Machine Learning

Lecture 1
Introduction and Overview

Felix Bießmann

Beuth University & Einstein Center for Digital Future

April 3, 2019

Why not just calling it Artificial Intelligence?

- Intelligence is very difficult to define [Tomasello, 2003]
- Artificial and human intelligence is difficult to compare [Turing, 1950]
- Scientists use a simpler term: Machine Learning (ML):

  $\Rightarrow$ **ML are Algorithms that learn from data.**

Machine Learning – The Big Picture

Given data $x \in \mathcal{X}$ (neural signals, internet user data, ...)

$\rightarrow$ predict variable $y \in \mathcal{Y}$ (thoughts, user intentions, ...)

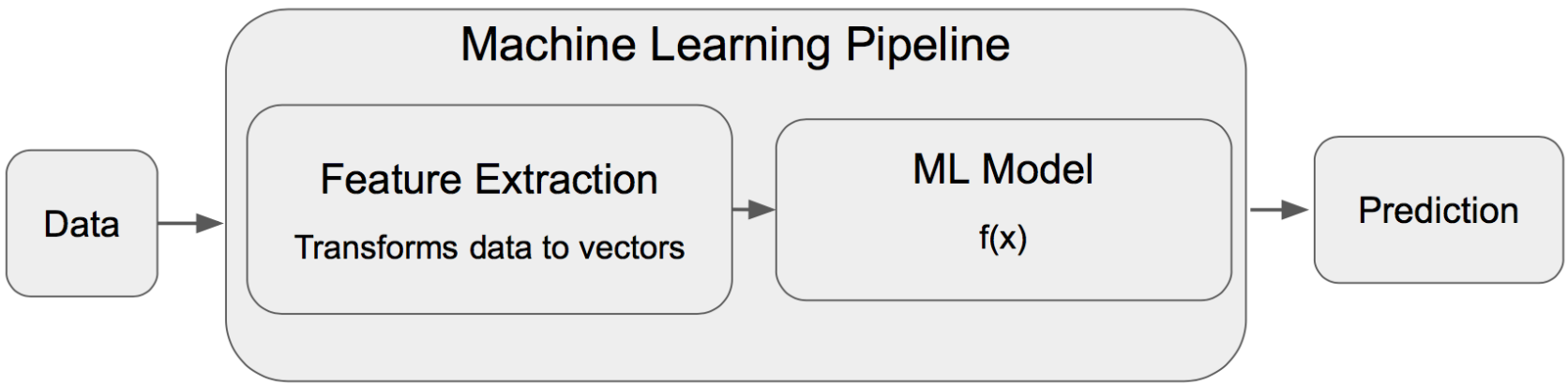
ML algorithms learn a function $f(.)$ **from examples** (x,y)

$$f(x) = y \tag{1}$$

Most of ML is about:

- Find the right function class for $f(.)$
- Fitting $f(.)$ correctly

Machine Learning Pipelines



Machine Learning In Practice

ML algorithms

- are defined using vector / matrix operations
- are usually some mathematical function

$\rightarrow$ That's why we will need some math basics.

AI, ML and Cognition

- Machine Learning can be defined from the algorithm point of view:

  $\Rightarrow$ **ML are Algorithms that learn from data.**

- Another useful view

  $\Rightarrow$ **Algorithms that emulate biological cognition**

What is Cognition?

- Derived from cognoscere (latin): *to recognize*
- Refers to processing of information

  Learning, recognizing and categorizing concepts
  Inference based on these concepts (e.g. language)
- Cognitive processes can be

  natural or artificial, conscious or unconscious
- **Easier to define and measure than intelligence**
- Cognition is studied by

  linguistics, neuroscience, philosophy, psychology
  AND computer science (artificial intelligence, machine learning)

A short history of Machine Learning

Where does Machine Learning come from?

The aim of building artificial cognitive systems goes back to

- The concept of **automata**
- The insight that our cognitive processes emerge from biological tissue (brains)
- The combination of these two ideas led to the fields of **Cybernetics, Artificial Intelligence**, and **Machine Learning**

A short history of Machine Learning

1632 Bodies as biological machines [Descartes, 1632]

1940 From machines to intelligent machines
Cybernetics – mechanics of cognition [Wiener, 1948]
Turing Test [Turing, 1950]: *Can machines think?*
Perceptron [Rosenblatt, 1958]: *Artificial 'brains'*
Eliza [Weizenbaum, 1966]: *Better than psychotherapy!*
Chinese Room argument [Searle, 1980]: Against functionalism

1973 AI winter [Lighthill, 1973]

$\rightarrow$ Classical AI was mainly built on hand made rules

Problem: No scalability, no unified theory of cognition

$\Rightarrow$ **New AI**:
Let machines learn rules from data [Russell and Norvig, 2010]

Cybernetics: First Approaches to Cognitive Algorithms

- **Macy Conferences** (1946-53, New York):
  Interdisciplinary effort to set the foundations for a general science of the workings of the human mind.

$\rightarrow$ Cybernetics – the art of controlling [Wiener, 1948]

- Some famous cyberneticists:
  Norbert Wiener (Wiener Filters), John von Neumann (Computers), Claude E. Shannon (Information Theory), Werner Reichhardt (Reichhardt Correlator), Valentino Braitenberg (Braitenberg Vehicles), ...

ELIZA [Weizenbaum, 1966]

Human All men are alike

Eliza *In what way?*

Human They're always bugging us about something or other

Eliza *Can you think of a specific example*

Human Well, my boyfriend made me come here

Eliza *Your boyfriend made you come here?*

Human He says I'm depressed much of the time

Eliza *I am sorry to hear you're depressed*

Human It's true, I am unhappy

Eliza *Do you think coming here will help you not to be unhappy?*

Human I need some help, that much seems certain.

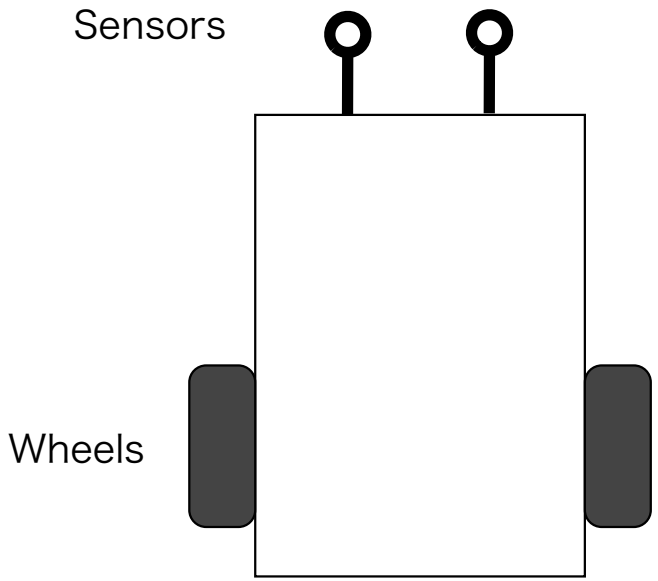
Eliza *What would it mean to you if you got some help?*

Human Perhaps I could learn to get along with my mother.

Eliza *Tell me more about your family*

...

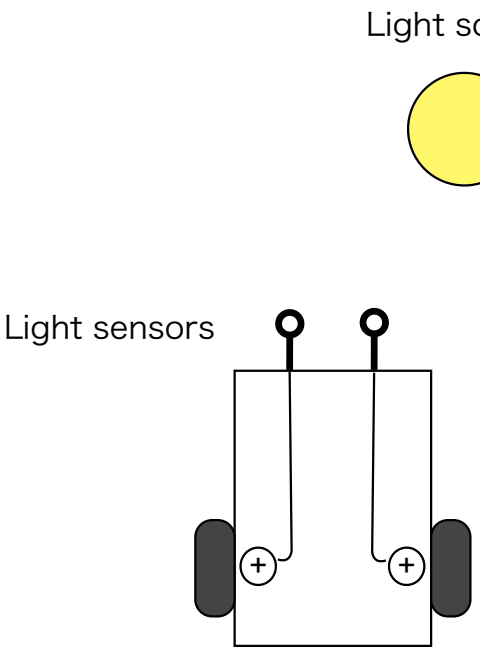
Braitenberg Vehicles
Artificial organisms with 'emotions'



Artificial organisms with very simple hardware can exhibit complex behavior.

Behaviour of Braitenberg Vehicles can be interpreted as different 'characters' [Braitenberg, 1984].

Braitenberg Vehicles



In which direction will
the Braitenberg Vehicle go?

Why is Cybernetics important?

- Many algorithms from Cyberneticists are still state of the art
- Understanding biological cognition improves artificial cognition
- Cybernetic principles are independent of sensory modality!
- This also a goal of Machine Learning:
  Modeling of cognitive mechanisms (independent of data type).

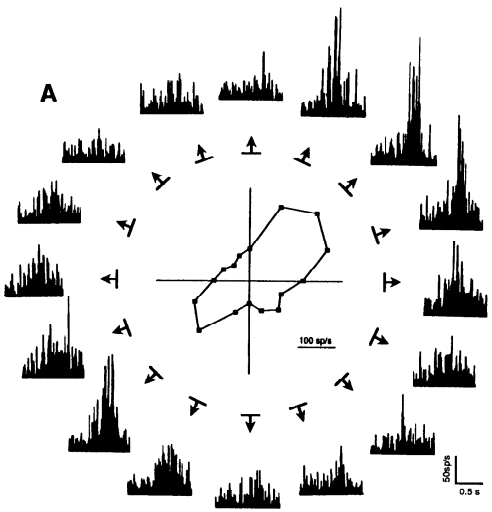
Also in biological systems cognitive mechanism can be decoupled from sensory modalities

$\rightarrow$ Audiovisual re-routing in animals [Roe et al., 1992]

$\rightarrow$ Sensory substitution [Bach-y Rita et al., 1969]

Audiovisual Re-routing

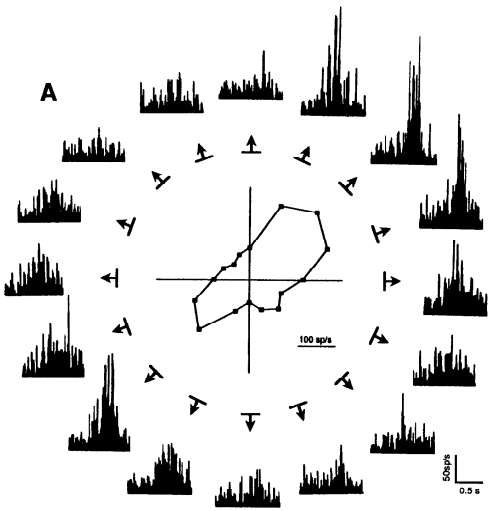
Neural response in Visual Brain Area



In an experiment on ferrets [Roe et al., 1992] the nerve fibers from the eye were re-routed to the brain areas that process information from the ear (auditory brain areas).

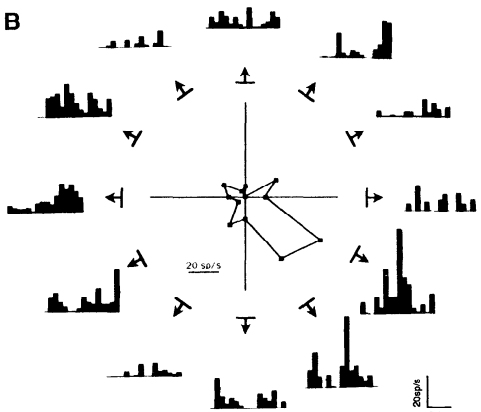
Audiovisual Re-routing

Neural response in Visual Brain Area



In an experiment on ferrets [Roe et al., 1992] the nerve fibers from the eye were re-routed to the brain areas that process information from the ear (auditory brain areas).

Neural response in Auditory Brain Area



$\Rightarrow$ Neural responses in auditory brain area became selective like neurons in visual areas

Sensory Substitution

Sensory Substition

Program new cognitive functions into existing sensory modality

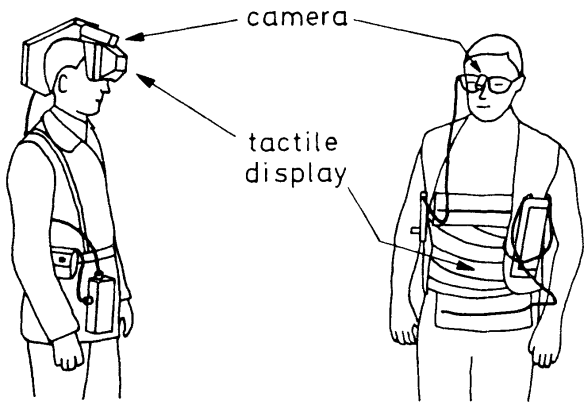
Examples:

Haptic Vision [Bach-y Rita et al., 1969]
→ Luminance on photosensors translated to vibrations on skin

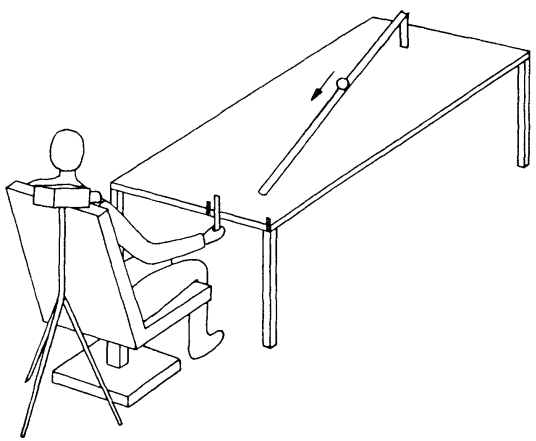
Feelspace [Nagel et al., 2005]
→ Allocentric orientation through haptic feedback

Sensory Substitution: Tactile Vision

Tactile vision device



Experimental setup



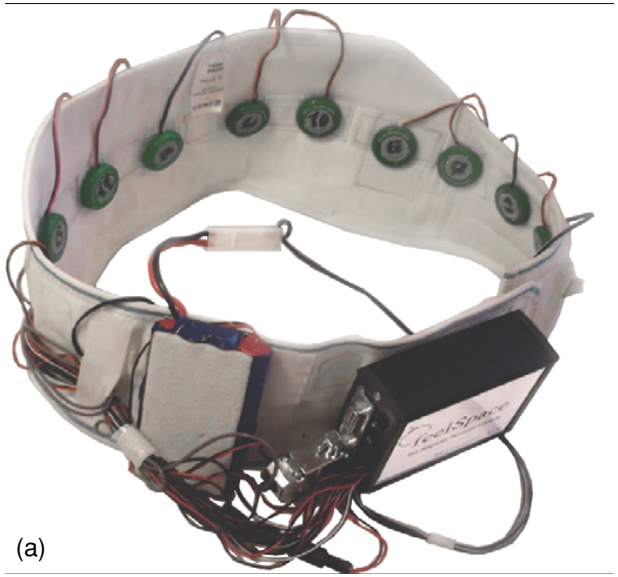
Pioneered by [Bach-y Rita et al., 1969], tested in [Jansson, 1983]

- Tactile matrix (20×20 vibrators) on the subject's back
- Tactile information about catching device and ball

$\Rightarrow$ **Blind subjects can catch the ball!**

Sensory Substitution: Feelspace

Feelspace belt



(a)

`http://feelspace.cogsci.uni-osnabrueck.de/`

Creating an allocentric sense of orientation [Nagel et al., 2005]

- Direction of north pole is translated to vibration in belt
- After wearing the belt for some weeks subjects acquire an allocentric sense of orientation

"I was intuitively aware of the direction of my home or of my office."

Summary

- Cognitive processes:
  Perception, recognition of / inference on semantic concepts
- Cybernetics
  - simple (hardwired) models of cognition
  - inspired by biological organisms
  - different from classical engineering approaches
  - modeled motion detection, navigation, ...

  $\rightarrow$ but what about higher cognitive functions?
- Artificial intelligence
  - took over ideas from Cybernetics
  - focused on (biologically inspired) models of higher cognition
  - Old AI: rule based systems (like ELIZA)
  - AI Winter: Most research stopped, AI hype broke down
  - New AI (machine learning): learns rules from data

References

P. Bach-y Rita, C. C. Collins, F. A. Saunders, B. White, and L. Scadden. Vision substitution by tactile image projection. *Nature*, 221(5184):963–4, 1969.

V. Braitenberg. *Vehicles - Experiments in Synthetic Psychology*. MIT Press, 1984.

R. Descartes. *Traité de l'homme*. 1632.

G. Jansson. Tactile guidance of movement. *Int J Neurosci*, 19(1-4):37–46, 1983.

J. Lighthill. Artificial intelligence: A general survey. In *Artificial Intelligence: a paper symposium, Science Research Council*, 1973.

S. K. Nagel, C. Carl, T. Kringe, R. Märtin, and P. König. Beyond sensory substitution–learning the sixth sense. *J Neural Eng*, 2(4):R13–26, 2005.

A. W. Roe, S. L. Pallas, Y. H. Kwon, and M. Sur. Visual projections routed to the auditory pathway in ferrets: receptive fields of visual neurons in primary auditory cortex. *J Neurosci*, 12(9):3651–64, 1992.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, Nov. 1958.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (3nd Edition)*. Prentice Hall series in artificial intelligence. Prentice Hall, 2 edition, Dec. 2010. ISBN 0137903952.

J. Searle. Minds, brains and programs. *Behavioral and Brain Sciences*, 3(3):417–457, 1980.

M. Tomasello. *The cultural origins of human cognition*. Harvard University Press, 2003.

A. Turing. The imitation game. *MIND*, pages 1–28, 1950.

J. Weizenbaum. Eliza - a computer program for the study of natural language communication between man and machine. *Commununications of the ACM*, 9(1):36–45, 1966.

N. Wiener. *Cybernetics, or control and communication in the animal and the machine*. Wiley, 1948.

Machine Learning

Lecture 2
Math Recap - Vectors, Matrices, Functions

Felix Bießmann

Beuth University & Einstein Center for Digital Future

April 10, 2019

Machine Learning In Practice

After importing and cleaning a data set we have:

- D-dimensional data points $\mathbf{x}_i \in \mathbb{R}^D,\ i \in \{1, 2, \ldots, N\}$
  Often the data is stored in matrix form

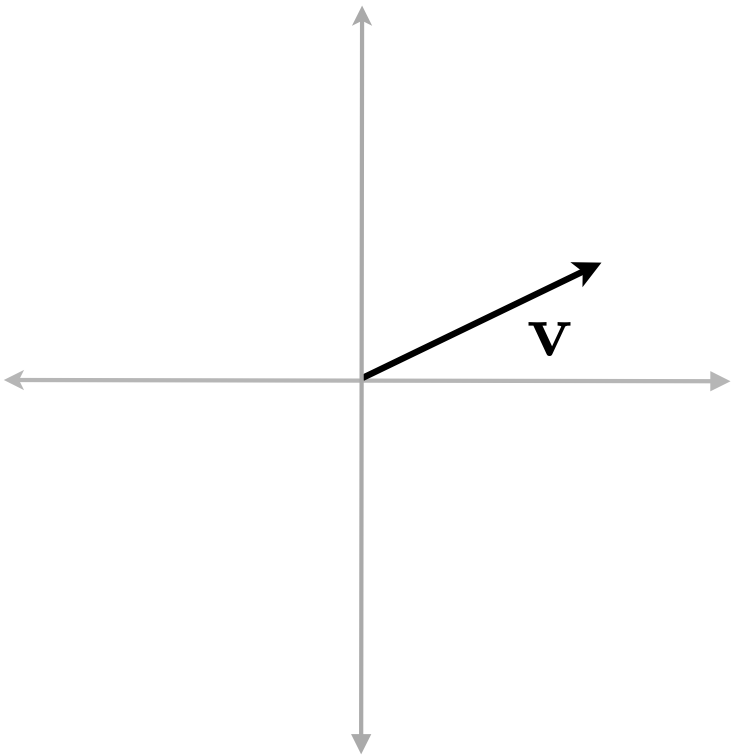
$$X = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{D \times N} \tag{2}$$

- Corresponding labels $\mathbf{y}_i \in \mathbb{R}^L$

The data points and labels are **vectors**.

$\rightarrow$ Today we recap some basics about vector spaces and matrices

What is a Vector?



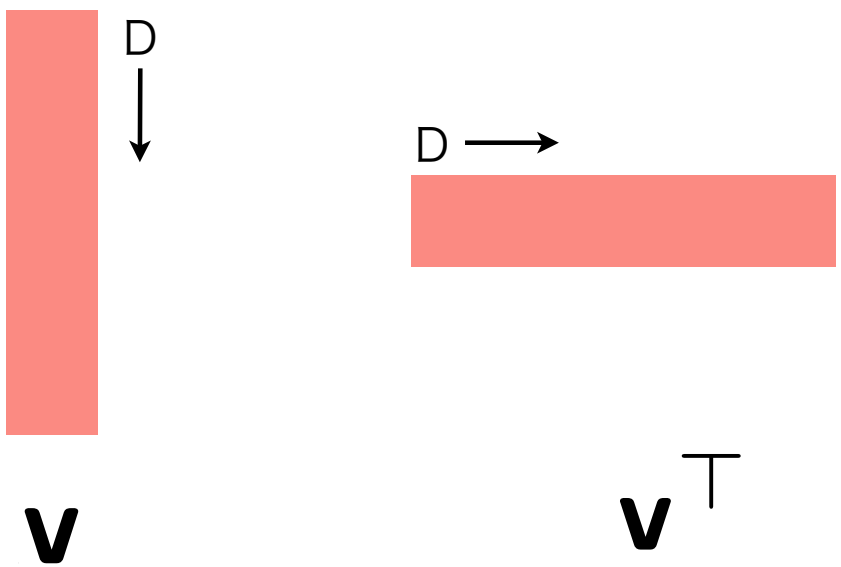
A vector $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_D \end{bmatrix}$

can be understood as a point in a D-dimensional space

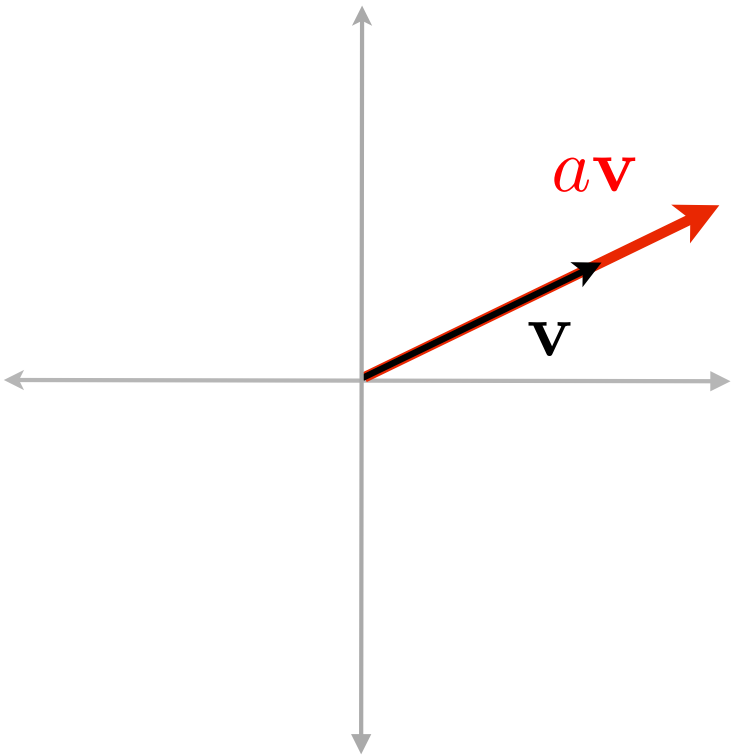
For comprehensibility we will restrict examples to $\mathbf{v} \in \mathbb{R}^2$.

What is a transposed Vector?

$\mathbf{v}^\top$ denotes $\mathbf{v}$ – **transposed**



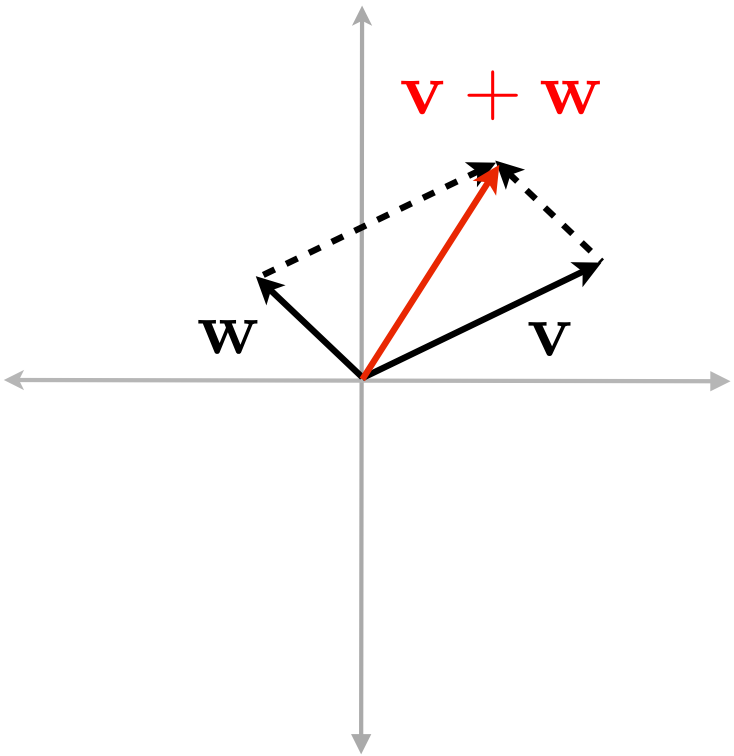
Vector Operations



Vectors can be scaled

$$a\mathbf{v} = \begin{bmatrix} av_1 \\ av_2 \\ \vdots \\ av_D \end{bmatrix} \tag{3}$$

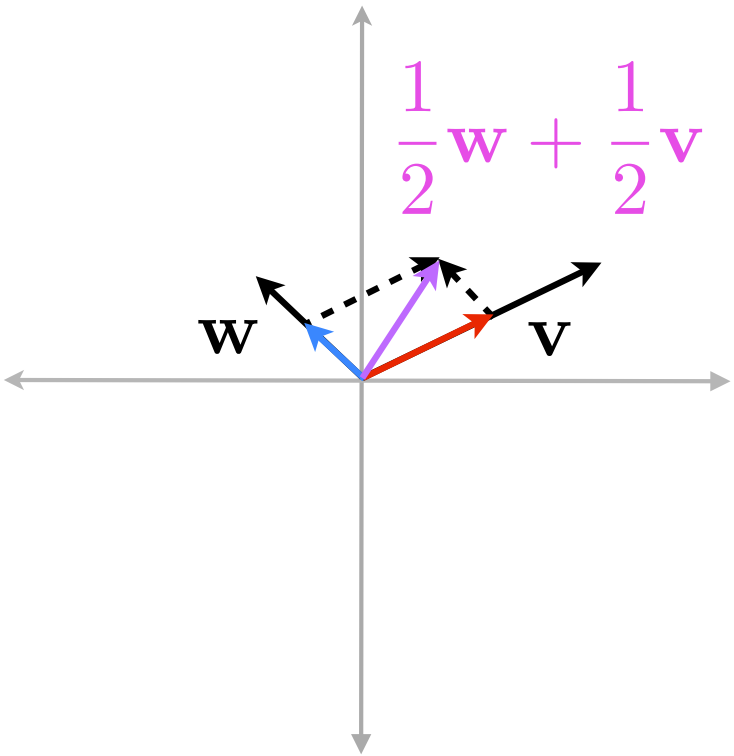
Vector Operations



Vectors from the same space can be added to each other

$$\mathbf{v} + \mathbf{w} = \begin{bmatrix} v_1 + w_1 \\ v_2 + w_2 \\ \vdots \\ v_D + w_D \end{bmatrix} \quad (4)$$

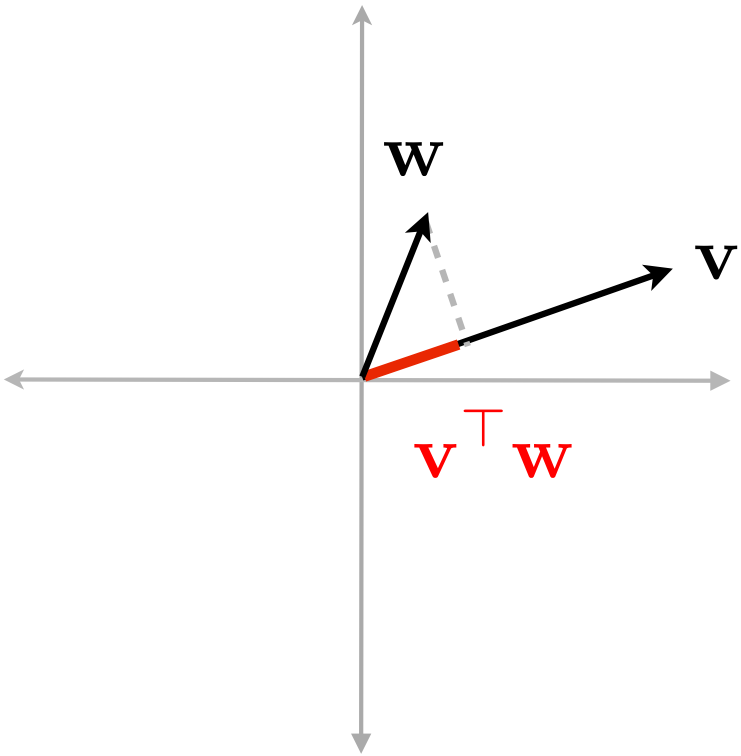
Vector Operations



Scaling and addition can be combined

$$a\mathbf{v} + b\mathbf{w} = \begin{bmatrix} av_1 + bw_1 \\ av_2 + bw_2 \\ \vdots \\ av_D + bw_D \end{bmatrix} \quad (5)$$

Vector Operations



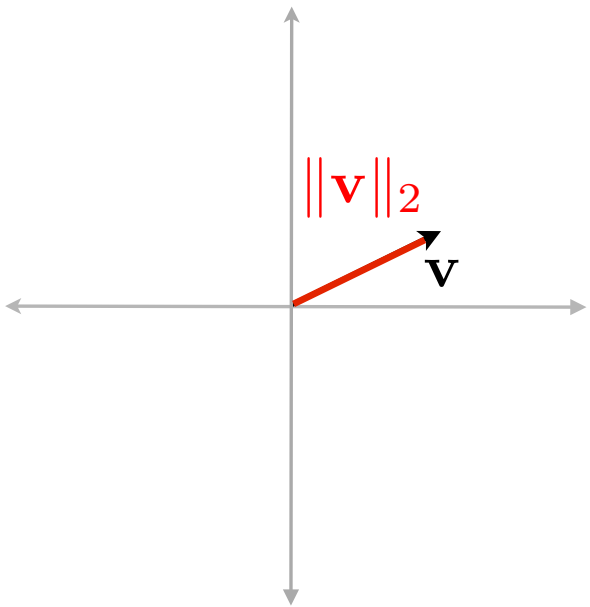
The **scalar product** computes the *similarity* between two vectors **v** and **w**

$$\langle \mathbf{v}, \mathbf{w} \rangle = \mathbf{v}^\top \mathbf{w} = \begin{bmatrix} v_1 & v_2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$$

$$= \sum_i^D v_i w_i \qquad (6)$$

$\mathbf{v}^\top \mathbf{w}$ is often called *projecting* **w** onto **v**

Length of a Vector



Sometimes we are not interested in the direction of a vector, but only in its length.

How can we measure the **length of a vector**?

Norm of a Vector

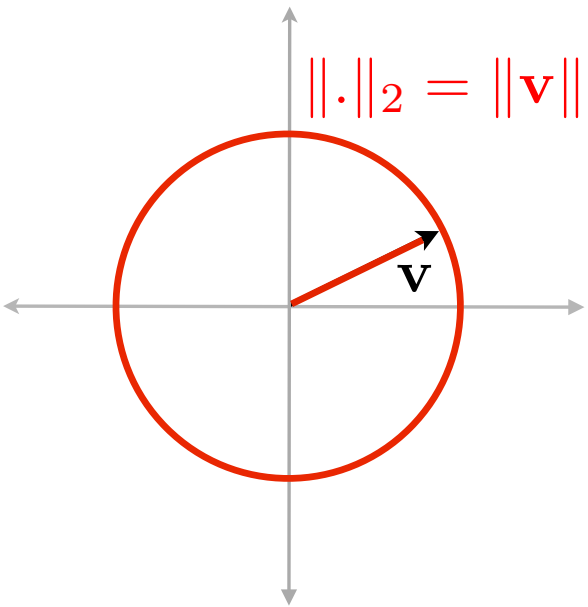
The p-**norm** $\|\mathbf{v}\|_p$ of a D-dimensional vector $\mathbf{v}$ is defined as

$$\|\mathbf{v}\|_p = \left(\sum_i^D |v_i|^p \right)^{1/p} \tag{7}$$

Usually (and in the following) $\|.\|$ denotes the **euklidean norm** ($p = 2$)

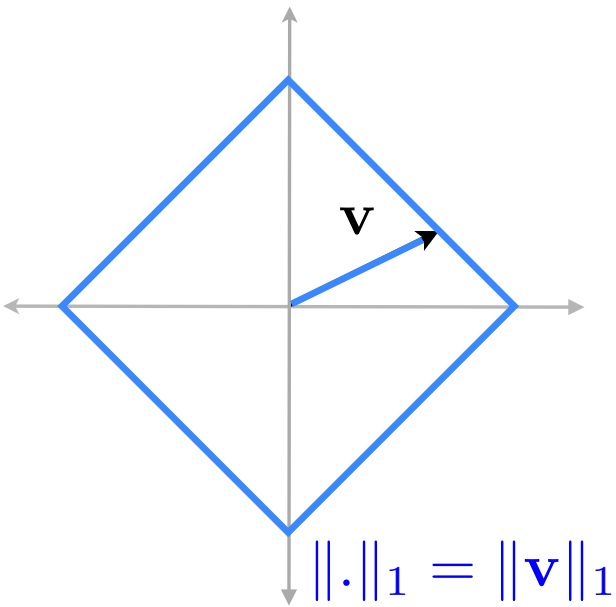
$$\|\mathbf{v}\|_2 = \left(\sum_i^D |v_i|^2 \right)^{1/2} = \sqrt{\sum_i^D v_i^2} = \sqrt{\mathbf{v}^\top \mathbf{v}} \tag{8}$$

The Euklidean (2-Norm)



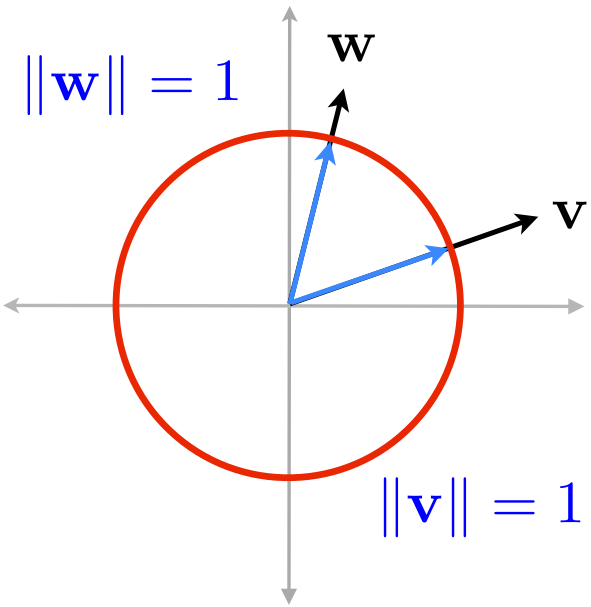
$$\|\mathbf{v}\|_2 = \left(\sum_i^D |v_i|^2\right)^{1/2} = \sqrt{\mathbf{v}^\top \mathbf{v}} \qquad (9)$$

The 1-Norm



$$\|\mathbf{v}\|_1 = \left(\sum_i^D |v_i|\right) = \sum_i^D |v_i| \qquad (10)$$

Unit vectors



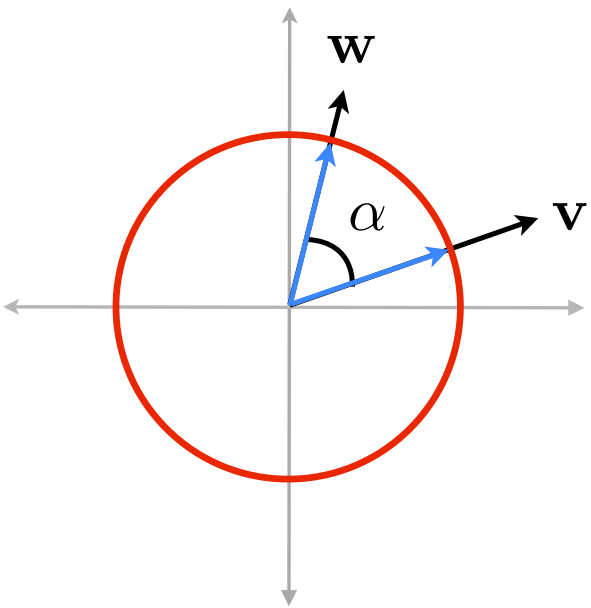
Every vector $\mathbf{v}$ can be scaled to length 1 by

$$\mathbf{v} \leftarrow \mathbf{v}/\|\mathbf{v}\| \tag{11}$$

This is useful. Why?

Because after normalizing each vector to unit length, our vector similarity measure $\mathbf{v}^\top \mathbf{w}$ has a standardized meaning.

Angle between two vectors



The scalar product of unit vectors is the cosine of the angle between the vectors

$$\frac{\mathbf{v}^\top \mathbf{w}}{\|\mathbf{v}\|\|\mathbf{w}\|} = cos(\alpha) \tag{12}$$

So the angle α between $\mathbf{v}$ and $\mathbf{w}$ is

$$\alpha = acos\left(\frac{\mathbf{v}^\top \mathbf{w}}{\|\mathbf{v}\|\|\mathbf{w}\|}\right). \tag{13}$$

Note that $-1 \leq cos(\alpha) \leq 1$.

Does that remind you of something?

Angles between vectors and correlation coefficients

Consider T zero-mean[1] samples from two univariate time series $x_i \in \mathbb{R}^1, y_i \in \mathbb{R}^1, i \in \{1, \ldots, T\}$ stored in vectors $\mathbf{x}$, $\mathbf{y}$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_T \end{bmatrix} \mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_T \end{bmatrix} \tag{14}$$

The **correlation coefficient** between $\mathbf{x}$ and $\mathbf{y}$ is defined as

$$\frac{\sum_i^T x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}} = \frac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} \tag{15}$$

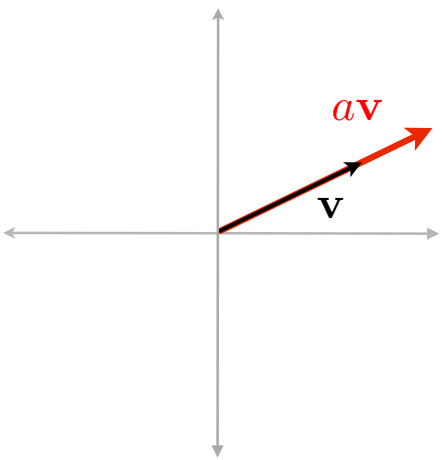
$\Rightarrow$ The cosine of the angle between $\mathbf{x}$ and $\mathbf{y}$ is their correlation!

[1] $\sum_i^T x_i = \sum_i^T y_i = 0$

Summary Vector Operations

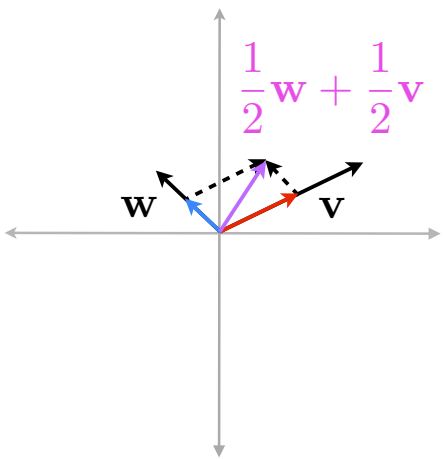
Vector Scaling

$$a\mathbf{v} = \begin{bmatrix} av_1 \\ av_2 \\ \vdots \\ av_D \end{bmatrix}$$



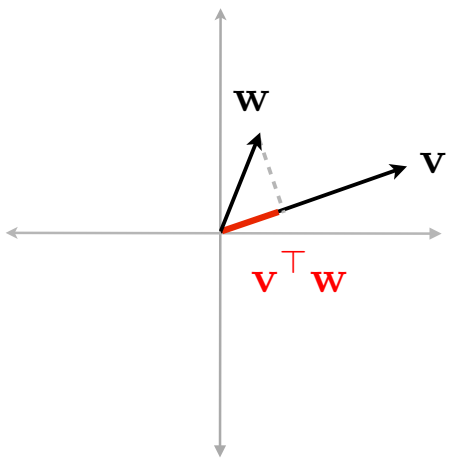
Vector Addition

$$a\mathbf{v} + b\mathbf{w} = \begin{bmatrix} av_1 + bw_1 \\ av_2 + bw_2 \\ \vdots \\ av_D + bw_D \end{bmatrix}$$



Scalar Product

$$\mathbf{v}^\top \mathbf{w} = \sum_i^D v_i w_i$$



Axioms of Vector Operations

1. Associativity of addition

$$\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w} \tag{16}$$

2. Commutativity of addition

$$\mathbf{v} + \mathbf{w} = \mathbf{w} + \mathbf{v} \tag{17}$$

3. Distributivity of scaling

$$a(\mathbf{v} + \mathbf{w}) = a\mathbf{w} + a\mathbf{v} \tag{18}$$

Matrices

This is an example of a matrix $A \in \mathbb{R}^{U \times V}$

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1V} \\ A_{21} & A_{22} & \dots & A_{2V} \\ \vdots & \vdots & \ddots & \vdots \\ A_{U1} & A_{U2} & \dots & A_{UV} \end{bmatrix}$$

Matrix Multiplication

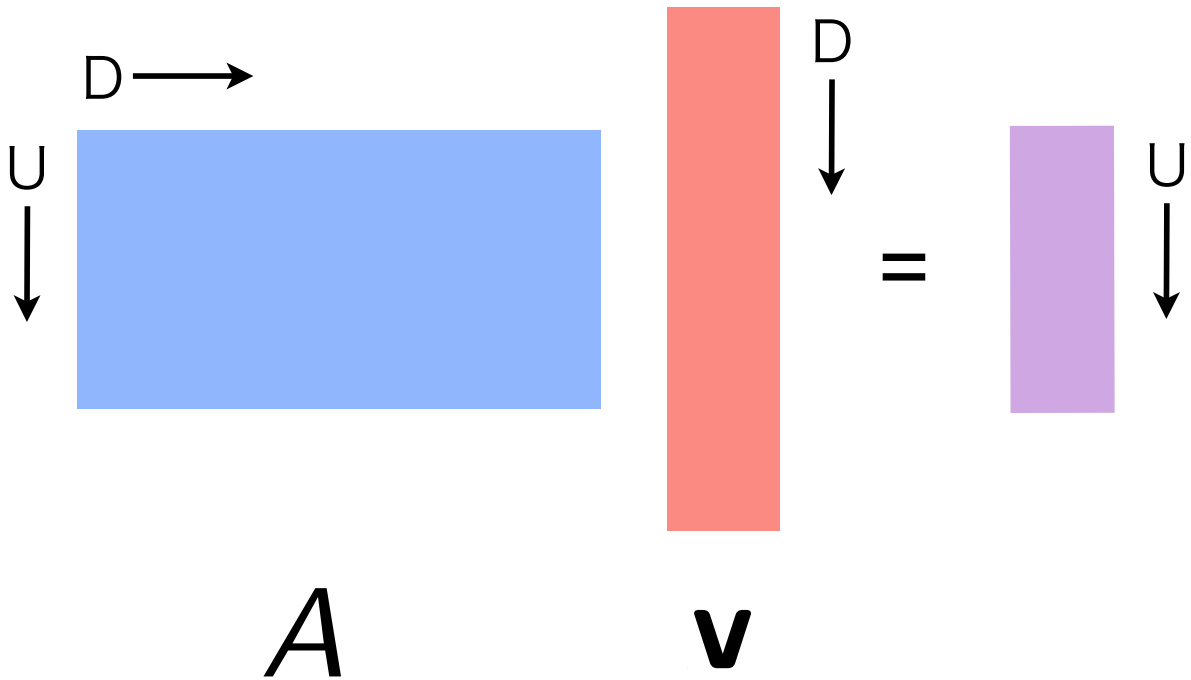
Matrix $A \in \mathbb{R}^{U \times D}$ can be multiplied with a vector $\mathbf{v} \in \mathbb{R}^{D}$ by

$$A\mathbf{v} = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1D} \\ A_{21} & A_{22} & \dots & A_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ A_{U1} & A_{U2} & \dots & A_{UD} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_D \end{bmatrix} \tag{19}$$

$$= \begin{bmatrix} A_{11}v_1 & + & A_{12}v_2 & \dots & + & A_{1D}v_D \\ A_{21}v_1 & + & A_{22}v_2 & \dots & + & A_{2D}v_D \\ \vdots & & \vdots & & \ddots & \vdots \\ A_{U1}v_1 & + & A_{U2}v_2 & \dots & + & A_{UD}v_D \end{bmatrix} \tag{20}$$

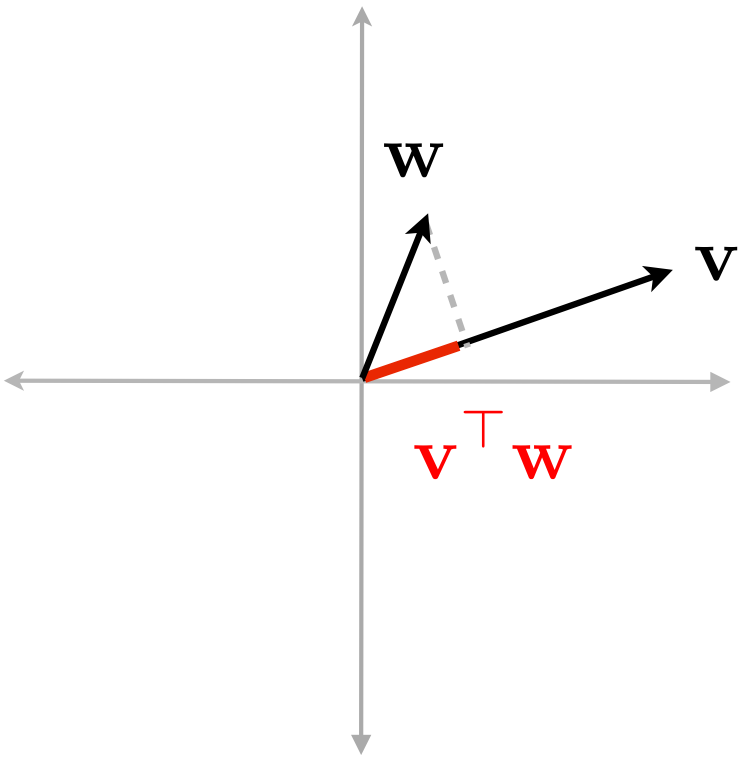
Matrix Multiplication

Matrix multiplication as pictorial illustration



Matrices as Linear Mappings

A matrix $A \in \mathbb{R}^{U \times V}$ can be understood as a **linear map** from a U-dimensional space to a V-dimensional space



Consider the scalar product $\mathbf{v}^\top \mathbf{w}$
$\mathbf{w}$ is mapped from $\mathbb{R}^2 \to \mathbb{R}^1$

But the mapping $\mathbf{v}^\top \in \mathbb{R}^{1 \times 2}$ is a very simple case

Matrices as Linear Mappings

- Permutation of dimensions:

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- Mirroring:

$$A = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

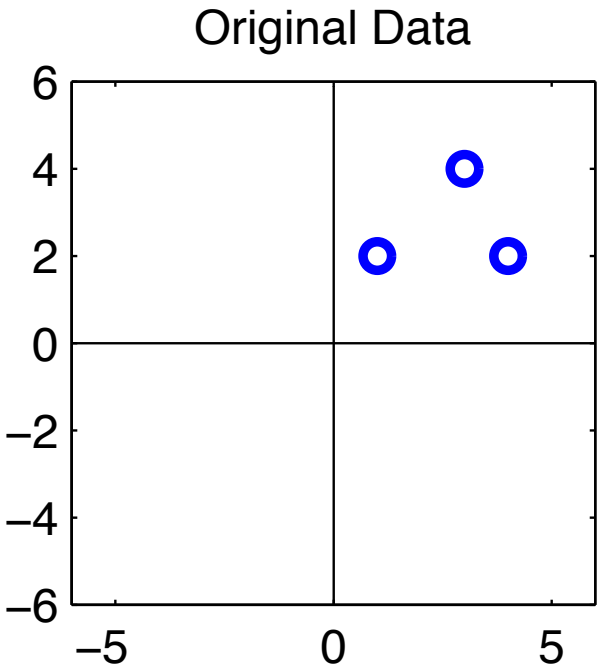
- Scaling:

$$A = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$$

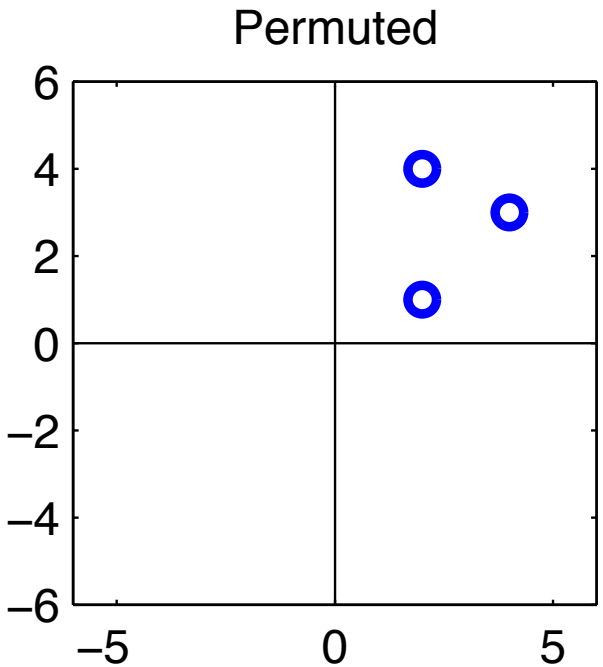
- Rotation by an angle of α in radians:

$$A = \begin{bmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{bmatrix}$$

Matrices as Linear Mappings: Permutation

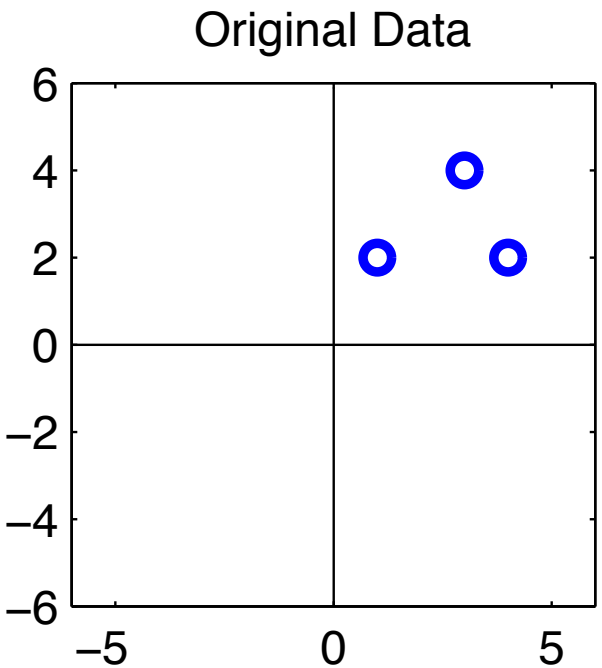


$AX \quad \rightarrow$

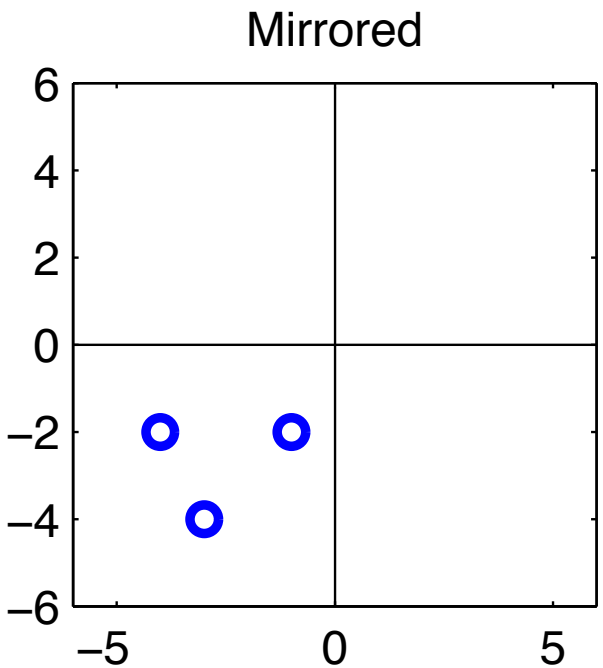


$$X = \begin{bmatrix} 1 & 3 & 4 \\ 2 & 4 & 2 \end{bmatrix} \qquad A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad AX = \begin{bmatrix} 2 & 4 & 2 \\ 1 & 3 & 4 \end{bmatrix}$$

Matrices as Linear Mappings: Mirroring

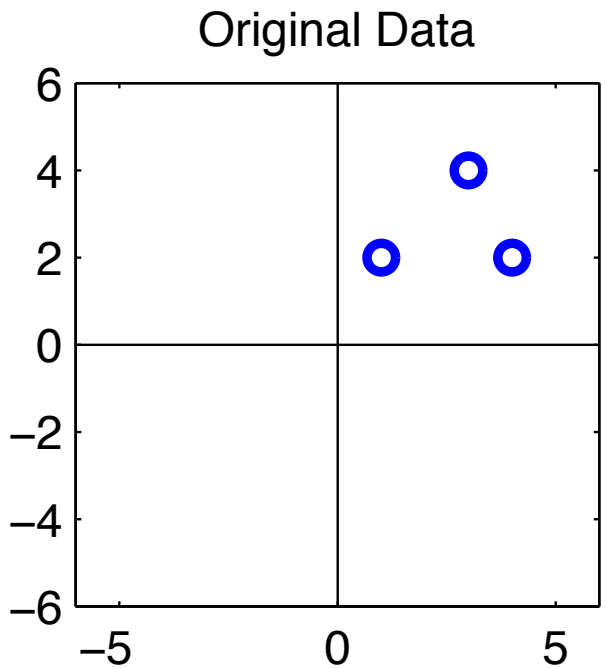


$$AX \quad \rightarrow$$

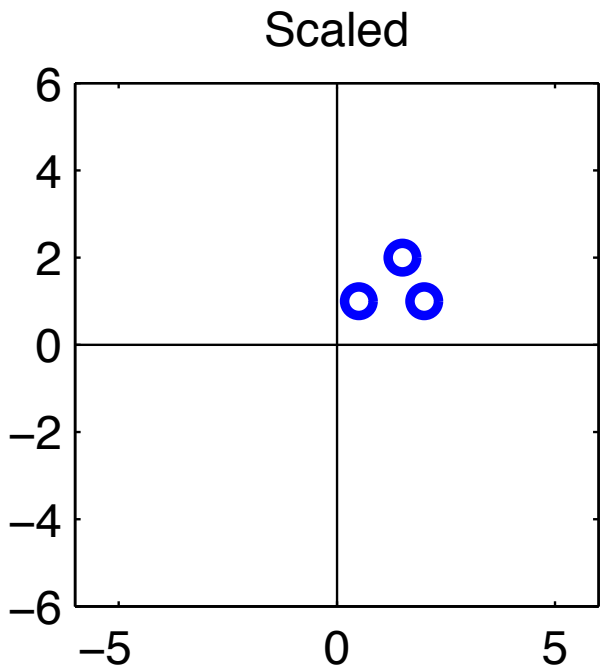


$$A = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$

Matrices as Linear Mappings: Scaling

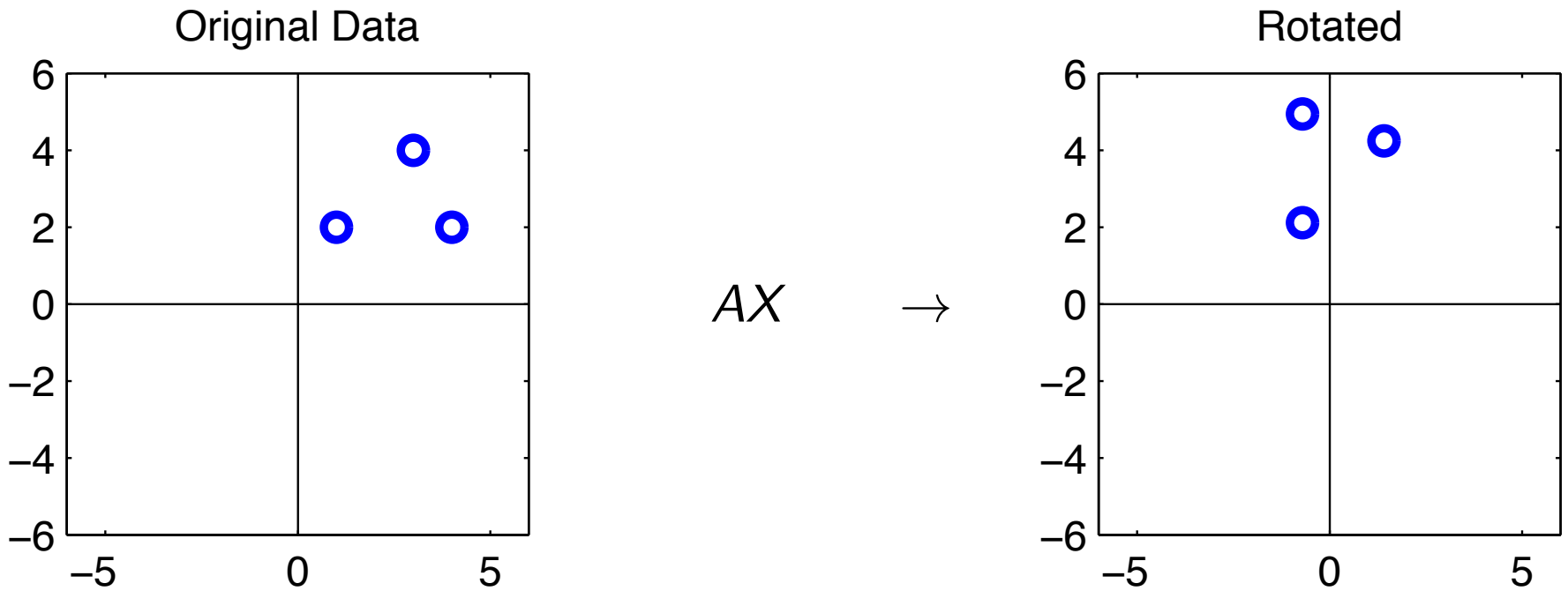


$AX \quad \rightarrow$



$$A = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/2 \end{bmatrix}$$

Matrices as Linear Mappings: Rotation



$$A = \begin{bmatrix} cos(\pi \cdot 45/180) & -sin(\pi \cdot 45/180) \\ sin(\pi \cdot 45/180) & cos(\pi \cdot 45/180) \end{bmatrix}$$

Axioms of Matrix Operations

1. Associativity

$$A(BC) = (AB)C \tag{21}$$

2. Distributivity

$$A(B + C) = AB + AC, \qquad (A + B)C = AC + BC \tag{22}$$

3. **No Commutativity!**

$$AB \neq BA \tag{23}$$

Matrix Multiplication – Some Examples

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ 15 & -17 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ -2 & 7 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 \\ -1 & 5 \end{bmatrix} \begin{bmatrix} 3 & 2 & 8 \\ 7 & 5 & 1 \end{bmatrix}$$

Matrix Multiplication – Some Examples

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ 15 & -17 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 15 & -17 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ -2 & 7 \end{bmatrix} = \begin{bmatrix} -2 & 7 \\ 3 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 \\ -1 & 5 \end{bmatrix} \begin{bmatrix} 3 & 2 & 8 \\ 7 & 5 & 1 \end{bmatrix} = \begin{bmatrix} 27 & 19 & 19 \\ 32 & 23 & -3 \end{bmatrix}$$

Minimizing Functions

Remember:

- ML algorithms are functions f that need to be fitted to data
- Training ML algorithms is finding minima/maxima of functions
- Function minimization/maximization is often done by **Gradient Descent**

Taking Derivatives of Univariate Functions

Function	Derivative
$f(x) = x^n$	$f'(x) = nx^{n-1}$
$cf(x)$	$cf'(x)$
$f(x) + g(x)$	$f'(x) + g'(x)$
$f(x)g(x)$	$f'(x)g(x) + f(x)g'(x)$
$\frac{f(x)}{g(x)}$	$\frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$
$f(g(x))$	$f'(g(x))\ g'(x)$

Taking Derivatives of Vector-Valued Functions

Consider a vector-valued function $f(\mathbf{x}) : \mathbb{R}^D \rightarrow \mathbb{R}^U, \mathbf{x} \in \mathbb{R}^D$

The gradient $\nabla f(\mathbf{x})$ at position $\mathbf{x}$ is

$$\nabla f(x) = \left[\frac{\partial f(x)}{\partial x_1}, \ldots, \frac{\partial f(x)}{\partial x_D}\right] \tag{24}$$

Taking Derivatives of Vector-Valued Functions

Function	Derivative
$f(\mathbf{x}) = \mathbf{u}^\top \mathbf{x}$	
$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{x}$	
$f(\mathbf{x}) = A\mathbf{x}$	
$f(\mathbf{x}) = \mathbf{x}^\top A\mathbf{x}$	

More complicated examples: The Matrix Cookbook

Taking Derivatives of Vector-Valued Functions

Function	Derivative
$f(\mathbf{x}) = \mathbf{u}^\top \mathbf{x}$	$f'(\mathbf{x}) = \mathbf{u}$
$f(\mathbf{x}) = \mathbf{x}^\top \mathbf{x}$	$f'(\mathbf{x}) = 2\mathbf{x}$
$f(\mathbf{x}) = A\mathbf{x}$	$f'(\mathbf{x}) = A$
$f(\mathbf{x}) = \mathbf{x}^\top A\mathbf{x}$	$f'(\mathbf{x}) = \mathbf{x}^\top (A + A^\top)$

More complicated examples: The Matrix Cookbook

Machine Learning

Lecture 3
Simple Classifiers: Nearest Centroids and Linear Classification

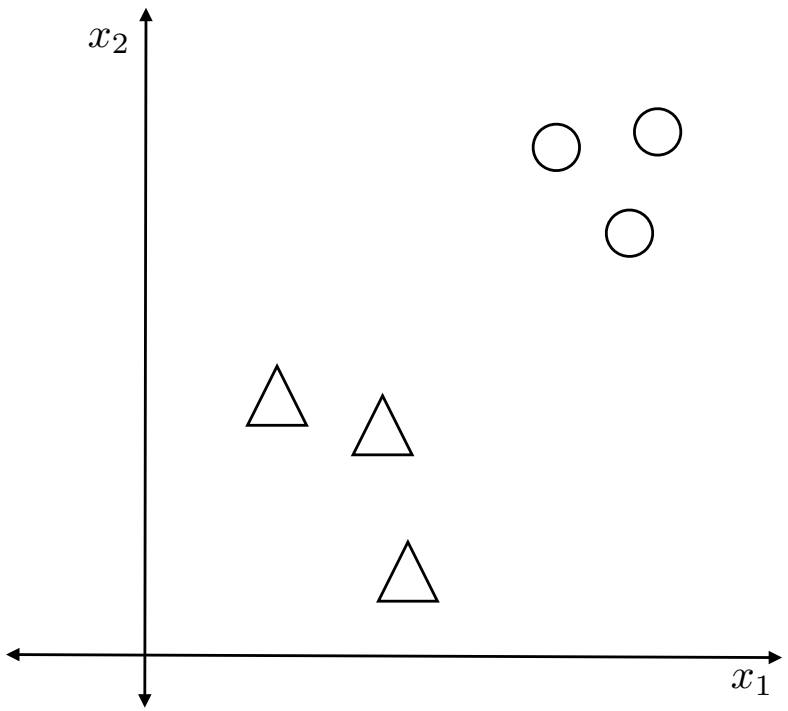
Felix Bießmann

Beuth University & Einstein Center for Digital Future

Overview of today's lecture

- Today we will introduce three simple classifiers
  1. Nearest Centroid Classifier (NCC)
  2. Perceptron
  3. ~~K-Nearest Neighbor (KNN)~~
- These algorithms are extremely powerful
- Often they can compete with complex algorithms
- Some aspects can be motivated by biological cognition

Prototypes: Psychological Models of Abstract Ideas



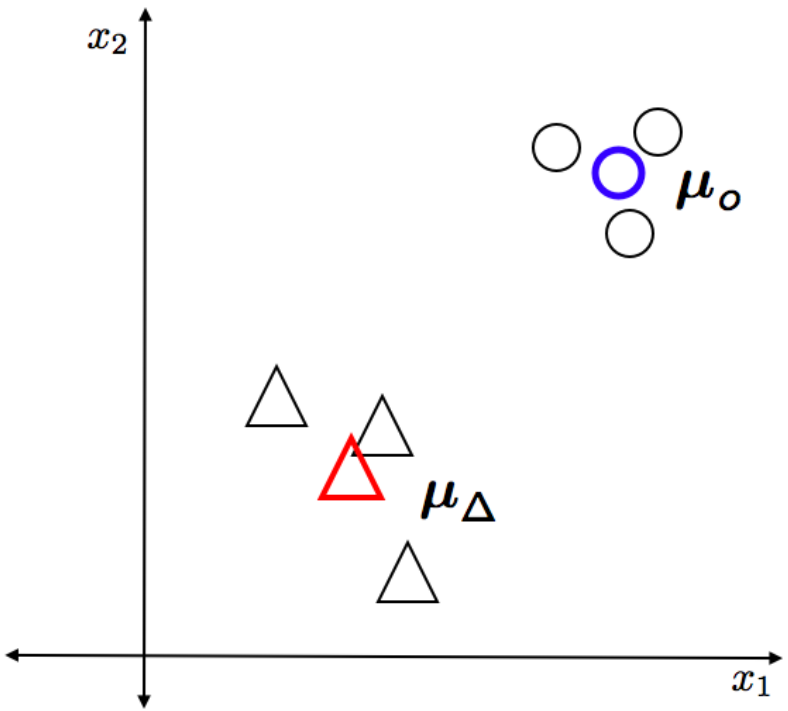
Psychologists postulated that we learn **prototypes** [Jaekel, 2007; Posner and Keele, 1968]

Toy data example:

Two dimensional input $\mathbf{x} \in \mathbb{R}^2$

Two *classes* of data, Δ and $\circ$

Prototypes: Psychological Models of Abstract Ideas



Prototypes $\boldsymbol{\mu}_\Delta$ and $\boldsymbol{\mu}_o$ can be the class means

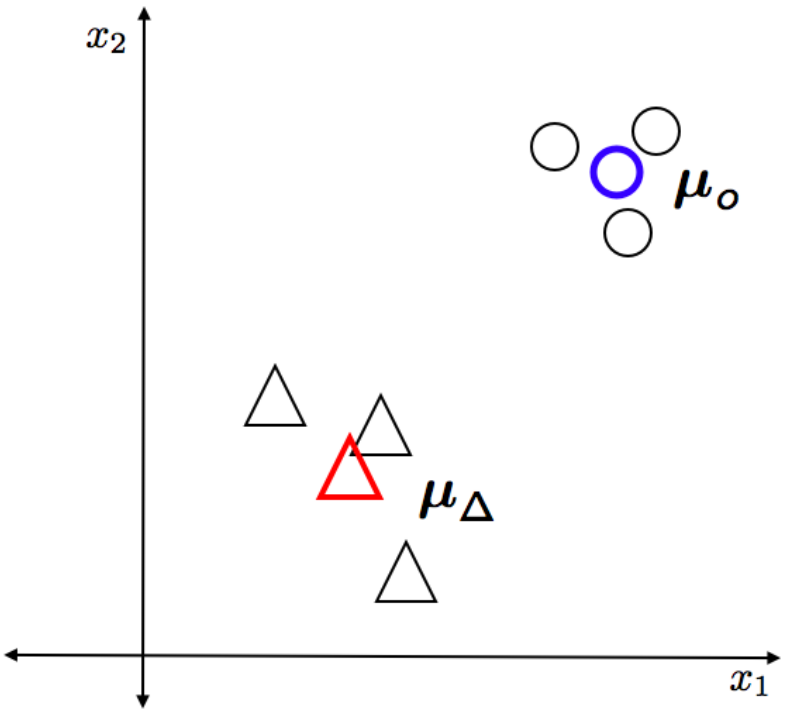
$$\boldsymbol{\mu}_\Delta = 1/N_\Delta \sum_{n}^{N_\Delta} \mathbf{x}_{\Delta,n}$$

$$\boldsymbol{\mu}_o = 1/N_o \sum_{n}^{N_o} \mathbf{x}_{o,n}$$

Distance from w_Δ to new data x

$$\|\boldsymbol{\mu}_\Delta - \mathbf{x}\|_2$$

Prototypes: Psychological Models of Abstract Ideas



For new data x check:
Is x more similar to μ_o?

$$\|\mu_\Delta - \mathbf{x}\| > \|\mu_o - \mathbf{x}\|$$

yes? $\rightarrow$ **x** belongs to μ_o

no? $\rightarrow$ **x** belongs to μ_Δ

This is called a
nearest centroid classifier

Nearest Centroid Classification Algorithm (Batch Mode)

Algorithm 1 Computation of Class-Centroids

Require: data $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^D$, labels $y_1, \ldots, y_N \in \{1, \ldots, K\}$
Ensure: Class means $\boldsymbol{\mu}_k,\ k \in \{1, \ldots, K\}$

```
# Initialize means and counters for each class
# Computation of class means
for Class k = 1, ..., K do
    μ_k = (1/N_k) Σ_{i=1}^{N_k} x_i
end for
```

Batch Computations vs. Streaming

Solutions for algorithms can be obtained

- In Batch Mode:
  - Use all available data at once
  - Requires to store all data in memory
- In Streaming Mode:
  - Use one data point at a time
  - Requires to store **only** centroids

Iterative Computation of the Mean

Given the mean $\boldsymbol{\mu}_{N-1}$ computed from $N-1$ samples we want to update $\boldsymbol{\mu}_{N-1}$ with the Nth sample $\mathbf{x}_N$ to obtain $\boldsymbol{\mu}_N$

$$
\begin{aligned}
\boldsymbol{\mu}_N &= \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n \\
&= \frac{1}{N}\sum_{n=1}^{N-1}\mathbf{x}_n + \frac{1}{N}\mathbf{x}_N \\
&= \frac{N-1}{N}\underbrace{\frac{1}{N-1}\sum_{n=1}^{N-1}\mathbf{x}_n}_{\boldsymbol{\mu}_{N-1}} + \frac{1}{N}\mathbf{x}_N \\
&= \frac{N-1}{N}\boldsymbol{\mu}_{N-1} + \frac{1}{N}\mathbf{x}_N
\end{aligned}
$$

Nearest Centroid Classification Algorithm (Streaming)

Algorithm 2 Iterative computation of Class-Centroids

Require: data $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^D$, labels $y_1, \ldots, y_N \in \{1, \ldots, K\}$
Ensure: Class means $\boldsymbol{\mu}_k, \ k \in \{1, \ldots, K\}$

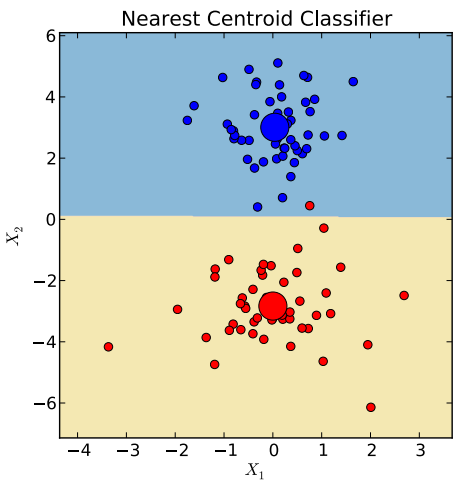
```
# Initialize means and counters for each class
∀ k :   μ_k = I · 0,  N_k = 0
# Iterative computation of class means
for Data point i = 1, ..., N do
    # Update means and counters
    k = y_i
    μ_k = N_k/(N_k+1) μ_k + 1/(N_k+1) x_i
    N_k = N_k + 1
end for
```

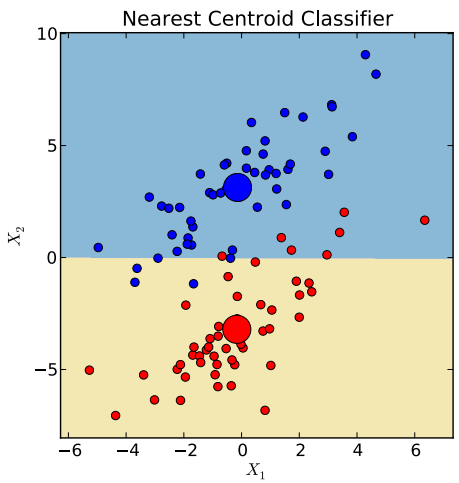
Nearest Centroid Classification

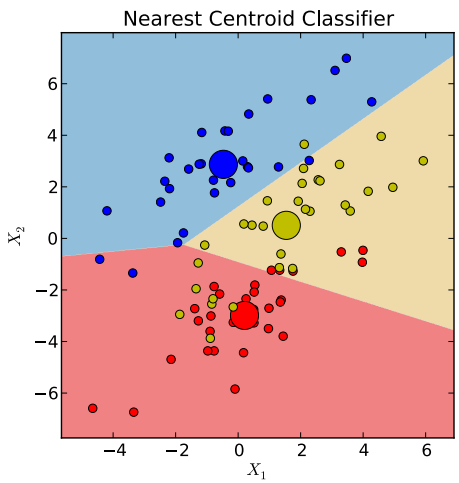
Algorithm 3 Nearest Centroid Prediction

Require: Data point $\mathbf{x} \in \mathbb{R}^D$, class centroids $\boldsymbol{\mu}_k,\ k \in \{1, \ldots, K\}$
Ensure: Class membership k^*
1: # Compute nearest class centroid in discriminative subspace
2: $k^* = \text{argmin}_k \|\boldsymbol{\mu}_k - \mathbf{x}\|_2$.

Toy Data Example NCC







From Prototypes to Linear Classification

$$\begin{aligned} \text{distance}(\mathbf{x}, \boldsymbol{\mu}_\Delta) &> \text{distance}(\mathbf{x}, \boldsymbol{\mu}_o) \\ \|\mathbf{x} - \boldsymbol{\mu}_\Delta\| &> \|\mathbf{x} - \boldsymbol{\mu}_o\| \end{aligned} \tag{1}$$

From Prototypes to Linear Classification

$$
\begin{aligned}
\text{distance}(\mathbf{x}, \boldsymbol{\mu}_\Delta) &> \text{distance}(\mathbf{x}, \boldsymbol{\mu}_o) \qquad (1)\\
\|\mathbf{x} - \boldsymbol{\mu}_\Delta\| &> \|\mathbf{x} - \boldsymbol{\mu}_o\| \\
\Leftrightarrow \|\mathbf{x} - \boldsymbol{\mu}_\Delta\|^2 &> \|\mathbf{x} - \boldsymbol{\mu}_o\|^2 \\
\Leftrightarrow \mathbf{x}^\top\mathbf{x} - 2\boldsymbol{\mu}_\Delta^\top\mathbf{x} + \boldsymbol{\mu}_\Delta^\top\boldsymbol{\mu}_\Delta &> \mathbf{x}^\top\mathbf{x} - 2\boldsymbol{\mu}_o^\top\mathbf{x} + \boldsymbol{\mu}_o^\top\boldsymbol{\mu}_o \\
\Leftrightarrow \boldsymbol{\mu}_\Delta^\top\mathbf{x} - \boldsymbol{\mu}_\Delta^2/2 &< \boldsymbol{\mu}_o^\top\mathbf{x} - \boldsymbol{\mu}_o^2/2 \\
\Leftrightarrow 0 &< \underbrace{(\boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta)^\top}_{\mathbf{w}}\mathbf{x} - 1/2\,\underbrace{(\boldsymbol{\mu}_o^\top\boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta^\top\boldsymbol{\mu}_\Delta)}_{\beta}
\end{aligned}
$$

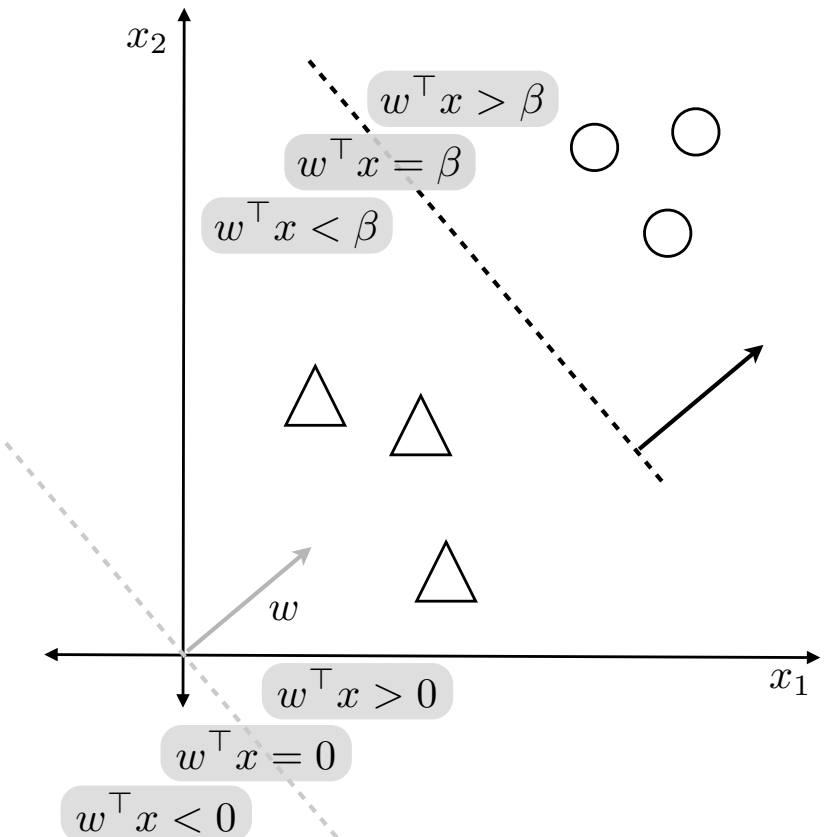
From Prototypes to Linear Classification

$$
\begin{aligned}
\text{distance}(\mathbf{x}, \boldsymbol{\mu}_\Delta) &> \text{distance}(\mathbf{x}, \boldsymbol{\mu}_o) \\
\|\mathbf{x} - \boldsymbol{\mu}_\Delta\| &> \|\mathbf{x} - \boldsymbol{\mu}_o\| \\
\Leftrightarrow \|\mathbf{x} - \boldsymbol{\mu}_\Delta\|^2 &> \|\mathbf{x} - \boldsymbol{\mu}_o\|^2 \\
\Leftrightarrow \mathbf{x}^\top\mathbf{x} - 2\boldsymbol{\mu}_\Delta^\top\mathbf{x} + \boldsymbol{\mu}_\Delta^\top\boldsymbol{\mu}_\Delta &> \mathbf{x}^\top\mathbf{x} - 2\boldsymbol{\mu}_o^\top\mathbf{x} + \boldsymbol{\mu}_o^\top\boldsymbol{\mu}_o \\
\Leftrightarrow \boldsymbol{\mu}_\Delta^\top\mathbf{x} - \boldsymbol{\mu}_\Delta^2/2 &< \boldsymbol{\mu}_o^\top\mathbf{x} - \boldsymbol{\mu}_o^2/2 \\
\Leftrightarrow 0 &< \underbrace{(\boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta)^\top}_{\mathbf{w}}\mathbf{x} - 1/2\,\underbrace{(\boldsymbol{\mu}_o^\top\boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta^\top\boldsymbol{\mu}_\Delta)}_{\beta}
\end{aligned}
\tag{1}
$$

Linear Classification

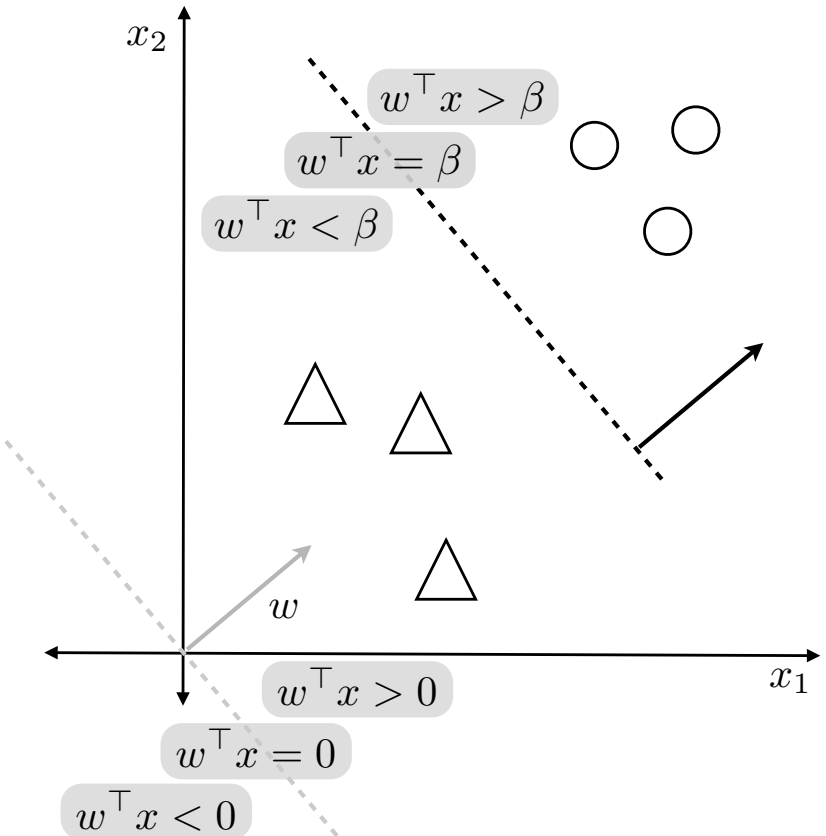
$$
\mathbf{w}^\top\mathbf{x} - \beta = \begin{cases} > 0 & \text{if } \mathbf{x} \text{ belongs to class } \circ \\ < 0 & \text{if } \mathbf{x} \text{ belongs to class } \Delta \end{cases}
\tag{2}
$$

Linear Classification



$$\mathbf{w}^\top \mathbf{x} - \beta = \begin{cases} > 0 & \text{if } \mathbf{x} \text{ belongs to } o \\ < 0 & \text{if } \mathbf{x} \text{ belongs to } \Delta \end{cases}$$

Linear Classification



$$\mathbf{w}^\top \mathbf{x} - \beta = \begin{cases} > 0 & \text{if } \mathbf{x} \text{ belongs to } o \\ < 0 & \text{if } \mathbf{x} \text{ belongs to } \Delta \end{cases}$$

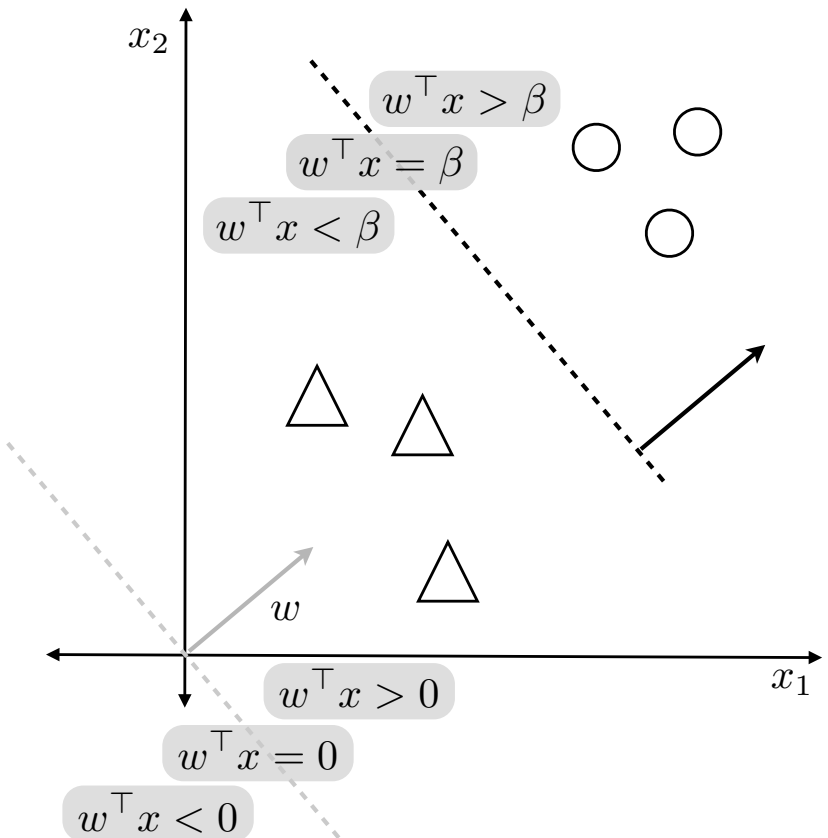
The *offset* β can be included in $\mathbf{w}$

$$\tilde{\mathbf{x}} \leftarrow \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \qquad \tilde{\mathbf{w}} \leftarrow \begin{bmatrix} -\beta \\ \mathbf{w} \end{bmatrix}$$

such that

$$\tilde{\mathbf{w}}^\top \tilde{\mathbf{x}} = \mathbf{w}^\top \mathbf{x} - \beta.$$

Linear Classification



What is a good **w**?

$\rightarrow$ We need an **error function** that tells us how good **w** is.

Lecture-4-Feature-Extraction

May 7, 2019

1 Machine Learning

Lecture 4
Feature Extraction, Scikit Learn API, Model Evaluation

2 Overview

- Feature Extraction
- Scikit Learn API: Estimators, Pipelines
- Model Evaluation

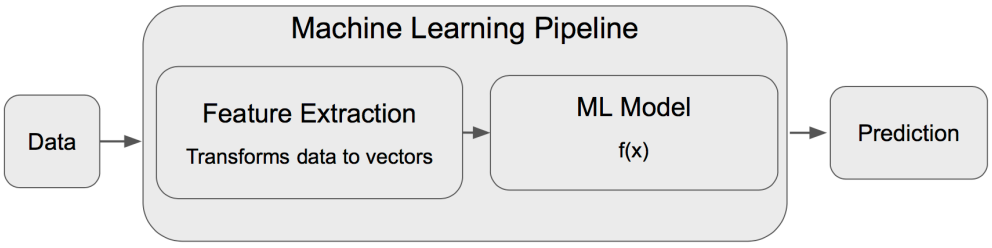
3 Machine Learning Pipelines

4 Scikit Learn API

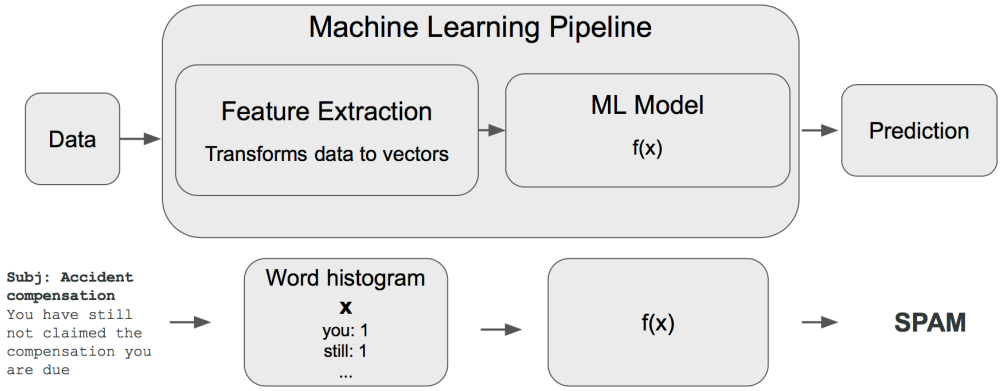
Scikit Learn is a popular machine learning library.
Its API was copied in many other libraries (e.g. spark.ml).
It offers most feature transformations, ML models, metrics and many useful tools.

4.1 Estimators

- implement `set_params()` and `get_params()`
- implement `fit` method
- Examples for what `fit` does:



ml-pipeline-2.png



ml-pipeline-2.png

  - sklearn.feature_extraction.text.CountVectorizer: learns dictionary
  - sklearn.linear_model.Perceptron: learns weight vector and Pipelines

- implement `transform` method
- Examples for what `transform` does:
  - sklearn.feature_extraction.text.CountVectorizer: extracts n-gram counts
  - sklearn.linear_model.Perceptron: projects data onto weight vector

5 Machine Learning Pipelines

5.1 Pipelines

Estimators with standardized API can be chained in a Pipeline.
This allows to chain all feature extraction and classification into one object
Pipelines behave like Estimators themselves, with `fit` and `transform` methods
All parameters of a Pipeline can be optimized jointly

```
In [1]: from sklearn.pipeline import Pipeline
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.neighbors import NearestCentroid

        reviews = [
                "This is a horrible movie",
                "a great movie",
                "a fantastic movie",
            ]
        sentiment = [-1, 1, 1]

        text_clf = Pipeline([('vect', CountVectorizer()),
                             ('clf', NearestCentroid())])

        text_clf.fit(reviews, sentiment)

        text_clf.predict(reviews)
```

```
Out[1]: array([-1,  1,  1])

In [2]: text_clf.steps

Out[2]: [('vect', CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                  dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                  lowercase=True, max_df=1.0, max_features=None, min_df=1,
                  ngram_range=(1, 1), preprocessor=None, stop_words=None,
                  strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                  tokenizer=None, vocabulary=None)),
         ('clf', NearestCentroid(metric='euclidean', shrink_threshold=None))]

In [6]: text_clf.steps[0][1].transform(reviews)

Out[6]: <3x6 sparse matrix of type '<class 'numpy.int64'>'
                with 8 stored elements in Compressed Sparse Row format>

In [8]: text_clf.steps[0][1].transform(reviews).toarray()

Out[8]: array([[0, 0, 1, 1, 1, 1],
               [0, 1, 0, 0, 1, 0],
               [1, 0, 0, 0, 1, 0]])

In [10]: x = text_clf.steps[0][1].transform(reviews)
         text_clf.steps[1][1].predict(x)

Out[10]: array([-1,  1,  1])
```

6 Feature Extraction

- Feature extraction describes the transformations from **data** to **vectors**
- Extracting good features is **more important** than choosing a ML model
- We will cover some standard feature extractors, but there are many more
- Often the best feature extractors include domain knowledge by human experts
- Feature extractors have to be optimized along with all other model parameters

7 Feature Extraction

- Continuous Features
- Categorical Features
- Text
- Images

7.1 Continuous Features

- Continuous features: $x \in R^d$
- For many models continuous features don't need to be transformed
- For some models it is necessary or beneficial to **normalize** continuous features
  - When optimizing with stochastic gradient descent
  - When regularizing models: to control regularization

7.2 Normalization: z-scoring

Given a feature $x \in R^1$ (and for multivariate x analogously) there are several standard normalization options:

Standard scaling / z-scoring: compute mean μ_x and standard deviation σ_x of x and compute

$$x \leftarrow \frac{(x - \mu_x)}{\sigma_x}$$

- Resulting variable has zero mean and unit variance
- See sklearn.preprocessing.StandardScaler

```
In [1]: from sklearn.preprocessing import StandardScaler
        data = [[0, 0], [0, 0], [1, 1], [1, 1]]
        scaler = StandardScaler()
        scaler.fit(data)
        print("Estimated Mean: {}".format(scaler.mean_))
        scaler.transform([[.5, .5], [2,2]])
```

```
Estimated Mean: [0.5 0.5]
```

```
Out[1]: array([[0., 0.],
               [3., 3.]])
```

7.3 Normalization: Min-Max Scaling

Min-max scaling: compute min $\min_x$ and max $\max_x$ of x

$$x \leftarrow \frac{(x - \min_x)}{\max_x - \min_x}$$

- Resulting variable is in range [0,1] (or any other range)
- See sklearn.preprocessing.MinMaxScaler

7.4 Categorical Features

Categorical features are variables $x \in \{0, 1, 2, \ldots, N\}$ taking one value of a set of cardinality N without an implicit ordering e.g.: - colors (red, green, blue) - user ids - Movie ids

Often used feature transformations are

- One-hot encoding
- More recently for neural networks: low dimensional embeddings

7.5 One-hot encoding

Given a set of values [red, green, blue], we transform the data into

- red $\leftarrow [1,0,0]$
- green $\leftarrow [0,1,0]$
- blue $\leftarrow [0,0,1]$

Sets of categorical variables can be represented as sum over single value vectors.
Examples: bag-of-words vectors

```
In [2]: from sklearn.preprocessing import OneHotEncoder
        enc = OneHotEncoder()
        X = [['red'], ['green'], ['blue'], ['red']]
        enc.fit(X)
        enc.transform([['red'], ['green'], ['blue']]).toarray()

Out[2]: array([[0., 0., 1.],
               [0., 1., 0.],
               [1., 0., 0.]])
```

7.6 One-hot encoding: Problems

- Cardinality needs to be estimated upfront
- New items / categories cannot be represented
- Similarity information is lost (light-blue and blue as different as black and white)

8 Feature Extraction for Text

- Bag-of-Words
- Hashed Bag-of-Words
- (Continuous Bag-of-words)

8.1 Bag-of-Words Features

Count word (=token) occurrences
"if you pay peanuts , you get monkeys ."

get	if	monkeys	pay	peanuts	you	...
1	1	1	1	1	2	...

- Only accounts for word histogram
- Most of language structure lost
- Very efficient

8.2 Bag-of-Words Model with N-Grams

Count n-gram occurrences can account for some language structure

"if you pay peanuts , you get monkeys ."

get monkeys	if you	you pay	pay peanuts	...
1	1	1	1	...

```
In [14]: from sklearn.feature_extraction.text import CountVectorizer
         corpus = [
              'This is the first document.',
              'This document is the second document.',
              'And this is the third one.',
              'Is this the first document?']
         vectorizer = CountVectorizer()
         X = vectorizer.fit_transform(corpus)
         print("Vocabulary: {}".format(vectorizer.get_feature_names()))
         X.toarray()
```

```
Vocabulary: ['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third', 'this']
```

```
Out[14]: array([[0, 1, 1, 1, 0, 0, 1, 0, 1],
                [0, 2, 0, 1, 0, 1, 1, 0, 1],
                [1, 0, 0, 1, 1, 0, 1, 1, 1],
                [0, 1, 1, 1, 0, 0, 1, 0, 1]], dtype=int64)
```

8.3 Character N-Grams

Instead of words as tokens you can take characters as tokens

"if you pay peanuts , you get monkeys ."

g	e	t	ge	et	...
1	1	1	1	1	...

Very powerful for language-independent models!

8.4 Term Frequency / Inverse Document Frequency

Frequently occurring words (like `the`) are often not very meaningful and are often weighted down by the **inverse document frequency**:

- Term frequency:

  $ tf(t,d) $: Frequency of a term t in document d

- Inverse document frequency:

  $\text{idf}(t,d) = \log \frac{|\text{Number of documents}|}{|\text{Number of Documents containing } t|}$

8.5 Alternatively: Stopwords

Just exclude some frequent meaningless words from the text.

8.6 N-Grams are expensive

V = number of tokens in your vocabulary (usually 1e4 to 1e6 for good models) - unigrams: all tokens

```
Memory requirement for n-gram counts: $V$
```

- bigrams: all ordered pairs of **two** tokens
  Memory requirement for n-gram counts: V^2
- trigrams: all ordered pairs of **three** tokens
  Memory requirement for n-gram counts: V^3
- …

See Google's n-gram corpus (up to 5-grams for many languages).

8.7 ~~Hashed N-Grams~~

- ~~Instead of counting all n-grams: Use a HashMap~~
- ~~Each token gets hashed, count of hash-bucket incremented~~
- ~~Advantages:~~
  - ~~Full control over memory footprint (often small HashMaps work surprisingly well)~~
  - ~~No need to compute a dictionary~~
- ~~Disadvantages:~~
  - ~~Hash collisions: multiple n-gram map to the same bucket~~
  - ~~No Tf-Idf weighting:~~

~~See Scikit-Learn's HashingVectorizer~~

~~8.8 Feature Extraction for Images~~

- ~~Classical features (e.g. HOG)~~
- ~~CNN features~~

Lecture-5-Model-Evaluation

May 14, 2019

1 Machine Learning

Classification Metrics and Model Evaluation

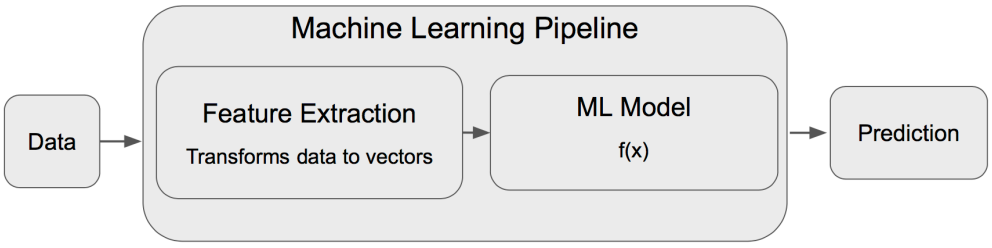
2 Machine Learning Pipelines

3 Model Evaluation

- Machine Learning models are evaluated by comparing the predictions $f(x) = \hat{y}$ and the target values y
- Useful Metrics
  - Regression (for continuous predictions - covered later)
  - Classification
- As ML models can memorize any training data set, **all metrics must always be computed using cross-validation**

3.1 Classification Metrics

- Accuracy
- Precision
- Recall
- F1



ml-pipeline-2.png

3.2 Accuracy

Defined as

$$\frac{\text{number of correct assignments}}{\text{number of data points}}$$

Problem: Imbalanced classes

```
In [53]: from sklearn.metrics import accuracy_score

         y_predicted = np.array([1, 0, 0, 0, 0])
         y_true = np.array([0, 1, 0, 0, 0])

In [54]: print(f"Accuracy: {accuracy_score(y_true, y_predicted)}")
```

```
Accuracy: 0.6
```

3.3 Precision

$$\text{precision} = \frac{|\{\text{relevant instances}\} \cap \{\text{predicted instances}\}|}{|\{\text{predicted instances}\}|}$$

```
In [55]: from sklearn.metrics import precision_score

         y_predicted = np.array([1, 0, 0, 0, 0])
         y_true = np.array([0, 1, 0, 0, 0])

In [56]: print(f"Accuracy: {accuracy_score(y_true, y_predicted)}")
         print(f"Precision: {precision_score(y_true, y_predicted)}")
```

```
Accuracy: 0.6
Precision: 0.0
```

```
In [57]: y_predicted = np.array([1, 1, 0, 0, 0])
         y_true = np.array([0, 1, 0, 0, 0])

In [58]: print(f"Accuracy: {accuracy_score(y_true, y_predicted)}")
         print(f"Precision: {precision_score(y_true, y_predicted)}")
```

```
Accuracy: 0.8
Precision: 0.5
```

3.4 Recall

$$\text{recall} = \frac{|\{\text{relevant instances}\} \cap \{\text{predicted instances}\}|}{|\{\text{relevant instances}\}|}$$

```
In [59]: from sklearn.metrics import recall_score

         y_predicted = np.array([1, 0, 0, 0, 0])
         y_true = np.array([0, 1, 0, 0, 0])
```

```
In [61]: print(f"Accuracy: {accuracy_score(y_true, y_predicted)}")
         print(f"Precision: {precision_score(y_true, y_predicted)}")
         print(f"Recall: {recall_score(y_true, y_predicted)}")
```

```
Accuracy: 0.6
Precision: 0.0
Recall: 0.0
```

```
In [62]: y_predicted = np.array([1, 1, 0, 0, 0])
         y_true = np.array([0, 1, 0, 0, 0])
```

```
In [63]: print(f"Accuracy: {accuracy_score(y_true, y_predicted)}")
         print(f"Precision: {precision_score(y_true, y_predicted)}")
         print(f"Recall: {recall_score(y_true, y_predicted)}")
```

```
Accuracy: 0.8
Precision: 0.5
Recall: 1.0
```

3.5 Precision and Recall

Source Wikipedia page on Precision and Recall

3.6 F1 Score

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

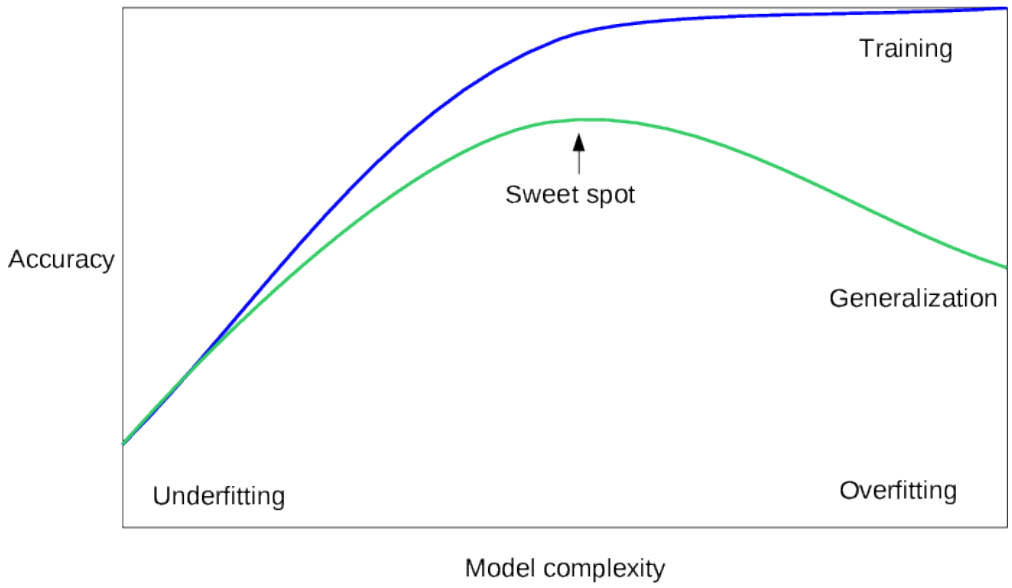
```
In [71]: from sklearn.metrics import f1_score

         y_predicted = np.array([1, 0, 0, 0, 0])
         y_true = np.array([0, 1, 0, 0, 0])
```

```
In [72]: print(f"Precision: {precision_score(y_true, y_predicted)}")
         print(f"Recall: {recall_score(y_true, y_predicted)}")
         print(f"F1: {f1_score(y_true, y_predicted)}")
```

```
Precision: 0.0
Recall: 0.0
F1: 0.0
```

```
In [66]: y_predicted = np.array([1, 1, 0, 0, 0])
         y_true = np.array([0, 1, 0, 0, 0])
```

```
In [70]: print(f"Precision: {precision_score(y_true, y_predicted)}")
         print(f"Recall: {recall_score(y_true, y_predicted)}")
         print(f"F1: {f1_score(y_true, y_predicted)}")
```



model_complexity

```
Precision: 0.5
Recall: 1.0
F1: 0.6666666666666666
```

4 Model Evaluation

4.1 Cross-Validation

- ML models can memorize any data set (**overfitting**)
- But we want our models to perform well on new data (**generalization of learned rules**)
- Cross-validation emulates the setting of new unseen data:
  - Split data in training and test
  - Train model on training set
  - Test model on test set

4.2 Simplest Cross-Validation for Model Evaluation

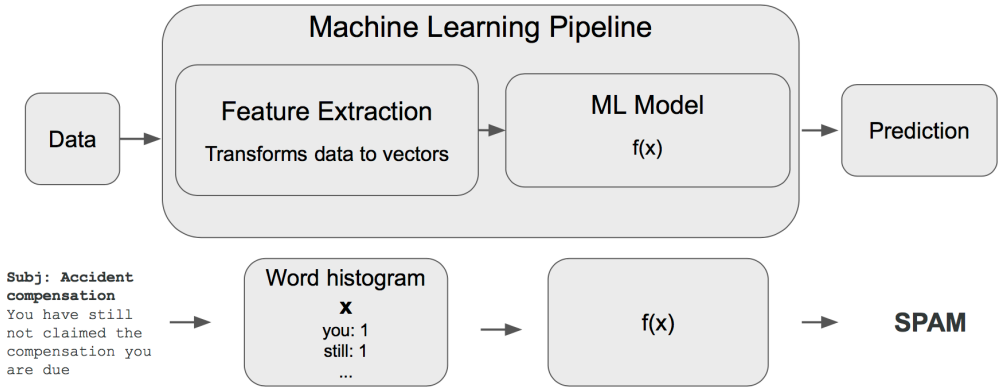
Split data into **two folds** and train / test **just once**

Fold 1 Test
Fold 2 Train

4.3 Example 3-Fold Cross-Validation for Model Evaluation:

Split data into three folds:

~~4~~
Fold 1
Fold 2



ml-pipeline-2.png

Fold 1	Train
Fold 2	Test
Fold 3	Train

4.3.3 Fold 3 is Test Data

Fold 1	Train
Fold 2	Train
Fold 3	Test

5 Recap: Cross-Validation for Model Evaluation

- Split dataset in training and test set
- Requires **at least two** different folds / partitions / data sets (train and test)
- Fast Cross-validation:
  - Make two partitions, train and test
- Cross-validation with best generalization performance estimates:
  - k-fold

6 Example Application: Classifying Parliament Speeches

- Task: Political bias prediction from text data
- Data: parliament speeches in German Bundestag

7 Machine Learning Pipeline

In [3]:
```
import os, gzip
import pandas as pd
import numpy as np
import urllib.request

import warnings
warnings.filterwarnings('ignore')

DATADIR = "data"

if not os.path.exists(DATADIR):
    os.mkdir(DATADIR)

file_name = os.path.join(DATADIR, 'bundestags_parlamentsprotokolle.csv.gzip')
if not os.path.exists(file_name):
    url_data = 'https://www.dropbox.com/s/1nlbfehnrwwa2zj/bundestags_parlamentsprotokol
    urllib.request.urlretrieve(url_data, file_name)

df = pd.read_csv(gzip.open(file_name), index_col=0).sample(frac=1)

alle_sprecher = df.sprecher.unique()
parteien = df.partei.unique()
partei_farben = {'cducsu':'black', 'linke':'purple', 'spd':'red', 'gruene':'green', 'f
```

In [2]:
```
df[:5]
```

Out[2]:

	sitzung	wahlperiode	sprecher	text	partei
30289	76	18	Dr.ăKonstantin von Notz	Ich kann Ihnen schon sagen, wie das Ihre Unabh...	gruene
11201	133	17	Peter Wichtel	Denn damit haben Sie am Ende genau das, was hi...	cducsu
24223	10	18	Dr.ăEva Högl	Unser Koalitionsvertrag ist voller guter Ideen...	spd
34597	127	18	Dr. Joachim Pfeiffer	Frau Präsidentin! Liebe Kolleginnen und Kolleg...	cducsu
7715	96	17	Bettina Kudla	Zur Gläubigerbeteiligung - auch dieses Thema w...	cducsu

In [31]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neighbors import NearestCentroid
from sklearn.linear_model import SGDClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import confusion_matrix, classification_report
```

```
In [22]: # Put some data aside for model evaluation
         train_data, test_data, train_labels, test_labels = train_test_split(df['text'], df['pa

         ncc_clf = Pipeline([('vect', TfidfVectorizer(max_features=int(1e8))),
                             ('clf', NearestCentroid())]).fit(train_data, train_labels)

         logreg_clf = Pipeline([('vect', TfidfVectorizer(max_features=int(1e8))),
                                ('clf', SGDClassifier())]).fit(train_data, train_labels)
```

7.1 Evaluating NCC on Training Data

```
In [23]: ncc_predictions = ncc_clf.predict(train_data)
         print(classification_report(ncc_predictions, train_labels))
```

```
              precision    recall  f1-score   support

      cducsu       0.49      0.65      0.56      9612
         fdp       0.42      0.21      0.29      5275
      gruene       0.43      0.34      0.38      6323
       linke       0.62      0.37      0.46      8222
         spd       0.28      0.48      0.35      5511

   micro avg       0.44      0.44      0.44     34943
   macro avg       0.45      0.41      0.41     34943
weighted avg       0.47      0.44      0.43     34943
```

7.2 Evaluating NCC on Test Data

```
In [27]: ncc_predictions_test = ncc_clf.predict(test_data)
         print(classification_report(ncc_predictions_test, test_labels))
```

```
              precision    recall  f1-score   support

      cducsu       0.48      0.61      0.54      2498
         fdp       0.35      0.18      0.23      1365
      gruene       0.40      0.32      0.35      1569
       linke       0.54      0.34      0.42      1894
         spd       0.26      0.45      0.33      1410

   micro avg       0.41      0.41      0.41      8736
   macro avg       0.41      0.38      0.38      8736
weighted avg       0.42      0.41      0.40      8736
```

7.3 Evaluating Logistic Regression on Training Data

```
In [26]: logreg_predictions = logreg_clf.predict(train_data)
         print(classification_report(logreg_predictions, train_labels))
```

```
              precision    recall  f1-score   support

      cducsu       0.96      0.66      0.78     18852
         fdp       0.27      0.98      0.42       738
      gruene       0.69      0.93      0.79      3708
       linke       0.85      0.87      0.86      4768
         spd       0.62      0.85      0.72      6877

   micro avg       0.76      0.76      0.76     34943
   macro avg       0.68      0.86      0.71     34943
weighted avg       0.84      0.76      0.77     34943
```

7.4 Evaluating Logistic Regression on Training Data

```
In [28]: logreg_predictions_test = logreg_clf.predict(test_data)
         print(classification_report(logreg_predictions_test, test_labels))
```

```
              precision    recall  f1-score   support

      cducsu       0.91      0.55      0.69      5273
         fdp       0.04      0.62      0.08        50
      gruene       0.33      0.62      0.43       658
       linke       0.63      0.64      0.63      1177
         spd       0.37      0.57      0.45      1578

   micro avg       0.57      0.57      0.57      8736
   macro avg       0.46      0.60      0.46      8736
weighted avg       0.72      0.57      0.61      8736
```

7.5 Summary Comparison NCC and Logistic Regression

Nearest Centroid Classifiers - NCC is a simple model - Overall performance not great - But not a lot of overfitting

Logistic Regression - More powerful model - Better prediction performance - More overfitting

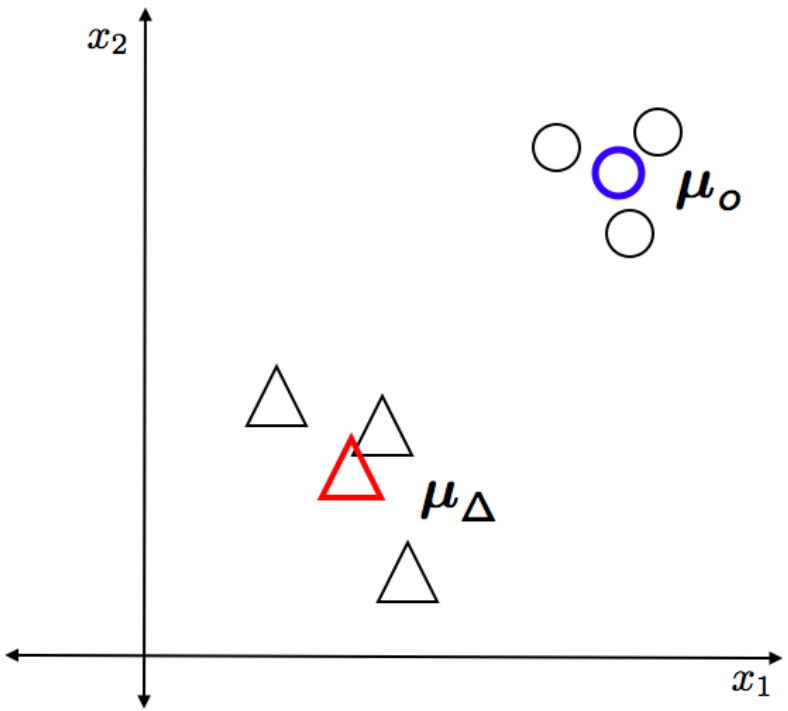
Machine Learning

Lecture 7
Perceptrons

Felix Bießmann

Beuth University & Einstein Center for Digital Future

Recap: Nearest Centroid Classification



Prototypes μ_Δ and μ_o can be the class means

$$\mu_\Delta = 1/N_\Delta \sum_n^{N_\Delta} \mathbf{x}_{\Delta,n}$$

$$\mu_o = 1/N_o \sum_n^{N_o} \mathbf{x}_{o,n}$$

Distance from w_Δ to new data x

$$\|\mu_\Delta - \mathbf{x}\|_2$$

From Prototypes to Linear Classification

$$\begin{aligned} \text{distance}(\mathbf{x}, \boldsymbol{\mu}_\Delta) &> \text{distance}(\mathbf{x}, \boldsymbol{\mu}_o) \\ \|\mathbf{x} - \boldsymbol{\mu}_\Delta\| &> \|\mathbf{x} - \boldsymbol{\mu}_o\| \end{aligned} \tag{1}$$

From Prototypes to Linear Classification

$$
\begin{aligned}
\text{distance}(\mathbf{x}, \boldsymbol{\mu}_\Delta) &> \text{distance}(\mathbf{x}, \boldsymbol{\mu}_o) \qquad (1)\\
\|\mathbf{x} - \boldsymbol{\mu}_\Delta\| &> \|\mathbf{x} - \boldsymbol{\mu}_o\| \\
\Leftrightarrow \|\mathbf{x} - \boldsymbol{\mu}_\Delta\|^2 &> \|\mathbf{x} - \boldsymbol{\mu}_o\|^2 \\
\Leftrightarrow \mathbf{x}^\top\mathbf{x} - 2\boldsymbol{\mu}_\Delta^\top\mathbf{x} + \boldsymbol{\mu}_\Delta^\top\boldsymbol{\mu}_\Delta &> \mathbf{x}^\top\mathbf{x} - 2\boldsymbol{\mu}_o^\top\mathbf{x} + \boldsymbol{\mu}_o^\top\boldsymbol{\mu}_o \\
\Leftrightarrow \boldsymbol{\mu}_\Delta^\top\mathbf{x} - \boldsymbol{\mu}_\Delta^2/2 &< \boldsymbol{\mu}_o^\top\mathbf{x} - \boldsymbol{\mu}_o^2/2 \\
\Leftrightarrow 0 &< \underbrace{(\boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta)^\top}_{\mathbf{w}}\mathbf{x} - 1/2\,\underbrace{(\boldsymbol{\mu}_o^\top\boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta^\top\boldsymbol{\mu}_\Delta)}_{\beta}
\end{aligned}
$$

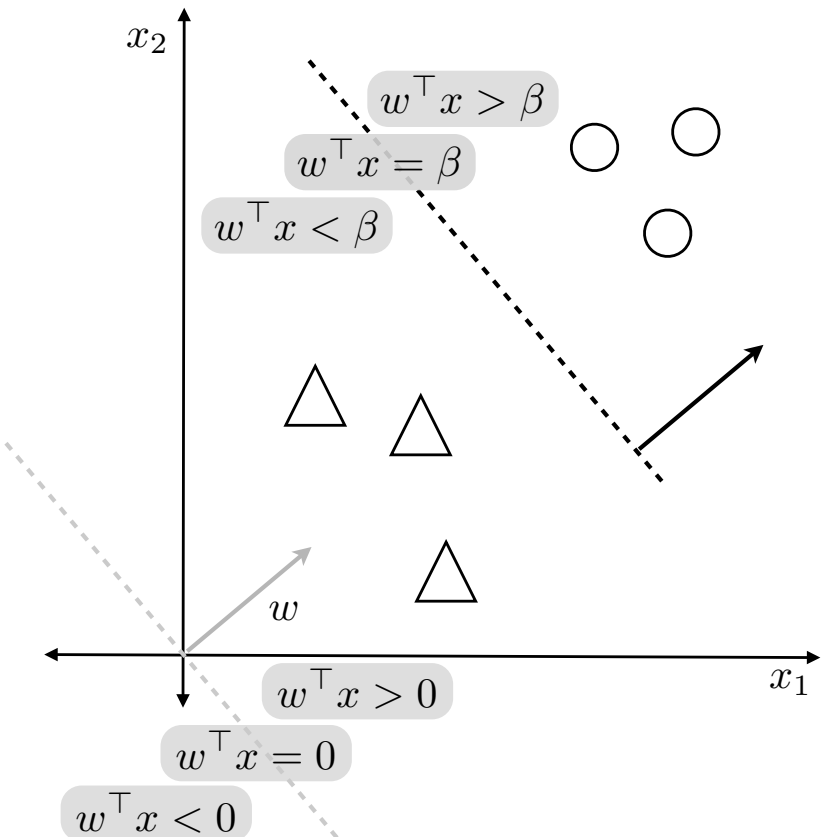
From Prototypes to Linear Classification

$$
\begin{aligned}
\text{distance}(\mathbf{x}, \boldsymbol{\mu}_\Delta) &> \text{distance}(\mathbf{x}, \boldsymbol{\mu}_o) \\
\|\mathbf{x} - \boldsymbol{\mu}_\Delta\| &> \|\mathbf{x} - \boldsymbol{\mu}_o\| \\
\Leftrightarrow \|\mathbf{x} - \boldsymbol{\mu}_\Delta\|^2 &> \|\mathbf{x} - \boldsymbol{\mu}_o\|^2 \\
\Leftrightarrow \mathbf{x}^\top\mathbf{x} - 2\boldsymbol{\mu}_\Delta^\top\mathbf{x} + \boldsymbol{\mu}_\Delta^\top\boldsymbol{\mu}_\Delta &> \mathbf{x}^\top\mathbf{x} - 2\boldsymbol{\mu}_o^\top\mathbf{x} + \boldsymbol{\mu}_o^\top\boldsymbol{\mu}_o \\
\Leftrightarrow \boldsymbol{\mu}_\Delta^\top\mathbf{x} - \boldsymbol{\mu}_\Delta^2/2 &< \boldsymbol{\mu}_o^\top\mathbf{x} - \boldsymbol{\mu}_o^2/2 \\
\Leftrightarrow 0 &< \underbrace{(\boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta)^\top}_{\mathbf{w}}\mathbf{x} - 1/2\,\underbrace{(\boldsymbol{\mu}_o^\top\boldsymbol{\mu}_o - \boldsymbol{\mu}_\Delta^\top\boldsymbol{\mu}_\Delta)}_{\beta}
\end{aligned}
\tag{1}
$$

Linear Classification

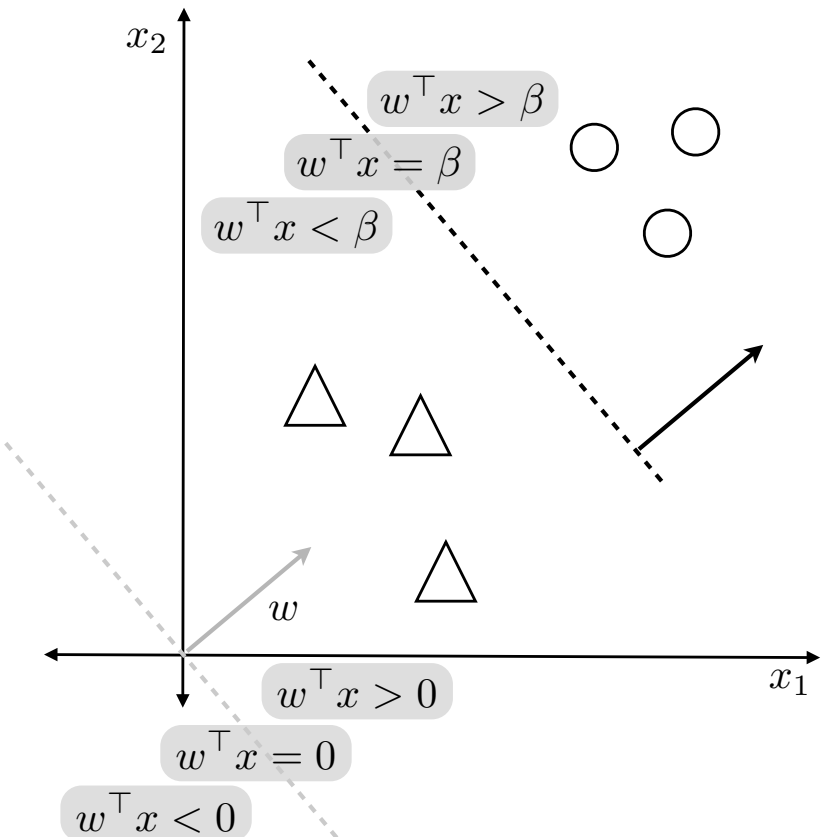
$$
\mathbf{w}^\top\mathbf{x} - \beta = \begin{cases} > 0 & \text{if } \mathbf{x} \text{ belongs to class } \circ \\ < 0 & \text{if } \mathbf{x} \text{ belongs to class } \Delta \end{cases} \tag{2}
$$

Linear Classification



$$\mathbf{w}^\top \mathbf{x} - \beta = \begin{cases} > 0 & \text{if } \mathbf{x} \text{ belongs to } o \\ < 0 & \text{if } \mathbf{x} \text{ belongs to } \Delta \end{cases}$$

Linear Classification



What is a good **w**?

$\rightarrow$ We need an **error function** that tells us how good **w** is.

Two classical Error Functions

Given data $\mathbf{x} \in \mathbb{R}^D$ and corresponding labels $y \in \{-1, +1\}$, two classical error functions $\mathcal{E}(\mathbf{x}, y, \mathbf{w})$ to find the optimal $\mathbf{w} \in \mathbb{R}^D$ are:

Error Function	Used in
$\frac{1}{2}(y - \mathbf{w}^\top \mathbf{x})^2$	Adaline [Widrow and Hoff, 1960]
$\max(0, -y\mathbf{w}^\top \mathbf{x})$	Perceptron [Rosenblatt, 1958]

Rosenblatt's Perceptron

Frank Rosenblatt
(1928-1969)

Rosenblatt proposed the **perceptron**, an artificial neural network for pattern recognition [Rosenblatt, 1958]

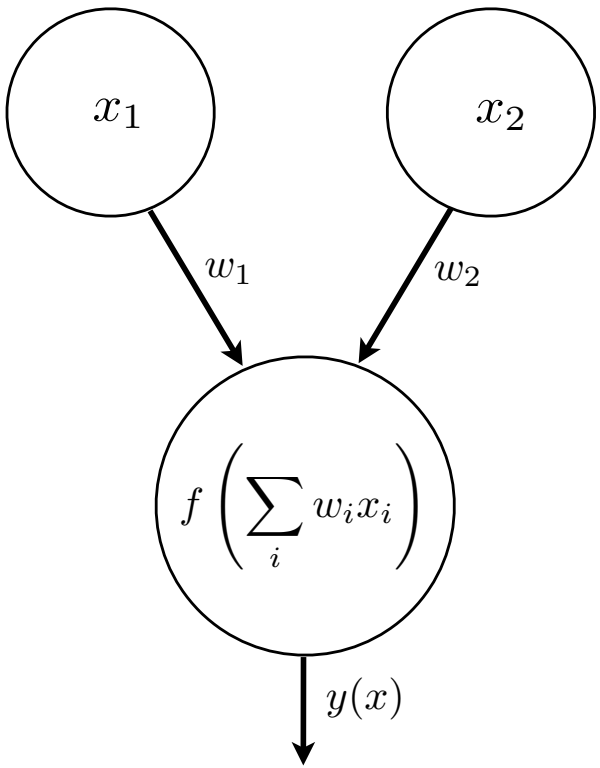
Perceptrons gave rise to the field of artificial neural networks

Rosenblatt's Perceptron

Rosenblatt studied Perceptrons, so that

" [. . .] the fundamental laws of organization which are common to all information handling systems, machines and men included, may eventually be understood."

Artificial Neural Networks



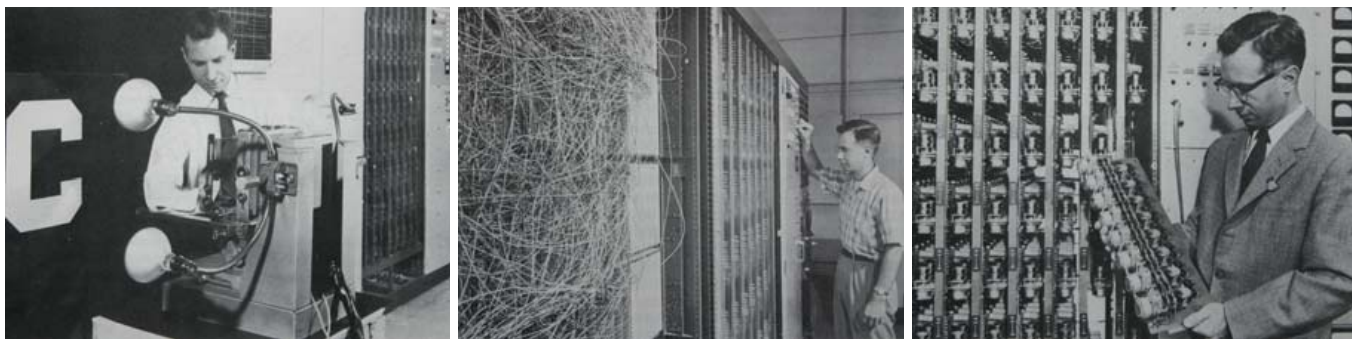
Input nodes x_i receive information

Inputs are multiplied with a weighting factor w_i and summed up

Integrated input is mapped through some (non-linear) function $f(.)$

$$f(x) = \begin{cases} +1 & \text{if } \mathbf{x} \text{ is preferred stimulus} \\ -1 & \text{if } \mathbf{x} \text{ is any other stimulus} \end{cases}$$

Rosenblatt's Perceptron



Perceptron Hardware (pictures from [Bishop, 2007])

The Perceptron Learning Algorithm

Goal Binary classification of multivariate data $\mathbf{x} \in \mathbb{R}^D$

The Perceptron Learning Algorithm

Goal Binary classification of multivariate data $\mathbf{x} \in \mathbb{R}^D$

Input Learning rate η and N tupels $(\mathbf{x}_n, y_n)$ where

$\mathbf{x}_n \in \mathbb{R}^D$ is the D-dimensional data

$y_n \in \{-1, +1\}$ is the corresponding label, such that

$$y_n = \begin{cases} +1 & \text{if } \mathbf{x}_n \text{ is a correct example} \\ -1 & \text{if } \mathbf{x}_n \text{ is } \textbf{not} \text{ a correct example} \end{cases}$$

The Perceptron Learning Algorithm

Goal Binary classification of multivariate data $\mathbf{x} \in \mathbb{R}^D$

Input Learning rate η and N tupels $(\mathbf{x}_n, y_n)$ where

$\mathbf{x}_n \in \mathbb{R}^D$ is the D-dimensional data

$y_n \in \{-1, +1\}$ is the corresponding label, such that

$$y_n = \begin{cases} +1 & \text{if } \mathbf{x}_n \text{ is a correct example} \\ -1 & \text{if } \mathbf{x}_n \text{ is } \textbf{not} \text{ a correct example} \end{cases}$$

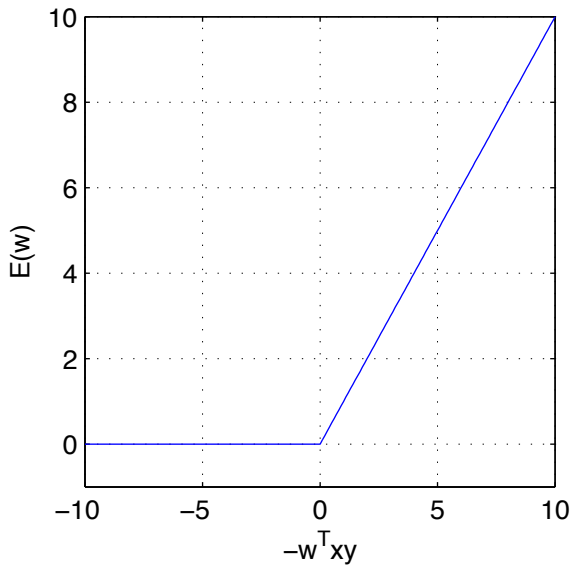
Output Weight vector $w \in \mathbb{R}^D$ such that

$$\mathbf{w}^\top \mathbf{x}_n = \begin{cases} \geq 0 & \text{if } y_n = +1 \\ < 0 & \text{if } y_n = -1 \end{cases}$$

The Perceptron Error Function

$$\mathcal{E}_{\mathcal{P}}(\mathbf{w}) = -\sum_{m \in \mathcal{M}} \mathbf{w}^{\top} \mathbf{x}_m y_m \tag{3}$$

Classification Error as Function of Weights

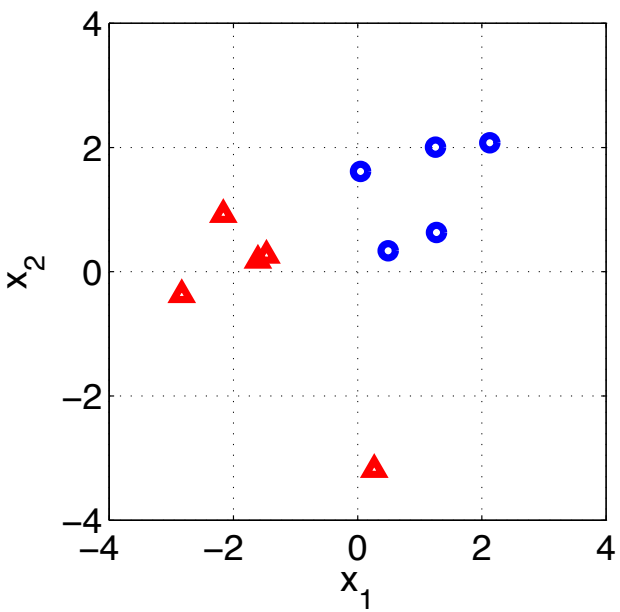


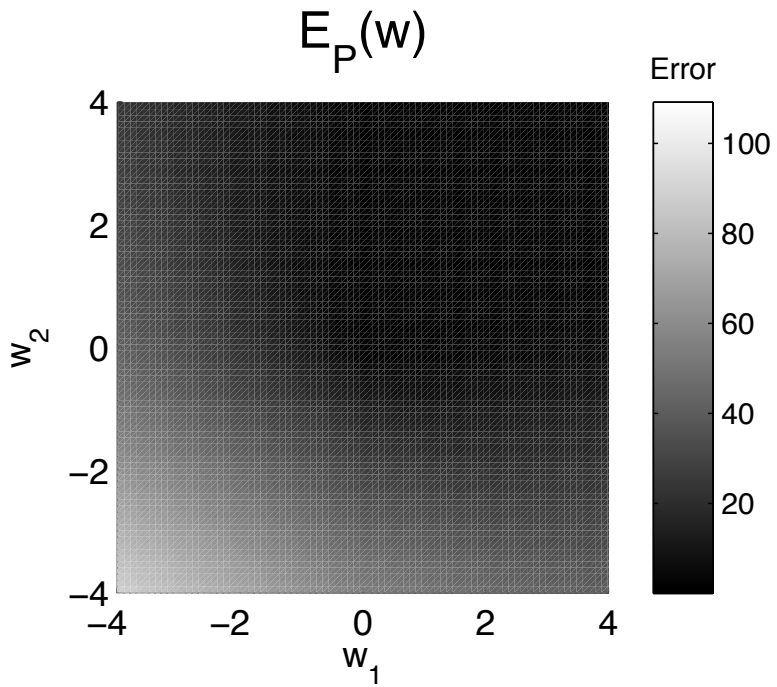
Given data $\mathbf{x} \in \mathbb{R}^D$ and corresponding labels $y \in \{-1, +1\}$ the classification error $\mathcal{E}$ is a function of the weights $\mathbf{w}$ (and the data $\mathbf{x}$, y)

$$\mathcal{E}(\mathbf{w}, \mathbf{x}_m, y_m) = -\sum_{m \in \mathcal{M}} \mathbf{w}^\top \mathbf{x}_m y_m \quad (4)$$

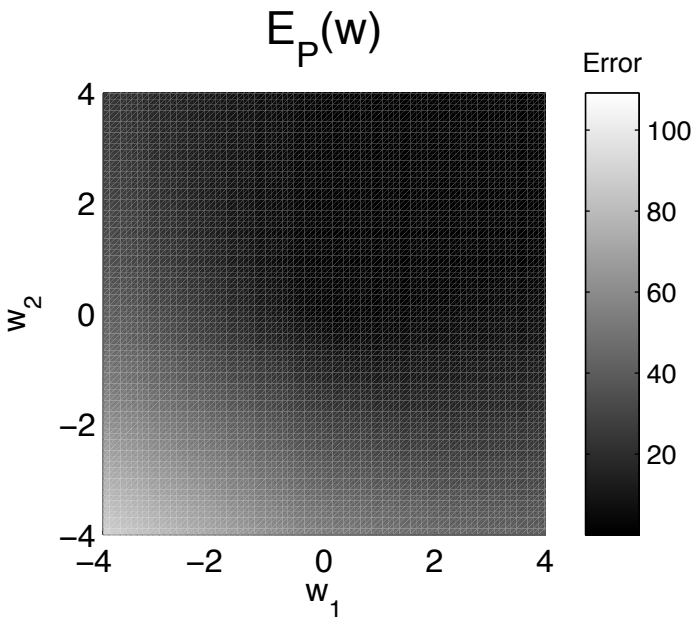
where $\mathcal{M}$ denotes the index set of all *misclassified* data $\mathbf{x}_m$

Classification Error as Function of Weights





Gradient Descent

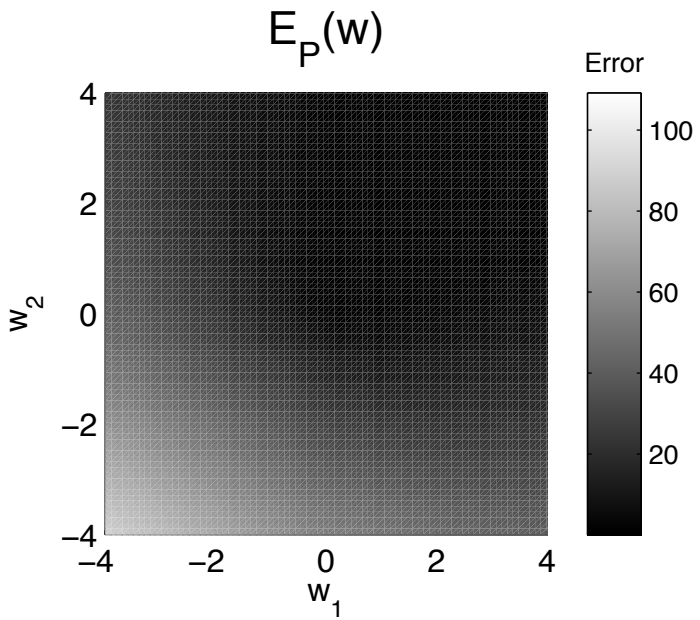


How to minimize the error function?

$$\mathcal{E}(\mathbf{w}, \mathbf{x}_m, y_m) = -\sum_{m \in \mathcal{M}} \mathbf{w}^\top \mathbf{x}_m y_m$$

$\rightarrow$ **Gradient Descent**

Gradient Descent

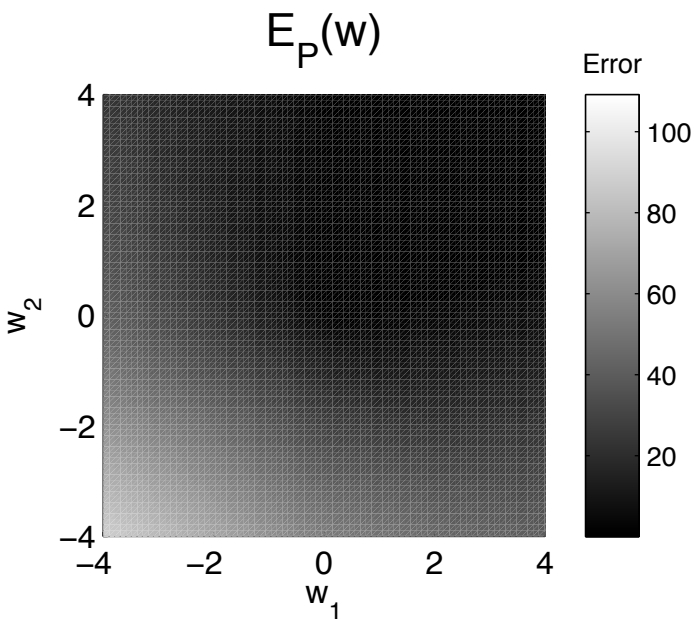


We minimize $\mathcal{E}(\mathbf{w}, \mathbf{x}_m, y_m)$ by walking in the opposite direction of the gradient.

$$\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} - \eta \frac{1}{|\mathcal{X}|} \sum_{i=1}^{|\mathcal{X}|} \nabla \mathcal{E}(\mathbf{w}, \mathbf{x}_i, y_i)$$

where $\mathcal{X}$ is the set of data points and η is called a **learning rate**.

Stochastic Gradient Descent



A noisy estimate of

$$\frac{1}{|\mathcal{X}|}\sum_{i=1}^{|\mathcal{X}|}\nabla\mathcal{E}(\mathbf{w},\mathbf{x},y)$$

is obtained by [Robbins and Monro, 1951]

$$\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} - \eta\nabla\mathcal{E}(\mathbf{w},\mathbf{x}_i,y_i)$$

Note that only $\mathbf{w}$ is stored and only one data point $\mathbf{x}_i$ and label y_i are considered at a time!

$\rightarrow$ Scales to large data sets [Bottou, 2010]

Perceptron Training

Training a Perceptron means finding weights $\mathbf{w}$ that minimize $\mathcal{E}_{\mathcal{P}}$:

$$\underset{w}{\text{argmin}} \quad \left(- \sum_{m \in \mathcal{M}} \mathbf{w}^{\top} \mathbf{x}_m y_m \right) \tag{5}$$

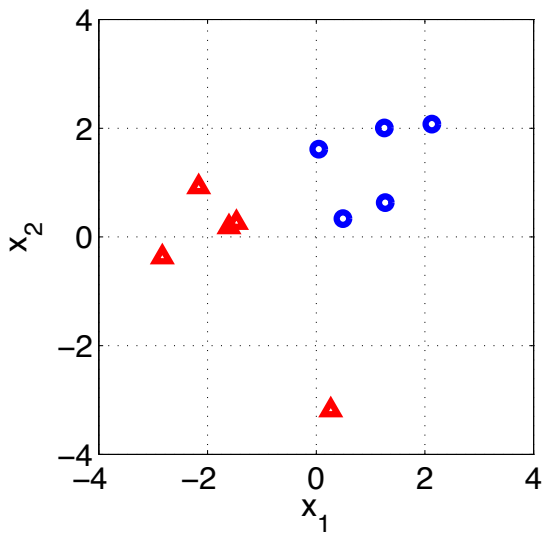
where $\mathcal{M}$ denotes the index set of all *misclassified* data $\mathbf{x}_m$

Perceptron Training

Training a Perceptron means finding weights $\mathbf{w}$ that minimize $\mathcal{E}_{\mathcal{P}}$:

$$\underset{w}{\operatorname{argmin}} \quad \left(- \sum_{m \in \mathcal{M}} \mathbf{w}^{\top} \mathbf{x}_m y_m \right) \tag{5}$$

where $\mathcal{M}$ denotes the index set of all *misclassified* data $\mathbf{x}_m$
Data $\mathbf{x} \in \mathbb{R}^2$



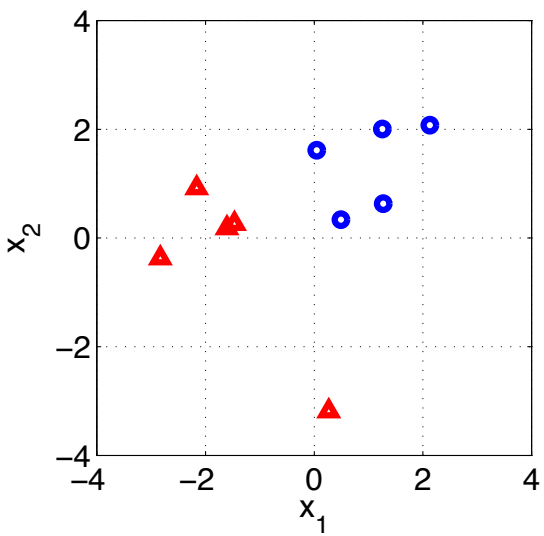
Perceptron Training

Training a Perceptron means finding weights $\mathbf{w}$ that minimize $\mathcal{E}_{\mathcal{P}}$:

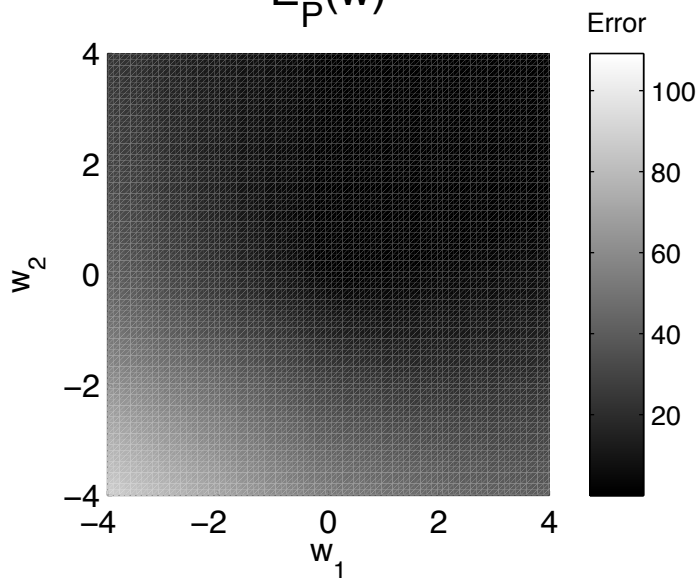
$$\underset{w}{\operatorname{argmin}} \left(- \sum_{m \in \mathcal{M}} \mathbf{w}^{\top} \mathbf{x}_m y_m \right) \tag{5}$$

where $\mathcal{M}$ denotes the index set of all *misclassified* data $\mathbf{x}_m$

Data $\mathbf{x} \in \mathbb{R}^2$



$E_P(w)$



The Perceptron Learning Algorithm

Eq. 5 can be minimized *iteratively*
using **stochastic gradient descent**
[Bottou, 2010; Robbins and Monro, 1951]

1. Initialize $\mathbf{w}^{\text{old}}$ (randomly, $1/n$, ...)
2. While there are misclassified data points $\mathbf{x}_m$

   Pick a random misclassified data point $\mathbf{x}_m$

$$\mathbf{w}^{\text{new}} \leftarrow \mathbf{w}^{\text{old}} - \eta \nabla \mathcal{E}_{\mathcal{P}}(\mathbf{w}) = \mathbf{w}^{\text{old}} + \eta \mathbf{x}_m y_m \tag{6}$$

The Perceptron Learning Algorithm

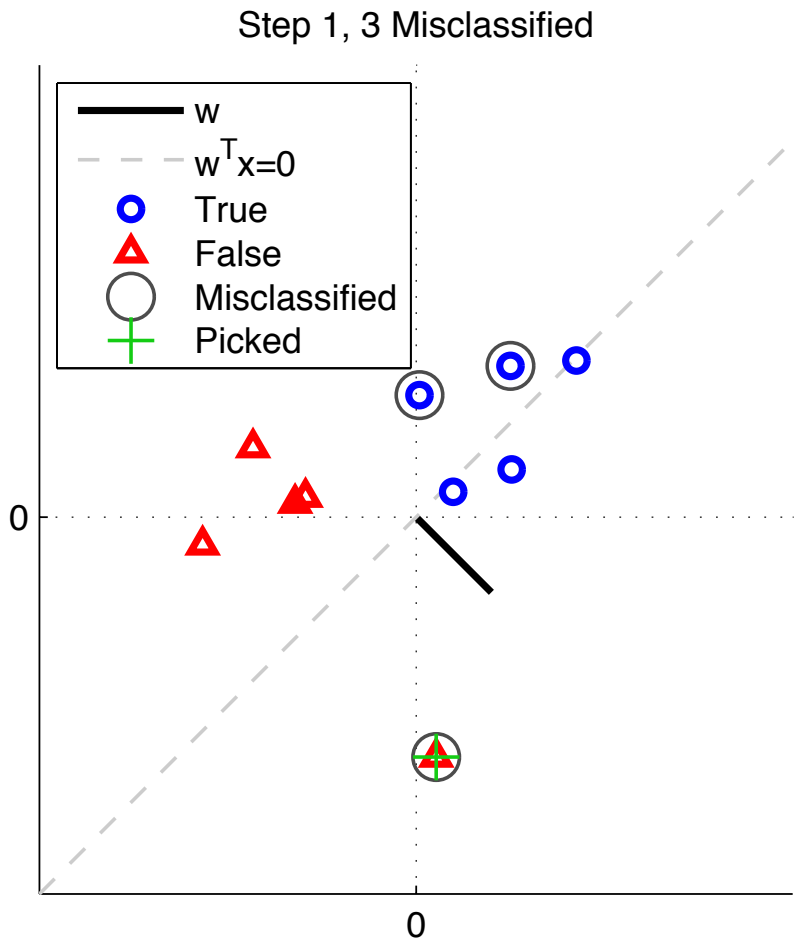
Algorithm 1 Perceptron learning

Require: Data $X = [\mathbf{x}_1, \ldots, \mathbf{x}_N], x_i \in \mathbb{R}^D$, Labels $y_1, \ldots, y_N \in \{-1, +1\}$, iterations *its*, learning rate η

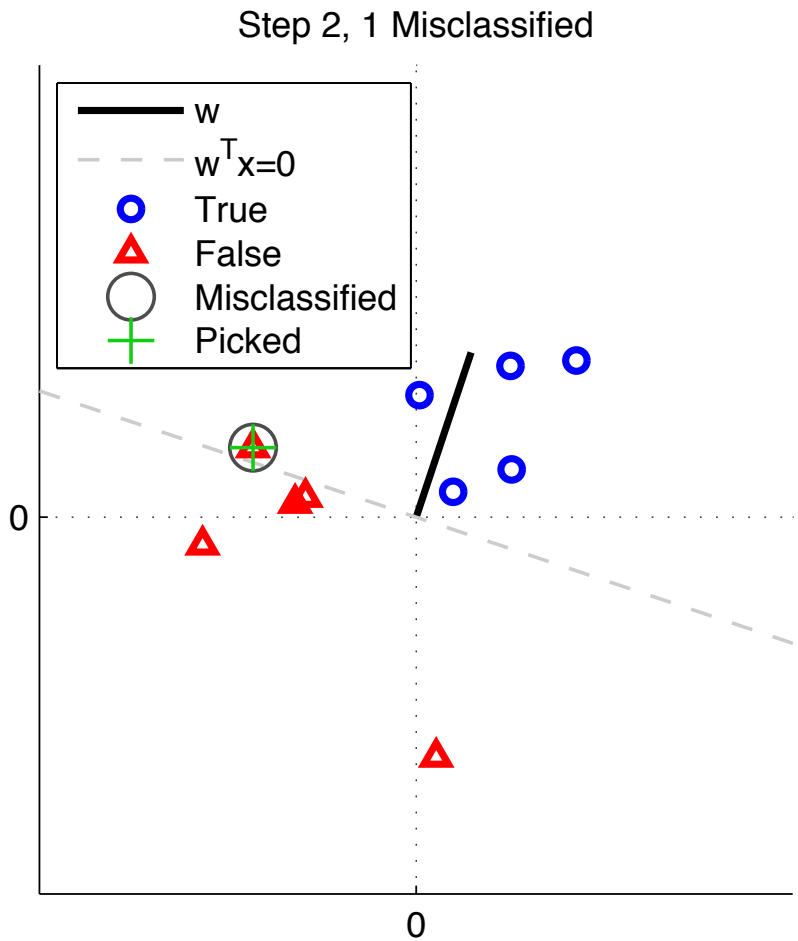
Ensure: $\mathbf{w}$

$\mathbf{w} = \mathbb{1}_D/(D)$
for $it = 1, \ldots,$ *its* **do**
    # Pick a random example
    $i =$ ceil(rand*N)
    # If this example is not correctly classified
    if $sign(\mathbf{w}^\top \mathbf{x}_i) \; != y_i$ **then**
        # Update weight vector
        $\mathbf{w} \leftarrow \mathbf{w} + \eta/it \; \mathbf{x}_i y_i$
    end if
end for

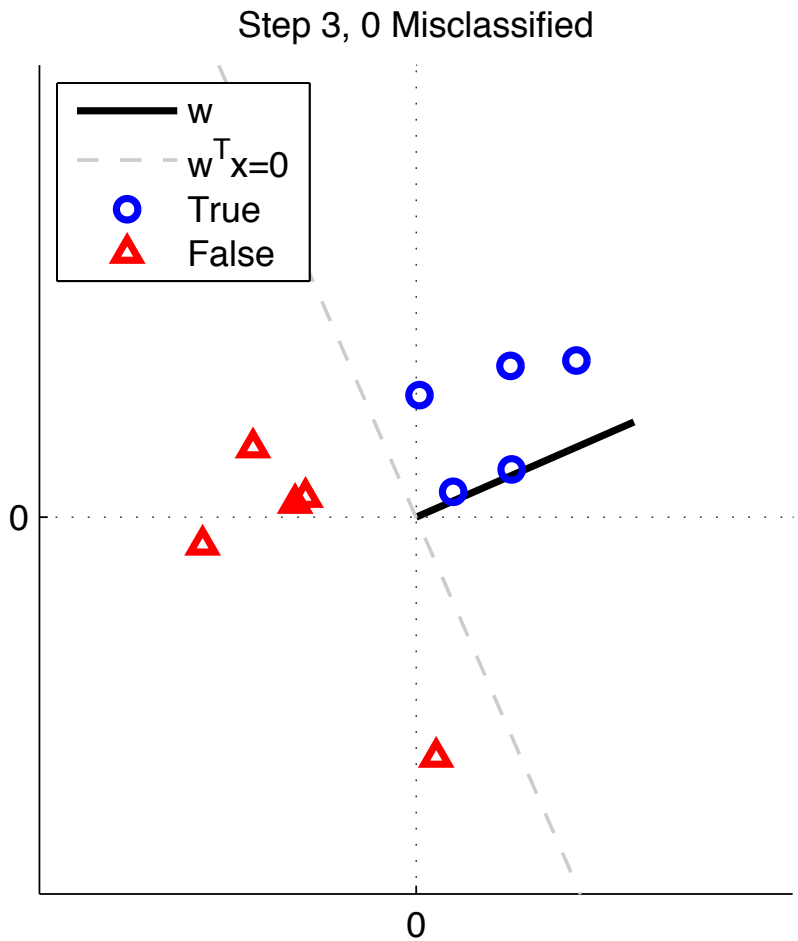
The Perceptron Learning Algorithm in Action



The Perceptron Learning Algorithm in Action



The Perceptron Learning Algorithm in Action



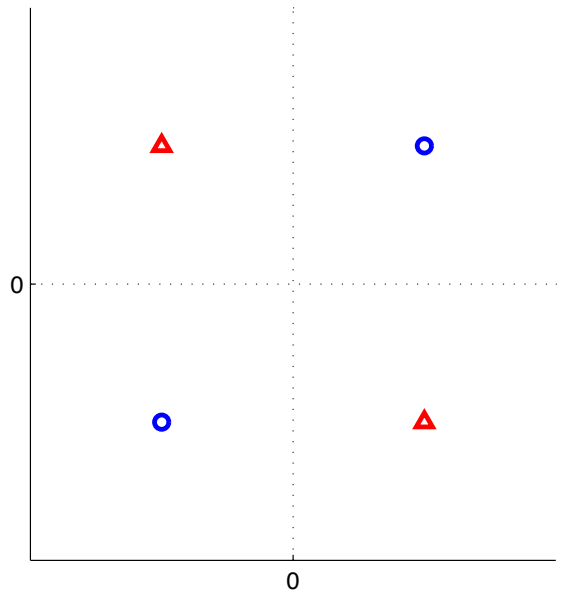
Problems with Perceptrons

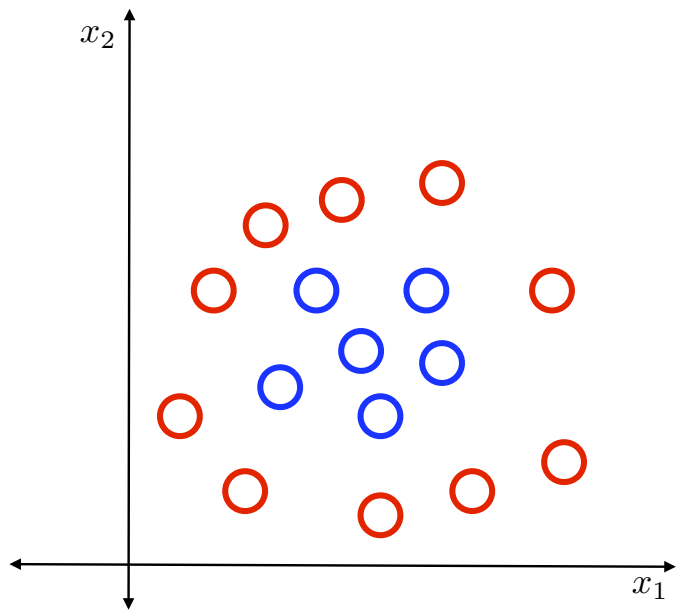
- Each update might lead to new misclassifications
- Many solutions might exist; which one is correct?
- Convergence might be slow
- Only solves linearly separable problems
- If there is no solution, the algorithm will not converge

Some solutions have been proposed by Freund and Schapire [1998]

Problems with Perceptrons

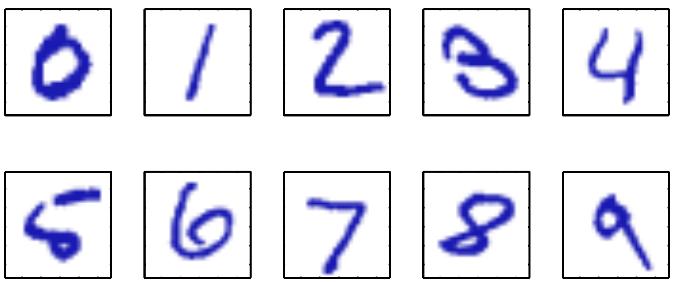
Perceptrons can only learn linearly separable problems.





Application example: Handwritten Digit Recognition

Handwritten digits from UPS data set



Each digit represented as 16×16 pixel image

$\rightarrow$ $x \in \mathbb{R}^{256}$ input nodes

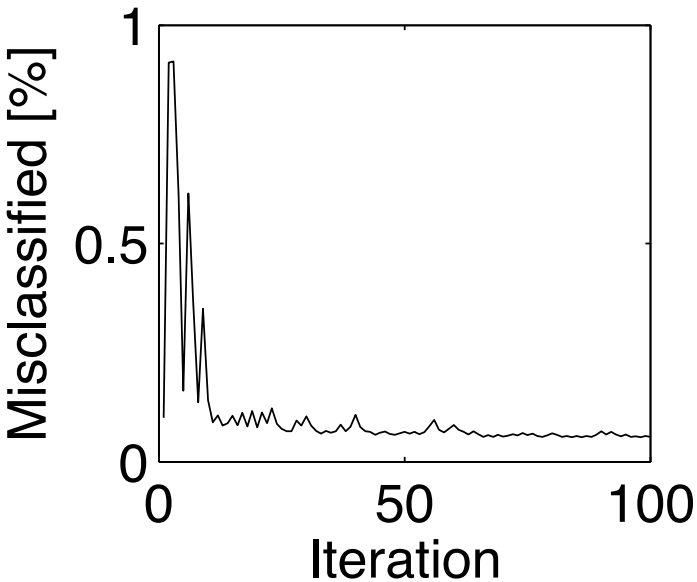
Each image is associated with a label $y \in \{0, 1, \ldots, 9\}$

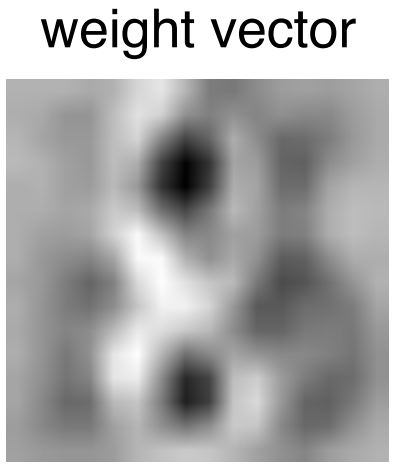
Goal Artificial neural network that recognizes the digit 8

$\Rightarrow$ We need a function $f(.)$ such that

$$f(x) = \begin{cases} -1 & \text{if } y \in \{0, 1, \ldots, 7, 9\} \\ +1 & \text{if } y = 8 \end{cases}$$

Application example: Handwritten Digit Recognition





Application example: Handwritten Digit Recognition

- Handwritten Digit Recognition is a **Multiclass Problem**
- But the Perceptron is a **two-class** (binary) classifier
- How can we **binarize** the multi class problem?

$\rightarrow$ Train a perceptron for each digit against all others

$\rightarrow$ Concatenate all weight vectors $W = [\mathbf{w_0}, \mathbf{w}_1, \ldots, \mathbf{w}_9]$

$\rightarrow$ For new data point $\mathbf{x}$ compute the label as $\operatorname{argmax}\left(W^\top \mathbf{x}\right)$

Minimizing Functions

Remember:

- ML algorithms are functions f that need to be fitted to data
- ML is finding minima/maxima of functions
- Function minimization is done by **Gradient Descent**

Taking Derivatives of Univariate Functions

Function	Derivative
$f(x) = x^n$	
$cf(x)$	
$f(x) + g(x)$	
$f(x)g(x)$	
$\frac{f(x)}{g(x)}$	
$f(g(x))$	

References

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer US, 2007.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, 2010. Springer.

Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, pages 277–296, 1998.

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400—407, 1951.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, Nov. 1958.

B. Widrow and M. E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104, New York, 1960. IRE.

Machine Learning

Lecture 8
Regression

Felix Bießmann

Beuth University & Einstein Center for Digital Future

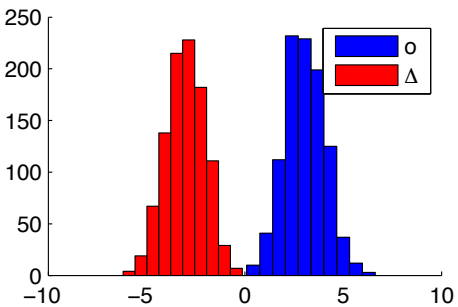
June 11, 2019

Signal-to-Noise Ratio in Classification Settings

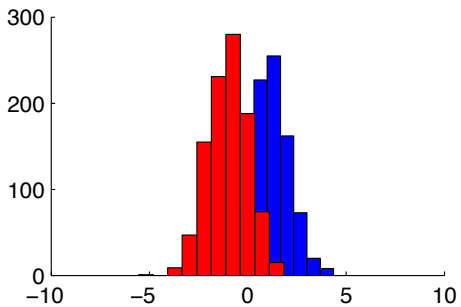
A useful definition of Signal-to-Noise Ratio for classification is

$$\frac{\text{Between Class Variance}}{\text{Within Class Variance}} \tag{1}$$

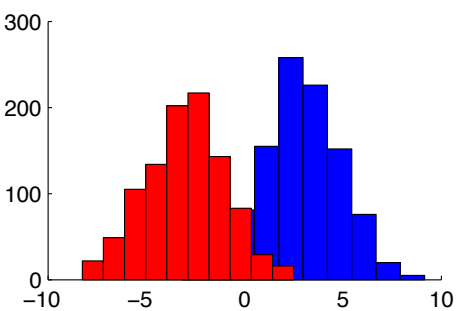
Good SNR



Bad SNR: Close means



Bad SNR: Large Variance

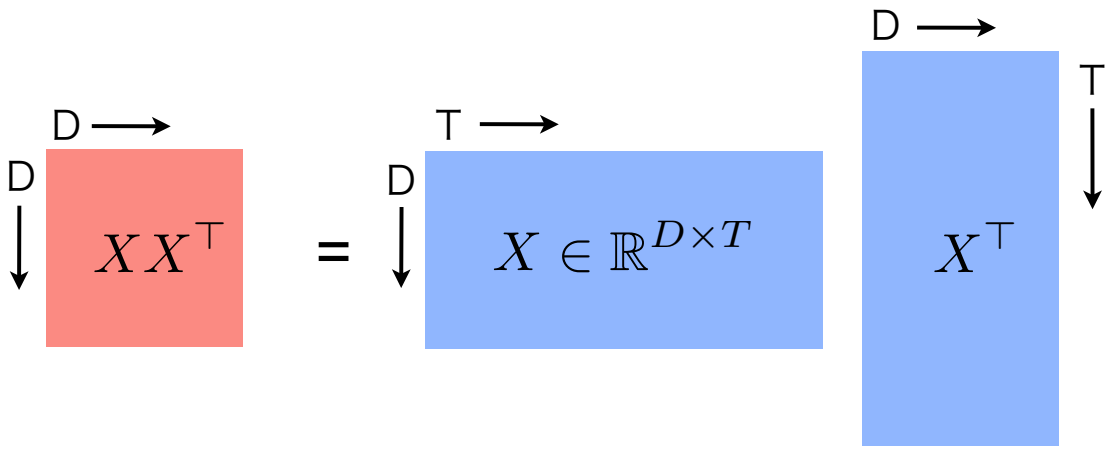


Covariance Matrices

Given T data points $\mathbf{x} \in \mathbb{R}^D$ in a data matrix $\mathbf{X} \in \mathbb{R}^{D \times T}$ the empirical estimate of the **covariance matrix** is defined as

$$1/T \ \mathbf{X}\mathbf{X}^\top \tag{2}$$

where we assume centered data, i.e. $\sum_{t=1}^{T} \mathbf{x}_t = 0$.

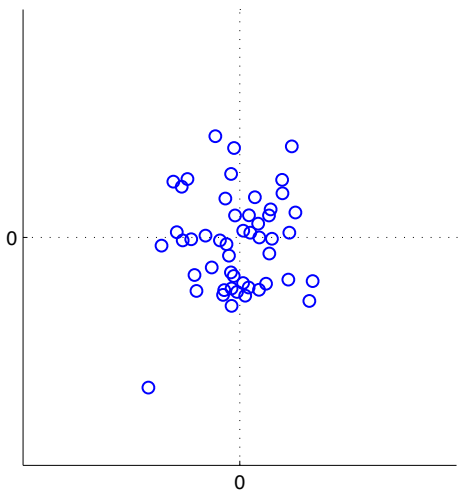


Correlated Data and Linear Mappings

Simulating correlated data can help understanding it

We generate uncorrelated data $\mathbf{x} \in \mathbb{R}^2$ drawn from a normal distribution $\mathbf{x} \sim \mathcal{N}(0, 1)$
We induce *correlations* by a diagonal scaling matrix D and a rotation matrix R

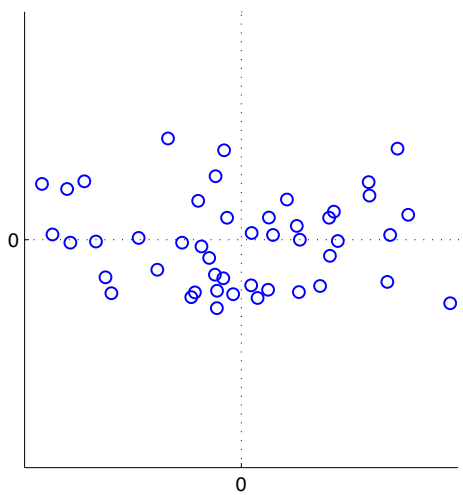
Uncorrelated



$$\mathbf{x} \sim \mathcal{N}(0, 1)$$

$$\mathbf{x}\mathbf{x}^\top = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

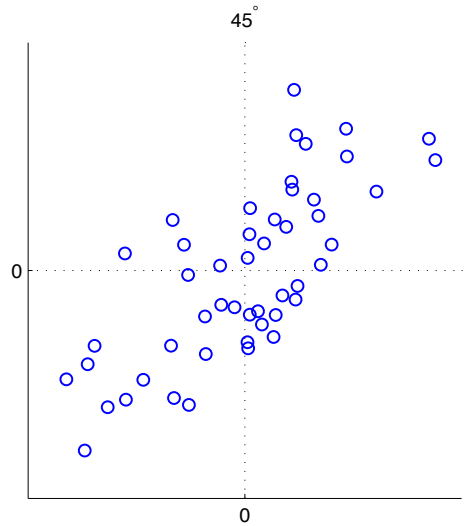
Uncorrelated, scaled



$$\begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x}$$

$$\mathbf{x}\mathbf{x}^\top = \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix}$$

Scaled, rotated by 45°



$$\begin{bmatrix} cos(\phi) & -sin(\phi) \\ sin(\phi) & cos(\phi) \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \mathbf{x}$$

$$\mathbf{x}\mathbf{x}^\top = \begin{bmatrix} 5 & 4 \\ 4 & 5 \end{bmatrix}$$

Fisher's Linear Discriminant Analysis

$$\underset{w}{\operatorname{argmax}} \frac{\mathbf{w}^\top \mathbf{S}_B \mathbf{w}}{\mathbf{w}^\top \mathbf{S}_W \mathbf{w}} \tag{3}$$

$$\mathbf{S}_B = \underbrace{(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)}_{\text{Distance Class Means}} (\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)^\top \tag{4}$$

$$\mathbf{S}_W = \sum_{i \in \mathcal{Y}_{+1}} (\mathbf{x}_i - \boldsymbol{\mu}_+)(\mathbf{x}_i - \boldsymbol{\mu}_+)^\top + \sum_{j \in \mathcal{Y}_{-1}} (\mathbf{x}_j - \boldsymbol{\mu}_-) \underbrace{(\mathbf{x}_j - \boldsymbol{\mu}_-)^\top}_{\text{Distance from Class Mean}} \tag{5}$$

Fisher's Linear Discriminant Analysis

Setting the first derivative of eq. 3 to zero, we obtain the generalized eigenvalue equation

$$\mathbf{S}_B w = \mathbf{S}_W w \lambda \tag{6}$$

Left multiplying with $\mathbf{S}_W^{-1}$ yields

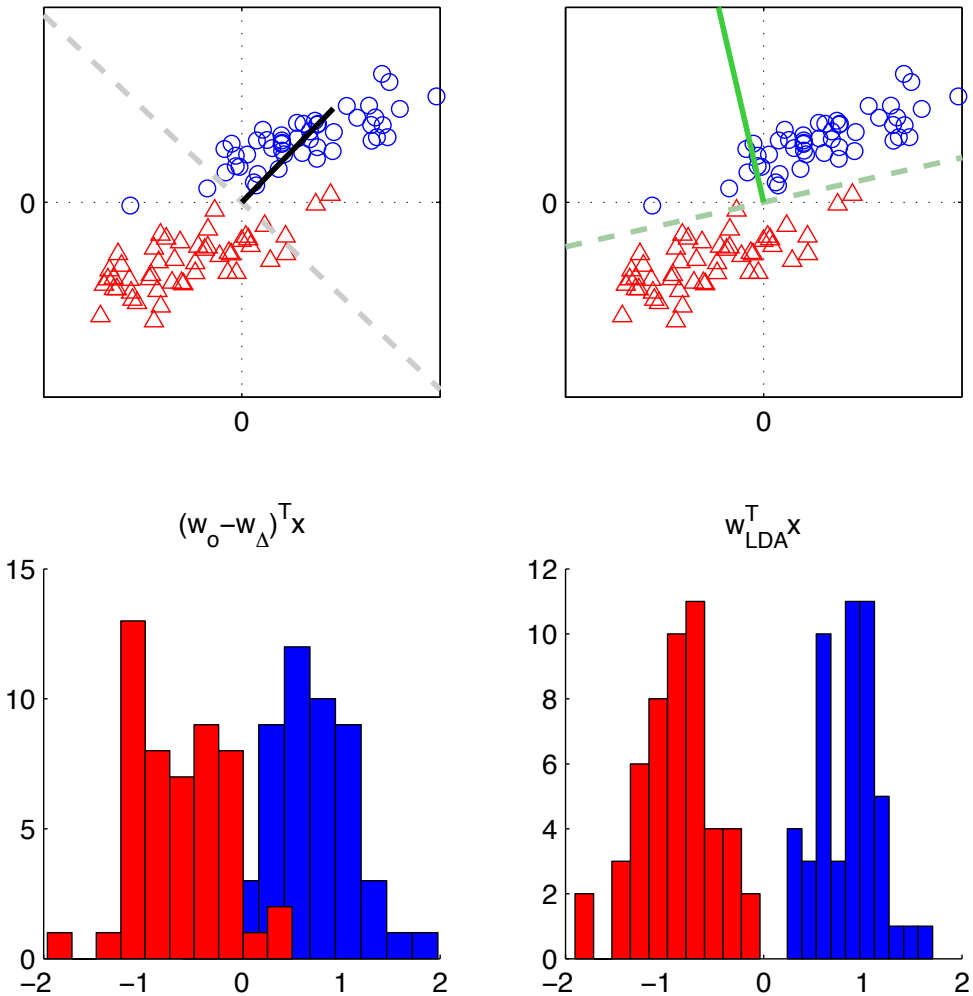
$$\mathbf{S}_W^{-1}\mathbf{S}_B w = \mathbf{S}_W^{-1}\mathbf{S}_W w \lambda$$

$$\mathbf{S}_W^{-1}(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)\underbrace{(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)^\top w}_{\beta} = w \tag{7}$$

$$w \propto \mathbf{S}_W^{-1}(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)$$

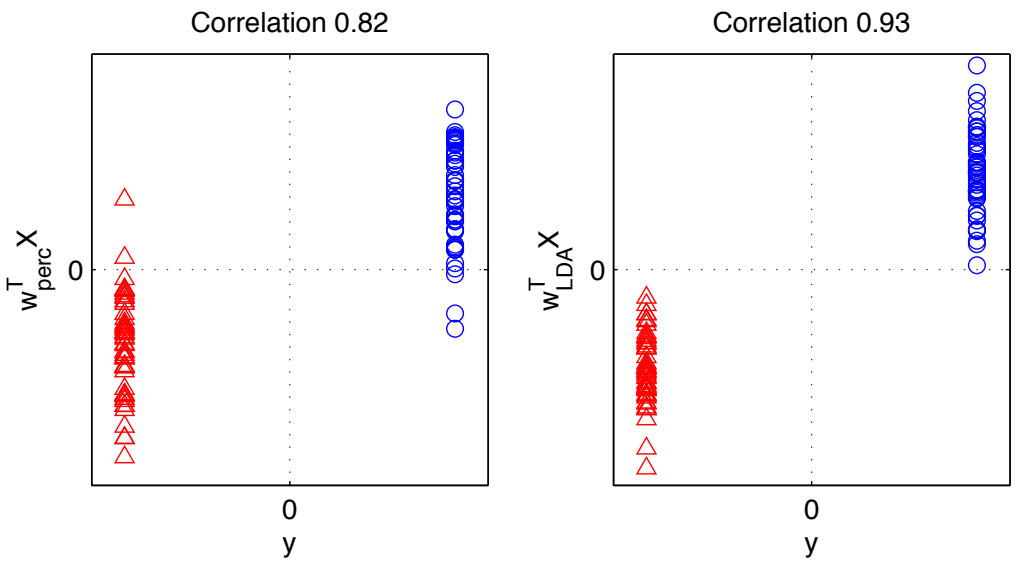
$\rightarrow$ Fisher's LDA first *decorrelates* the data followed by nearest centroid classification

Fisher Linear Discriminant Analysis



Fisher Linear Discriminant Analysis

Another view on LDA:
Maximizing the correlation between class labels $\mathbf{y}$ and $\mathbf{w}^\top \mathbf{X}$:



What if our labels are not $\in \{-1, +1\}$ but $\in \mathbb{R}$?

From Classification to Regression

$y \in \{-1, +1\}$	$y \in \mathbb{R}$
Classification	**Regression**

The most popular and best understood type of regression is **linear regression** using a *least-squares cost function*.

Linear Regression

Target variable $y \in \mathbb{R}$ is modeled as a **linear combination** $w \in \mathbb{R}^N$ of N regressors $\phi(\mathbf{x}) \in \mathbb{R}^N$

$$y = \mathbf{w}^\top \phi(\mathbf{x}) \tag{8}$$

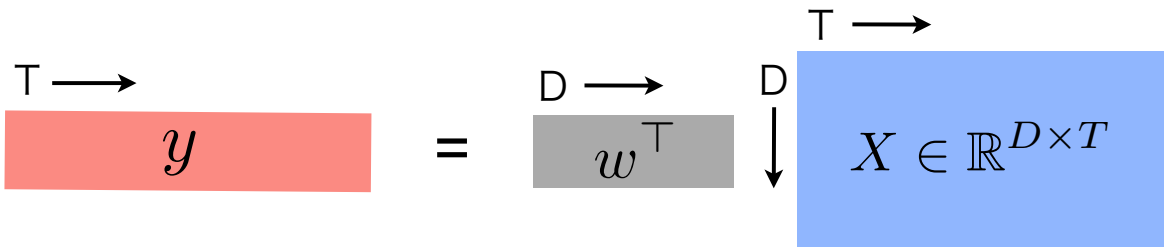
where $\phi(.)$ is one or more (potentially non-linear) function on $\mathbf{x}$.

For the sake of simplicity we assume $\phi(\mathbf{x}) = \mathbf{x}$.

Linear Regression

Let T be the number of samples, so $\mathbf{y} \in \mathbb{R}^{1\times T}$ and $\mathbf{X} \in \mathbb{R}^{N\times T}$. The Linear Regression model in matrix notation then becomes

$$\mathbf{y} = \mathbf{w}^\top \mathbf{X}. \tag{9}$$



Linear Regression

The most popular loss function to optimize w
is the **least-square error** [Gauß, 1809; Legendre, 1805]

$$\mathcal{E}_{lsq}(w) = \sum_{t=1}^{T} (y_t - \mathbf{w}^\top \mathbf{X}_t)^2 \tag{10}$$

C.F. Gauß (1777-1855)

A.M. Legendre (1752-1833)

Linear Regression

To minimize the least-squares loss function in eq. 10

$$\mathcal{E}_{lsq}(w) = (\mathbf{y} - \mathbf{w}^\top \mathbf{X})^2 = \mathbf{y}\mathbf{y}^\top - 2\mathbf{w}^\top \mathbf{X}\mathbf{y}^\top + \mathbf{w}^\top \mathbf{X}\mathbf{X}^\top \mathbf{w}$$

we compute derivative w.r.t. $\mathbf{w}$

Linear Regression

To minimize the least-squares loss function in eq. 10

$$\mathcal{E}_{lsq}(w) = (\mathbf{y} - \mathbf{w}^\top \mathbf{X})^2 = \mathbf{y}\mathbf{y}^\top - 2\mathbf{w}^\top \mathbf{X}\mathbf{y}^\top + \mathbf{w}^\top \mathbf{X}\mathbf{X}^\top \mathbf{w}$$

we compute derivative w.r.t. $\mathbf{w}$

$$\frac{\partial \mathcal{E}_{lsq}(\mathbf{w})}{\partial \mathbf{w}} = -2\mathbf{X}\mathbf{y}^\top + 2\mathbf{X}\mathbf{X}^\top \mathbf{w} \tag{11}$$

Linear Regression

To minimize the least-squares loss function in eq. 10

$$\mathcal{E}_{lsq}(w) = (\mathbf{y} - \mathbf{w}^\top \mathbf{X})^2 = \mathbf{y}\mathbf{y}^\top - 2\mathbf{w}^\top \mathbf{X}\mathbf{y}^\top + \mathbf{w}^\top \mathbf{X}\mathbf{X}^\top \mathbf{w}$$

we compute derivative w.r.t. $\mathbf{w}$

$$\frac{\partial \mathcal{E}_{lsq}(\mathbf{w})}{\partial \mathbf{w}} = -2\mathbf{X}\mathbf{y}^\top + 2\mathbf{X}\mathbf{X}^\top \mathbf{w} \tag{11}$$

set it to zero and solve for w

$$\begin{aligned} -2\mathbf{X}\mathbf{y}^\top + 2\mathbf{X}\mathbf{X}^\top \mathbf{w} &= 0 \\ \mathbf{X}\mathbf{X}^\top \mathbf{w} &= \mathbf{X}\mathbf{y}^\top \\ \mathbf{w} &= (\underbrace{\mathbf{X}\mathbf{X}^\top}_{\text{Cov. Mat.}})^{-1}\mathbf{X}\mathbf{y}^\top \end{aligned} \tag{12}$$

Linear Regression for Vector Labels

Prediction of vector-valued labels $\mathbf{y} \in \mathbb{R}^M$
is called **Multiple Linear Regression**:

For a measurement $\mathbf{X} \in \mathbb{R}^{N \times T}$, $\mathbf{Y} \in \mathbb{R}^{M \times T}$ the MLR model is

$$\mathbf{Y} = \mathbf{W}^\top \mathbf{X} \tag{13}$$

where $\mathbf{W}^\top \in \mathbb{R}^{M \times N}$ is a **linear mapping** from data to labels.

Linear Regression for Vector Labels

Given Data $\mathbf{X} \in \mathbb{R}^{N \times T}$ and labels $\mathbf{Y} \in \mathbb{R}^{M \times T}$
the error function for multiple linear regression is

$$\mathcal{E}_{MLR}(\mathbf{W}) = \sum_{m=1}^{M} (\mathbf{Y}_m - \mathbf{W}_m^\top \mathbf{X})^2 \tag{14}$$

where $\mathbf{Y}_m$ denotes the m-th output dimension
and $\mathbf{W}_m$ the corresponding weight vector

Linear Regression for Vector Labels

Given Data $\mathbf{X} \in \mathbb{R}^{N \times T}$ and labels $\mathbf{Y} \in \mathbb{R}^{M \times T}$
the error function for multiple linear regression is

$$\mathcal{E}_{MLR}(\mathbf{W}) = \sum_{m=1}^{M} (\mathbf{Y}_m - \mathbf{W}_m^\top \mathbf{X})^2 \tag{14}$$

where $\mathbf{Y}_m$ denotes the m-th output dimension
and $\mathbf{W}_m$ the corresponding weight vector

Eq. 14 is minimized by

$$W = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{Y}^\top \tag{15}$$

Linear Discriminant Analysis and Linear Regression

Remember the solution to linear discriminant analysis

$$\mathbf{w}_{LDA} \propto \mathbf{S}^{-1}(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-)$$

Linear Discriminant Analysis and Linear Regression

Remember the solution to linear discriminant analysis

$$\begin{aligned}\mathbf{w}_{LDA} \propto & \mathbf{S}^{-1}(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-) \\ \propto & \mathbf{S}^{-1}(\underbrace{(+1)}_{y} \underbrace{(1/N_{+1})}_{\gamma_1} \sum_{i \in \mathcal{Y}_{+1}} \mathbf{x}_i + \underbrace{(-1)}_{y} \underbrace{(1/N_{-1})}_{\gamma_2} \sum_{j \in \mathcal{Y}_{-1}} \mathbf{x}_j)\end{aligned}$$

Linear Discriminant Analysis and Linear Regression

Remember the solution to linear discriminant analysis

$$
\begin{aligned}
\mathbf{w}_{LDA} \propto & \mathbf{S}^{-1}(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-) \\
\propto & \mathbf{S}^{-1}(\underbrace{(+1)}_{y}\ \underbrace{(1/N_{+1})}_{\gamma_1} \sum_{i \in \mathcal{Y}_{+1}} \mathbf{x}_i + \underbrace{(-1)}_{y}\ \underbrace{(1/N_{-1})}_{\gamma_2} \sum_{j \in \mathcal{Y}_{-1}} \mathbf{x}_j) \\
\propto & \mathbf{S}^{-1}\mathbf{X}\mathbf{y}^\top \text{ assuming } N_{+1} = N_{-1}
\end{aligned}
$$

Linear Discriminant Analysis and Linear Regression

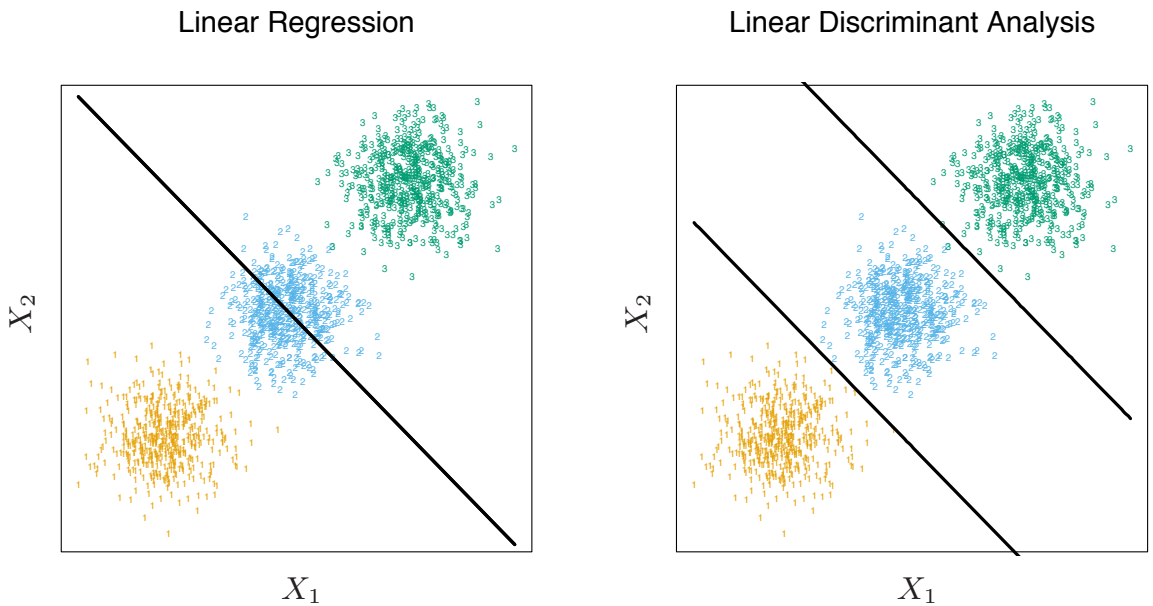
Remember the solution to linear discriminant analysis

$$
\begin{aligned}
\mathbf{w}_{LDA} \propto & \mathbf{S}^{-1}(\boldsymbol{\mu}_+ - \boldsymbol{\mu}_-) \\
\propto & \mathbf{S}^{-1}(\underbrace{(+1)}_{y}\ \underbrace{(1/N_{+1})}_{\gamma_1} \sum_{i \in \mathcal{Y}_{+1}} \mathbf{x}_i + \underbrace{(-1)}_{y}\ \underbrace{(1/N_{-1})}_{\gamma_2} \sum_{j \in \mathcal{Y}_{-1}} \mathbf{x}_j) \\
\propto & \mathbf{S}^{-1}\mathbf{X}\mathbf{y}^\top \text{ assuming } N_{+1} = N_{-1}
\end{aligned}
$$

LDA	Linear Regression
$\mathbf{w} \propto \mathbf{S}^{-1}\mathbf{X}\mathbf{y}^\top$	$\mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y}^\top$

Classification by Linear Regression?

So when coding the class as a boolean multivariate label vector, can we do classification with linear regression?



No, for more than 2 classes, this can lead to poor classification

References

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer US, 2007.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, 2010. Springer.

C. F. Gauß. Theoria motus corporum coelestium in sectionibus conicis solem ambientium. *Göttingen*, 1809.

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. 2003.

A. E. Hoerl and R. W. Kennar. Ridge regression: Applications to nonorthogonal problems. *Technometrics*, 12(1):69–82, 1970.

A.-M. Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*, chapter Sur la methode des moindres quarres. Firmin Didot, http://imgbase-scd-ulp.u-strasbg.fr/displayimage.php?pos=-141297, 1805.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning. The MIT Press, 1 edition, 2012. ISBN 0262018020,9780262018029.

A. Rahimi and B. Recht. Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning. In D. Koller, D. Schuurmans, Y. Bengio, L. Bottou, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *NIPS*, pages 1313–1320. MIT Press, 2008.

R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1996.

A. N. Tychonoff. On the stability of inverse problems. *Doklady Akademii Nauk SSSR*, 39(5):195–198, 1943.

Machine Learning

Lecture 9
Principal Component Analysis and Extensions

Felix Bießmann

Beuth University & Einstein Center for Digital Future

June 10, 2019

Supervised vs Unsupervised Algorithms

Often there is no label information available

- Ongoing neural activity
- Mixtures of different speakers in a audio recording
- Complex artefacts in experimental recordings

Unsupervised algorithms

- Find structure in data sets
- Allow partitioning of data in *meaningful* parts
- Allow to remove unwanted aspects (e.g. noise)

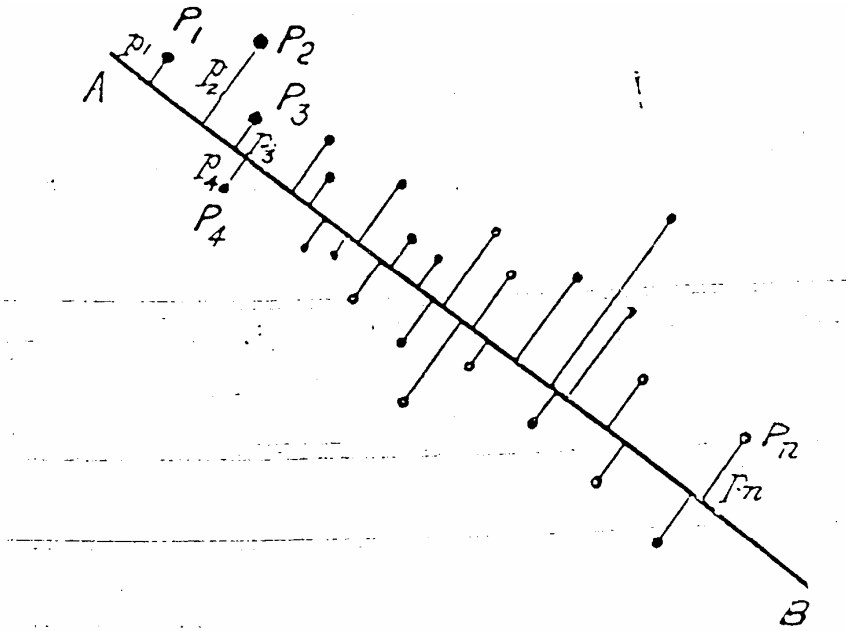
Principal Component Analysis

Principal Component Analysis (PCA):

Popular dimensionality reduction technique

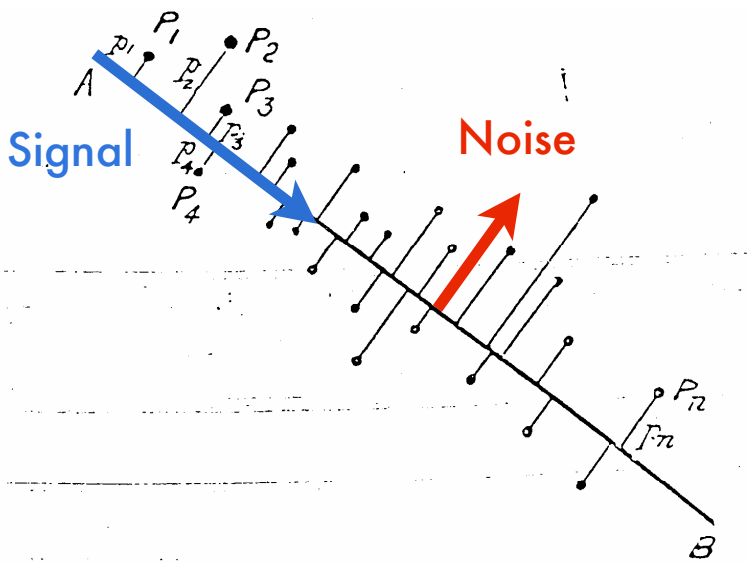
Easy to implement

Principal Component Analysis



Which line fits data best?

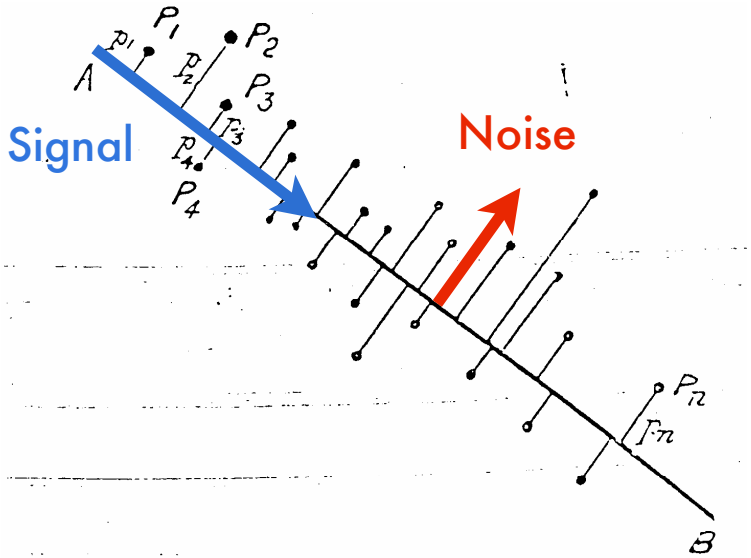
Principal Component Analysis



Which line fits data best?

The line w that minimizes the noise and maximizes the signal [Pearson, 1901]

Principal Component Analysis



Which line fits data best?

The line w that minimizes the noise and maximizes the signal [Pearson, 1901]

Or equivalently:

The line w that maximizes the variance within the data set

Maximizing variance in a data set

We obtained some data $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$

PCA finds a direction $\mathbf{w}^* \in \mathbb{R}^D$ such that

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}}\, \mathbf{w}^\top \mathbf{X}\mathbf{X}^\top \mathbf{w} \tag{1}$$

Maximizing variance in a data set

We obtained some data $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N] \in \mathbb{R}^{D \times N}$

PCA finds a direction $\mathbf{w}^* \in \mathbb{R}^D$ such that

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmax}} \, \mathbf{w}^\top \mathbf{X}\mathbf{X}^\top \mathbf{w} \tag{1}$$

When optimizing eq. 1 we have to constrain $\mathbf{w}$

$$\|\mathbf{w}\|^2 = \mathbf{w}^\top \mathbf{w} = 1 \tag{2}$$

yielding the Lagrangian

$$\mathcal{L} = \mathbf{w}^\top \mathbf{X}\mathbf{X}^\top \mathbf{w} + \lambda(1 - \mathbf{w}^\top \mathbf{w}) \tag{3}$$

Short excursion: Lagrangians and Optimization

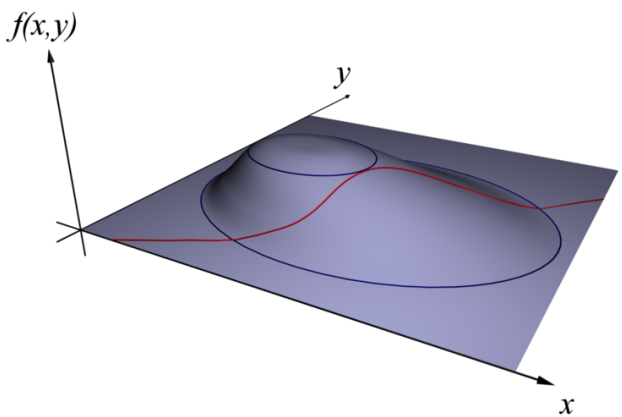
Optimizing a function subject to some constraint

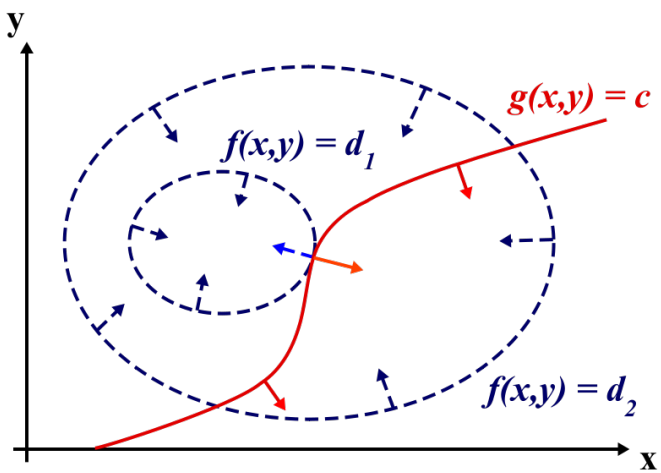
$$\text{maximize } f(x, y)$$

$$\text{subject to the constraint } g(x, y) = c$$

$$\text{Lagrangian: } \mathcal{L}(x, y, \lambda) = f(x, y) + \lambda(g(x, y) - c)$$

where λ is called a *Lagrangian Multiplier*





Source: `http://en.wikipedia.org/wiki/Lagrange_multipliers`

Maximizing variance in a data set

$$\mathcal{L} = \mathbf{w}^\top \mathbf{X}\mathbf{X}^\top \mathbf{w} + \lambda(1 - \mathbf{w}^\top \mathbf{w})$$

Setting the derivative w.r.t. w to zero yields

$$\frac{\partial \mathcal{L}}{\partial w} = 2\mathbf{X}\mathbf{X}^\top \mathbf{w} - 2\lambda \mathbf{w} = 0$$

$$\Rightarrow \mathbf{X}\mathbf{X}^\top \mathbf{w} = \lambda \mathbf{w} \tag{4}$$

This is a standard eigenvalue problem.

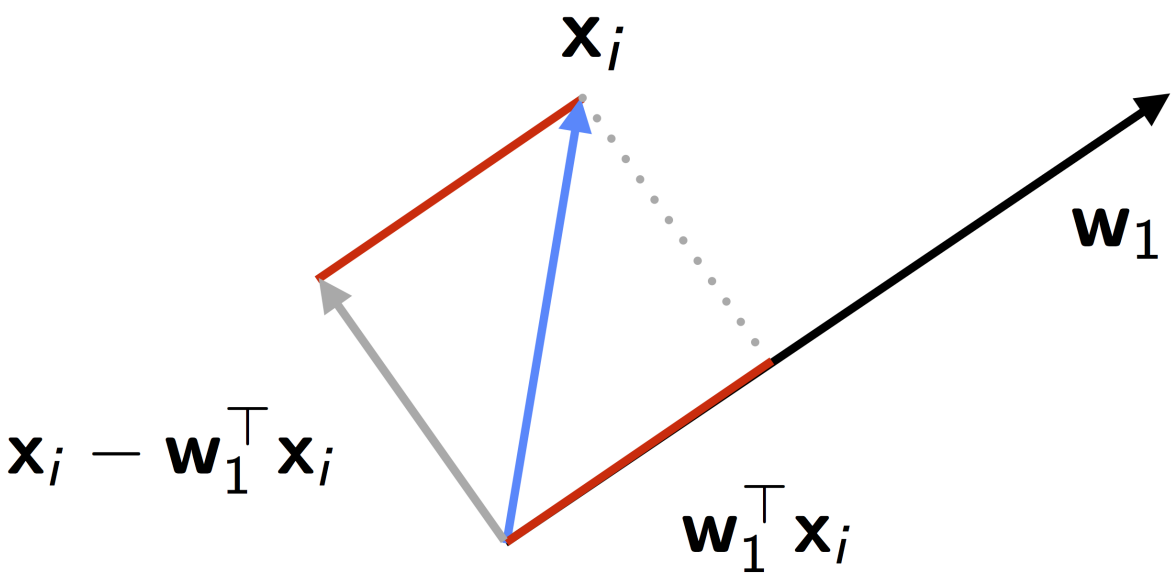
$\mathbf{w}$ is the eigenvector of $\mathbf{X}\mathbf{X}^\top$ corresponding to the largest eigenvalue

Finding k Principal Components

- We found the strongest variance direction
- Now we want to find the second strongest variance direction
- The second direction should not be correlated with the first
- We *project out* the variance explained by the first direction
- And again find the strongest variance direction

Finding k Principal Components

How can we *project out* the variance along the first principal direction $\mathbf{w}_1$?



Finding k Principal Components

How can we *project out* the variance along the first principal direction $\mathbf{w}_1$?

Project data on $\mathbf{w}_1$ and back through $\mathbf{w}_1$

$$\mathbf{X}_{\mathbf{w}_1} = \mathbf{w}_1 \underbrace{\mathbf{w}_1^\top \mathbf{X}}_{\text{First Principal Component}} \tag{5}$$

Finding k Principal Components

How can we *project out* the variance
along the first principal direction $\mathbf{w}_1$?

Project data on $\mathbf{w}_1$ and back through $\mathbf{w}_1$

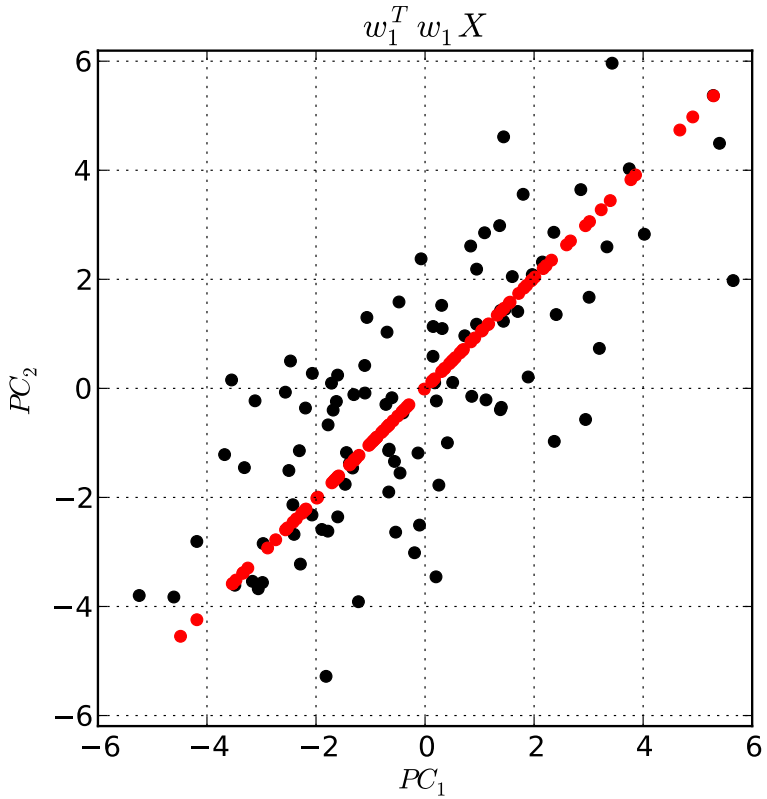
$$\mathbf{X}_{\mathbf{w}_1} = \mathbf{w}_1 \underbrace{\mathbf{w}_1^\top \mathbf{X}}_{\text{First Principal Component}} \tag{5}$$

Now we can subtract $\mathbf{X}_{\mathbf{w}_1}$ from the original data $\mathbf{X}$.

$$\mathbf{X} - \mathbf{X}_{\mathbf{w}_1} = \mathbf{X} - \mathbf{w}_1 \mathbf{w}_1^\top \mathbf{X} = (\mathbf{I} - \mathbf{w}_1 \mathbf{w}_1^\top)\mathbf{X} \tag{6}$$

Finding k Principal Components

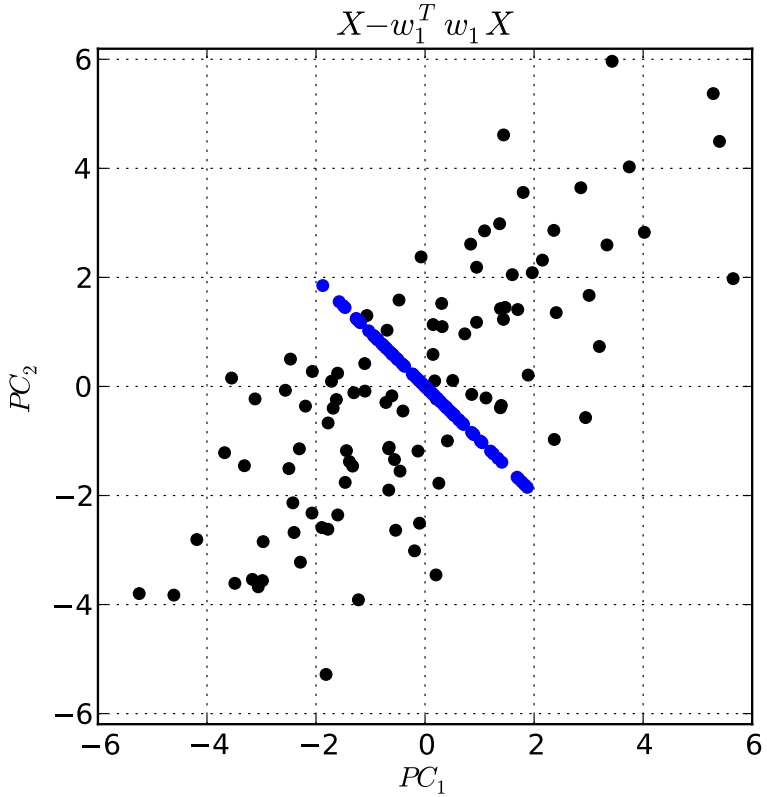
$$\mathbf{X}_{\mathbf{w}_1} = \mathbf{w}_1 \mathbf{w}_1^\top \mathbf{X}$$



Data projected on $\mathbf{w}_1$ and back through $\mathbf{w}_1$

Finding k Principal Components

$$\mathbf{X} - \mathbf{w}_1\mathbf{w}_1^\top\mathbf{X} = (\mathbf{I} - \mathbf{w}_1\mathbf{w}_1^\top)\mathbf{X}$$



Data projected on $\mathbf{w}_1$ and back through $\mathbf{w}_1$ subtracted from $\mathbf{X}$

Finding k Principal Components

Finding the largest eigenvector using gradient descent, projecting it out and finding the next eigenvector is called the **Power Method** for solving eigenvalue equations

Principal Directions are Eigenvectors of Covariance Matrix

The k first PCA basis vectors are the eigenvectors corresponding to the largest k eigenvalues

$$\mathbf{X}\mathbf{X}^\top \mathbf{W} = \mathbf{W}\Lambda \tag{7}$$

where $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k]$ contains the eigenvectors sorted according to their eigenvalues and Λ is a diagonal matrix containing all eigenvalues.

Principal Directions are Eigenvectors of Covariance Matrix

The k first PCA basis vectors are the eigenvectors corresponding to the largest k eigenvalues

$$\mathbf{X}\mathbf{X}^\top\mathbf{W} = \mathbf{W}\Lambda \tag{7}$$

where $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k]$ contains the eigenvectors sorted according to their eigenvalues and Λ is a diagonal matrix containing all eigenvalues.

A useful property of the new PCA basis:
Eigenvectors $\mathbf{w}_i,\ i \in \{1, 2, \ldots, k\}$ are orthogonal to each other:

$$\mathbf{w}_i^\top \mathbf{w}_j = 0, \forall i \neq j$$

PCA Algorithm

Algorithm 2 Principal Component Analysis

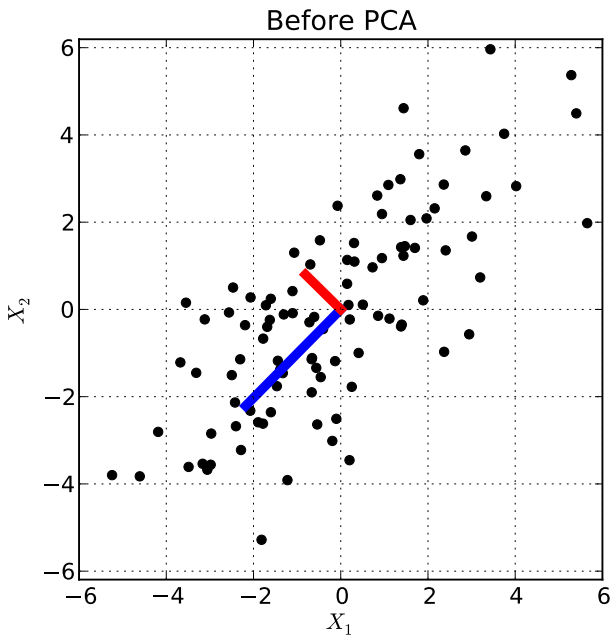
Require: data $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^d$, number of principal components k

Ensure: $\mathbf{W}$

1: # Center Data
2: $\mathbf{X} = \mathbf{X} - 1/N \sum_i \mathbf{x}_i$
3: # Compute Covariance Matrix
4: $\mathbf{C} = 1/N \, \mathbf{X}\mathbf{X}^\top$
5: # Compute largest k eigenvectors
6: $\mathbf{W} = \text{eig}(\mathbf{C})$

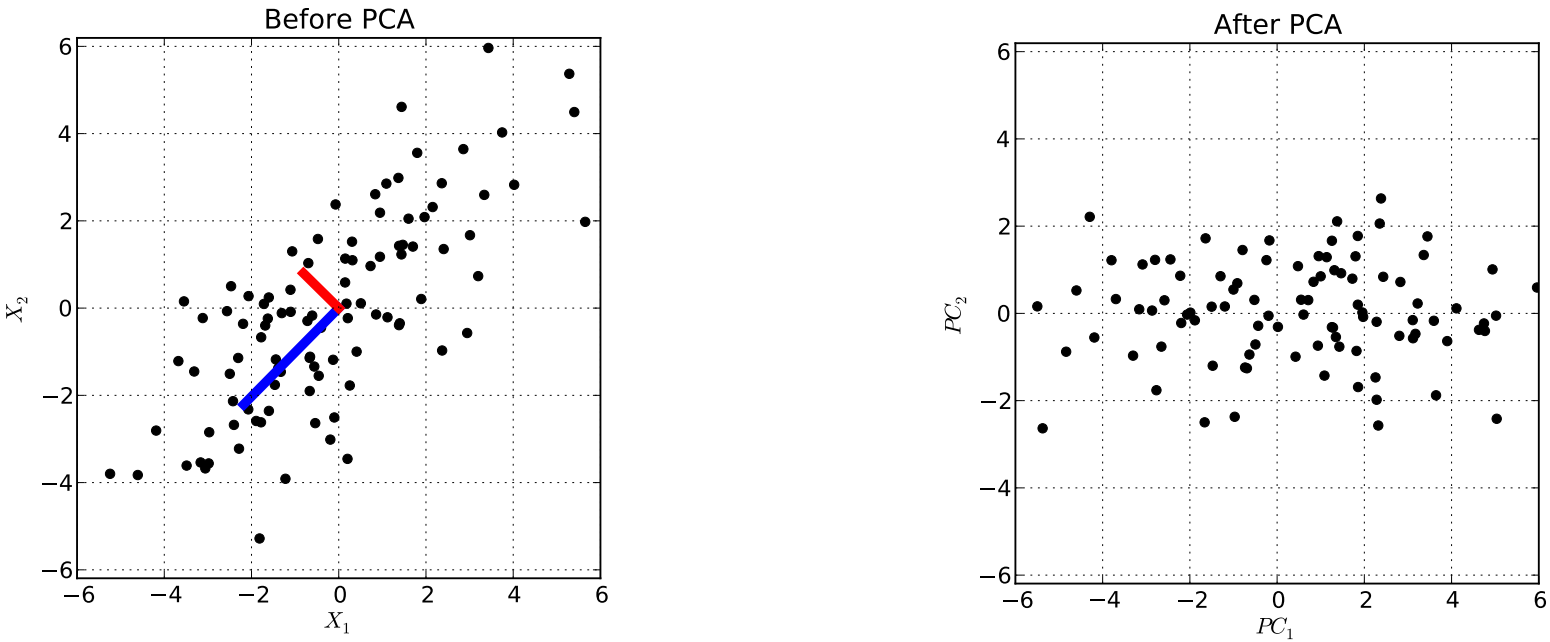
Principal Component Analysis

$\mathbf{X}$



Principal Component Analysis

$\mathbf{x}$ $\mathbf{w}^\top \mathbf{x}$



PCA aligns maximum variance directions with standard basis

$\rightarrow$ Variance along each dimension is **uncorrelated**

$\rightarrow$ Now we can remove each dimension separately

Dimensionality Reduction by PCA

We can reduce the dimensionality of $\mathbf{X}$ from d to k

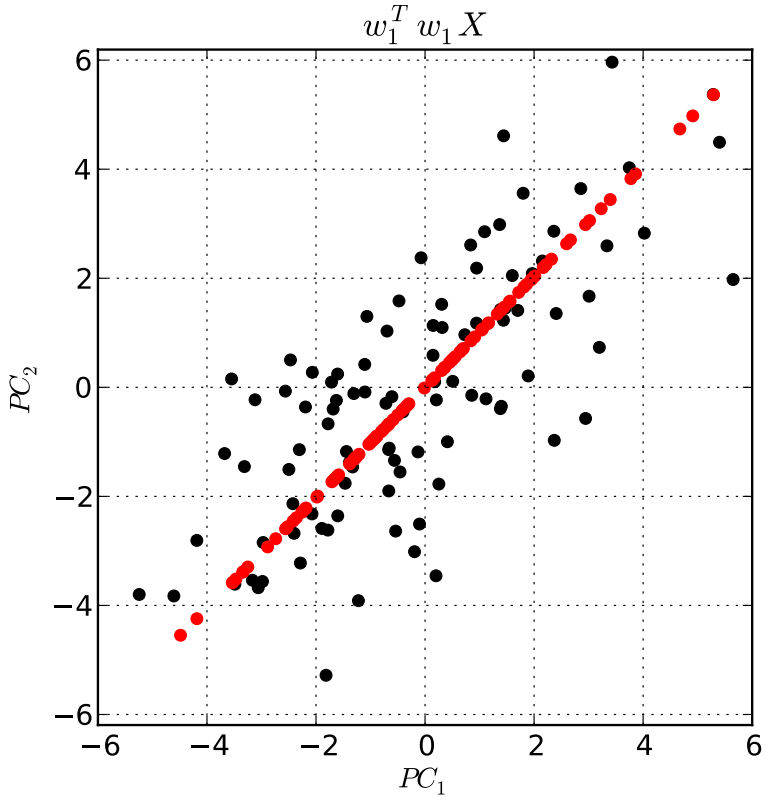
$$\mathbf{X}_{PCA} = \mathbf{W}^\top \mathbf{X} \tag{8}$$

If we want only a set $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ of principal components

$$\mathbf{X}_{\mathcal{I}} = \sum_{i \in \mathcal{I}} \mathbf{w}_i \mathbf{w}_i^\top \mathbf{X}$$

Note that $\mathcal{I}$ does not need to contain the *strongest* components

Dimensionality Reduction by PCA



Here we assume the relevant signal is along the high variance direction

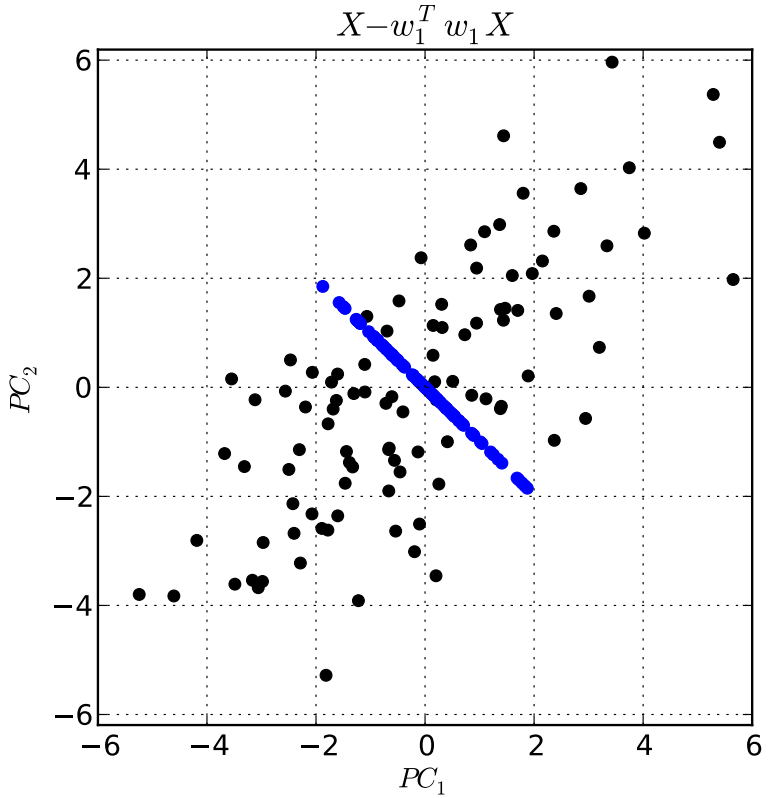
Denoising by PCA

Assuming that noise has high (or low) variance we can remove those components

If we want to project out a set $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ of principal components but we want the data to be in the input space

$$\mathbf{X}_{PCA} = \mathbf{X} - \mathbf{W}_{\mathcal{I}}\mathbf{W}_{\mathcal{I}}^{\top}\mathbf{X} = (\mathbf{I} - \mathbf{W}_{\mathcal{I}}\mathbf{W}_{\mathcal{I}}^{\top})\mathbf{X} \tag{8}$$

Denoising by PCA



Here we assume noise is along the high variance direction

Summary

Unsupervised Data Analysis

Finds structure in data in explorative fashion
Can be used for

Dimensionality reduction
Visualization
Denoising

Principal Component Analysis (PCA)

Finds directions of maximal variance
Is solved by eigendecomposition of Covariance/Kernel Matrix

Linear PCA

Finds *linear* subspaces
If there are more dimensions than data points
$\rightarrow$ Do eigendecomposition on kernel matrix

References

K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572, 1901.

B. Schölkopf, A. J. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(6):1299–1319, 1998.

Machine Learning

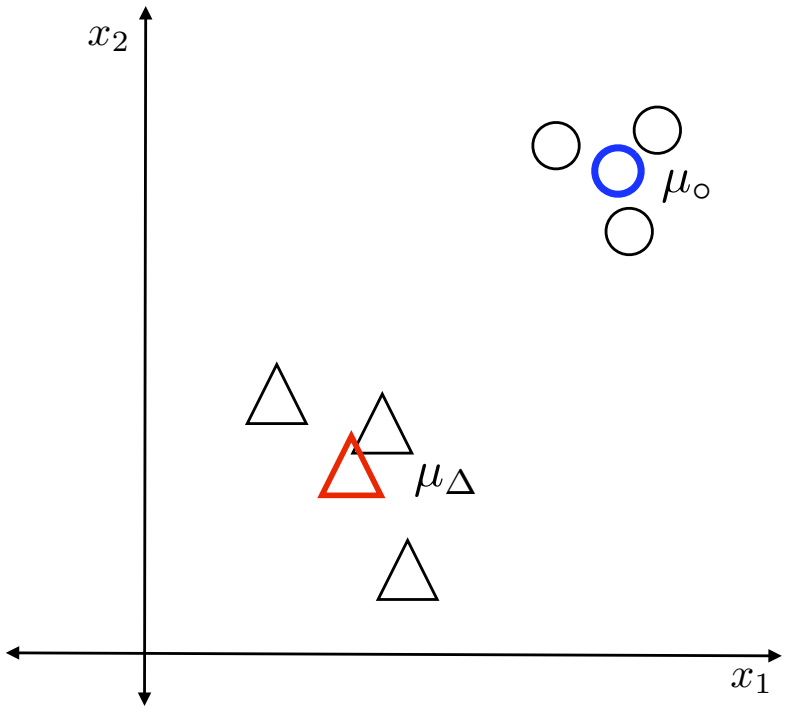
Lecture 10
Clustering

Felix Bießmann

Beuth University & Einstein Center for Digital Future

Clustering

Psychological Models of Categorization: Prototypes



Prototypes $\boldsymbol{\mu}_\Delta$ and $\boldsymbol{\mu}_o$:

$$\boldsymbol{\mu}_\Delta = 1/N_\Delta \sum_n^{N_\Delta} \mathbf{x}_{\Delta,n}$$

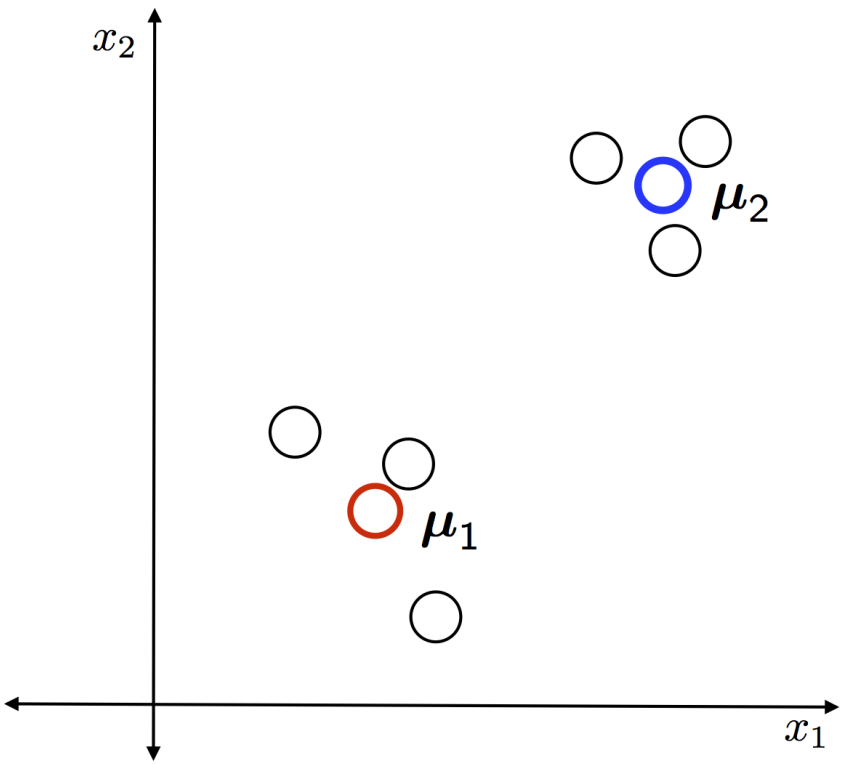
$$\boldsymbol{\mu}_o = 1/N_o \sum_n^{N_o} \mathbf{x}_{o,n}$$

New data points x are assigned to their closest cluster center μ^*

$$\boldsymbol{\mu}^* = \operatorname*{argmin}_i(\|\boldsymbol{\mu}_i - \mathbf{x}\|_2) \qquad (1)$$

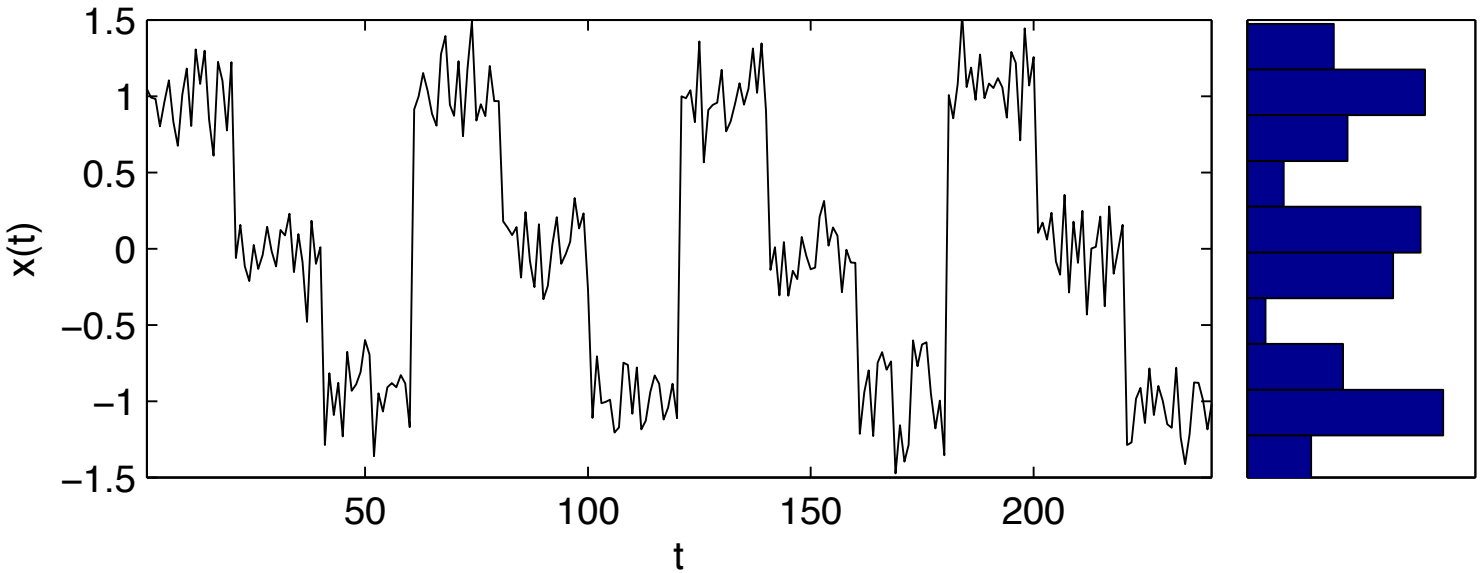
Clustering

Psychological Models of Categorization: Prototypes



The only difference for clustering is:
We do not have labels.

Clustering For Quantization of Analog Signals



Quantization transforms an analog signal into discretized states
This is important for Audio Processing, Compression, ...
The most popular Clustering Algorithm was proposed for Quantization [Lloyd, 1982]

K-means Clustering

K-Means Algorithm

Re-iterating two steps:

1. Assign each data point $\mathbf{x}_i$ to their closest cluster $\boldsymbol{\mu}_k$
2. Update $\boldsymbol{\mu}_k$ to the mean of the members in that cluster

K-means Clustering

Goal: Given data $\mathbf{x}_1, \ldots, \mathbf{x}_N$ find cluster centers $\boldsymbol{\mu}_1, \ldots \boldsymbol{\mu}_K$ such that the distances of data points to their respective cluster centre are minimized

$$\mathcal{J} = \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbf{c}_{n,k} \|\mathbf{x}_n - \boldsymbol{\mu}_{\mathbf{c}_k}\| \tag{2}$$

$$\text{where } \mathbf{c}_{n,k} \begin{cases} 1 & \text{if } \mathbf{x}_n \text{ belongs to cluster } k \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

K-means Clustering Algorithm

Algorithm 1 K-means clustering

Require: data $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^D$, number of clusters k, iterations m.

Choose random data points as initial cluster centres $\boldsymbol{\mu}_1 \leftarrow \mathbf{x}_{i_1}, \ldots, \boldsymbol{\mu}_k \leftarrow \mathbf{x}_{i_k}$ where $i_j \neq i_l$ for all $j \neq l$.
$\mathbf{c} \leftarrow \mathbf{0}_N$
$\mathbf{c}^{\text{old}} \leftarrow \mathbf{0}_N$
$i \leftarrow 0$
while $i < m$ **do**
    for $j = 1$ to N **do**
        Find nearest cluster centre $\mathbf{c}_j \leftarrow \text{argmin}_{1 \leq l \leq k} \|\mathbf{x}_j - \boldsymbol{\mu}_l\|_2$
    end for
    for $j \leftarrow 1$ to k **do**
        Compute new cluster centre $\boldsymbol{\mu}_j \leftarrow \frac{1}{|\{l : \mathbf{c}_l = j\}|} \sum_{l : \mathbf{c}_l = j} \mathbf{x}_l$
    end for
    if $\mathbf{c}^{\text{old}} = \mathbf{c}$ **then**
        break
    end if
    $\mathbf{c}^{\text{old}} \leftarrow \mathbf{c}$
    $i \leftarrow i + 1$
end while
return cluster centres $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k \in \mathbb{R}^D$, assignment vector $\mathbf{c}^{\text{old}} \in \mathbb{R}^n$

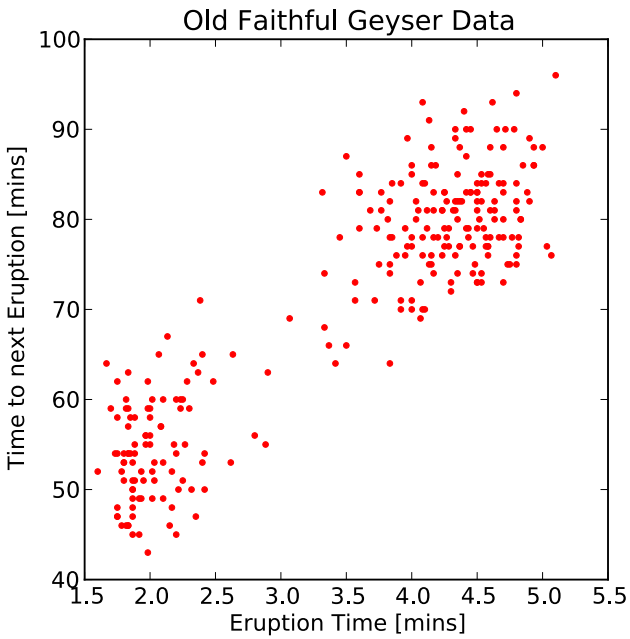
Application Example: Geyser Eruptions

Old Faithful Geyser
Yellowstone National Park,
USA

Famous data set for clustering

- Old Faithful Eruptions
- Two dimensions
  1. Time of Eruption [mins]
  2. Time until next Eruption [mins]

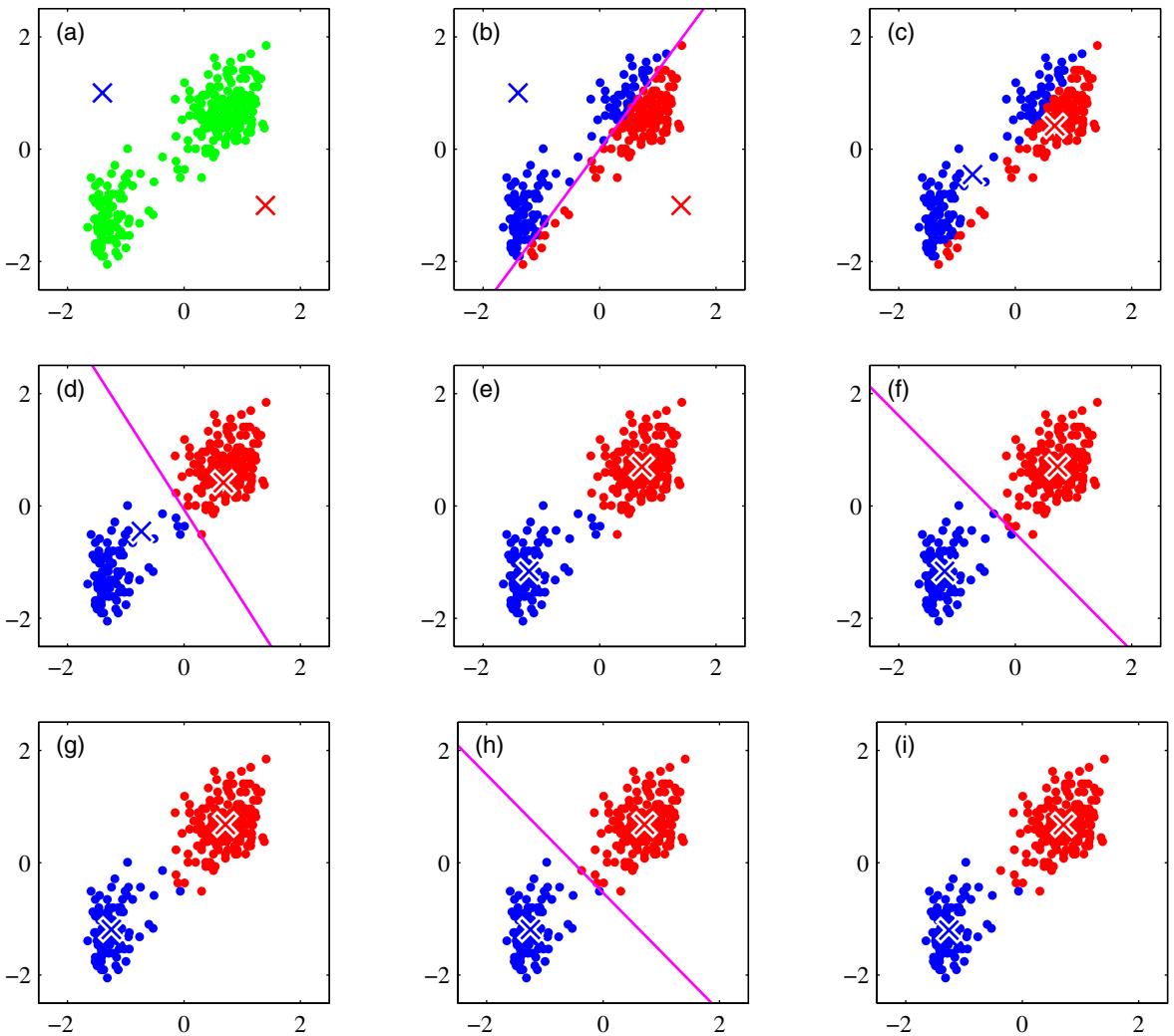
Application Example: Geyser Eruptions



Famous data set for clustering

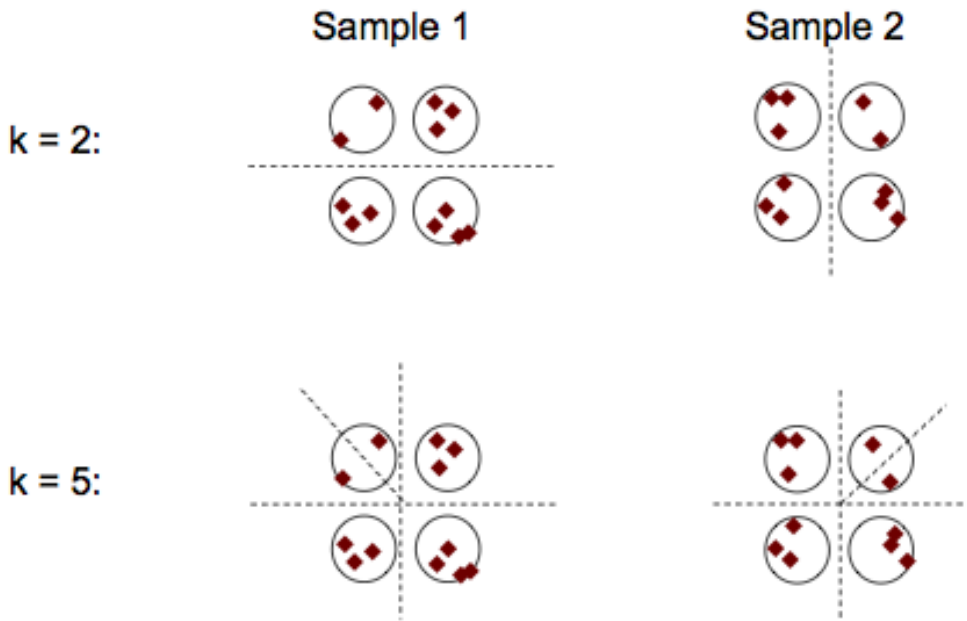
- Old Faithful Eruptions
- Two dimensions
  1. Time of Eruption [mins]
  2. Time until next Eruption [mins]

K-means Clustering Step-by-Step



Clustering Instability

Number of Clusters is a critical parameter



Clusterings are instable if number of clusters is too small or too large

Clustering Instability

- Number of clusters is critical hyper parameter
- In supervised settings we use cross-validation to optimize hyper-parameters for accuracy on test data
- How can we optimize the number of clusters?

$\rightarrow$ Choose that k that results in most **stable** clusterings
For a review see e.g. [von Luxburg, 2009]

Clustering Instability Algorithm

Algorithm 2 Clustering Instability

Require: data points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, clustering algorithm $\mathcal{A}$, maximal number of clusters K, iterations i.

Ensure: optimal number of clusters k^*

1: **for** $k = 2$ to K **do**
2: Resample data set (e.g. random draws with replacement)
3: **for** $it = 1$ to i **do**
4: Cluster data using algorithm $\mathcal{A}$ into k clusters
5: **end for**
6: Compute minimal (across all label permutations) distance between clusterings
7: **end for**
8: Chose that k that has the minimal instability over resamplings

Distance Measures for Real-Valued Data $\mathbf{x} \in \mathbb{R}^D$

Clustering Algorithms need a **distance function** $d(\mathbf{x}_i, \mathbf{x}_j)$

- For real valued data $\mathbf{x} \in \mathbb{R}^D$ we can use the Euclidean distance

$$d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 \tag{5}$$

- More robust (less sensitive to outliers) is the **city block distance** or $\mathcal{L}_1$ norm

$$d(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\|_1 \tag{6}$$

- Another alternative is the correlation coefficient (also called cosine similarity)

$$d(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_1^\top \mathbf{x}_2}{\|\mathbf{x}_1\|_2 \|\mathbf{x}_2\|_2} \tag{7}$$

For standardized data $\sum_i \mathbf{x}_i = 0, \ \sum_i \mathbf{x}_i^2 = 1$ maximizing correlation is the same as minimizing euclidean distance.

Distance Measures for Non-Real-Valued Data

- For ordinal variables $\mathbf{x} \in \{\text{low}, \text{medium}, \text{high}\}^D$ we can transform the values into real-valued numbers (for three possible values e.g. $1/3, 2/3, 3/3$) and then apply distance functions for real-valued data
- For categorial variables $\mathbf{x} \in \{\text{red}, \text{green}, \text{blue}\}^D$ we can use a binary coding for the differences

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sum_{d}^{D} \mathbf{x}_{id} \neq \mathbf{x}_{jd} \tag{8}$$

This metric is called **Hamming Distance**

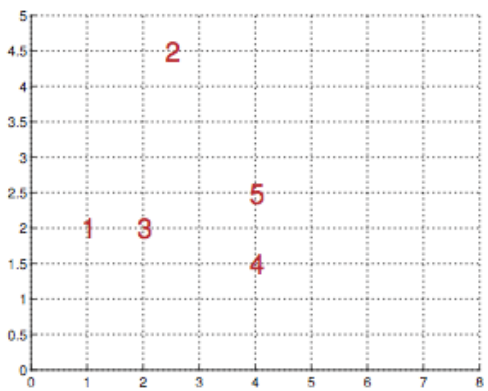
For the sake of simplicity we only consider the euclidean distance

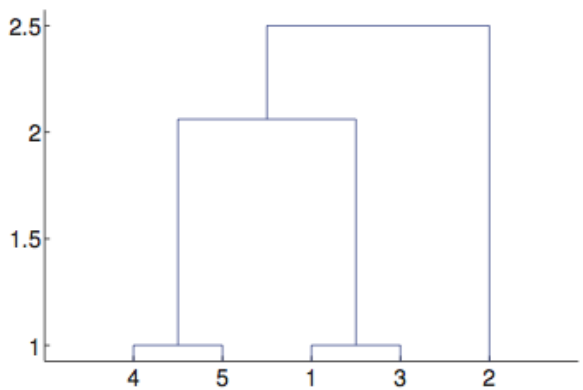
Hierarchical Clustering

- K-Means produces **flat** clusterings
- Often we are interested in a **hierarchy** of clusterings
- Examples:
  - Biological Species
  - Topics in Text Documents

Hierarchical Clustering

- K-Means produces **flat** clusterings
- Often we are interested in a **hierarchy** of clusterings
- Examples:
  - Biological Species
  - Topics in Text Documents

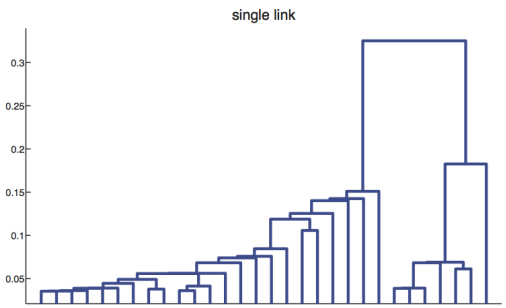




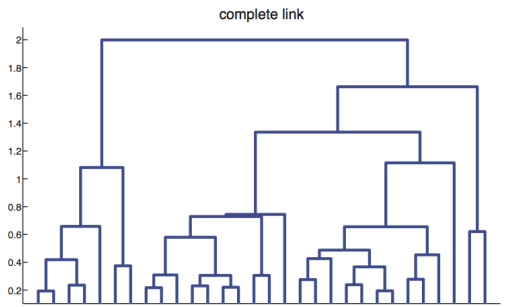
Examples Hierarchical Clustering

Dendrograms (binary clustering trees) of yeast gene expression data

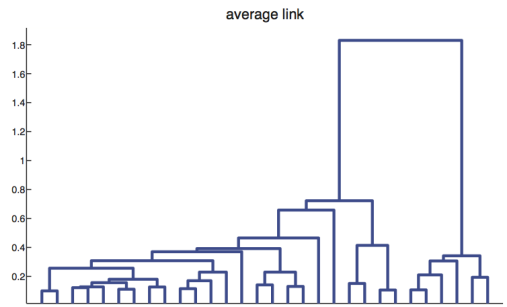
Single Linkage



Complete Linkage



Average Linkage



Taken from [Murphy, 2012]

Summary

- Clustering Algorithms find clusters in data
- Clustering Performance depends on distance function used
- K-Means is one of the most popular clustering algorithms
- For large data sets use Online K-Means
- Wrong number of clusters leads to unstable results
- Hierarchical clustering

References

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer US, 2007.

S. Lloyd. Least squares quantization in pcm. *Information Theory, IEEE Transactions on*, 28(2):129–137, Mar. 1982. ISSN 0018-9448.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning. The MIT Press, 1 edition, 2012. ISBN 0262018020,9780262018029.

U. von Luxburg. Clustering stability: An overview. *Foundations and Trends in Machine Learning*, 2(3):235–274, 2009.

Machine Learning

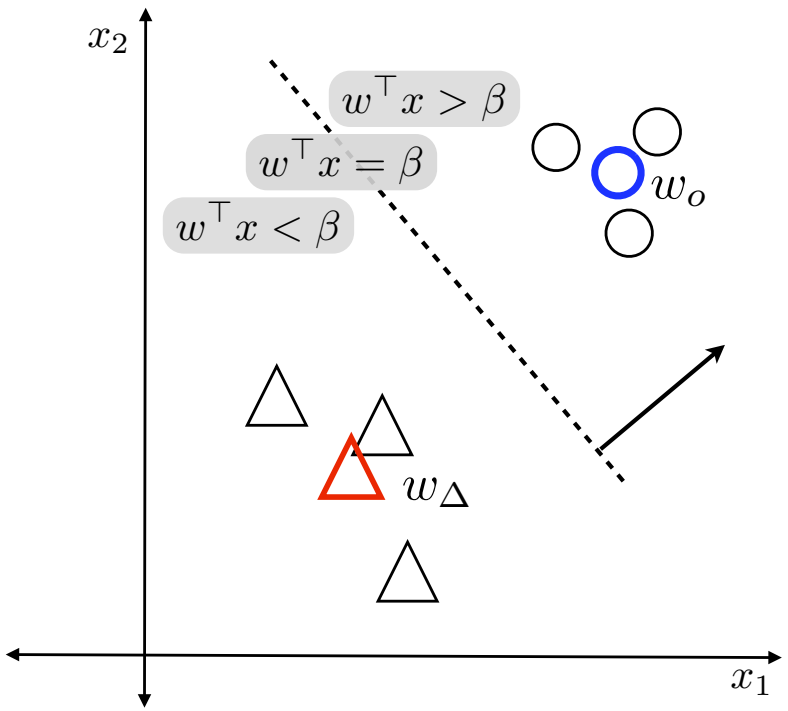
Neural Networks

Felix Bießmann

Beuth University & Einstein Center for Digital Future

June 24, 2019

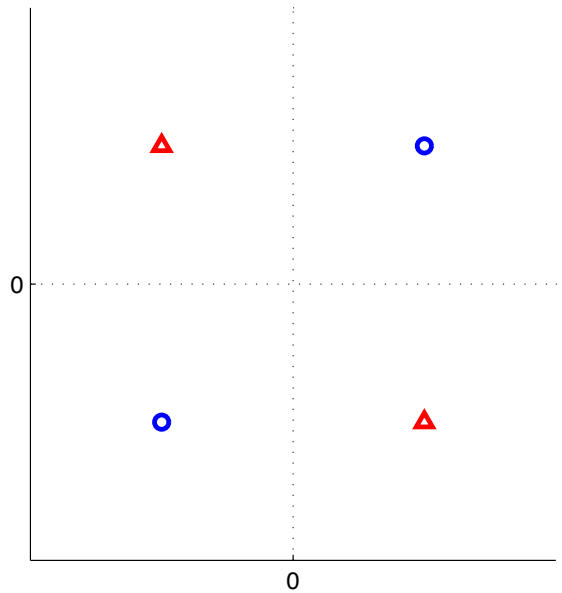
Linear Classification

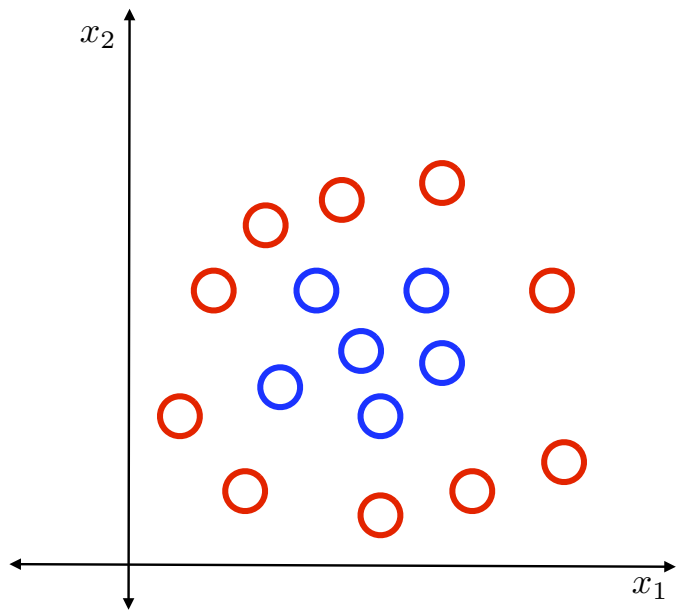


$$\phi(\mathbf{w}^\top \mathbf{x} - \beta) = \begin{cases} > 0 & \text{if } \mathbf{x} \text{ is from class } o \\ < 0 & \text{if } \mathbf{x} \text{ is from class } \Delta \end{cases}$$

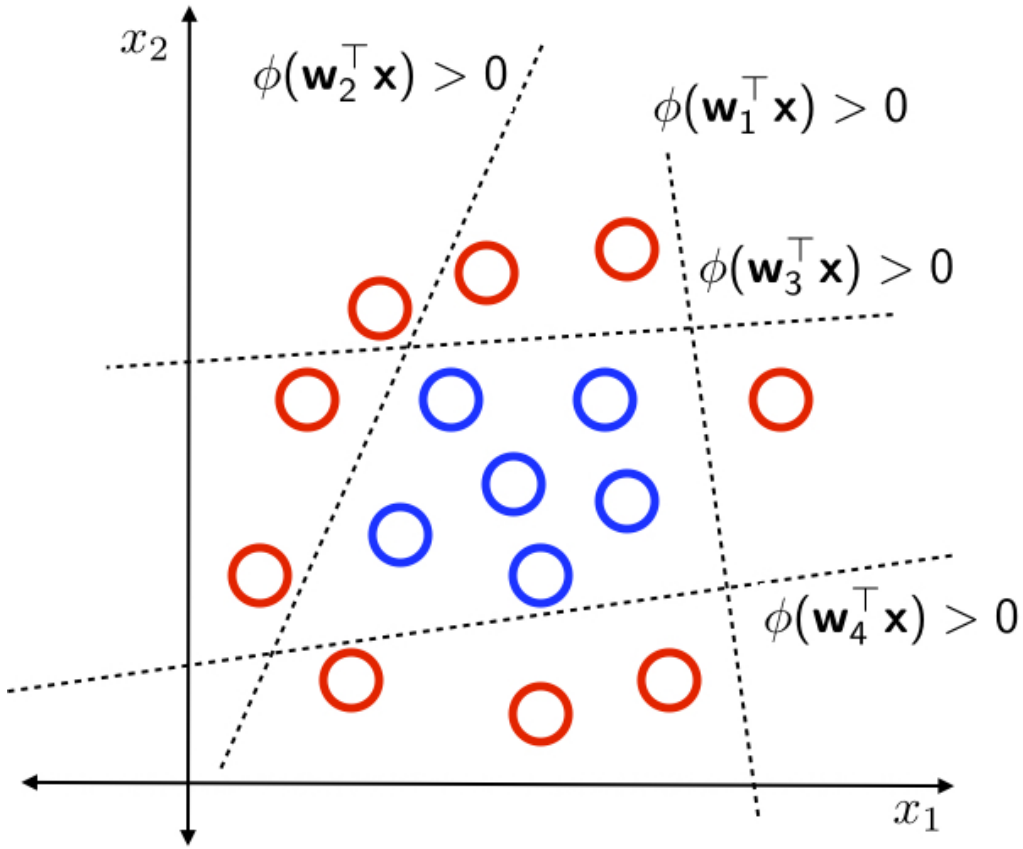
Problems with Perceptrons

Perceptrons can only learn linearly separable problems.



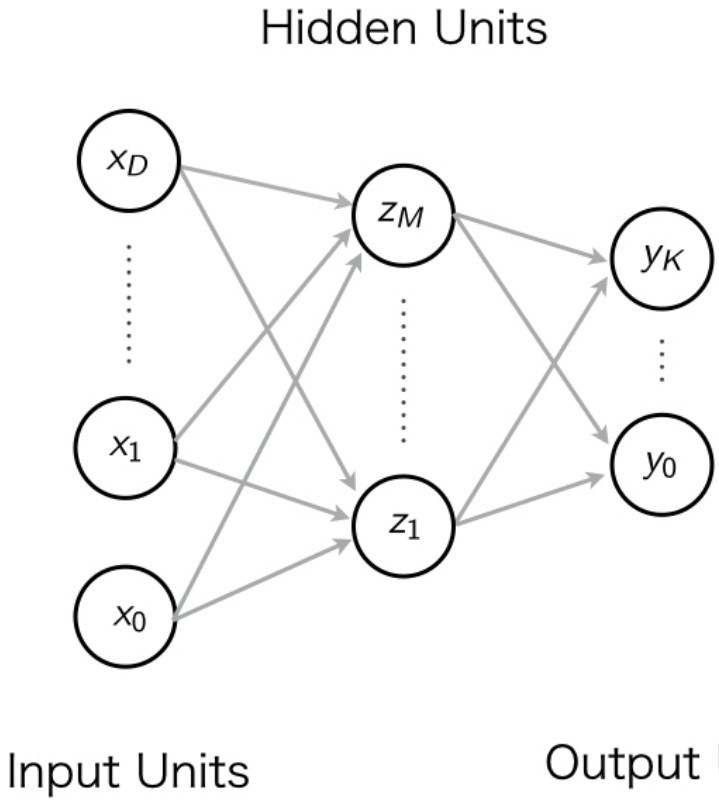


Deep Neural Networks



Deep Neural Networks

Combinations of Perceptrons (Multi Layer Perceptrons):



Neurons (Units), that are neither output nor input are called **Hidden Units**.

A Short History of Deep Learning

1943 First mathematical Neuron Model (Mcculloch and Pitts, 1943)

1957 Perceptron Algorithm (Rosenblatt, 1958)

1969 Perceptrons cannot solve non-linearen Problems (Minsky and Papert, 1969)

1970 Backpropagation: Efficient gradient computations (Linnainmaa, 1970)

1980 Computer Hardware $\approx$ 10,000 faster compared to 1960/1970 – Automatic Differentiation (Speelpenning, 1980)

1986 Backpropagation learns meaningful representations (Rumelhart et al., 1986), NETtalk (Sejnowski and Rosenberg, 1986)

1992 Support-Vector Machines (SVMs) (Boser et al., 1992)

2000 Computer Hardware (GPUs) $\approx$ 10,000 faster compared to 1980/1990, Bigger datasets render kernel SVMs computationally infeasible

2012 Deep Convolutional Networks wins ImageNet (Krizhevsky et al., 2012)

2014 Neural Machine Translation surpasses traditional methods

2017 Neural Networks for Reinforcement Learning excell at Go (AlphaGo Zero)

2018 ImageNet Moment for Neural Language Models (BERT / ELMO)

Sources: Juergen Schmidhuber's page and others

Universal Approximation Theorem

[Cybenko, 1989]

Multilayer Perceptrons with one hidden layer and a finite number of hidden units can approximate any function.

Training of Deep Neural Networks

- Training: Gradient Descent
- Problem: Gradient Computations
  - Mathematically challenging for complex models
  - Computationally challenging

$\rightarrow$ Solution: **Backpropagation**

  - Elegant formulation
  - Efficient implementation

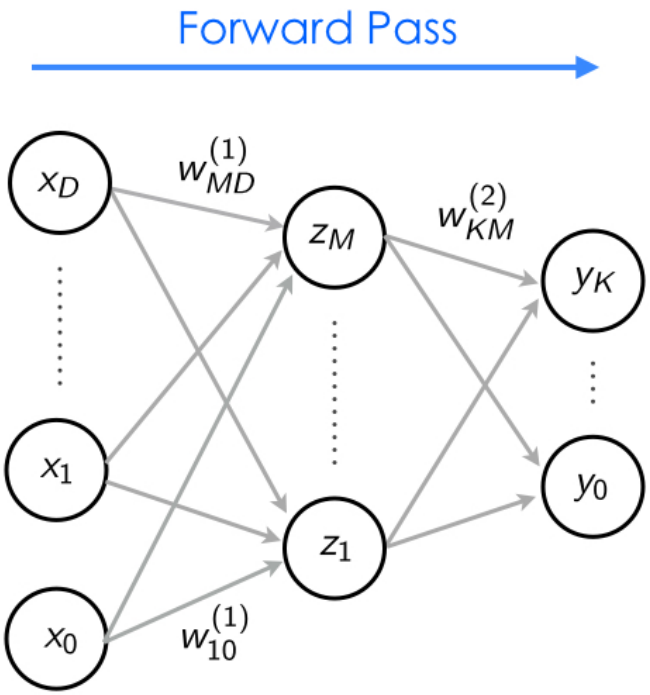
Backpropagation Algorithm

Algorithm 1 Backpropagation Algorithm - Pseudocode

Require: Data $\mathbf{X} \in \mathbb{R}^{D \times N}$, labels $\mathbf{Y} \in \mathbb{R}^{K \times N}$, untrained network
Ensure: Network parameters $\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(V)}$

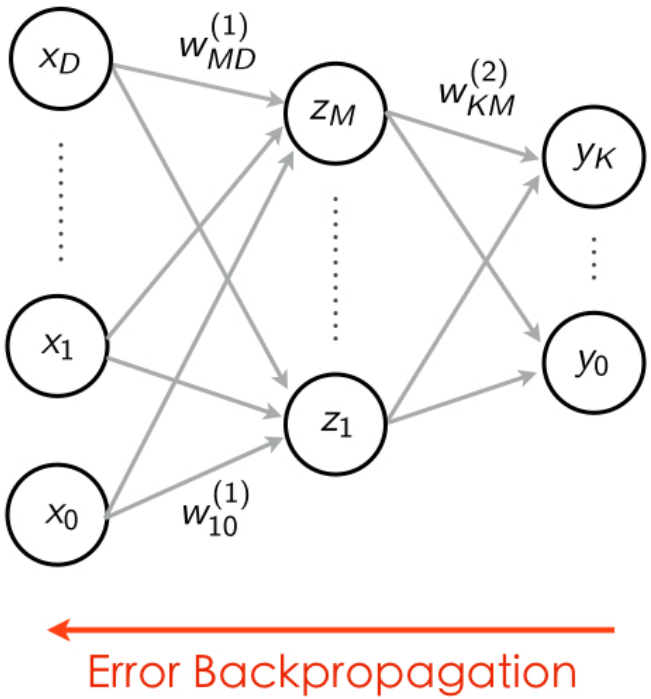
```
while Not converged do
    # Compute network predictions
    # Evaluate error function
    # Propagate error from output layer back to input layer
    # Take gradient descent step
end while
```

Learning with Backpropagation in Neural Networks



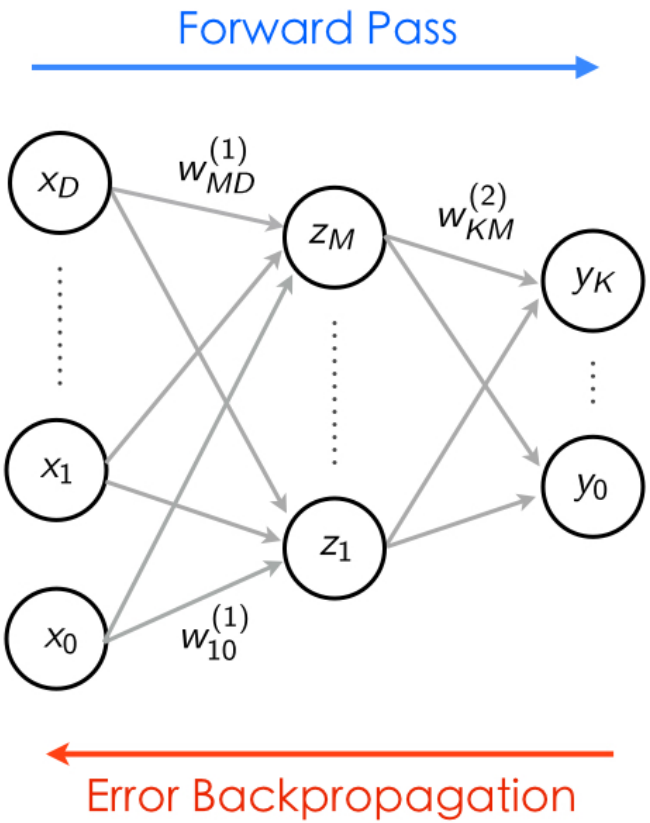
Computation of network predictions is called **Forward Propagation**.

Learning with Backpropagation in Neural Networks



Backpropagation refers to efficient computation of error function gradients for all connections.

Learning with Backpropagation in Neural Networks



After a forward and backward pass a gradient step is performed.

Summary

- Perceptrons cannot separate linearly non-separable problems
- Using combinations and stacking of standard Perceptrons, Multi Layer Perceptrons (MLPs) can approximate any function with one hidden layer
- Gradient descent for MLPs is challenging, mathematically and computationally
- Backpropagation: Efficient Gradient computation

References

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer US, 2007.

G. Cybenko. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. 2003.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning. The MIT Press, 1 edition, 2012. ISBN 0262018020,9780262018029.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, Nov. 1958.

B. Widrow and M. E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104, New York, 1960. IRE.