Machine Learning Slides

draft - under construction

Jorge S. Marques
(revised by Alexandre Bernardino)

September 7, 2024

Table of contents

Machine Learning

Linear Regression

Regularization

Evaluation & Generalization

Optimization

Neural networks

Data classification

Linear classifiers

Support vector machines

Decision Trees and Random Forest

What is machine learning?

Many engineering problems can be solved by using models that depend on a small number of variables.

Examples:

- motion of a rocket $\rightarrow$ Newton law: $m\ddot{x}(t) = F(t)$
- electromagnetic waves $\rightarrow$ Maxwell equations

... but other problems are more complex and cannot be tackled with closed form expressions.

Hospital problem

Suppose a patient enters in an hospital and we wish to predict if he/she is going to live or die.

There is no general principle that can be used to solve this problem.

May be we have a data set of previous examples, including information about the patient status (medical tests / symptoms) and the outcome (live/die).

T_1	T_2	...	...	...	T_p	y

T_i *ith* medical test / /symptom

y outcome

How can we use this information to predict the outcome for a new patient?

What is Machine Learning?

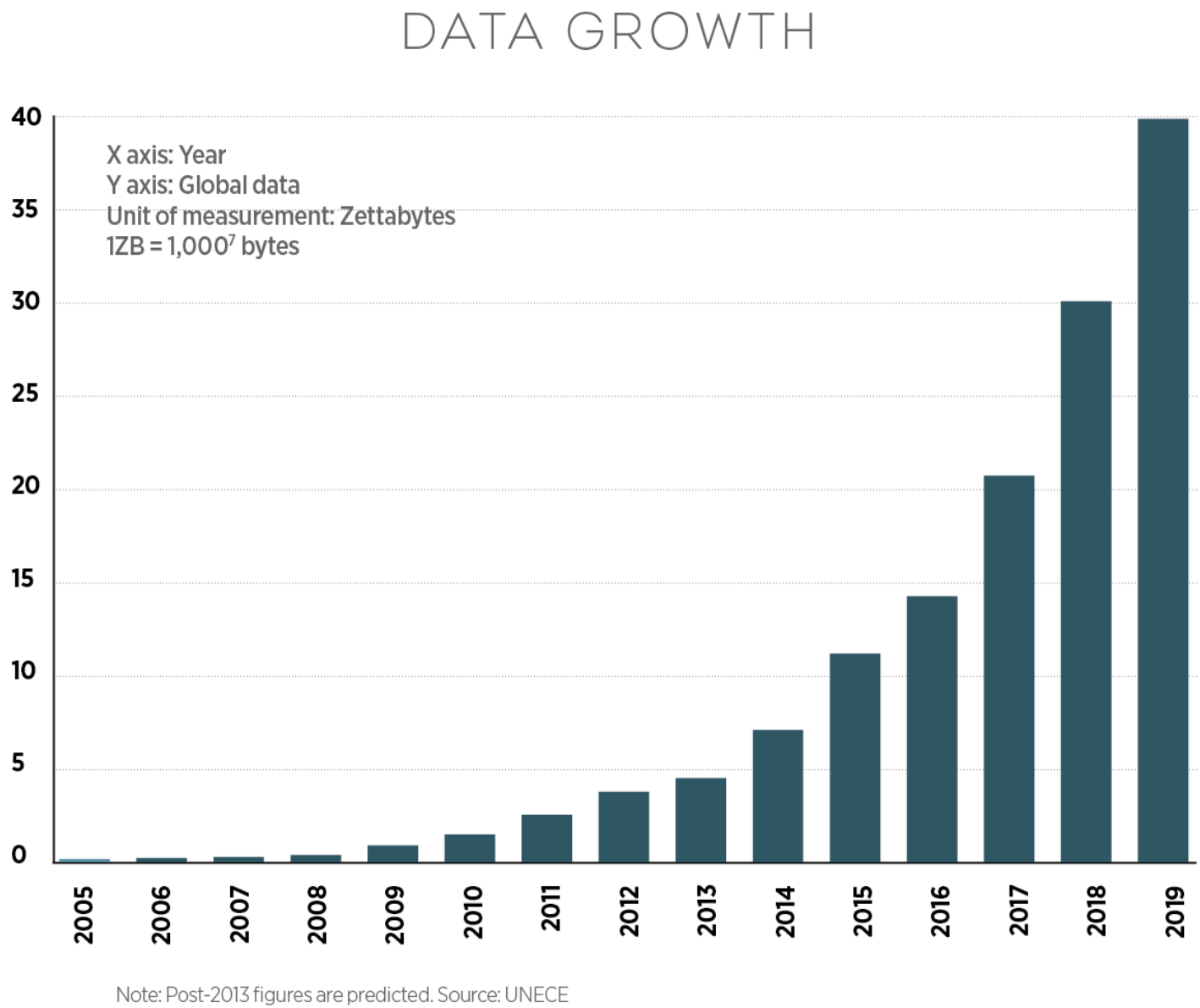
"the field of study that gives computers the ability to learn without being explicitly programmed." (Arthur Samuel, 1959)

Arthur Samuel was a pioneer in the area of Machine Learning.

"A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience." (Tom Mitchell, 1998)

Tom Mitchell is a professor of Computer Science at CMU.

Data growth in internet



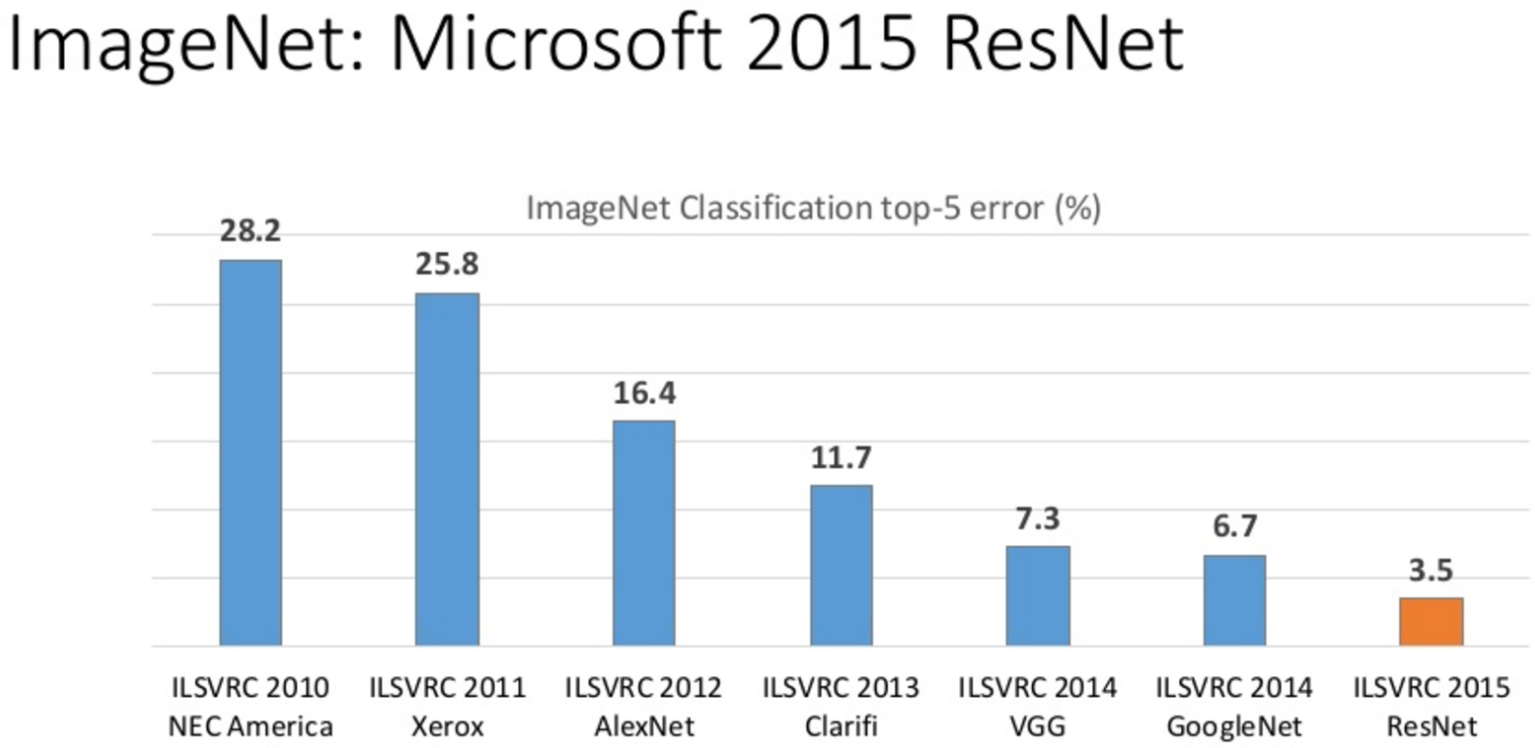
The economist

Applications

- prediction
- time series analysis
- speech recognition - conversion of the speech signal into text
- machine translation
- detection of failures
- image denoising
- human activity recognition
- medical image analysis - *e.g.*, cancer detection in images
- robot navigation
- self driving car

Some of these are amongst the most difficult problems in engineering.

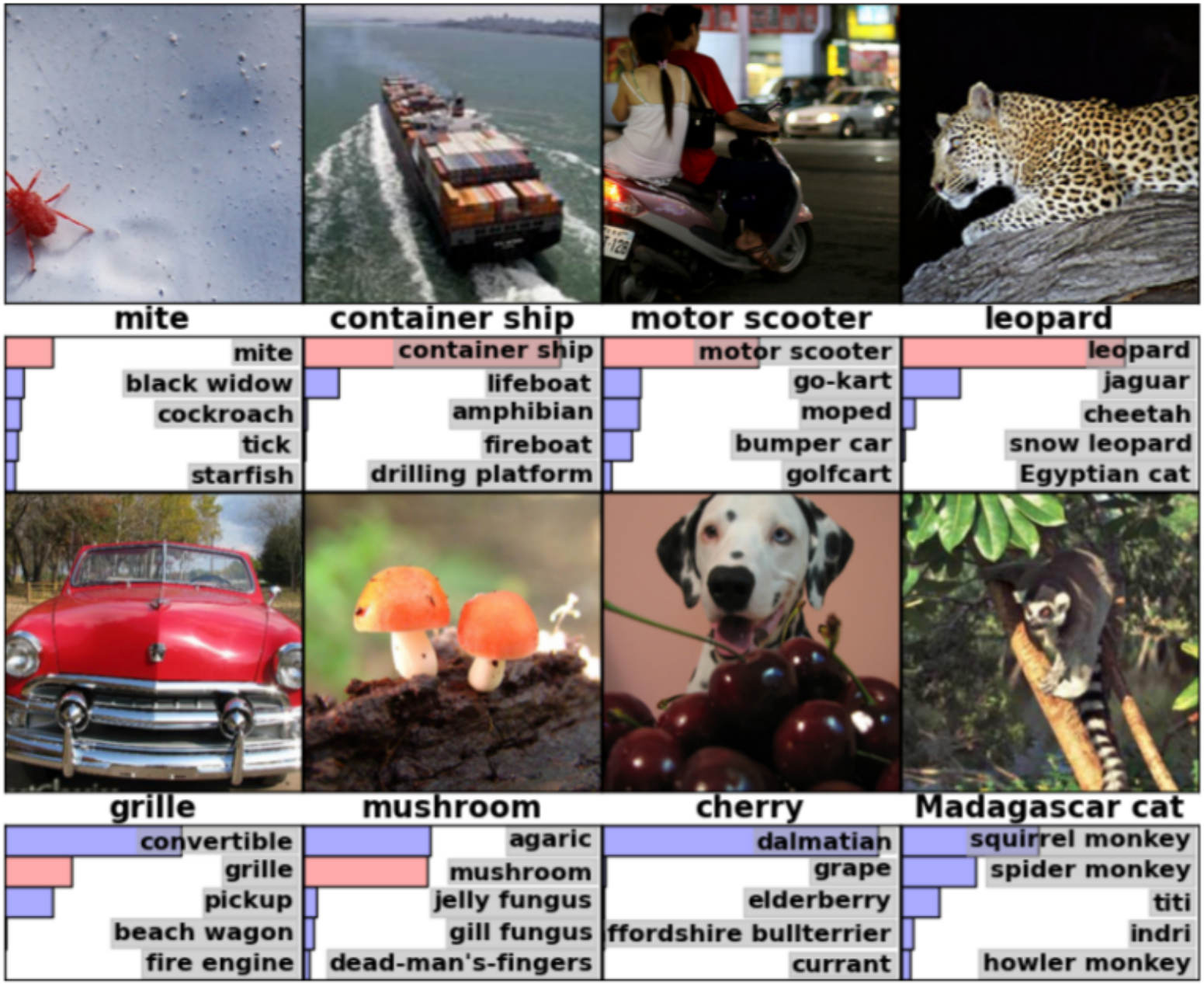
Amazing progress in image recognition



Xuedong Huang - Deep Learning and Intelligent Applications

Current machine learning methods perform comparably to humans in this task.

Visual recognition - AlexNet (2012)



Alexnet 2012 - Krizhevsky, Sutskever, Hinton

Visual recognition - AlexNet (2012)

Alexnet 2012 - Krizhevsky, Sutskever, Hinton

Image description (2015)

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

"man in blue wetsuit is surfing on wave."

Karpathy, Fei-Fei - CVPR 2015

Course overview

Structure: lectures (4h/week) + lab (1.5h/week) + problem sessions (1.5h/week).

Lab: students organized in groups of 2, should perform a project and a 10 pages report. The project includes 2 parts: regression problem and a classification problem.

Programming: Python. An introduction is provided in 1st week (problem sessions).

Grading: exam (50%) + Lab (50%).

Learning problems

There are several learning problems. The major categories are:

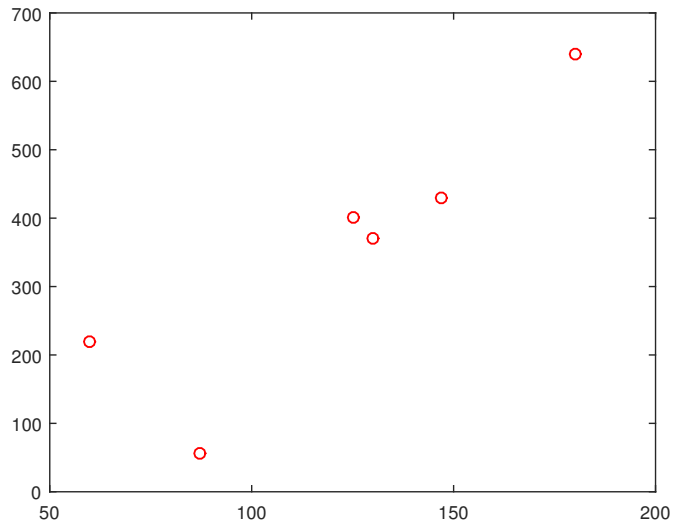
- Supervised learning - the computer receives a set of inputs and desired outputs and aims to find the map between them (e.g., hospital problem).
- Unsupervised learning - the computer receives a set of inputs but no desired outputs. The goal is to find the structure of data (probability distribution, groups).
- Reinforcement learning - aims to learn the behavior of software agents or robots based on feedback from the environment (e.g., learning how to play a game).

This course is focused on the first learning category.

Example 1 (supervised learning)

Suppose we want to predict the price of a flat in Lisbon (in K euros), knowing its area (in m^2). Fortunately, we know some examples.

area	price
130	370
60	220
87	57
125	400
147	430
180	640



The area is known as a feature and the price is the outcome.

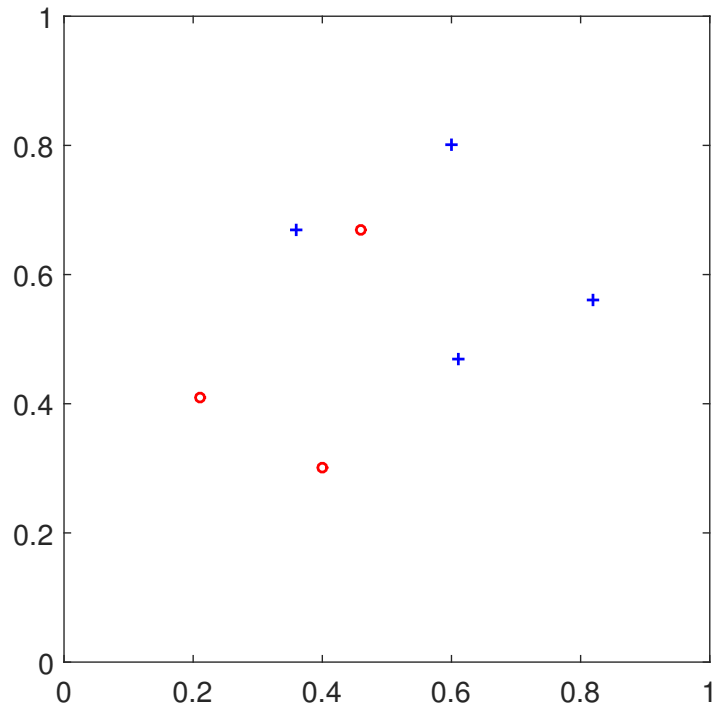
Question: how can we predict the price of a flat?

Example 2 (supervised learning)

A fisher boat has a sonar system that measures the length and volume of fish (features).

Given the table with the length and volume of tuna and swordfish (class), we wish to design a system that predicts the class.

length	volume	class
0.36	0.67	tuna
0.82	0.56	tuna
0.46	0.67	sword
0.40	0.30	sword
0.60	0.80	tuna
0.61	0.47	tuna
0.21	0.41	sword



Question: how can we predict the type of fish?

Key concepts

Identify the key concepts in previous examples:

- features
- outcome
- predictor

Problem formulation: supervised learning

Given an input variable $\boldsymbol{x} \in \mathbb{R}^p$ (vector of features), we wish to predict an output variable y (outcome) *i.e.*, we wish to find a map between the input space and the output space, assuming we know a set input-output pairs.

This operation is known as model learning.

Problem: Given a set of examples (training set)

$$\mathcal{T} = \left\{ (\boldsymbol{x}^{(i)}, y^{(i)}), i = 1, \ldots, n \right\}, \qquad \boldsymbol{x}^{(i)} \in \mathbb{R}^p,$$

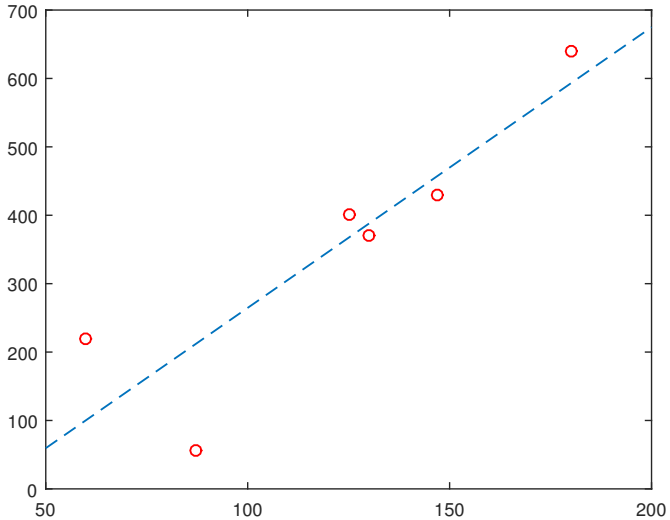
we wish to estimate a function (predictor)

$$\hat{y} = f(\boldsymbol{x}),$$

such that $\hat{y}$ is, in some sense, close to y.

Regression vs classification

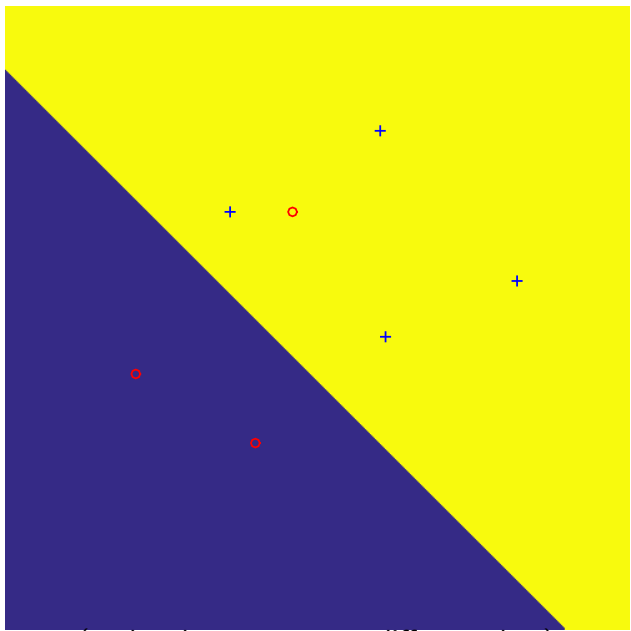
If the output y is continuous ($y \in \mathbb{R}$ or $y \in \mathbb{R}^k$) the problem is known as a regression problem.



If the output y is a label (categorical variable)

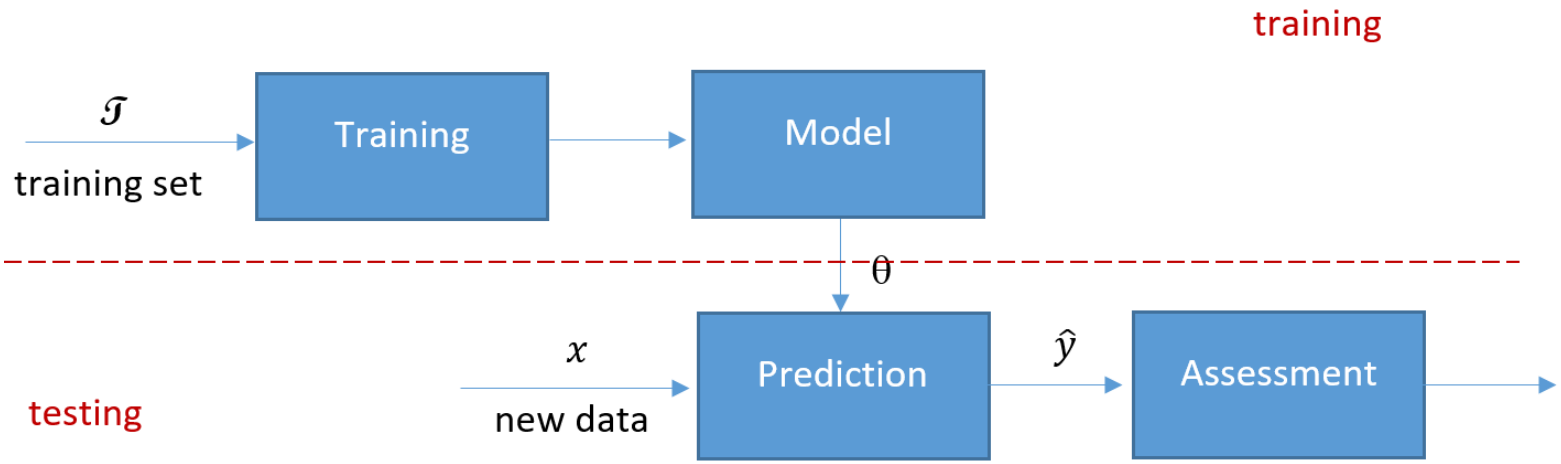
$$y \in \Omega, \qquad \Omega = \{\omega_0, \ldots, \omega_{K-1}\},$$

the problem is known as a classification problem.



(each color represents a different class)

System architecture



The design of a machine learning system comprises a training phase to learn a model and a testing phase to predict new data and to eventually assess the model performance.

This diagram does not consider the choice of features and their extraction, *e.g.*, if we are dealing with an image or speech analysis problem, what features do we extract from the signal. This issue is application dependent and will not be considered.

Main questions

The block diagram suggests three main questions:

- what class of functions should we consider?
- how do we fit the function f to the training data *i.e.*, how do we select the function?
- how do we evaluate the predictor?

These questions have multiple answers that we will discuss along this course.

Data sets

Data sets are important tools to train and evaluate machine learning systems. They allow to compare different techniques and they often foster the development of new methods.

There are many sites with data sets. One example is:
https://archive.ics.uci.edu/datasets

An old data set: Iris flower (Fisher)

setosa

versicolor

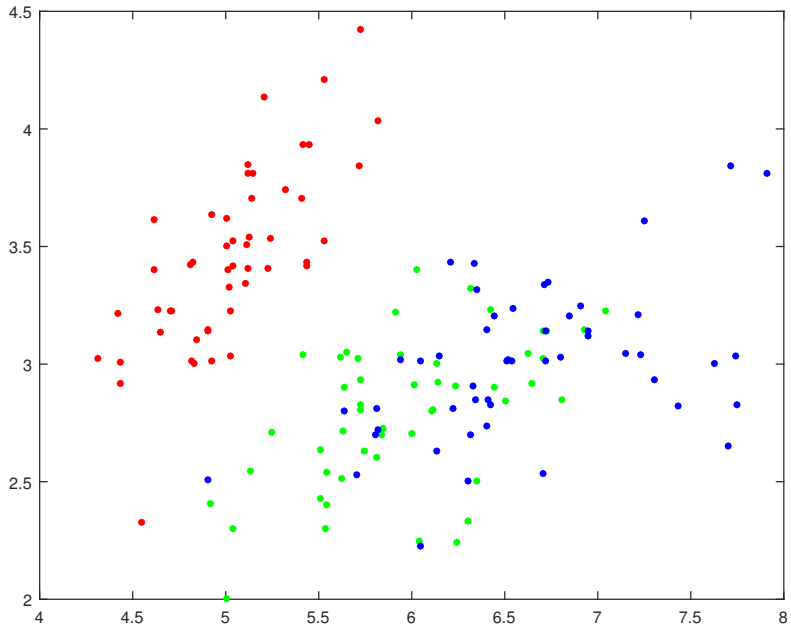
virginica

Wikimedia

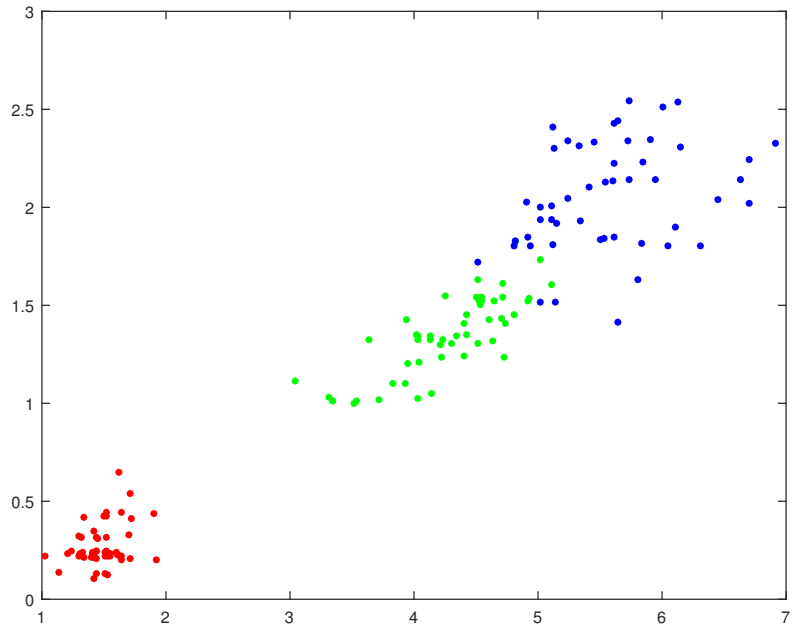
Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	I. setosa
4.9	3.0	1.4	0.2	I. setosa
4.7	3.2	1.3	0.2	I. setosa
4.6	3.1	1.5	0.2	I. setosa
5.0	3.6	1.4	0.3	I. setosa
⋮	⋮	⋮	⋮	⋮
7.7	2.6	6.9	2.3	I. virginica
7.9	3.8	6.4	2.0	I. virginica

150 examples

An old data set: Iris flower (Fisher)



feature 1 vs 2



feature 3 vs 4

Species: setosa (red), versicolor (green), virginica (blue)

Please note that the scale is not the same in both axis.

A recent data set: ImageNet 2012

URL: www.image-net.org

Data set: 10 million images Classes: 1000+

Nearest neighbor method

Suppose we wish to predict a variable y knowing an input vector $\boldsymbol{x} \in \mathbb{R}^p$.

Suppose we also know a collection of training examples (training set)

$$\mathcal{T} = \left\{ (\boldsymbol{x}^{(i)}, y^{(i)}), i = 1, \ldots, n \right\}.$$

A simple strategy to predict y for new values of $\boldsymbol{x}$ consists of finding the training pattern $\boldsymbol{x}^{(i)}$ nearest to $\boldsymbol{x}$ and approximating y by $y^{(i)}$.

Let $\left\{ (\boldsymbol{x}_{(1)}, y_{(1)}), \ldots, (\boldsymbol{x}_{(n)}, y_{(n)}) \right\}$ be a reordering of the training set such that

$$\|\boldsymbol{x}_{(1)} - \boldsymbol{x}\| \leq \|\boldsymbol{x}_{(2)} - \boldsymbol{x}\| \leq \cdots \leq \|\boldsymbol{x}_{(n)} - \boldsymbol{x}\|.$$

The nearest neighbor (NN) method assigns $\boldsymbol{x}$ to the outcome of the nearest neighbor.

$$f(\boldsymbol{x}) = y_{(1)}.$$

This is valid for both classification and regression problems.

k nearest neighbor

The NN method can be extended to take into account not one but k nearest neighbors of $\boldsymbol{x}$.

In classification problems, the predicted class is chosen as the most voted class in the sequence $(y_{(1)}, \ldots, y_{(k)})$

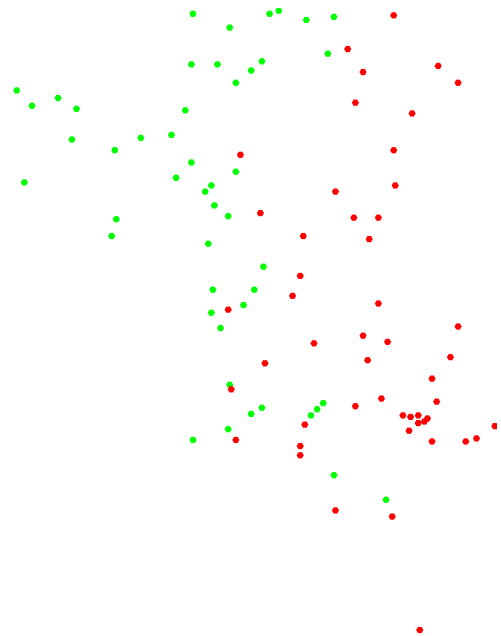
$$f(\boldsymbol{x}) = \text{most voted class in } (y_{(1)}, \ldots, y_{(k)}).$$

In regression problems, the predicted value in chosen as the average of $(y_{(1)}, \ldots, y_{(k)})$

$$f(\boldsymbol{x}) = \frac{1}{k} \sum_{i=1}^{k} y_{(i)}.$$

Example: supervised classification problem

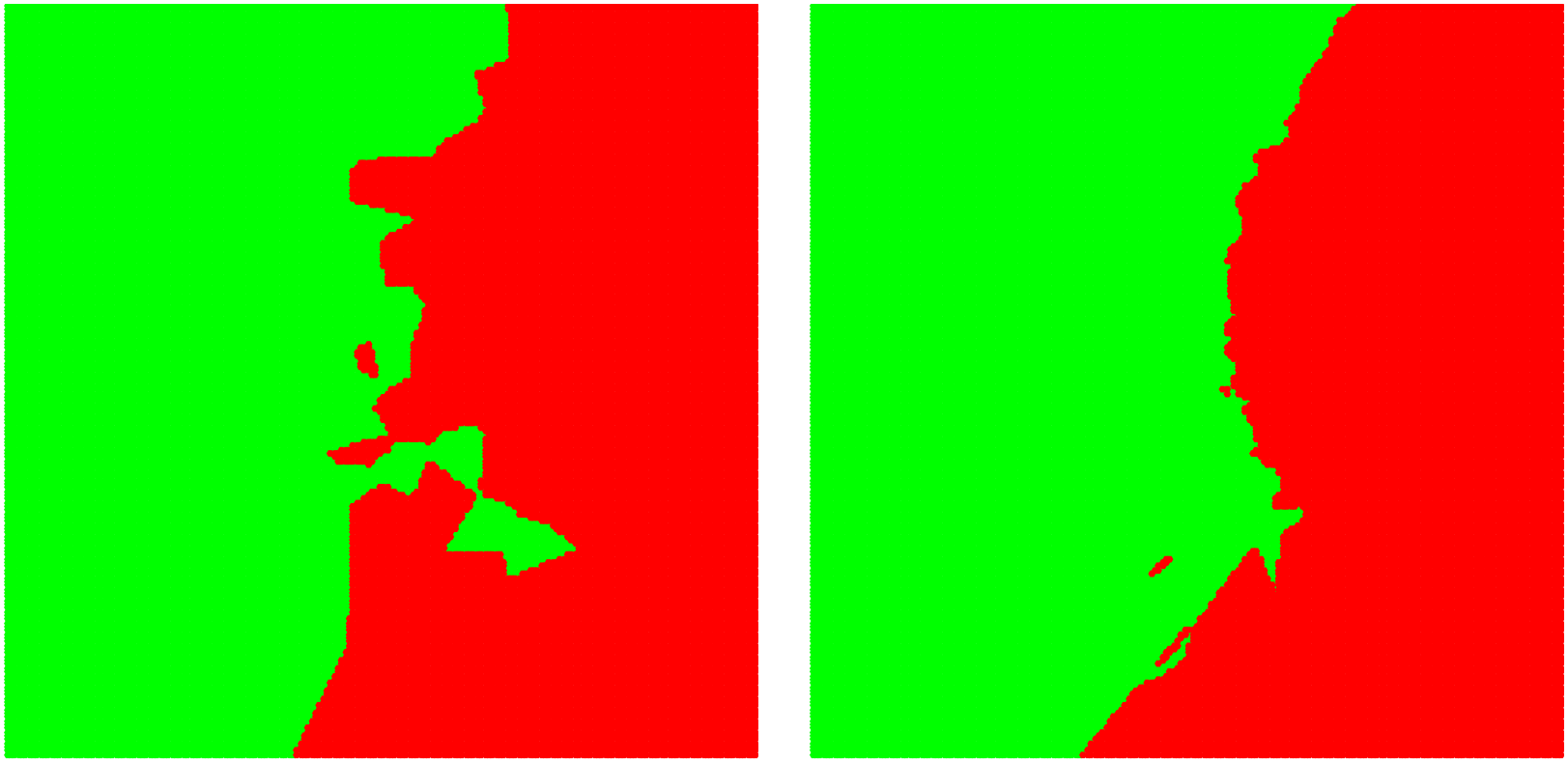
Consider a binary classification problem. The training set is shown in the figure and was generated by a mixture of Gaussians.



training data

How would you classify this data?

k nearest neighbor



Decision regions of kNN classifier with $k = 1$ (left) and $k = 10$ (right)

What k would you choose?

(k is known as a hyperparameter)

Recommended Bibliography

- T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning, Springer, 2009
- C. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
- T. Michell, Machine Learning, McGraw Hill, 1997.
- J. S. Marques, Reconhecimento de Padrões. Métodos Estatísticos e neuronais, ISTPress, 2nd ed. 2005.
- R. Duda, P. Hart, D. Stork, Pattern Classification, Wiley, 2nd edition, 2000.
- L. Almeida, Multilayer Perceptrons, in Handbook of Neural Computation, Oxford Press, 1997.
- I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, `http://www.deeplearningbook.org`, 2016
- T. Fletcher, Support Vector Machines, UCL, 2008
- K. Murphy, Probabilistic Machine Learning: An Introduction, MIT Press, `https://probml.github.io/pml-book/book1.html`, 2022

Machine Learning

Linear Regression

Regularization

Evaluation & Generalization

Optimization

Neural networks

Data classification

Linear classifiers

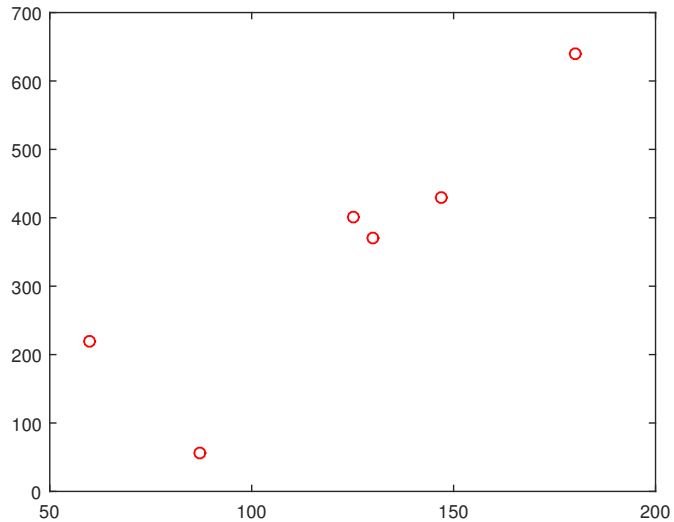
Support vector machines

Decision Trees and Random Forest

Regression Problem

Consider the data set $\mathcal{T} = \left\{(x^{(i)}, y^{(i)}), i = 1, \ldots, n\right\}$ defined in the table.

area	price
130	370
60	220
87	57
125	400
147	430
180	640



We wish to predict the price of a flat in Lisbon, taking its area into account.

The simplest prediction model is a straight line

$$\hat{y} = f(x) = \beta_0 + \beta_1 x.$$

β_0 is called the intercept or offset.

Predictor estimation

How should we estimate the coefficients β_0, β_1?

If we know one training example $(x^{(1)}, y^{(1)})$, we obtain a single equation

$$\hat{y}^{(1)} = \beta_0 + \beta_1 x^{(1)} \rightarrow \text{infinite } (\beta_0, \beta_1) \text{ solutions.}$$

If we know two training examples, we obtain two equations

$$\begin{aligned} \hat{y}^{(1)} &= \beta_0 + \beta_1 x^{(1)} \\ \hat{y}^{(2)} &= \beta_0 + \beta_1 x^{(2)} \end{aligned} \rightarrow \text{unique solution, but bad (noisy).}$$

If we know three training examples, we obtain three equations

$$\begin{aligned} \hat{y}^{(1)} &= \beta_0 + \beta_1 x^{(1)} \\ \hat{y}^{(2)} &= \beta_0 + \beta_1 x^{(2)} \\ \hat{y}^{(3)} &= \beta_0 + \beta_1 x^{(3)} \end{aligned} \rightarrow \text{no solution, impossible.}$$

To solve this problem, we must assume that there is an error between the output of the model $\hat{y}^{(i)}$ and the data $y^{(i)}$.

Prediction loss

We assume that there is a prediction error associated to each training example

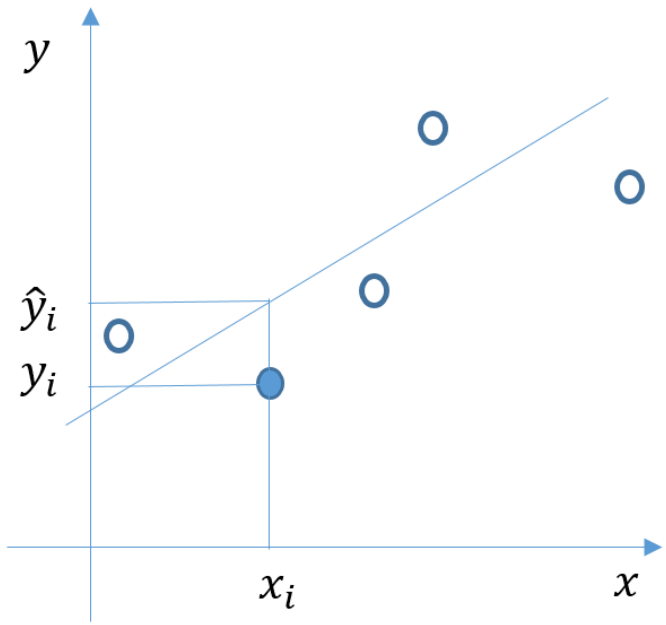
$$e^{(i)} = y^{(i)} - \hat{y}^{(i)} = y^{(i)} - f(x^{(i)}),$$

and define a quadratic loss (cost)

$$L(y^{(i)}, \hat{y}^{(i)}) = (y^{(i)} - \hat{y}^{(i)})^2.$$

The total loss in the training set

$$SSE = \sum_{i=1}^{n} (y^{(i)} - f(x^{(i)}))^2,$$



also known as (aka) sum of squared errors (SSE), or least squares criterion (LS).

Minimization: first order model

Model fit is achieved by minimizing the total loss in the training set

$$\min_{\beta_0,\beta_1} \sum_{i=1}^{n} (y^{(i)} - \beta_0 - \beta_1 x^{(i)})^2.$$

The minimum is achieved at a point $(\hat{\beta}_0, \hat{\beta}_1)$ such that the partial derivatives are zero (gradient vector is the null vector)

$$\nabla_\beta SSE = \begin{bmatrix} \frac{\partial SSE}{\partial \beta_0} \\ \frac{\partial SSE}{\partial \beta_1} \end{bmatrix} = 0,$$

This leads to

$$\begin{cases} \frac{\partial SSE}{\partial \beta_0} = 0 \\ \frac{\partial SSE}{\partial \beta_1} = 0. \end{cases} \Rightarrow \begin{cases} -2\sum_{i=1}^{n} (y^{(i)} - \hat{\beta}_0 - \hat{\beta}_1 x^{(i)}) = 0 \\ -2\sum_{i=1}^{n} (y^{(i)} - \hat{\beta}_0 - \hat{\beta}_1 x^{(i)}) x^{(i)} = 0. \end{cases}$$

Note: in this course we consider gradients as column vectors.

Analytic optimization

$$\begin{cases} -2\sum_{i=1}^{n}(y^{(i)} - \hat{\beta}_0 - \hat{\beta}_1 x^{(i)}) = 0 \\ \\ -2\sum_{i=1}^{n}(y^{(i)} - \hat{\beta}_0 - \hat{\beta}_1 x^{(i)})x^{(i)} = 0. \end{cases}$$

$$\begin{cases} \sum_{i=1}^{n}\hat{\beta}_0 & +\sum_{i=1}^{n}\hat{\beta}_1 x^{(i)} & = \sum_{i=1}^{n} y^{(i)} \\ \\ \sum_{i=1}^{n}\hat{\beta}_0 x^{(i)} & +\sum_{i=1}^{n}\hat{\beta}_1 x^{(i)}x^{(i)} & = \sum_{i=1}^{n} y^{(i)}x^{(i)}. \end{cases}$$

This leads to (normal equations):

$$\begin{bmatrix} \sum_{i=1}^{n} 1 & \sum_{i=1}^{n} x^{(i)} \\ \sum_{i=1}^{n} x^{(i)} & \sum_{i=1}^{n} x^{(i)2} \end{bmatrix} \begin{bmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} y^{(i)} \\ \sum_{i=1}^{n} y^{(i)}x^{(i)} \end{bmatrix}.$$

By solving this system of equations, we obtain $(\hat{\beta}_0, \hat{\beta}_1)$.

Analytic optimization

To guarantee a minimum is achieved at this point, we should evaluate the matrix of second derivatives (Hessian matrix)

$$H = \begin{bmatrix} \frac{\partial^2 SSE}{\partial \beta_0^2} & \frac{\partial^2 SSE}{\partial \beta_0 \partial \beta_1} \\ \frac{\partial^2 SSE}{\partial \beta_1 \partial \beta_0} & \frac{\partial^2 SSE}{\partial \beta_1^2} \end{bmatrix} = 2 \begin{bmatrix} \sum_{i=1}^{n} 1 & \sum_{i=1}^{n} x^{(i)} \\ \sum_{i=1}^{n} x^{(i)} & \sum_{i=1}^{n} x^{(i)2} \end{bmatrix},$$

and check if it is positive definite.

Reminder: positive definite matrix

A symmetric $p \times p$ matrix, M, is a positive definite matrix if the scalar $\boldsymbol{z}^T M \boldsymbol{z}$ is positive for every non-zero vector $\boldsymbol{z} \in \mathbb{R}^p$.

If M is a positive definite matrix, then

- it is a non singular matrix ($\det M \neq 0$);
- all the eigenvalues are real and positive;
- all the leading principal minors are positive. The *kth* leading principal minor of a matrix M is the determinant of its upper-left k by k sub-matrix.

Exercise

Suppose we remove the average value of the feature x and outcome y,

$$x' \leftarrow x - \bar{x} \qquad y' \leftarrow y - \bar{y},$$

where $\bar{x}, \bar{y}$ are average values computed in the training set.

Show that the least squares estimates of β_0 is $\hat{\beta}_0 = 0$.

This result is useful to simplify the estimation of the β coefficients.

Exercise

Minimize

$$SSE = \sum_{i=1}^{n} \left(y'^{(i)} - \beta'_0 - \beta'_1 x'^{(i)} \right)^2,$$

$$\frac{\partial SSE}{\partial \beta_0} = 0 \Rightarrow -2 \sum_{i=1}^{n} \left(y'^{(i)} - \hat{\beta}'_0 - \hat{\beta}'_1 x'^{(i)} \right) = 0,$$

$$\sum_{i=1}^{n} y'^{(n)} - n\hat{\beta}'_0 - \hat{\beta}'_1 \sum_{i=1}^{n} x'^{(n)} = 0,$$

$$0 - n\hat{\beta}'_0 - 0 = 0,$$

$$\hat{\beta}'_0 = 0.$$

Linear regression model (general case)

Let us extend linear regression to the general case in which we have p features $x_1,\ x_2, \ldots, x_p \in \mathbb{R}$. The linear regression model is given by

$$\hat{y} = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p.$$

Using vector notation, we obtain

$$\hat{y} = [1\ x_1\ \ldots\ x_p] \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} = \begin{bmatrix} 1 & \boldsymbol{x}^T \end{bmatrix} \boldsymbol{\beta}\ ,$$

where $\boldsymbol{x} = [x_1\ x_2, \ldots, x_p]^T \in \mathbb{R}^p$ and $\boldsymbol{\beta} = [\beta_0,\ \beta_1,\ \ldots, \beta_p]^T \in \mathbb{R}^{p+1}$, $\hat{y} \in \mathbb{R}$.

Problem formulation

Consider a training set $\mathcal{T} = \left\{(\boldsymbol{x}^{(i)}, y^{(i)}), i = 1, \ldots, n\right\}$, where $\boldsymbol{x}^{(i)} \in \mathbb{R}^p$ and $y^{(i)} \in \mathbb{R}$, $i = 1, \ldots, n$.

The linear model

$$f(\boldsymbol{x}) = \begin{bmatrix} 1 & \boldsymbol{x}^T \end{bmatrix} \boldsymbol{\beta},$$

is trained by finding the vector of coefficients $\hat{\boldsymbol{\beta}} \in \mathbb{R}^{p+1}$ that minimizes the total cost

$$SSE(\boldsymbol{\beta}) = \sum_{i=1}^{n} (y^{(i)} - f(\boldsymbol{x}^{(i)}))^2 .$$

Matrix notation

Adopting matrix notation

$$X = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_p^{(1)} \\ 1 & x_1^{(2)} & \dots & x_p^{(2)} \\ \vdots & \vdots & \dots & \vdots \\ 1 & x_1^{(n)} & \dots & x_p^{(n)} \end{bmatrix}, \qquad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix},$$

X is called the design matrix and $\boldsymbol{y} \in \mathbb{R}^n$ is the vector of outcomes.

cost function:

$$SSE(\boldsymbol{\beta}) = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|^2 = \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2$$

where $\|\boldsymbol{z}\| = \sqrt{\boldsymbol{z}^T \boldsymbol{z}}$ denotes the Euclidean norm.

Normal equations

The minimization of the SSE cost functional leads to a system of equations that are denoted normal equations:

$$(X^T X)\hat{\boldsymbol{\beta}} = X^T \boldsymbol{y}.$$

The normal equations have a unique solution iif $\det(X^T X) \neq 0$.

The normal equations can be derived from the stationary condition (necessary condition)

$$\nabla SSE(\boldsymbol{\beta}) = 0.$$

Gradient properties

The proof of the normal equations requires two properties of the gradient.

Let $f(\boldsymbol{x})$ be a scalar function, where $\boldsymbol{x} = [x_1, \ldots, x_p]^T$ is a vector.

The gradient of f is defined by

$$\boldsymbol{\nabla}_x f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_p} \end{bmatrix}.$$

Useful properties:

inner product: $\boldsymbol{\nabla}_x(\boldsymbol{b}^T\boldsymbol{x}) = \boldsymbol{b}\,, \qquad \boldsymbol{b} \in \mathbb{R}^p,$

quadratic form: $\boldsymbol{\nabla}_x(\boldsymbol{x}^T M\boldsymbol{x}) = (M + M^T)\boldsymbol{x}\,, \quad M \in \mathbb{R}^{p\times p}.$

Proof

Cost function

$$\begin{aligned} SSE &= \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2 = (\boldsymbol{y} - X\boldsymbol{\beta})^T(\boldsymbol{y} - X\boldsymbol{\beta}) && \text{norm definition} \\ &= \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{y}^T X\boldsymbol{\beta} - \boldsymbol{\beta}^T X^T\boldsymbol{y} + \boldsymbol{\beta}^T X^T X\boldsymbol{\beta} && \text{distributive prop.} \\ &= \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T X\boldsymbol{\beta} + \boldsymbol{\beta}^T X^T X\boldsymbol{\beta} && \text{transpose prop..} \end{aligned}$$

Computing the gradient and making it equal to zero

$$\boldsymbol{\nabla}_\beta SSE = -2X^T\boldsymbol{y} + 2X^T X\boldsymbol{\beta} = 0,$$

we conclude

$$(X^T X)\hat{\boldsymbol{\beta}} = X^T\boldsymbol{y}.$$

The inverse of matrix $X^T X$ may not exist due to two main reasons:

- small amount of data *e.g.*, number of data points smaller than the number of features.
- redundant features (linearly dependent) *e.g.*, duplicated features.

R^2 - Coefficient of Determination

Although the SSE is good to compute the parameters of the linear regression, it does not reflect the quality of the fit in a intuitive manner.

Question: If, for a given linear regression problem with 100 samples, the computed SSE is 25.3 units, is the regressor good or bad ?

To compute a value for the linear regression quality independent of the scale of the variables we can use the r^2 (R squared, or Coefficient of Determination).

$$r^2 = 1 - \frac{SS_{res}}{SS_{tot}}$$

$$SS_{res} = SSE = \sum_i (y_i - \hat{y}_i)^2 \qquad SS_{tot} = var(y) = \sum_i (y_i - \bar{y}_i)^2$$

Interpretation:

- $r^2 = 1 \rightarrow$ linear regression perfectly fits the data
- $r^2 = 0 \rightarrow$ linear regression always predicts the average of the data
- $r^2 < 0 \rightarrow$ linear regression is worse than just predicting the average

Summary of linear regression

Model training

normal equations: $(X^T X)\hat{\boldsymbol{\beta}} = X^T \boldsymbol{y}$,

parameter estimates: $\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \boldsymbol{y}$.

Prediction

new data: $f(\boldsymbol{x}_0) = [1 \;\; \boldsymbol{x}_0^T]\hat{\boldsymbol{\beta}}$,

training data: $\hat{\boldsymbol{y}} = X\hat{\boldsymbol{\beta}} = X(X^T X)^{-1} X^T \boldsymbol{y}$.

Attention: the inverse of matrix $X^T X$ does not exist if $\det(X^T X) = 0$.

Example

Consider the standard linear (affine) model

$$\hat{\boldsymbol{y}} = \beta_0 + \boldsymbol{x}_0^T \boldsymbol{\beta},$$

where $\boldsymbol{x}_0 \in \mathbb{R}^n$ is a feature vector and $\beta_0 \in \mathbb{R}$ is the offset and $\boldsymbol{\beta} \in \mathbb{R}^p$ a vector of coefficients.

Consider a second linear model relating the feature vector after removing the mean value $(\boldsymbol{x}_0 - \bar{\boldsymbol{x}})$ with the centered outcome $(y - \bar{y})$

$$\hat{y}' - \bar{y} = (\boldsymbol{x}_0 - \bar{\boldsymbol{x}})^T \beta',$$

where $\boldsymbol{\beta}' \in \mathbb{R}^p$ is a vector of coefficients.

What is the relationship between the parameters of both predictors?

Using the method of undetermined coefficients, we conclude

$$\begin{aligned} \text{constant terms:} \quad & \beta_0 = \bar{y} - \bar{\boldsymbol{x}}_0^T \beta' \\ \text{linear terms:} \quad & \boldsymbol{\beta}' = \boldsymbol{\beta} \end{aligned}$$

Gauss Markov Theorem

Let $\boldsymbol{y} = X\boldsymbol{\beta} + \boldsymbol{w}$ where $\boldsymbol{\beta}$ is unknown, X is known (deterministic) and $\boldsymbol{w}$ is a realization of a random vector of zero mean and covariance $\sigma^2\boldsymbol{I}$. Then,

- The least squares estimate of $\boldsymbol{\beta}$

$$\hat{\boldsymbol{\beta}} = (X^T X)^{-1} X^T \boldsymbol{y},$$

  is unbiased with covariance matrix $\sigma^2(X^T X)^{-1}$.
- If $\tilde{\boldsymbol{\beta}} = P\boldsymbol{y}$ is another unbiased estimator of $\boldsymbol{\beta}$, it has a covariance matrix that is equal or larger then $\text{Cov}\{\hat{\boldsymbol{\beta}}\}$ [1].

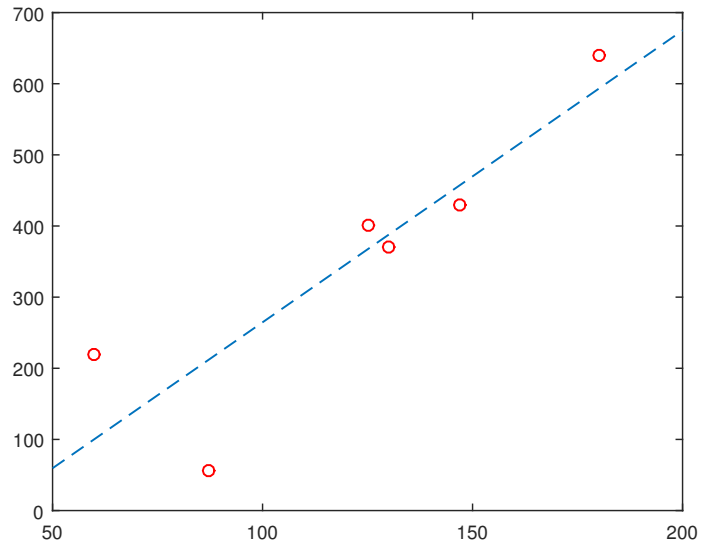
Proof: Hastie et al., *Elements of Statistical Learning, Springer, 2009.*

[1] the inequality $A \geq B$ means that $A - B$ is a semi-definite positive matrix

Example

Figure shows the least squares fit of a linear model (straight line) to the flat data.

area	price
130	370
60	220
87	57
125	400
147	430
180	640



Polynomial model

The linear model is often very rigid, especially when the number of features is small.

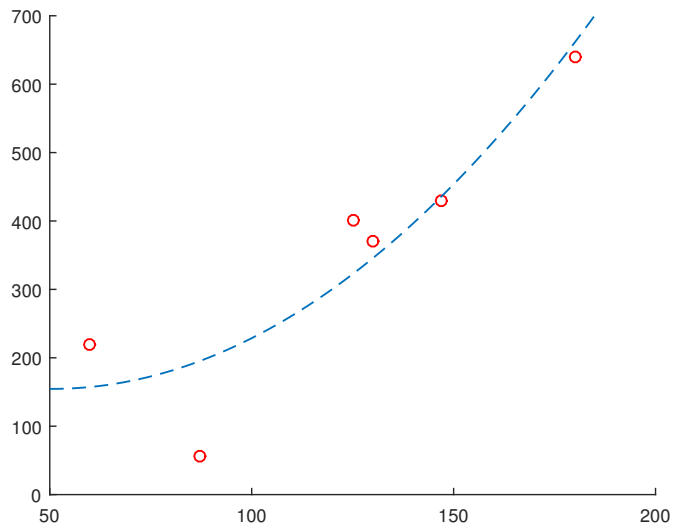
An alternative is the polynomial model (x scalar)

$$f(x) = \beta_0 + \beta_1 x + \cdots + \beta_p x^p.$$

This can be considered as linear model whose features are the powers of x (scalar).

The model is non-linear in x but linear in the parameters β_i.

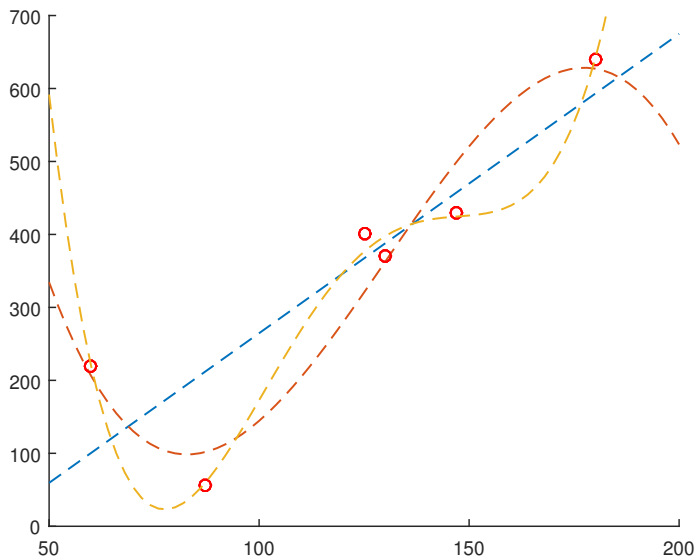
Therefore, the β coefficients can be obtained by the least squares method described before, leading to a linear set of equations.



Model order

The polynomial model becomes numerically unstable when we increase the order of the polynomial.

Figure shows polynomial fits for $p = 1, 3, 4$.



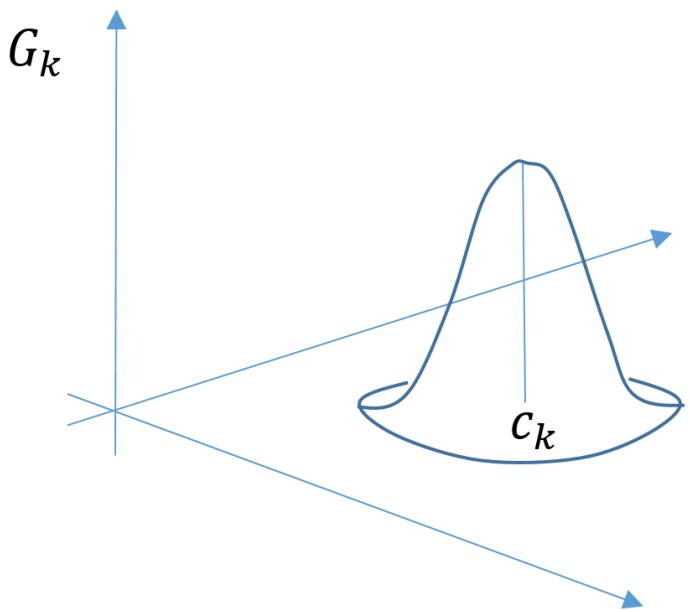
How should we choose the best order? is the SSE a good criterion?

Radial basis functions: model

This model is based on a sum of radial basis functions (Gaussian local functions) defined by

$$G_k(\boldsymbol{x}) = e^{-\frac{1}{2\sigma^2}\|\boldsymbol{x}-\boldsymbol{c}^{(k)}\|^2} \qquad k = 1, \ldots, p\,,$$

where $\boldsymbol{c}^{(k)} \in \mathbb{R}^d$ is a center vector (aka centroid) to be computed from the data.



The Radial basis function model approximates the outcome y by a weighted sum of local basis functions

$$f(\boldsymbol{x}) = \sum_{k=1}^{p} w_k G_k(\boldsymbol{x}).$$

Radial basis functions: training

The model is estimated as follows. First we estimate the p centroids, $\boldsymbol{c}_k$. This is not done by least squares. The centroids $\boldsymbol{c}_k$ are obtained using a clustering algorithm such as k-means (to be presented later in this course).

Then, the coefficients $\boldsymbol{w} = [w_1, \ldots, w_p]^T$ are estimated by least squares

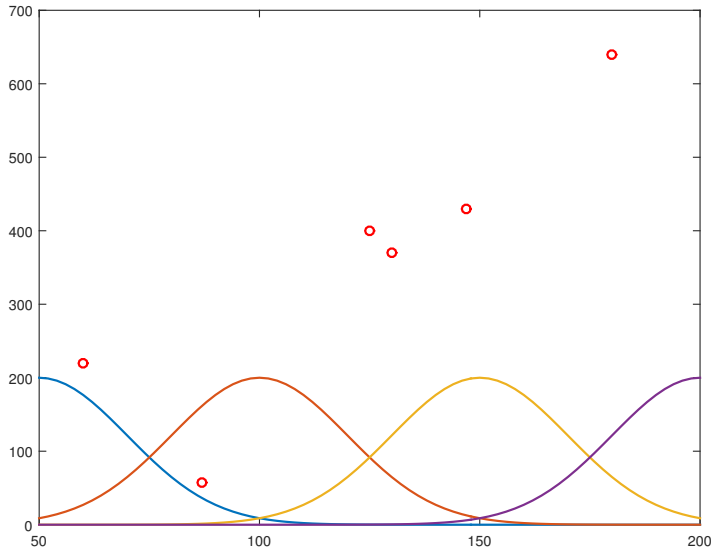
$$\boldsymbol{w} = (X^T X)^{-1} X^T \boldsymbol{y},$$

where X is given by

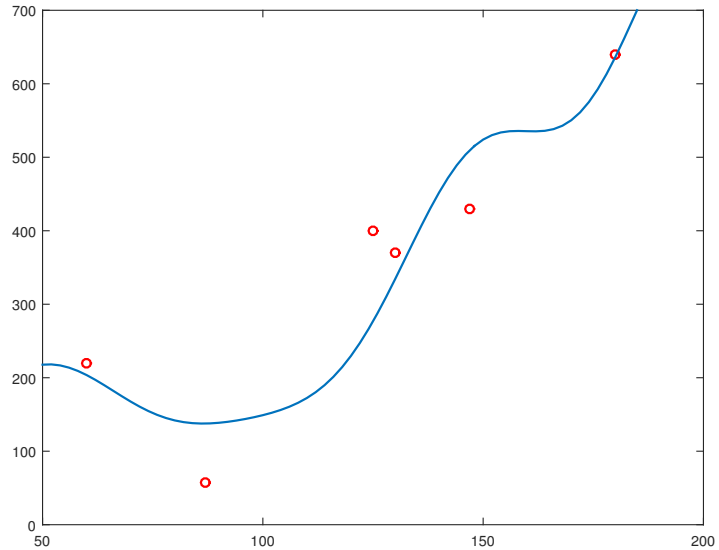
$$X = \begin{bmatrix} G_1(\boldsymbol{x}^{(1)}) & G_2(\boldsymbol{x}^{(1)}) & \ldots & G_p(\boldsymbol{x}^{(1)}) \\ \vdots & \vdots & \ldots & \vdots \\ G_1(\boldsymbol{x}^{(n)}) & G_2(\boldsymbol{x}^{(n)}) & \ldots & G_p(\boldsymbol{x}^{(n)}) \end{bmatrix},$$

σ^2 is an hyperparameter chosen by the user or estimated from the data.

Example - radial basis functions



radial basis functions



regression results

(centroids are equally spaced instead of being computed from the data)

Regression with multiple outputs

Attention: we use a different notation in this and in the next slides.

Suppose that we have multiple output vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K \in \mathbb{R}^n$, each of them approximated by a linear model with the same features but different coefficients $\boldsymbol{\beta}_k \in \mathbb{R}^{p+1}$

$$\boldsymbol{y}_k = X\boldsymbol{\beta}_k + \boldsymbol{w}_k \ , \quad k = 1, \ldots, K.$$

The SSE for the multiple outputs is the sum of the SSE for each output.

$$SSE = \sum_{k=1}^{K} SSE_k(\boldsymbol{\beta}_k).$$

Each regression problem can be independently solved *i.e.*, $\hat{\boldsymbol{\beta}}_k$ can be obtained by minimizing $SSE_k(\boldsymbol{\beta}_k)$

$$(X^T X)\hat{\boldsymbol{\beta}}_k = X^T \boldsymbol{y}_k.$$

Regression with multiple outputs

The problem can also be formulated using matrix notation (more difficult)

$$Y = XB + W,$$

where $Y = [\boldsymbol{y}_1 \ldots \boldsymbol{y}_K] \in \mathbb{R}^{n \times K}$, $B = [\boldsymbol{\beta}_1 \ldots \boldsymbol{\beta}_K] \in \mathbb{R}^{p \times K}$, $W = [\boldsymbol{w}_1 \ldots \boldsymbol{w}_K] \in \mathbb{R}^{n \times K}$.

The minimization of the sum of squared errors criterion

$$SSE(B) = tr\left\{(Y - XB)^T (Y - X\beta)\right\},$$

leads to

$$\hat{B} = (X^T X)^{-1} X^T Y.$$

This is equivalent to independently solving each of the K least squares problem sharing the same design matrix X.

$tr\{\}$ denotes the trace of a matrix (sum of the diagonal elements).

Exercises

1. Consider a training set $\mathcal{T} = \left\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\right\}$. Write the normal equations for a least squares fit of a second order polynomial model (x scalar) to the training data.

2. Repeat the previous problem, assuming 2D features $\boldsymbol{x} \in \mathbb{R}^2$.

3. What happens to the predicted outcome $\hat{y}$ estimated by least squares (without offset), if the observed features are scaled *i.e.*, $\boldsymbol{x}'^{(i)} = D\boldsymbol{x}^{(i)},\ i = 1, \ldots, n$, where D is a diagonal matrix.

Machine Learning

Linear Regression

Regularization

Evaluation & Generalization

Optimization

Neural networks

Data classification

Linear classifiers

Support vector machines

Decision Trees and Random Forest

Motivation

A linear model can be estimated by minimizing the least squares criterion in the training set

$$\|\boldsymbol{y} - X\boldsymbol{\beta}\|^2,$$

where

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}, \qquad X = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & \dots & x_p^{(1)} \\ x_1^{(2)} & x_2^{(2)} & \dots & x_p^{(2)} \\ \vdots & \vdots & \dots & \vdots \\ x_1^{(n)} & x_2^{(n)} & \dots & x_p^{(n)} \end{bmatrix}, \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}.$$

We removed the mean $\bar{\boldsymbol{x}}, \bar{y}$ from the data in order to make $\beta_0 = 0$. The vector $\boldsymbol{\beta}$ does not include β_0 and X does not include a column of ones.

Drawbacks

LS approach leads to the normal equations

$$(X^T X)\hat{\boldsymbol{\beta}} = X^T \boldsymbol{y}.$$

If $(X^T X)$ is a singular matrix, the least squares estimate is not unique. Infinite solutions are available.

Example

Suppose we wish to estimate the model from a single example.

$$\hat{y} = x_1\beta_1 + x_2\beta_2$$

x_1	x_2	y
1	1	3

Example (cont.)

We obtain 2 parameters and 1 constraint, leading to infinite solutions

$$\beta_1 + \beta_2 = 3.$$

How can we solve this difficulty?

By adding a new constraint: minimizing the squared norm of the coefficients

$$\beta_1^2 + \beta_2^2$$

This is a measure of "model complexity".

Ridge regression

An alternative criterion is ridge regression

$$\hat{\boldsymbol{\beta}}_{\text{ridge}} = \arg\min_{\beta} \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|^2,$$

where $\|.\|$ denotes the Euclidean norm. The new term $\|\boldsymbol{\beta}\|^2$ penalizes the use of large coefficients and it is denoted a regularization term. This criterion aims to represent the data, keeping the coefficients small.

λ represents the trade-off between both objectives.

Furthermore, we assume that training data $X, \boldsymbol{y}$ have zero mean. The coefficient β_0 is not usually included in the regularization term.

Ridge regression

The ridge problem can be rewritten as a constrained optimization problem

$$\hat{\boldsymbol{\beta}}_{\text{ridge}} = \arg\min_{\beta} \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2, \qquad s.t., \qquad \|\boldsymbol{\beta}\|^2 \leq \tau.$$

There is a correspondence between the values of τ and λ. The exact map cannot be obtained.

The ridge regression can be solved by computing the gradient vector and making it equal to zero, leading to

$$\hat{\beta}_{\text{ridge}} = (X^T X + \lambda I)^{-1} X^T y \ .$$

Matrix $(X^T X + \lambda I)$, with $\lambda > 0$, is always non-singular, even if $(X^T X)$ is singular.

Proof

Cost function

$$\begin{aligned} E_{\text{Ridge}} &= \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|^2 \\ &= (\boldsymbol{y} - X\boldsymbol{\beta})^T(\boldsymbol{y} - X\boldsymbol{\beta}) + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta} \\ &= \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T X\boldsymbol{\beta} + \boldsymbol{\beta}^T X^T X\boldsymbol{\beta} + \lambda\boldsymbol{\beta}^T\boldsymbol{\beta} \\ &= \boldsymbol{y}^T\boldsymbol{y} - 2\boldsymbol{y}^T X\boldsymbol{\beta} + \boldsymbol{\beta}(X^T X + \lambda I)\boldsymbol{\beta}. \end{aligned}$$

Computing the gradient and making it equal to zero

$$\nabla_\beta E_{\text{Ridge}} = -2X^T\boldsymbol{y} + 2(X^T X + \lambda I)\boldsymbol{\beta} = 0,$$

we conclude

$$(X^T X + \lambda I)\hat{\boldsymbol{\beta}}_{\text{ridge}} = X^T\boldsymbol{y}.$$

Exercise

Find the relationship between the eigenvector and eigenvalues of the LS matrix $(X^T X)$ and ridge matrix $(X^T X + \lambda I)$.

Try to solve it by yourself.

Tentative solution

$$\text{LS:} \quad (X^T X)\boldsymbol{v}^{\text{ls}} = \lambda^{\text{ls}}\boldsymbol{v}^{\text{ls}} \Rightarrow (X^T X - \lambda^{ls} I)\boldsymbol{v}^{\text{ls}} = 0$$

$$\text{Ridge:} \quad (X^T X + \lambda I)\boldsymbol{v}^{\text{ridge}} = \lambda^{\text{ridge}}\boldsymbol{v}^{\text{ridge}}$$

$$(X^T X + \lambda I - \lambda^{\text{ridge}} I)\boldsymbol{v}^{\text{ridge}} = 0$$

Comparing,

$$\lambda^{\text{ridge}} = \lambda^{\text{ls}} + \lambda$$

$$\boldsymbol{v}^{\text{ridge}} = \boldsymbol{v}^{\text{ls}}$$

Conclusion:
The eigenvectors are equal and the eigenvalues are shifted by λ. If $\lambda > 0$ the eigenvalues of the ridge matrix are positive and the matrix is non singular.

The Lasso

Another alternative is Lasso.

Lasso regression aims to minimize the sum of squared errors (with $\beta_0 = 0$)

$$\min_{\beta} \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2,$$

with a different constraint on the coefficients that penalizes large errors less.

$$\sum_{j=1}^{p} |\beta_j| \leq \tau.$$

This constraint can be expressed in terms of the ℓ_1 norm: $|\boldsymbol{\beta}\|_1 \leq \tau$. Since we are dealing with two norms, the ℓ_2 norm (Euclidean) will be denoted by $\|.\|_2$ and the ℓ_1 norm by $\|.\|_1$.

The Lasso

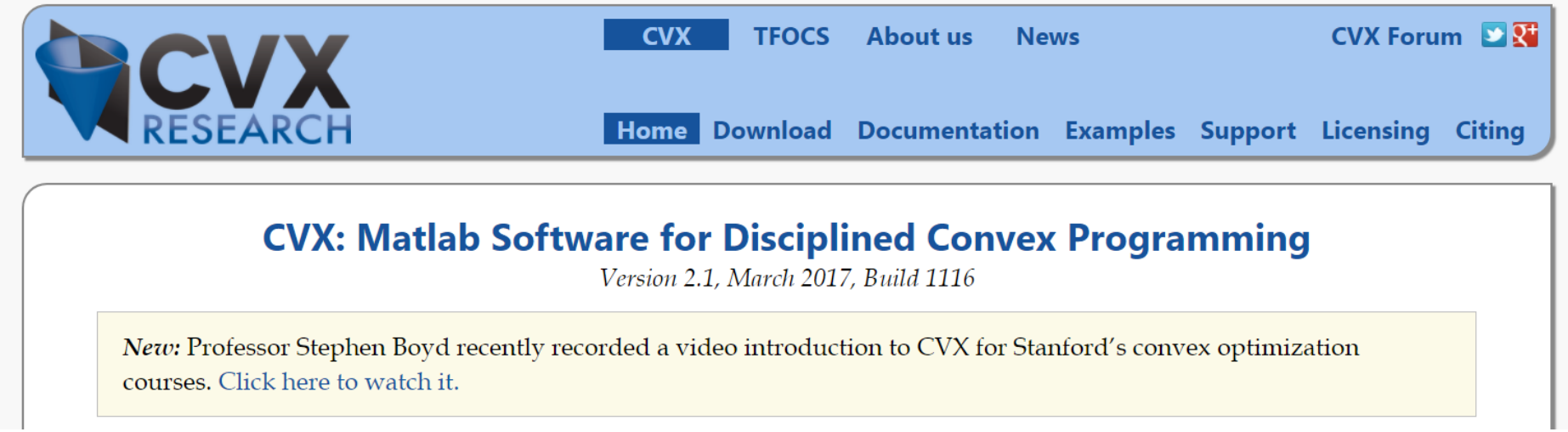
The Lagrangian formulation is given by

$$\boldsymbol{\beta}_{\text{lasso}} = \arg\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - X\boldsymbol{\beta}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1 \ ,$$

where the last term can be interpreted as a regularization term.

This optimization problem cannot be solved by a linear system of equations as before. In fact we have to resort to convex optimization methods to numerically solve this problem.

CVX software package



$$\begin{array}{ll} \text{minimize} & \|Ax - b\|_2 \\ \text{subject to} & Cx = d \\ & \|x\|_\infty \leq e \end{array}$$

```
m = 20; n = 10; p = 4;
A = randn(m,n); b = randn(m,1);
C = randn(p,n); d = randn(p,1); e = rand;
cvx_begin
    variable x(n)
    minimize( norm( A * x - b, 2 ) )
    subject to
        C * x == d
        norm( x, Inf ) <= e
cvx_end
```

Sparse solutions

We often wish to find sparse solutions for $\boldsymbol{\beta}$ (with some zero coefficients) which corresponds to selecting only a subset of features (feature selection).

The problem can be formulated as

$$\hat{\boldsymbol{\beta}}_{\text{sparse}} = \arg\min_{\beta} \|\boldsymbol{y} - X\boldsymbol{\beta}\|_2^2 + \lambda \text{ number of non-zero coefficients},$$

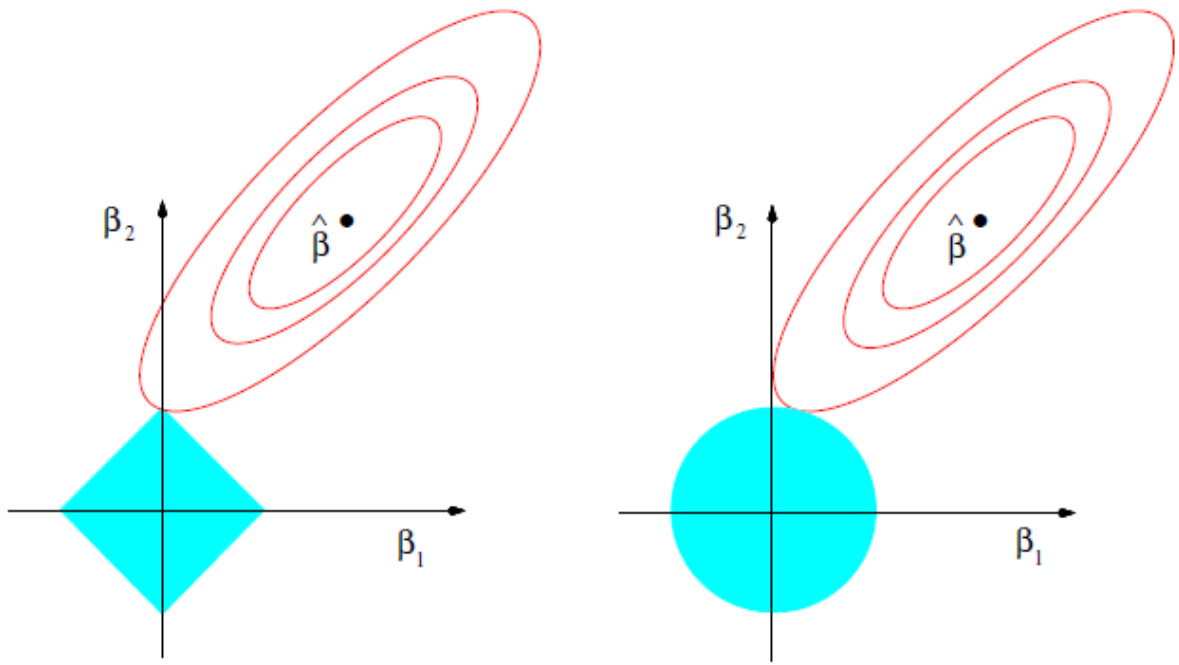
where the number of non-zero coefficients is often called "ℓ_0 norm", $\|.\|_0$, although it does not verify the axioms of a norm.

Regularization with the ℓ_0 norm is difficult to solve numerically. However, the solution is often well approximated by the lasso regression which also leads to sparse solutions in many problems: when a feature is not important the corresponding coefficient is made equal to zero.

Feature Selection

The lasso estimate β_{lasso} is often a sparse vector of coefficients where less important features receive a zero coefficient.

This can be interpreted as a feature selection operation. Since unimportant features are removed, the other ones are better estimated.

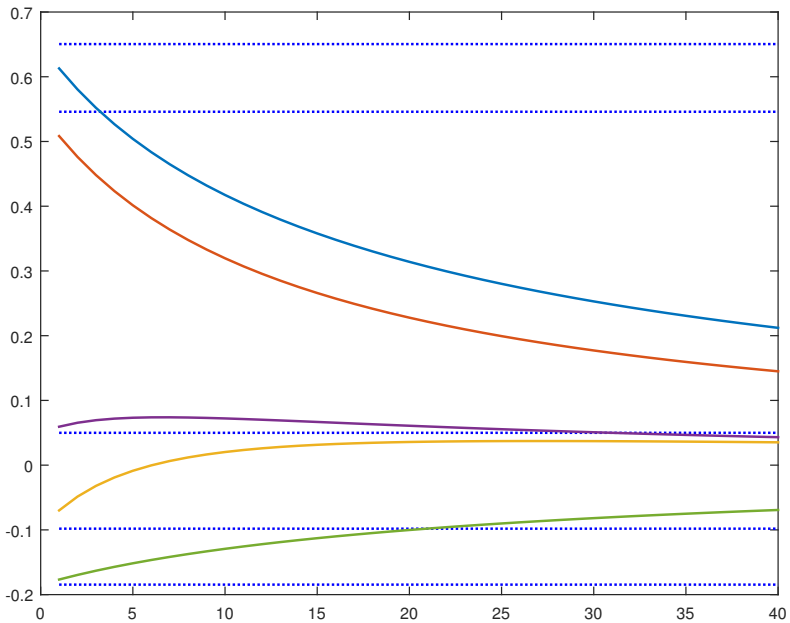


Example: lasso vs ridge regression

Regression problem with a subset of features uncorrelated with the outcome. The training data was generated by $y = \boldsymbol{x}^T\boldsymbol{\beta} + w$ where $w \sim N(0, \sigma^2)$, $x \sim N(0, I)$ and $\boldsymbol{\beta} = [1\ 0.5\ 0\ 0\ 0]$.

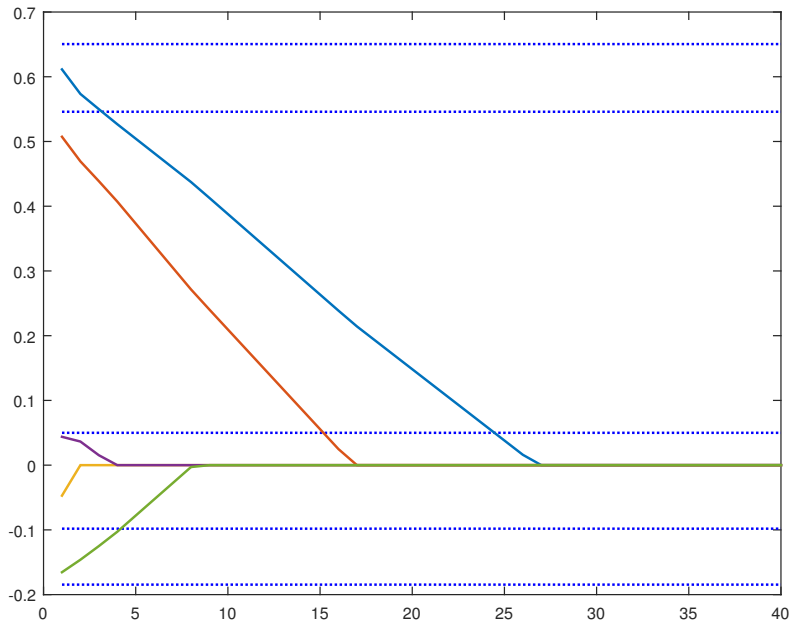
Estimates obtained by least squares (horizontal lines), ridge regression and lasso as a function of λ.

ridge regression



λ

lasso



λ

Non-centered data

How should we proceed if the training data $\mathcal{T} = \left\{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(n)}, y^{(n)})\right\}$ are not centered?

1. pre-processing: $\boldsymbol{x}'^{(i)} = \boldsymbol{x}^{(i)} - \bar{\boldsymbol{x}}, \quad y'^{(i)} = y^{(i)} - \bar{y}$ ($\bar{\boldsymbol{x}}, \bar{y}$ average values computed in the training set);

2. estimate linear model without intercept: estimate model $y' = \boldsymbol{x}'^T \boldsymbol{\beta}'$, with $\boldsymbol{\beta}' \in \mathbb{R}^p$, using the pre-processed data $\mathcal{T} = \left\{(\boldsymbol{x}'^{(1)}, y'^{(1)}), \ldots, (\boldsymbol{x}'^{(n)}, y'^{(n)})\right\}$ and regularization;

3. invert pre-processing: $\hat{\boldsymbol{\beta}} = [\hat{\beta}_0 \ \hat{\boldsymbol{\beta}}'^T]^T$ where $\hat{\beta}_0 = \bar{y} - \bar{\boldsymbol{x}}^T \hat{\boldsymbol{\beta}}'$;

Matlab commands *ridge, lasso* perform all the tree steps.

Exercises

1. Does a linear regressor, estimated by the least squares method, depend on the scale of the features? what happens if we we use ridge regression instead?

2. Suppose we wish to predict a variable $y \in \mathbb{R}$, using a single feature $x \in \mathbb{R}$ (without intercept). Given a training set $\mathcal{T} = \left\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\right\}$ and assuming that the feature x is normalized

$$\frac{1}{n}\sum_{i=1}^{n}(x^{(i)})^2 = 1\,,$$

   Find the ridge and lasso coefficients, $\hat{\beta}^{ridge}$, $\hat{\beta}^{lasso}$, as a function of the least squares coefficient, $\hat{\beta}^{ls}$, and plot them.

Try to solve by yourself

Tentative solution

1. Suppose we multiply the features by a scale factor $X' = sX$, where s is the scale factor. Then, the vector of coefficients becomes

$$\beta'^{ls} = (X'^T X')^{-1} X'^T y = (s^2 X^T X)^{-1} s X^T y = s^{-2} (X^T X)^{-1} s X^T y$$

$$\beta'^{ls} = s^{-1} \beta^{ls}.$$

LS predictor: $\hat{y}' = x'^T \beta'^{ls} = s x^T s^{-1} \beta^{ls} = x^T \beta^{ls} = \hat{y}$ is invariant under scaling.

The ridge coefficient are the solution of $(X'^T X' + \lambda I)\beta'^{ridge} = X' y$.

Matrix $(X'^T X' + \lambda I)$ has two terms: one that depends on the scale and the other that does not. Therefore, the ridge predictor is not invariant to scale.

Tentative solution

2.

$$LS: \quad \hat{\beta}^{ls} = (X^T X)^{-1} X' y = \frac{1}{n} X' y$$

$$Ridge: \quad \hat{\beta}^{ridge} = (X^T X + \lambda I)^{-1} X' y = \frac{1}{n+\lambda} X' y$$

$$\hat{\beta}^{ridge} = \frac{n}{n+\lambda} \hat{\beta}^{ls}$$

$$Lasso: \quad \min_\beta \|y - X\beta\|_2^2 + \lambda \|\beta\|_1$$

$$\min_\beta (y^T y - 2 y^T X \beta + \beta^2 X^T X) + \lambda |\beta|$$

$$Hypothesis: \hat{\beta} > 0 \quad \frac{d(\ldots)}{d\beta} = 0 \Rightarrow -2n\hat{\beta}^{ls} + 2n\hat{\beta}^{lasso} + \lambda = 0$$

$$\hat{\beta}^{lasso} = \hat{\beta}^{ls} - \frac{\lambda}{2n}$$

the same should be repeated for $\hat{\beta}^{lasso} < 0$.

These relationships should be graphically represented.

Table of contents

Machine Learning

Linear Regression

Regularization

Evaluation & Generalization

Optimization

Neural networks

Data classification

Linear classifiers

Support vector machines

Decision Trees and Random Forest

Supervised learning

We wish to predict an outcome y, given a vector of features $x \in \mathbb{R}^p$.

Predictor (model):

$$\hat{y} = f(\boldsymbol{x}, \theta).$$

The parameters of the model, θ, are estimated from a training set

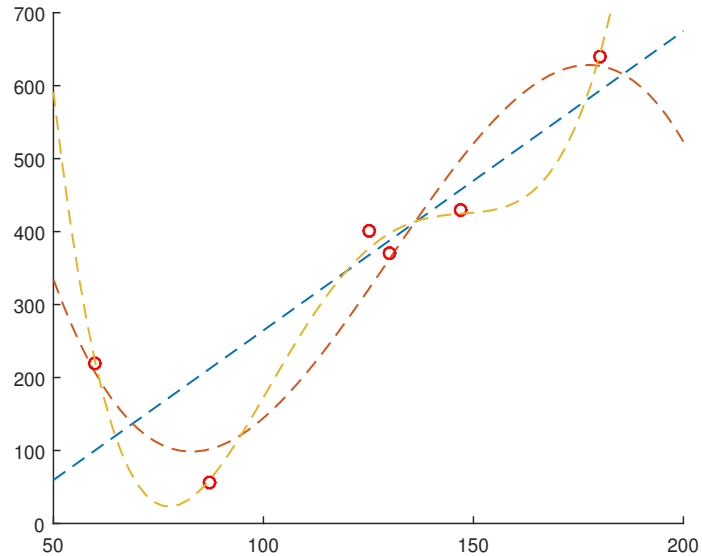
$$\mathcal{T} = \left\{ (\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(n)}, y^{(n)}) \right\}.$$

But, learned systems are not perfect. The output of a learned system is not always the desired output.

Learning systems need to be evaluated.

Example: regressor

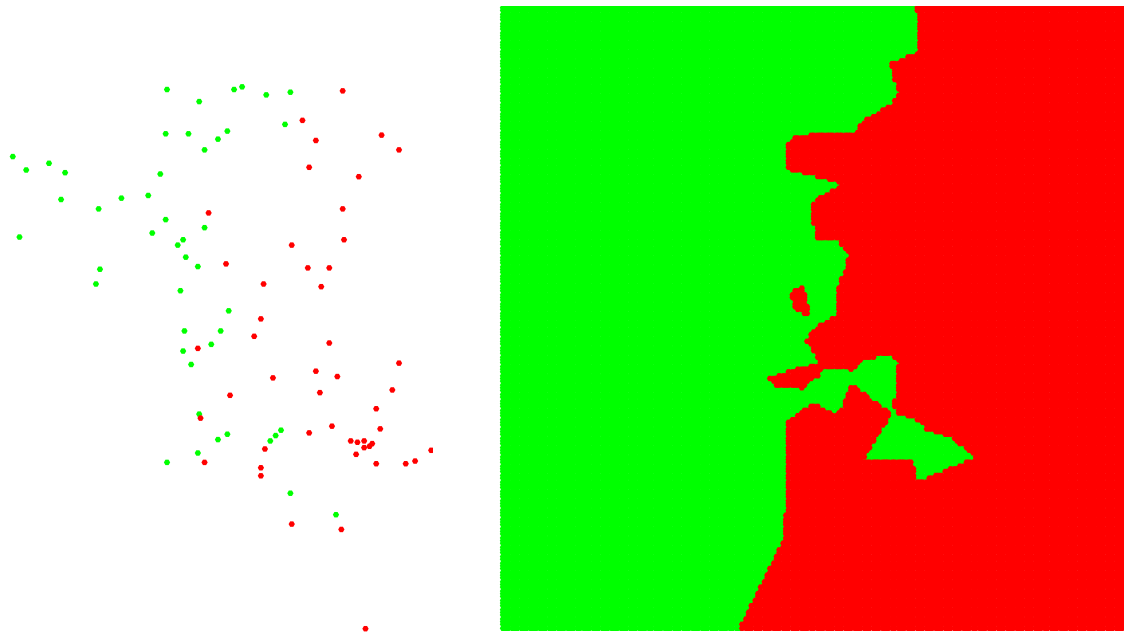
How do we measure the performance of a regressor?



Polynomial fits (order 1, 3, and 4).

Example: classifier

How do we measure the performance of a classifier?



Training set (left) and predicted classes (right) using the k nearest neighbor method.

Loss function

If the desired output, y, is different from the predicted outcome, $\hat{y} = f(\boldsymbol{x})$, we define a loss $L(y, \hat{y})$, *e.g.*,

Regression

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

Classification

$$L(y, \hat{y}) = \begin{cases} 0 & y = \hat{y} \\ 1 & \text{otherwise} \end{cases}$$

or

$$L(y = i, \hat{y} = j) = L_{ij}$$

diagonal terms (no error) equal to zero.

In the classification problem, the second loss function is more flexible since it may assign different penalties to different kinds of errors.

Risk

If x, y are realizations of two random variables, it makes sense to define the expected (average) value of the loss, also known as risk.

$$\mathcal{R} = E\left\{L(y, \hat{y}(\boldsymbol{x}))\right\}.$$

In the case of regression, the risk would be

$$\mathcal{R} = \int\int L(y, \hat{y}(\boldsymbol{x}))p(\boldsymbol{x}, y)\, dx\, dy.$$

This requires the joint distribution of input and output $p(\boldsymbol{x}, y)$ which is usually unknown.

Risk

In the case of classification problems, the risk would be

$$\mathcal{R} = \sum_y \sum_x L(y, \hat{y}(\boldsymbol{x}))P(\boldsymbol{x}, y) = \sum_y \sum_{\hat{y}} L(y, \hat{y})P(y, \hat{y})$$

This requires the joint distribution of true and predicted class $P(y, \hat{y})$ which is usually unknown.

Empirical risk

Since the risk cannot be computed in most problems, we can replace the expected value by an average of the loss computed with the training data,

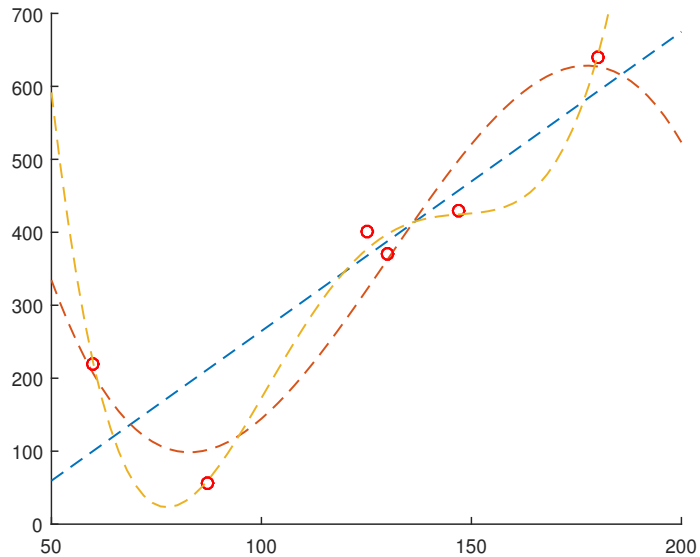
$$\mathcal{R}_e = \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, f(\boldsymbol{x}^{(i)})) \ .$$

This is called the empirical risk.

The empirical risk is often used to train the predictor. However, is it a good criterion to evaluate the system?

Example: polynomial fit

Polynomial fits of order 1, 3, 4: which model is the best?



The empirical risk of the forth order polynomial is the smallest. But is this the best model?

The model order is often considered as an hyperparameter.

Generalization

We want to measure the performance of the system with new data. This property is known as generalization.

To evaluate the generalization of a system, we should consider an independent data set

$$\mathcal{T}' = \left\{ (\boldsymbol{x}'^{(i)}, y'^{(i)}), i = 1, \ldots, n' \right\}$$

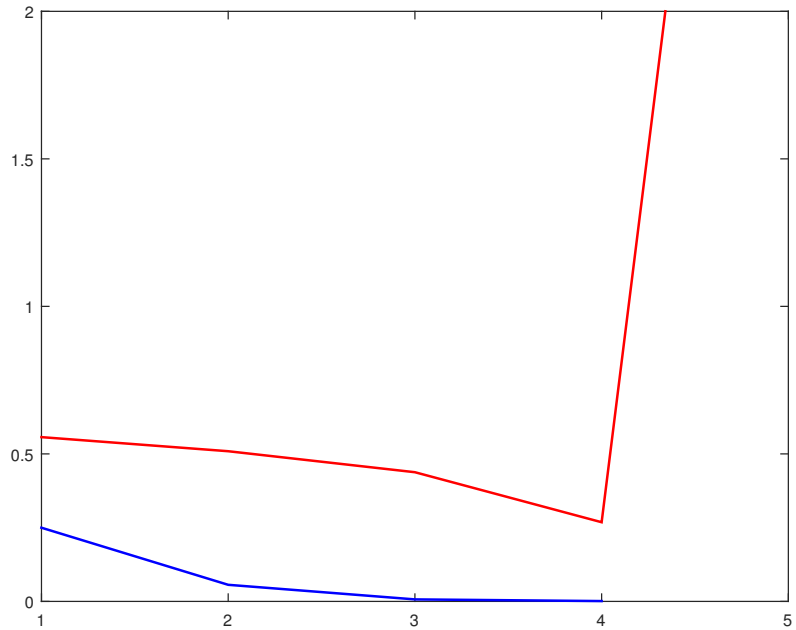
and evaluate the model in it.

$$\mathcal{R}'_e = \frac{1}{n'} \sum_{i=1}^{n'} L(y'^{(i)}, f(\boldsymbol{x}'^{(i)}))$$

Important questions:

- is $\mathcal{R}_e$, (computed in the training set) a good estimate of $\mathcal{R}'_e$ (computed in an independent set)?
- can $\mathcal{R}_e$ or $\mathcal{R}'_e$ be used to choose the model hyperparameters (e.g., polynomial order)?

Evaluation of polynomial order



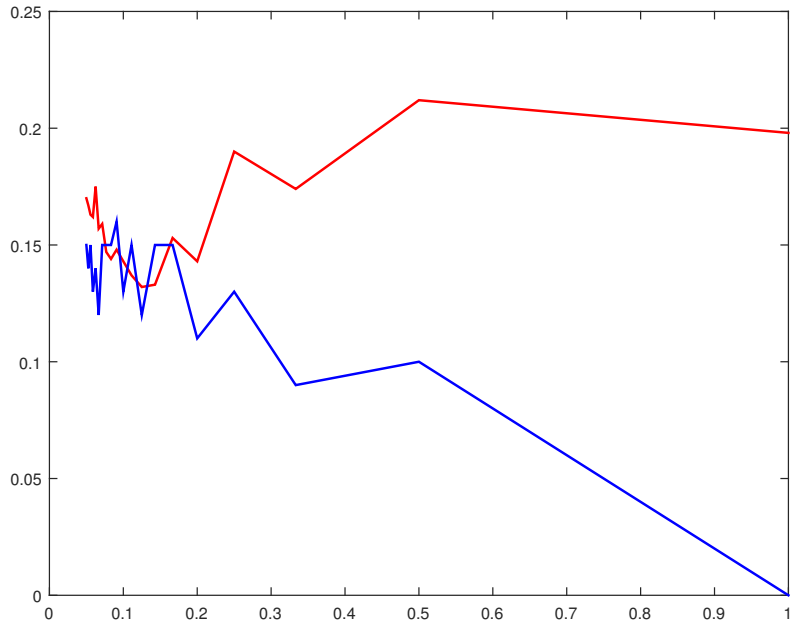
Average loss in the training set (blue) and in an independent set (red), as a function of polynomial degree (hyperparameter).

Conclusions:

- The evaluation in the training set is too optimistic. An independent data set is mandatory to obtain a reliable evaluation.

- The use of an independent data set allows the choice of model hyperparameters (polynomial degree).

Evaluation of k Nearest neighbor



Percentage of classification error in the training set (blue) and in an independent set (red), as a function of $1/k$

The conclusions are the same as before.

Overfitting

When there is a large difference between the evaluation of the model in the training set and in an independent set, it means that the model is too specialized in representing the training data and performs much worse with new data.

This fenomena is known as overfitting.

Summary (until now)

To estimate a model and evaluate it it is recommended the use of 2 independent data sets: training set, and a test set.

To estimate a model, select the hyperparameters and evaluate the selected model it is recommended the use of 3 independent data sets: training set, validation set, and a test set

More sophisicated techniques are available (cross validation, leave-one-out)

Model training and testing (without hyperparameters)

In most learning problems, we need to train and evaluate the model. These operations should be done using two independent data sets known as training set and test set.

This can be written in pseudocode using the functions $f = \text{train}(\mathcal{T})$, $P = \text{perform}(f, \mathcal{T}')$

Data: training set $\mathcal{T}$ and test set $\mathcal{T}'$.

$f = \text{train}(\mathcal{T})$;
$P = \text{perform}(f, \mathcal{T}')$;

Algorithm 1: Train and test of a model

Hyperparameter selection

If we need to choose the value of hyperparameters ξ (*e.g.*, polinomial degree), this should be done using a third independent set known as validation set.
This can be written in pseudocode using the functions $f = \text{train}(\mathcal{T}, \xi)$, $P = \text{perform}(f, \mathcal{T}')$

Data: training set $\mathcal{T}$, validation set $\mathcal{T}_v$ and test set $\mathcal{T}'$.
Result: Select hyperparameters ξ and evaluate model.

for *all values of* ξ **do**
  $f = \text{train}(\mathcal{T}, \xi)$;
  $P(\xi) = \text{perform}(f, \mathcal{T}_v)$;
end
$\hat{\xi} = \arg\min_\xi P(\xi)$;
$f = \text{train}(\mathcal{T} \cup \mathcal{T}_v, \hat{\xi})$;
$P = \text{perform}(f, \mathcal{T}')$;

Algorithm 2: Training, optimization and testing of a model

This method requires a lot of data.

Cross-validation

Cross validation is a very useful technique when we do not have a large amount of data. The data set is split into k folds $\mathcal{T}_k$ (subsets with the same number of examples). One fold is used for testing and the others for training. After, the test fold rotates K times.

Data: k folds $\mathcal{T}_k$.
for k=*1, . . . , K* **do**
  $f = \text{train}(\mathcal{T} \setminus \mathcal{T}_k)$;
  $P_k = \text{perform}(f, \mathcal{T}_k)$;
end
$P = \bar{P}_k$

Algorithm 3: Cross validation without hyperparameters. The bar denotes average for all the folders.

The final score is a combination of the evaluation of K models. The method does not produce a final classifier/regressor.

Question: if we need one, what should we do?.

Cross-validation with hyperparameters (nested)

Cross validation can be extended to account for the estimation of hyperparameters. The data is again divided into K folds and two of them will be used for validation and test.

Data: k folds $\mathcal{T}_k$.
for $i = 1, \ldots, K$ **do**
  for *all values of* ξ **do**
    for $j \neq i$ **do**
      $f = \text{train}(\mathcal{T} \setminus (\mathcal{T}_i \cup \mathcal{T}_j), \xi)$;
      $P(\xi)_j = \text{perform}(f, \mathcal{T}_j)$;
    end
    $P(\xi) = P(\bar{\xi})_j$
  end
  $\hat{\xi}_i = \arg\min_\xi P(\xi)$;
  $f = \text{train}(\mathcal{T} \setminus \mathcal{T}_i, \hat{\xi}_i)$;
  $P_i = \text{perform}(f, \mathcal{T}_i)$;
end
$P = \bar{P}_i$

Algorithm 4: Cross validation with hyperparameters (nested).

Table of contents

Machine Learning

Linear Regression

Regularization

Evaluation & Generalization

Optimization

Neural networks

Data classification

Linear classifiers

Support vector machines

Decision Trees and Random Forest

Optimization

Linear regression boils down to minimizing

$$\text{SSE} = \|\boldsymbol{y} - X\boldsymbol{\beta}\|^2,$$

that can be analytically solved.

Most regression / classification problems involve the solution of an optimization problem,

$$\hat{\boldsymbol{\theta}} = \arg\min_{\theta} J(\boldsymbol{\theta}) \ ,$$

where $J : \mathbb{R}^p \rightarrow \mathbb{R}$ is the cost function and $\boldsymbol{\theta} \in \mathbb{R}^p$ denotes the model parameters.

In most cases, this cannot be analytically solved and we must rely on numerical (iterative) optimization algorithms that deliver approximate values for the parameters.

Optimization methods

Optimization methods may use different types of information

1. function values: $J(\boldsymbol{\theta})$,
2. first derivatives (gradient vector): $\nabla_\theta J$,
3. second derivatives (Hessian matrix): H.

Global and Local minima

A function $J : \mathbb{R}^p \to \mathbb{R}$ has a local minimum at $\boldsymbol{\theta}^* \in \mathbb{R}^p$ if there is an $\epsilon > 0$ such that

$$\|\boldsymbol{\theta} - \boldsymbol{\theta}^*\| < \epsilon \quad \Rightarrow \quad J(\boldsymbol{\theta}) \geq J(\boldsymbol{\theta}^*) \,,$$

where

- $J(\boldsymbol{\theta}^*)$ is called a local minimum,
- $\boldsymbol{\theta}^*$ is called a local minimizer.

A function $J : \mathbb{R}^p \to \mathbb{R}$ has a global minimum at $\boldsymbol{\theta}^* \in \mathbb{R}^p$ if, for all $\boldsymbol{\theta} \in \mathbb{R}^p$,

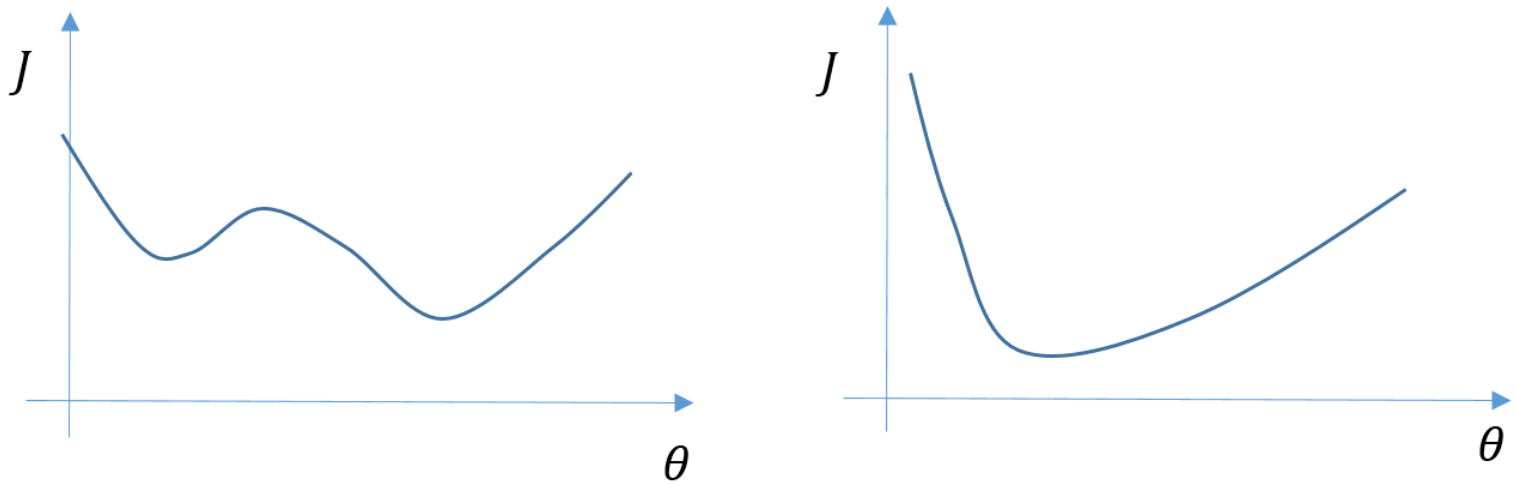
$$J(\boldsymbol{\theta}) \geq J(\boldsymbol{\theta}^*) \,,$$

where

- $J(\boldsymbol{\theta}^*)$ is called a global minimum,
- $\boldsymbol{\theta}^*$ is called a global minimizer.

Global and Local minima

We are interested in the global minimum but most algorithms get trapped in the local minima (left), if they exist and may not converge to the global minimum.



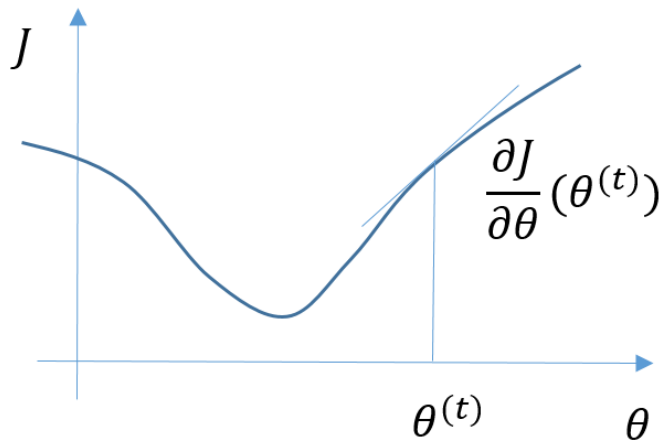
Notice that if the function is convex (right) it has no more than one minimum.

Gradient descent -1D case

If θ is a scalar, the derivative of $J(\theta)$ conveys information about the tilt of the function to be minimized.

If we move a small amount in the opposite direction of the derivative, the function decreases:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \frac{dJ}{d\theta}(\theta^{(t)}),$$



where η controls the displacement of the point $\theta^{(t)}$ and is known as step size or learning step.

The process starts with an initial guess $\theta^{(0)}$.

Gradient descent - vector case

If $\boldsymbol{\theta} \in \mathbb{R}^p$ and $J(\boldsymbol{\theta})$ is a differentiable function in a neighborhood of a point $\boldsymbol{\theta}^{(t)}$, then $J(\boldsymbol{\theta})$ decreases fastest if we move along the opposite direction of the gradient:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_\theta J(\boldsymbol{\theta}^{(t)}).$$

This procedure is repeated until the function stops decreasing, meaning that we are in the vicinity of a local minimum or in a plateau. This algorithm is called gradient descent or stepest descent algorithm. I

Gradient descent - vector case

Another way to motivate the gradient algorithm is based on the first order approximation of the cost function $J(\boldsymbol{\theta}^{(t)} + \boldsymbol{\Delta})$, using a Taylor series expansion

$$J(\boldsymbol{\theta}^{(t)} + \boldsymbol{\Delta}) = J(\boldsymbol{\theta}^{(t)}) + \nabla_\theta J(\boldsymbol{\theta}^{(t)})^T \Delta \ ,$$

valid for a small displacement $\boldsymbol{\Delta}$.

If we make $\boldsymbol{\Delta} = -\eta \nabla_\theta J(\boldsymbol{\theta}^{(t)})^T$, we obtain

$$J(\boldsymbol{\theta}^{(t)} + \boldsymbol{\Delta}) = J(\boldsymbol{\theta}^{(t)}) - \eta \|\nabla_\theta J(\boldsymbol{\theta}^{(t)})\|^2 \ ,$$

which corresponds to a decrease of the cost function.

The choice of η is a difficult aspect of the algorithm. The first order approximation becomes invalid if η is too "large".

Choice of step size η

The choice of step size involves a trade-off and it is often obtained by trial-and-error. If η is too small, the update process becomes very slow.

Figures: isotropic valley + narrow valley

On the contrary, if η is too large the algorithm may skip a local minima or produce an update of $\boldsymbol{\theta}$ that increases the objective function $J(\boldsymbol{\theta})$.

Acceleration techniques can be used to speed up convergence e.g., adaptive step size and momentum technique.

Momentum technique

This method performs a lowpass filtering of the gradient sequence and updates $\boldsymbol{\theta}^{(t+1)}$ using the filtered gradient $\boldsymbol{v}^{(t+1)}$

$$\boldsymbol{v}^{(t+1)} = \alpha \boldsymbol{v}^{(t)} - \eta \nabla_\theta J(\boldsymbol{\theta}^{(t)})$$
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \boldsymbol{v}^{(t+1)}.$$

The parameter α (pole) typically ranges from 0.5 to 0.95.

This technique improves the convergence rate, specially if the cost function $J(\boldsymbol{\theta})$ exhibits deep valleys in which the gradient method is slow.

The objective function J is evaluated in each iteration to check if it decreases as expected. If not, the memory of the moment term is set to zero.

L. Almeida, Multilayer Perceptrons, in Handbook of Neural Computation, 1997.

Nesterov accelerated gradient

This method is similar to the momentum technique but it computes the gradient in a different position. It computes an approximate position for the parameters in the next iteration and computes the gradient there (look ahead).

$$\boldsymbol{v}^{(t+1)} = \alpha \boldsymbol{v}^{(t)} - \eta \nabla_\theta J(\boldsymbol{\theta}^{(t)} + \alpha \boldsymbol{v}^{(t)})$$
$$\boldsymbol{\theta}(t+1) = \boldsymbol{\theta}^{(t+1)} + \boldsymbol{v}^{(t)}.$$

This algorithm performs better than the momentum term in many problems.

Sutskever, Martens, Dahl, Hinton, On the importance of initialization and momentum in deep learning, 2013.

Adaptive step size (Almeida & Silva)

This method assumes that the step size η is different for each component of $\boldsymbol{\theta}$ and changes in each iteration. Therefore,

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \eta_i^{(t)} \frac{\partial J}{\partial \theta_i}\left(\theta^{(t)}\right).$$

Step size update

$$\eta_i^{(t)} = \begin{cases} u\,\eta_i^{(t-1)} & \text{if } \frac{\partial J}{\partial \theta_i}\left(\boldsymbol{\theta}^{(t)}\right) \cdot \frac{\partial J}{\partial \theta_i}\left(\boldsymbol{\theta}^{(t-1)}\right) > 0 \\ d\,\eta_i^{(t-1)} & \text{otherwise} \end{cases}.$$

Typical values for the parameters: $u = 1.2, d = 0.8$. This technique performs very well if the cost function contains valleys aligned with the x axes.

The objective function J is evaluated in each iteration to check if it decreases as expected. If not, the previous values of the parameters are kept and the step sizes are reduced.

Newton method

The Newton methods assumes that we know not only the gradient vector $\nabla_\theta J(\boldsymbol{\theta})$ but also the matrix of second derivatives (Hessian matrix).

$$\nabla_\theta J = \begin{bmatrix} \frac{\partial J}{\partial \theta_1} \\ \frac{\partial J}{\partial \theta_2} \\ \vdots \\ \frac{\partial J}{\partial \theta_d} \end{bmatrix}, \qquad H = \begin{bmatrix} \frac{\partial^2 J}{\partial \theta_1^2} & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 J}{\partial \theta_1 \partial \theta_d} \\ \frac{\partial^2 J}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 J}{\partial \theta_2^2} & \cdots & \frac{\partial^2 J}{\partial \theta_2 \partial \theta_d} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial^2 J}{\partial \theta_d \partial \theta_1} & \cdots & \frac{\partial^2 J}{\partial \theta_2 \partial \theta_2} & \frac{\partial^2 J}{\partial \theta_d^2} \end{bmatrix},$$

and requires the inversion of H in each iteration.

Newton method

Given a guess $\boldsymbol{\theta}^{(t)}$ we can approximate the cost function $J(\boldsymbol{\theta})$ by the 2nd order Taylor expansion

$$J(\boldsymbol{\theta}^{(t)} + \boldsymbol{\Delta}) = J(\boldsymbol{\theta}^{(t)}) + \nabla_\theta J(\boldsymbol{\theta}^{(t)})^T \boldsymbol{\Delta} + \frac{1}{2} \boldsymbol{\Delta}^T H(\boldsymbol{\theta}^{(t)}) \boldsymbol{\Delta}\,,$$

where $\boldsymbol{\Delta}$ is a small displacement vector.

Minimization of $J(\boldsymbol{\theta}^{(t)} + \boldsymbol{\Delta})$ with respect to the displacement $\boldsymbol{\Delta}$, can be achieved by the necessary condition

$$\nabla_\Delta J(\boldsymbol{\theta}^{(t)} + \boldsymbol{\Delta}) = 0\,.$$

Newton method

Necessary condition for optimality,

$$\nabla_{\Delta} J(\boldsymbol{\theta}^{(t)} + \boldsymbol{\Delta}) = 0\,,$$

$$\nabla_{\Delta} \left[J(\boldsymbol{\theta}^{(t)}) + \nabla_{\theta} J(\boldsymbol{\theta}^{(t)})^T \boldsymbol{\Delta} + \frac{1}{2} \boldsymbol{\Delta}^T H(\boldsymbol{\theta}^{(t)}) \boldsymbol{\Delta} \right] = 0\,,$$

$$\nabla_{\theta} J(\boldsymbol{\theta}^{(t)}) + H(\boldsymbol{\theta}^{(t)}) \boldsymbol{\Delta} = 0\,,$$

$$\boldsymbol{\Delta} = - \left[H(\boldsymbol{\theta}^{(t)}) \right]^{-1} \nabla_{\theta} J(\boldsymbol{\theta}^{(t)})\,.$$

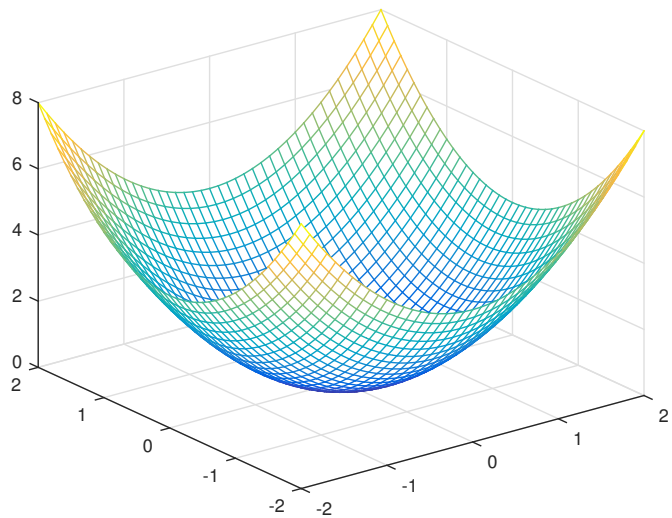
Therefore,

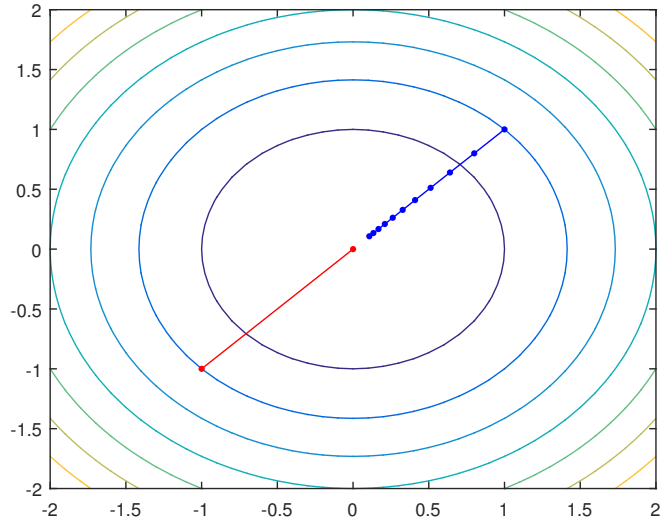
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \left[H(\boldsymbol{\theta}^{(t)}) \right]^{-1} \nabla_{\theta} J(\boldsymbol{\theta}^{(t)})\,.$$

The Newton method gives an exact solution for the parameters if J is a quadratic function.

Example 1 - gradient vs Newton method

Quadratic function: $J(x_1, x_2) = x_1^2 + x_2^2$.

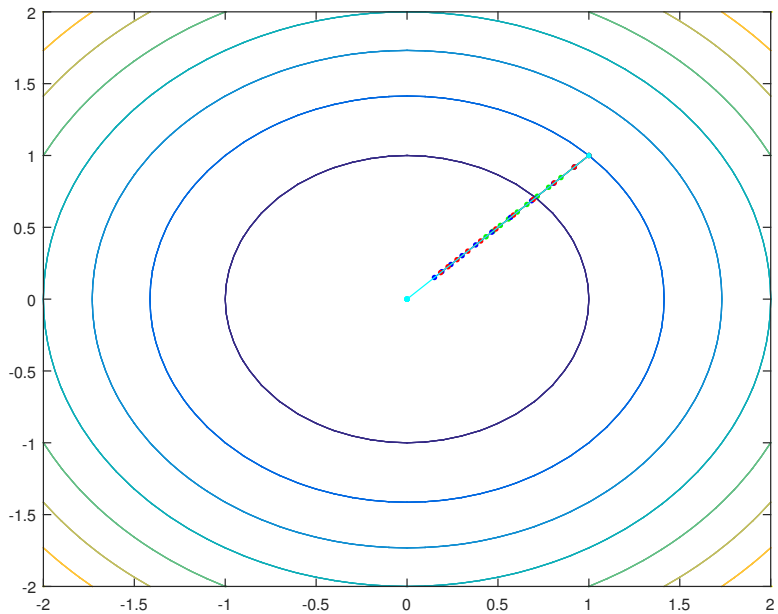


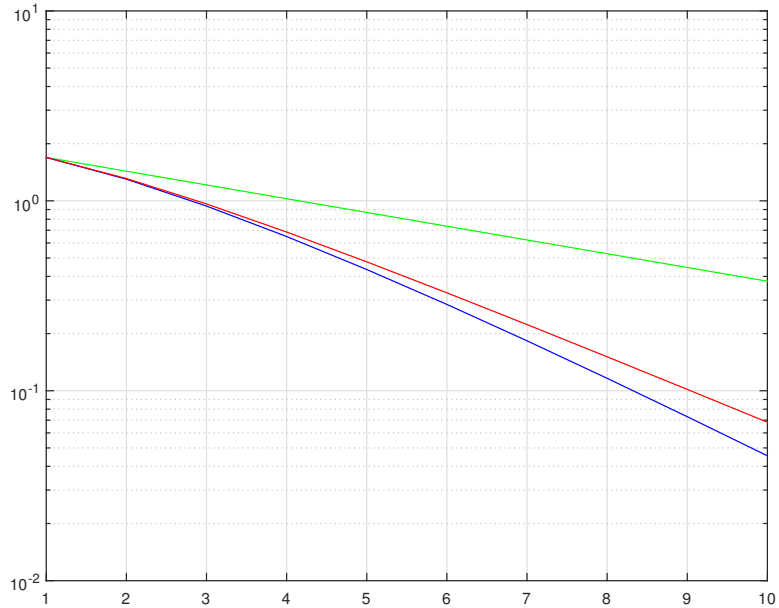


10 iterations of gradient descent (blue) and 1 iteration of Newton method (red).

Example 1 - gradient, momentum, Nesterov, Newton

Quadratic function: $J(x_1, x_2) = x_1^2 + x_2^2$.

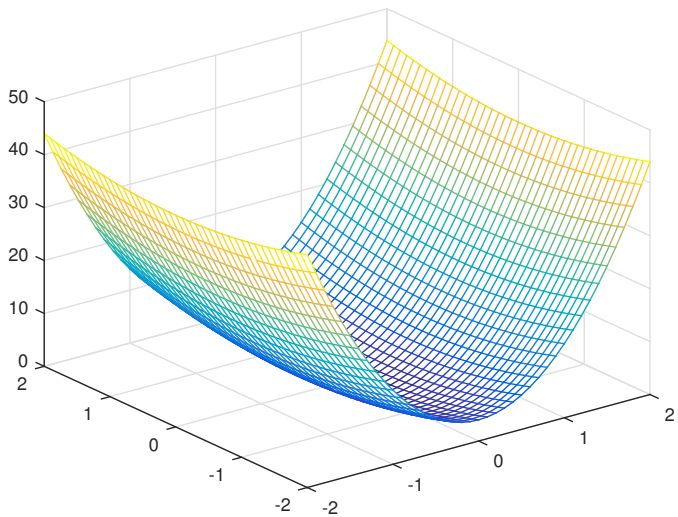


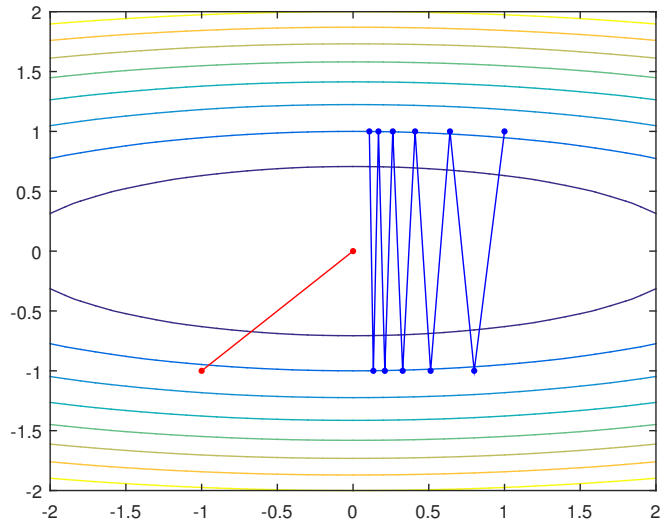


Left: 10 iterations of gradient descent (green), gradient with momentum (blue) Nesterov accelerated gradient (red) and Newton method (cyan). Right: cost function. Newton outside scope.

Example 2 - gradient vs Newton method

Quadratic function: $J(x_1, x_2) = x_1^2 + 10x_2^2$.

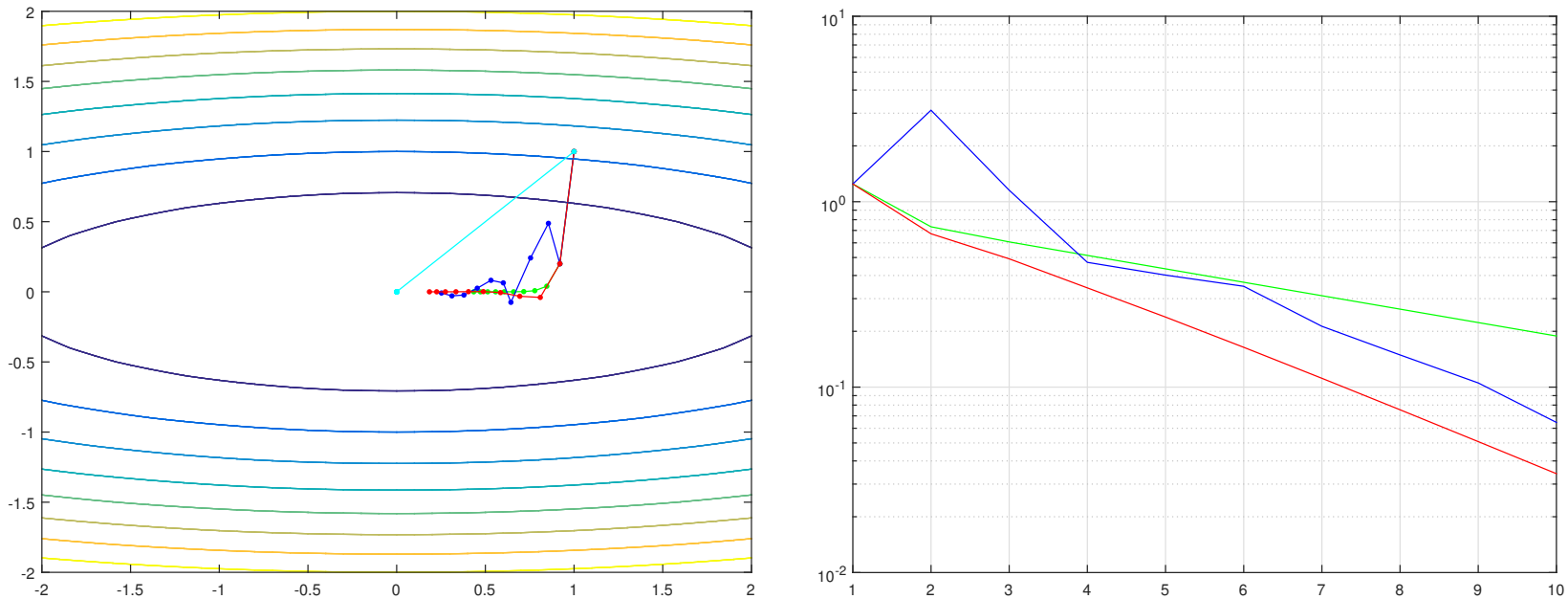




10 iterations of gradient descent (blue) and 1 iteration of Newton method (red).

Example 2 - gradient, momentum, Nesterov, Newton

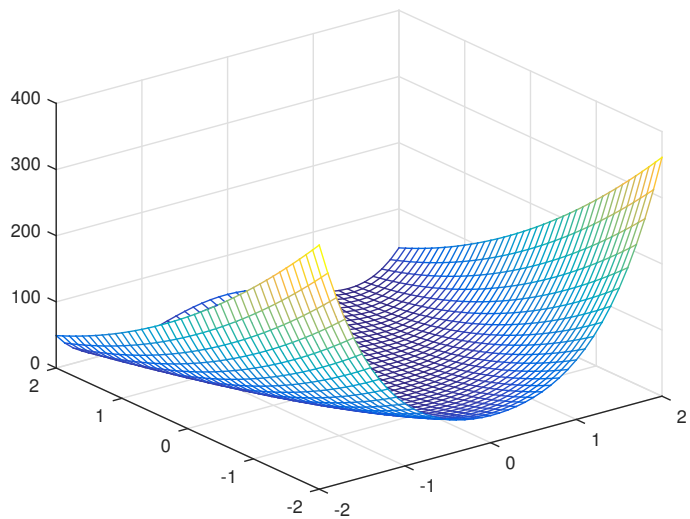
Quadratic function: $J(x_1, x_2) = x_1^2 + 10x_2^2$.

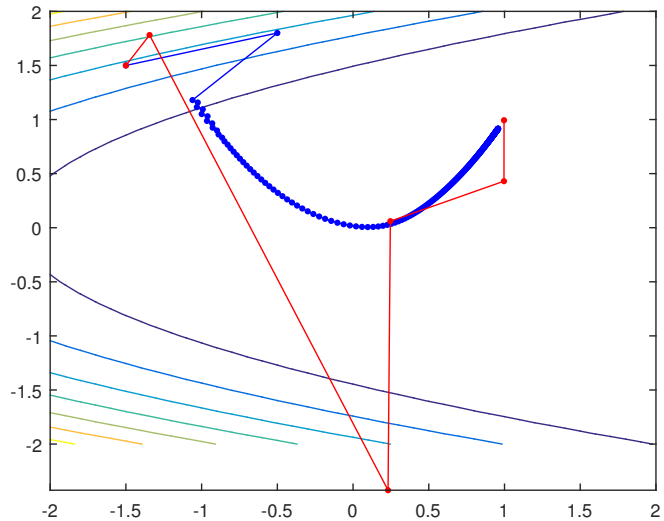


Left: 10 iterations of gradient descent (green), gradient with momentum (blue) and Nesterov accelerated gradient (red) and Newton method (cyan). Right: cost function. Newton outside scope.

Example 3 - gradient vs Newton method

Rosenbrock function with constant 10, instead of 100, to simplify the problem: $J(x_1, x_2) = (1 - x_1)^2 + 10(x_2 - x_1^2)^2$.

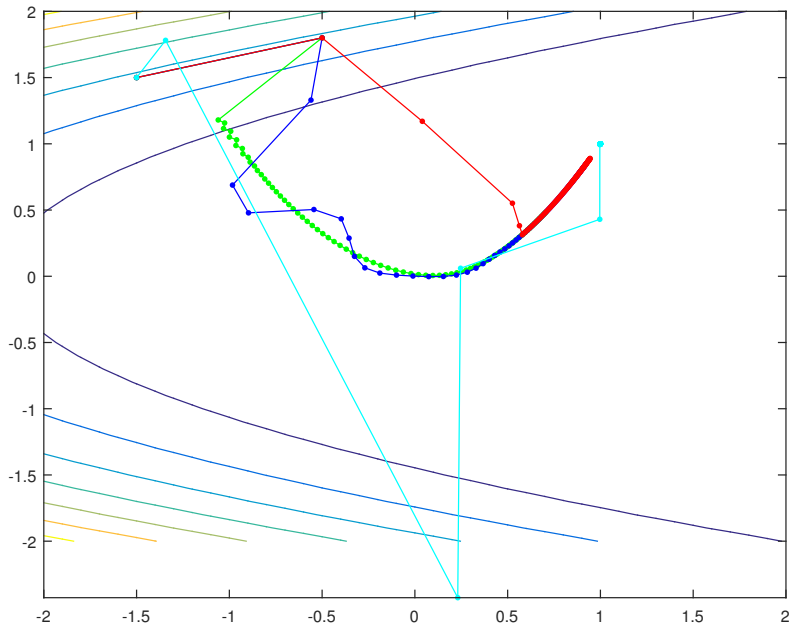


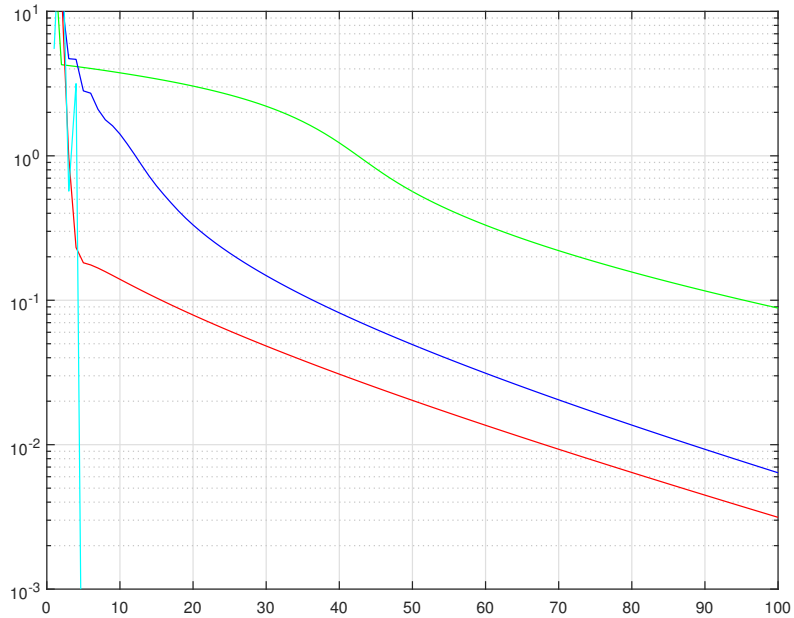


300 iterations of gradient descent (blue) and 5 iteration of Newton method (red).

Example 3 - gradient, momentum, Nesterov, Newton

Rosenbrock function with constant 10, instead of 100, to simplify the problem: $J(x_1, x_2) = (1 - x_1)^2 + 10(x_2 - x_1^2)^2$.





Left: 100 iterations of gradient descent (green), gradient with momentum (blue), Nesterov accelerated gradient (red) and Newton method (cyan). Right: cost function. Newton outside scope.

Machine Learning

Linear Regression

Regularization

Evaluation & Generalization

Optimization

Neural networks

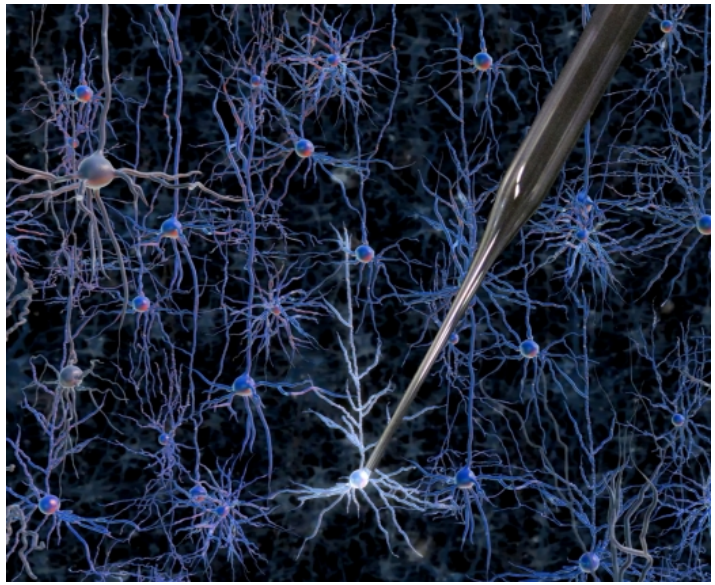
Data classification

Linear classifiers

Support vector machines

Decision Trees and Random Forest

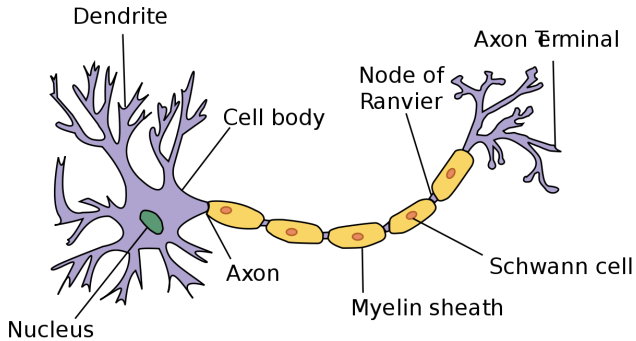
History



MIT

- The human brain has been a source of inspiration in Computer Science.
- The brain has a large number of processing units ($\sim 10^{11}$ neurons) that are slow (~ 1 ms) but highly connected. Although they are slow, the neurons are able to perform very complex tasks, *e.g.*, visual tasks, in real time and almost effortless.
- One of the first models of a neuron was proposed by McCulloch & Pitts in the 40s and was a starting point for an exciting area of artificial neural networks (ANN).

Neuron



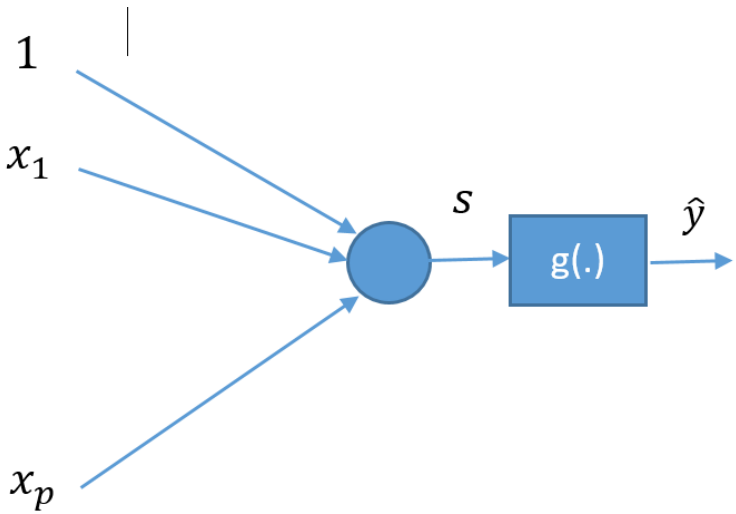
Wikipedia

The neuron is a cell consisting of dendrites (inputs), a soma (cell body) and an axon (output).

It receives input signals through its dendrites. These signals are combined in the soma and, from time to time, an electric impulse is generated that travels through the axon and influences other cells.

McCulloch & Pitts model

The neuron model proposed by McCulloch & Pitts (1942) has a linear part, followed by a nonlinearity:



weighted sum of inputs (activation)

$$s = [1\ x^T]w = \tilde{x}^T w\,.$$

output (Heaviside function)

$$\hat{y} = z = g(s) = \begin{cases} 1 & \text{if } s \geq 0 \\ 0 & \text{otherwise} \end{cases}\,.$$

The weighted sum s is called activation, the nonlinear function $g : \mathbb{R} \to \mathbb{R}$ is known as activation function and the vector $w = [w_0 \ldots w_p]^T$ is the weight vector.

Rosenblatt algorithm

Rosenblatt proposed an iterative algorithm in the 50s to train the weights of the McCulloch & Pitts unit for the prediction of binary outcomes.

Rosenblatt algorithm

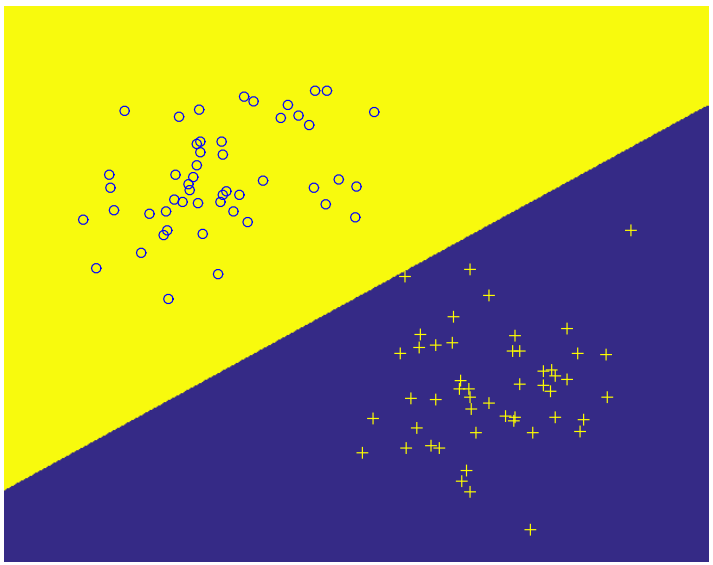
1. training set: $\mathcal{T} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$, with $x^{(k)} \in \mathbb{R}^p, y^{(k)} \in \{0, 1\}$;
2. initialization: randomly initialize the weights $w_i(0), i = 0, \ldots, p$;
3. new training example: present a new training pattern $(x(t), y(t))$ to the model and compute the model output $\hat{y}(t) = g(\tilde{x}^T(t)w(t-1))$;
4. update: update the weights according to

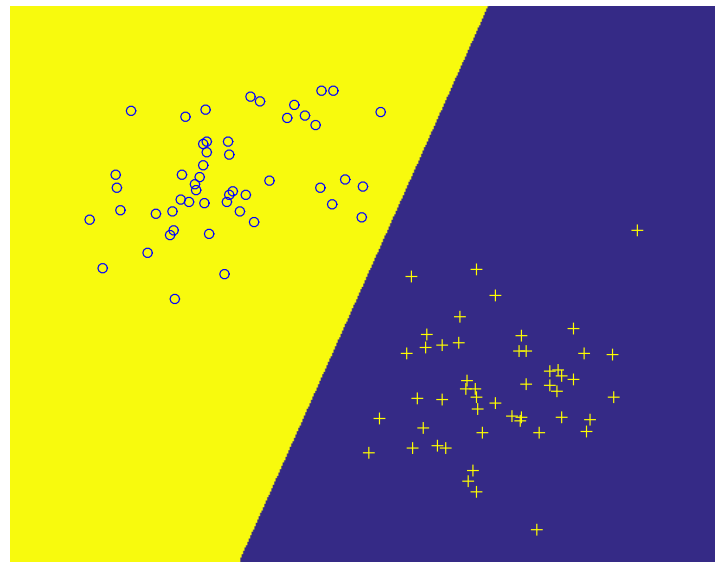
$$w_i(t) = w_i(t-1) + \eta \tilde{x}_i(t)\epsilon(t), \quad \epsilon(t) = y(t) - \hat{y}(t),$$

   where $y(t)$ is the desired outcome for the input $x(t)$.
5. cycle: return to step 3, until a stop condition is met.

Example: Gaussian data

Two trials of the Rosenblatt algorithm applied to the same linearly separable data.





The training data is the same but the outcome is different in each experiment (why?)

Pros & cons

Pros

It can be proved that the Rosenblatt algorithm solves any binary problem in a finite number of iterations, provided the training data can be separated by a hyperplane in feature space.

Cons

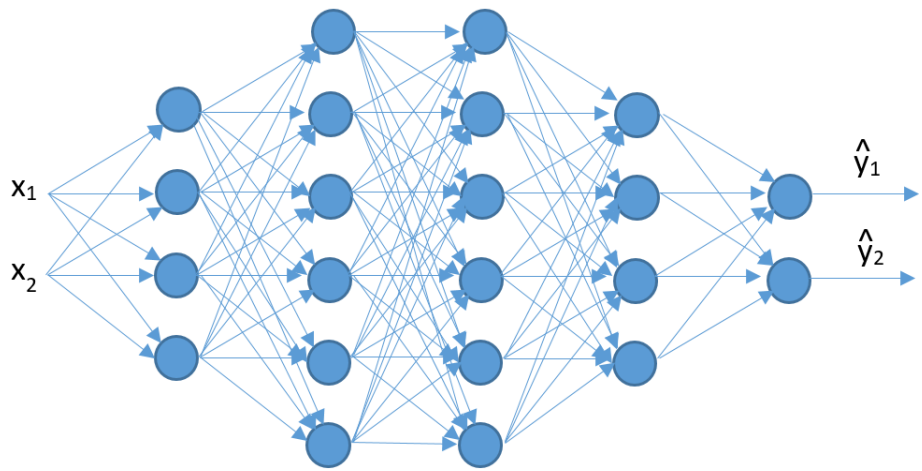
It does not provide a hint to deal with data that cannot be separated by a hyperplane or to deal with regression problems that are not binary.

Most practical problems are noisy and fit into one of these categories. Therefore, a single unit trained by the Rosenblatt algorithm is seldom useful in practice.

Multilayer perceptron

To overcome previous limitations of a single unit, three important issues were proposed:

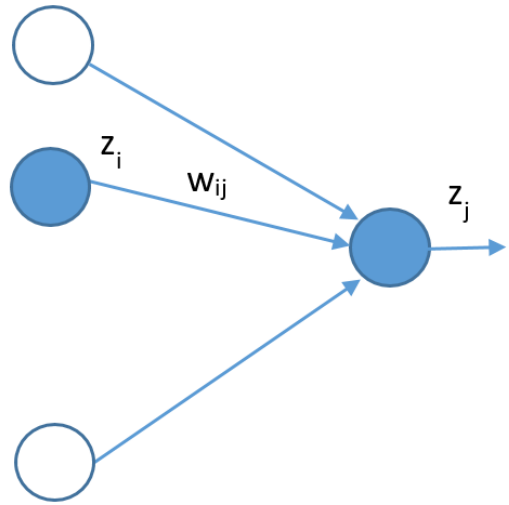
- architectures with multiple units, usually organized in layers, known as multilayer perceptron (MLP), and
- continuous and differentiable activation functions.
- training based on the minimization of a cost function.



For the sake of simplicity, the activation function of each unit is not explicitly represented (but it exists!). offsets are not shown.

Weights

Each unit i is connected to a unit j of the next layer through a weight w_{ij}.



unit j (layer $\ell = 1$)

$$s_j = w_{0j} + \sum_{i \in \text{input}} w_{ij} x_i$$

$$z_j = g(s_j)$$

unit j (layer $\ell > 1$)

$$s_j = w_{0j} + \sum_{i \in \text{previous layer}} w_{ij} z_i$$

$$z_j = g(s_j)$$

$g(.)$ is the activation function and the weight w_{0j} is called the offset.

Visible and hidden units

The units of the last layer are considered as visible. They are the output of the network and we denote their output by $\hat{y}_i$

The units of the other layers are considered as hidden since we do not know their desired values in the training phase. They are intermediate variables used to compute the network output.

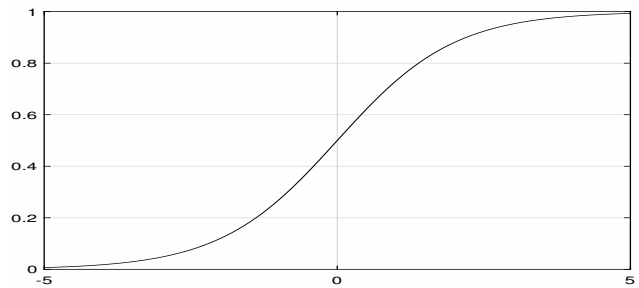
Activation function

The activation functions should be continuous and differentiable to allow the evaluation of the influence weight changes on the network output (credit assignment).

Some common choices are:

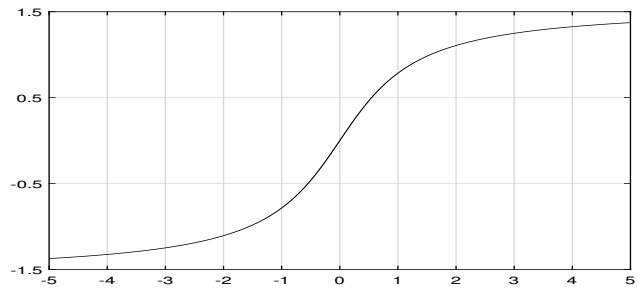
sigmoid: logistic function

$$g(s) = \frac{1}{1 + e^{-s}}.$$



sigmoid: arctangent function

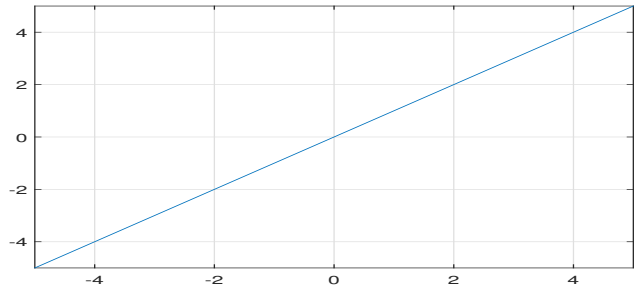
$$g(s) = \arctan s.$$



Activation function

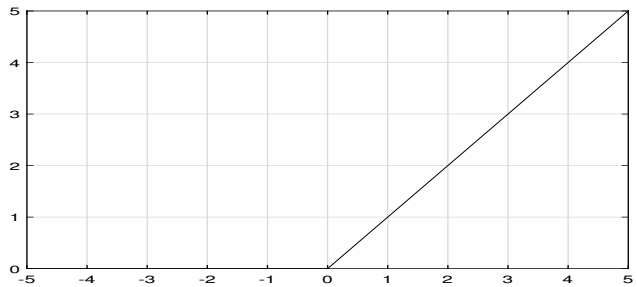
Linear unit

$$g(s) = s.$$



ReLU: rectified linear unit (recent)

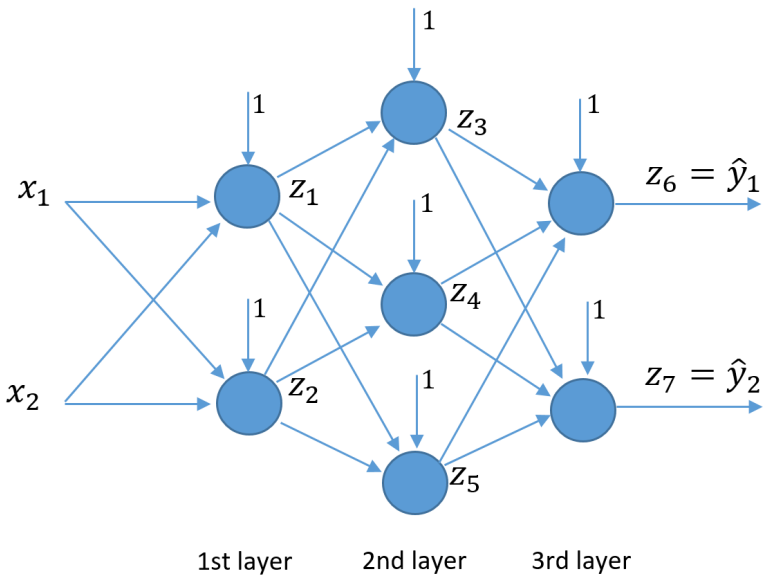
$$g(s) = \max(0, s).$$



The ReLU is currently the recommended activation function since it does not saturate making the convergence of the gradient algorithm faster. Other units (linear, softmax) are often used in the output layer.

Exercises

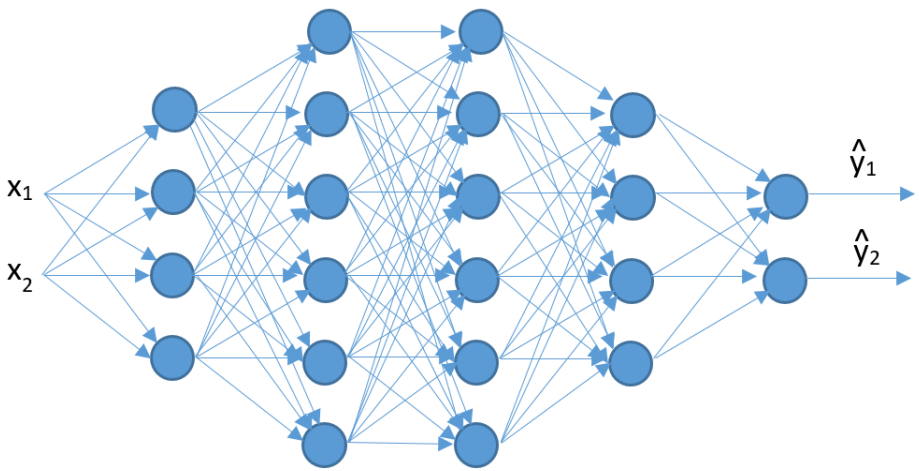
1. Compute the derivative of the activation functions:
   - $g(s) = \frac{1}{1+e^{-s}}$,
   - $g(s) = \arctan s$,
   - $g(s) = s$,
   - $g(s) = \max(0, s)$.
2. Write the equations for the network



Arquitecture & weights

To specify a multilayer perceptron we need to indicate the arquitecture:

- number of layers
- number of units per layer



We also need to indicate

- activation functions
- weights

$$w = \{w_{ij}\}$$

where w_{ij} is the weight connecting the output of unit i to unit j

The network is thus a nonlinear map $\hat{y} = f(x, w)$ from the input space to the output space, controlled by a set of weights w.

How do we choose the architecture?

How should we choose the number of layers?

Cybenko (1989) proved that a multilayer perceptron with 1 hidden layer is an universal approximator of any continuous function defined on a compact subset of $\mathbb{R}^p$. This is a useful theorem but it does not explain how many units are needed nor how should the weights be chosen.

- Common practice shows that it is often better to use more layers since the network can synthesize a wider variety of nonlinear functions with less units.
- It also shows that deeper networks (with more layers) are more difficult to train.

Great improvements were achieved in the last 10 years in the training of deep neural networks. The state of the art in many problems (vision, speech, text processing) is now based on neural networks.

Multi-layer perceptron training

After choosing the NN architecture, we need to learn all the weights, using a training set of labeled patterns $\mathcal{T} = \left\{(x^{(k)}, y^{(k)}), k = 1, \ldots, n\right\}$.

Goal: minimize the total loss (cost)

$$\mathcal{C} = \sum_{k=1}^{n} L(y^{(k)}, \hat{y}^{(k)}) = \sum_{k=1}^{n} L^{(k)},$$

where $\hat{y}^{(k)}$ is the network output for the input $x^{(k)}$. A typical choice for the loss function is the quadratic error

$$L(y, \hat{y}) = \|y - \hat{y}\|^2 .$$

The minimization of $\mathcal{C}$ is often achieved by using the gradient algorithm

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t) \ , \qquad \Delta w_{ij}(t) = -\eta \left. \frac{\partial \mathcal{C}}{\partial w_{ij}} \right|_{w(t)} ,$$

or a modified version of it; η denotes the learning step.

Training modes: batch, mini-batch and on-line

The gradient vector includes the contribution of all the training patterns. The weight update using all the training patterns in each iteration is called the batch mode.

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{C}}{\partial w_{ij}} = -\eta \sum_k \frac{\partial L(y^{(k)}, \hat{y}^{(k)})}{\partial w_{ij}} = -\eta \sum_k \frac{\partial L^{(k)}}{\partial w_{ij}} \ .$$

Another alternative consists of using one training pattern k, only, and updating the weights with that information. This is called the on-line mode or stochastic gradient

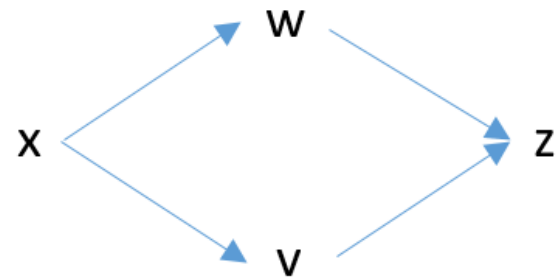
$$\Delta w_{ij} = -\eta \frac{\partial L^{(k)}}{\partial w_{ij}} \ .$$

A third hypothesis consists of updating the NN weights using a small subset of training patterns. This is known as mini-batch mode.

Chain rule of differentiation

To train the weights w_{ij} we need the gradient of the loss function L.

This task relies on the chain rule of differentiation.



$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

$$\frac{dz}{dx} = \frac{\partial z}{\partial w}\frac{dw}{dx} + \frac{\partial z}{\partial v}\frac{dv}{dx}$$

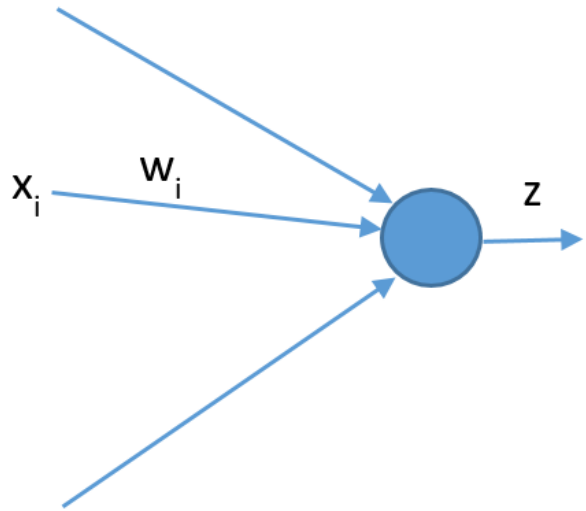
Training a single unit

Let us start by a simple problem: a network with a single unit trained with a single pattern (x, y).

Forward network:

$$s = w_0 + \sum_{i=1}^{p} w_i x_i,$$

$$\hat{y} = z = g(s)$$



Gradient:

$$\frac{\partial L}{\partial w_p} = \frac{dL}{ds}\,\frac{\partial s}{\partial w_p} = \epsilon\, x_p\,.$$

Therefore, the gradient is given by

$$\frac{\partial L}{\partial w_p} = x_p\,\epsilon \qquad \epsilon = \frac{dL}{ds} = \frac{dL}{d\hat{y}}\frac{d\hat{y}}{ds} = g'(s)\frac{dL}{d\hat{y}}.$$

Compare with the Rosenblatt algorithm.

Gradient structure

The structure of the gradient can be extended to more general cases.

If unit q belongs to a layer ℓ higher than 1,

$$s_q = w_{0q} + \sum_{i \in \text{previous layer}} w_{iq} z_i .$$

Using the chain rule, the derivative of L with respect to a weight w_{pq} can be computed as

$$\frac{\partial L}{\partial w_{pq}} = \frac{\partial L}{\partial s_q} \frac{\partial s_q}{\partial w_{pq}} = z_p \, \epsilon_q ,$$

Therefore,

$$\frac{\partial L}{\partial w_{pq}} = z_p \epsilon_q , \qquad \epsilon_q = \frac{\partial L}{\partial s_q} \qquad q \in \text{layer higher than 1.}$$

If the unit q belongs to the first layer, z_p is replaced by x_p.

Training the output layer

These ideas can be applied to NN with multiple layers. Let us start by the output layer.

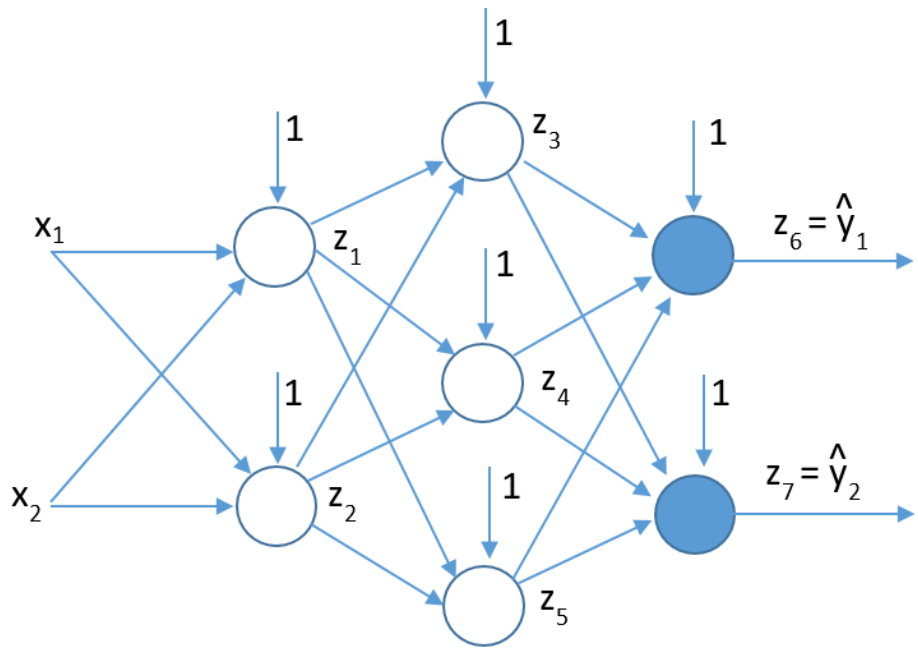
Forward network (unit $j \in \{6, 7\}$):

$$s_j = w_{0j} + \sum_{i \in \text{previous layer}} w_{ij} z_i,$$

$$z_j = g(s_j).$$

Gradient (unit $q \in \{6, 7\}$):

$$\frac{\partial L}{\partial w_{pq}} = z_p \epsilon_q.$$



where

$$\epsilon_q = \frac{\partial L}{\partial s_q} = \frac{\partial L}{\partial z_q} \frac{\partial z_q}{\partial s_q} = g'(s_q) \frac{\partial L}{\partial z_q}.$$

Training a hidden layer

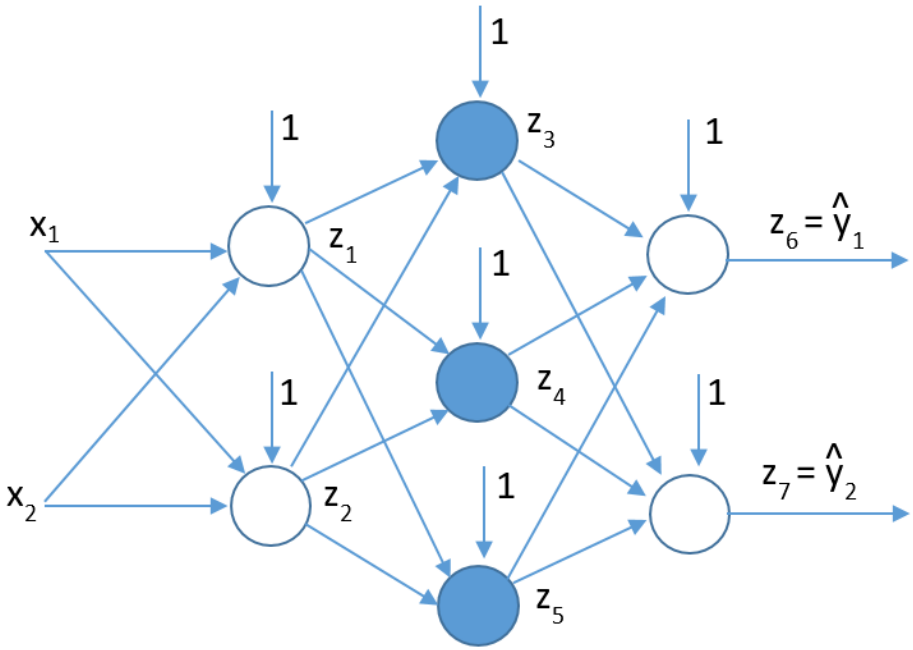
Let us consider units from a hidden layer.

Forward network ($j \in \{3, 4, 5\}$):

$$s_j = w_{0j} + \sum_{i=1}^{p} w_{ij} z_i,$$
$$z_j = g(s_j).$$

Gradient ($q \in \{3, 4, 5\}$):

$$\frac{\partial L}{\partial w_{pq}} = z_p \epsilon_q.$$



where

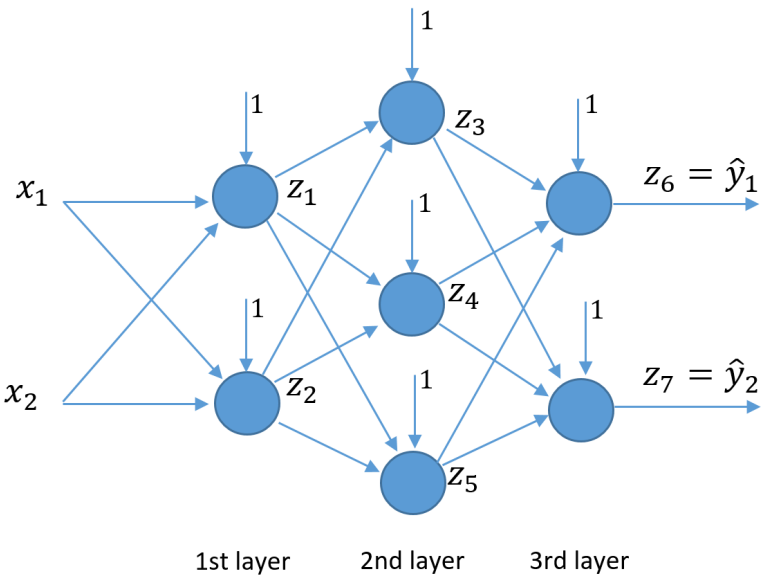
$$\epsilon_q = \frac{\partial L}{\partial s_q} = \sum_{j \in \text{next layer}} \left(\frac{\partial L}{\partial s_j}\right)\left(\frac{\partial s_j}{\partial z_q}\right)\left(\frac{\partial z_q}{\partial s_q}\right) = g'(s_q) \sum_{j \in \text{next layer}} w_{qj} \epsilon_j.$$

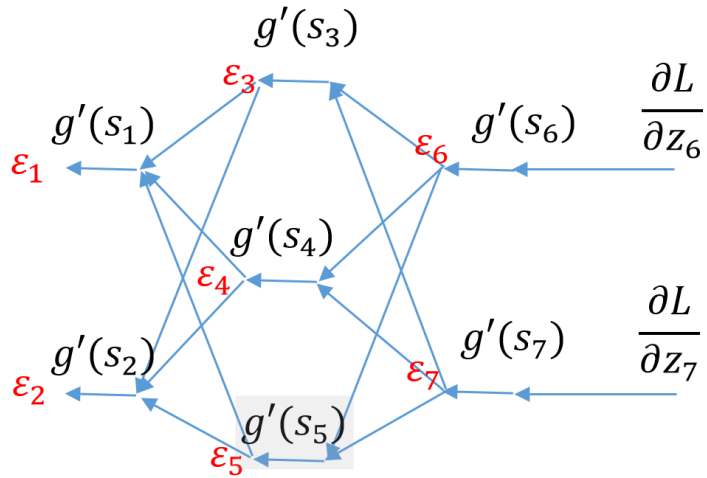
Backpropagation algorithm

The gradient components are given by

$$\frac{\partial L}{\partial w_{ij}} = z_i \epsilon_j \,,$$

where z_i is obtained from the multilayer perceptron and ϵ_j is obtained from auxiliary network called backpropagation network.

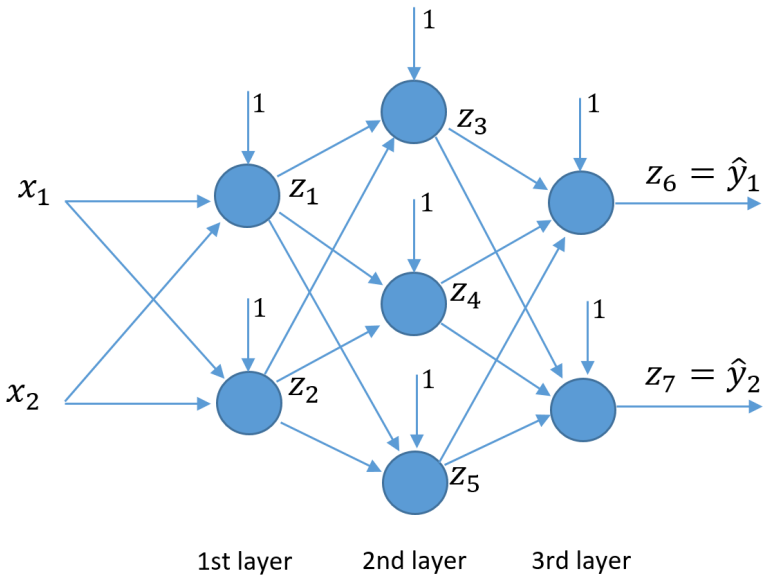


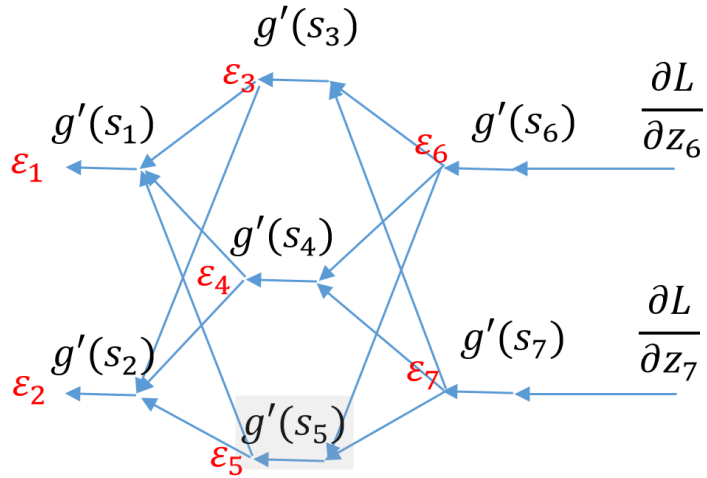


This algorithm for the computation of the gradient using the backpropagation network is known as the backpropagation algorithm.

Backpropagation network

How do we build the backpropagation network?





The backpropagation network (right) is obtained from the original network (left)by

- linearizing nonlinear units (activation functions);
- inverting the direction of links, converting sums into derivation points (and vice versa);
- the output of linearized branches are the variables ϵ_i;
- the input of backpropagation network are the derivatives of the loss with respect to forward network outputs.

Acceleration techniques

The convergence of the gradient algorithm is often very slow and acceleration techniques are usually adopted, namely:

- momentum term;
- adaptive weights.

These techniques modify the weight update rule and were discussed before in the optimization lesson. Next we summarize the steps involved in the gradient algorithm with momentum term (batch and on-line).

Gradient algorithm (batch) with momentum term

Set $t = 1$ and $\Delta w_{ij}(0) = 0$. Repeat steps 1 through 4 below until stopping criteria is met

1. Set the variables g_{ij} to zero. These variables will be used to accumulate the gradient components.
2. For $k = 1, \ldots, n$, perform steps 2.1 through 2.4
   - 2.1 propagate forward: apply the training pattern $x^{(k)}$ to the perceptron and compute the variables z_i and outputs $\hat{y}_j^{(k)}$
   - 2.2 compute the cost derivatives: $\frac{\partial L^k}{\partial \hat{y}_j^{(k)}}$
   - 2.3 propagate backwards: apply $\frac{\partial L^k}{\partial \hat{y}_j^{(k)}}$ to the inputs of backpropagation network and compute its internal variables ϵ_j
   - 2.4 compute and accumulate components: compute the variables $\frac{\partial L^k}{\partial w_{ij}} = z_i \epsilon_j$ and accumulate each of them in the corresponding variable *i.e.*, $g_{ij} \leftarrow g_{ij} + z_i \epsilon_j$
3. Apply momentum: set $\Delta w_{ij}(t) = -\eta g_{ij} + \alpha \Delta w_{ij}(t-1)$
4. Update the weights: set $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$

adapted from L. Almeida, Handbook of Neural Computation, 1997.

Gradient algorithm (on-line) with momentum term

Set $t = 1$ and $\Delta w_{ij}(0) = 0$. Repeat step 1 until stopping criteria is met

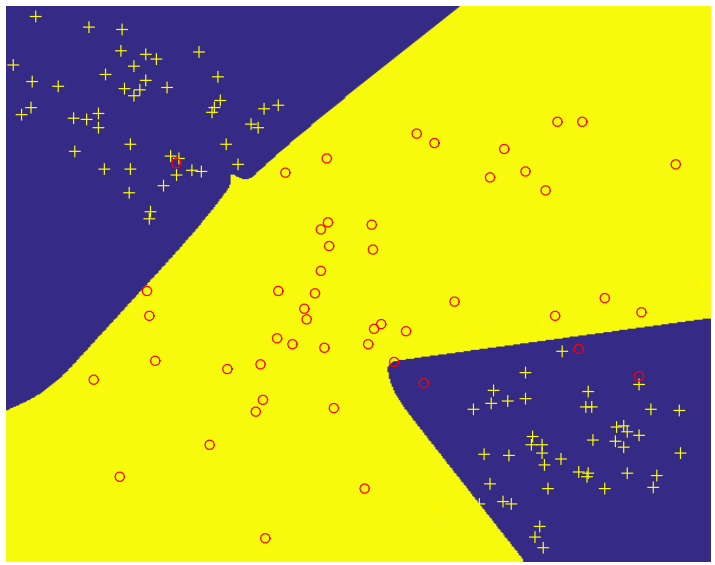
1. For $k = 1, \ldots, n$, perform steps 1.1 through 1.6
   - 1.1 propagate forward: apply the training pattern x^k to the perceptron and compute the variables z_i and outputs $\hat{y}^{(k)}$
   - 1.2 compute the cost derivatives: $\frac{\partial L^k}{\partial \hat{y}_j^{(k)}}$
   - 1.3 propagate backwards: apply $\frac{\partial L^k}{\partial \hat{y}_j^{(k)}}$ to the inputs of backpropagation network and compute its internal variables ϵ_j
   - 1.4 compute the gradient components: compute the variables $\frac{\partial L^k}{\partial w_{ij}} = z_i \epsilon_j$
   - 1.5 Apply momentum: set $\Delta w_{ij}(t) = -\eta z_i \epsilon_j + \alpha \Delta w_{ij}(t-1)$
   - 1.6 Update the weights: set $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$

adapted from L. Almeida, Handbook of Neural Computation, 1997.

Example

Output of a multi layer perceptron trained by the gradient algorithm using the backpropagation method.

- data: 150 training patterns; binary outcome
- architecture: 2 inputs, 2 hidden layers and one output layer (5-3-1 units)
- activation function: logistic
- training mode: on-line, no speeding algorithms



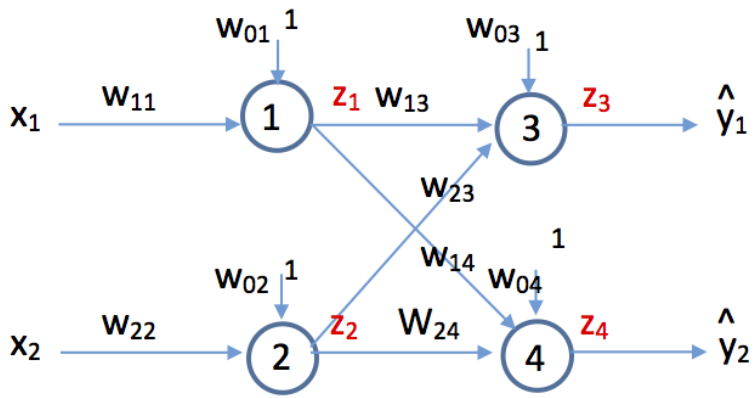
Regression vs classification

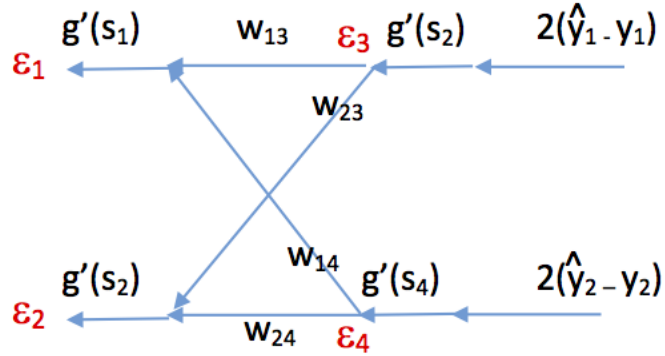
MLPs can be used for regression and for classification tasks.

In regression tasks the output units typically have linear activation functions and the network is trained with quadratic loss (SSE).

In classification tasks the output units typically have logistic or Softmax activation functions and the network is often trained with negative log-likelihood (cross-entropy) loss. This will be discussed later.

Example





Write all the equations required to compute the gradient components, assuming one training example and the SSE cost

$$SSE = (y_1 - \hat{y}_1)^2 + (y_2 - \hat{y}_2)^2$$

Forward network

$$s_1 = w_0 1 + w_{11} x_1, \quad z_1 = g(s_1)$$

$$s_2 = w_0 2 + w_{22} x_2, \quad z_2 = g(s_2)$$

$$s_3 = w_0 3 + w_{13} z_1 + w_{23} z_2, \quad z_3 = g(s_3)$$

$$s_4 = w_0 4 + w_{14} z_1 + w_{24} z_2, \quad z_4 = g(s_4)$$

Backward network

input: $2(\hat{y}_1 - y_1)$ $2(\hat{y}_2 - y_2)$

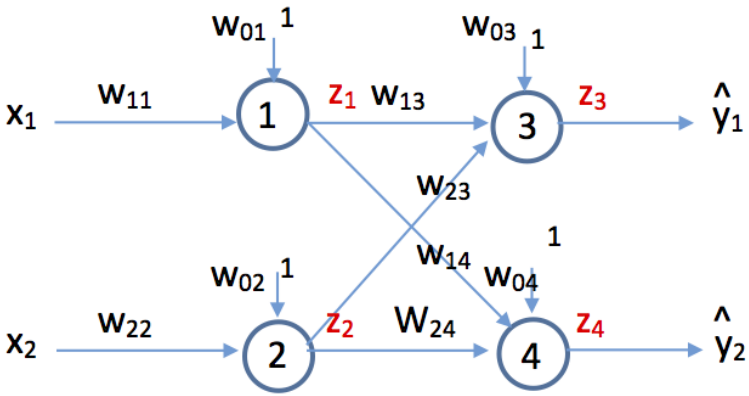
$$\epsilon_4 = g'(s_4) \times 2(\hat{y}_2 - y_2)$$

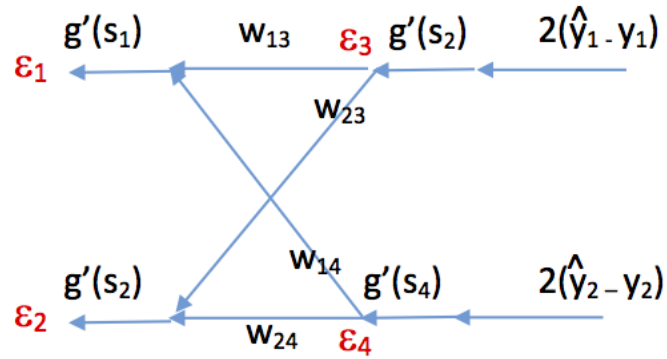
$$\epsilon_3 = g'(s_3) \times 2(\hat{y}_1 - y_1)$$

$$\epsilon_2 = g'(s_2)[w_{24}\epsilon_4 + w_{23}\epsilon_3]$$

$$\epsilon_1 = g'(s_1)[w_{14}\epsilon_4 + w_{13}\epsilon_3]$$

Example (cont)





$$\Delta_{01} = 1\epsilon_1, \quad \Delta_{11} = x_1\epsilon_1 \qquad \Delta_{03} = 1\epsilon_3, \quad \Delta_{13} = z_1\epsilon_3, \quad \Delta_{23} = z_2\epsilon_3$$

$$\Delta_{02} = 1\epsilon_2, \quad \Delta_{22} = x_2\epsilon_2 \qquad \Delta_{04} = 1\epsilon_4, \quad \Delta_{14} = z_1\epsilon_4, \quad \Delta_{24} = z_2\epsilon_4$$

Exercises

1. Consider the multi-layer perceptron sketched in previous slides. Write the equations for the gradient of the loss function with respect to all the weights.

2. Write the equations for the forward and backpropagation networks using matrix notation. Consider vectors $s^{(\ell)}, y^{(\ell)}, \epsilon^{(\ell)}$ containing the s, y and ϵ variables associated to layer ℓ.

Exercises

- Consider a MLP with one layer and linear units. Prove that the input output map performed by the perceptron is a linear (affine) transformation

$$f(x) = Ax + b$$

- prove this statement for the case of a MLP with two layers and linear units.

This property can be extended to an arbitrary number of layers provided the units are linear. Matrix A may be rank deficient if some of the hidden layers has less units than the input or the output.

Analysis of images with neural networks

Imagine that you want to distinguish images of horses and cats. How would you proceed?

Analysis of images with neural networks

Images encode information in very complicated ways (different viewpoints, shapes, colors, textures, illumination).

Finding a set of rules on low level image features (e.g., color, corners) seems to be unfeasible!

Analysis of images with neural networks

However, some properties should hold:

- There is a spatial dependence in images.
- There are small regions that convey important information (e.g., eyes, ears). Some kind of pattern matching might work.
- The system should be invariant to translation, color, and illumination changes.

How do we put this information in an algorithm?

May be the human brain can inspire us again.

Convolutional neural networks

Convolutional neural networks (CNN) have recently achieved an enormous success in the analysis of images.

History

- receptive fields (Hubel and Wiebel, 1950s, 60s): individual neurons in the visual cortex respond to small regions in the field of view.
- neocognitron (Fukushima, 1980): hierarchical model using receptive fields.
- LeNet-5 (LeCun et al., 1998): convolutional neural network proposed for digit recognition.
- Alexnet (Krizhevsky et al., 2012): convolutional neural network. Breakthrough in Imagenet international challenge.

ImageNet - Large Scale Visual Recognition Challenge

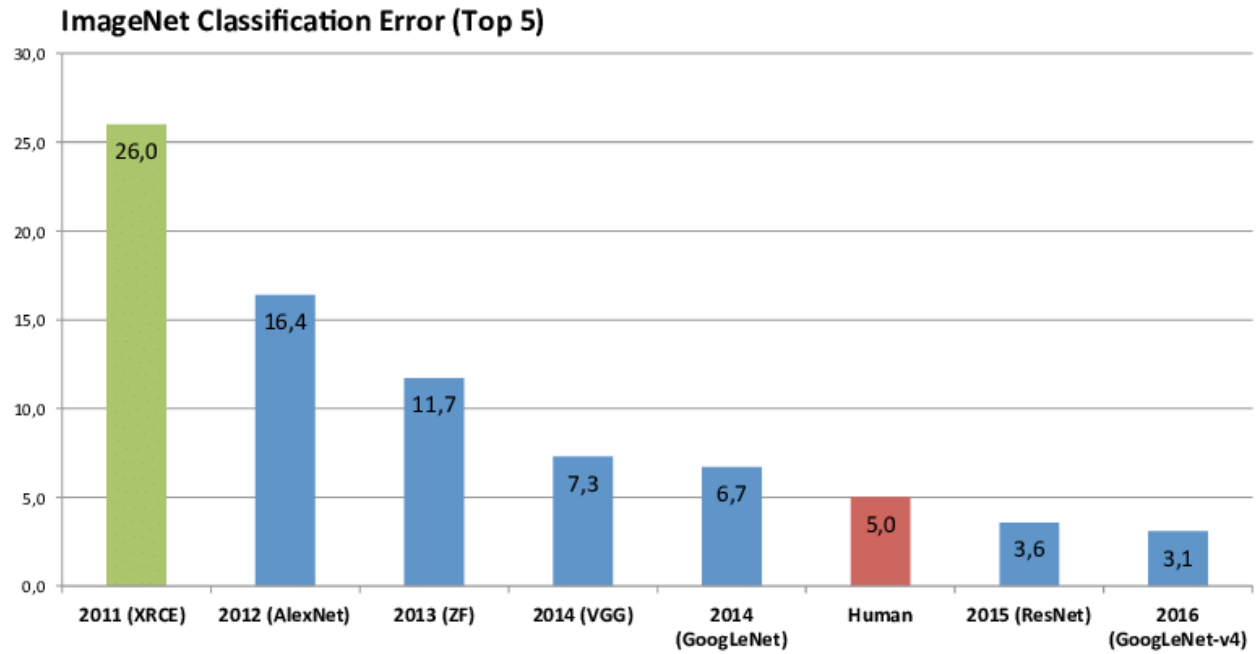
LSVRC - Imagenet dataset

- 1K categories
- 1M images (1K images per category)
- annotation: manual annotation using Amazon Mechanical Turk.

Breakthrough 2012 - Alexnet

Alexnet won the ImageNet challenge in 2012 by a large margin.

A. Krizhevsky, I. Sutskever, G. Hinton, ImageNet classification with Deep Convolution Neural Networks, NIPS, 2012



Since then, all the winners of the ImageNet challenge are convolutional neural networks

End-to-end architecture

In Alexnet, no image features (handcrafted features) are defined by the user.

Alexnet learns to directly compute the image label (class) for the input image. Of course this requires a large training set (more than 1 million images).

This strategy is called the end-to-end approach.

The classic blocks (feature extraction and classification) are both learned from the training data without the use of handcrafted features.

Basic CNN architecture

A convolutional neural network (CNN) receives an input image and predicts an output image or a label, based on a sequence of internal representations that extract useful information (features) from the image.

Most of these representations are 3D arrays. Each 3D array can be viewed as a collection of 2D arrays known as channels or feature maps.

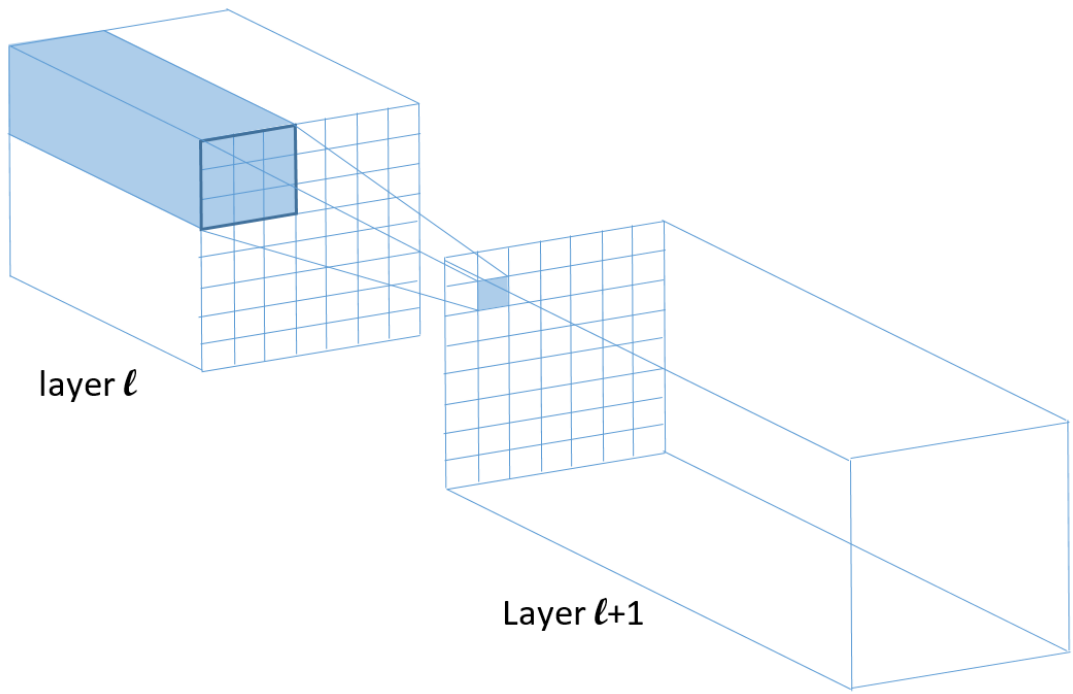
Internal representations of the image are obtained by a concatenation of layers, including:

- convolutional layers: convolution followed by non-linearity (activation function)
- pooling layers: dimensionality reduction
- fully connected layers: used in classification problems

Convolution layer

A convolution layer receives a 3D input, convolves it with a set of kernels (filters) and applies an activation function (typically RELU) to the filter outputs.

Each kernel has a localized support in the first two (spatial) coordinates and it is full range in the third (depth) coordinate.



Convolutional layer

3D input: $z_{ijk}^{\ell-1}$ $\ell-1$ - number of input layer

3D kernel: h_{ijk}^{ℓ}

2D output:

$$s_{ij}^{\ell} = \sum_p \sum_q \sum_r h_{pqr}^{\ell} z_{i+p,j+q,0+r}^{\ell-1}$$

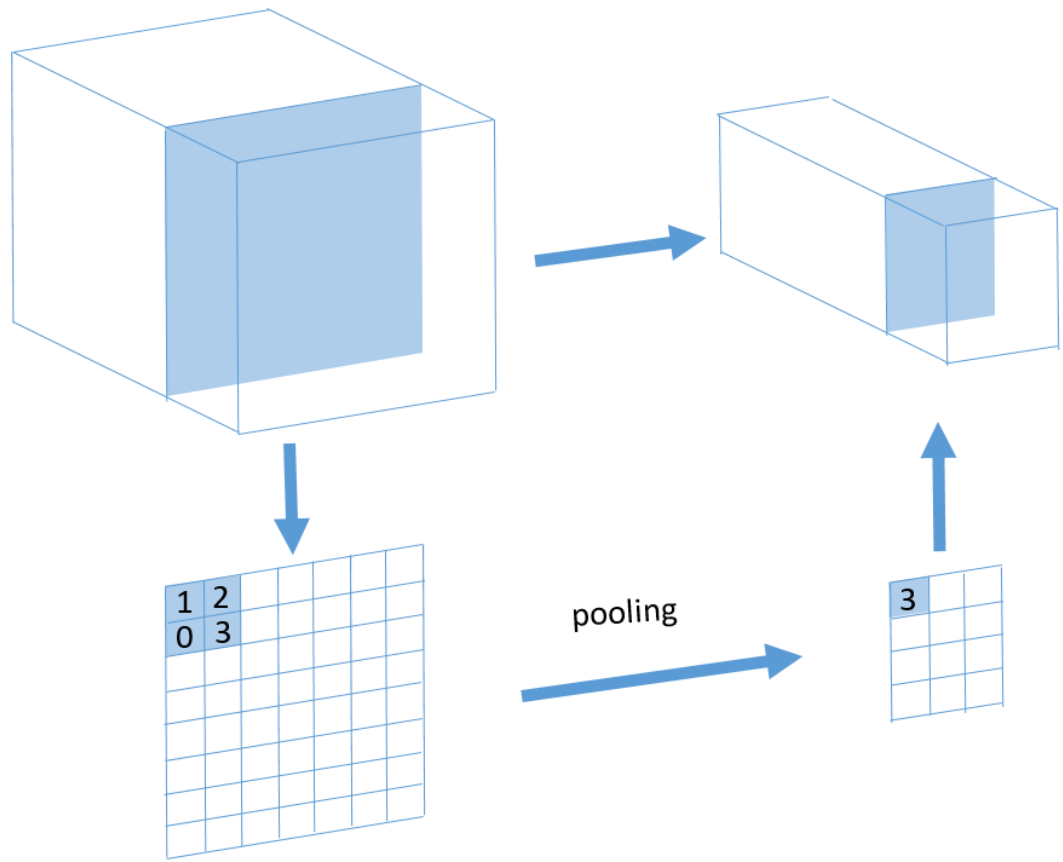
$$z_{ij}^{\ell} = g(s_{ij}^{\ell})$$

Each filter produces a 2D output known as a feature map. Stacking the feature maps produced by multiple filters leads to a 3D array.

Pooling

Pooling reduces the size of a 3D array.

Each channel is separately processed. First the channel is divided into non-overlapping cells (*e.g.*, $\Delta \times \Delta$). Then, each cell is replaced by a numeric value (*e.g.*, its maximum, or mean).



Pooling

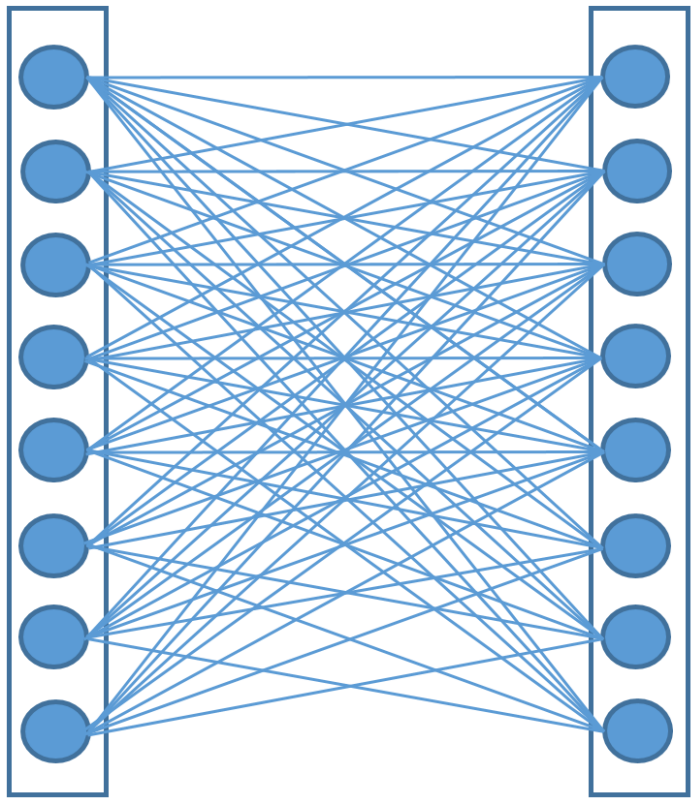
3D input: $z_{ijk}^{\ell-1}$ ℓ - number of input layer

3D output:

$$z_{ijk}^{\ell} = \max_{p,q \in \{0,\ldots,\Delta-1\}} \left\{ z_{\Delta i+p,\Delta j+q,k}^{\ell-1} \right\}$$

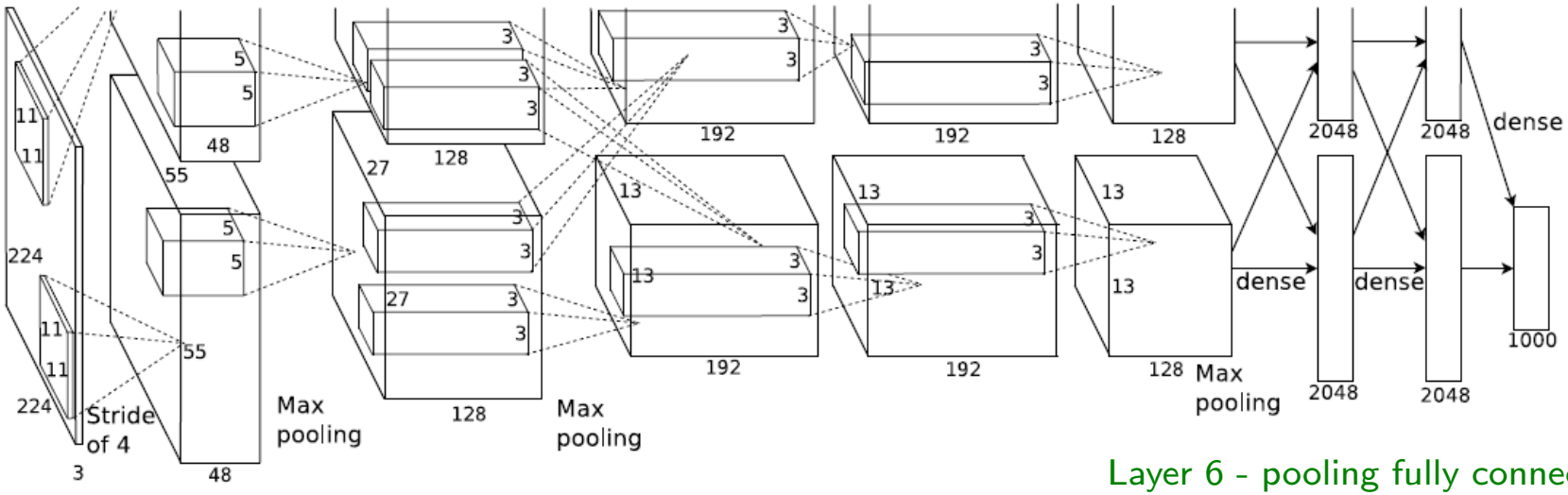
Fully connected layer

The fully connected layer is used when the image representation is converted into a 1D array.



It is often used as an output layer in classification problems.

Alexnet



Layer 1- convolutional

- maxpooling: No
- Input: $224 \times 224 \times 3$
- kernel: $96 \times 11 \times 11 \times 3$
- stride: 4
- units: $55 \times 55 \times 96$

Layer 2- max pooling followed by convolutional

- maxpooling: 2×2
- Input: $55 \times 55 \times 96$
- kernel: $256 \times 5 \times 5 \times 96$
- stride: 1
- units: $27 \times 27 \times 256$

Layer 3,4,5- similar

Layer 6 - pooling fully connected

- maxpooling: 2×2
- input: $13 \times 13 \times 256$
- units: 4096

Layer 7 - fully connected

- input: 4096
- units: 4096

Layer 8- fully connected

- input: 4096
- units: 1000

Alexnet

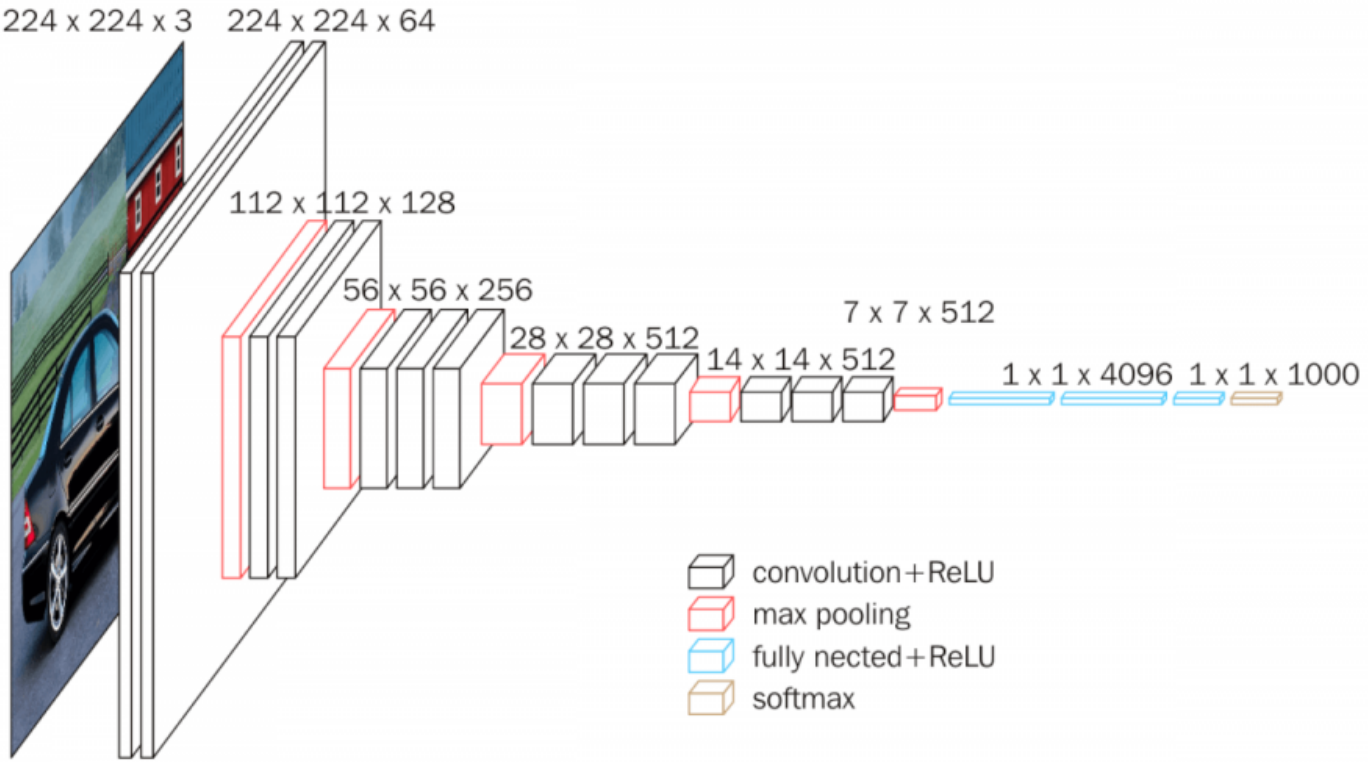
Number of weights to be learned

layer	expression	weights
1	$(11 \times 11) \times 3 \times 96$	0.03 M
2	$(5 \times 5) \times 96 \times 256$	0.6 M
3	$(3 \times 3) \times 256 \times 384$	0.8 M
4	$(3 \times 3) \times 384 \times 384$	1.3 M
5	$(3 \times 3) \times 384 \times 256$	0.8 M
6	$(6 \times 6) \times 256 \times 4096$	37.7 M
7	4096×4096	16.7M
8	4096×1000	4.1 M

The Alexnet has 60 million weights. Almost all of them associated to the last three layers (fully connected layers).

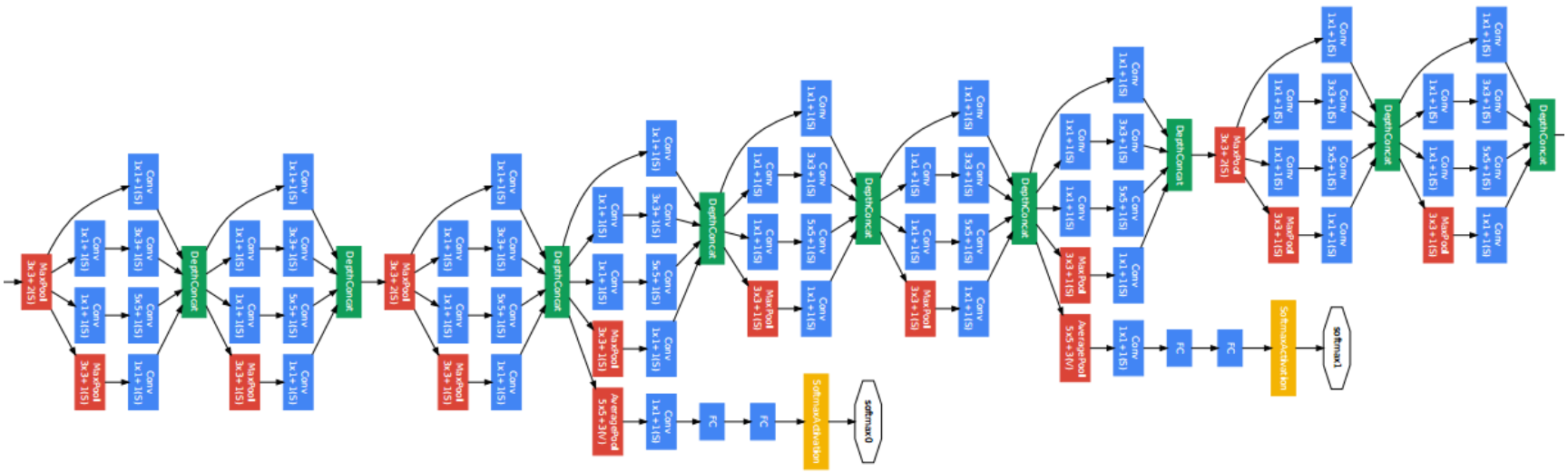
This high number of weights leads to overfitting problems in the training phase. Some kind of regularization must be considered.

Other convolutional neural networks: VGG



- deeper network
- deeper layers
- kernels: smaller spatial dimensions (3×3)

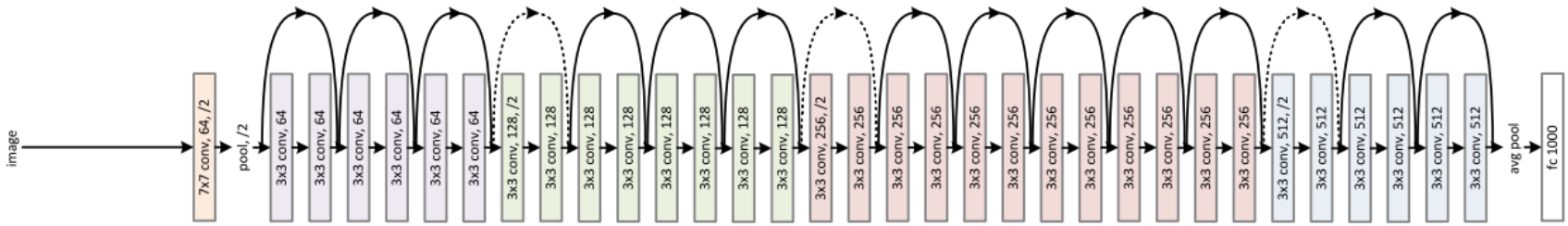
Other convolutional neural networks: GoogLeNet



convolution, pooling, softmax, merge

- inception module
- 1x1 convolution

Other convolutional neural networks: ResNet



- very deep network
- shortcut connections

Table of contents

Machine Learning

Linear Regression

Regularization

Evaluation & Generalization

Optimization

Neural networks

Data classification

Linear classifiers

Support vector machines

Decision Trees and Random Forest

What is a classifier?

Example: fish classification

length	volume	class
0.36	0.67	tuna
0.82	0.56	tuna
0.46	0.67	sword
0.40	0.30	sword
0.60	0.80	tuna
0.61	0.47	tuna
0.21	0.41	sword

Given an observation $x \in \mathbb{R}^d$, we wish to predict its class $y \in \Omega$, where $\Omega = \{\omega_0, \ldots, \omega_{K-1}\}$ or $\Omega = \{0, \ldots, K-1\}$.

K is the number of classes and the *ith* class will be denoted by ω_i or simply by i.

We wish to learn a function $f(x)$ that associates each feature vector $x \in \mathbb{R}^p$ with the predicted class $\hat{y} = f(x) \in \Omega$. This function is known as a classifier.

Discriminant functions

An alternative way to define a classifier is by using K functions

$$f_i : \mathbb{R}^p \rightarrow \mathbb{R} \quad , \; i = 0, \ldots, K-1$$

such that x is classified in class ω_i iif

$$f_i(x) \geq f_j(x) \; , \quad \forall j \neq i.$$

These functions $f_i(x)$ are called discriminant functions.

Note: if $f_i(x) = f_j(x)$ the classification is ambiguous.

Decision regions and decision boundary

A classifier $f(x)$ splits the input space $\mathbb{R}^d$ into K disjoint regions, $\mathcal{R}_j$, each of them associated to a specific class ω_j, with $j \in \{0, \ldots, K-1\}$.

$$\mathcal{R}_j = \left\{ x \in \mathbb{R}^d : f(x) = \omega_j \right\}.$$

These regions are known as decision regions.

The boundary points of these decision regions are called decision boundaries or decision surfaces.

Knowing the decision regions is equivalent to knowing the classifier $f(x)$, or a set of discriminant functions $f_i(x), i = 0, \ldots, K-1$. In fact, the indicator functions of the regions is a set of discriminant functions.

Classifier design

The main question is:

Given a classification problem, how do we define the classifier $f(x)$ or a set of discriminant functions $f_0(x), \ldots, f_{K-1}(x)$ or the decision regions $\mathcal{R}_0, \ldots, \mathcal{R}_{K-1}$?

The three representations are equivalent.

Two cases will be considered:

- we know the probability distribution of the data (ideal case);
- we only know a data set (training set).

Classifier evaluation - confusion matrix

The confusion matrix P is a $K \times K$ matrix whose generic element P_{ij} is the joint probability of true class i being predicted as class j.

$$P_{ij} = Pr\{y = i, \hat{y} = j\}$$

Properties:

$$P_{ij} \in [0, 1], \qquad \forall i, j$$

$$\sum_{i=0}^{K-1} \sum_{j=0}^{K-1} P_{ij} = 1.$$

Matrix P is a joint probability distribution: the diagonal elementos correspond to true decisions and the off diagonal elements correspond to errors.

If we normalize each line i to sum 1, the $i - th$ normalized line explains how the classifier predicts the data from class i: what is the probaility of error in class i and what errors are most probable.

Probability of error

The probability of error can be obtained from the confusion matrix P

$$P(\text{error}) = 1 - \sum_{i=0}^{K-1} P_{ii}$$

Proof

$$\begin{aligned} P(\text{error}) &= 1 - P(\text{correct decision}) \\ &= 1 - \textstyle\sum_{i=0}^{K-1} P(\text{correct decision}, y = i) \\ &= 1 - \textstyle\sum_{i=0}^{K-1} P(y = i, \hat{y} = i) \\ &= 1 - \textstyle\sum_{i=0}^{K-1} P_{ii} \end{aligned}$$

How to compute the confusion matrix

In simple cases, the confusion matrix can be analytically evaluated.

Assuming that a feature vector associated to class i is generated according to a pdf $p(x|y=i)$, the class j is chosen by the classifier if $x \in \mathcal{R}_j$,

$$P_{ij} = Pr\{y = i, x \in \mathcal{R}_j\} = \int_{\mathcal{R}_j} p(x|y=i)P(y=i)\, dx$$

When the integral cannot be evaluated, the confusion matrix may be experimentally obtained:

- perform N classification experiments;
- count how many training examples from class i are classified in class j (N_{ij});
- estimate P_{ij} using the relative frequency $\hat{P}_{ij} = \frac{N_{ij}}{\sum_{p=0}^{K-1}\sum_{q=0}^{K-1} N_{pq}}$.

Example (1)

Compute the confusion matrix and the probability of error, assuming that

- $x \in [0,1], y \in \{0,1\}$, $p(x|y=0) = 1, p(x|y=1) = 2x, \forall x \in [0,1]$;
- the classifier is characterized by the decision regions $\mathcal{R}_0 = [0, T[, \mathcal{R}_1 = [T, 1]$.

Confusion matrix

$$P_{00} = Pr(y=0, \hat{y}=0) = Pr(\hat{y}=0|y=0)Pr(y=0) =$$

$$= P_0 \int_{\mathcal{R}_0} p(x|y=0)dx = P_0 \int_0^T 1 \, dx = P_0 \, T,$$

$$P_{01} = P_0 - P_{00} = P_0(1-T),$$

$$P_{10} = Pr(y=1, \hat{y}=0) = Pr(\hat{y}=0|y=1)Pr(y=1) =$$

$$= P_1 \int_{\mathcal{R}_0} p(x|y=1)dx = P_1 \int_0^T 2x \, dx = P_1 T^2,$$

$$P_{11} = P_1 - P_{10} = P_1(1-T^2).$$

Example (2)

Confusion matrix

$$P = \begin{bmatrix} P_0\,T & P_0(1-T) \\ \\ P_1\,T^2 & P_1(1-T^2) \end{bmatrix}$$

Probability of error

$$P(\text{error}) = 1 - (P_{00} + P_{11}) = P_1\,T^2 - P_0\,T + 1 - P_1$$

The threshold T is chosen by the user. For example, it can be chosen by minimizing $P(\text{error})$.

Loss function

The confusion matrix is enough if all the errors are equally important and the classes are equally probable.

Sometimes this is not true. We need to define a loss function $L(y, \hat{y})$ that assigns a penalty when the true class of x is y and the predicted class is $\hat{y}$.

Examples:

$$\text{binary loss: } L(y, \hat{y}) = \begin{cases} 0 & \text{if } \hat{y} = y \\ 1 & \text{otherwise} \end{cases} .$$

$$\text{general loss: } L(y = \omega_i, \hat{y} = \omega_j) = L_{ij} , \quad L_{ii} = 0 , \quad L_{ij} > 0 , \; i \neq j.$$

The first case is a binary loss (no error/error).

The second case is a square $K \times K$ matrix of penalties with zeros in the diagonal (decisions without error) and different costs associated to the different types of errors.

Loss function (cont)

Example: medical diagnosis: ω_0 - no tumor , ω_1 - tumor

$$L = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad L = \begin{bmatrix} 0 & 1 \\ 5 & 0 \end{bmatrix}$$

Question: which loss is more appropriate for this problem?

The loss function is not differentiable. We cannot use optimization algorithms based on the gradient or Hessian matrix to reduce the loss :-(.

Is there an ideal classifier?

The answer is Yes, if x, y are realizations of random variables X, Y with known distribution and we wish to minimize the expected loss, also known as risk (ideal case of known distributions)

$$\mathcal{R} = E\left\{L(y, \hat{y}(x))\right\} \ .$$

If the loss is binary, the optimal classifier is given by

$$\hat{y} = \arg\max_{\omega \in \Omega} P(y = \omega | x) \ .$$

This is known as the Bayes classifier and chooses the class with greatest *a posteriori* probability (the most probable class, given the observations).

The Bayes classifier with binary loss is optimal in the sense that it minimizes the probability of decision error.

Is there an ideal classifier? (cont)

If we adopt a general loss function, the optimal classifier is also simple. We compute the expected cost of choosing class $\hat{y} = \omega$

$$c_\omega(x) = \sum_{y \in \Omega} L(y, \omega) P(y|x),$$

and choose the class with smallest cost, i.e., the feature vector x should be classified as follows

$$f(x) = \arg\min_{\omega \in \Omega} \; c_\omega(x).$$

This is an optimal classifier in the sense that minimizes the risk for a general loss function and it is also known as Bayes classifier.

Proof

Risk with general loss matrix

$$\mathcal{R} = E\left\{L(y, \hat{y}(x))\right\} = \int \sum_{y \in \Omega} L(y, \hat{y}(x)) p(x, y) dx$$

$$= \int \left[\sum_{y \in \Omega} L(y, \hat{y}(x)) P(y|x) \right] p(x) dx$$

Minimization can be independently performed at each feature vector x.

$$f(x) = \arg\min_{\omega \in \Omega} \left[\sum_{y \in \Omega} L(y, \omega) P(y|x) \right] = \arg\min_{\omega \in \Omega} c_\omega(x)$$

If the loss is binary we obtain,

$$f(x) = \arg\min_{\omega \in \Omega} \left[1 - P(\omega|x)\right] = \arg\max_{\omega \in \Omega} P(\omega|x)$$

a posteriori distribution of the classes

The *a posteriori* distribution of the classes $P(y = i|x)$ is the distribution of the classes after observing the feature vector x.

These probabilities can be obtained by using the Bayes law

$$P(y = i|x) = \frac{p(x|y = i)P(y = i)}{p(x)},$$

where

- $p(x|y = i)$ - distribution of the feature vector x associated to class i;
- $P(y = i)$ - *a priori* distribution of the classes (before knowing the observations);
- $p(x)$ - normalization term that does not influence the decision,

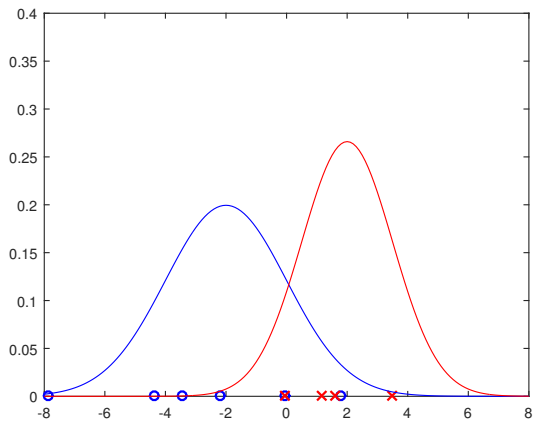
$$p(x) = \sum_{y \in \Omega} p(x|y)P(y).$$

Example: binary classification with Gaussian features (1d)

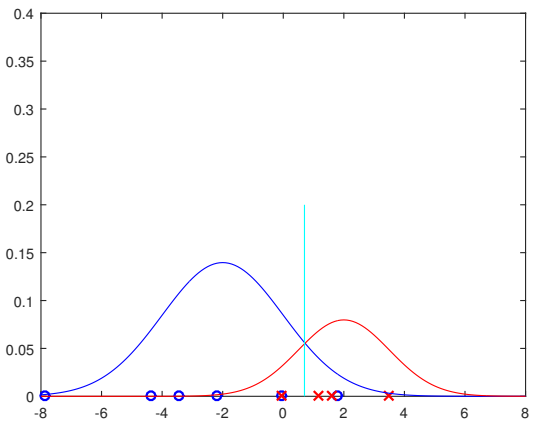
This example considers two classes ($K = 2$) with Gaussian 1d features: $x|\omega_i \sim N(\mu_i, \sigma_i^2),\ i = 0, 1$ ($\mu_0 = -2, \mu_1 = 2, \sigma_0^2 = 2, \sigma_1^2 = 1.5$).

A priori distribution: $P_0 = P(\omega_0) = 0.7, P_1 = P(\omega_1) = 0.3$.

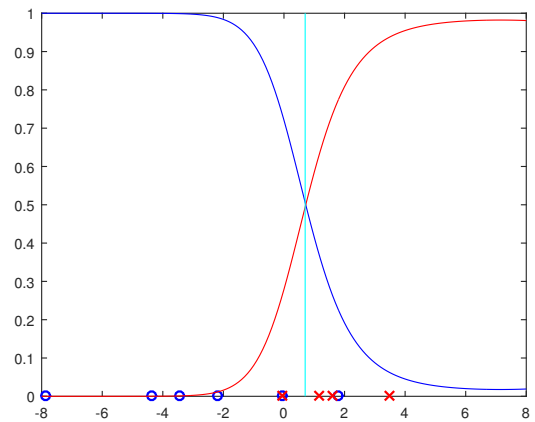
conditional distribution of data $p(x|y)$



joint distribution of data and classes $p(x, y)$



a posteriori distribution $P(y|x)$



Data classification can be obtained by maximizing the joint distribution of data and class $p(x, y)$ or the a posteriori distribution of classes $P(y|x)$, with respect to y.

Exercises

1. Consider a discrete feature variable $x \in \{1, 2, 3, 4\}$ and an associate binary class $y \in \{\omega_0, \omega_1\}$. Assume that y is characterized by the *a priori* distribution $P(\omega_0) = 0.4, P(\omega_1) = 0.6$ and the observations x are characterized by a conditional distribution defined by the table.

		ω	
	$p(x\|y)$	ω_0	ω_1
x	1	0.3	0.2
	2	0.2	0.3
	3	0.1	0.4
	4	0.4	0.1

Derive the Bayes classifier assuming a binary loss.

Exercises

2. Consider an observation $x \in \{0, \ldots, p\}$ generated by one of the two binomial distributions

$$P(x|\omega_i) = \begin{pmatrix} p \\ x \end{pmatrix} \alpha_i^x \, (1-\alpha_i)^{p-x} \qquad i = 0, 1,$$

where $p, \alpha_1 > \alpha_0$ are known parameters. Find the decision regions of the Bayes classifier, assuming a binary loss matrix and equally probable classes.

3. Assume that $x \in \mathbb{R}_0^+$ is a realization of a random variable with one of the following density functions

$$p(x|y=k) = \alpha_k e^{-\alpha_k x} \, , \qquad k = 0, 1, \quad \alpha_1 > \alpha_0 > 0.$$

Find the decision regions associated to both classes. Assume that $P_1 = 2 \, P_0$

Learning the classifier

In practice, we often do not know the joint distribution of the features and true class $p(x, y)$, required in the design of the Bayes classifier.

In many practical problems, all we know is a training set $\mathcal{T} = \left\{(x^{(i)}, y^{(i)}), i = 1, \ldots, n\right\}$ with n realizations of the pair X, Y.

We could learn a classifier by minimizing the empirical risk

$$\mathcal{R} = \frac{1}{n} \sum_{i=1}^{n} L(y^{(i)}, f(x^{(i)}))$$

However, this is a difficult approach because $y^{(i)}$ and $f(x^{(i)})$ are categorical variables and most optimization algorithms cannot be used.

Learning the classifier

Alternative approaches are required. Some classification techniques try to approximate the ideal (Bayes) classifier by estimating the *a posteriori* probabilities of the classes, $P(y|x)$, as a function of the feature vector x.

This can be done directly by proposing a class of functions for such probabilities or by estimating the data distribution $p(x|y = k)$, $k = 1, \ldots, K$ and applying the Bayes law.

Other methods try to directly estimate a set of discriminant functions without trying to estimate the data distribution which is considered to be a more difficult problem.

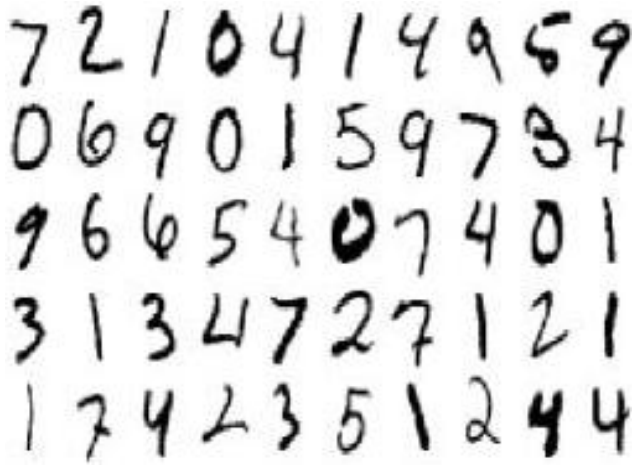
This approach is supported by the Vapnik principle.

Vapnik principle: When trying to solve a problem, we should not solve a more difficult problem as an intermediate step.

Example - digit recognition

Digit recognition aims to recognize handwritten digits in images, in an automatic way. It involves two steps:

- The first step consists of computing a bounding box for each digit with, *e.g.*, 20×20 pixels.
- The second step involves the classification of each 20×20 image.



examples from MNIST data set

If the feature vector, x, contains the intensity of 400 pixels, it is very difficult to estimate the conditional distribution $p(x|\omega_i) : \mathbb{R}^{400} \rightarrow \mathbb{R}$.

Naïve Bayes classifier

When the feature vector $\boldsymbol{x} = [x_1, \ldots, x_p]^T$ contains many features, the estimation of the conditional distribution $p(\boldsymbol{x}|y = k)$ is a difficult problem.

The Naïve Bayes classifier simplifies the problem by making a drastic assumption: it assumes that features are conditionally independent

$$p(x_1, \ldots, x_p|y = k) = \prod_{i=1}^{p} p(x_i|y = k)$$

This means that we only need to estimated the conditional distribution of each feature.

In the digit recognition problem this means that we have to estimate the conditional distribution of each pixel which is a simple task.

The Naïve Bayes classifier is a suboptimal classifier if the independence assumption is not true but it often leads to surprisingly good results.

Exercises

1. Draw a pair of scatter plots for two features (x_1, x_2), assuming that they are
   - dependent;
   - independent.

2. Discuss the problem of e-mail classification (spam/non-spam).

Table of contents

Machine Learning

Linear Regression

Regularization

Evaluation & Generalization

Optimization

Neural networks

Data classification

Linear classifiers

Support vector machines

Decision Trees and Random Forest

Linear methods for classification

We denote by linear classifiers those whose decision boundaries are linear (hyperplane) or piece-wise linear (hyperplane patches).

One example are the methods based on affine discriminant functions $f_i(\boldsymbol{x}) = [1\ \boldsymbol{x}^T]\beta_i$, where i stands for the class.

The decision boundary between two classes ω_i, ω_j is the set

$$\left\{\boldsymbol{x} \in \mathbb{R}^p : [1\ \boldsymbol{x}^T](\beta_i - \beta_j) = 0\right\},$$

which is (a subset of) an hyperplane in the feature space $\mathbb{R}^p$.

Class coding

Classification problems aim to predict a class label $y \in \{\omega_0, \ldots, \omega_{K-1}\}$.

Some classifiers represent the class label by numbers and use regression methods to predict those numbers.

One idea:

$$\begin{aligned} \omega_0 &\rightarrow 0 \\ \omega_1 &\rightarrow 1 \\ \omega_2 &\rightarrow 2 \end{aligned}$$

This does not make much sense because in most problems there is no natural order among the class labels.

A more interesting approach is the use of binary indicator variables:

	y_0	y_1	y_2
$\omega_0 \rightarrow$	1	0	0
$\omega_1 \rightarrow$	0	1	0
$\omega_2 \rightarrow$	0	0	1

One hot encoding

The indicator variable of class ω_i, is

$$y_i = \begin{cases} 1 & \text{if class } \omega_i \text{ occurs} \\ 0 & \text{otherwise} \end{cases}$$

The representation of the class labels through a set of indicator variables is known as one hot encoding.

Classification works as followed. In the training phase, a set of predictors $f_i(\boldsymbol{x})$ are learned to fit the indicator variables.

In the test phase, new feature vectors are classified by computing the predictors $f_i(\boldsymbol{x})$ and selecting the one with greatest value

$$\hat{y} = \arg\max_i f_i(\boldsymbol{x})$$

Training indicator variable with constant input

Consider n realizations of an indicator variable y associated to an arbitrary class ω: $y^{(1)}, \ldots, y^{(n)}$ ($\boldsymbol{x}^{(k)}$ is assumed constant).

Let us minimize the

$$SSE = \sum_{k=1}^{n} (y^{(k)} - \hat{y})^2$$

Since $y^{(k)}$ is a binary variable, the SSE can be split into two terms

$$SSE = n_0(0 - \hat{y})^2 + n_1(1 - \hat{y})^2$$

where n_0, n_1 is the number of $0s$ and $1s$. The minimization of SSE leads to

$$\hat{y} = \frac{n_1}{n_0 + n_1}$$

This means that the minimization of SSE leads to the estimation of the class probability $P(\omega)$. This idea can be extended as we will see.

Linear regression of indicator variables

Consider a binary classification problem with classes ω_0, ω_1 and let us assume that y is the indicator variable of class ω_1

We fit a linear model $f(\boldsymbol{x}) = [1\ \boldsymbol{x}^T]\beta$ to the training set $\mathcal{T} = \left\{(\boldsymbol{x}^{(i)}, y^{(i)}), i = 1, \ldots, n\right\}$ using least squares.

The function $f(\boldsymbol{x})$ can be considered as an estimate of the *a posteriori distribution* of class ω_1. Since $f(\boldsymbol{x})$ is linear, it takes values outside the interval $[0, 1]$.

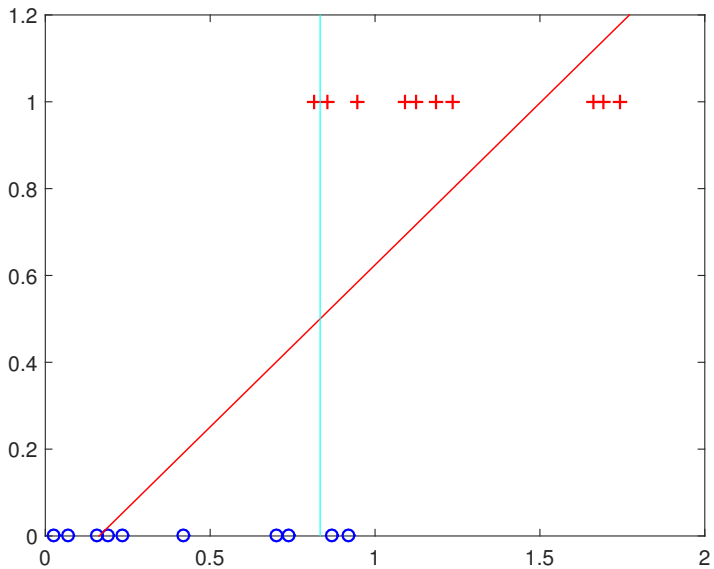
After training the model, a new observation $\boldsymbol{x}$ can be classified by comparing $f(\boldsymbol{x})$ with a threshold 0.5

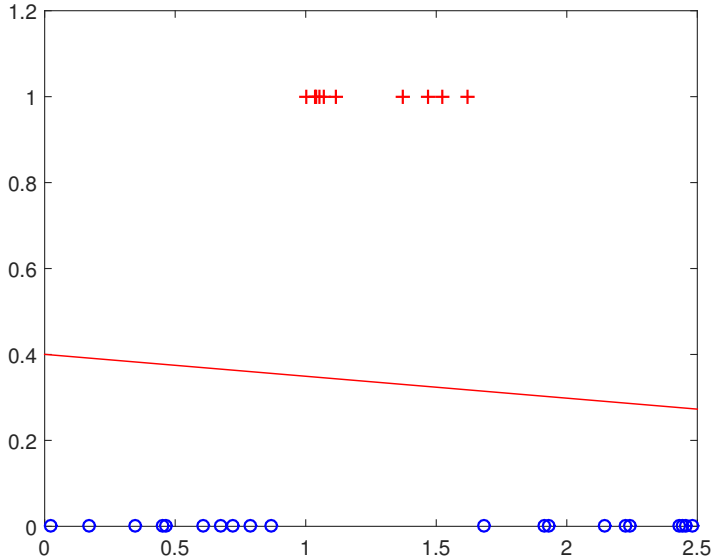
$$\hat{y} = \begin{cases} 1 & \text{if } f(\boldsymbol{x}) > 0.5 \\ 0 & \text{otherwise} \end{cases} .$$

Example - 1D data

This example discusses two binary classification problems with 1D features for which we know a training set (see figures). The data was fit by a straight line and we display the decision boundary (cyan).

The first problem is well solved by linear regression of the indicator variables. The second is not: all the features are classified in the same class. Why?

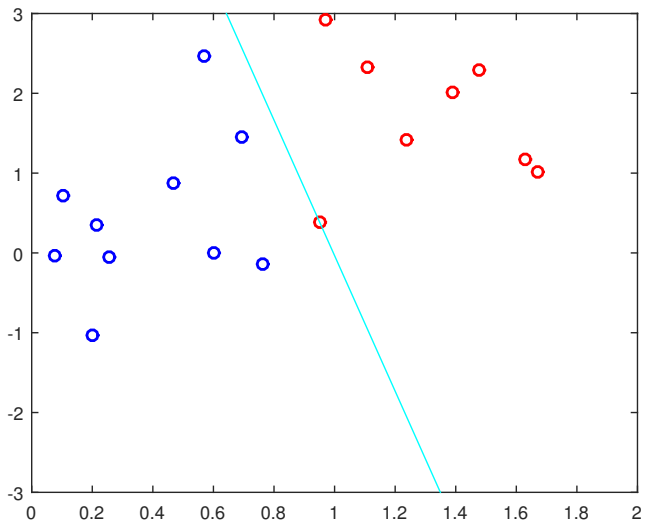


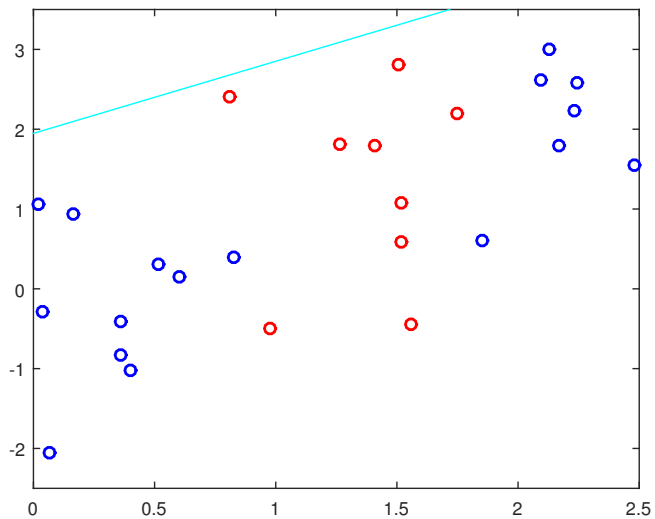


Why does the linear regression with indicator variables fail in the second example?

Example - 2D data

This slide shows two problems with 2D features and a linear model. Only the first problem can be solved by linear models. Why?

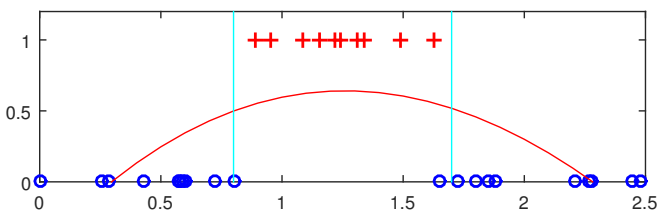


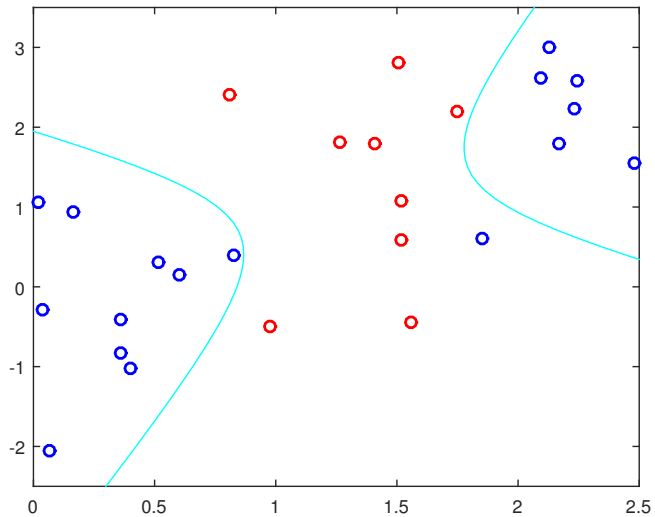


In the second case all the training features are classified in the same class.

Regression with more flexible models

The previous difficulties can be circumvented by using more flexible models (*e.g.* 2nd order polynomials in $\mathbb{R}$ and $\mathbb{R}^2$).





Notice, though that these are not linear models with respect to $\boldsymbol{x}$ but they are linear models in the parameters that can be estimated by a linear system of equations.

Drawbacks & extensions

The regressor function $f(x)$ can be interpreted as an estimate of the a posteriori probabilty $P(\omega_1|\boldsymbol{x})$ but it is not constrained to be in the interval $[0, 1]$. Since the model is linear, it will take all real values.

The decision boundary between two classes is hyperplane. Therefore, the technique can only be used if the data is well separated by a hyperplane.

The model can be easily extended to more flexible classes of functions *e.g.*, polynomials, radial basis functions, neural networks.

This approach can be easily extended to more than 2 classes by considering K indicator variables (one per class) and fit a linear model to predict these labels (one vs. all). This is known as one hot encoding. Label prediction is performed by choosing the discriminant function with the highest value

$$\hat{y}(\boldsymbol{x}) = \arg\max_i f_i(\boldsymbol{x}).$$

Logistic regression

Consider a binary classification problem in which $y \in \{0, 1\}$. The Bayes classifier is based on the *a posteriori* distribution of the classes

$$P(y = 1|\boldsymbol{x}), \quad P(y = 0|\boldsymbol{x}).$$

The logistic regression proposes a parametric model for the *a posteriori* probabilities

$$P(y = 1|\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{x}^T \boldsymbol{\beta}}}, \qquad P(y = 0|x) = \frac{e^{-\boldsymbol{x}^T \boldsymbol{\beta}}}{1 + e^{-\boldsymbol{x}^T \boldsymbol{\beta}}}.$$

Where $\boldsymbol{x} \in \mathbb{R}^{p+1}$ is the feature vector and $\boldsymbol{\beta} \in \mathbb{R}^{p+1}$ the vector of parameters to be estimated. We have included β_0 in vector $\boldsymbol{\beta}$ and extended the feature vector $\boldsymbol{x}$ with 1.

Logistic regression

This model guarantees that

$$P(y = 1|\boldsymbol{x}), P(y = 0|\boldsymbol{x}) \in [0, 1]$$

$$P(y = 0|\boldsymbol{x}) + P(y = 1|\boldsymbol{x}) = 1.$$

It can be rewritten as follows

$$P(y = 1|\boldsymbol{x}) = g(\boldsymbol{x}^T \beta) \ , \quad P(y = 0|\boldsymbol{x}) = 1 - g(\boldsymbol{x}^T \beta)$$

where

$$g(s) = \frac{1}{1 + e^{-s}}$$

What is the relationship between the logistic regression and a perceptron unit?

What is the meaning of the perceptron output $\hat{y}$, in this context?

Logistic regression: learning

Given a training set $\mathcal{T} = \left\{(\boldsymbol{x}^{(i)}, y^{(i)}), i = 1, \ldots, n\right\}$, the coefficients $\boldsymbol{\beta}$ can be estimated by the maximum likelihood method

$$\hat{\boldsymbol{\beta}} = \arg\max_{\boldsymbol{\beta}} \ell(\boldsymbol{\beta}) \ ,$$

where $\ell(\boldsymbol{\beta})$ is the conditional log-likelihood function

$$\ell(\boldsymbol{\beta}) = \log P(y^{(1)}, \ldots, y^{(n)} | x^{(1)}, \ldots, x^{(n)}; \boldsymbol{\beta}) \ .$$

Since the training examples are independent

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \log P(y^{(i)} | \boldsymbol{x}^{(i)}) \ .$$

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left\{ y^{(i)} \log[g(\boldsymbol{x}^{(i)\,T} \boldsymbol{\beta})] + (1 - y^{(i)}) \log[1 - g(\boldsymbol{x}^{(i)\,T} \boldsymbol{\beta})] \right\} \ .$$

This function cannot be analytically optimized. We have to use numerical optimization algorithms *e.g.*, gradient ascent method.

Logistic regression - gradient ascent

The gradient of the conditional log-likelihood function $\ell(\beta)$ can be easily computed

$$\nabla_{\boldsymbol{\beta}} \ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} [y^{(i)} - g(\boldsymbol{x}^{(i)T} \boldsymbol{\beta})] \boldsymbol{x}^{(i)} \ .$$

Therefore, the gradient ascent algorithm is given by

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + \gamma \nabla_{\boldsymbol{\beta}} \ell(\boldsymbol{\beta}^{(t)}) \ , \qquad \gamma > 0 \ .$$

Question: is it possible to train the logistic regressor using the SSE criterion?

Log-likelihood gradient

Proof for 1 pattern (drop variable i)

$$\nabla_{\boldsymbol{\beta}} \ell(\beta) = \nabla_{\boldsymbol{\beta}} \left\{ y \log g(\boldsymbol{x}^T \boldsymbol{\beta}) + (1-y) \log[1 - g(\boldsymbol{x}^T \boldsymbol{\beta})] \right\}$$

$$= y \, \frac{g'(\boldsymbol{x}^T \boldsymbol{\beta})}{g(\boldsymbol{x}^T \boldsymbol{\beta})} \, \boldsymbol{x} + (1-y) \, \frac{-g'(\boldsymbol{x}^T \boldsymbol{\beta})}{[1-g(\boldsymbol{x}^T \boldsymbol{\beta})]} \, \boldsymbol{x}$$

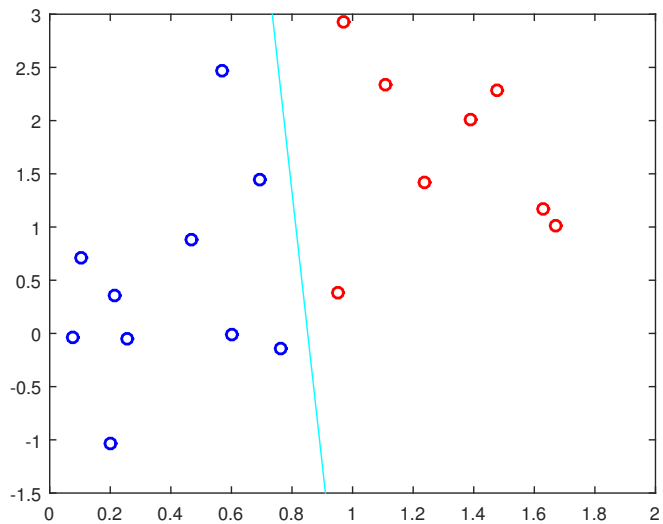
$$= y \, \frac{g(\boldsymbol{x}^T \boldsymbol{\beta}) \, [1-g(\boldsymbol{x}^T \boldsymbol{\beta})]}{g(\boldsymbol{x}^T \boldsymbol{\beta})} \, \boldsymbol{x} + (1-y) \, \frac{-g(\boldsymbol{x}^T \boldsymbol{\beta}) \, [1-g(\boldsymbol{x}^T \boldsymbol{\beta})]}{[1-g(\boldsymbol{x}^T \boldsymbol{\beta})]} \, \boldsymbol{x}$$

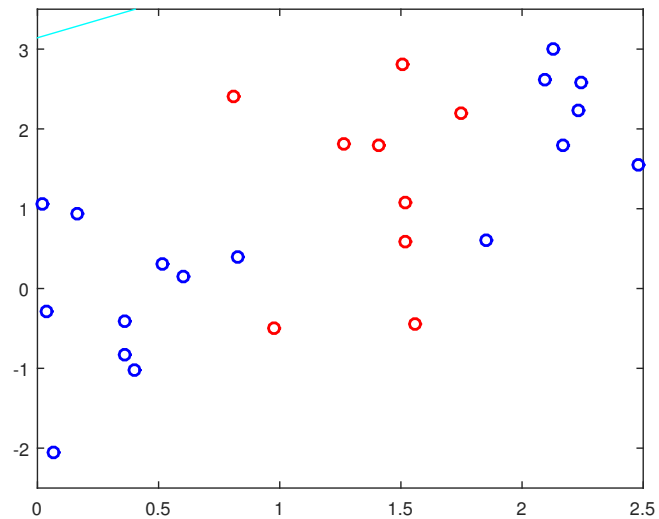
$$= y \, [1 - g(\boldsymbol{x}^T \boldsymbol{\beta})] \, \boldsymbol{x} - (1-y) \, g(\boldsymbol{x}^T \boldsymbol{\beta}) \, \boldsymbol{x}$$

$$= [y - g(\boldsymbol{x}^T \boldsymbol{\beta})] \, \boldsymbol{x}$$

Example - logistic regression with 2D data

Consider two classification problems with 2D features described before. Figures show the decision boundaries obtained by logistic regression. Only the first problem can be solved by linear models.

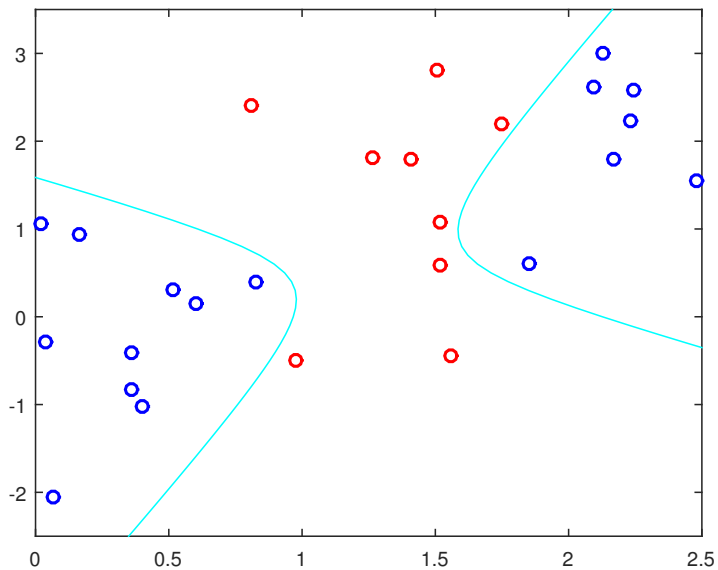




In the second case all the training features are classified in the same class. The model is too rigid.

Logistic regression with more flexible models

The previous difficulties can be circumvented by using more flexible models (*e.g.*, 2nd order polynomials).



Notice that these models are not linear models with respect to $\boldsymbol{x}$.

Softmax

Softmax extends logistic regression to classification problems with an arbitrary number of classes K.
The true class is expressed using indicator variables, a.k.a. one hot encoding

$$y = (y_0, \ldots, y_{K-1}), \quad y_i \in \{0, 1\}, \quad \sum_{i=0}^{K-1} y_i = 1$$

Softmax proposes a model for the *a posteriori* probabilities of the classes

$$\hat{y}_i = P(y_i = 1 | \boldsymbol{x}, \boldsymbol{\beta}) = \frac{e^{s_i}}{\sum_{c=0}^{K-1} e^{s_c}}$$

where

$$s_i = \sum_{j=0}^{p} \beta_{ji} x_j$$

This model guarantees that $P(y_i = 1 | \boldsymbol{x}, \boldsymbol{\beta}) \in [0, 1],\ i = 0, \ldots, K-1$ and $\sum_{i=0}^{K-1} P(y_i = 1 | \boldsymbol{x}, \boldsymbol{\beta}) = 1$.

Softmax: learning

Given a training set $\mathcal{T} = \left\{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(n)}, y^{(n)})\right\}$, the coefficients $\boldsymbol{\beta}$ can be estimated by the maximum likelihood method

$$\hat{\boldsymbol{\beta}} = \arg\max_{\boldsymbol{\beta}} \ell(\boldsymbol{\beta}) \ ,$$

where $\ell(\boldsymbol{\beta})$ is the conditional log-likelihood function

$$\ell(\boldsymbol{\beta}) = \log P(y^{(1)}, \ldots, y^{(n)} | \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}; \boldsymbol{\beta}) \ .$$

Since the training examples are independent

$$\ell(\boldsymbol{\beta}) = \sum_{m=1}^{n} \log P(y^{(m)} | \boldsymbol{x}^{(m)}; \boldsymbol{\beta}) = \sum_{m=1}^{n} \sum_{i=0}^{K-1} \left\{ y_i^{(m)} \log(\hat{y}_i^{(m)}) \right\} \ .$$

This function cannot be analytically optimized. We must resort to numerical optimization algorithms *e.g.*, gradient ascent method method.

Softmax: gradient

log-likelihood (1 training example): $\ell = \sum_{i=0}^{K-1} y_i \log(\hat{y}_i)$

Derivatives:

$$\frac{\partial \ell}{\partial \hat{y}_i} = \frac{y_i}{\hat{y}_i}$$

$$\frac{\partial \hat{y}_i}{\partial s_k} = \begin{cases} \hat{y}_i(1-\hat{y}_i) & i = k \\ -\hat{y}_i \hat{y}_k & i \neq k \end{cases}$$

$$\frac{\partial \ell}{\partial s_i} = \sum_{k=1}^{K} \frac{\partial \ell}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial \hat{s}_i} = \frac{\partial \ell}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial \hat{s}_i} + \sum_{k \neq i} \frac{\partial \ell}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial \hat{s}_i}$$

$$= \frac{y_i}{\hat{y}_i} \hat{y}_i(1-\hat{y}_i) - \sum_{k \neq i} \frac{y_k}{\hat{y}_k} (\hat{y}_k \hat{y}_i) = y_i - \hat{y}_i$$

$$\frac{\partial \ell}{\partial \beta_{ij}} = \frac{\partial \ell}{\partial s_i} \frac{\partial s_i}{\partial \beta_{ij}} = (y_i - \hat{y}_i) x_j$$

Softmax: update

The previous expressions can be extended to multiple training examples.

log-likelihood (multiple examples):

$$\ell(\beta) = \sum_{m=1}^{n} \sum_{i=0}^{K-1} y_i^{(m)} \log(\hat{y}_i^{(m)})$$

gradient:

$$\frac{\partial \ell}{\partial \beta_{ij}} = \sum_{m=1}^{n} (y_i^{(m)} - \hat{y}_i^{(m)}) x_j^{(m)}$$

Therefore, the gradient ascent algorithm is given by

$$\beta_{ij}^{(t+1)} = \beta_{ij}^{(t)} + \gamma \frac{\partial \ell}{\partial \beta_{ij}} , \qquad \gamma > 0 .$$

Linear discriminant analysis

Let us consider a binary classification problem. The *a posteriori* probabilities can be obtained from the Bayes law

$$P(y=i|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|y=i)P(y=i)}{p(\boldsymbol{x})} \propto p(\boldsymbol{x}|y=i)P_i \ ,$$

where the *a priori* probabilities P_i are approximated by the relative frequencies of each class in the training set. The main challenge is the estimation of $p(\boldsymbol{x}|y=i)$.

Linear discriminant analysis assumes that the distribution of the observations $\boldsymbol{x} \in \mathbb{R}^p$ associated to class $y=i$ follows a normal distribution $\mathcal{N}(\boldsymbol{\mu}_i, \Sigma_i)$ and the data from all classes share the same covariance matrix $\Sigma_i = \Sigma$. Therefore,

$$p(\boldsymbol{x}|y=i) = C \ e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T \Sigma^{-1} (\boldsymbol{x}-\boldsymbol{\mu}_i)} \ ,$$

where $\boldsymbol{\mu}_i \in \mathbb{R}^p$, $\Sigma = \mathbb{R}^{p \times p}$ and $C = 1/\left((2\pi)^{p/2}|\Sigma|^{1/2}\right)$ is a normalization constant.

Linear discriminant analysis

Under these hypotheses, the decision boundary between classes i, j is given by

$$p(\boldsymbol{x}|y=i)P_i = p(\boldsymbol{x}|y=j)P_j$$

$$Ce^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i)}P_i = Ce^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j)}P_j.$$

The constants C are equal because the covariance matrices are the same. Taking logs leads to

$$(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T\Sigma^{-1}\boldsymbol{x} = \frac{1}{2}(\boldsymbol{\mu}_i + \boldsymbol{\mu}_j)^T\Sigma^{-1}(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j) + \log\frac{P_j}{P_i}.$$

This is the equation of an hyperplane with normal vector $(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)^T\Sigma^{-1}$.

Linear discriminant analysis

Proof

$$\log[p(\boldsymbol{x}|y=i)P_i] = \log[p(\boldsymbol{x}|y=j)P_j]$$

$$\log C - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) + \log P_i = \log C - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_j)^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j) + \log P_j$$

$$(\boldsymbol{x}-\boldsymbol{\mu}_i)^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_i) - 2\log P_i = (\boldsymbol{x}-\boldsymbol{\mu}_j)^T\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_j) - 2\log P_j$$

$$-2\boldsymbol{\mu}_i^T\Sigma^{-1}\boldsymbol{x} + \boldsymbol{\mu}_i^T\Sigma^{-1}\boldsymbol{\mu}_i = -2\boldsymbol{\mu}_j^T\Sigma^{-1}\boldsymbol{x} + \boldsymbol{\mu}_j^T\Sigma^{-1}\boldsymbol{\mu}_j - 2\log\frac{P_j}{P_i}$$

$$(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^T\Sigma^{-1}\boldsymbol{x} = \frac{1}{2}\boldsymbol{\mu}_i^T\Sigma^{-1}\boldsymbol{\mu}_i - \frac{1}{2}\boldsymbol{\mu}_j^T\Sigma^{-1}\boldsymbol{\mu}_j + \log\frac{P_j}{P_i}$$

$$(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j)^T\Sigma^{-1}\boldsymbol{x} = \frac{1}{2}(\boldsymbol{\mu}_i+\boldsymbol{\mu}_j)^T\Sigma^{-1}(\boldsymbol{\mu}_i-\boldsymbol{\mu}_j) + \log\frac{P_j}{P_i}$$

Linear discriminant analysis

In practice, the parameters of the Gaussian distributions are learned from the training data

$$\hat{P}(\omega_k) = \frac{n_k}{n}$$

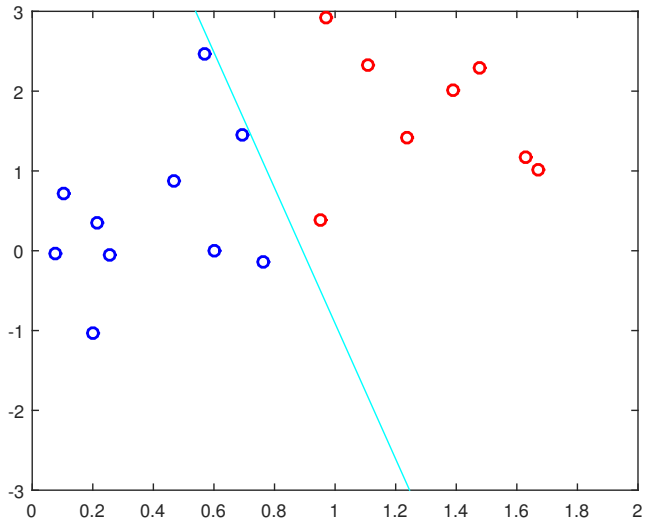
$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k} \sum_{i:y^{(i)}=k} \boldsymbol{x}^{(i)}$$

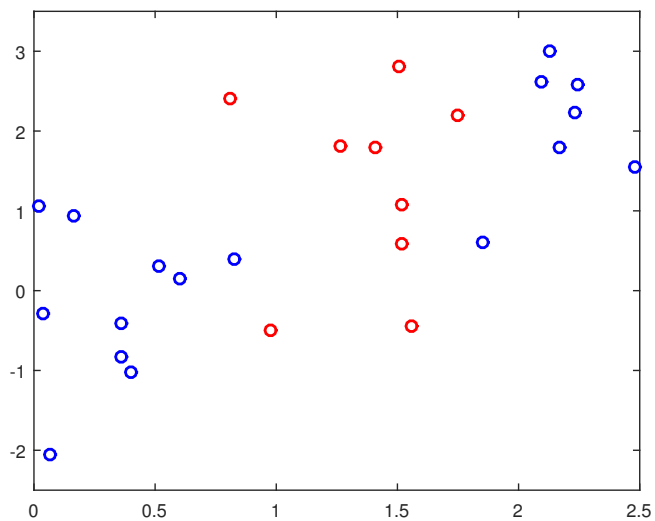
$$\hat{\Sigma} = \frac{1}{n-K} \sum_{k=1}^{K} \left[\sum_{i:y^{(i)}=k} (\boldsymbol{x}^{(i)} - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}^{(i)} - \hat{\boldsymbol{\mu}}_k)^T \right] .$$

The covariance matrix has d^2 entries. If d is large (hundreds or thousands) , $\hat{\Sigma}$ may be inaccurate and singular. Its inverse does not exist and it is required in LDA. It is common practice to enforce additional constraints on Σ *e.g.*, assume Σ is a diagonal matrix.

Example: LDA with 2D data

Consider two classification problems with 2D features described before. Only the first can be solved by LDA.





In the second case all the training features are classified in the same class. The model is too rigid.

Table of contents

Machine Learning

Linear Regression

Regularization

Evaluation & Generalization

Optimization

Neural networks

Data classification

Linear classifiers

Support vector machines

Decision Trees and Random Forest

Support vector machines

Support vector machines (SVMs) were proposed by Vapnik and Chervonenkis in 1963 for binary classification problems with linear decision boundaries. They were extended later for nonlinear decision boundaries and for regression problems.

Main idea: separate the cloud of data in two regions, using a carefully chosen hyperplane.

The SVM classifiers are often described in three steps:

- linear classifiers with hard margin
- linear classifiers with soft margin
- non linear classifiers

We follow: Fletcher, Support Vector Machines Explained, UCL report, 2008.

Hyperplanes

How do we define a hyperplane in $\mathbb{R}^p$?

$$\boldsymbol{x} \cdot \boldsymbol{w} + b = 0$$

where

- $\boldsymbol{x} \in \mathbb{R}^p$ - point on the hyperplane
- $\boldsymbol{w} \in \mathbb{R}^p$ - normal vector to the hyperplane
- $b \in \mathbb{R}$ - offset
- $\boldsymbol{x} \cdot \boldsymbol{w}$ is the inner product between $\boldsymbol{x}, \boldsymbol{w} \in \mathbb{R}^p$

distance to the origin: $\frac{|b|}{\|\boldsymbol{w}\|}$

note: parameters $\boldsymbol{w}, b$ are defined up to a scale factor (requires some kind of normalization to become unique).

Linear classifiers

A linear classifier compares each input vector with an hyperplane decision boundary

$$\boldsymbol{x} \cdot \boldsymbol{w} + b > 0 \Rightarrow \hat{y} = +1$$

$$\boldsymbol{x} \cdot \boldsymbol{w} + b < 0 \Rightarrow \hat{y} = -1.$$

For the sake of simplicity we assume that the binary label $y \in \{-1, +1\}$.

Therefore,

$$\hat{y} = \text{sign}(\boldsymbol{x} \cdot \boldsymbol{w} + b)\,.$$

Main question: how do we learn the hyperplane parameters from the training data?

Case I: linearly separable data

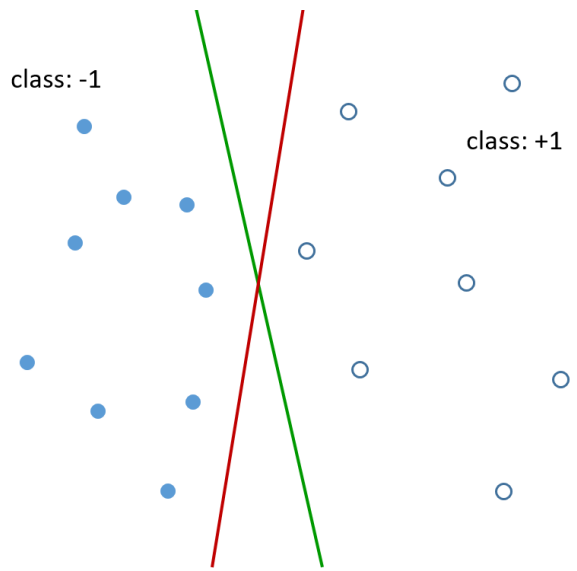
Training set

$$\mathcal{T} = \left\{ (\boldsymbol{x}^{(i)}, y^{(i)}), i = 1, \ldots, n \right\} \qquad \text{with } \boldsymbol{x}^{(i)} \in \mathbb{R}^p,\ y^{(i)} \in \{-1, 1\}.$$

Hypothesis: we assume that the training data can be separated by an hyperplane without errors (linearly separable data). But, if there is one hyperplane, there is an infinite number of hyperplanes that separate the data.

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0.$$

Problem: which hyperplane should we choose?

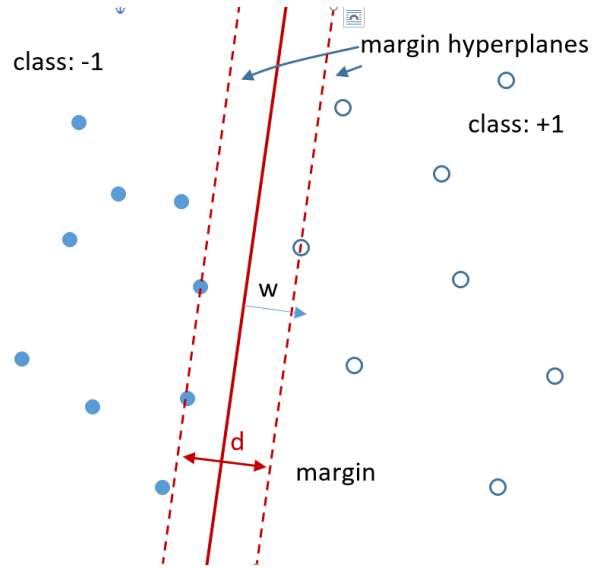


Hard margin

Consider a hyperplane that separates the training data without errors and is equally distant to the nearest examples of both classes.

The training points closest to the hyperplane are called support vectors.

The sum of two distances from the support vectors of each class to the decision hyperplane is the margin.



The hyperplanes parallel to the decision hyperplane, that contain the support vectors are known as margin hyperplanes.

Margin hyperplanes

Margin hyperplanes: training data on the margin hyperplanes verify

$$\left.\begin{array}{ll} \boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b = +1 & \text{for } y^{(i)} = +1 \\ \boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b = -1 & \text{for } y^{(i)} = -1 \end{array}\right\}.$$

Right hand side values ± 1 correspond to a specific choice of the scale factor that multiplies $(\boldsymbol{w}, b)$.

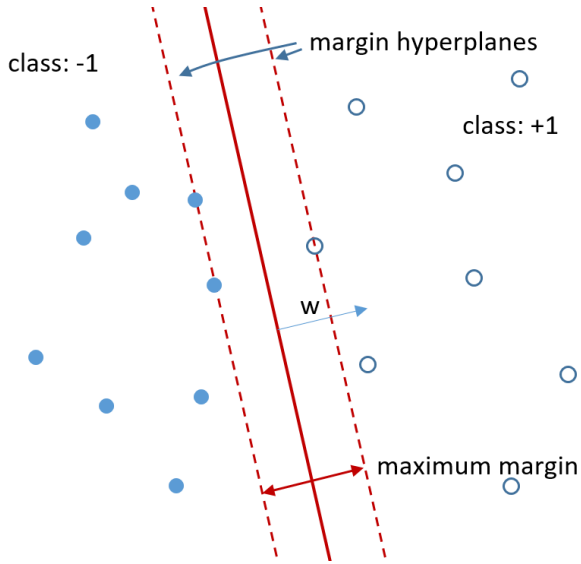
Margin: $\frac{2}{\|\boldsymbol{w}\|}$.

Constrains: training data must obey

$$\left.\begin{array}{ll} \boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b \geq +1 & \text{for } y^{(i)} = +1 \\ \boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b \leq -1 & \text{for } y^{(i)} = -1 \end{array}\right\} \Rightarrow\ y^{(i)}(\boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b) - 1 \geq 0,\ \forall i.$$

Maximum margin classifier

The SVM classifier chooses the hyperplane with the maximum margin (maximum margin classifier).



Difficulty:

The decision hyperplane is computed from the support vectors.

However, initially we do not know the support vectors. Their selection requires the decision hyperplane.

Question: how do we break this tie?

Exercise

Consider the following data sets.

x_1	x_2	y
0	3	-1
0	-3	-1
4	1	-1
4	-2	-1
0	0	$+1$
0	-2	$+1$
1	1	$+1$

x_1	x_2	y
0	9	-1
4	1	-1
4	4	-1
0	0	$+1$
0	4	$+1$
1	1	$+1$

For each of them,

- plot the data,
- find by inspection if it is linearly separable,
- find the support vectors and margin hyperplanes
- find the margin

Exercise: solution

First data set: it is non-separable and cannot be separated by an hyperplane.

Second data set: it is separable by an hyperplane.

Support vectors:

$$\begin{array}{ll} \text{class -1}: & (0,9),(4,1) \\ \text{class } +1: & (0,4) \end{array}$$

Margin hyperplanes:

$$\begin{bmatrix} 0 & 9 & 1 \\ 4 & 1 & 1 \\ 0 & 4 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ +1 \end{bmatrix} \qquad \boldsymbol{x} = -\frac{1}{5}\begin{bmatrix} 4 \\ 2 \end{bmatrix}, \qquad b = \frac{13}{5}.$$

Margin: $\frac{2}{\|\boldsymbol{w}\|} = \sqrt{5}$

Optimization problem (hard margin)

We break the tie by solving an optimization problem.

We wish to maximize the margin $(2/\|w\|)$ under the constrains described above. This can leads to the following optimization problems

Optimization problem 1: $\min \|w\|$, s.t. $y^{(i)}(\boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b) - 1 \geq 0, \ \forall i.$

Optimization problem 2: $\min \frac{1}{2}\|\boldsymbol{w}\|^2$, s.t. $y^{(i)}(\boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b) - 1 \geq 0, \ \forall i.$

This is a quadratic optimization problem with linear constraints.

Lagrangian formulation (primary)

Let us adopt a Lagrangian formulation in order to deal with the constraints on the training points.

Lagrangian function

$$L_P = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i [y^{(i)}(\boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b) - 1] \,,$$

$$L_P = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i y^{(i)}(\boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b) + \sum_{i=1}^{n} \alpha_i \,,$$

where $\alpha_i \geq 0$ are Lagrange multipliers.

$\boldsymbol{w}, b$ should be chosen to minimize L_P, and α_i to maximize it. This is known as the primary Lagrangian problem.

Lagrangian formulation (dual)

Optimization

$$\frac{\partial L_P}{\partial \boldsymbol{w}} = 0 \Rightarrow \boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)} \ ,$$

The normal vector $\boldsymbol{w}$ is obtained by a linear combination of the training patterns and only the support vectors contribute.

$$\frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \ .$$

Lagrangian formulation (dual)

Replacing these variables, we obtain the dual formulation,

$$L_D = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i y^{(i)} (x^{(i)} \cdot x^{(j)}) y^{(j)} \alpha_j \ , \quad s.t. \quad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \ ,$$

where $\alpha_i \geq 0$ are Lagrange multipliers.

The dual formulation depends only on the inner products between input vectors $x^{(i)} \cdot x^{(j)}$. This is very important!

Proof

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i \left[y^{(i)}(\boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b) - 1 \right]$$

$$L_p = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)} \cdot \boldsymbol{w} - b \sum_{i=1}^{n} \alpha_i y^{(i)} + \sum_{i=1}^{n} \alpha_i$$

Since $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)}$, $\sum_{i=1}^{n} \alpha_i y^{(i)} = 0$

$$L_p = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\|\boldsymbol{w}\|^2$$

$$L_p = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \left(\sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)} \right) \cdot \left(\sum_{j=1}^{n} \alpha_j y^{(j)} \boldsymbol{x}^{(j)} \right)$$

$$L_p = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i y^{(i)} (\boldsymbol{x}^{(i)} \cdot \boldsymbol{x}^{(j)}) y^{(j)} \alpha_j$$

Dual Lagrangian problem

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\alpha^T H \alpha \ , \quad s.t. \quad \alpha_i \geq 0 \ \ \forall \ i \ , \ \ \sum_{i=1}^{n} \alpha_i y^{(i)} = 0 \ ,$$

where $\alpha = [\alpha_1 \ \ldots \ \alpha_n]^T$ and $H_{ij} = y^{(i)}(x^{(i)} \cdot x^{(j)})y^{(j)}$.

This is a convex quadratic programming (QP) problem that can be solved by standard QP algorithms and provides all α_i.

Knowing α's we may obtain:

1. support vectors (S): all $\boldsymbol{x}^{(s)}$ such that $\alpha_s > 0$,
2. normal vector: $\boldsymbol{w} = \sum_{s \in S} \alpha_s y^{(s)} \boldsymbol{x}^{(s)}$
3. offset: $b = \frac{1}{N_s} \sum_{s \in S} \left[y^{(s)} - \sum_{m \in S} \alpha_m y^{(m)} (\boldsymbol{x}^{(m)} \cdot \boldsymbol{x}^{(s)}) \right]$
4. Classification of data: $f(\boldsymbol{x}) = sign(\boldsymbol{x} \cdot \boldsymbol{w} + b)$

S is the set of support vector indices.

Comments

We note that only support vectors contribute to the estimation of $\boldsymbol{w}, b$.

Matrix H does not require the training patterns themselves, $\boldsymbol{x}^{(i)}$, but only inner products of training vectors $\boldsymbol{x}^{(i)} \cdot \boldsymbol{x}^{(j)}$.

The SVM algorithm provides not only a decision but also a score $f(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{w} + b$.

Exercises

1. Prove that if we know $\alpha_i,\ i = 1, \ldots, n$, we can obtain the offset from the support vectors by

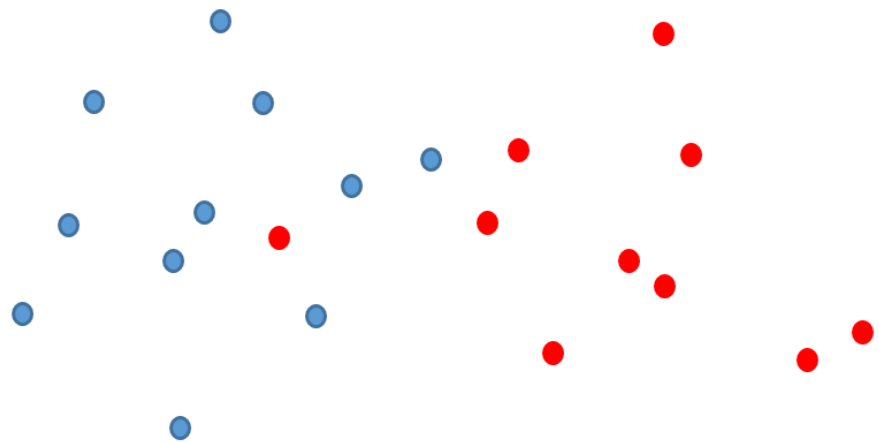
$$b = \frac{1}{N_s} \sum_{s \in S} \left[y^{(s)} - \sum_{m \in S} \alpha_m y^{(m)} (\boldsymbol{x}^{(m)} \cdot \boldsymbol{x}^{(s)}) \right]$$

2. Which formulation (primary or dual) has more parameters to optimize?

Case II: data that cannot be separated by an hyperplane

SVMs can be extended to deal with data that is not linearly separable. In this case it is not possible to classify all the training vectors without errors, using an hyperplane.

Idea: allow data points on the wrong side of the margin hyperplane, provided that they suffer a penalty. This is known as soft margin.

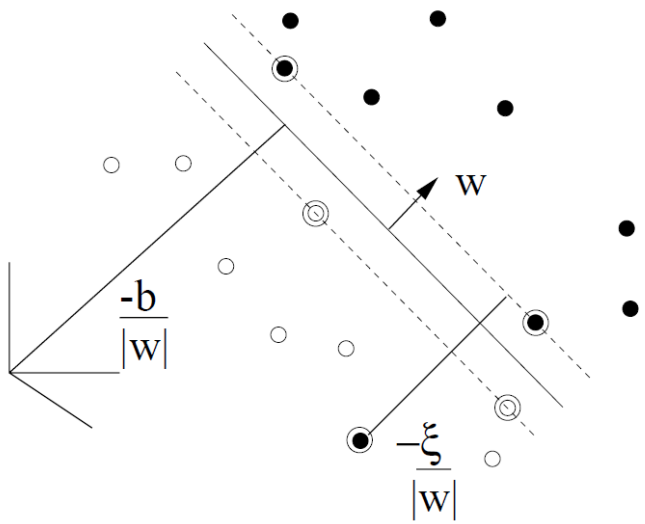


Soft margin

The idea is to assign a slack (folga) variable ξ_i to each data point $\boldsymbol{x}^{(i)}$ defined in such way that $\xi_i = 0$ if no margin violation occurs and $\xi_i > 0$ if the *ith* point is on the wrong side of the margin hyperplane.

Soft margin penalty: $C \sum_{i=1}^{n} \xi_i$

All the training examples in the wrong side of the margin are considered as support vectors since they influence the decision boundary.



The constraints can be written as follows

$$\left.\begin{array}{ll} \boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b \geq +1 - \xi_i & \text{for } y^{(i)} = +1 \\ \boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b \leq -1 + \xi_i & \text{for } y^{(i)} = -1 \end{array}\right\} \Rightarrow y^{(i)}(\boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b) - 1 + \xi_i \geq 0, \ \forall i$$

with $\xi_i \geq 0$.

Optimization problem (soft margin)

Optimization problem:

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad s.t. \quad y^{(i)}(\boldsymbol{x}^{(i)}\cdot\boldsymbol{w}+b) - 1 + \xi_i \geq 0,\ \forall i.$$

Lagrangian function

$$L_P = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[y^{(i)}(\boldsymbol{x}^{(i)}\cdot\boldsymbol{w}+b) - 1 + \xi_i] - \sum_{i=1}^{n}\mu_i\xi_i$$

where $\alpha_i, \mu_i \geq 0$ are Lagrange multipliers.

$\boldsymbol{w}, b$ and ξ_i should be chosen to minimize L_P, and α_i, μ_i to maximize it.

Dual Lagrangian problem

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\boldsymbol{\alpha}^T H \boldsymbol{\alpha} \ , \quad s.t. \quad 0 \leq \alpha_i \leq C \ \ \forall \ i \ , \ \ \sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

where $\alpha = [\alpha_1 \ \ldots \ \alpha_n]^T$ and $H_{ij} = y^{(i)}(\boldsymbol{x}^{(i)} \cdot \boldsymbol{x}^{(i)})y^{(i)}$.

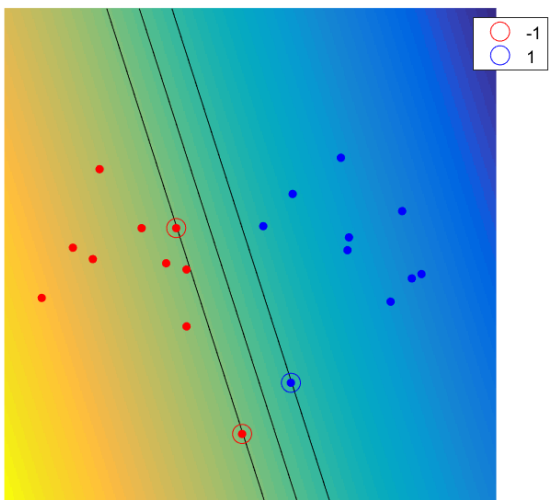
This is a convex quadratic programming (QP) problem that can be solved by standard QP algorithms and provides the all α_i.

The classifier parameters, $\boldsymbol{w}, b$, are obtained the same way as before. However, the lagrange multipliers α_i are now constrained to the interval $[0, C]$.

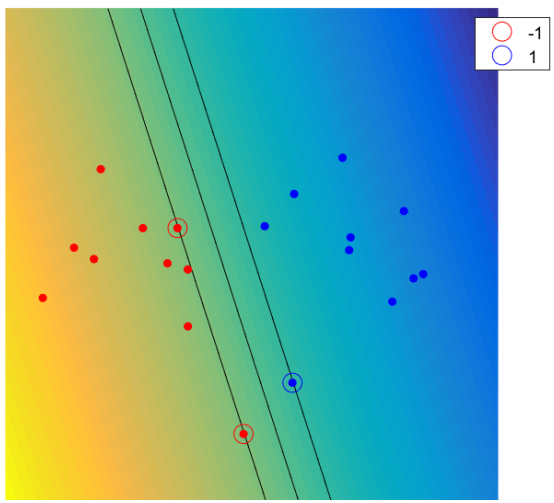
Example - linearly separable data

This example shows separable data set classified with hard margin (left) and soft margin (center, right).

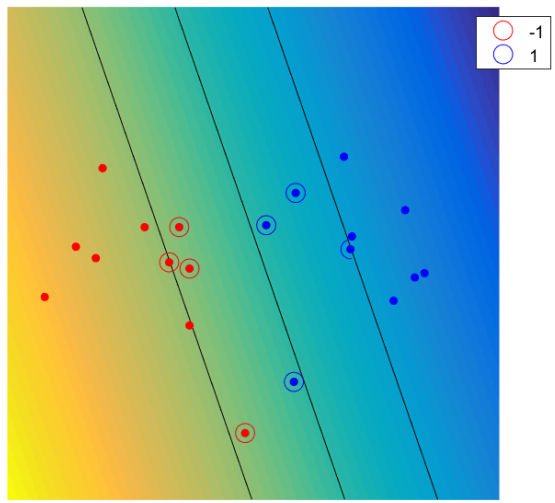
hard margin



soft margin (C=10)



soft margin (C=0.1)



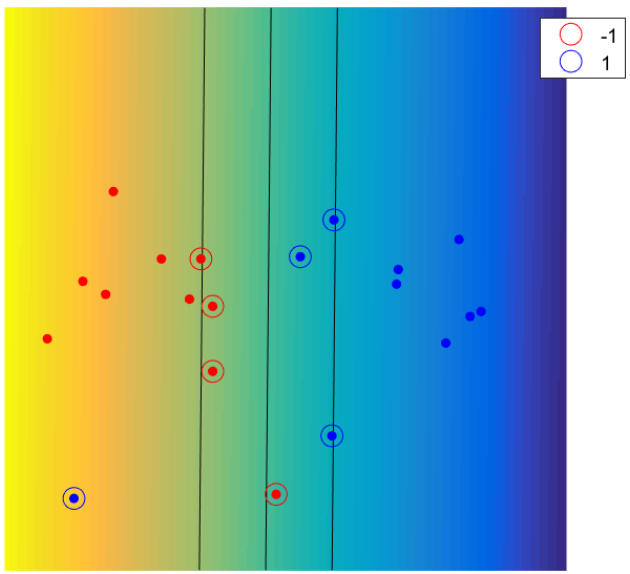
support vectors are identified with a circle.

soft margin classifier with large C is equal to the hard margin classifier.

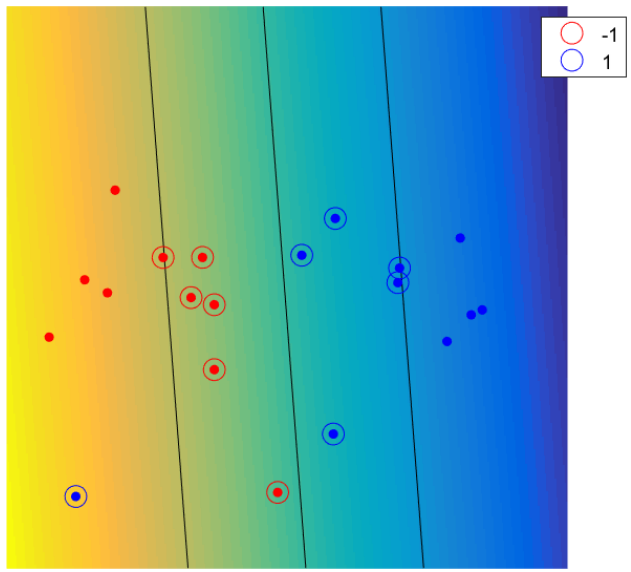
Example - data not linearly separable

This example shows data not linearly separable, classified with soft margin.

soft margin (C=10)



soft margin (C=0.1)



The choice of C controls the margin width.

Interpretation of SVM with Hinge loss

The slack variables can be obtained by using the hinge loss

$$\xi_i = \max\left(0, 1 - y^{(i)}(\boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b)\right)$$

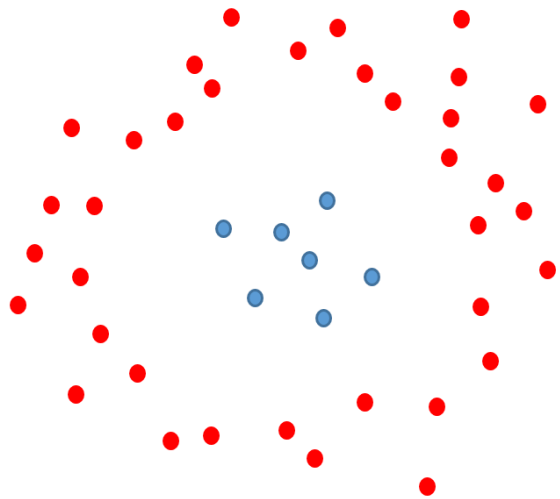
Therefore, the linear SVM with soft margin minimizes

$$\sum_{i=1}^{n} \max\left(0, 1 - y^{(i)}(\boldsymbol{x}^{(i)} \cdot \boldsymbol{w} + b)\right) + \lambda \|\boldsymbol{w}\|^2$$

with $\lambda = 1/(2C)$.

Case III: non linear SVM

Linear SVMs classify data using an hyperplane trained with hard margin or soft margin. This is too restrictive, especially when the dimension of input space is low.



Transformed space

Many problems require a decision boundary with curvature which cannot be synthesized by a linear SVM.

Linear classifiers work better in higher dimensional spaces. Therefore, one strategy consists of mapping the data from the original input space into a high dimension space (feature space)

$$\tilde{\boldsymbol{x}} = \phi(\boldsymbol{x})$$

where the data can be separated by an hyperplane. ϕ is a nonlinear map.

Questions: can SVMs be extended to these high dimension feature spaces? or will they become unfeasible?

The kernel trick

The linear SVM algorithm (dual formulation) does not require the input vectors $\boldsymbol{x}(i)$ but only inner products between them $\boldsymbol{x}^{(i)} \cdot \boldsymbol{x}^{(j)}$.

This means that we do not need to compute the feature vectors (transformed input vectors) $\tilde{\boldsymbol{x}}^{(i)} = \phi(\boldsymbol{x}^{(i)})$ but only their inner products $\phi(\boldsymbol{x}^{(i)}) \cdot \phi(\boldsymbol{x}^{(j)})$

The good new is that we can compute these inner products using a kernel function

$$k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \phi(\boldsymbol{x}^{(i)}) \cdot \phi(\boldsymbol{x}^{(j)}),$$

that can be computed in low dimension input space.

The non linear SVM can be trained and tested using low dimension data, by replacing the inner products by the kernel.

Typical kernels

The most common choices are:

linear: $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \boldsymbol{x}^{(i)T}\boldsymbol{x}^{(j)}$

rbf: $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = e^{-\frac{1}{2\sigma^2}\|\boldsymbol{x}^{(i)}-\boldsymbol{x}^{(j)}\|^2}$

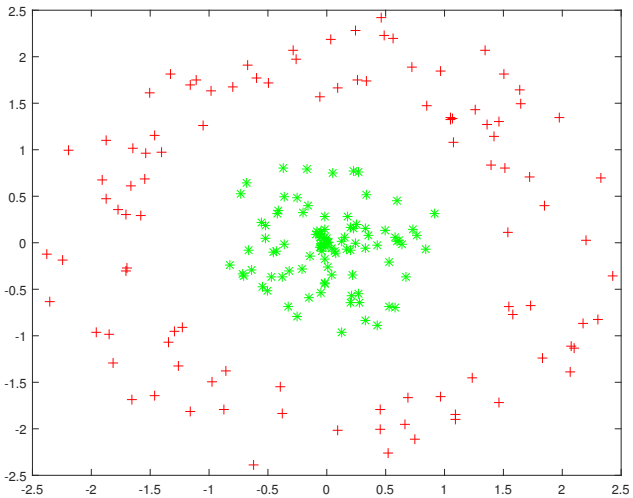
polynomial: $k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = (\boldsymbol{x}^{(i)T}\boldsymbol{x}^{(j)} + a)^b$

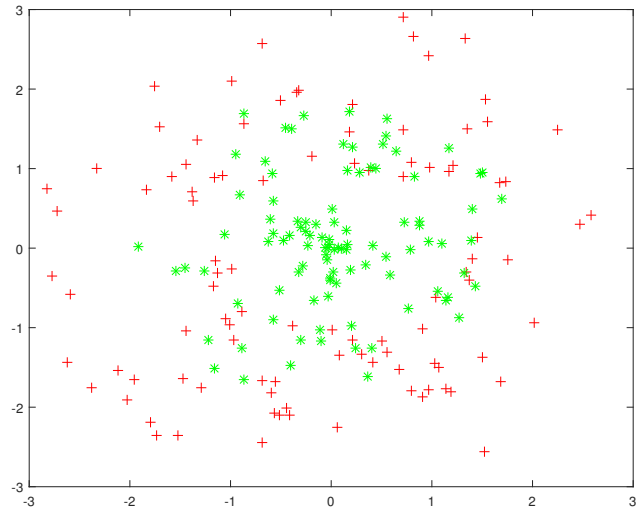
The linear kernel is the one adopted in linear SVM.

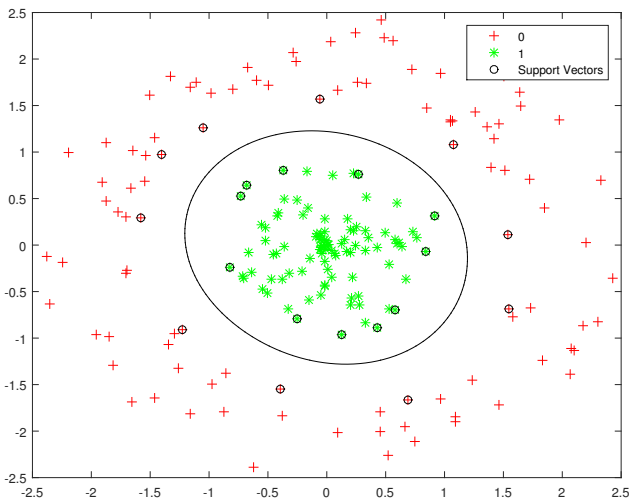
We note that some kernels depend on hyperparameters that have to be specified or learned during the training phase *e.g.*, typically by *ad hoc* procedures or by cross validation.

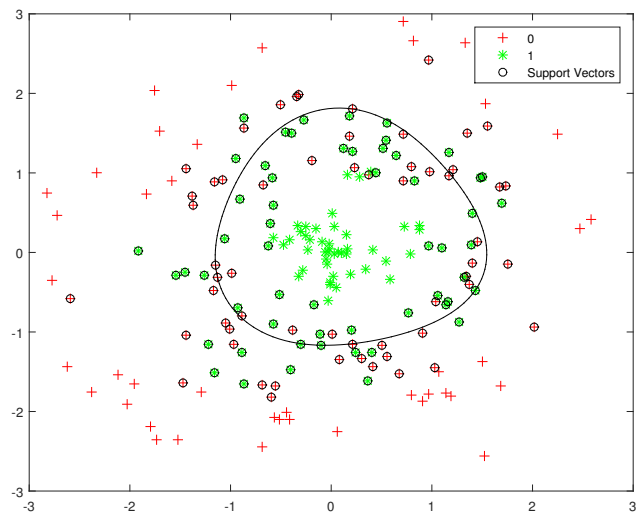
Examples - SVM

Two examples (training data) solved by Matlab (function *svmtrain*) with rbf kernel. Support vectors are identified with a circle.









Extension

Question: How can we solve muti-class classification problems with SVMs?

Exercise

We wish to design a SVM classifier uaing the following training data,

x_1	x_2	*class*
2	0	*A*
0	3	*A*
3	0	*B*
2	−5	*B*
−4	2	*B*

Consider the transformation from the input space to the feature space defined by

$$(x_1, x_2) \rightarrow (x_1^2, x_2)$$

1. Find the equations of the decision and margin surfaces in feature space. Sketch them together with the support vectors.
2. Determine the margin value of the classifier in feature space.
3. Compute and sketch the decision and margin surfaces in input space as well as the support vectors.
4. Draw the decision regions of both classes in input space.

Machine Learning

Linear Regression

Regularization

Evaluation & Generalization

Optimization

Neural networks

Data classification

Linear classifiers

Support vector machines

Decision Trees and Random Forest

Decision trees

Decision trees are popular classifiers since they allow us to understand why the input pattern is classified in a specific class.

This property is important in several applications (*e.g.*, medical diagnosis).

Decision trees are often used when the features are categorical, although they have been extended to numerical features as well.

Next slides adress:

- how is categorical data classified by a tree?
- how is the tree trained?
- how can trees be extended to numerical features?

Example: Good days to play tennis

Suppose we wish to predict what are the good days to play tennis and we have the following dataset associated to a player (John).

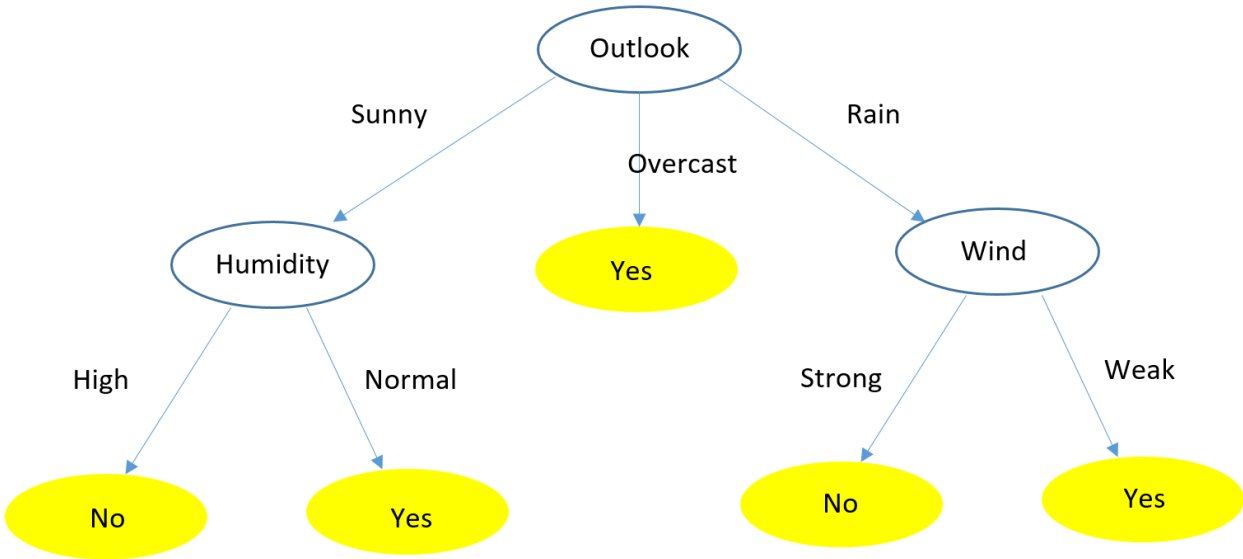
Day	Outlook	Humidity	Wind	Play
1	Sunny	High	Weak	No
2	Sunny	High	Strong	No
3	Overcast	High	Weak	Yes
4	Rain	High	Weak	Yes
5	Rain	Normal	Weak	Yes
6	Rain	Normal	Strong	No
7	Overcast	Normal	Strong	Yes
8	Sunny	High	Weak	No
9	Sunny	Normal	Weak	Yes
10	Rain	Normal	Weak	Yes
11	Sunny	Normal	Strong	Yes
12	Overcast	High	Strong	Yes
13	Overcast	Normal	Weak	Yes
14	Rain	High	Strong	No

adapted from Quinlan, 1986

Days with the same attributes may have different outcomes. This is known as noisy labels.

A decision tree

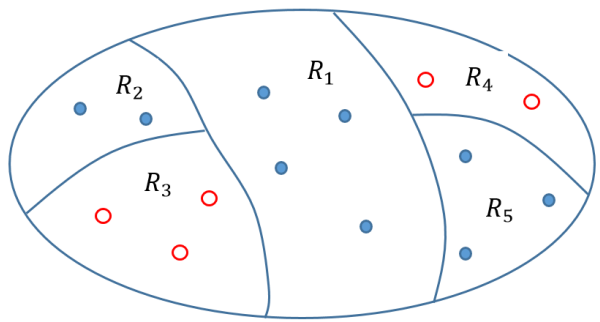
A decision tree that solves the problem is



adapted from Quinlan, 1986

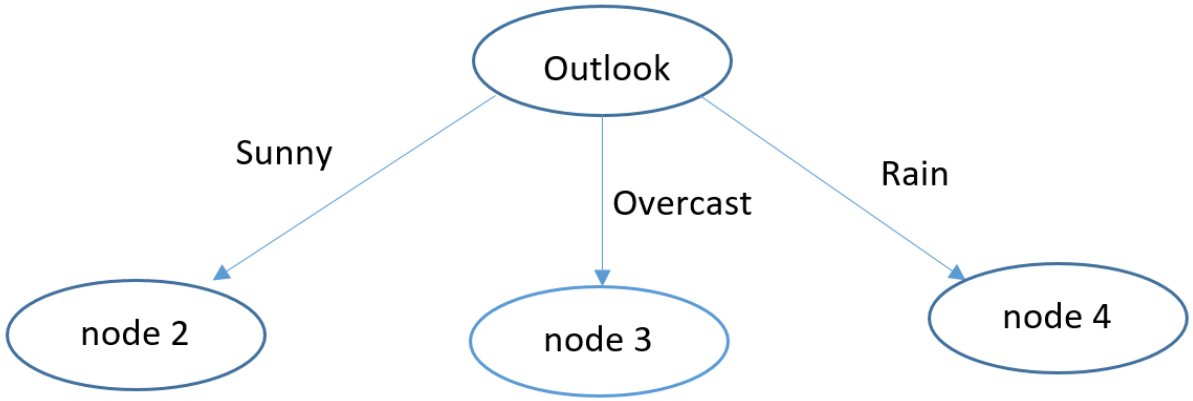
This tree contains three types of nodes:

- one root node;
- splitting nodes;
- leave nodes (associated to labels).



Each splitting node is associated to a question involving one of the features.

What training data is associated to each node?



Each node has a subset of training examples associated to it.

node 3

Day	Outlook	Humidity	Wind	Play
3	Overcast	High	Weak	Yes
7	Overcast	Normal	Strong	Yes
12	Overcast	High	Strong	Yes
13	Overcast	Normal	Weak	Yes

pure subset!

node 2

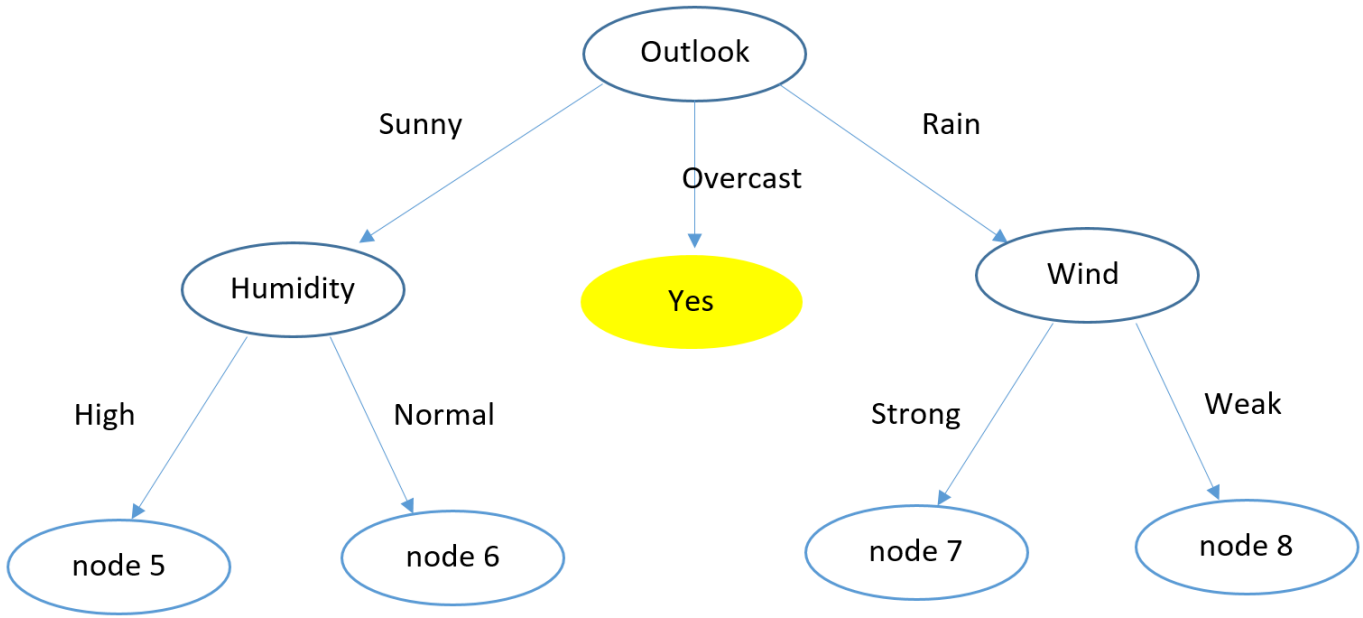
Day	Outlook	Humidity	Wind	Play
1	Sunny	High	Weak	No
2	Sunny	High	Strong	No
8	Sunny	High	Weak	No
9	Sunny	Normal	Weak	Yes
11	Sunny	Normal	Strong	Yes

node 4

Day	Outlook	Humidity	Wind	Play
4	Rain	High	Weak	Yes
5	Rain	Normal	Weak	Yes
6	Rain	Normal	Strong	No
10	Rain	Normal	Weak	Yes
14	Rain	High	Strong	No

If the training examples associated to a node have the same label, the node is called pure. Pure nodes are not split anymore and receive a label.

What training data is associated to each node? (2)



node 5

Day	Outlook	Humidity	Wind	Play
9	Sunny	Normal	Weak	Yes
11	Sunny	Normal	Strong	Yes

node 6

Day	Outlook	Humidity	Wind	Play
1	Sunny	High	Weak	No
2	Sunny	High	Strong	No
8	Sunny	High	Weak	No

both subsets are pure!

node 7

Day	Outlook	Humidity	Wind	Play
4	Rain	High	Weak	Yes
5	Rain	Normal	Weak	Yes
10	Rain	Normal	Weak	Yes

node 8

Day	Outlook	Humidity	Wind	Play
6	Rain	Normal	Strong	No
14	Rain	High	Strong	No

both subsets are pure!

Classification of new data

Classification of new data is done in the same way. Given a feature vector x, we travel along the tree according to the questions, until we reach a leaf (with a label).

There will be classification errors not only in the test set but also in the training set, if there are examples with the same attributes and different labels. This is known as noisy labels

Posterior distribution of classes at each node

Consider a training set $\mathcal{T} = \left\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\right\}$. A decision tree associates each training pattern $x^{(i)}$ to a node m through a sequence of questions.

We can estimate the *a posteriori* distribution of the labels associated to each tree node m, using the training data

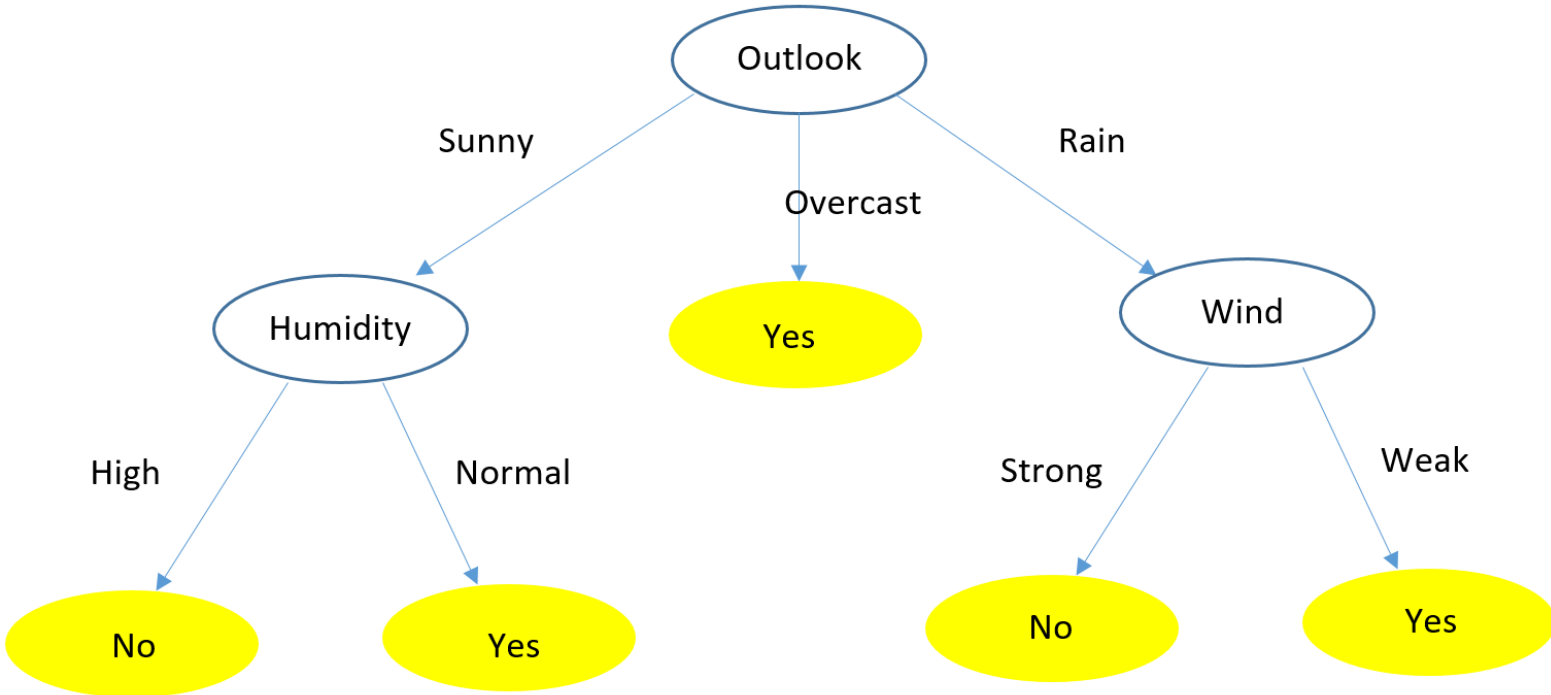
$$P(k|m) = \frac{1}{\#\mathcal{T}_m} \sum_{x^{(i)} \in \mathcal{T}_m} I(y^{(i)} = k),$$

where $\mathcal{T}_m$ is the set of training patterns associated to node m and $I(.)$ is the indicator function (the indicator function is 1 if the argument is true and 0 otherwise).

If node m is a leaf, the most probable label is

$$\hat{k}(m) = \arg\max_k P(k|m).$$

Exercise



Consider the tennis data set and the decision tree shown above. Find the probability of each label (Yes or No), at each node.

Exercise (cont.)

m node	#Yes	#No	$P(Yes\|m)$	$\hat{k}(m)$
1	9	5	9/14	*Yes*
2	2	3	2/5	*No*
3	4	0	1	*Yes*
4	3	2	3/5	*Yes*
5	0	3	0	*No*
6	2	0	1	*Yes*
7	0	2	0	*No*
8	3	0	1	*Yes*

Node impurity

Ideally, each leaf m should be pure *i.e.*, all the training examples arriving at leaf node m should have the same label (class). Since this is not always true we need a measure of impurity.

Several impurity measures have been proposed. They all achieve a minimum if all the data associated to a leaf comes from a single class.

- Misclassification error:

$$i(m) = 1 - \max_k P(k|m) = 1 - P(\hat{k}(m)|m),$$

- Entropy:

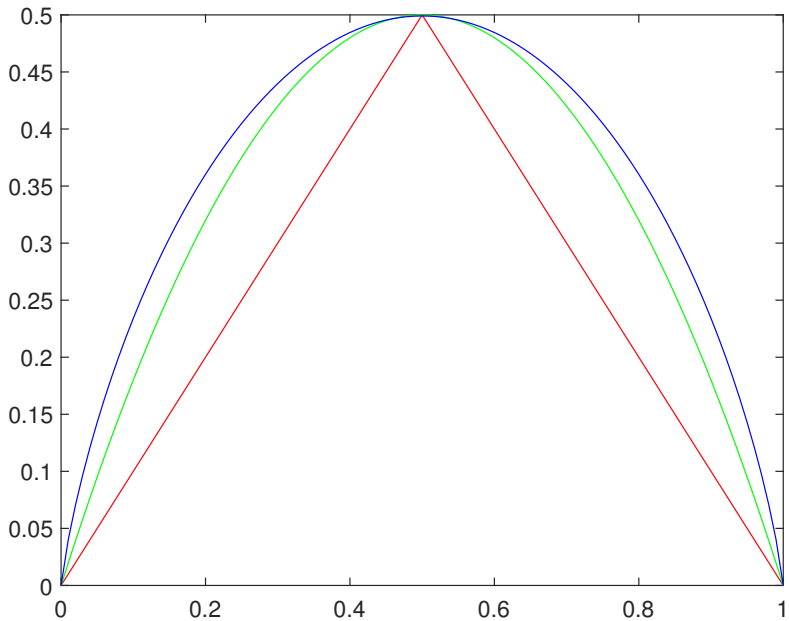
$$i(m) = -\sum_{k=1}^{K} P(k|m) \log_2 P(k|m),$$

- Gini index:

$$i(m) = -\sum_{k=1}^{K} P(k|m) \left(1 - P(k|m)\right).$$

Node impurity (binary case)

In binary classification problems (2 labels), the impurity depends on a single statistic $P(k = 1|m)$. Figure shows the misclassification error (red), the Gini index (green), and the entropy (blue) as a function of $P(k = 1|m)$. The vertical scale was modified.



The entropy and Gini index are smoother and they are usually preferred in model training.

Optimal tree training

Given a data set, we wish to learn a tree T. Each splitting node corresponds to a question and each leaf corresponds to a label.

Training a tree amounts to minimizing the tree impurity

$$I(T) = \sum_{m \in \tilde{T}} P(m) i(m),$$

where $P(m)$ is the fraction of training patterns associated to leaf m and $\tilde{T}$ is the set of all the leaf nodes.

The tree impurity is an average impurity of the leaf nodes.

Optimal training of a tree amounts to finding a tree that minimizes $I(T)$. We should generate all the tree configurations, compute the tree impurity for each configuration, and choose the one with smallest impurity.

Drawbacks

This approach has two drawbacks:

1. The optimal solution cannot be found in most problems. Exhaustive search of all tree configurations is not feasible and greedy approaches are used instead.

2. The criterion $I(T)$ optimizes the performance on the training set but this is highly optimistic, leading to overfitting.

Tree growing

To overcome the first difficulty, we start with a single node m (root) and choose the best attribute for splitting the node. This is done as follows.

For each attribute X_j, we split m and create children nodes $s \in S$, each of them associated to a different value of the attribute. We compute the impurity of each son, $i(s)$, and the impurity drop with respect to the impurity of node m

$$\Delta I = i(m) - \sum_{s \in S} \frac{p(s)}{p(m)} i(s).$$

The attribute that achieves the greatest drop is selected.

The splitting process is repeated for another leaf node until a stop condition is met. For example, until all the leafs are pure or all the attributes have been tested.

ID3 algorithm

The ID3 algorithm is a basic tree learning method for categorical data.

It has the following features:

- impurity criterion: entropy;
- stop criterion: stop when each leaf is pure or, if not, all the attributes have been tested along the path from the root to the impure leaf.

When the data is noisy (noisy label or noisy attributes) the ID3 algorithm may overfit the training data leading to poor performance in independent data sets (test sets).

This drawback can be alleviated by early stop or post-processing (tree pruning).

Exercises

1. Apply the ID3 algorithm to the tenis data set.

2. Consider the following data set (vertebrates). Check which of the attributes is chosen by the ID3 algorithm for the root node (don't consider the name and the skin cover).

#	name	body temperature	skin cover	gives birth	aquatic creature	aerial creature	has legs	hiber-nates	class label
1	human	warm-blooded	hair	Yes	No	No	yes	no	mammal
2	python	cold-blooded	scales	No	No	No	no	yes	non-mam
3	salmon	cold-blooded	scales	No	Yes	No	no	no	non-mam
4	whale	warm-blooded	hair	Yes	Yes	No	no	no	mammal
5	frog	cold-blooded	none	No	Semi	No	yes	yes	non-mam
6	Komodo	cold-blooded	scales	No	No	No	yes	no	non-mam
7	dragon								
8	bat	warm-blooded	hair	Yes	No	yes	yes	yes	mammal
9	pigeon	warm-blooded	feathers	No	No	yes	yes	no	non-mam
10	cat	warm-blooded	fur	Yes	No	No	yes	no	mammal
11	leopard	warm-blooded	fur	Yes	No	No	yes	no	mammal
12	turtle	cold-blooded	scales	No	semi	No	yes	no	non-mam
13	penguin	warm-blooded	feathers	No	semi	No	yes	no	non-mam
14	porcupine	warm-blooded	quills	Yes	No	No	yes	yes	mammal
15	heel	cold-blooded	scales	No	Yes	No	yes	no	non-mam
16	salamander	cold-blooded	none	No	semi	No	no	yes	non-mam

adapted from Kumar, Introduction to Data Mining, 2014.

Solution of first exercise

O	H	W	P
S	H	W	N
S	H	S	N
O	H	W	Y
R	H	W	Y
R	N	W	Y
R	N	S	N
O	N	S	Y
S	H	W	N
S	N	W	Y
R	N	W	Y
S	N	S	Y
O	H	S	Y
O	N	W	Y
R	H	S	N

Test root

Outlook	*N*	*Y*
R	2	3
S	3	2
O	0	4

$$i(O) = \frac{5}{14} \times 0.97 + \frac{5}{14} \times 0.97 + \frac{4}{14} \times 0 = 0.69$$

Humidity	*N*	*Y*
H	4	3
N	1	6

$$i(H) = \frac{7}{14} \times 0.98 + \frac{7}{14} \times 0.59 = 0.78$$

Wind	*N*	*Y*
S	3	3
W	2	6

$$i(W) = \frac{6}{14} \times 1 + \frac{8}{14} \times 0.81 = 0.89$$

Best choice for the root is Outlook and Outlook=Overcast is a pure node with label Yes.

Solution (cont.)

Node: Outlook=Sunny

H	W	P
H	W	N
H	S	N
H	W	N
N	W	Y
N	S	Y

Humidity	*N*	*Y*
H	3	0
N	0	2

Two pure nodes: $i(H) = 0$

Node: Outlook=Rain

H	W	P
H	W	Y
N	W	Y
N	S	N
N	W	Y
H	S	N

Humidity	*N*	*Y*
H	1	1
N	1	2

$$i(H) = \frac{2}{5} \times 1 + \frac{3}{5} \times 0.91 = 0.95$$

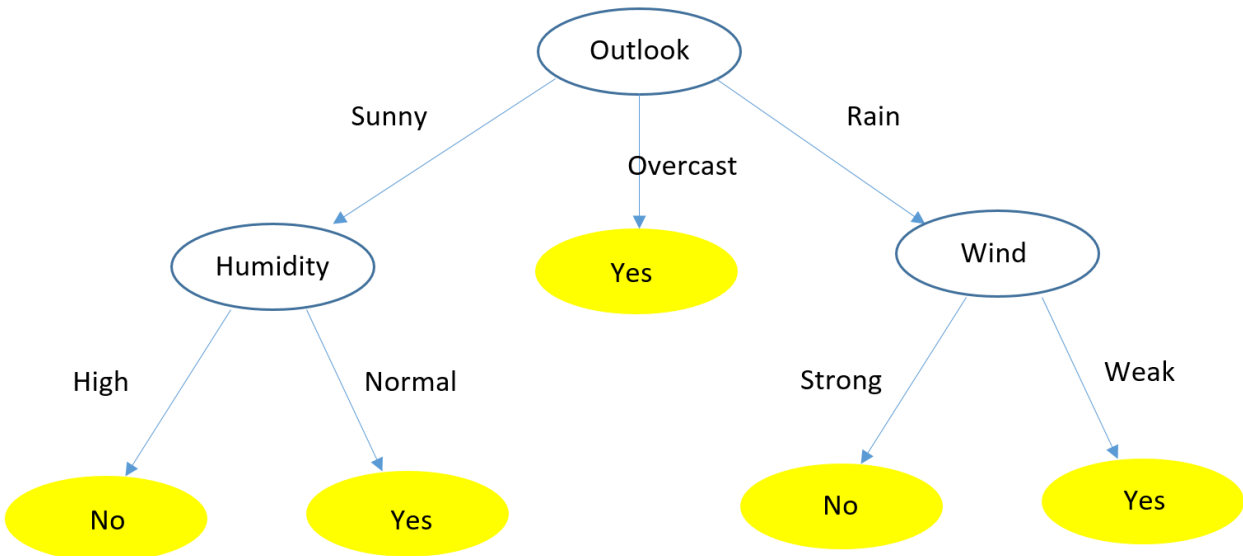
Wind	*N*	*Y*
S	2	0
W	0	3

Two pure nodes: $i(W) = 0$

Best choice for the node Outlook=Sunny is Humidity and leads to two pure nodes. Best choice for Outlook=Rain is Wind and leads to two pure nodes.

Solution (cont.)

The decision tree we have obtained:



When to stop?

- The impurity of the tree drops or remains constant every time a node is split. In the limit we can grow the tree until each leaf is pure or all the attributes along that path have been used. This approach leads to overfitting.

- A second approach consists of using a validation technique. The tree is grown using a subset of training data (70%) and evaluated using the remaining patterns (30%) (validation set).

- Another strategy consists of growing the tree while the impurity drop is above a threshold $\Delta I > \beta$.

- Another approach is based on a regularization criterion

$$I(T) + \alpha \tilde{N}$$

where $\tilde{N}$ is the number of leaf nodes.

Example - exclusive OR

Consider a toy problem:

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

- would you be able to predict y with a tree classifier?
- compute the impurity drop for the splitting of the root node, using the entropy criterion. What do you conclude?

Drawbacks

- Early stop is not a good strategy to train the model because it suffers from lack of sufficient look ahead.
- It is better growing the tree until the leaves are pure or all attributes have been used and then prune the tree.
- Instability: a small change in the training patterns may lead to big changes of the decision boundaries.

Tree pruning

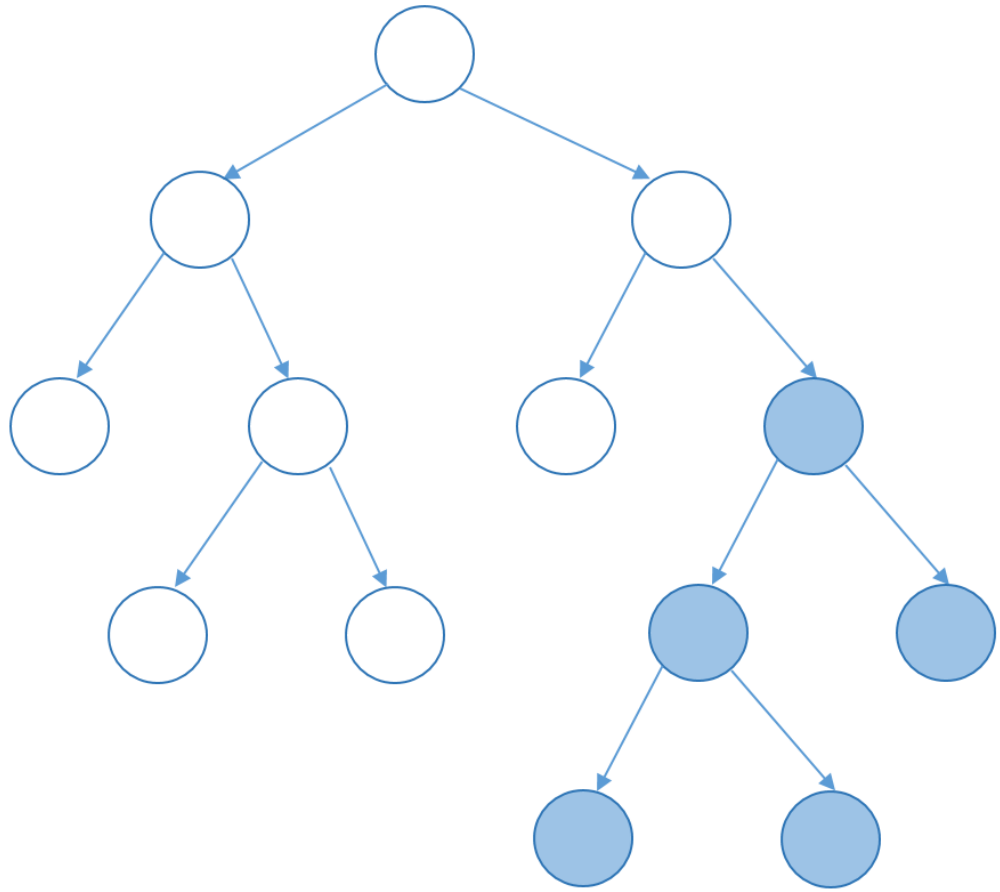
Tree pruning is a tree simplification, aiming to improve the performance of the tree. The performance is usually measured as number of classification errors in the validation set.

This usually involves three steps:

- trying several tree simplifications
- evaluate each of them in the validation set
- choose the best

The process is repeated until no further improvements can be achieved.

Subtree replacement



Test all the splitting nodes in a bottom-up way. For each tested node, remove its descendants (subtree) and replace the splitting node by a leaf. The change is accepted if the modified tree has a better or equal performance (number of errors in the validation set).

Error estimation

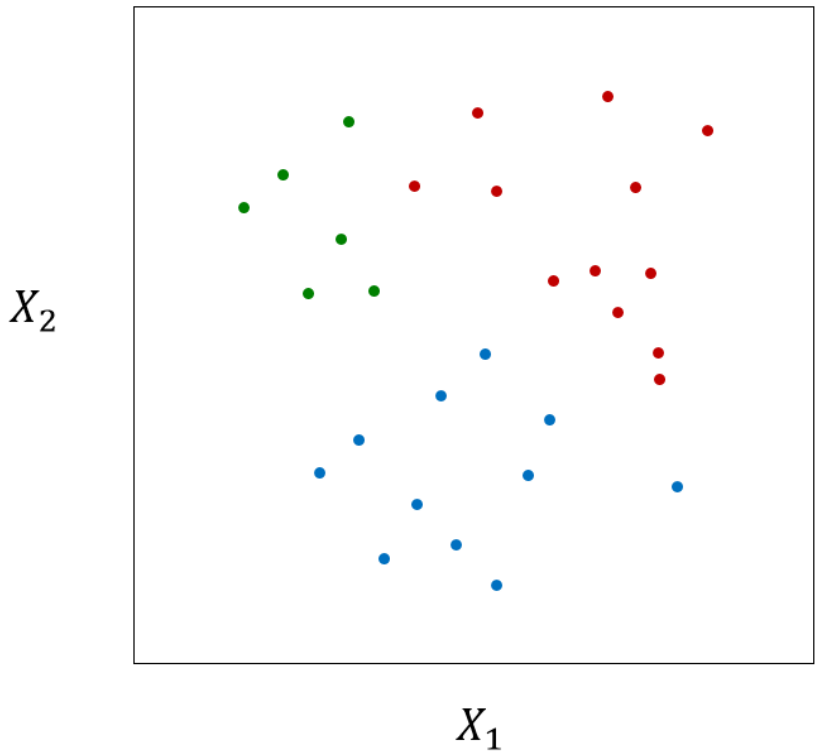
When there is no validation set (too few training examples) the error in the validation set is predicted by using a pessimistic estimator given by

$$e = \frac{f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}} ,$$

where

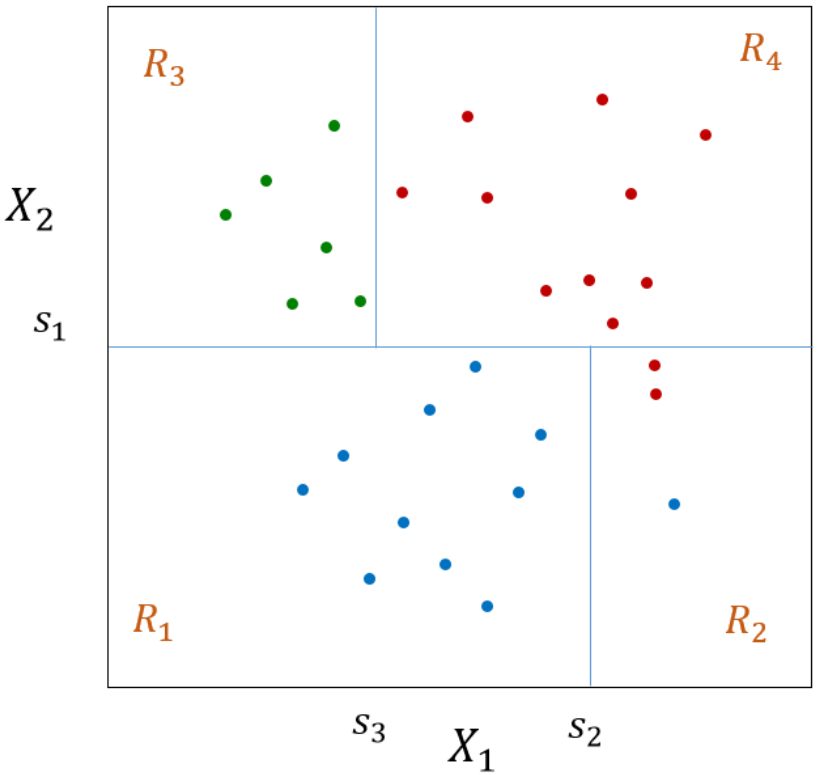
- z - parameter that depends on the confidence degree c (if $c = 25\%$, $z = 0.69$)
- f - percentage of error in the training set
- N - number of training examples in the leaf

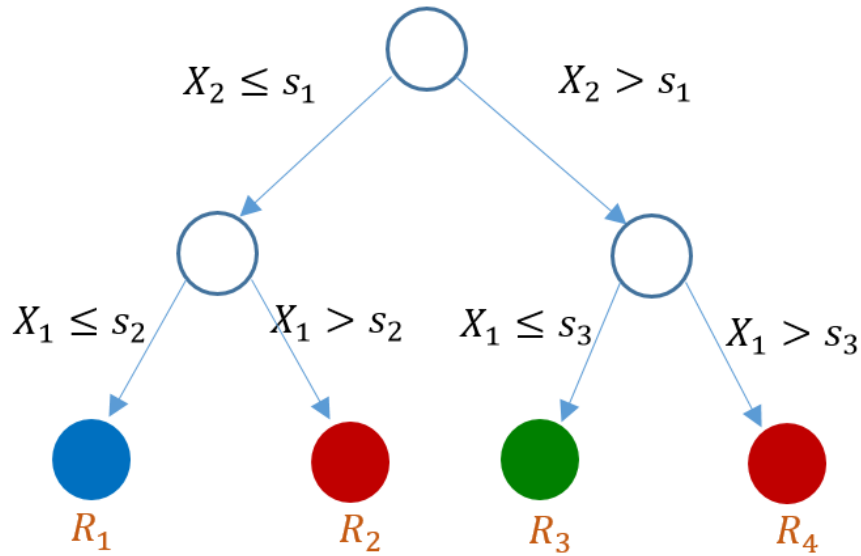
Can trees be applied with numerical features?



Three classes (colors)

Yes! using thresholds





Threshold values have to be estimated during the tree growing process, usually by exhaustive search. All threshold values are considered for each feature and the best impurity drop is selected.

Bootstrap aggregation (bagging) for regression

Bootstrap aggregation, also called bagging, is an ensemble method that can be used to improve the performance of regressors and classifiers.

Consider a regression problem with a training set of n independent and identically distributed (iid) patterns, drawn from a distribution $\mathcal{P}$

$$\mathcal{T} = \left\{ (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \right\}$$

Using previous methods we can learn a function $f(x)$ under uncertainty.

One way to improve the estimate $f(x)$ would be to consider multiple training sets.

Multiple training sets

Consider B training sets generated from the same (ideal) distribution $\mathcal{P}$

$$\begin{array}{rcl} \mathcal{T}^{(1)} = & \left\{(x^{(1,1)}, y^{(1,1)}), \ldots, \right. & \left. (x^{(1,n)}, y^{(1,n)})\right\} \\ \vdots & \vdots & \vdots \\ \mathcal{T}^{(B)} = & \left\{(x^{(B,1)}, y^{(B,1)}), \ldots, \right. & \left. (x^{(B,n)}, y^{(B,n)})\right\} \end{array}$$

This allows us to estimate B regression functions $f^{(1)}(x), \ldots, f^{(B)}(x)$, each from a different training set. These functions can be combined (aggregated) by averaging

$$\hat{f}(x) = \frac{1}{B} \sum_{i=1}^{B} f^{(i)}(x)$$

to reduce the uncertainty.

Bootstrap

There is only one difficulty. We do not know the ideal distribution $\mathcal{P}$. All we know is the first data set $\mathcal{T}$ and the empirical distribution computed from it.

The trick consists of generating the multiple data sets $\mathcal{T}^{(i)}$ using the Bootstrap method i.e., by sampling the set $\mathcal{T}$, n times, (with replacement).

Of course, we cannot claim as before that the $\mathcal{T}^{(i)}$ are statistically independent, but the technique still improves the estimation of f.

Bagging in classification problems

Let us consider a classification problem with a training set

$$\mathcal{T} = \left\{ (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \right\}$$

where the outcomes $y^{(i)}$ belong to a finite set of labels $\{0, \ldots, K-1\}$.

Given a new input pattern x, the trained classifier produces a vector of probability estimates $[P(y=0|x), \ldots, P(y=K-1|x)]$ subject to estimation errors.

The Bagging algorithm is as follows

1. generate B training sets $\mathcal{T}^{(i)}$ from $\mathcal{T}$, by bootstrap *i.e.*, by sampling $\mathcal{T}$ with replacement.
2. train a classifier from each set $\mathcal{T}^{(i)}$ and compute the *a posteriori* distributions $\left[P^{(i)}(y=0|x), \ldots, P^{(i)}(y=K-1|x)\right]$.
3. aggregate all the estimates

$$\hat{P}(y=k|x) = \frac{1}{B} \sum_{i=1}^{B} P^{(i)}(y=k|x)$$

Random forest

Random forest is a very simple and yet a very powerful classifier. Achieves state-of-the-art results in many problems.

The algorithm is based on a ensemble of tree classifiers trained with bagging.

The Random forest algorithm

1. generate B training sets $\mathcal{T}^{(i)}$ from $\mathcal{T}$ by bootstrap *i.e.*, by sampling $\mathcal{T}$ with replacement.
2. train a tree classifier from each set $\mathcal{T}^{(i)}$ with the following issue. Randomly select a subset of features at each node and only those features are candidates for splitting features. This procedure is known as random subspace. The percentage of feature candidates at each node is a parameter of the algorithm.
3. compute the *a posteriori* distributions $\left[P^{(i)}(y=0|x), \ldots, P^{(i)}(y=K-1|x)\right]$ for each tree.

Random forest

Random forest is a very simple and yet a very powerful classifier. Achieves state-of-the-art results in many problems.

The algorithm is based on a ensemble of tree classifiers trained with bagging.

The Random forest algorithm

1. generate B training sets $\mathcal{T}^{(i)}$ from $\mathcal{T}$ by bootstrap *i.e.*, by sampling $\mathcal{T}$ with replacement.
2. train a tree classifier from each set $\mathcal{T}^{(i)}$ with the following issue. Randomly select a subset of features at each node and only those features are candidates for splitting features. This procedure is known as random subspace. The percentage of feature candidates at each node is a parameter of the algorithm.
3. compute the *a posteriori* distributions $\left[P^{(i)}(y=0|x), \ldots, P^{(i)}(y=K-1|x)\right]$ for each tree.
4. aggregate all the *a posteriori* distributions

$$\hat{P}(y=k|x) = \frac{1}{B}\sum_{i=1}^{B} P^{(i)}(y=k|x)$$