and the definition  $w_i^{(m)} = \exp(-y_i f_{(m-1)}(x_i))$  we obtain:

$$w_i^{(m+1)} = w_i^{(m)} \cdot \exp(-\beta_m y_i G_m(x_i)). \quad (24)$$

Using the fact that  $-y_i G_m(x_i) = 2 \cdot \mathbb{I}(y_i \neq G_m(x_i)) - 1$ , this becomes

$$w_i^{(m+1)} = w_i^{(m)} \cdot \exp(\alpha_m \mathbb{I}(y_i \neq G_m(x_i))) \cdot \exp(-\beta_m), \quad (25)$$

where  $\alpha_m = 2\beta_m$  and  $\exp(-\beta_m)$  is a constant which multiplies all weights and thus has no effect.

Now we have all parts together for the AdaBoost.M1 algorithm:

---

**Algorithm 1:** AdaBoost.M1

---

1. Initialize the observation weights  $w_i = \frac{1}{N}$ ,  $i = 1, 2, \dots, N$ .
  2. For  $m = 1$  to  $M$ :
    - a) Fit a classifier  $G_m(x)$  to the training data using weights  $w_i$ .
    - b) Compute  $\epsilon = \frac{\sum_{y \neq G(x)} w_i^{(m)}}{\sum_{i=1}^N w_i^{(m)}}$ .
    - c) Compute  $\alpha = \log \frac{1-\epsilon}{\epsilon}$ .
    - d) Set  $w_i \leftarrow w_i \cdot \exp(\alpha_m \cdot \mathbb{I}(y_i \neq G_m(x_i)))$ ,  $i = 1, 2, \dots, N$
  - Output  $G(x) = \text{sign}(\sum_{m=1}^M \alpha_m G_m(x))$
-

# Lecture 15: Deep Learning

Machine Learning, Summer Term 2019

July 11, 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# Motivation: Deep Learning in the News

The New York Times

Science

WORLD U.S. N.Y. / REGION BUSINESS TECHNOLOGY SCIENCE HEALTH SPORTS OPINION ENVIRONMENT SPACE & COSMOS



Scientists See Promise in Deep-Learning



A voice recognition program translated a speech given by Richard F. Rashid. [Read more](#)

By JOHN MARKOFF  
Published: November 23, 2012

Using an artificial intelligence technique inspired by theories of the brain recognizes patterns, technology companies are reporting gains in fields as diverse as computer vision, speech recognition

THE NEW YORKER

EVERY STORE. EVERY DEVICE. - REVIEW - GIVE A GIFT - SPOTLIGHT - DISCOUNTS

NEWS CULTURE BOOKS & FICTION SCIENCE & TECH BUSINESS HUMOR MAGAZINE ARCHIVE SUBSCRIBE

SEARCH

NOVEMBER 25, 2012

## IS "DEEP LEARNING" A REVOLUTION IN ARTIFICIAL INTELLIGENCE?

BY GARY MARCUS



Can a new technique known as deep learning revolutionize artificial intelligence, as yesterday's front-page article at the New York Times suggests? There is good reason to be excited about deep learning, a sophisticated "machine learning" algorithm that far exceeds many of its predecessors in its abilities to recognize syllables and images. But there's also good reason to be skeptical. While the Times reports that "advances in an artificial intelligence technology that can recognize patterns offer

MIT Technology Review

## 10 BREAKTHROUGH TECHNOLOGIES 2013

Introduction Past Years The 10 Technologies

### Deep Learning

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.



# Lecture Overview

- 1 Motivation: Why is Deep Learning so Popular?
- 2 Representation Learning and Deep Learning
- 3 Multilayer Perceptrons
- 4 Optimization of Neural Networks in a Nutshell
- 5 Overview of Some Advanced Topics
  - Convolutional neural networks
  - Recurrent neural networks
  - Deep reinforcement learning
- 6 Limitations
- 7 Wrapup

# Lecture Overview

1 Motivation: Why is Deep Learning so Popular?

2 Representation Learning and Deep Learning

3 Multilayer Perceptrons

4 Optimization of Neural Networks in a Nutshell

5 Overview of Some Advanced Topics

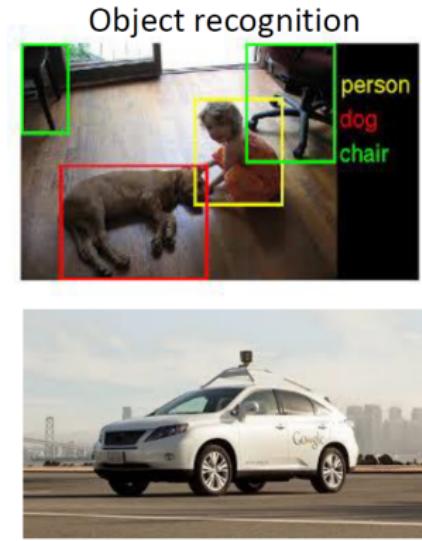
- Convolutional neural networks
- Recurrent neural networks
- Deep reinforcement learning

6 Limitations

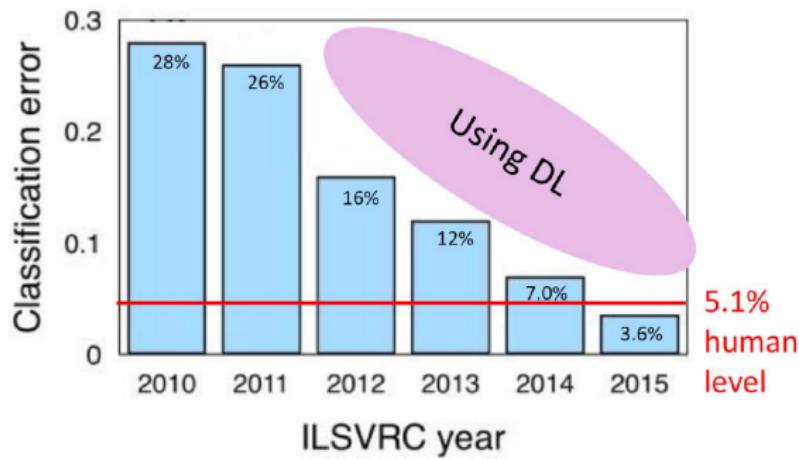
7 Wrapup

# Motivation: Why is Deep Learning so Popular?

- Excellent empirical results, e.g., in computer vision



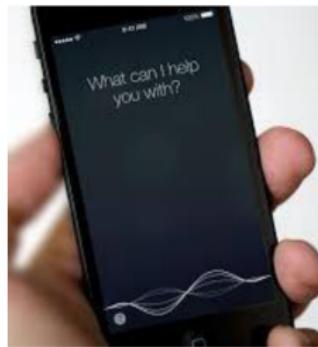
Object recognition  
Self-driving cars



# Motivation: Why is Deep Learning so Popular?

- Excellent empirical results, e.g., in speech recognition

Speech recognition



Auto-Translator

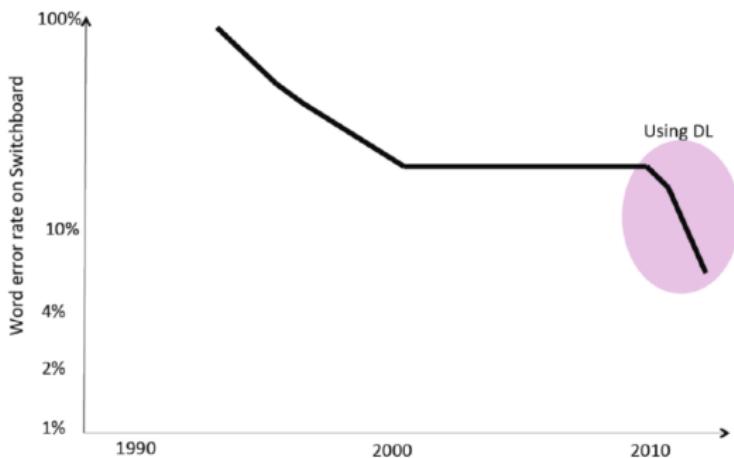
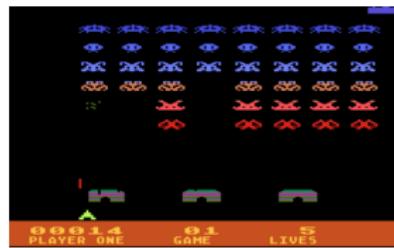


Image credit: Yoshua Bengio (data from Microsoft speech group)

# Motivation: Why is Deep Learning so Popular?

- Excellent empirical results, e.g., in reasoning in games

- Superhuman performance in playing Atari games  
[Mnih et al, Nature 2015]



- Beating the world's best Go player  
[Silver et al, Nature 2016]



# An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Speech recognition
  - Playing Atari games
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience

# An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Speech recognition
  - Playing Atari games
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience
- If the task changes, we simply re-train

# An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- We don't understand how the human brain solves certain problems
  - Face recognition
  - Speech recognition
  - Playing Atari games
  - Picking the next move in the game of Go
- We can nevertheless learn these tasks from data/experience
- If the task changes, we simply re-train
- We can construct computer systems that are too complex for us to understand anymore ourselves...
  - E.g., deep neural networks have millions of weights.
  - E.g., AlphaGo, the system that beat world champion Lee Sedol
    - + David Silver, lead author of AlphaGo cannot say why a move is good
    - + Paraphrased: "You would have to ask a Go expert."

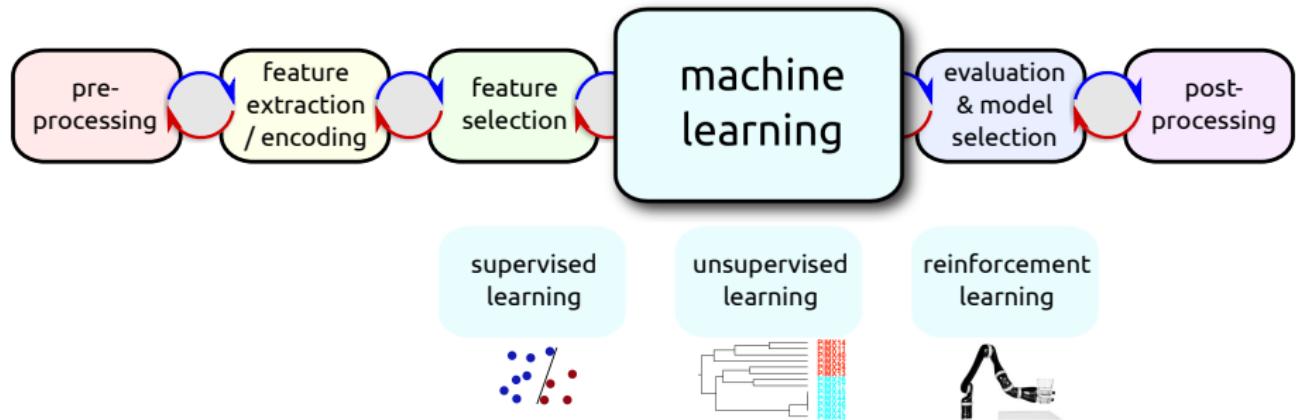
# An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

- Learning from data / experience may be more human-like
  - Babies develop an intuitive understanding of physics in their first 2 years
  - Formal reasoning and logic comes **much** later in development

# An Exciting Approach to AI: Learning as an Alternative to Traditional Programming

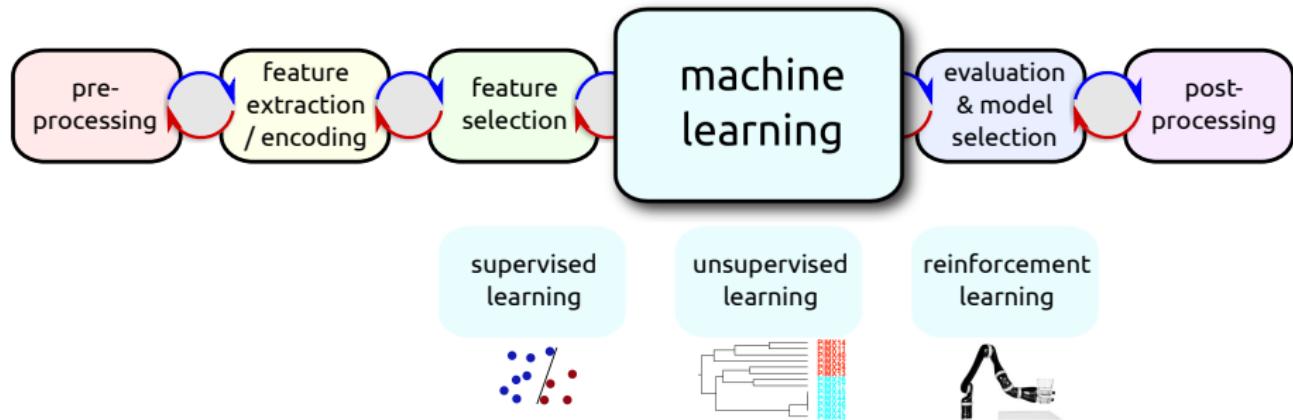
- Learning from data / experience may be more human-like
  - Babies develop an intuitive understanding of physics in their first 2 years
  - Formal reasoning and logic comes **much** later in development
- Learning enables **fast reaction times**
  - It might take a long time to train a neural network
  - But predicting with the network is very fast
  - Contrast this to running a planning algorithm every time you want to act

# Deep Learning in the ML Design Cycle



- Much reduced need for preprocessing
- No more need for manual feature engineering (extraction & selection)
  - useful features are learned automatically

# Deep Learning in the ML Design Cycle



- Much reduced need for preprocessing
- No more need for manual feature engineering (extraction & selection)
  - useful features are learned automatically
- End-to-end training:
  - replacing most of the ML pipeline by a single approach
  - directly optimizing a single loss function

# Lecture Overview

1 Motivation: Why is Deep Learning so Popular?

2 Representation Learning and Deep Learning

3 Multilayer Perceptrons

4 Optimization of Neural Networks in a Nutshell

5 Overview of Some Advanced Topics

- Convolutional neural networks
- Recurrent neural networks
- Deep reinforcement learning

6 Limitations

7 Wrapup

# From fixed basis functions to learned representations

- So far: use fixed basis functions or features defined by an expert to extend linear models

# From fixed basis functions to learned representations

- So far: use fixed basis functions or features defined by an expert to extend linear models
- Sometimes not easy to define what the right features are!

# From fixed basis functions to learned representations

- So far: use fixed basis functions or features defined by an expert to extend linear models
- Sometimes not easy to define what the right features are!
- Yet: good representations make solving a task easier: consider adding number in roman literals vs arabic numbers, e.g.

CCLXXII + LXVIII

# From fixed basis functions to learned representations

- So far: use fixed basis functions or features defined by an expert to extend linear models
- Sometimes not easy to define what the right features are!
- Yet: good representations make solving a task easier: consider adding number in roman literals vs arabic numbers, e.g.

CCLXXII + LXVIII

vs

$$272 + 68 = 340$$

# From fixed basis functions to learned representations

- So far: use fixed basis functions or features defined by an expert to extend linear models
- Sometimes not easy to define what the right features are!
- Yet: good representations make solving a task easier: consider adding number in roman literals vs arabic numbers, e.g.

CCLXXII + LXVIII

vs

$$272 + 68 = 340$$

- **Representation Learning:** parametrize basis functions and adapt to the data to discover useful representations automatically

## Representation learning

“a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification”

# Some definitions

## Representation learning

“a set of methods that allows a machine to be fed with raw data and to automatically discover the representations needed for detection or classification”

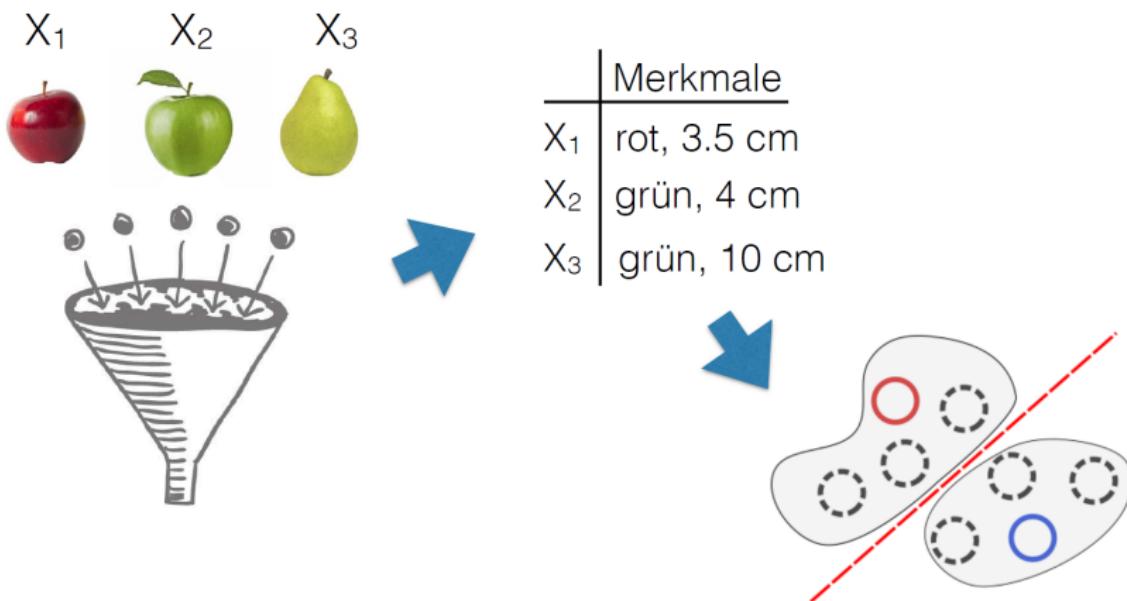
## Deep learning

“representation learning methods with multiple levels of representation, obtained by composing simple but nonlinear modules that each transform the representation at one level into a [...] higher, slightly more abstract (one)”

(LeCun et al., 2015)

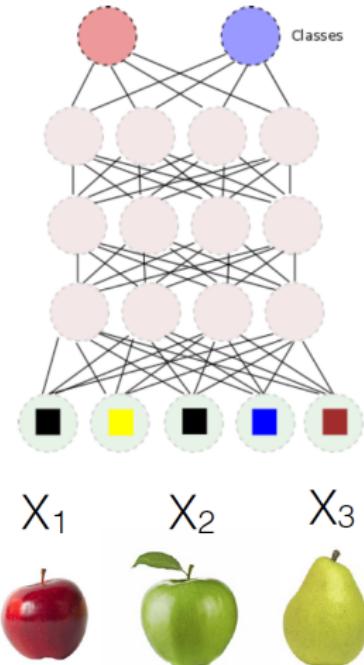
# Standard Machine Learning Pipeline

- Standard machine learning algorithms are based on high-level **attributes** or **features** of the data
- E.g., the binary attributes we used for decisions trees
- This requires (often substantial) **feature engineering**

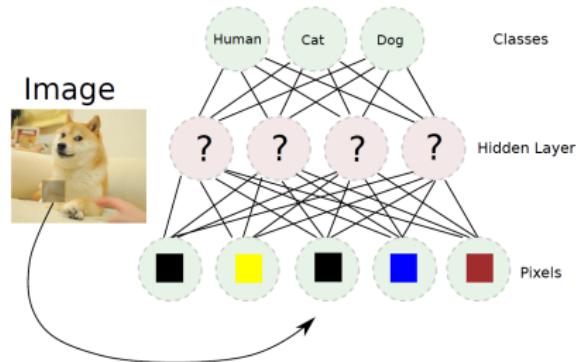


# Representation Learning Pipeline

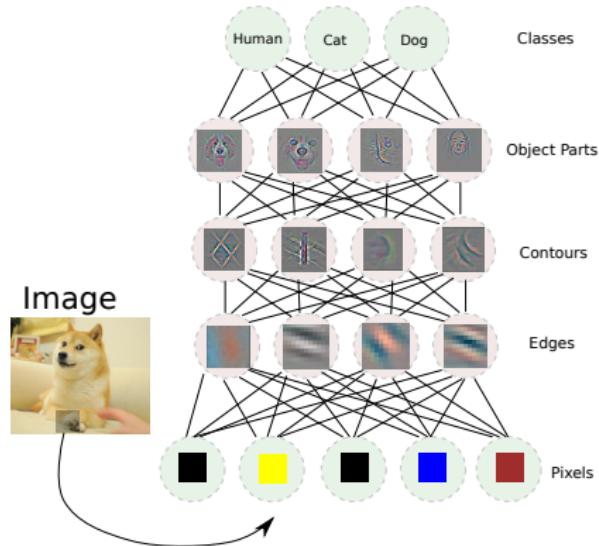
- Jointly learn features and classifier, directly from raw data
- This is also referred to as **end-to-end learning**



# Shallow vs. Deep Learning

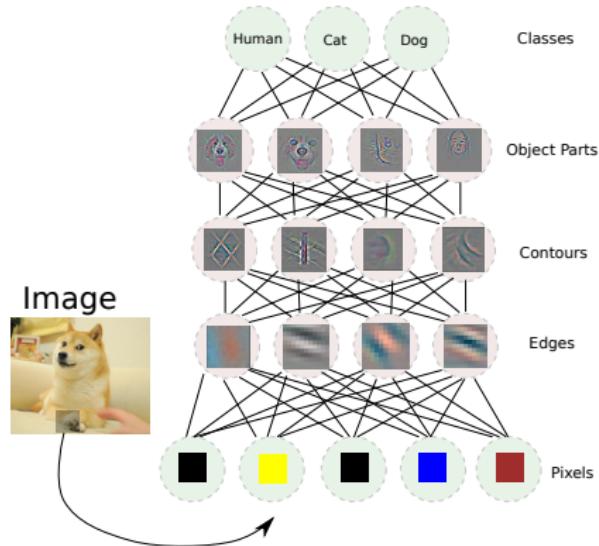


# Shallow vs. Deep Learning



- **Deep Learning:** learning a hierarchy of representations that build on each other, **from simple to complex**

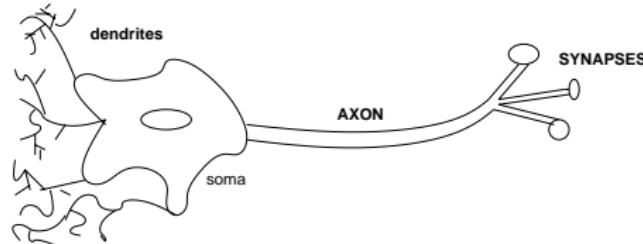
# Shallow vs. Deep Learning



- **Deep Learning:** learning a hierarchy of representations that build on each other, **from simple to complex**
- Quintessential deep learning model: **Multilayer Perceptrons**

# Biological Inspiration of Artificial Neural Networks

- Dendrites input information to the cell
- Neuron fires (has action potential) if a certain threshold for the voltage is exceeded
- Output of information by axon
- The axon is connected to dendrites of other cells via synapses
- Learning: adaptation of the synapse's efficiency, its synaptic weight



# A Very Brief History of Neural Networks

- Neural networks have a [long history](#)
  - 1942: artificial neurons (McCulloch/Pitts)
  - 1958/1969: perceptron (Rosenblatt; Minsky/Papert)
  - 1986: multilayer perceptrons and backpropagation (Rumelhart)
  - 1989: convolutional neural networks (LeCun)

# A Very Brief History of Neural Networks

- Neural networks have a [long history](#)
  - 1942: artificial neurons (McCulloch/Pitts)
  - 1958/1969: perceptron (Rosenblatt; Minsky/Papert)
  - 1986: multilayer perceptrons and backpropagation (Rumelhart)
  - 1989: convolutional neural networks (LeCun)
- Alternative theoretically motivated methods outperformed NNs
  - Exaggerated expectations: “It works like the brain” (No, it does not!)

# A Very Brief History of Neural Networks

- Neural networks have a [long history](#)
  - 1942: artificial neurons (McCulloch/Pitts)
  - 1958/1969: perceptron (Rosenblatt; Minsky/Papert)
  - 1986: multilayer perceptrons and backpropagation (Rumelhart)
  - 1989: convolutional neural networks (LeCun)
- Alternative theoretically motivated methods outperformed NNs
  - Exaggerated expectations: “It works like the brain” (No, it does not!)
- Why the sudden success of neural networks in the last 5 years?
  - [Data](#): Availability of massive amounts of labelled data
  - [Compute power](#): Ability to train very large neural networks on GPUs
  - [Methodological advances](#): many since first renewed popularization

# Lecture Overview

1 Motivation: Why is Deep Learning so Popular?

2 Representation Learning and Deep Learning

3 Multilayer Perceptrons

4 Optimization of Neural Networks in a Nutshell

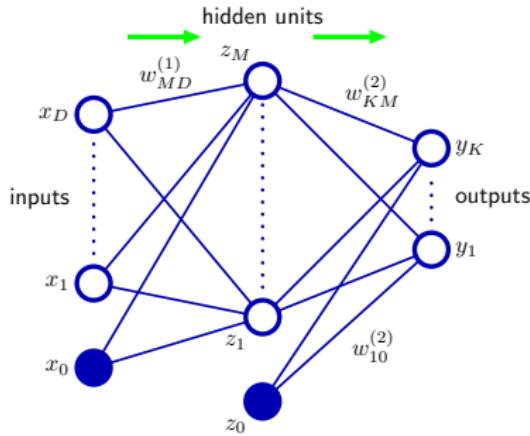
5 Overview of Some Advanced Topics

- Convolutional neural networks
- Recurrent neural networks
- Deep reinforcement learning

6 Limitations

7 Wrapup

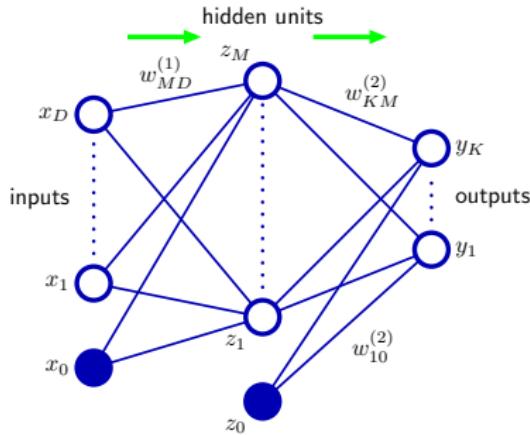
# Multilayer Perceptrons



[figure from Bishop, Ch. 5]

- Network is organized in **layers**
  - Outputs of  $k$ -th layer serve as inputs of  $k + 1$ th layer
- Each layer  $k$  only does quite simple computations:
  - Linear function of previous layer's outputs  $\mathbf{z}_{k-1}$ :  $\mathbf{a}_k = \mathbf{W}_k \mathbf{z}_{k-1} + \mathbf{b}_k$
  - Nonlinear transformation  $\mathbf{z}_k = h_k(\mathbf{a}_k)$  through **activation function**  $h_k$

# Multilayer Perceptrons



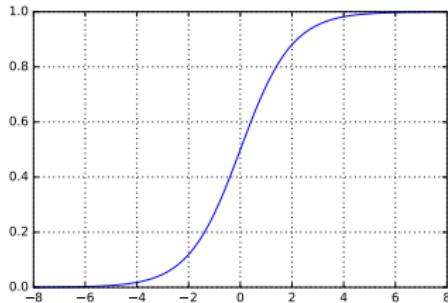
[figure from Bishop, Ch. 5]

- Network is organized in **layers**
  - Outputs of  $k$ -th layer serve as inputs of  $k + 1$ th layer
- Each layer  $k$  only does quite simple computations:
  - Linear function of previous layer's outputs  $\mathbf{z}_{k-1}$ :  $\mathbf{a}_k = \mathbf{W}_k \mathbf{z}_{k-1} + \mathbf{b}_k$
  - Nonlinear transformation  $\mathbf{z}_k = h_k(\mathbf{a}_k)$  through **activation function**  $h_k$
- **Parameters/weights w** of the network: all  $\mathbf{W}_k, \mathbf{b}_k$ , flattened into a single vector

# Activation Functions - Examples

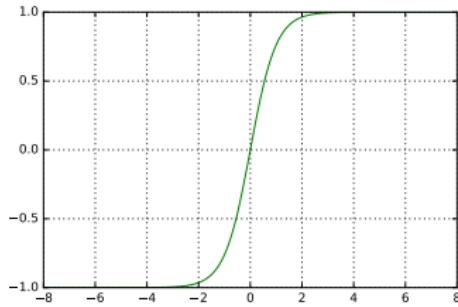
Logistic sigmoid activation function:

$$h_{logistic}(a) = \frac{1}{1 + \exp(-a)}$$



Logistic hyperbolic tangent activation function:

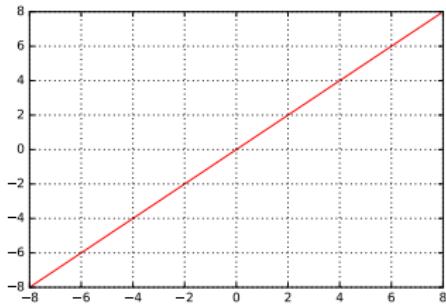
$$\begin{aligned} h_{tanh}(a) &= \tanh(a) \\ &= \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} \end{aligned}$$



# Activation Functions - Examples (cont.)

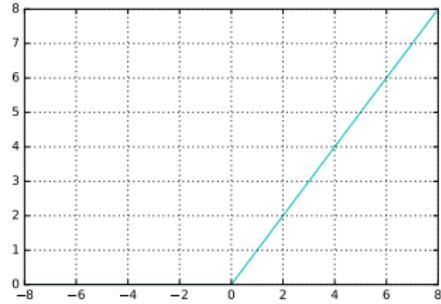
Linear activation function:

$$h_{linear}(a) = a$$



Rectified Linear (ReLU) activation function:

$$h_{relu}(a) = \max(0, a)$$



# Output layer and loss functions

- For regression:
  - Single output neuron with linear activation function

$$\hat{y}(\mathbf{x}, \mathbf{w}) = h_{linear}(a) = a$$

- Loss function: e.g., squared error:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{\hat{y}(\mathbf{x}_n, \mathbf{w}) - y_n\}^2$$

# Output layer and loss functions

- For regression:

- Single output neuron with linear activation function

$$\hat{y}(\mathbf{x}, \mathbf{w}) = h_{linear}(a) = a$$

- Loss function: e.g., squared error:

$$L(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{\hat{y}(\mathbf{x}_n, \mathbf{w}) - y_n\}^2$$

- For classification:

- Single output unit with, e.g., logistic activation function:

$$\hat{y}(\mathbf{x}, \mathbf{w}) = h_{logistic}(a) = \frac{1}{1 + \exp(-a)}$$

- Loss function: negative log likelihood of the data under the predictive distribution this specifies; (aka cross entropy):

$$L(\mathbf{w}) = - \sum_{n=1}^N \{y_n \ln \hat{y}_n + (1 - y_n) \ln(1 - \hat{y}_n)\}$$

# Optimizing a loss / error function

- Given training data  $\mathcal{D} = \langle (\mathbf{x}_i, y_i) \rangle_{i=1}^N$  and topology of an MLP
- Task: adapt weights  $w$  to minimize the loss:

$$\underset{\mathbf{w}}{\text{minimize}} \ L(\mathbf{w}; \mathcal{D})$$

# Optimizing a loss / error function

- Given training data  $\mathcal{D} = \langle (\mathbf{x}_i, y_i) \rangle_{i=1}^N$  and topology of an MLP
- Task: adapt weights  $w$  to minimize the loss:

$$\underset{\mathbf{w}}{\text{minimize}} \ L(\mathbf{w}; \mathcal{D})$$

- We optimize this function by gradient-based optimization
  - We can compute gradients of  $L(\mathbf{w}; \mathcal{D})$
  - Efficiently, using a technique called backpropagation

# Optimizing a loss / error function

- Given training data  $\mathcal{D} = \langle (\mathbf{x}_i, y_i) \rangle_{i=1}^N$  and topology of an MLP
- Task: adapt weights  $w$  to minimize the loss:

$$\underset{\mathbf{w}}{\text{minimize}} \ L(\mathbf{w}; \mathcal{D})$$

- We optimize this function by gradient-based optimization
  - We can compute gradients of  $L(\mathbf{w}; \mathcal{D})$ 
    - Efficiently, using a technique called backpropagation
  - Stochastic gradient descent (SGD)**
    - We can use small batches of the data, i.e.,  $L(\mathbf{w}; \mathcal{D}_{batch})$
    - This yields approximate gradients quickly

# Lecture Overview

1 Motivation: Why is Deep Learning so Popular?

2 Representation Learning and Deep Learning

3 Multilayer Perceptrons

4 Optimization of Neural Networks in a Nutshell

5 Overview of Some Advanced Topics

- Convolutional neural networks
- Recurrent neural networks
- Deep reinforcement learning

6 Limitations

7 Wrapup

# Optimization theory

- A **global minimum**  $\mathbf{u}^*$  is a point such that:

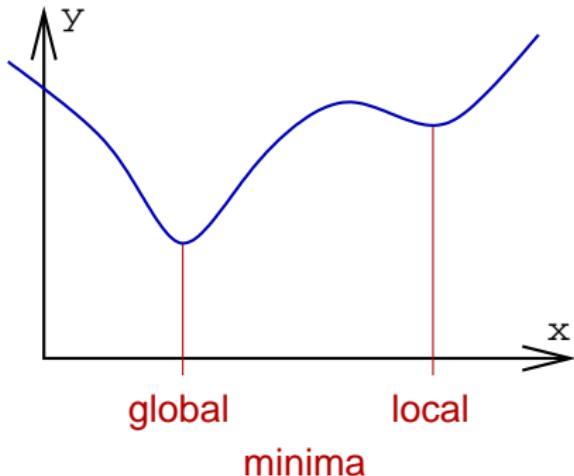
$$f(\mathbf{u}^*) \leq f(\mathbf{u})$$

for all  $\mathbf{u}$ .

- A **local minimum**  $\mathbf{u}^+$  is a point such that exist  $r > 0$  with

$$f(\mathbf{u}^+) \leq f(\mathbf{u})$$

for all points  $\mathbf{u}$  with  
 $\|\mathbf{u} - \mathbf{u}^+\| < r$



## Optimization theory (cont.)

- Analytical way to find a minimum:

For a local minimum  $\mathbf{u}^+$ , the gradient of  $f$  becomes zero:

$$\frac{\partial f}{\partial u_i}(\mathbf{u}^+) = 0 \quad \text{for all } i$$

Hence, calculating all partial derivatives and looking for zeros is a good idea

# Optimization theory (cont.)

- Analytical way to find a minimum:

For a local minimum  $\mathbf{u}^+$ , the gradient of  $f$  becomes zero:

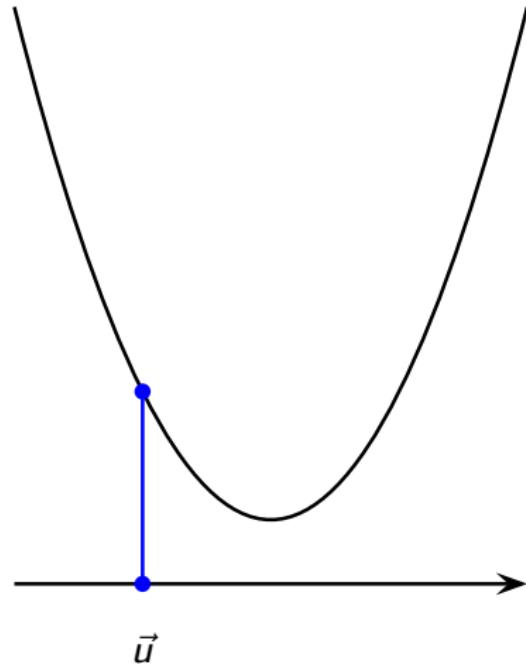
$$\frac{\partial f}{\partial u_i}(\mathbf{u}^+) = 0 \quad \text{for all } i$$

Hence, calculating all partial derivatives and looking for zeros is a good idea

- But: for neural networks, we can't write down a solution for the minimization problem in closed form
  - even though  $\frac{\partial f}{\partial u_i} = 0$  holds at (local) solution points
  - need to resort to iterative methods

# Optimization theory (cont.)

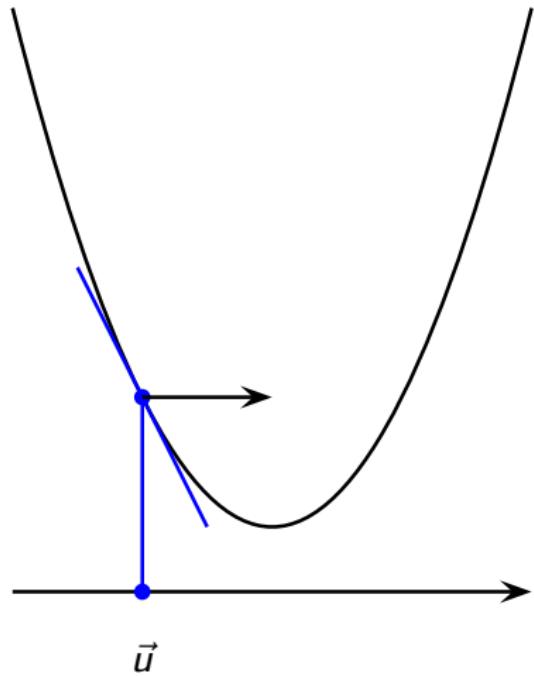
- Numerical way to find a minimum, searching:  
assume we start at point  $\mathbf{u}$ .  
Which is the best direction to search for a point  $\mathbf{v}$  with  
 $f(\mathbf{v}) < f(\mathbf{u})$  ?



# Optimization theory (cont.)

- Numerical way to find a minimum, searching:  
assume we start at point  $\mathbf{u}$ .

Which is the best direction to search for a point  $\mathbf{v}$  with  $f(\mathbf{v}) < f(\mathbf{u})$  ?

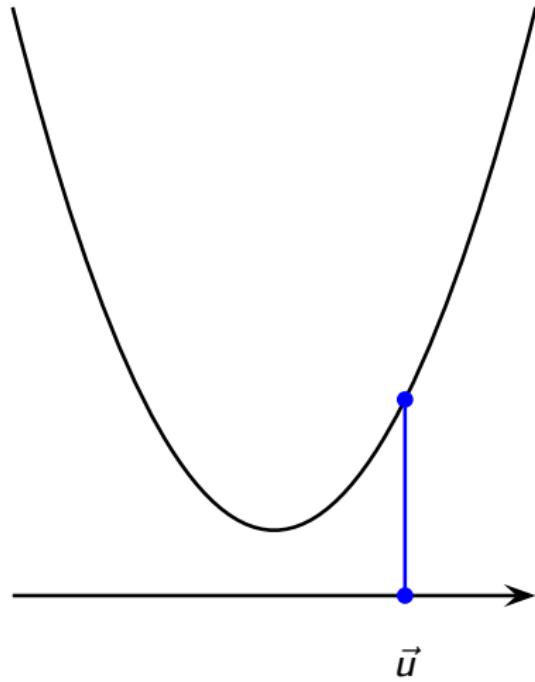


slope is negative (descending), go right!

# Optimization theory (cont.)

- Numerical way to find a minimum, searching:  
assume we start at point  $\mathbf{u}$ .

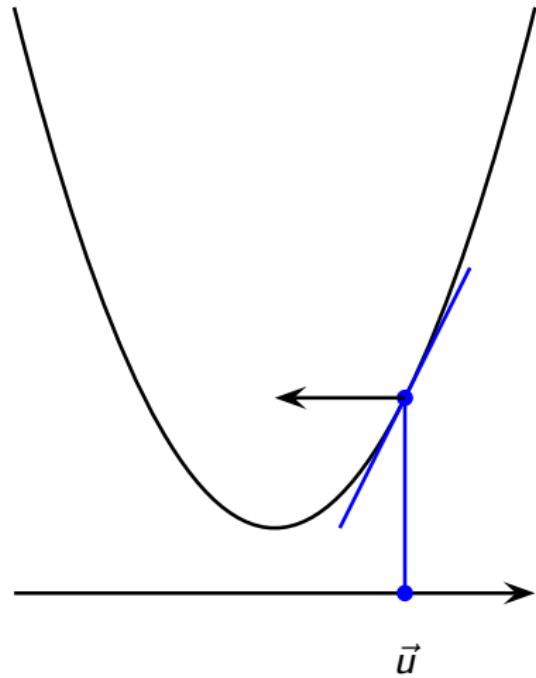
Which is the best direction to search for a point  $\mathbf{v}$  with  $f(\mathbf{v}) < f(\mathbf{u})$  ?



# Optimization theory (cont.)

- Numerical way to find a minimum, searching:  
assume we start at point  $\mathbf{u}$ .

Which is the best direction to search for a point  $\mathbf{v}$  with  $f(\mathbf{v}) < f(\mathbf{u})$  ?



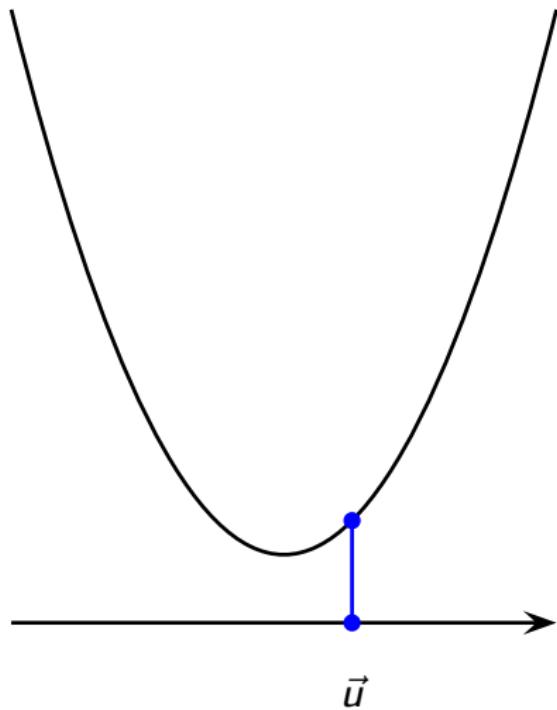
slope is positive (ascending), go left!

# Optimization theory (cont.)

- Numerical way to find a minimum, searching:  
assume we start at point  $\mathbf{u}$ .

Which is the best direction to search for a point  $\mathbf{v}$  with  $f(\mathbf{v}) < f(\mathbf{u})$  ?

Which is the best stepwidth?

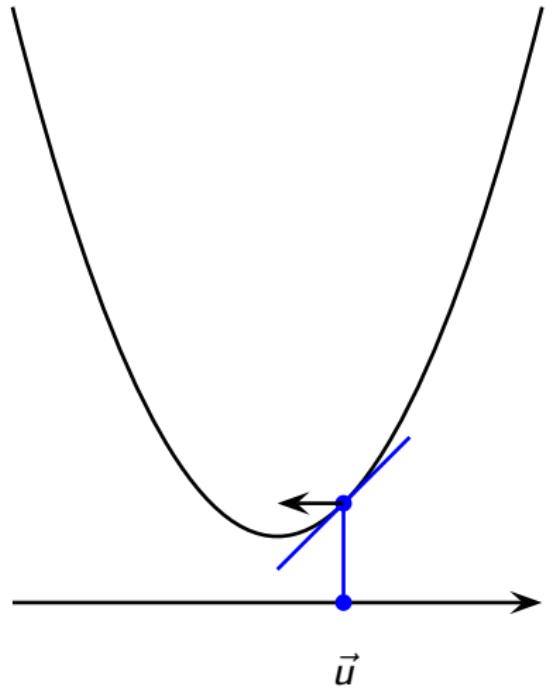


# Optimization theory (cont.)

- Numerical way to find a minimum, searching:  
assume we start at point  $\mathbf{u}$ .

Which is the best direction to search for a point  $\mathbf{v}$  with  $f(\mathbf{v}) < f(\mathbf{u})$  ?

Which is the best stepwidth?



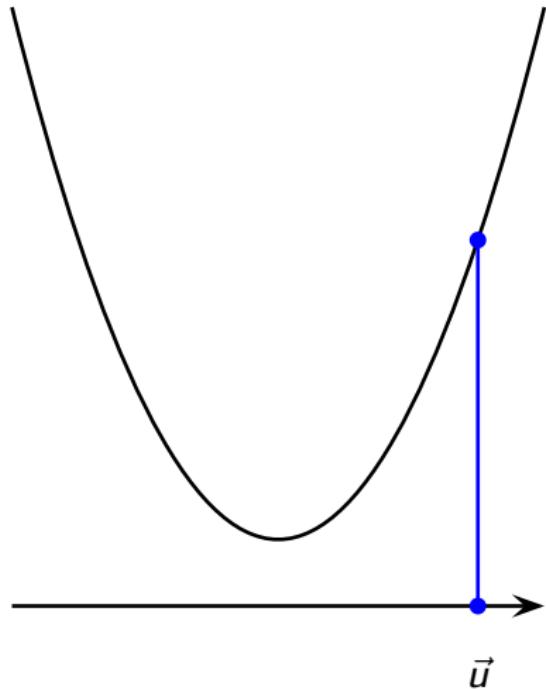
slope is small, small step!

# Optimization theory (cont.)

- Numerical way to find a minimum, searching:  
assume we start at point  $\mathbf{u}$ .

Which is the best direction to search for a point  $\mathbf{v}$  with  $f(\mathbf{v}) < f(\mathbf{u})$  ?

Which is the best stepwidth?

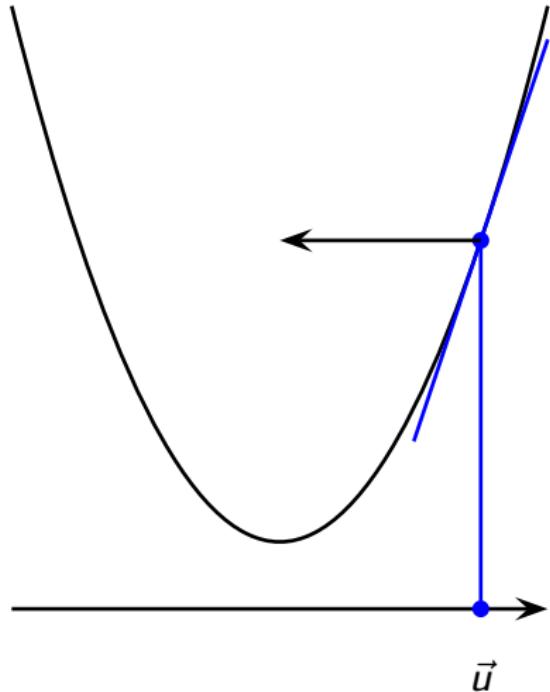


# Optimization theory (cont.)

- Numerical way to find a minimum, searching:  
assume we start at point  $\mathbf{u}$ .

Which is the best direction to search for a point  $\mathbf{v}$  with  $f(\mathbf{v}) < f(\mathbf{u})$  ?

Which is the best stepwidth?



slope is large, large step!

## Optimization theory (cont.)

- Numerical way to find a minimum, searching:  
assume we start at point  $\mathbf{u}$ .

Which is the best direction to search for a point  $\mathbf{v}$  with  $f(\mathbf{v}) < f(\mathbf{u})$  ?

Which is the best stepwidth?

- general principle:

$$v_i \leftarrow u_i - \epsilon \frac{\partial f}{\partial u_i}$$

$\epsilon > 0$  is called learning rate

# Gradient descent

- Gradient descent approach:

**Require:** mathematical function  $f$ , learning rate  $\epsilon > 0$

**Ensure:** returned vector is close to a local minimum of  $f$

- 1: choose an initial point  $\mathbf{u}$
- 2: **while**  $\|grad f(\mathbf{u})\|$  not close to 0 **do**
- 3:    $\mathbf{u} \leftarrow \mathbf{u} - \epsilon \cdot grad f(\mathbf{u})$
- 4: **end while**
- 5: **return**  $\mathbf{u}$

# Calculating partial derivatives

- Our typical loss functions are defined across data points:

$$L(\mathbf{w}) = \sum_{n=1}^N L_n(\mathbf{w}) = L(f(\mathbf{x}_n; \mathbf{w}), y_n)$$

# Calculating partial derivatives

- Our typical loss functions are defined across data points:

$$L(\mathbf{w}) = \sum_{n=1}^N L_n(\mathbf{w}) = L(f(\mathbf{x}_n; \mathbf{w}), y_n)$$

- We can compute their partial derivatives as a sum over data points:

$$\frac{\partial L}{\partial w_j} = \sum_{n=1}^N \frac{\partial L_n}{\partial w_j}$$

# Calculating partial derivatives

- Our typical loss functions are defined across data points:

$$L(\mathbf{w}) = \sum_{n=1}^N L_n(\mathbf{w}) = L(f(\mathbf{x}_n; \mathbf{w}), y_n)$$

- We can compute their partial derivatives as a sum over data points:

$$\frac{\partial L}{\partial w_j} = \sum_{n=1}^N \frac{\partial L_n}{\partial w_j}$$

- The method of **backpropagation** makes consistent use of the **chain rule** of calculus to compute the partial derivatives  $\frac{\partial L_n}{\partial w_j}$  w.r.t. each network weight  $w_j$ , **re-using previously computed results**
  - Backpropagation is not covered here, but, e.g., in ML lecture

Do we need gradients based on the entire data set?

- Using the entire set is referred to as **batch gradient descent**

# Do we need gradients based on the entire data set?

- Using the entire set is referred to as **batch gradient descent**
- Gradients get more accurate when based on more data points
  - But using more data has diminishing returns w.r.t reduction in error
  - Usually **faster progress** by **updating more often** based on cheaper, less accurate estimates of the gradient

# Do we need gradients based on the entire data set?

- Using the entire set is referred to as **batch gradient descent**
- Gradients get more accurate when based on more data points
  - But using more data has diminishing returns w.r.t reduction in error
  - Usually **faster progress** by **updating more often** based on cheaper, less accurate estimates of the gradient
- Common approach in practice: compute gradients over **mini-batches**
  - Mini-batch: small subset of the training data
  - Today, this is commonly called **stochastic gradient descent (SGD)**

# Stochastic gradient descent

- Stochastic gradient descent (SGD)

**Require:** mathematical function  $f$ , learning rate  $\epsilon > 0$

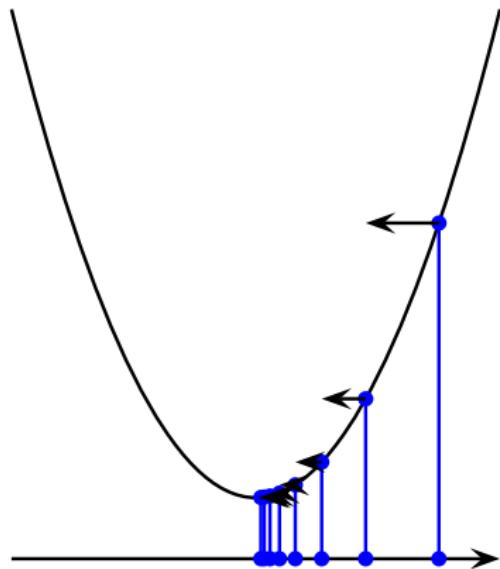
**Ensure:** returned vector is close to a local minimum of  $f$

- 1: choose an initial point  $\mathbf{w}$
- 2: **while** stopping criterion not met **do**
- 3:   Sample a minibatch of  $m$  examples  $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$  with corresponding targets  $\mathbf{y}^{(i)}$  from the training set
- 4:   Compute gradient  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\mathbf{w}} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}; \mathbf{w}), \mathbf{y}^{(i)})$
- 5:   Update parameter:  $\mathbf{w} \leftarrow \mathbf{w} - \epsilon \cdot \mathbf{g}$
- 6: **end while**
- 7: **return**  $\mathbf{w}$

- Note:  $\nabla_{\mathbf{w}} L := [\frac{\partial L}{\partial w_1}, \dots, \frac{\partial L}{\partial w_K}]$  for  $K$  weights

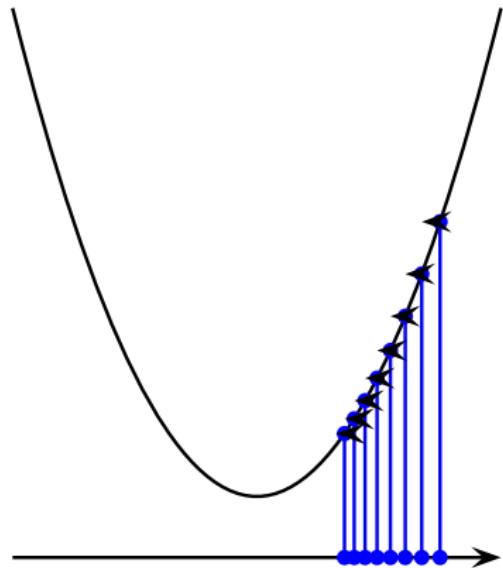
# Problems with suboptimal choices for learning rate

- choice of  $\epsilon$ 
  - case small  $\epsilon$ : convergence



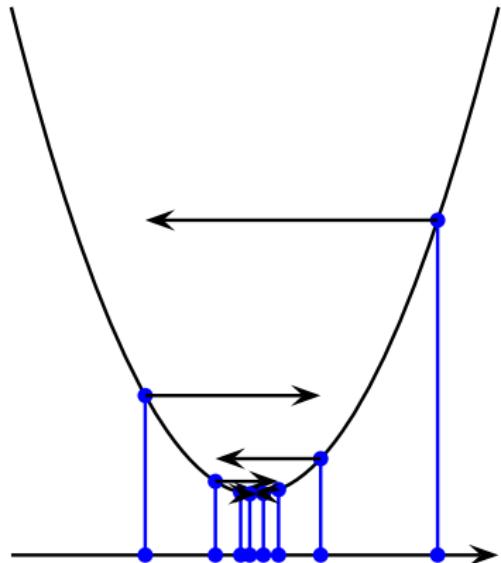
# Problems with suboptimal choices for learning rate

- choice of  $\epsilon$ 
  - 2. case very small  $\epsilon$ : convergence, but it may take very long



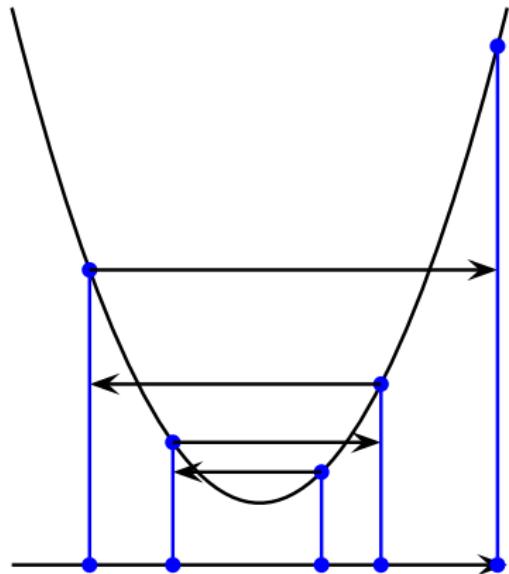
# Problems with suboptimal choices for learning rate

- choice of  $\epsilon$ 
  - 3. case medium size  $\epsilon$ : convergence



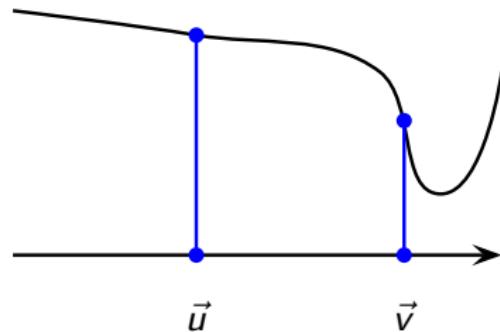
# Problems with suboptimal choices for learning rate

- choice of  $\epsilon$ 
  - 4. case large  $\epsilon$ : divergence



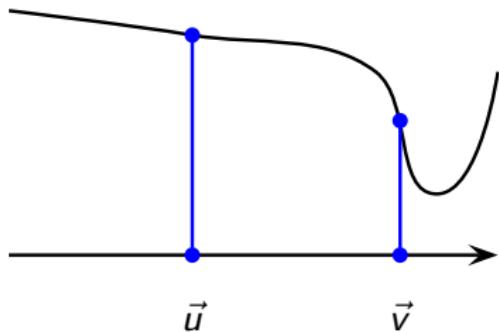
## Other reasons for problems with gradient descent

- flat spots and steep valleys:  
need larger  $\epsilon$  in  $\mathbf{u}$  to jump over  
the uninteresting flat area but  
need smaller  $\epsilon$  in  $\mathbf{v}$  to meet the  
minimum

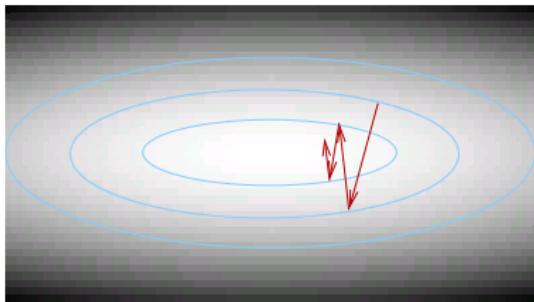


# Other reasons for problems with gradient descent

- flat spots and steep valleys:  
need larger  $\epsilon$  in  $\mathbf{u}$  to jump over  
the uninteresting flat area but  
need smaller  $\epsilon$  in  $\mathbf{v}$  to meet the  
minimum

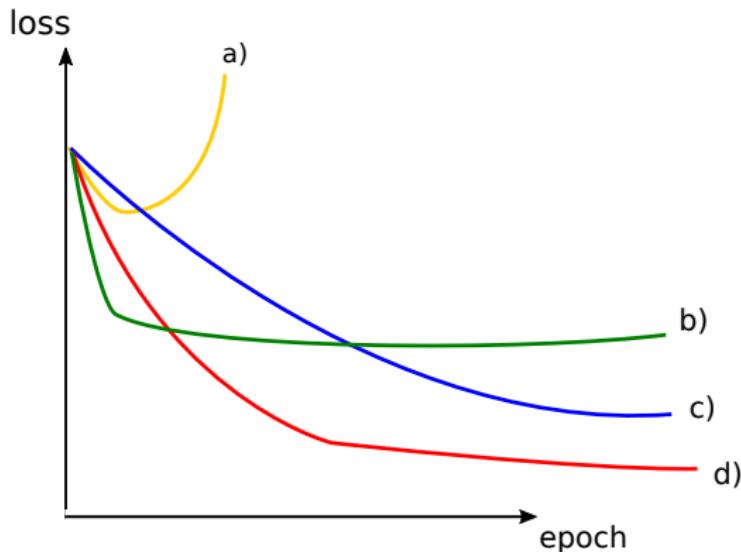


- zig-zagging:  
in higher dimensions:  $\epsilon$  is not  
appropriate for all dimensions



# Learning rate quiz

Which curve denotes low, high, very high, and good learning rate?



# Gradient descent – Conclusion

- Pure gradient descent is a nice framework
- In practice, stochastic gradient descent is used
- Finding the right learning rate  $\epsilon$  is tedious

# Gradient descent – Conclusion

- Pure gradient descent is a nice framework
- In practice, stochastic gradient descent is used
- Finding the right learning rate  $\epsilon$  is tedious

Heuristics to overcome problems of gradient descent:

- Gradient descent with **momentum**
- Individual learning rates for each dimension
- Adaptive learning rates
- Decoupling steplength from partial derivates

# Lecture Overview

1 Motivation: Why is Deep Learning so Popular?

2 Representation Learning and Deep Learning

3 Multilayer Perceptrons

4 Optimization of Neural Networks in a Nutshell

5 Overview of Some Advanced Topics

- Convolutional neural networks
- Recurrent neural networks
- Deep reinforcement learning

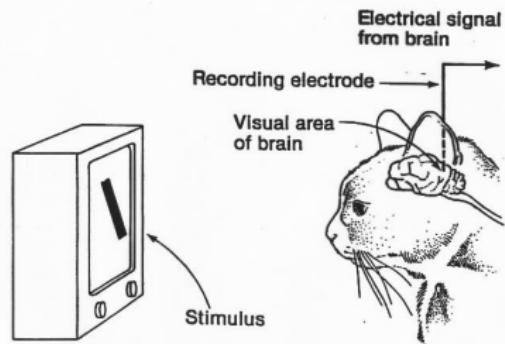
6 Limitations

7 Wrapup

# Historical context and inspiration from Neuroscience

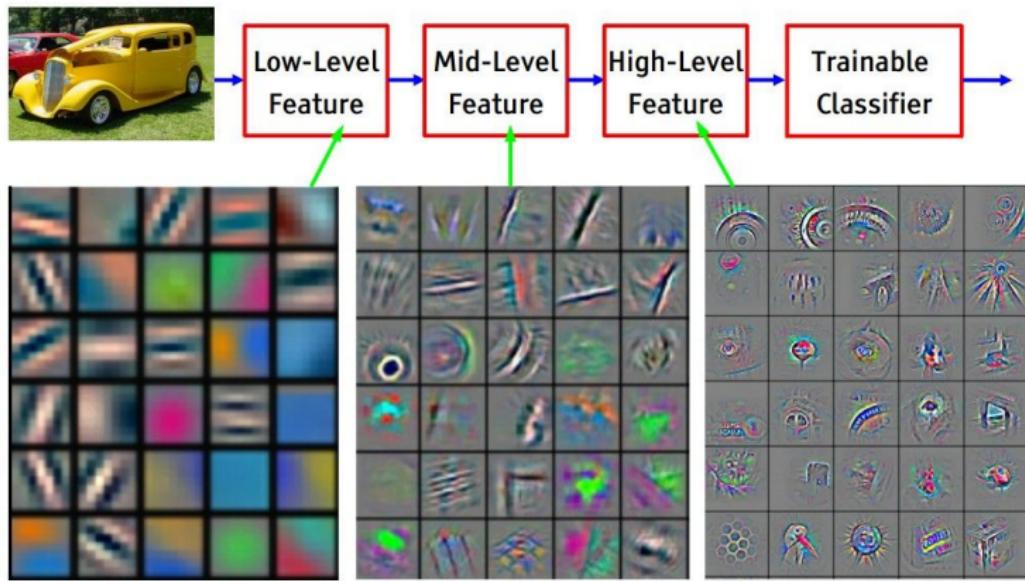
Hubel & Wiesel (Nobel prize 1981) found in several studies in the 1950s and 1960s:

- Visual cortex has feature detectors (e.g., cells with preference for edges with specific orientation)
  - edge **location** did not matter
- **Simple cells** as local feature detectors
- **Complex cells** pool responses of simple cells
- There is a **feature hierarchy**



# Learned feature hierarchy

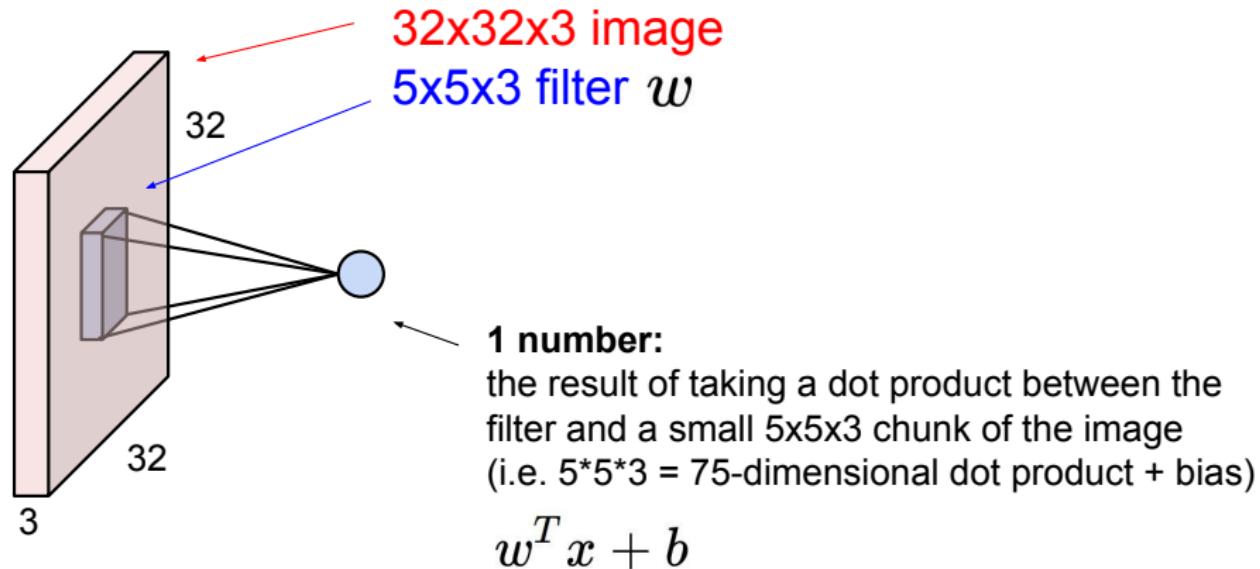
[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

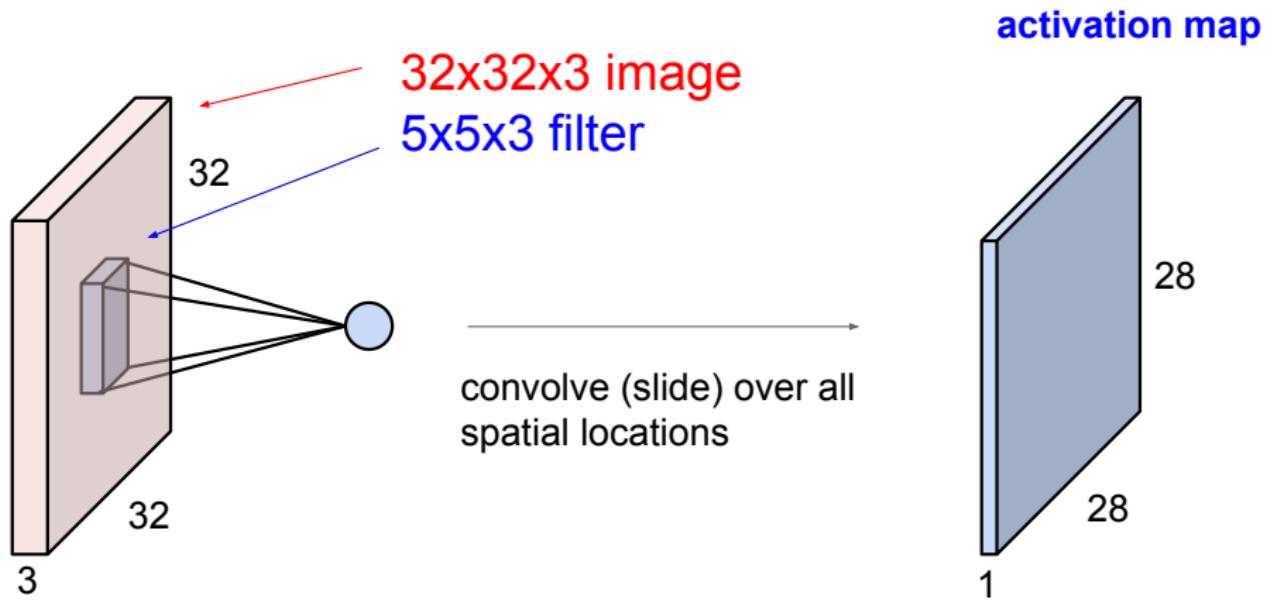
[slide credit: Andrej Karpathy]

# Convolutions illustrated



[slide credit: Andrej Karpathy]

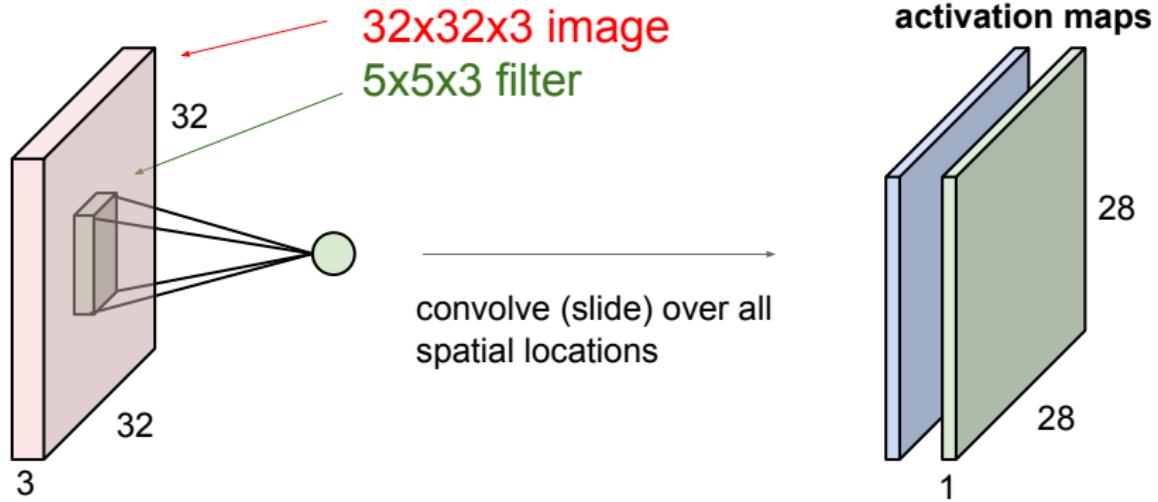
## Convolutions illustrated (cont.)



[slide credit: Andrej Karpathy]

# Convolutions – several filters

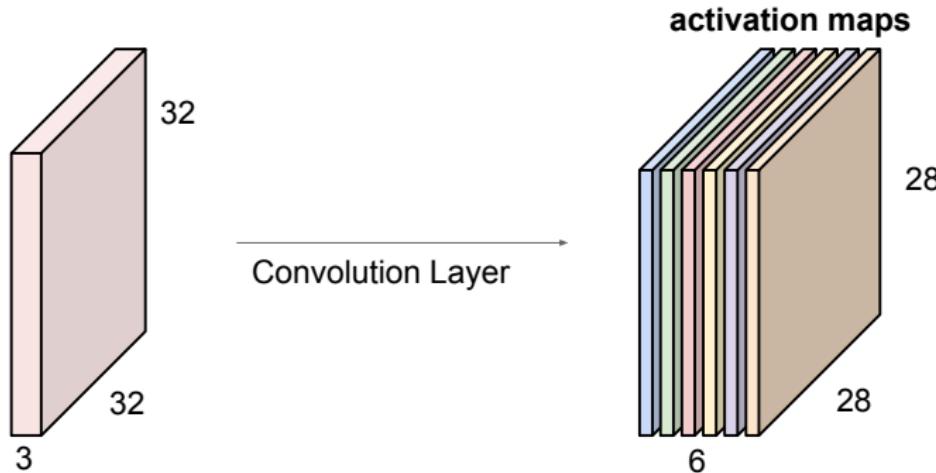
consider a second, green filter



[slide credit: Andrej Karpathy]

# Convolutions – several filters

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

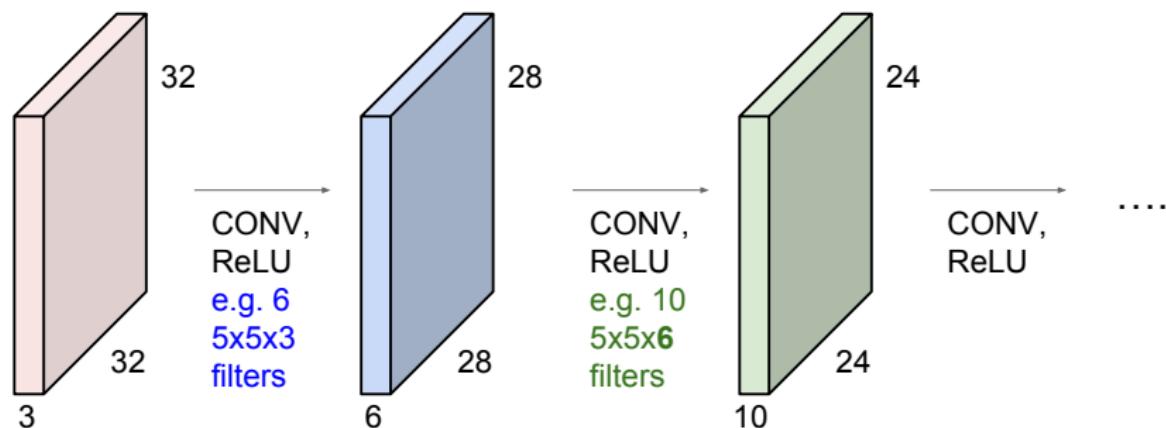


We stack these up to get a “new image” of size 28x28x6!

[slide credit: Andrej Karpathy]

# Stacking several convolutional layers

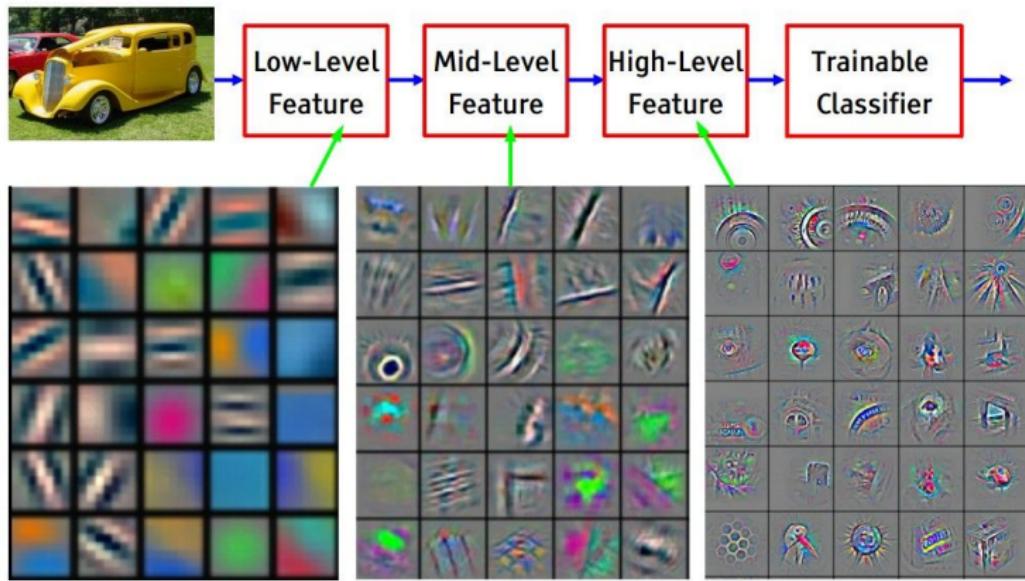
Convolutional layers stacked in a ConvNet



[slide credit: Andrej Karpathy]

# Learned feature hierarchy

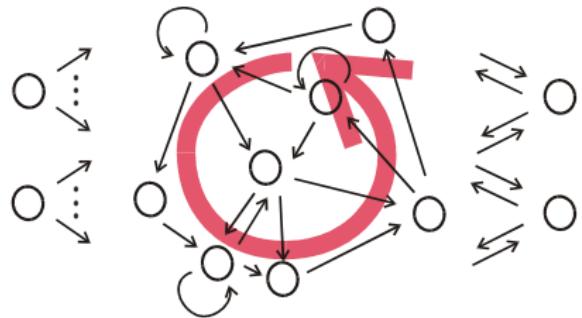
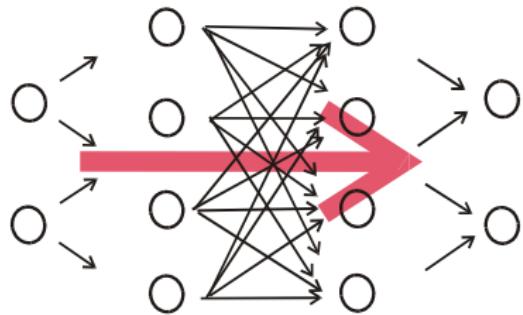
[From recent Yann LeCun slides]



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

[slide credit: Andrej Karpathy]

# Feedforward vs Recurrent Neural Networks



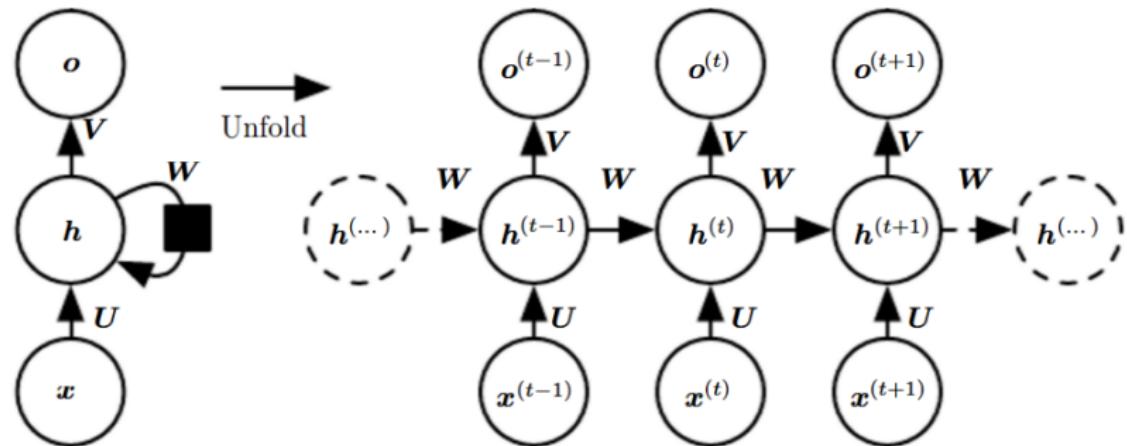
[Source: Jaeger, 2001]

# Recurrent Neural Networks (RNNs)

- Neural Networks that allow for **cycles** in the connectivity graph
- Cycles let information persist in the network for some time (state), and provide a **time-context** or (fading) memory
- Very powerful for processing **sequences**
- Implement **dynamical systems** rather than function mappings, and can approximate any dynamical system with arbitrary precision
- They are **Turing-complete** [Siegelmann and Sontag, 1991]

# Abstract schematic

With fully connected hidden layer:



[Goodfellow et al'2016]

# Sequence to sequence mapping

one to many

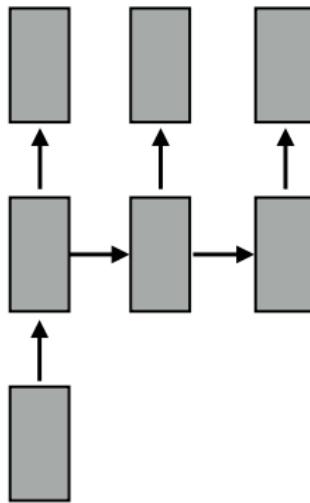
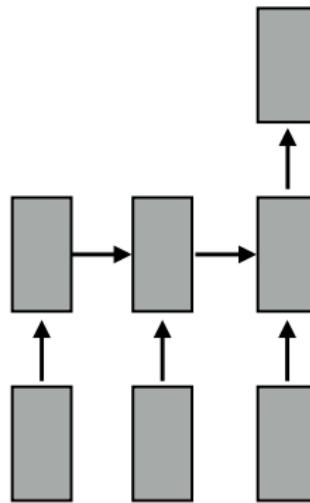


image caption  
generation

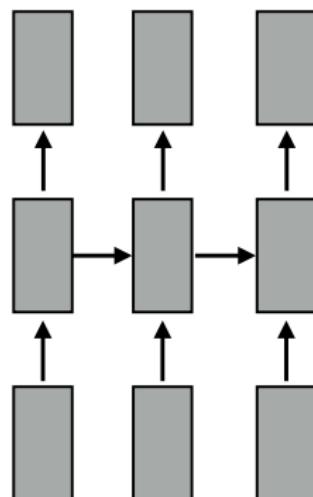
many to one



temporal  
classification

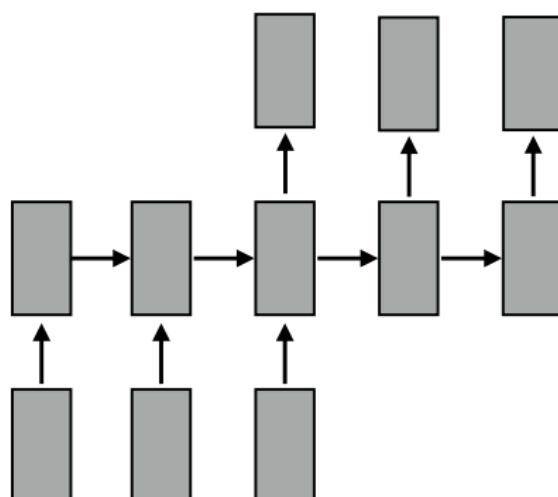
## Sequence to sequence mapping (cont.)

many to many



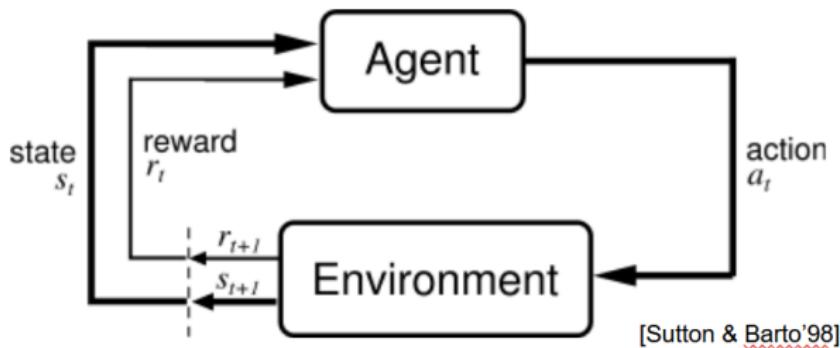
video  
frame labeling

many to many



automatic  
translation

# Reinforcement Learning



- Finding optimal policies for MDPs
- Reminder: states  $s \in S$ , actions  $a \in A$ , transition model  $T$ , rewards  $r$
- Policy: complete mapping  $\pi : S \rightarrow A$  that specifies for each state  $s$  which action  $\pi(s)$  to take

# Deep Reinforcement Learning

- Policy-based deep RL

- Represent policy  $\pi : S \rightarrow A$  as a deep neural network with weights  $w$
- Evaluate  $w$  by “rolling out” the policy defined by  $w$
- Optimize weights to obtain higher rewards (using approx. gradients)
- Examples: AlphaGo & modern Atari agents

# Deep Reinforcement Learning

- Policy-based deep RL

- Represent policy  $\pi : S \rightarrow A$  as a deep neural network with weights  $w$
- Evaluate  $w$  by “rolling out” the policy defined by  $w$
- Optimize weights to obtain higher rewards (using approx. gradients)
- Examples: AlphaGo & modern Atari agents

- Value-based deep RL

- Basically value iteration, but using a deep neural network (= function approximator) to generalize across many states and actions
- Approximate optimal state-value function  $U(s)$  or state-action value function  $Q(s, a)$

# Deep Reinforcement Learning

- Policy-based deep RL

- Represent policy  $\pi : S \rightarrow A$  as a deep neural network with weights  $w$
- Evaluate  $w$  by “rolling out” the policy defined by  $w$
- Optimize weights to obtain higher rewards (using approx. gradients)
- Examples: AlphaGo & modern Atari agents

- Value-based deep RL

- Basically value iteration, but using a deep neural network (= function approximator) to generalize across many states and actions
- Approximate optimal state-value function  $U(s)$  or state-action value function  $Q(s, a)$

- Model-based deep RL

- If transition model  $T$  is not known
- Approximate  $T$  with a deep neural network (learned from data)
- Plan using this approximate transition model

# Deep Reinforcement Learning

- Policy-based deep RL

- Represent policy  $\pi : S \rightarrow A$  as a deep neural network with weights  $w$
- Evaluate  $w$  by “rolling out” the policy defined by  $w$
- Optimize weights to obtain higher rewards (using approx. gradients)
- Examples: AlphaGo & modern Atari agents

- Value-based deep RL

- Basically value iteration, but using a deep neural network (= function approximator) to generalize across many states and actions
- Approximate optimal state-value function  $U(s)$  or state-action value function  $Q(s, a)$

- Model-based deep RL

- If transition model  $T$  is not known
- Approximate  $T$  with a deep neural network (learned from data)
- Plan using this approximate transition model

→ Use deep neural networks to represent policy / value function / model

# Lecture Overview

1 Motivation: Why is Deep Learning so Popular?

2 Representation Learning and Deep Learning

3 Multilayer Perceptrons

4 Optimization of Neural Networks in a Nutshell

5 Overview of Some Advanced Topics

- Convolutional neural networks
- Recurrent neural networks
- Deep reinforcement learning

6 Limitations

7 Wrapup

# Deep Learning Focuses on Perception

- Excellent results for perception tasks from raw data
  - Computer vision (from raw pixels)
  - Speech recognition (from raw audio)
  - Text recognition (from raw characters)
  - ...

# Deep Learning Focuses on Perception

- Excellent results for perception tasks from raw data
  - Computer vision (from raw pixels)
  - Speech recognition (from raw audio)
  - Text recognition (from raw characters)
  - ...
- But all of this is bottom-up
  - No top-down reasoning
  - No logic, planning, etc.
  - Although there are some modern works on [memory mechanisms](#), [attention](#), etc.

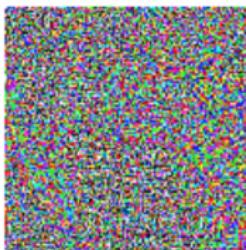
# Deep Learning Focuses on Perception

- Excellent results for perception tasks from raw data
  - Computer vision (from raw pixels)
  - Speech recognition (from raw audio)
  - Text recognition (from raw characters)
  - ...
- But all of this is bottom-up
  - No top-down reasoning
  - No logic, planning, etc.
  - Although there are some modern works on [memory mechanisms](#), [attention](#), etc.
- Deep networks can be combined with more traditional methods
  - E.g., AlphaGo: combination with Monte Carlo Tree Search (MCTS)
  - Some work on combining logic with deep learning

# Adversarial examples: we're very far from human-level performance



+ .007 ×



=



$\mathbf{x}$

$y = \text{"panda"}$   
w/ 57.7%  
confidence

$\text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y))$

"nematode"  
w/ 8.2%  
confidence

$\mathbf{x} + \epsilon \text{sign}(\nabla_{\mathbf{x}} J(\boldsymbol{\theta}, \mathbf{x}, y))$

"gibbon"  
w/ 99.3%  
confidence

- Even for very strong networks we can find adversarial examples
  - By following the gradient of the cost function w.r.t the input

# Lecture Overview

1 Motivation: Why is Deep Learning so Popular?

2 Representation Learning and Deep Learning

3 Multilayer Perceptrons

4 Optimization of Neural Networks in a Nutshell

5 Overview of Some Advanced Topics

- Convolutional neural networks
- Recurrent neural networks
- Deep reinforcement learning

6 Limitations

7 Wrapup

# Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: **not a silver bullet**

# Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: **not a silver bullet**
- Analogy to the ways humans process information
  - mostly tangential

# Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: **not a silver bullet**
- Analogy to the ways humans process information
  - mostly tangential
- Allows end-to-end learning
  - no more need for many complicated subsystems
  - e.g., dramatically simplified Google's translation pipeline

# Summary: Why is Deep Learning so Popular?

- Excellent empirical results in many domains
  - very scalable to big data
  - but beware: **not a silver bullet**
- Analogy to the ways humans process information
  - mostly tangential
- Allows end-to-end learning
  - no more need for many complicated subsystems
  - e.g., dramatically simplified Google's translation pipeline
- Very versatile/flexible
  - easy to combine building blocks
  - allows supervised, unsupervised, and reinforcement learning

# Lots of Work on Deep Learning in Freiburg

- Computer Vision (Thomas Brox)
    - Images, video
  - Robotics (Wolfram Burgard)
    - Navigation, grasping, object recognition
  - Neurorobotics (Joschka Boedecker)
    - Robotic control
  - Machine Learning (Frank Hutter)
    - Foundations: optimization, neural architecture search, learning to learn
  - Neuroscience (Tonio Ball, Michael Tangermann, and others )
    - EEG data and other applications from BrainLinks-BrainTools
- Details when the individual groups present their research

# Summary by learning goals

Having heard this lecture, you can now ...

- Explain the terms **representation learning** and **deep learning**
- Explain why deep learning is so popular
- Describe the main principles behind **MLPs**
- Discuss some **limitations** of deep learning
- On a high level, describe
  - Convolutional Neural Networks
  - Recurrent Neural Networks
  - Deep Reinforcement Learning

# Lecture 16: Clustering

Machine Learning, Summer Term 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



# Lecture Overview

1 Motivation

2 Criteria for Clustering

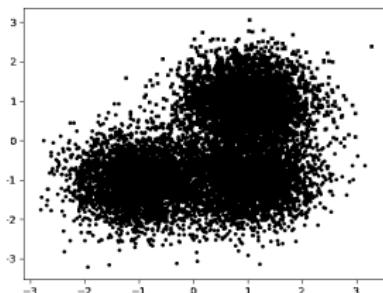
3 K-Means

4 DBSCAN

# What is Cluster Analysis?

Also called: clustering, segmentation analysis, taxonomy analysis, automatic classification, numerical taxonomy, botryology, typological analysis, community detection, ...

(Plot modified from scikit-learn clustering tutorial)



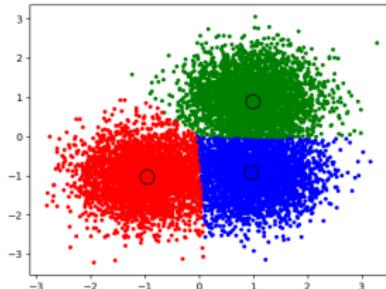
Unsupervised task:

Group objects such that objects within a group are more similar/related to each other (*in some sense*) than to objects of another group.

# What is Cluster Analysis?

Also called: clustering, segmentation analysis, taxonomy analysis, automatic classification, numerical taxonomy, botryology, typological analysis, community detection, ...

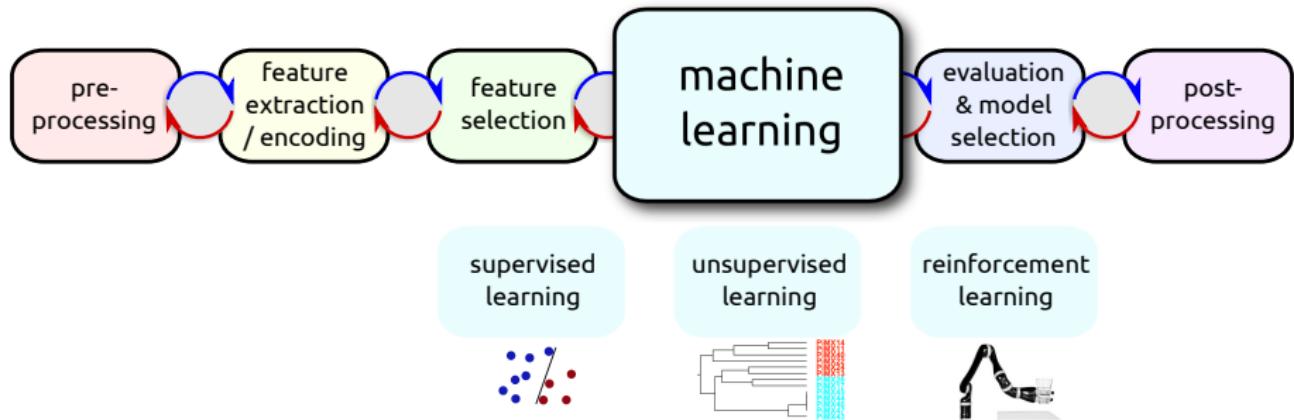
(Plot modified from scikit-learn clustering tutorial)



Unsupervised task:

Group objects such that objects within a group are more similar/related to each other (*in some sense*) than to objects of another group.

# ML Design Cycle



Cluster analysis is an unsupervised learning task

- Ground truth about clusters is not provided → evaluation is tricky!

Given:

- $N$  high dimensional data points  $\mathbf{x}_i \in \mathbb{R}^D$  with  $i = 1 \dots N$ .
- Data is collected in matrix  $\mathbf{X} \in \mathbb{R}^{N \times D}$

# Applications



# Example Applications (I)

- Medical imaging (fMRI, CT, PET): differentiate between different types of tissues, find tissue boundaries
- Biology: determine communities of organisms in space and time, compute data-driven phylogenetic trees
- Genetics: group DNA sequences into gene families
- Biochemistry / chemistry / pharmacology: group compounds according to their reaction mechanism
- Market research: detect clusters of customers with similar behavior, find market segments
- Social networks: recognize communities
- Search engines: Post-processing of search results into groups of hits that refer to vastly different topics

## Example Applications (II)

- Image segmentation: border detection, track objects
- Anomaly detection: identify outliers in data streams, network attacks, misbehaving software, sensor failures (robotics, production lines), predictive maintenance
- Finance: find stock clusters of similar behaviour
- Text analysis: clustering of documents into topics
- ...

# Lecture Overview

1 Motivation

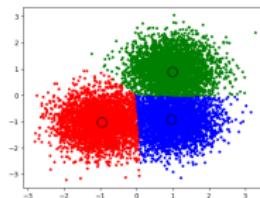
2 Criteria for Clustering

3 K-Means

4 DBSCAN

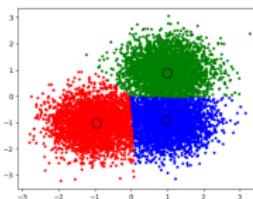
# How could Clusters be Determined?

Which metrics might be used to define clusters?



# How could Clusters be Determined?

Which metrics might be used to define clusters?



Zoo of clustering methods available, that exploit e.g.:

- distance/similarity function (between cluster members, between members of different clusters)
- connectivity structure using distances → single/avg/max linkage clustering, graph-based → clique
- centroid + neighborhood
- densities
- expected distributions

# Lecture Overview

1 Motivation

2 Criteria for Clustering

3 K-Means

4 DBSCAN

# K-Means Clustering (Steinhaus, 1957)

Find set  $C = C_1, \dots, C_k$  of  $k$  clusters represented by cluster centroids  $\mu_k$  such, that the clusters have equal variance.

→ Minimize the *inertia* or *within-cluster sum-of-squares* criterion:

$$\operatorname{argmin}_C \sum_{i=1}^k \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mu_j\|^2$$

Observations:

- Cluster centroids  $\mu_j$  do not need to be points of the training data sets
- Unfortunately NP-hard problem!
- Clustering can be represented by Voronoi tessellation

# K-Means Clustering

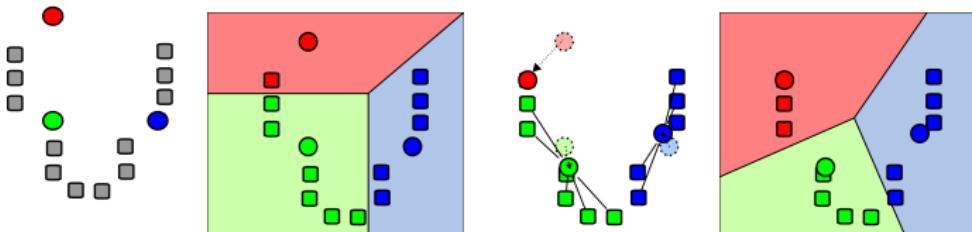
Practical solution by heuristic approximation

(e.g. Lloyd's algorithm (1957, 1982), similar to expectation-maximization):

Initialize  $k$  data points as cluster centroids.

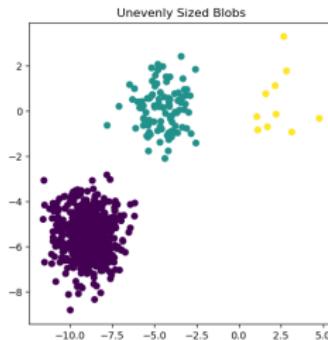
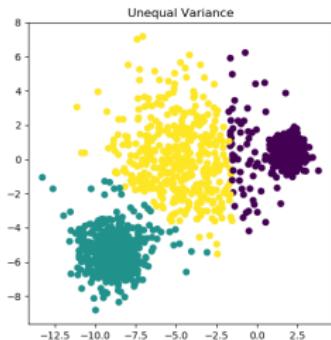
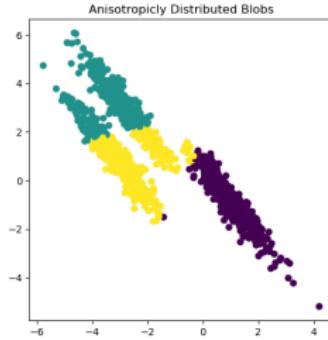
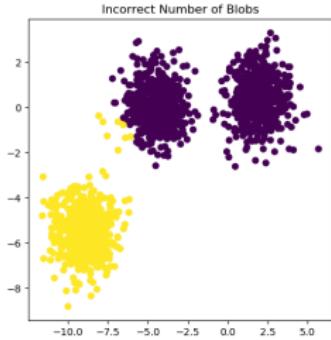
Then iterate these two steps until convergence of the centroids:

- ① Assign each data point to its nearest centroid  
(→ approximations necessary for high dimensions!)
- ② Create  $k$  new centroids by taking the mean value of all of the data points assigned to each novel centroid  
(→ approximations necessary for many data points)



# K-Means Clustering

Problematic data sets for k-means:



# K-Means Clustering

## Pros:

- conceptually simple algorithm
- mini-batches and different kind of initialization strategies are available  
(→ [k-means++](#))
- scales to many data points (if approximations are utilized)

## Cons:

- sensitive to initialization of centroids
- can not model noise or outliers
- concave cluster shapes are problematic
- can not deal with uneven variance between clusters
- number  $k$  of clusters needs to be provided

# Lecture Overview

1 Motivation

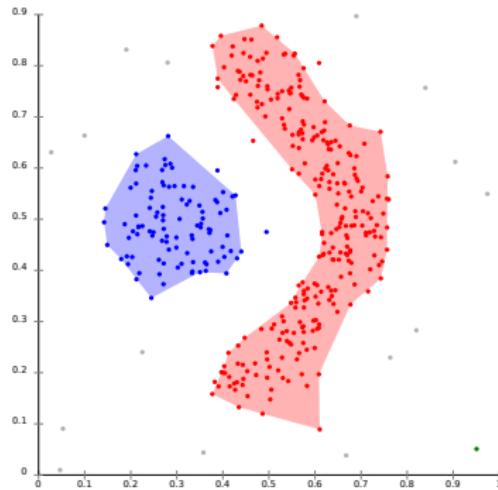
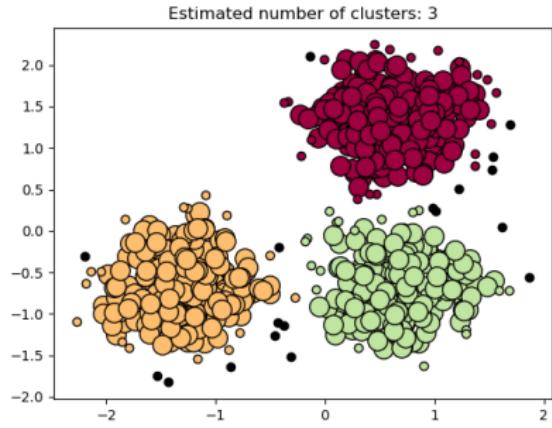
2 Criteria for Clustering

3 K-Means

4 DBSCAN

# DBSCAN

- DBSCAN (Ester et al., 1996) is a density-based, non-parameteric clustering method.
- It received the **test of time award** at KDD conference in 2014.

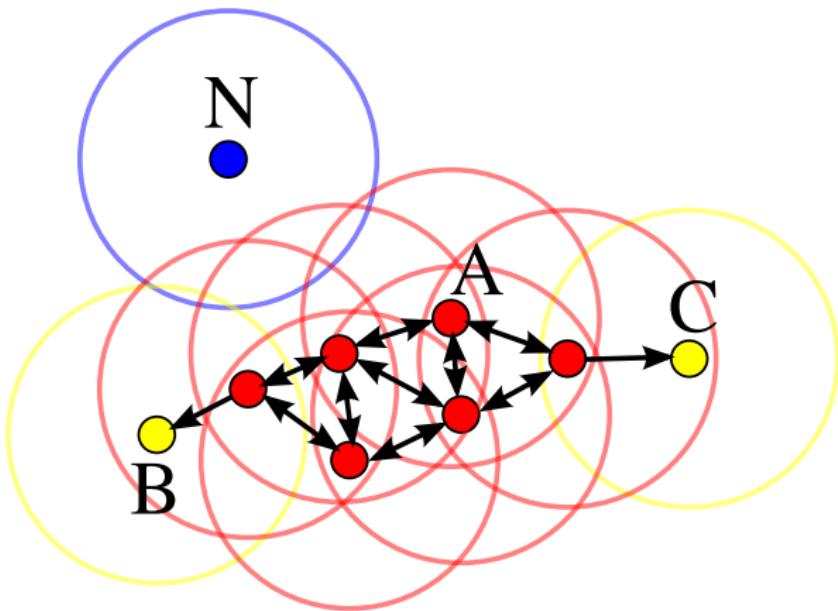


# DBSCAN: A Cluster has High Density

# DBSCAN: A Cluster has High Density

- A **core point**  $p$  is a data point in an area of high density. It is defined by having at least `minPts` points (including  $p$ ) within an `eps`-neighborhood.
- A point  $q$  is **directly reachable** from  $p$ , if  $p$  is in the `eps` neighborhood of a core point  $q$ .
- A point  $q$  is **reachable** from  $p$  if there is a path along the points  $p_1, \dots, p_n$  with  $p_1 = p$  and  $p_n = q$ , where each  $p_{i+1}$  is directly reachable from  $p_i$ . Note that this implies that all points on the path must be core points, with the possible exception of  $q$  (if  $q$  is on the fringe of a cluster).
- All points not reachable from any other point are considered **outliers** or **noise** points.

If  $p$  is a core point, then it forms a cluster together with all points (core or non-core) that are reachable from it.



By Chire - Own work, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=17045963>

Example with  $\text{minPts}=4$ . The  $\text{eps}$ -neighborhoods are indicated by circles.

# DBSCAN Algorithm

DBSCAN creates clusters by sequentially considering the training data points, starting with an arbitrary point:

- ① Find the points in the `eps` neighborhood of every point, and identify the core points with more than `minPts` neighbors.
- ② Find the connected components of core points on the neighbor graph, ignoring all non-core points.
- ③ Assign each non-core point to a nearby cluster if the cluster is an `eps` neighbor, otherwise assign it to noise.

# Pseudocode for DBSCAN

```
DBSCAN(DB, distFunc, eps, minPts) {
    C = 0
    for each point P in database DB {
        if label(P) ≠ undefined then continue
        Neighbors N = RangeQuery(DB, distFunc, P, eps)
        if |N| < minPts then {
            label(P) = Noise
            continue
        }
        C = C + 1
        label(P) = C
        Seed set S = N \ {P}
        for each point Q in S {
            if label(Q) = Noise then label(Q) = C
            if label(Q) ≠ undefined then continue
            label(Q) = C
            Neighbors N = RangeQuery(DB, distFunc, Q, eps)
            if |N| ≥ minPts then {
                S = S ∪ N
            }
        }
    }
}

RangeQuery(DB, distFunc, Q, eps) {
    Neighbors = empty list
    for each point P in database DB {
        if distFunc(Q, P) ≤ eps then {
            Neighbors = Neighbors ∪ {P}
        }
    }
    return Neighbors
} /* Cluster counter */
/* Previously processed in inner loop */
/* Find neighbors */
/* Density check */
/* Label as Noise */
/* next cluster label */
/* Label initial point */
/* Neighbors to expand */
/* Process every seed point */
/* Change Noise to border point */
/* Previously processed */
/* Label neighbor */
/* Find neighbors */
/* Density check */
/* Add new neighbors to seed set */
/* Scan all points in the database */
/* Compute distance and check epsilon */
/* Add to result */
```

# Characteristics of DBSCAN

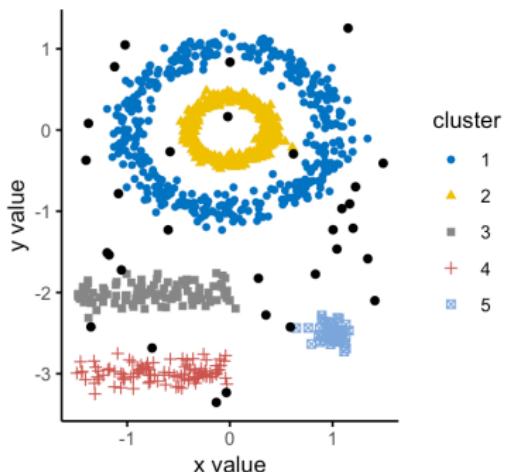
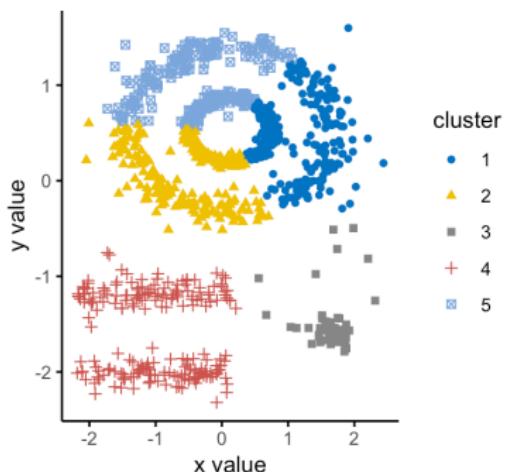
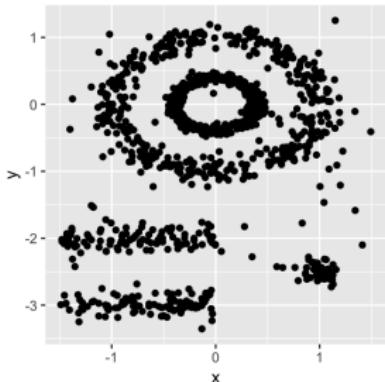
## Pros:

- DBSCAN is fast! With  $n$  data points:  $O(n^2)$  as worst case (all points belong to single cluster), but  $O(n \log(n))$  with good data structures and for typical data.
- DBSCAN is deterministic for a fixed processing sequence.
- Number of clusters is determined automatically.
- Hyperparameter `min samples` can express prior knowledge about noise.
- Different distance metrics can be utilized.
- Hierarchical variant HDBSCAN is available.

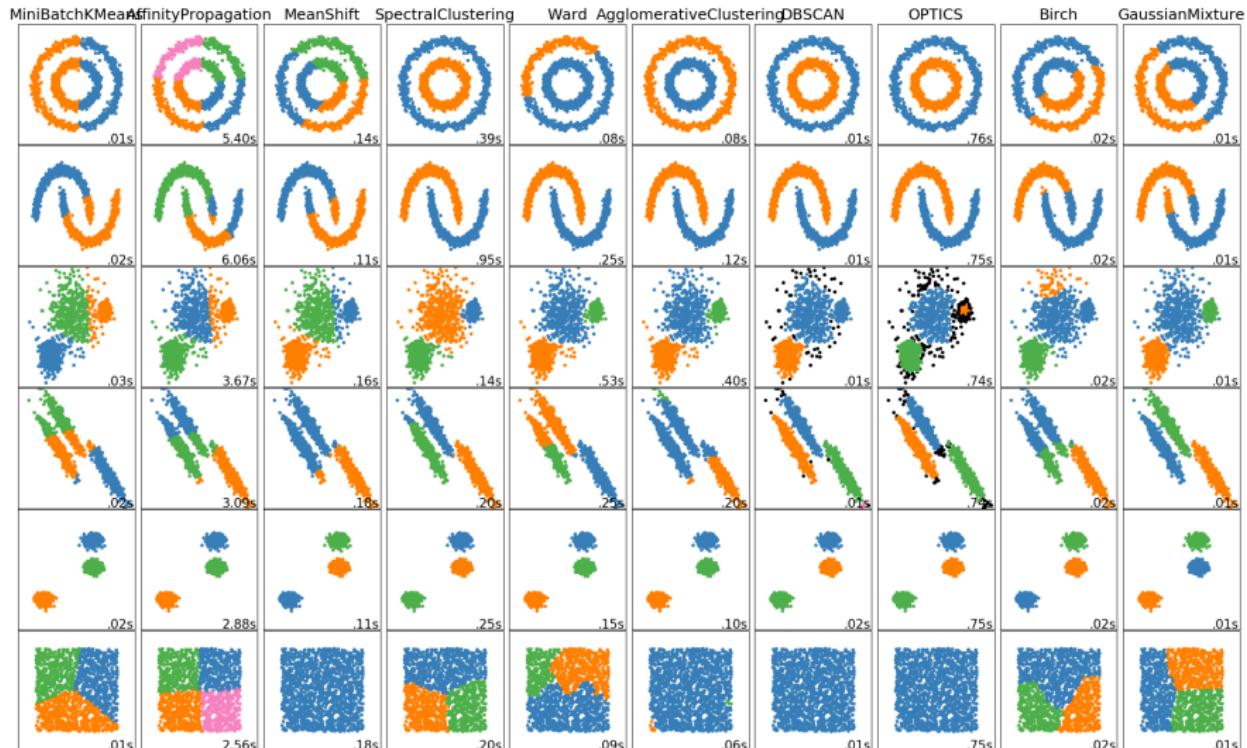
## Cons:

- Varying the sequence of data points processed can lead to different clusterings.
- Hyperparameter `eps` is critical, no good default!

# Comparison of K-Means and DBSCAN

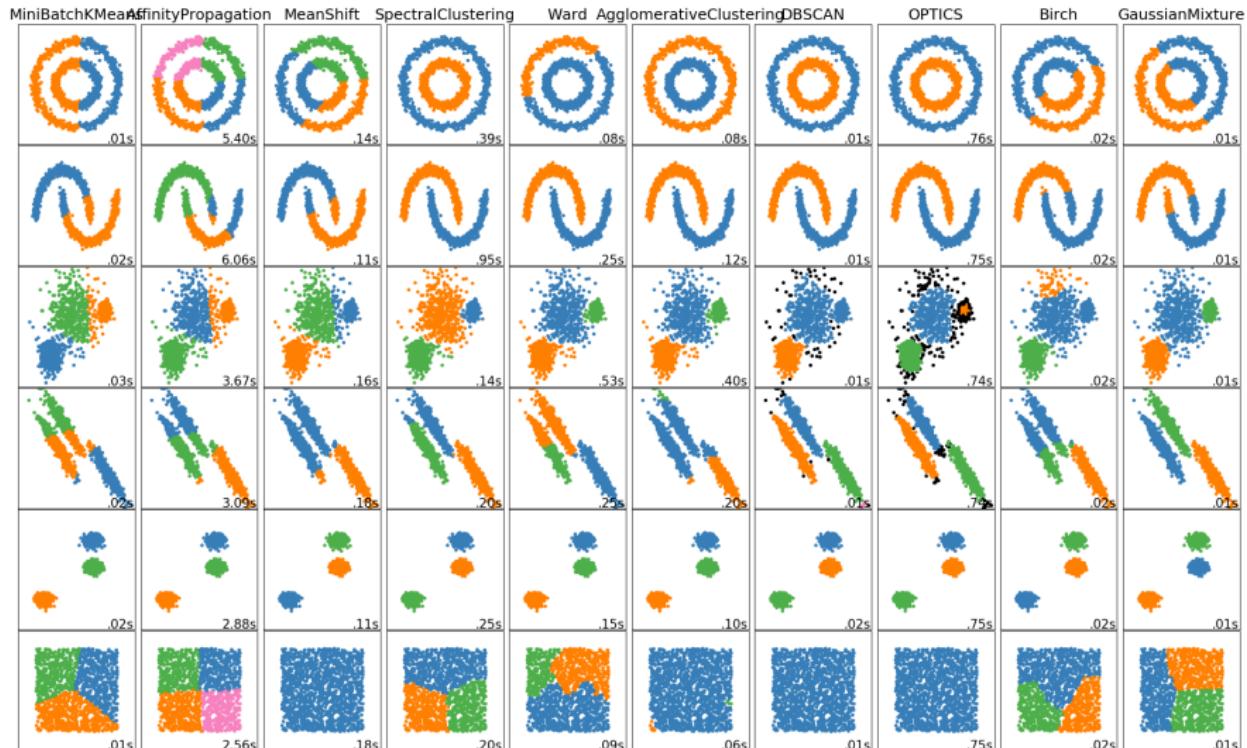


# A few Clustering Algorithms on Toy Data



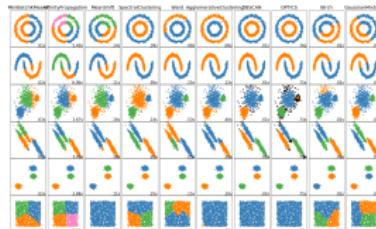
<https://scikit-learn.org/stable/modules/clustering.html>

# A few Clustering Algorithms on Toy Data



<https://scikit-learn.org/stable/modules/clustering.html>

# Criteria for the Choice of Clustering Algorithms



Does the algorithm...

- expects each cluster to follow a specific distribution? (e.g. Gaussian)?
- considers density of data points?
- deal well with noisy data / high-dimensional data / redundant dimensions / irrelevant dimensions?
- deliver hard / soft clustering?
- deliver a strict partitioning (i.e. each object belongs to exactly one cluster)?
- deliver a hierarchical clustering?

## Wrap-Up: Summary by Learning Goals

Having heard this lecture and doing the assignment on clustering, you will be able to:

- Explain, which metrics can be used to create a clustering from unlabeled data
- formulate the optimization problem for k-means clustering and implement an iterative heuristic
- Describe pros and cons of k-means and DBSCAN
- Derive a metric for the quality of a given clustering (e.g. via the "**silhouette score**", see assignment)

# Lecture 17: Fairness

Machine Learning, Summer Term 2019

July 18, 2019

Michael Tangermann   Frank Hutter   Marius Lindauer

University of Freiburg



## Acknowledgement of the sources for these slides

- Draft text book: [Fairness and machine learning: Limitations and Opportunities](#) by Solon Barocas, Moritz Hardt and Arvind Narayanan
- URL of the book: <https://fairmlbook.org/>
- NIPS 2017 Tutorial by Solon Barocas and Moritz Hardt

# Lecture Overview

1 Motivation and Background

2 A Concrete Example

3 Two Definitions of Fairness

4 Wrapup

# Connection of Machine Learning and Fairness

- Machine Learning is being used for making very important decisions
  - ↝ We need to make sure that these decisions are fair

# Connection of Machine Learning and Fairness

- Machine Learning is being used for making very important decisions
  - ↝ We need to make sure that these decisions are fair
- What does it mean to be fair? This is a question for **ethics** and **law**:
  - Credit (Equal Credit Opportunity Act)
  - Education (Civil Rights Act of 1964; Education Amendments of 1972)
  - Employment (Civil Rights Act of 1964)
  - Housing (Fair Housing Act)
  - Public Accommodation (Civil Rights Act of 1964)
- These laws extend to marketing and advertising in these domains!

# Connection of Machine Learning and Fairness

- Machine Learning is being used for making very important decisions
  - ~~ We need to make sure that these decisions are fair
- What does it mean to be fair? This is a question for **ethics** and **law**:
  - Credit (Equal Credit Opportunity Act)
  - Education (Civil Rights Act of 1964; Education Amendments of 1972)
  - Employment (Civil Rights Act of 1964)
  - Housing (Fair Housing Act)
  - Public Accommodation (Civil Rights Act of 1964)
- These laws extend to marketing and advertising in these domains!
- Machine learning allows for predictions and thus differentiation
  - ~~ Take objection to **unjustified** basis for differentiation
    - Practical irrelevance (e.g., bias in the training data)
    - Moral irrelevance

## Legally recognized protected classes

- Race (Civil Rights Act of 1964)
- Color (Civil Rights Act of 1964)
- Gender (Equal Pay Act of 1963; Civil Rights Act of 1964)
- Religion (Civil Rights Act of 1964)
- National origin (Civil Rights Act of 1964)
- Age (Age Discrimination in Employment Act of 1967)
- Pregnancy (Pregnancy Discrimination Act)
- Familias status (Civil Rights Act of 1964)
- Disability status (Rehabilitation Act of 1973)
- Genetic information (Genetic Information Nondiscrimination Act)

# Legally recognized protected classes

- Race (Civil Rights Act of 1964)
- Color (Civil Rights Act of 1964)
- Gender (Equal Pay Act of 1963; Civil Rights Act of 1964)
- Religion (Civil Rights Act of 1964)
- National origin (Civil Rights Act of 1964)
- Age (Age Discrimination in Employment Act of 1967)
- Pregnancy (Pregnancy Discrimination Act)
- Familias status (Civil Rights Act of 1964)
- Disability status (Rehabilitation Act of 1973)
- Genetic information (Genetic Information Nondiscrimination Act)

Note: society is far from being fair; with ML we aim to do better!

- Opportunity to **objectively assess and remove bias**
- Opportunity to get rid of implicit (even unconscious) biases

# Issues to watch out for in machine learning

- Skewed sample
  - E.g., predictive policing
    - Future observations of crime confirm predictions
    - Risks of a feedback loop
      - (less opportunities to observe crime that contradicts predictions)

# Issues to watch out for in machine learning

- Skewed sample
  - E.g., predictive policing
    - Future observations of crime confirm predictions
    - Risks of a feedback loop
      - (less opportunities to observe crime that contradicts predictions)
- Tainted examples
  - Learn to predict hiring decisions
  - Learn to predict job success (e.g., review scores)
  - Learn to predict objective score (e.g., sales)

# Issues to watch out for in machine learning

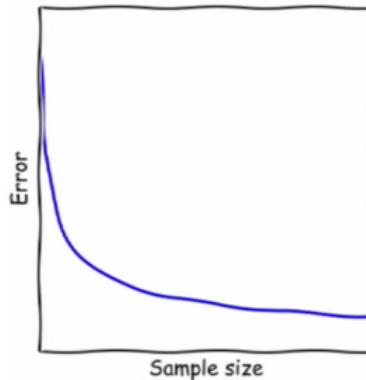
- Skewed sample
  - E.g., predictive policing
    - Future observations of crime confirm predictions
    - Risks of a feedback loop
      - (less opportunities to observe crime that contradicts predictions)
- Tainted examples
  - Learn to predict hiring decisions
  - Learn to predict job success (e.g., review scores)
  - Learn to predict objective score (e.g., sales)
- Limited features
  - Features may be informative for the majority group, but not for a minority group
  - Minority may even just be a minority in the training/validation data (example: skin cancer)
  - Different models with same validation accuracy can differ substantially concerning fairness

# Issues to watch out for in machine learning

- Proxies
  - Many features are **correlated** with sensitive features
  - With rich data, sensitive features are largely encoded across other features

# Issues to watch out for in machine learning

- Proxies
  - Many features are **correlated** with sensitive features
  - With rich data, sensitive features are largely encoded across other features
- Sample size disparity
  - By definition, there is more data for majority groups
  - Routinely, more data leads to smaller errors



# Three different problems

- Discovering unobserved differences in performance
  - Skewed sample
  - Tained examples
- Coping with observed differences in performance
  - Limited features
  - Sample size disparity
- Understanding the causes of disparities in predicted outcome
  - Proxies

# Lecture Overview

1 Motivation and Background

2 A Concrete Example

3 Two Definitions of Fairness

4 Wrapup

# Example: decisions about granting a loan or not

## Loan Strategy

Maximize profit with:

**MAX PROFIT**

No constraints

**GROUP UNAWARE**

Blue and orange thresholds  
are the same

**DEMOGRAPHIC  
PARITY**

Same fractions blue / orange loans

**EQUAL  
OPPORTUNITY**

Same fractions blue / orange loans  
to people who can pay them off

## Blue Population

0

10

20

30

40

50

60

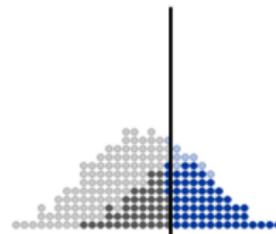
70

80

90

100

loan threshold: 61



## Orange Population

0

10

20

30

40

50

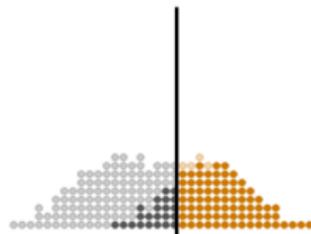
60

70

80

90

loan threshold: 50



denied loan / would default    granted loan / defaults  
denied loan / would pay back    granted loan / pays back

denied loan / would default    granted loan / defaults  
denied loan / would pay back    granted loan / pays back

**Total profit = 32400**

URL:

<https://research.google.com/bigpicture/attacking-discrimination-in-ml>

# Lecture Overview

1 Motivation and Background

2 A Concrete Example

3 Two Definitions of Fairness

4 Wrapup

## Example: job advertisement for software engineers

- $X$  features of an individual (here: browsing history)
- $A$  sensitive attribute (here: gender)
- $C = c(X, A)$  predictor (here: show ad or not)
- $Y$  target variable (here: software engineer?)
- Notation:  $P_a(E) = P(E \mid A = a)$

## Two definitions of fairness

- Independence:  $C$  independent of  $A$
- Separation:  $C$  independent of  $A$  conditional on  $Y$

Recall notation:

- $A$  sensitive attribute (example: gender)
- $C = c(X, A)$  predictor (example: show ad or not)
- $Y$  target variable (example: software engineer?)

# Independence

- Definition:  $C$  is independent of  $A$ 
  - For all groups  $a$  and  $b$  and all outcomes  $c$  of  $C$ ,

$$P_a(C = c) = P_b(C = c)$$

# Independence

- Definition:  $C$  is independent of  $A$ 
  - For all groups  $a$  and  $b$  and all outcomes  $c$  of  $C$ ,

$$P_a(C = c) = P_b(C = c)$$

- Other names: [demographic parity](#), [statistical parity](#)
- [Approximate versions](#)

$$\frac{P_a(C = 1)}{P_b(C = 1)} \geq 1 - \epsilon \quad |P_a(C = 1) - P_b(C = 1)| \leq \epsilon$$

# Independence

- Definition:  $C$  is independent of  $A$ 
  - For all groups  $a$  and  $b$  and all outcomes  $c$  of  $C$ ,

$$P_a(C = c) = P_b(C = c)$$

- Other names: [demographic parity](#), [statistical parity](#)
- [Approximate versions](#)

$$\frac{P_a(C = 1)}{P_b(C = 1)} \geq 1 - \epsilon \quad |P_a(C = 1) - P_b(C = 1)| \leq \epsilon$$

- Achieving independence
  - Post-processing
  - Training time constraint
  - Pre-processing, e.g., via representation learning

# Independence

- Definition:  $C$  is independent of  $A$ 
  - For all groups  $a$  and  $b$  and all outcomes  $c$  of  $C$ ,

$$P_a(C = c) = P_b(C = c)$$

- Other names: demographic parity, statistical parity
- Approximate versions

$$\frac{P_a(C = 1)}{P_b(C = 1)} \geq 1 - \epsilon \quad |P_a(C = 1) - P_b(C = 1)| \leq \epsilon$$

- Achieving independence
  - Post-processing
  - Training time constraint
  - Pre-processing, e.g., via representation learning
    - Find a representation  $Z = f(X, A)$ , aiming for  $\max I(X; Z)$  and  $\min I(A; Z)$
    - Then we base our classifier  $C$  only on  $Z$

# Separation

- Define a score  $r = R(A, X)$
- $R$  independent of  $A$  conditional on  $Y$ 
  - For all groups  $a$  and  $b$ , all outcomes  $r$  of  $R$ , and all outcomes  $y$  of  $Y$ :

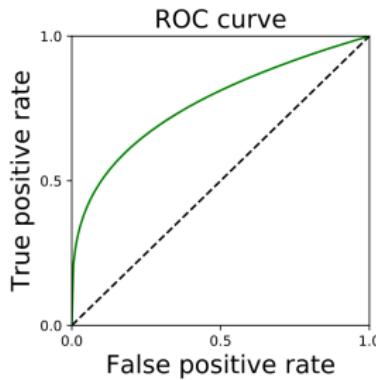
$$P_a(R = r \mid Y = y) = P_b(R = r \mid Y = y)$$

# Separation

- Define a score  $r = R(A, X)$
- $R$  independent of  $A$  conditional on  $Y$ 
  - For all groups  $a$  and  $b$ , all outcomes  $r$  of  $R$ , and all outcomes  $y$  of  $Y$ :

$$P_a(R = r \mid Y = y) = P_b(R = r \mid Y = y)$$

- Again, you can define approximate versions like for independence
- Achieving independence: training constraint or post-processing by looking at (TPR, FPR) for all possible thresholds on score  $r$

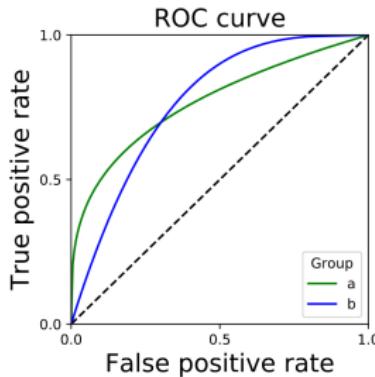


# Separation

- Define a score  $r = R(A, X)$
- $R$  independent of  $A$  conditional on  $Y$ 
  - For all groups  $a$  and  $b$ , all outcomes  $r$  of  $R$ , and all outcomes  $y$  of  $Y$ :

$$P_a(R = r \mid Y = y) = P_b(R = r \mid Y = y)$$

- Again, you can define approximate versions like for independence
- Achieving independence: training constraint or post-processing by looking at (TPR, FPR) for all possible thresholds on score  $r$

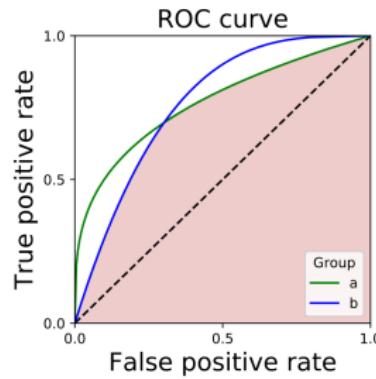


# Separation

- Define a score  $r = R(A, X)$
- $R$  independent of  $A$  conditional on  $Y$ 
  - For all groups  $a$  and  $b$ , all outcomes  $r$  of  $R$ , and all outcomes  $y$  of  $Y$ :

$$P_a(R = r \mid Y = y) = P_b(R = r \mid Y = y)$$

- Again, you can define approximate versions like for independence
- Achieving independence: training constraint or post-processing by looking at (TPR, FPR) for all possible thresholds on score  $r$



# Tradeoffs are Necessary

Theorem: impossibility of both independence and separation

If neither  $A$  nor  $R$  are independent of  $Y$ , then you cannot get both separation and independence at once.

- Someone (probably not a computer scientist) has to decide what fairness criteria we want to fulfill in a particular application
- Impossibility only holds for exact independence/separation
  - We can still aim for good **tradeoffs** of their approximate versions

# Lecture Overview

1 Motivation and Background

2 A Concrete Example

3 Two Definitions of Fairness

4 Wrapup

## Summary by learning goals

Having heard this lecture, you can now ...

- List some features that are **illegal** to use in some cases
- Motivate the need for research on fairness in machine learning
- Discuss two notions of **algorithmic fairness** and their relation