Statistical Machine Learning
Hilary Term 2021

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

What is Machine Learning?

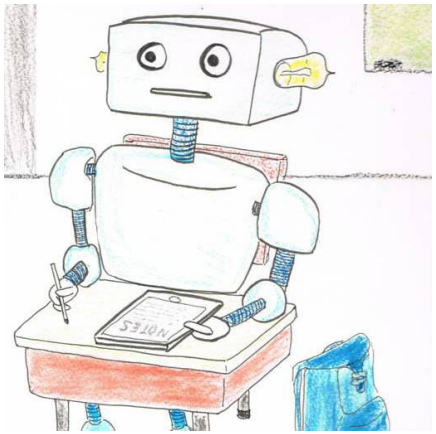


Image: www.gureckislab.org

Bottom-up, data-driven approach. Allows machines to adapt and improve their knowledge based on new data.

What is Machine Learning?

Arthur Samuel, 1959

Field of study that gives computers the ability to **learn** without being explicitly programmed.

What is Machine Learning?

Arthur Samuel, 1959

Field of study that gives computers the ability to **learn** without being explicitly programmed.

Tom Mitchell, 1997

Any computer program that **improves its performance** at some task **through experience**.

What is Machine Learning?

Arthur Samuel, 1959

Field of study that gives computers the ability to **learn** without being explicitly programmed.

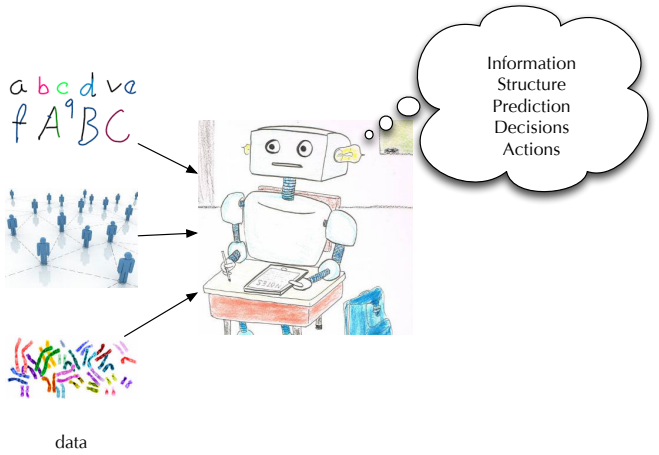
Tom Mitchell, 1997

Any computer program that **improves its performance** at some task **through experience**.

Kevin Murphy, 2012

To develop methods that can **automatically** detect **patterns in data**, and then to use the uncovered patterns to **predict** future data or other outcomes of interest.

What is Machine Learning?



Larry Page about DeepMind's ML systems that can learn to play video games like humans

Statistics vs Machine Learning

Traditional Problems in Applied Statistics

- Well formulated question that we would like to answer.
- Expensive data gathering and/or expensive computation.
- Create specially designed experiments to collect high quality data.

Information Revolution

- Improvements in data processing and data storage.
- Powerful, cheap, easy data capturing.
- Lots of (low quality) data with **potentially valuable** information inside.

Statistics vs Machine Learning

CS and Stats forced **back together**: unified framework of data, inferences, procedures, algorithms

- statistics taking computation seriously
- computing taking statistical risk seriously

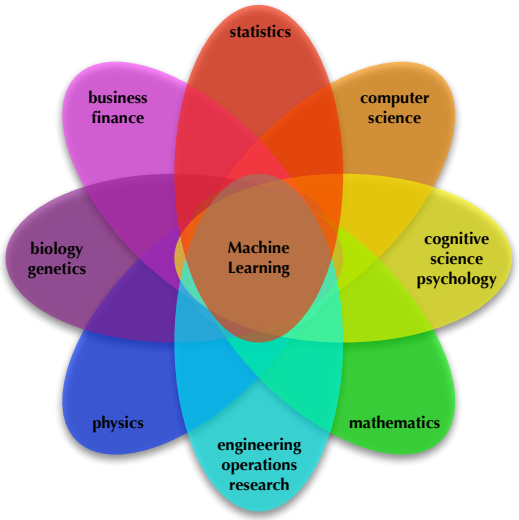
Michael I. Jordan (UC Berkeley):

- *"Really what machine learning has been is ideas from statistics blended with ideas from computer science"*

Michael I. Jordan: On the Computational and Statistical Interface and "Big Data"

Max Welling: Are Machine Learning and Statistics Complementary?

Machine Learning is a highly interdisciplinary field



Turing Award: “Nobel prize" of Computing for ML

2018 Turing Award, known as the "Nobel Prize of computing"
Awarded to Yann LeCun, Geoff Hinton and Yoshua Bengio for their work on deep learning

Photo: Botler AI

Godfathers of AI honored with Turing Award, the Nobel Prize of computing

Deep learning

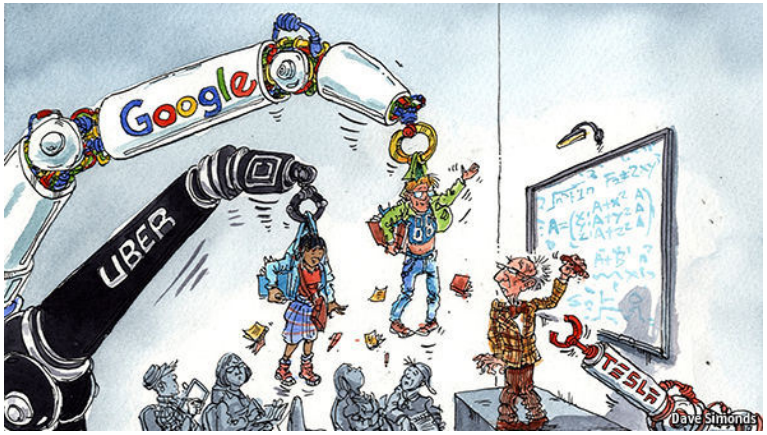
Deep Learning is a particular class of machine learning method
- Can process large amounts of data
- Uses a lot of parameters

In the last twelve years, gave state-of-the-art performances on various prediction tasks (image classification, natural language processing, etc.)

Widespread in industry

“Unreasonable effectiveness" of deep learning still poorly understood; Theory of Deep Learning active field of research

Machine Learning + industry



Applications of Machine Learning

spam filtering

recommendation systems

fraud detection

self-driving cars

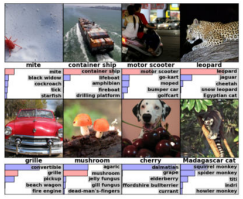


image recognition

stock market analysis

ImageNet: Computer Eyesight Gets a Lot More Accurate, Krizhevsky et al, 2012 New applications of ML: Machine Learning is Eating the World

Types of Machine Learning

Unsupervised learning

Extract key features of the "unlabelled" data

clustering, signal separation, dimensionality reduction

Goal: **representation, compression, noise filtering, visualisation**

Supervised learning

Data contains "labels": every example is an input-output pair

classification, regression

Goal: **prediction on new examples**

Types of Machine Learning

Semi-supervised Learning

A database of examples, only a small subset of which are labelled.

Reinforcement Learning

An agent acting in an environment, given rewards for performing appropriate actions, learns to maximize their reward.

Software

Python: scikit-learn, mlpy, Theano

R

Matlab/Octave

Weka, mlpack, Torch, Shogun, TensorFlow...

Course Aims

Understand statistical fundamentals of machine learning
- Unsupervised/Supervised learning.
- Risk, Overfitting, Generalisation, Complexity

Understand difference between generative and discriminative learning frameworks.

Learn to identify and use appropriate methods and models for given data and task.

Learn to use the relevant Python packages to analyse data, interpret results, and evaluate methods.

Syllabus I

Part I: Unsupervised learning (~ 3 lectures)
- Definition and formalisation
- Principal Component Analysis
- K-means Clustering

Syllabus II

Part II: Supervised learning (~ 8 lectures)
- Statistical learning for supervised learning
  - Loss function and Risk
  - Plug-in estimators and Empirical Risk minimisation
  - Generative vs Discriminative methods
- Generative Classifiers
  - Linear and Quadratic Linear Discriminant Analysis
  - Naive Bayes Classifier
- Further concepts in Statistical Machine Learning
  - Feature Expansion
  - Overfitting and Bias-Variance trade-off
  - Validation and cross-validation
  - Performance metrics
- Optimisation for ML
  - Gradient Descent
  - Stochastic Gradient Descent
  - Early Stopping

Syllabus III

Part III: Useful methods for supervised learning (~ 5 lectures)
- Linear Classifiers:
  - Least-square classifier
  - Perceptron
  - Logistic Regression
- K-nearest neighbors
- Decision trees and random forests
- Boosting
- Neural Networks and deep learning

References

L. Wasserman. All of Statistics. Springer, 2010.

K. Murphy. Machine Learning. A probabilistic perspective. The MIT Press, 2012

C. M. Bishop. Pattern Recognition and Machine Learning. Springer, 2006.

T. Hastie, R. Tibshirani, J. Friedman. The Elements of Statistical Learning. Springer, 2009.

I. Goodfellow, Y. Bengio and A. Courville. Deep Learning. The MIT Press, 2016.

Course Structure

Course material on canvas:
https://canvas.ox.ac.uk/courses/65441
Lectures:
- Pre-recorded, videos and slides available on canvas
- Lecture notes

MSc + Part B:
- 4 classes, one problem sheet per class
- Team: François Caron (lecturer + tutor), Juba Nait Saada (tutor), Valerie Bradley (tutor), Tom Hadfield (tutor), Charline Le Lan (TA), Yiyang Shi (TA), Miriam Stricker (TA), Viet Wild (TA), Chao Zhang (TA), Lorenzo Pacchiardi (TA).

Part B only:
- Classes split in 6 groups
- HT weeks 3, 5, 7, and TT week 1
- Check website for details and deadline for submitting your work

MSc only:
- 2 practicals (first unassessed, second group-assessed), weeks 5, 8.
- Classes on Tuesdays 9-10, HT weeks 3, 5, 7, 8.

Python

The course contains several examples, written in Python

Python Notebooks available on Canvas to reproduce most of the results/figures

Highly recommended to look at the notebooks, apply modifications to the code, to gain a deeper understanding of the material

Scikit-learn: machine learning library

Other useful libraries:
- Numpy and Scipy: Core toolboxes for scientific computing
- Matplotlib: Visualisation
- Pandas: Data manipulation
- Seaborn: Statistical visualisation
- Pytorch: Deep Learning

Python

New to Python? Numerous online resources

https://www.python.org/about/gettingstarted/

Free Python code editors: PyCharm, Spyder, Atom, Jupyter

Anaconda is a distribution of Python and R programming languages for data science and machine learning. It comes with code editors such as Spyder and Jupyter for Python.

Python

Problem sheets contain coding questions

Recommended to use Python, but can alternatively use R (you should look for the relevant package)

MSc practicals set in Python; recommended to use Python, but could use R instead

Notations

Notation

Data consists of p variables (features/attributes/dimensions) on n examples (items/observations).

$\mathbf{X} = (x_{ij})$ is a $n \times p$-matrix with $x_{ij} :=$ the j-th variable for the i-th example

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1j} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2j} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{i1} & x_{i2} & \dots & x_{ij} & \dots & x_{ip} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nj} & \dots & x_{np} \end{bmatrix}.$$

Denote the i-th data item by $x_i \in \mathbb{R}^p$ (we will treat it as a column vector: it is the transpose of the i-th row of $\mathbf{X}$).

Assume $x_1, \dots, x_n$ are **independently and identically distributed** samples of a **random vector** X over $\mathbb{R}^p$. The j-th dimension of X will be denoted X_j.

Crabs Data ($n = 200$, $p = 5$)

Campbell (1974) studied rock crabs of the genus **leptograpsus**. One species, **L. variegatus**, had been split into two new species according to their colour: orange and blue. Preserved specimens lose their colour, so it was hoped that morphological differences would enable museum material to be classified.

Data are available on 50 specimens of each sex of each species. Each specimen has measurements on:

- the width of the frontal lobe `FL`,
- the rear width `RW`,
- the length along the carapace midline `CL`,
- the maximum width `CW` of the carapace, and
- the body depth `BD` in mm.

photo from: inaturalist.org

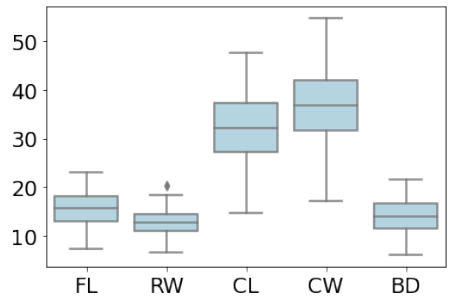
in addition to colour/species and sex (we will later view these as labels, but will ignore for now).

Crabs Data

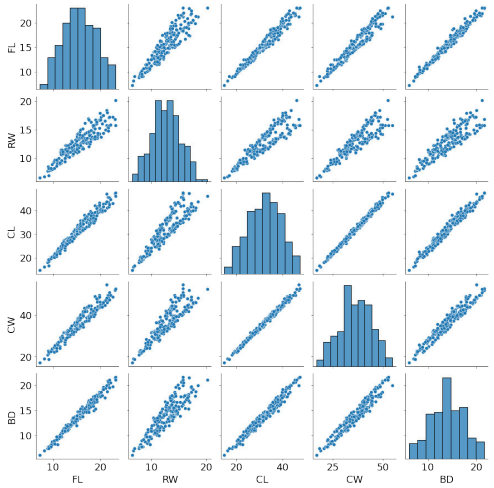
```
## look at raw data
Crabs
```

```
     FL  RW   CL   CW   BD
1   8.1  6.7 16.1 19.0  7.0
2   8.8  7.7 18.1 20.8  7.4
3   9.2  7.8 19.0 22.4  7.7
4   9.6  7.9 20.1 23.1  8.2
5   9.8  8.0 20.3 23.0  8.2
6  10.8  9.0 23.0 26.5  9.8
7  11.1  9.9 23.8 27.1  9.8
8  11.6  9.1 24.5 28.4 10.4
9  11.8  9.6 24.2 27.8  9.7
10 11.8 10.5 25.2 29.3 10.3
11 12.2 10.8 27.3 31.6 10.9
12 12.3 11.0 26.8 31.5 11.4
13 12.6 10.0 27.7 31.7 11.4
14 12.8 10.2 27.2 31.8 10.9
15 12.8 10.9 27.4 31.5 11.0
16 12.9 11.0 26.8 30.9 11.4
17 13.1 10.6 28.2 32.3 11.0
18 13.1 10.9 28.3 32.4 11.2
19 13.3 11.1 27.8 32.3 11.3
20 13.9 11.1 29.2 33.3 12.1
```

Univariate Boxplots



Simple Pairwise Scatterplots



Visualisation and Dimensionality Reduction

The summary plots are useful, but limited use if the dimensionality p is high (can be a few dozens or even thousands).

- Constrained to view data in 2 or 3 dimensions
- Approach: look for 'interesting' projections of $\mathbf{X}$ into lower dimensions
- Hope that even though p is large, considering only carefully selected $K \ll p$ dimensions is just as informative.

Unsupervised Learning

Unsupervised Learning

Goals:

- Find the variables that summarise the data / capture relevant information.
- Discover informative ways to visualise the data.
- Discover the subgroups among the observations.

It is often much easier to obtain unlabeled data than labeled data!

Unsupervised Learning

For a data item $x_i \in \mathbb{R}^p$, we want to find a lower dimensional representation $z_i \in \mathbb{R}^K$ with $K \ll p$.

Such representation should capture important aspects of the data.

Let $\text{enc}_\theta : \mathbb{R}^p \to \mathbb{R}^K$ be an encoder, parameterised by a vector $\theta \in \Theta$, with

$$z_i = \text{enc}_\theta(x_i).$$

z_i is called latent representation, latent feature or code

Another transformation $\text{dec}_\theta : \mathbb{R}^K \to \mathbb{R}^p$, called a decoder, transforms the latent representation z_i back to the original space $\mathbb{R}^p$; let

$$\widehat{x}_i = \text{dec}_\theta(z_i)$$

be the reconstructed example.

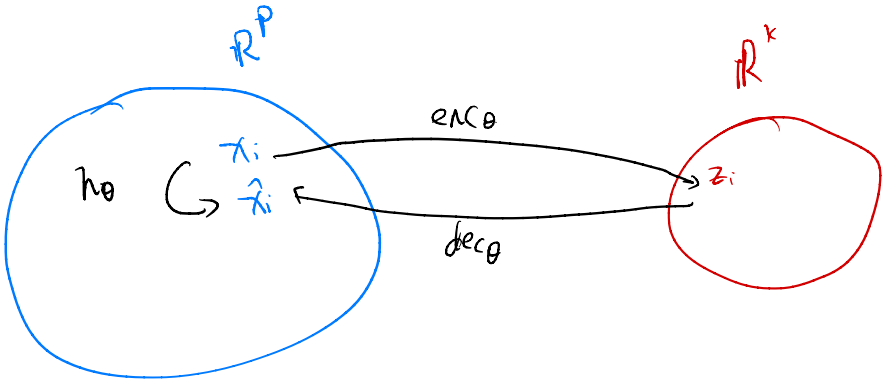
Unsupervised Learning

Let $h_\theta : \mathbb{R}^p \to \mathbb{R}^p$ be the autoencoder function defined by

$$h_\theta(x_i) = \text{dec}_\theta(\text{enc}_\theta(x_i)) = \widehat{x}_i$$

First encodes the data example into its latent representation, then decodes the latent representation

Unsupervised Learning



Unsupervised Learning

Toy Example

$\theta \in \{1, \ldots, p\}$, $K = 1$, $x_i \in \mathbb{R}^p$

$$\text{enc}_\theta(x_i) = x_{i\theta}$$
$$\text{dec}_\theta(z_i) = (0, \ldots, 0, \underbrace{z_i}_{\text{coordinate } \theta}, 0, \ldots, 0)^\intercal$$

For example, if $\theta = 1$

$$x_i = \begin{pmatrix} x_{i1} \\ x_{i2} \\ \ldots \\ x_{ip} \end{pmatrix} \xrightarrow[\text{encoder}]{} z_i = x_{i1} \xrightarrow[\text{decoder}]{} \widehat{x}_i = \begin{pmatrix} x_{i1} \\ 0 \\ \ldots \\ 0 \end{pmatrix}$$

Unsupervised Learning

Depending on the unsupervised problem, the object of interest may be

- the transformed vectors z_i,
- the parameters θ,
- the reconstructed signal $\widehat{x}_i$ or
- the decoding function dec_θ.

Unsupervised Learning

Dimensionality reduction and visualisation: latent representation $z_i \in \mathbb{R}^K$ where $K << p$

Data compression: Storing $\mathbf{X}$ requires np parameters; if θ of dimension p_θ, storing $(z_1, \ldots, z_n)$ and θ only requires $n \times K + p_\theta$ parameters

Clustering: $z_i \in \{1, \ldots, K\}$ represents the cluster membership of the data example x_i to one of K groups

Restoration: For a noisy input x_i, the autoencoder h_θ filters/restores/denoises the input x_i

Generation of new examples: Simulate pseudo-latent features $\widetilde{Z}$, and transform them via $\widetilde{X} = \text{dec}_\theta(\widetilde{Z})$to obtain pseudo-examples

Unsupervised Learning

Loss

How to quantify the error between the original example x_i and its reconstruction $\widehat{x}_i$?

Loss function $L : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_+$

$L(x_i, \widehat{x}_i)$ is a measure of the discrepancy between the example and its reconstruction.

For unsupervised learning, we will focus on the squared loss function

$$L(x_i, \widehat{x}_i) = ||x_i - \widehat{x}_i||^2 = \sum_{j=1}^{p} (x_{ij} - \widehat{x}_{ij})^2.$$

Unsupervised Learning

Risk

What is the optimal parameter θ?

Definition (Risk (unsupervised learning))

Let h_θ be an autoencoder, and L a loss function. The (population) risk of h_θ under the loss L is

$$R(h_\theta) = \mathbb{E}[L(X, h_\theta(X))]$$

where the expectation is taken with respect to the random variable X.

The risk corresponds to the expected reconstruction cost of an autoencoder h_θ

The optimal parameter θ^* under the loss L is the parameter minimising the risk:

$$\theta^* = \arg\min_{\theta \in \Theta} R(h_\theta).$$

Unsupervised Learning

Empirical Risk Minimisation

The true distribution of X being unknown, we cannot compute the true risk R, nor find θ^*.

What we have is a sample $x_1, \dots, x_n$ of the random variable X

Use instead the empirical risk, defined as

$$\widehat{R}_n(h_\theta) = \frac{1}{n} \sum_{i=1}^{n} L(x_i, h_\theta(x_i))$$

Empirical risk minimiser

$$\widehat{\theta} = \arg\min_{\theta \in \Theta} \widehat{R}_n(h_\theta).$$

Unsupervised Learning

Toy Example

$\widehat{x}_{ij} = x_{ij}$ if $j = \theta$ and 0 otherwise

The loss is

$$L(x_i, h_\theta(x_i)) = \sum_{j \neq \theta} x_{ij}^2$$

Assume that X has mean μ and covariance matrix Σ. The risk is given by

$$R(h_\theta) = \sum_{j \neq \theta} \mathbb{E}(X_j^2) = \sum_{j \neq \theta} \Sigma_{jj} + \mu_j^2 = \left(\sum_{j=1}^{p} \Sigma_{jj} + \mu_j^2 \right) - \Sigma_{\theta\theta} - \mu_\theta^2$$

The optimal parameter is

$$\theta^* = \arg\max_{\theta = 1, \dots, p} \Sigma_{\theta\theta} + \mu_\theta^2.$$

Unsupervised Learning

Toy Example

The Empirical Risk is

$$\widehat{R}_n(h_\theta) = \frac{1}{n} \sum_{i=1}^{n} \sum_{j \neq \theta} x_{ij}^2 = \left(\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{p} x_{ij}^2 \right) - \frac{1}{n} \sum_{i=1}^{n} x_{i\theta}^2$$

The empirical risk minimiser is

$$\widehat{\theta} = \arg\max_{\theta = 1, \dots, p} \frac{1}{n} \sum_{i=1}^{n} x_{i\theta}^2$$

Unsupervised Learning

We will see two unsupervised methods that can be framed as empirical risk minimisers for a given class of autoencoder

- Principal Component Analysis
- K-means clustering

Statistical Machine Learning
Hilary Term 2021

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

Principal Components Analysis (PCA)

PCA considers interesting directions to be those with greatest **variance**.

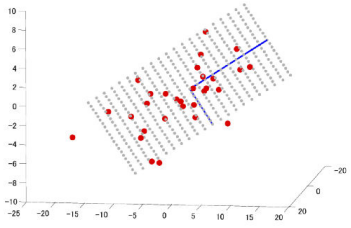
A **linear** dimensionality reduction technique: looks for a **new basis** to represent a noisy dataset.

Workhorse for many different types of data analysis (often used for data preprocessing before supervised techniques are applied).

Often the first thing to run on high-dimensional data.

Principal Components Analysis (PCA): Overview

For simplicity, we will assume from now on that our dataset is centred, i.e., we subtract the average $\bar{x}$ from each x_i.



PCA

Find an orthogonal basis $v_1, v_2, \ldots, v_p$ for the data space such that:

- The first principal component (PC) v_1 is the **direction of greatest variance** of data.
- The j-th PC v_j is the **direction orthogonal to** $v_1, v_2, \ldots, v_{j-1}$ **of greatest variance**, for $j = 2, \ldots, p$.

Principal Components Analysis (PCA): Overview

The K-dimensional representation of data item $x_i \in \mathbb{R}^p$ is the vector of projections of x_i onto first K PCs:

$$z_i = V_{1:K}^\intercal x_i = (v_1^\intercal x_i, \ldots, v_K^\intercal x_i)^\intercal \in \mathbb{R}^K,$$

where $V_{1:K} = (v_1, \ldots, v_K)$ is a p-by-K orthonormal matrix (matrix with orthonormal columns)

Reconstruction of x_i: encoder

$$\hat{x}_i = V_{1:K} z_i = V_{1:K} V_{1:K}^\intercal x_i.$$

Principal Components Analysis (PCA): Overview

For a general orthonormal p-by-K matrix A, that is with $A^\intercal A = I_K$, denote enc_A and dec_A the linear encoder and decoder defined by

$$\mathrm{enc}_A(x_i) = A^\intercal x_i$$
$$\mathrm{dec}_A(z_i) = A z_i$$

Let $h_A(x_i) = AA^\intercal x_i$ be the associated linear autoencoder.

The matrix $V_{1:K}$ is an empirical risk minimiser for the class of linear autoencoder h_A under the squared loss:

$$V_{1:K} \in \arg\min_{A \in \mathcal{A}} \frac{1}{n} \sum_{i=1}^{n} ||x_i - h_A(x_i)||^2 \qquad (1)$$

where $\mathcal{A}$ is the set of p-by-K orthonormal matrices.

PCA gives the **optimal linear reconstruction** of the original data based on a K-dimensional compression.

Eigenvalue decomposition of Σ

Our dataset is an i.i.d. sample $(x_1, \ldots, x_n)$ of a random vector $X = (X_1 \ldots X_p)^\intercal$.

Let Σ denote the p-by-p covariance matrix of the random vector X.

Σ is a

real and symmetric matrix, so there exist p eigenvectors $v_1^*, \ldots, v_p^*$ that are pairwise orthogonal and p associated eigenvalues $\lambda_1^* \geq \lambda_2^* \ldots \geq \lambda_p^*$ which satisfy the eigenvalue equation $\Sigma v_i^* = \lambda_i^* v_i^*$. In particular, the p-by-p matrix $V^* = (v_1^*, \ldots, v_p^*)$ is an orthogonal matrix:

$$V^*(V^*)^\intercal = (V^*)^\intercal V^* = I_p.$$

positive-semidefinite matrix, so the eigenvalues are non-negative:

$$\lambda_i^* \geq 0, \ \forall i.$$

The **eigenvalue decomposition** of the covariance matrix is given by

$$\Sigma = V^* \Lambda^* (V^*)^\intercal$$

where Λ^* is a diagonal matrix with eigenvalues

$$\lambda_1^* \geq \lambda_2^* \geq \cdots \geq \lambda_p^* \geq 0.$$

Eigenvalue decomposition of Σ

The **eigenvalue decomposition** of the covariance matrix is

$$\Sigma = V^* \Lambda^* (V^*)^\intercal$$

where Λ^* is a diagonal matrix with eigenvalues

$$\lambda_1^* \geq \lambda_2^* \geq \cdots \geq \lambda_p^* \geq 0.$$

The total variance of the random vector X is

$$\mathrm{TV}(X) = \mathbb{E}\left[\sum_{j=1}^{p} (X_j - \mathbb{E}[X_j])^2\right] = \sum_{j=1}^{p} \Sigma_{jj} = \mathrm{trace}(\Sigma)$$

$$= \mathrm{trace}(V^* \Lambda^* (V^*)^\intercal) = \mathrm{trace}((V^*)^\intercal V^* \Lambda^*) = \mathrm{trace}(\Lambda^*) = \sum_{j=1}^{p} \lambda_j^*$$

Eigenvalue decomposition of Σ S.

Let S be the sample covariance matrix

$$S = \frac{1}{n-1} \sum_{i=1}^{n} x_i x_i^\intercal = \frac{1}{n-1} \mathbf{X}^\intercal \mathbf{X}$$

S is a

real and symmetric matrix, so there exist p eigenvectors $v_1, \ldots, v_p$ that are pairwise orthogonal and p associated eigenvalues $\lambda_1 \geq \lambda_2 \ldots \geq \lambda_p$ which satisfy the eigenvalue equation $\Sigma v_i = \lambda_i v_i$. In particular, the p-by-p matrix $V = (v_1, \ldots, v_p)$ is an orthogonal matrix:

$$V(V)^\intercal = (V)^\intercal V = I_p.$$

positive-semidefinite matrix, so the eigenvalues are non-negative:

$$\lambda_i \geq 0, \ \forall i.$$

Eigenvalue decomposition of S

The **eigenvalue decomposition** of the sample covariance matrix S is given by

$$S = V\Lambda V^\intercal.$$

where Λ is a diagonal matrix with eigenvalues

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0.$$

The total sample variance of the data $\mathbf{X}$ is

$$\begin{aligned}\mathrm{TV}(\mathbf{X}) &= \frac{1}{n-1}\sum_{i=1}^{n}\sum_{j=1}^{p} x_{ij}^2 = \mathrm{trace}(S) \\ &= \mathrm{trace}(V\Lambda V^\intercal) = \mathrm{trace}(\Lambda) = \sum_{j=1}^{p}\lambda_j\end{aligned}$$

Derivation of the population PCs

We will first describe how to derive the population PCs $(v_1^*, \ldots, v_p^*)$, that is, the directions of maximum variance of the random vector X

Then we will give the procedure to obtain the empirical PCs (just called PCs) $(v_1, \ldots, v_p)$, based on the dataset $(x_1, \ldots, x_n)$

Deriving the First Principal Component

Assume for simplicity that X has zero mean (otherwise take $X - \mathbb{E}[X]$).
For the 1^{st} PC, we seek a derived scalar variable Z_1 of the form

$$Z_1 = a_1^\intercal X = a_{11}X_1 + a_{12}X_2 + \cdots + a_{1p}X_p$$

where $a_1 = (a_{11}, \ldots, a_{1p})^\intercal \in \mathbb{R}^p$ is a unit-norm vector, that is $a_1^\top a_1 = 1$.
For any fixed vector a_1,

$$\mathsf{var}(Z_1) = \mathsf{var}(a_1^\intercal X) = a_1^\intercal \Sigma a_1.$$

where Σ is the covariance matrix of the random vector X.

Deriving the First Principal Component

The (population) first principal component v_1^* is the unit-norm vector a_1 that maximises $\mathsf{var}(Z_1)$, that is

$$\begin{aligned}v_1^* &= \underset{a_1 \in \mathbb{R}^p}{\arg\max}\ a_1^\intercal \Sigma a_1 \\ &\text{subject to: } a_1^\intercal a_1 = 1.\end{aligned}$$

The Lagrangian of the problem is given by:

$$\mathcal{L}(a_1, \gamma_1) = a_1^\intercal \Sigma a_1 - \gamma_1 (a_1^\intercal a_1 - 1).$$

Deriving the First Principal Component

The Lagrangian of the problem is given by:

$$\mathcal{L}(a_1, \gamma_1) = a_1^\intercal \Sigma a_1 - \gamma_1 (a_1^\intercal a_1 - 1).$$

The corresponding vector of partial derivatives is

$$\frac{\partial \mathcal{L}(a_1, \gamma_1)}{\partial a_1} = 2\Sigma a_1 - 2\gamma_1 a_1.$$

Setting this to zero gives

$$\Sigma a_1 = \gamma_1 a_1$$

We recognize the eigenvector equation, i.e. a_1 must be an eigenvector of Σ and the dual variable γ_1 is the corresponding eigenvalue.

Deriving the First Principal Component

Since

$$a_1^\intercal \Sigma a_1 = \gamma_1 a_1^\intercal a_1 = \gamma_1,$$

which is the quantity we want to maximise, the first PC must be the eigenvector v_1^* associated with the largest eigenvalue λ_1^* of Σ.
The projection $Z_1 = (v_1^*)^\intercal X$ of X on the first PC then satisfies

$$\text{var}(Z_1) = \lambda_1^*.$$

Deriving the Second PC

To derive the subsequent principal components, we proceed as before.
Consider the scalar variable

$$Z_2 = a_2^\intercal X$$

where $a_2 \in \mathbb{R}^p$ is a unit-norm vector, that is $a_2^\top a_2 = 1$, and is also orthogonal to the first PC: $a_2^\intercal v_1^* = 0$.
We wish to find the vector a_2 that maximises

$$\text{var}(Z_2) = a_2^\intercal \Sigma a_2.$$

Let

$$v_2^* = \underset{a_2 \in \mathbb{R}^p}{\arg\max}\ a_2^\intercal \Sigma a_2$$
$$\text{subject to: } a_2^\intercal a_2 = 1 \text{ and } a_2^\intercal v_1^* = 0.$$

Deriving the Second PC

The Lagrangian of the problem is

$$\mathcal{L}(a_2, \gamma_2, \mu) = a_2^\intercal \Sigma a_2 - \gamma_2 (a_2^\intercal a_2 - 1) - \mu (v_1^*)^\intercal a_2.$$

The corresponding vector of partial derivatives is

$$\frac{\partial \mathcal{L}(a_2, \gamma_2, \mu)}{\partial a_2} = 2\Sigma a_2 - 2\gamma_2 a_2 - \mu v_1^* = 0.$$

Left-multiplying the above expression by $a_2^\intercal$ gives

$$a_2^\intercal \Sigma a_2 = \gamma_2.$$

Multiplying by a_2 gives

$$\Sigma a_2 = \gamma_2 a_2.$$

Deriving the Second PC

Hence a_2 must be an eigenvector of Σ with corresponding eigenvalue γ_2, where $a_2^\intercal \Sigma a_2 = \gamma_2$ is the quantity we wish to maximise.

The second PC is therefore the eigenvector v_2^* of Σ, with corresponding second highest eigenvalue λ_2^*.

We have accordingly

$$\text{var}(Z_2) = \lambda_2^*.$$

Proceeding by induction, the population PCs $v_1^*, \ldots, v_p^*$ are eigenvectors of Σ, with associated eigenvalues $\lambda_1^* \geq \lambda_2^* \geq \ldots \geq \lambda_p^* \geq 0$. The PCs are orthogonal.

(Empirical) PCs

In practice, we do not know the **true** covariance matrix Σ, so cannot compute the population PCs.

We need to replace Σ with the sample covariance matrix S.

(Empirical) PCs

For any vector $a \in \mathbb{R}^p$, let $a^\intercal x_i$ be the scalar projection of the example x_i on a

$(a^\intercal x_1, \ldots, a^\intercal x_n)$ are i.i.d. realisations of the random variable $a^\intercal X$ with sample variance

$$\frac{1}{n-1} \sum_{i=1}^{n} (a^\intercal x_i)(a^\intercal x_i)^\intercal = a^\intercal S a$$

The first two empirical PCs v_1 and v_2 are therefore given by

$$v_1 = \underset{a_1 \in \mathbb{R}^p}{\arg\max}\ a_1^\intercal S a_1 \quad \text{subject to: } a_1^\intercal a_1 = 1$$

$$v_2 = \underset{a_2 \in \mathbb{R}^p}{\arg\max}\ a_2^\intercal S a_2 \quad \text{subject to: } a_2^\intercal a_2 = 1 \text{ and } v_1^\intercal a_2 = 0.$$

and so on.

Using the same proof, the PCs $v_1, v_2, \ldots, v_p$ are eigenvectors of the sample covariance matrix S, with corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_p$.

(Empirical) PCs

The projection of the ith example x_i onto the jth PC component is therefore

$$z_{ij} = v_j^\intercal x_i$$

and we note $z_i = (z_{i1}, \ldots, z_{iK})^\intercal$ the K-dimensional latent representation of the example x_i.

In matrix notation

$$\mathbf{Z} = \mathbf{X} \times V_{1:K}$$

where

$$\mathbf{Z} = (z_{ij})_{i=1,\ldots,n, j=1,\ldots,K} = (z_1, \ldots, z_n)^\intercal$$

is an n-by-K matrix.

The matrix $V_{1:K}$ of PCs is also called the loading matrix[1]

[1] Beware that there are different definitions.

Illustration in 2D

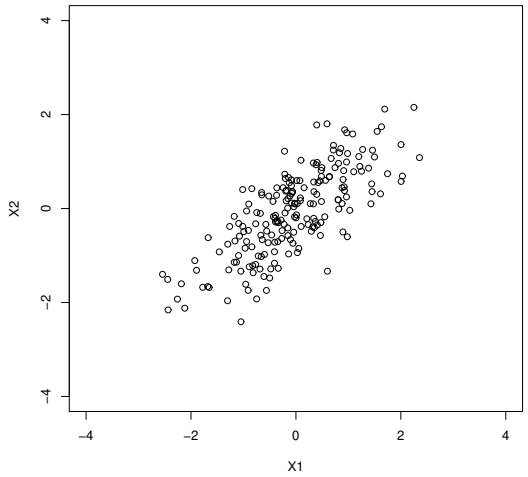


Illustration in 2D

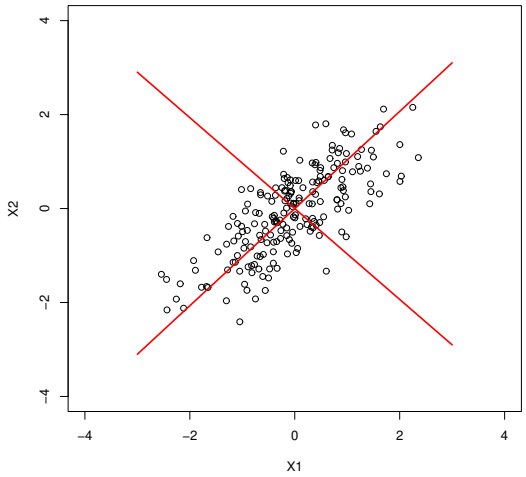
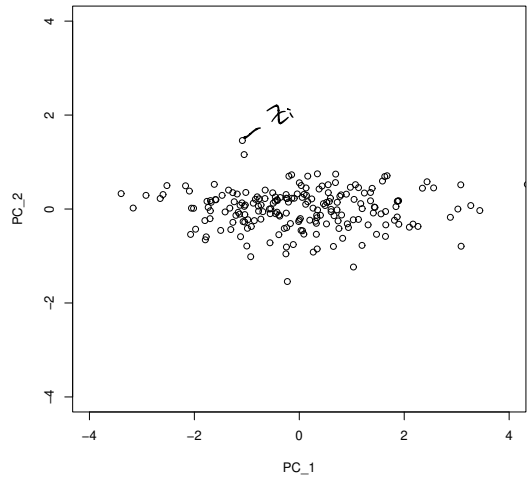


Illustration in 2D



Properties

Projections onto principal components have variance/sample variance given by the eigenvalues of Σ/S:

$$\text{var}(Z_j) = \text{var}((v_j^*)^\intercal X) = \lambda_j^*$$

$$\frac{1}{n-1}\sum_{i=1}^{n} z_{ij}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(v_j^\intercal x_i)^2 = \lambda_j$$

Projections onto principal components are uncorrelated: for $j \neq k$,

$$\text{cov}(Z_j, Z_k) = (v_j^*)^\intercal \Sigma v_k^* = \lambda_k^*(v_j^*)^\intercal v_k^* = 0$$

$$\frac{1}{n-1}\sum_{i=1}^{n} z_{ij} z_{ik} = v_j^\intercal S v_k = \lambda_k v_j^\intercal v_k = 0$$

Properties

The total variance of X is $\mathrm{TV}(X) = \sum_{i=1}^{p} \Sigma_{ii} = \sum_{j=1}^{p} \lambda_j^*$.

The total variance explained by the j^{th} PC is $\mathrm{TV}(Z_j) = \lambda_j^*$. The proportion of total variance of X explained by the first K (population) PCs is therefore

$$\frac{\sum_{j=1}^{K} \lambda_j^*}{\sum_{\ell=1}^{p} \lambda_\ell^*}.$$

Similarly, the total sample variance is $\sum_i S_{ii} = \sum_{j=1}^{p} \lambda_j$.

The proportion of total variance of $\mathbf{X}$ explained by the first K PCs is

$$\frac{\sum_{j=1}^{K} \lambda_j}{\sum_{\ell=1}^{p} \lambda_\ell}.$$

PCA as empirical risk minimisation

Let $h_A(x_i) = AA^\intercal x_i$ be the autoencoder parameterised by p-by-K matrix A, with orthonormal columns, that is $A^\intercal A = I_K$

The risk of this autoencoder for a squared loss function is

$$R(h_A) = \mathbb{E}\left[\|X - AA^\intercal X\|^2\right]$$

where the expectation is taken with respect to the random vector X.

The empirical risk of this autoencoder is

$$\widehat{R}_n(h(A)) = \frac{1}{n}\sum_{i=1}^{n} ||x_i - AA^\intercal x_i||^2.$$

PCA as empirical risk minimisation

Proposition

Let $V_{1:K}^* = (v_1^*, \ldots, v_K^*)$ be p-by-K matrix of the first K population PCs. Then $V_{1:K}^*$ minimises the risk $R(h_A)$ amongst the set of p-by-K orthonormal matrices A

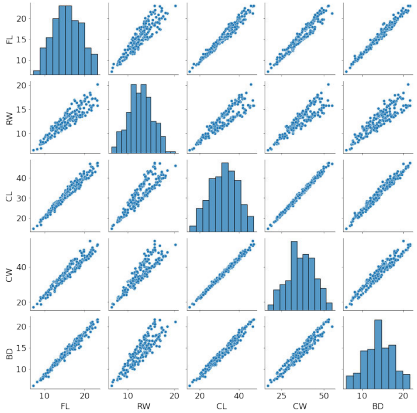
Proposition

Let $V_{1:K} = (v_1, \ldots, v_K)$ be p-by-K matrix of the first K PCs. Then $V_{1:K}$ minimises the empirical risk $\widehat{R}_n(h(A))$ amongst the set of p-by-K orthonormal matrices A.

Proof: see problem sheet.

Crabs dataset

We apply PCA to the crabs dataset ($p = 5$ variables), with $K = 5$.



Crabs dataset

Python code (scikit-learn)

```
from sklearn.decomposition import PCA
K = 5
pca = PCA(n_components = K)
pca.fit(X)
V = pca.components_.T
lam = pca.explained_variance_
Normalised_lam = pca.explained_variance_ratio_
```


Crabs dataset

We apply PCA to the crabs dataset ($p = 5$ variables), with $K = 5$.

The estimated principal components are as follows:

```
Principal Components:
   Comp.1 Comp.2 Comp.3 Comp.4 Comp.5
FL -0.289 -0.323 -0.507  0.734 -0.125
RW -0.197 -0.865  0.414 -0.148  0.141
CL -0.599  0.198 -0.175 -0.144  0.742
CW -0.662  0.288  0.491  0.126 -0.471
BD -0.284 -0.160 -0.547 -0.634  0.439
```

$= V$

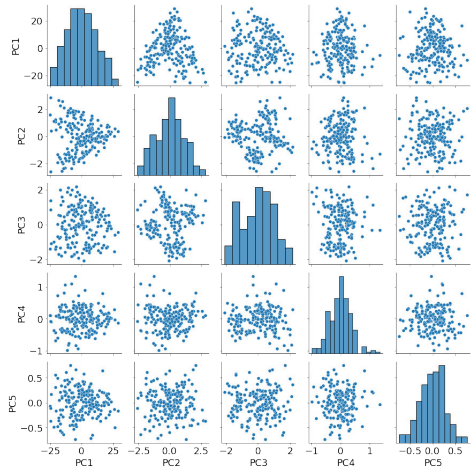
V_1 V_2 V_3 V_4 V_5

```
Importance of components:
                            Comp.1      Comp.2      Comp.3       Comp.4       Comp.5
Proportion of Variance  0.9824718 0.009055108 0.006984337 0.0009447218 0.0005440328
Cumulative Proportion   0.9824718 0.991526908 0.998511245 0.9994559672 1.0000000000
```

$\frac{\lambda_1}{\lambda_1 + \cdots + \lambda_5}$

PCA of Crabs Data

```
Z_pca = pca.transform(X)
```



What did we discover?

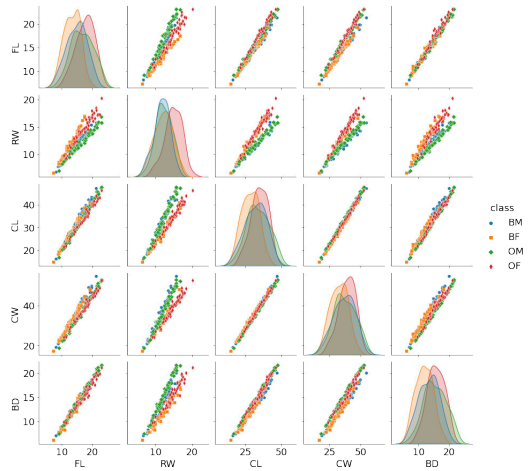
Blue or Orange.

$M \sim F$

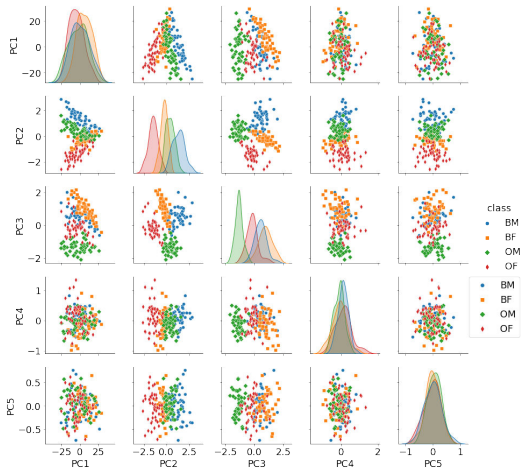
Now let us use our label information (species+sex): 4 classes

```
  [1] BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM
 [27] BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BM BF BF
 [53] BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF
 [79] BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF BF OM OM OM OM
[105] OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM
[131] OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OM OF OF OF OF OF OF
[157] OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF
[183] OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF OF
```

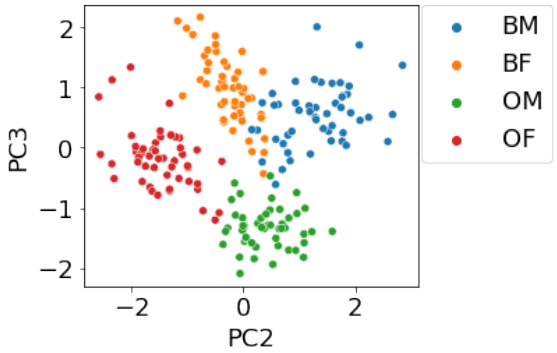
Raw Crabs Data - with labels



PCA of Crabs Data - with labels



PC 2 vs PC 3



PCA on face images: Eigenfaces

Application of PCA to dimension reduction and compression for image data

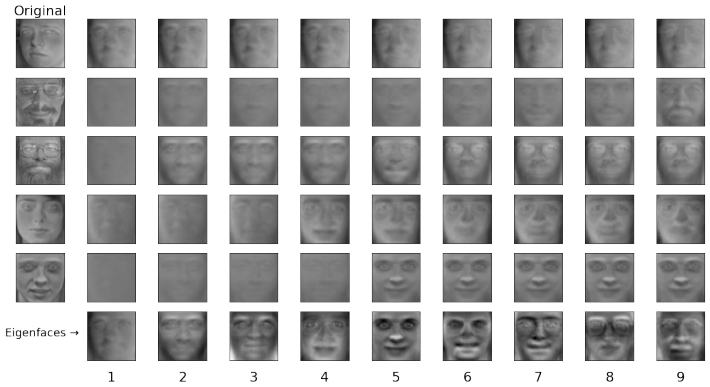
Olivetti dataset: $n = 400$ 64-by-64 images, quantized to 256 grey levels.

Dimension $p = 64 \times 64 = 4096$

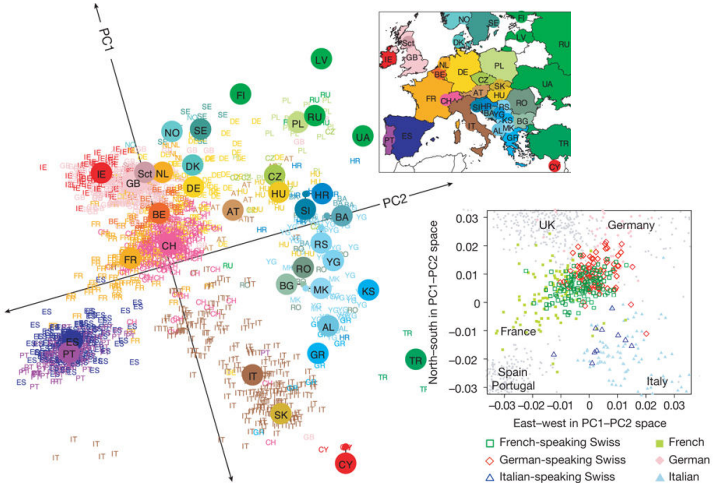
Figure: Sample images from the Olivetti faces dataset.

PCA on face images: Eigenfaces

Principal components/eigenvectors $v_1, v_2 \ldots,$ are called eigenfaces
Used as a dimensionality reduction technique. Each images can be approximated by a weighted sum of a set of K eigenfaces
$K(n+p)$ parameters to store instead of np parameters
Eigenface coefficients z_i can also be used as low dimensional feature in a supervised learning algorithm



PCA on real genomes: European genetic variation



Genes mirror geography within Europe, Nature 2008

Comments on the use of PCA

PCA commonly used to project data X onto the first K PCs giving the K-dimensional view of the data that best preserves **the first two moments**.

Although PCs are uncorrelated, scatterplots sometimes reveal structures in the data other than linear correlation.

Emphasis on variance is where the weaknesses of PCA stem from:

- Assuming large variances are meaningful (high signal-to-noise ratio)
- The PCs depend heavily on the units measurement. Where the data matrix contains measurements of vastly differing orders of magnitude, the PC will be greatly biased in the direction of larger measurement. In these cases, it is recommended to calculate PCs from $\mathrm{Corr}(X)$ instead of $\mathrm{cov}(X)$.
- Lack of robustness to outliers: variance is affected by outliers and so are PCs.
- Sample size (e.g. the number of crabs collected for each sub-species) will have an effect on the PCs.

Summary: PCA

PCA

Find an orthogonal basis $\{v_1, v_2, \ldots, v_p\}$ for the data space such that:

- The first principal component (PC) v_1 is the **direction of greatest variance** of data.
- The j-th PC v_j is the **direction orthogonal to** $v_1, v_2, \ldots, v_{j-1}$ **of greatest variance**, for $j = 2, \ldots, p$.

Eigendecomposition of the sample covariance matrix $S = \frac{1}{n-1}\sum_{i=1}^{n} x_i x_i^\top$.

$$S = V \Lambda V^\top.$$

- Λ is a diagonal matrix with eigenvalues (variances along each principal component) $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$
- V is a $p \times p$ orthogonal matrix whose columns are the p eigenvectors of S, i.e. the principal components $v_1, \ldots, v_p$

Dimensionality reduction by projecting $x_i \in \mathbb{R}^p$ onto first K principal components:

$$z_i = \left(v_1^\top x_i, \ldots, v_K^\top x_i\right)^\top \in \mathbb{R}^K.$$

Singular Value Decomposition (SVD)

SVD

Any real-valued $n \times p$ matrix $\mathbf{X}$ can be written as $X = UDV^\top$ where

- U is an $n \times n$ orthogonal matrix: $UU^\top = U^\top U = I_n$
- D is a $n \times p$ matrix with decreasing **non-negative** elements on the diagonal (the singular values) and zero off-diagonal elements.
- V is a $p \times p$ orthogonal matrix: $VV^\top = V^\top V = I_p$

SVD **always** exists, even for non-square matrices.

Fast and numerically stable algorithms for SVD are available in most packages.The relevant Python command is `scipy.linalg.svd` or `numpy.linalg.svd`.

SVD and PCA

Let $\mathbf{X} = UDV^\top$ be the SVD of the $n \times p$ data matrix $\mathbf{X}$.
Note that

$$(n-1)S = \mathbf{X}^\top\mathbf{X} = (UDV^\top)^\top(UDV^\top) = VD^\top U^\top UDV^\top = VD^\top DV^\top,$$

using orthogonality ($U^\top U = I_n$) of U.
The eigenvalues of S are thus the diagonal entries of $\Lambda = \frac{1}{n-1}D^\top D$.
We also have (using orthogonality $V^\top V = I_p$)

$$\mathbf{X}\mathbf{X}^\top = (UDV^\top)(UDV^\top)^\top = UDV^\top VD^\top U^\top = UDD^\top U^\top,$$

Gram matrix

$\mathbf{B} = \mathbf{X}\mathbf{X}^\top$, $\mathbf{B}_{ij} = x_i^\top x_j$ is called the Gram matrix of dataset $\mathbf{X}$.
$\mathbf{B}$ and $(n-1)S = \mathbf{X}^\top\mathbf{X}$ have the same nonzero eigenvalues, equal to the non-zero squared singular values of $\mathbf{X}$.

Projection:

$$\mathbf{Z} = \mathbf{X}V = UDV^\top V = UD.$$

Can be obtained by eigendecomposition of $\mathbf{B}$, less computation if $p > n$.

Statistical Machine Learning
Hilary Term 2021

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

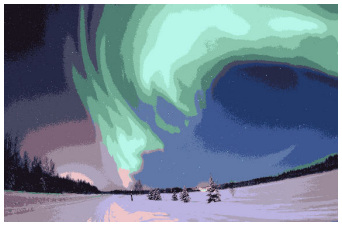
Clustering

Clustering

Many datasets consist of multiple heterogeneous subsets.

Cluster analysis: Given unlabelled data, want algorithms that automatically group the datapoints into coherent subsets/clusters.
Examples:

- market segmentation of shoppers based on browsing and purchase histories
- different types of breast cancer based on the gene expression measurements
- discovering communities in social networks
- image segmentation



Axiomatic approach

Let $n \geq 1$ an integer. A partition $\Pi = \{C_1, \ldots, C_K\}$ of the set of integers $\{1, \ldots, n\}$ is such that

- For all $k = 1, \ldots, K$, $C_k \neq \emptyset$ and $C_k \subseteq \{1, \ldots, n\}$
- For all $k \neq k'$, $C_k \cap C_{k'} = \emptyset$
- $\bigcup_{k=1}^{K} C_k = \{1, \ldots, n\}$.

C_k is known as the kth group or cluster, of size $1 \leq |C_k| \leq n$
K is the number of groups/clusters
For instance

$$\Pi = \{\underbrace{\{1,2,5\}}_{C_1}, \underbrace{\{3,4,7\}}_{C_2}, \underbrace{\{6\}}_{C_3}\}$$

is a partition of $\{1,2,3,4,5,6,7\}$ with $K = 3$ groups.
A partition can be encoded by cluster labels $(z_1, \ldots, z_n)$, where $z_i \in \{1, \ldots, K\}$ is such that $z_i = k$ if $i \in C_k$.
For the example above

$$z_1 = z_2 = z_5 = 1, \quad z_3 = z_4 = z_7 = 2, \quad z_6 = 3.$$

Axiomatic approach

Let $d = (x_1, \ldots, x_n)$ be a dataset of size n, where $x_i \in \mathbb{R}^p$ and $\rho : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}_+$ be a dissimilarity function. Clustering method is a map $\mathcal{F} : (d, \rho) \mapsto \{C_1, \ldots, C_K\}$ which takes as an input dataset d of size n and a dissimilarity function ρ and returns a partition of $\{1, \ldots, n\}$. Three basic properties required

Scale invariance. For any $\alpha > 0$, $\mathcal{F}(d, \alpha\rho) = \mathcal{F}(d, \rho)$.

Richness. For any partition $\Pi = \{C_1, \ldots, C_K\}$ of $\{1, \ldots, n\}$, there exists dissimilarity ρ, such that $\mathcal{F}(d, \rho) = \Pi$.

Consistency. If ρ and ρ' are two dissimilarities such that for all x_i, x_j the following holds:

$$x_i, x_j \text{ belong to the same cluster in } \mathcal{F}(d, \rho) \implies \rho'(x_i, x_j) \leq \rho(x_i, x_j)$$
$$x_i, x_j \text{ belong to different clusters in } \mathcal{F}(d, \rho) \implies \rho'(x_i, x_j) \geq \rho(x_i, x_j),$$

then $\mathcal{F}(d, \rho') = \mathcal{F}(d, \rho)$.

Kleinberg (2003) proves that there exists no clustering method that satisfies all three properties!

Types of Clustering

Model-free clustering:

Defined by **similarity/dissimilarity** among instances within clusters.

Model-based clustering:

Each cluster is described using a probability model. (We will not look at this type of algorithms in this course).

Model-free clustering

notion of similarity/dissimilarity between data items is central: many ways to define and the choice will depend on the dataset being analysed and dictated by domain specific knowledge

Intuitively, clustering aims to group similar items together and to place dissimilar items into different groups

two objectives can contradict each other (similarity is not a transitive relation, while being in the same cluster is an equivalence relation)

K-means

Goal: divide data items $x_i \in \mathbb{R}^p$ into a number K of clusters $C_1, \ldots, C_K$, where K is pre-defined.

Dissimilarity measure: Squared Euclidian distance

$$\rho(x_i, x_{i'}) = \|x_i - x_{i'}\|^2 = \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

Find the partition that minimises

$$\widetilde{W}(C_1, \ldots, C_K) = \frac{1}{2n} \sum_{k=1}^{K} \sum_{i,i' \in C_k} \rho(x_i, x_{i'}) = \frac{1}{2n} \sum_{k=1}^{K} \sum_{i,i' \in C_k} \|x_i - x_{i'}\|^2$$

Total number of partitions of n elements in K clusters is $S(n, K)$, the Stirling number of the second kind

$$S(100, 4) = 66955751844038698560793085292692610900187911879206859351901$$

Exhaustive search is practically impossible!

K-means

Let

$$\bar{x}_{C_k} = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$$

Empirical mean

We have (see Problem Sheet)

$$\frac{1}{2n} \sum_{i,i' \in C_k} \|x_i - x_{i'}\|^2 = \sum_{i \in C_k} \|x_i - \bar{x}_{C_k}\|^2$$

Additionally, note that

$$\bar{x}_{C_k} = \arg\min_{\mu_k \in \mathbb{R}^p} \sum_{i \in C_k} \|x_i - \mu_k\|^2$$

Hence we can write

$$\widetilde{W}(C_1, \ldots, C_K) = \min_{\mu_1, \ldots, \mu_K} \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|^2$$

K-means

Consider a joint optimisation problem over the partition $\{C_1, \ldots, C_K\}$ and **prototypes** or **cluster centroid** $(\mu_1, \ldots, \mu_K)$, with objective function

$$W(C_1, \ldots, C_K, \mu_1, \ldots, \mu_K) = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|^2.$$

K-means

We wish to find

$$(\widehat{C}_1, \ldots, \widehat{C}_K, \widehat{\mu}_1, \ldots, \widehat{\mu}_K) = \arg\min_{C_1, \ldots, C_K, \mu_1, \ldots, \mu_K} W(C_1, \ldots, C_K, \mu_1, \ldots, \mu_K)$$

Joint minimisation over both the partition and cluster centroids is computationally difficult.

K-means

$$W = \sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|^2 = \sum_{i=1}^{n} \|x_i - \mu_{z_i}\|^2$$

$W_k(\mu_k)$ $g(z_i)$

where $z_i = k$ if and only if $i \in C_k$.

Given partition $\{C_1, \ldots, C_K\}$, we can find the optimal prototypes easily by differentiating W with respect to μ_k:

$$\frac{\partial W}{\partial \mu_k} = -2 \sum_{i \in C_k} (x_i - \mu_k) = 0 \qquad \Rightarrow \mu_k = \frac{1}{|C_k|} \sum_{i \in C_k} x_i$$

Given prototypes $(\mu_1, \ldots, \mu_K)$, we can easily find the optimal partition/cluster labels by assigning each data point to the closest cluster prototype:

$$z_i = \arg\min_{k=1,\ldots,K} \|x_i - \mu_k\|^2$$

Use coordinate descent optimisation

Coordinate descent optimisation

Let $f(\theta_1, \theta_2) \in \mathbb{R}$ be some objective. Assume we want to find

$$(\widehat{\theta}_1, \widehat{\theta}_2) = \arg\min_{\theta_1, \theta_2} f(\theta_1, \theta_2)$$

but joint minimisation is difficult.
The coordinate descent algorithm proceeds as follows:

- Initialise $\theta_2^{(0)}$ to some value and set $t = 0$
- Repeat until convergence
  - Set $t \leftarrow t + 1$
  - Set $\theta_1^{(t)} = \arg\min_{\theta_1} f(\theta_1, \theta_2^{(t-1)})$
  - Set $\theta_2^{(t)} = \arg\min_{\theta_2} f(\theta_1^{(t)}, \theta_2)$
- Return $(\theta_1^{(t)}, \theta_2^{(t)})$

Each iteration of the algorithm decreases the value of the objective function:
$f(\theta_1^{(1)}, \theta_2^{(1)}) \geq f(\theta_1^{(2)}, \theta_2^{(2)}) \geq f(\theta_1^{(3)}, \theta_2^{(3)}) \dots$
The algorithm need not converge to the global minimum.

K-means

The K-means algorithm is a widely used method that returns a **local optimum** of the objective function W, using coordinate descent.

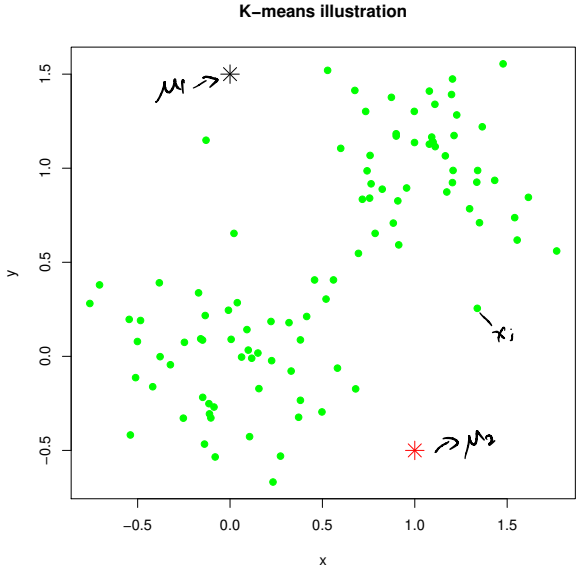
- Randomly initialise K cluster centroids $\mu_1, \dots, \mu_K$.
- **Cluster assignment:** For each $i = 1, \dots, n$, assign each x_i to the cluster with the nearest centroid,

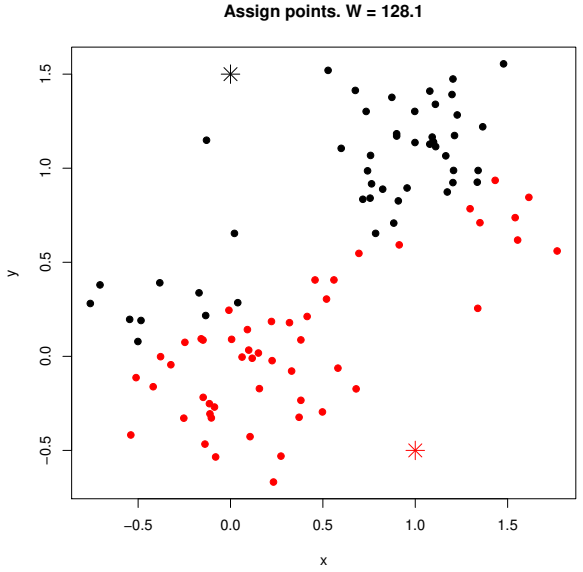
$$z_i := \arg\min_k \|x_i - \mu_k\|^2$$

- Set $C_k := \{i : z_i = k\}$ for each k.
- **Move centroids:** Set $\mu_1, \dots, \mu_K$ to the averages of the new clusters:

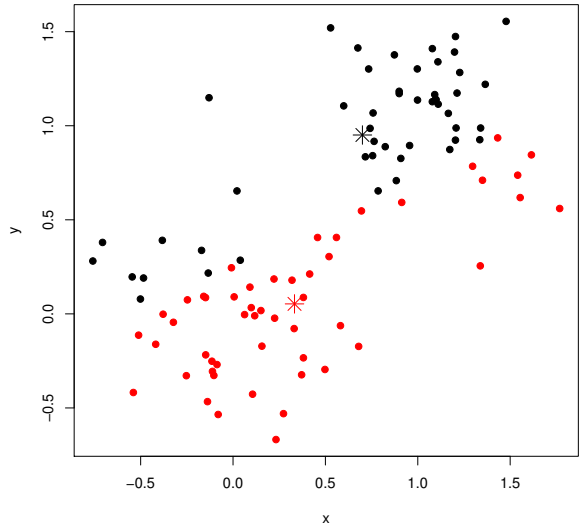
$$\mu_k := \frac{1}{|C_k|} \sum_{i \in C_k} x_i$$

- Repeat steps 2-3 until convergence.
- Return the partition $\{C_1, \dots, C_K\}$ and means $\mu_1, \dots, \mu_K$.

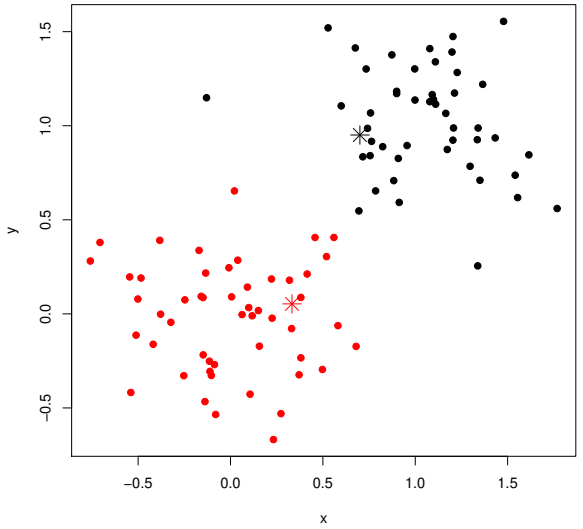




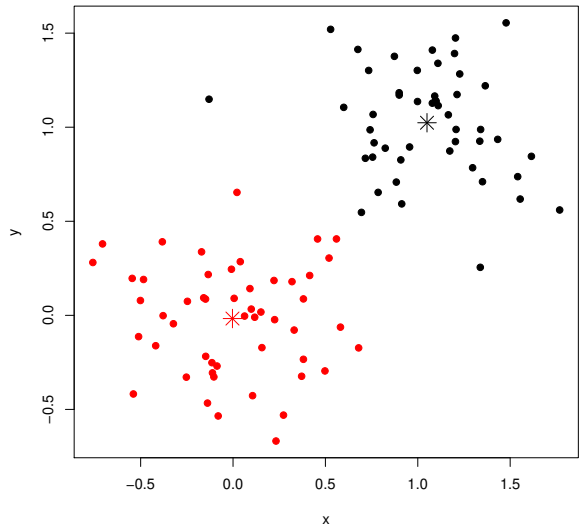
Move centroids. W = 50.979



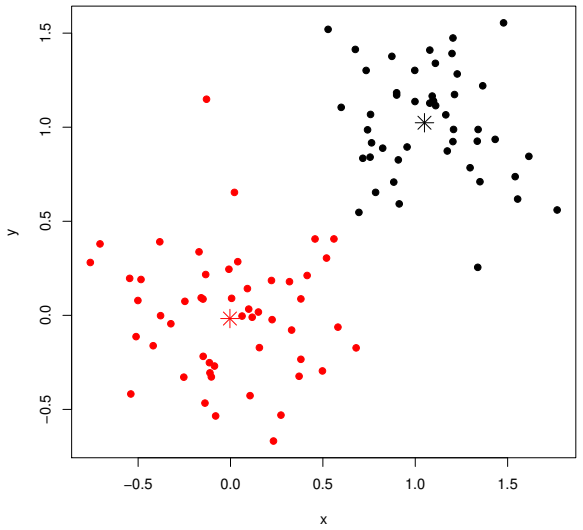
Assign points. W = 31.969

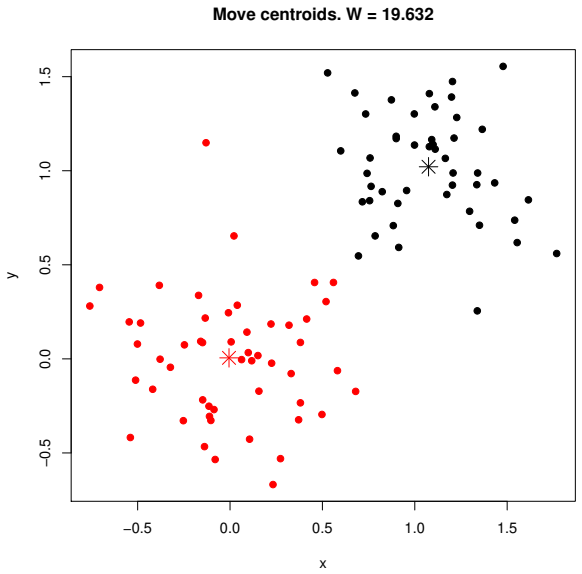


Move centroids. W = 19.72



Assign points. W = 19.688

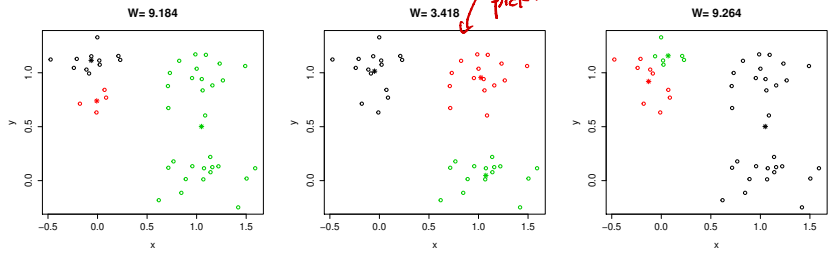




K-means

The algorithm stops in a finite number of iterations. Between steps 2 and 3, W either stays constant (same partitions) or it decreases, this implies that we never revisit the same partition. As there are only finitely many partitions, the number of iterations cannot exceed this.

The K-means algorithm need not converge to global optimum. K-means can get stuck at suboptimal configurations and the result depends on the starting configuration. Typically perform a number of runs from different configurations, and pick the end result with minimum W.



K-means as empirical risk minimisation

$(x_1, \ldots, x_n)$ are iid realisations of a random variable X

For a parameter $\theta = (\mu_1, \ldots, \mu_K)$, define the encoder function $\text{enc}_\theta : \mathbb{R}^p \to \{1, \ldots, K\}$ and decoder $\text{dec}_\theta : \{1, \ldots, K\} \to \mathbb{R}^p$ as

$$\begin{cases} z_i = \text{enc}_\theta(x_i) = \underset{k=1,\ldots,K}{\arg\min} ||x_i - \mu_k||^2 \\ \widehat{x}_i = \text{dec}_\theta(z_i) = \mu_{z_i} \end{cases}$$

and let $h_\theta(x_i) = \text{dec}_\theta(\text{enc}_\theta(x_i))$ be the autoencoder.

For a squared loss function L, define the risk and empirical risk

$$R(h_\theta) = \mathbb{E}[L(X, h_\theta(X))] = \mathbb{E}[||X - h_\theta(X)||^2]$$

$$\widehat{R}_n(h_\theta) = \frac{1}{n}\sum_{i=1}^n L(x_i, h_\theta(x_i)) = \frac{1}{n}\sum_{i=1}^n ||x_i - h_\theta(x_i)||^2$$

K-means as empirical risk minimisation

Empirical risk

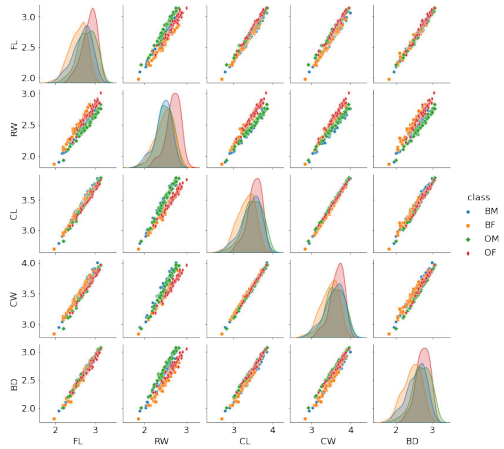
$$\begin{aligned} \widehat{R}_n(h_\theta) &= \frac{1}{n}\sum_{i=1}^n ||x_i - h_\theta(x_i)||^2 \\ &= \min_{z_1,\ldots,z_n} \frac{1}{n}\sum_{i=1}^n ||x_i - \mu_{z_i}||^2 \\ &= \min_{C_1,\ldots,C_K} \frac{1}{n} W(C_1, \ldots, C_K, \mu_1, \ldots, \mu_K) \end{aligned}$$

The value $\widehat{\theta} = (\widehat{\mu}_1, \ldots, \widehat{\mu}_K)$ that minimises W can be interpreted as the empirical risk minimiser

$$\widehat{\theta} = \underset{\theta}{\arg\min}\, \widehat{R}_n(h_\theta)$$

K-means on Crabs

Looking at the Crabs data again



K-means on Crabs

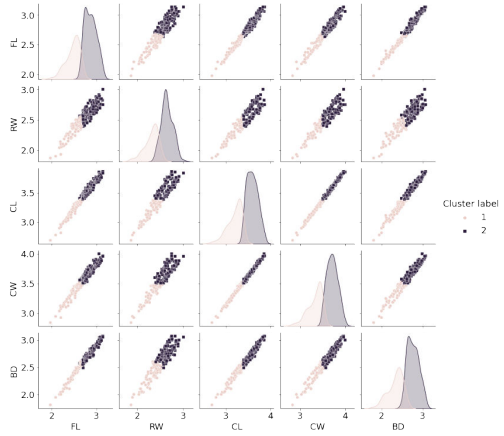
Apply K-means with 2 clusters and plot results.

```
from sklearn.cluster import KMeans
K = 2
n_init = 100
kmeans_crabs = KMeans(n_clusters = K, init='random', n_init = n_init)

# Run K-means algorithm
kmeans_crabs.fit(X)
# Collect cluster labels
z = kmeans_crabs.labels_
```


K-means on Crabs

Apply K-means with 2 clusters and plot results.



Splits the data in the middle as it favours compact clusters: the two clusters correspond to big or small crabs

K-means on Crabs

$z_{ij} = v_j^T x_i$

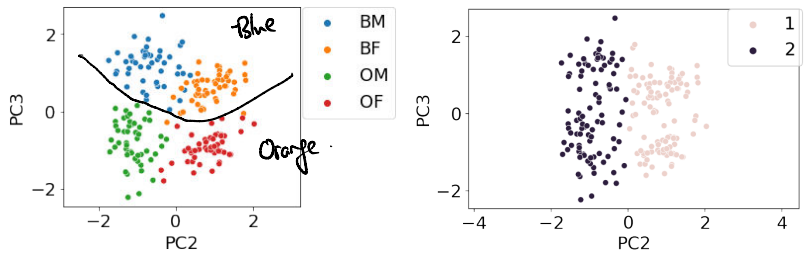
$\tilde{z}_{ij} = z_{ij} / \sqrt{\lambda_j}$

'Whiten' or 'sphere' the data using PCA

We obtain uncorrelated latent features with unit variance

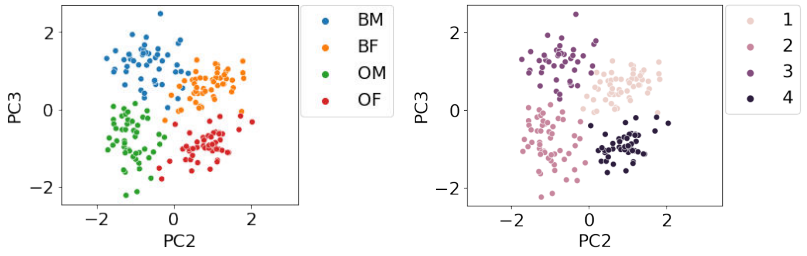
We consider the projections on the second and third components and apply K-means again with 2 clusters

K-means on Crabs



Discovers the difference between Male and Female
But the result depends crucially on sphering the data first!

K-means on Crabs with $K = 4$



We recover the class labels - note that this was fully unsupervised - we did not used the class labels to learn the clusters

K-means Additional Comments

Good practice initialisation. Randomly pick K training examples (without replacement) and set $\mu_1, \mu_2, \ldots, \mu_K$ equal to those examples

Sensitivity to distance measure. Euclidean distance can be greatly affected by measurement unit and by strong correlations. Can use Mahalanobis distance instead:

$$\|x - y\|_M = \sqrt{(x - y)^\top M^{-1} (x - y)}$$

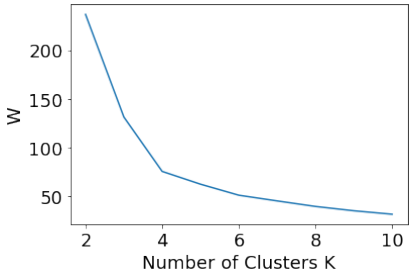
where M is positive semi-definite matrix, e.g. sample covariance.

Sensitivity to outliers.

Non-convex cluster shapes.

K-means: Choice of K

The K-means objective will always improve with larger number of clusters K.



Heuristic: Elbow

Determination of K requires using a different criterion

K-means: Choice of K

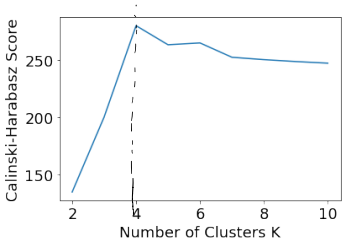
Determination of K requires using a different criterion

For example Calinski-Harabasz score

$$CH = \frac{\sum_{k=1}^{K} |C_k| \times \|\mu_k - \bar{x}\|^2}{\sum_{k=1}^{K} \sum_{i \in C_k} \|x_i - \mu_k\|^2} \times \frac{n-K}{K-1}$$

is a measure of the separability of the clusters: higher values correspond to dense and well-separated clusters

Crabs data:



Other partition based methods

Other partition-based methods with related ideas:

K-medoids: requires cluster centroids μ_k to be an observation x_i

K-medians: cluster centroids represented by a median in each dimension

K-modes: cluster centroids represented by a mode estimated from a cluster

Stochastic Optimisation

Each iteration of K-means requires a pass through whole dataset. In extremely large datasets, this can be computationally prohibitive.

Stochastic optimisation: update cluster means after assigning each data point to the closest cluster. (more on this in later lectures)

Repeat for $t = 1, 2, \ldots$ until satisfactory convergence:

Pick data item x_i either randomly or in order.

Assign x_i to the cluster with the nearest centroid,

$$z_i := \arg\min_k \|x_i - \mu_k\|^2$$

Update cluster centroid:

$$\mu_{z_i} := \mu_{z_i} + \alpha_t (x_i - \mu_{z_i})$$

where $\alpha_t > 0$ are **step sizes**.

Algorithm stochastically minimises the objective function. Convergence requires slowly decreasing step sizes:

$$\sum_{t=1}^{\infty} \alpha_t = \infty \qquad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Vector Quantisation

A related algorithm developed in the signal processing literature for **lossy data compression**.

$\mathbf{X}$ represented by $n \times p$ real numbers

Store instead:

the **codebook** of K **codewords** $\mu_1, \ldots, \mu_K$ ($K \times p$ real numbers)

for each vector x_i its cluster assignment z_i ($\lceil \log K \rceil \times n$ bits).

As with K-means, K must be specified. Increasing K improves the quality of the compressed image but worsens the data compression rate, so there is a clear tradeoff.

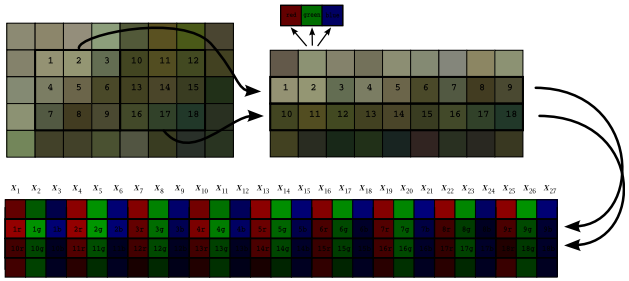
Some audio and video codecs use this method.

Stochastic optimization algorithm for K-means was originally developed for VQ.

VQ Image Compression

3×3 block VQ: View each block of 3×3 pixels in RBG colors as a single observation $x_i \in \mathbb{R}^{27}$, called a super-pixel

An image is composed of n superpixels $(x_1, \ldots, x_n)$



VQ Image Compression

Original image (24 bits/pixel, uncompressed size 1,402 kB)

VQ Image Compression

Codebook length 1024 (1.11 bits/pixel, total size 88kB)

VQ Image Compression

Codebook length 128 (0.78 bits/pixel, total size 50kB)

VQ Image Compression

Codebook length 16 (0.44 bits/pixel, total size 27kB)

Statistical Machine Learning
Hilary Term 2021

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

Supervised Learning

Supervised learning

Data contains "labels": every example is an input-output pair

classification, regression

Goal: prediction of the label of new input examples

The next two lectures

Basics of statistical learning for supervised learning

- Notations and Assumptions
- Loss, risk, optimal prediction rule
- Optimal classifier under Gaussian class distributions
- Empirical risk minimisation and plug-in methods
- Generative versus discriminative learning

Background definitions

Let $X \in \mathcal{X} \subseteq \mathbb{R}^p$ be a random variable with cumulative distribution function (cdf) F.

$$\mathbb{E}[\phi(X)] = \int_{\mathcal{X}} \phi(x) dF(x) = \begin{cases} \int_{\mathcal{X}} \phi(x) f_X(x) dx & \text{if } X \text{ is a continuous RV} \\ \sum_{x \in \mathcal{X}} \phi(x) f_X(x) & \text{if } X \text{ is a discrete RV} \end{cases}$$

where f_X denotes the probability density function of X if X is a continuous random variable, or the probability mass function of X if X is a discrete random variable.

Definition (Statistical Functional)

Let F be a cumulative distribution function. A statistical functional is a map T that maps a cdf F to a real number (or vector) $T(F)$.

For example, for a random variable X with distribution F, its median m and mean μ are both statistical functionals of F, with

$$m = F^{-1}(1/2), \quad \mu = \int_0^\infty x dF(x).$$

Notations and definitions

Let $\mathcal{X}$ be the input space and $\mathcal{Y}$ be the target space.

Regression: $\mathcal{Y} = \mathbb{R}$

Classification: $\mathcal{Y}$ is a finite set with $K \geq 2$ elements

By convention, we will assume that $\mathcal{Y} = \{1, \ldots, K\}$ if $K \geq 3$ (multiclass classification) and $\mathcal{Y} = \{1, -1\}$ if $K = 2$ (binary classification).

Notations and definitions

Definition (Prediction rule)

Let $\mathcal{X}$ be the input space and $\mathcal{Y}$ be the target space. A prediction rule is a function

$$h : \mathcal{X} \to \mathcal{Y}.$$

Let $\mathcal{F} = \{h : \mathcal{X} \to \mathcal{Y}\}$ be the set of all prediction rules.

Notations and definitions

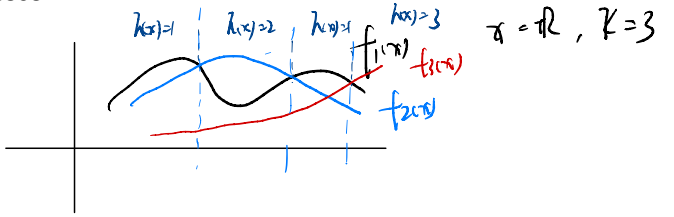
In the classification case, the prediction rule is called a classification rule, or a classifier.

A classifier can be (non-uniquely) defined via a collection of real functions $(f_k)_{k=1,\ldots,K}$:

$$h(x) = \arg\max_{k=1,\ldots,K} f_k(x)$$

where $f_k : \mathcal{X} \to \mathbb{R}$, $k = 1, \ldots, K$ are called discriminant functions, or smooth classifiers.

For two classes k and k', the set of solutions to the equation $f_k(x) = f_{k'}(x)$ is called the decision boundary between these two classes.



Notations and definitions

For a binary classifier h, one can consider a single discriminant function $f : \mathcal{X} \to \mathbb{R}$

$$h(x) = \begin{cases} 1 & \text{if } f(x) \geq 0 \\ -1 & \text{if } f(x) < 0. \end{cases}$$

The decision boundary between the two classes of the binary classifier is given by the solutions to the equation $f(x) = 0$.

Definition (Linear prediction rule)

A linear prediction rule is such that, for $x \in \mathbb{R}^p$

$$h(x) = \beta_0 + \beta^\intercal x$$

for regression, where $\beta_0 \in \mathbb{R}$, $\beta = (\beta_1, \ldots, \beta_p)^\intercal \in \mathbb{R}^p$, and

$$f_k(x) = \beta_{k0} + \beta_k^\intercal x$$

for classification, where $\beta_{k0} \in \mathbb{R}$ and $\beta_k = (\beta_{k1}, \ldots, \beta_{kp})^\intercal \in \mathbb{R}^p$ for $k = 1, \ldots, K$. The decision boundary between two classes k and k' is the hyperplane defined by the equation

$$(\beta_{k0} - \beta_{k'0}) + (\beta_k - \beta_{k'})^\intercal x = 0.$$

Notations and definitions

Dataset $d = (x_i, y_i)_{i=1,\ldots,n}$ where $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ is the ith input/target observation

Our objective is, based on the dataset d, to learn (or train) a prediction rule $\widehat{h}^{(d)} \in \mathcal{F}$.

This prediction rule is then used, for a set of new examples with known input values $(x_{n+1}, x_{n+2}, \ldots)$, to predict their target values $(\widehat{y}_{n+1}, \widehat{y}_{n+2}, \ldots)$ using the learned prediction rule

$$\widehat{y}_{n+i} = \widehat{h}^{(d)}(x_{n+i}).$$

Notations and definitions

Example (Least square linear regression)

Consider univariate regression ($\mathcal{X} = \mathcal{Y} = \mathbb{R}$), with the least square prediction rule, defined as

$$\widehat{h}^{(d)}(x) = \widehat{\beta}_0 + x\widehat{\beta}_1$$

where the least square estimates $\widehat{\beta}_0$ and $\widehat{\beta}_1$ are defined as

$$\widehat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x}) y_i}{\sum_{i=1}^n (x_i - \bar{x})^2}, \quad \widehat{\beta}_0 = \bar{y} - \widehat{\beta}_1 \bar{x}$$

where $\bar{x} = \frac{1}{n}\sum_{i=1}^n x_i$ and $\bar{y} = \frac{1}{n}\sum_{i=1}^n y_i$. The prediction rule $\widehat{h}^{(d)}(x)$ is linear.

Notations and definitions

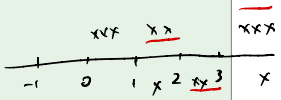
Example (Histogram binary classifier)

For $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \{-1, 1\}$ consider the histogram classifier

$$\widehat{h}^{(d)}(x) = \begin{cases} 1 & \text{if } \widehat{f}^{(d)}(x) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

where the discriminant function $\widehat{f}^{(d)}$ is defined as

$$\widehat{f}^{(d)}(x) = \sum_{i=1}^{n} \mathbb{1}_{\lfloor x_i \rfloor = \lfloor x \rfloor} (\mathbb{1}_{y_i = 1} - \mathbb{1}_{y_i = -1}).$$



This simple classifier is piecewise constant, and assigns a point x in the interval $[\lfloor x \rfloor, \lfloor x \rfloor + 1)$ to the most frequent class in that interval in the dataset d.

Assumptions

The dataset $d = (x_i, y_i)_{i=1,\ldots,n}$ is a realisation from a random sample $D = (X_i, Y_i)_{i=1,\ldots,n}$ where $(X_1, Y_1), \ldots, (X_n, Y_n)$ are independent and identically distributed (iid) random variables with the same distribution as (X, Y).

Let P_0 denote the distribution of (X, Y), called the true data distribution, or population distribution.

Note that $\widehat{h}^{(D)}$ therefore denotes a random prediction rule, as it is parameterised by the random sample D; the prediction rule $\widehat{h}^{(d)}$ is a realisation of $\widehat{h}^{(D)}$.

Decomposition of the variance and class separability

For classification with K classes, using the law of total variance, we have the following decomposition of the covariance matrix of the input vector X

$$\begin{aligned} \operatorname{cov}(X) &= \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^\intercal] \\ &= \underbrace{\mathbb{E}[\operatorname{cov}(X \mid Y)]}_{\text{within-class variance } W} + \underbrace{\operatorname{cov}(\mathbb{E}[X \mid Y])}_{\text{between-class variance } B}. \end{aligned}$$

Let $\mu_k = \mathbb{E}[X|Y = k]$, $\Sigma_k = \operatorname{cov}(X \mid Y = k)$ and $\mu = \mathbb{E}[X]$, and $\pi_k = \Pr(Y = k)$. The two covariance terms can be expressed as

$$B = \operatorname{cov}(\mathbb{E}[X \mid Y]) = \sum_{k=1}^{K} \pi_k (\mu_k - \mu)(\mu_k - \mu)^\intercal$$

$$W = \mathbb{E}[\operatorname{cov}(X \mid Y)] = \sum_{k=1}^{K} \pi_k \Sigma_k$$

Decomposition of the variance and class separability

If $X \in \mathbb{R}$, the ratio

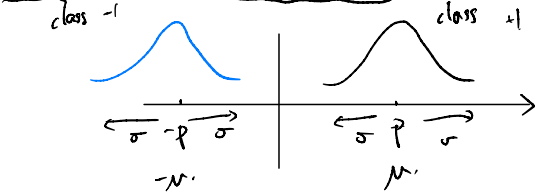
$$\frac{\operatorname{cov}(\mathbb{E}[X \mid Y])}{\mathbb{E}[\operatorname{cov}(X \mid Y)]} \geq 0$$

between the between-class and within-class variances is a measure of the separability between the classes. Larger values indicate more separated classes.

For instance, if $\Pr(Y = 1) = \Pr(Y = -1) = 1/2$ and $X \mid Y = y \sim \mathcal{N}(y\mu, \sigma^2)$, for some $\mu \geq 0$ and $\sigma > 0$, we have

$$\mathbb{E}[\operatorname{cov}(X \mid Y)] = \sigma^2, \quad \operatorname{cov}(\mathbb{E}[X \mid Y]) = \mu^2.$$

σ^2 controls the within-class variance, and μ^2 the between-class variance.



In order to formalise mathematically the supervised learning task, we need to answer the following questions:

- How do we quantify the accuracy of a prediction rule?

In order to formalise mathematically the supervised learning task, we need to answer the following questions:

- How do we quantify the accuracy of a prediction rule?
  - Loss function and risk

In order to formalise mathematically the supervised learning task, we need to answer the following questions:

- How do we quantify the accuracy of a prediction rule?
  - Loss function and risk
- If we were to know the true data distribution, what would be the associated (optimal) prediction rule?

In order to formalise mathematically the supervised learning task, we need to answer the following questions:

- How do we quantify the accuracy of a prediction rule?
  - Loss function and risk
- If we were to know the true data distribution, what would be the associated (optimal) prediction rule?
  - Bayes prediction rule

In order to formalise mathematically the supervised learning task, we need to answer the following questions:

- How do we quantify the accuracy of a prediction rule?
  - Loss function and risk
- If we were to know the true data distribution, what would be the associated (optimal) prediction rule?
  - Bayes prediction rule
- Based on a dataset of size n, how to choose a prediction rule whose accuracy is close to the optimal one?

In order to formalise mathematically the supervised learning task, we need to answer the following questions:

- How do we quantify the accuracy of a prediction rule?
  - Loss function and risk
- If we were to know the true data distribution, what would be the associated (optimal) prediction rule?
  - Bayes prediction rule
- Based on a dataset of size n, how to choose a prediction rule whose accuracy is close to the optimal one?
  - Empirical risk minimisation

In order to formalise mathematically the supervised learning task, we need to answer the following questions:

- How do we quantify the accuracy of a prediction rule?
  - Loss function and risk
- If we were to know the true data distribution, what would be the associated (optimal) prediction rule?
  - Bayes prediction rule
- Based on a dataset of size n, how to choose a prediction rule whose accuracy is close to the optimal one?
  - Empirical risk minimisation
  - Plug-in approach

Loss function

Consider a prediction rule $h : \mathcal{X} \to \mathcal{Y}$ where $h(x) \in \mathcal{Y}$ is the predicted target for an input x.

We need a way to measure how "bad" it is to predict $h(x)$ when the true target is indeed y.

Definition

Let $\mathcal{Y}$ be the target space. A loss function is a non-negative function

$$L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+.$$

The loss $L(y, h(x))$ represents the "cost" of predicting a target $h(x)$ when the true target is y. We typically have $L(y, h(x)) = 0$ when $h(x) = y$, and $L(y, h(x))$ increases as y and $h(x)$ are "further away".

Loss function

Classical loss functions

Regression: Squared error loss

$$L(y, h(x)) = (y - h(x))^2$$

Classification: $0-1$ loss

$$L(y, h(x)) = \begin{cases} 0 & \text{if } h(x) = y \\ 1 & \text{if } h(x) \neq y \end{cases}$$

Risk

Definition (Risk - supervised learning)

For a given loss function L, the (population) risk $R(h)$ of a prediction rule h is the expected loss

$$R(h) = \mathbb{E}\left[L(Y, h(X))\right]$$

where the expectation is taken over the true (unknown) joint distribution of (X, Y).

The risk is therefore given by

$$R(h) = \begin{cases} \mathbb{E}\left[(Y - h(X))^2\right] & \text{under the squared loss} \\ \Pr(Y \neq h(X)) & \text{under the } 0-1 \text{ loss.} \end{cases}$$

Bayes prediction rule

We assume for simplicity that $R(h)$ admits a unique minimum over the set of prediction rules $\mathcal{F}$.

The optimal prediction rule, called Bayes prediction rule, is the function h^* that minimises the risk $R(h)$.

For classification: known as Bayes classifier

Definition (Bayes prediction rule)

The Bayes prediction rule is given by

$$h^* = \arg\min_{h \in \mathcal{F}} R(h).$$

Bayes prediction rule

Proposition

For any $x \in \mathcal{X}$, we have

$$h^*(x) = \arg\min_{y^* \in \mathcal{Y}} \mathbb{E}[L(Y, y^*) | X = x],$$

where the expectation is taken over the true (unknown) conditional distribution of Y given $X = x$.

Bayes prediction rule

Proof.

Note that for any function f, $\mathbb{E}[f(X,Y)] = \mathbb{E}_X[\mathbb{E}_Y[f(X,Y)|X]]$. It follows that, for any prediction rule h,

$$R(h) = \mathbb{E}_X[\mathbb{E}_Y[L(Y,h(X)) \mid X]].$$

As, for any x,

$$\begin{aligned}\mathbb{E}_Y[L(Y,h(X)) \mid X = x] &\geq \min_{y\in\mathcal{Y}} \mathbb{E}_Y[L(Y,y) \mid X = x] \\ &= \mathbb{E}_Y[L(Y,h^*(X)) \mid X = x]\end{aligned}$$

we therefore obtain, for any $h \in \mathcal{F}$, $R(h) \geq R(h^*)$. □

Bayes prediction rule

Let $F_x(y) = \Pr(Y \leq y \mid X = x)$ denote the conditional cdf of Y given $X = x$. For each x, the Bayes prediction $h^*(x)$ is a statistical functional of the conditional cdf F_x:

$$h^*(x) = T(F_x)$$

with

$$T(F_x) = \arg\min_{y^*\in\mathcal{Y}} \int_{\mathcal{Y}} L(y,y^*)dF_x(y).$$

Bayes prediction rule

The risk $R(h^*)$ of the Bayes prediction function h^* is called the Bayes risk

h^* is optimal in the sense that it achieves the lowest risk amongst all prediction rules: for any function $h \in \mathcal{F}$,

$$R(h) \geq R(h^*).$$

The difference

$$R(h) - R(h^*) \geq 0$$

is called the excess risk of the prediction rule h.

Bayes prediction rule

In the case of the squared loss function we have

$$\begin{aligned}h^*(x) &= \mathbb{E}[Y|X = x] \\ R(h^*) &= \mathbb{E}_X[\mathsf{var}(Y \mid X)] \\ R(h) - R(h^*) &= \mathbb{E}_X[(h(X) - h^*(X))^2]\end{aligned}$$

Proof: See problem sheets

Bayes prediction rule

For example, let (X, Y) be defined as

$$X \sim \mathsf{U}(0, 1)$$

and given $X = x$,

$$Y = m(x) + \varepsilon$$

where ε is independent of X, with $\mathbb{E}[\varepsilon] = 0$, $\mathsf{var}(\varepsilon) = \sigma^2$.
The Bayes prediction rule is

$$h^*(x) = \mathbb{E}[Y|X = x] = \mathbb{E}[m(X) + \epsilon|X = x] = m(x)$$

with Bayes risk

$$R(h^*) = \mathbb{E}\,\mathsf{var}(Y|X) = \sigma^2$$

The risk of a prediction rule h is

$$R(h) = \mathbb{E}[(Y - h(X))^2] = \mathbb{E}[(m(X) + \varepsilon - h(X))^2]$$

$$= \mathbb{E}[(m(X) - h(X))^2] + \mathbb{E}[\varepsilon^2] = \underbrace{\int_0^1 (m(x) - h(x))^2 dx}_{\text{Excess risk}} + \sigma^2$$

Bayes prediction rule

In the case of the $0 - 1$ loss

$$h^*(x) = \arg\max_{k \in \mathcal{Y}} \Pr(Y = k|X = x)$$

$$R(h^*) = 1 - \mathbb{E}_X \left[\max_{k \in \mathcal{Y}} \Pr(Y = k|X)\right]$$

The discriminant functions of the optimal classifier are therefore

$$f_k^*(x) = \Pr(Y = k|X = x).$$

Proof: See problem sheets

Bayes classifier under Gaussian class distributions

Joint distribution of (X, Y), where $X \in \mathbb{R}^p$ and $Y \in \{1, \ldots, K\}$:

$$\Pr(Y = k) = \pi_k$$

$$X \mid Y = k \sim \mathcal{N}(\mu_k, \Sigma_k)$$

where $\pi_k \geq 0$ are the class probabilities, with $\sum_{k=1}^K \pi_k = 1$, $\mu_k \in \mathbb{R}^p$ is the mean of observations in class k and Σ_k is the $p \times p$ covariance matrix of observations in class k.

Denote

$$g_k(x) = (2\pi)^{-p/2} |\Sigma_k|^{-1/2} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k)\right)$$

the conditional pdf of X given $Y = k$.

Bayes classifier under Gaussian class distributions

$\sum_{l=1}^K \pi_l g_l(x)$ doesn't depend on k.

An application of Bayes' theorem gives

$$\Pr(Y = k|X = x) = \frac{\pi_k g_k(x)}{\sum_{\ell=1}^K \pi_\ell g_\ell(x)}.$$

$$\Pr(Y = k \mid X = x) = \frac{\Pr(X = x \mid Y = k) \Pr(Y = k)}{\Pr(X = x)}$$

Bayes classifier under Gaussian class distributions

Under a $0-1$ loss, the (optimal) Bayes classifier is

$$\begin{aligned} h^*(x) &= \underset{k\in\{1,\dots,K\}}{\arg\max}\ \Pr(Y=k|X=x) \\ &= \underset{k\in\{1,\dots,K\}}{\arg\max}\ \pi_k g_k(x) \\ &= \underset{k\in\{1,\dots,K\}}{\arg\max}\ \log(\pi_k)+\log(g_k(x)) \\ &= \underset{k\in\{1,\dots,K\}}{\arg\max}\ f_k(x) \end{aligned}$$

where the discriminant functions f_k, $k=1,\dots,K$, are defined as

$$\begin{aligned} f_k(x) &= \log(\pi_k) - \frac{1}{2}\log|\Sigma_k| - \frac{1}{2}(x-\mu_k)^\intercal\Sigma_k^{-1}(x-\mu_k) \\ &= \log(\pi_k) - \frac{1}{2}(\log|\Sigma_k| + \mu_k^\intercal\Sigma_k^{-1}\mu_k) + \mu_k^\intercal\Sigma_k^{-1}x - \frac{1}{2}x^\intercal\Sigma_k^{-1}x \\ &= a_k + b_k^\intercal x + x^\intercal c_k x \end{aligned}$$

where $a_k = \log(\pi_k) - \frac{1}{2}(\log|\Sigma_k| + \mu_k^\intercal\Sigma_k^{-1}\mu_k)$, $b_k = \Sigma_k^{-1}\mu_k$ and $c_k = -\frac{1}{2}\Sigma_k^{-1}$.

Bayes classifier under Gaussian class distributions

Discriminant functions

$$f_k(x) = a_k + b_k^\intercal x + x^\intercal c_k x$$

are quadratic functions of x

The decision boundary between class k and class k' is obtained by looking at the solutions to the equation $f_k(x) - f_{k'}(x) = 0$:

$$a_k + b_k^\intercal x + x^T c_k x - (a_{k'} + b_{k'}^\intercal x + x^\intercal c_{k'} x) = a_* + b_*^\intercal x + x^\intercal c_* x = 0.$$

where $a_* = a_k - a_{k'}$, $b_* = b_k - b_{k'}$ and $c_* = c_k - c_{k'}$.

The decision boundary is therefore given by the roots of a quadratic function of x.

Bayes classifier under Gaussian class distributions

Equal covariances

If the covariance matrices are all equal, $\Sigma_k = \Sigma$ for all k, the discriminant functions simplify to

$$\begin{aligned} f_k(x) &= \log(\pi_k) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}\underbrace{(x-\mu_k)^\intercal\Sigma^{-1}(x-\mu_k)}_{\text{Mahalanobis distance}} \\ &= -\frac{1}{2}(x^T\Sigma^{-1}x + \log|\Sigma|) + \log(\pi_k) - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \mu_k^T\Sigma^{-1}x. \end{aligned}$$

As the first term $-\frac{1}{2}(x^T\Sigma^{-1}x + \log|\Sigma|)$ does not depend on k,

$$\underset{k\in\{1,\dots,K\}}{\arg\max}\ f_k(x) = \underset{k\in\{1,\dots,K\}}{\arg\max}\ \widetilde{f}_k(x)$$

with discriminant functions

$$\begin{aligned} \widetilde{f}_k(x) &= \log(\pi_k) - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \mu_k^T\Sigma^{-1}x \\ &= a_k + b_k^T x \end{aligned}$$

where $a_k = \log(\pi_k) - \frac{1}{2}(\mu_k^T\Sigma^{-1}\mu_k)$, $b_k = \Sigma^{-1}\mu_k$.

Bayes classifier under Gaussian class distributions

Equal covariances

Discriminant functions

$$\widetilde{f}_k(x) = a_k + b_k^T x$$

are therefore linear, and the decision boundary between two classes is an hyperplane, solution to the equation

$$a_* + b_*^\intercal x = 0$$

where $a_* = a_k - a_{k'}$ and $b_* = b_k - b_{k'}$. The resulting optimal classifier is therefore a linear classifier.

Bayes classifier under Gaussian class distributions

Example

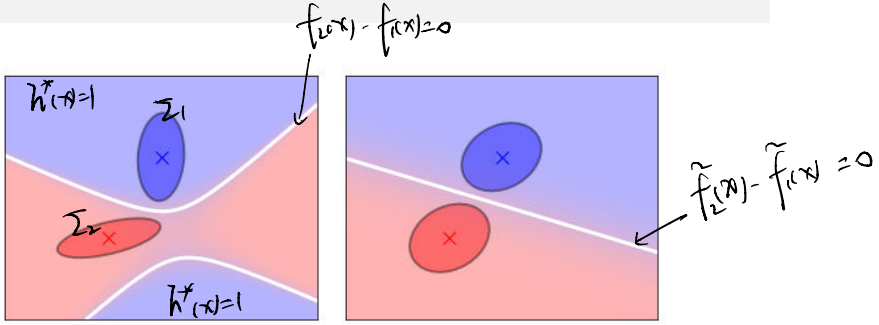


Figure: Optimal Bayes classifiers (background color) and associated decision boundaries (white line) under Gaussian class distributions, with (left) different covariances within each class and (right) the same covariance. Crosses correspond to the class means, and ellipses represent the contour of constant pdf of the Gaussian class distributions.

Equal covariances: discriminant coordinates

In order to gain some intuition, consider first the binary case, with a discriminative function $\eta(x) = \widetilde{f}_2(x) - \widetilde{f}_1(x)$

$$\begin{aligned}\eta(x) &= \log(\pi_2) - \log(\pi_1) - \frac{1}{2}\left(\mu_2^\intercal\Sigma^{-1}\mu_2 - \mu_1^\intercal\Sigma^{-1}\mu_1\right) + (\mu_2-\mu_1)^\intercal\Sigma^{-1}x \\ &= \log\frac{\pi_2}{\pi_1} - \frac{1}{2}(\mu_2-\mu_1)^\intercal\Sigma^{-1}(\mu_2+\mu_1) + (\mu_2-\mu_1)^\intercal\Sigma^{-1}x\end{aligned}$$

Letting

$$x_0 = \frac{1}{2}(\mu_1+\mu_2) - (\mu_2-\mu_1)\frac{\log(\pi_2/\pi_1)}{(\mu_2-\mu_1)^\intercal\Sigma^{-1}(\mu_2-\mu_1)}$$

we obtain

$$\eta(x) = (\mu_2-\mu_1)^\intercal\Sigma^{-1}(x-x_0).$$

Note that x_0 lies on the line passing through the class means μ_1 and μ_2, and $x_0 = \frac{1}{2}(\mu_1+\mu_2)$ is half-way if $\pi_2 = \pi_1 = 1/2$.

Equal covariances: discriminant coordinates

For a vector $x \in \mathbb{R}^p$, writing $x^\bullet = \Sigma^{-1/2}x$, we obtain

$$\eta(x) = (\mu_2^\bullet - \mu_1^\bullet)^\intercal(x^\bullet - x_0^\bullet).$$

The decision boundary therefore only depends on x through the projection of $x^\bullet = \Sigma^{-1/2}x$ onto the vector of the difference of the transformed class means $(\mu_2^\bullet - \mu_1^\bullet)$.

Hence, for the purpose of classification, only the one-dimensional projection $(\mu_2^\bullet - \mu_1^\bullet)\Sigma^{-1/2}x$ of the p-dimensional vector x is relevant.

Called discriminant coordinate of the vector x

Equal covariances: discriminant coordinates

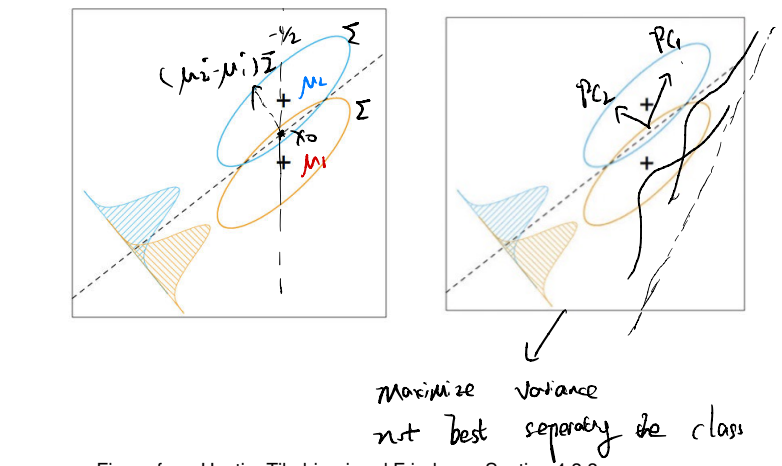


Figure from Hastie, Tibshirani and Friedman, Section 4.3.3

Equal covariances: discriminant coordinates

For general K,

$$f_k(x) = \log(\pi_k) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}\underbrace{(x-\mu_k)^\intercal\Sigma^{-1}(x-\mu_k)}_{\text{Mahalanobis distance}}$$

The classifier relies on the Mahalanobis distances

$$(x-\mu_k)^\intercal\Sigma^{-1}(x-\mu_k)$$

between the vector x to the class means or, equivalently, on the squared Euclidian distances

$$(x^\bullet-\mu_k^\bullet)^\intercal(x^\bullet-\mu_k^\bullet) = \|x^\bullet-\mu_k^\bullet\|^2$$

between the transformed $x^\bullet$ and $\mu_k^\bullet$.

Equal covariances: discriminant coordinates

The K sphered class means $\mu_k^\bullet$ lie in an affine manifold H_{K-1} of dimension lower or equal to $K-1$.

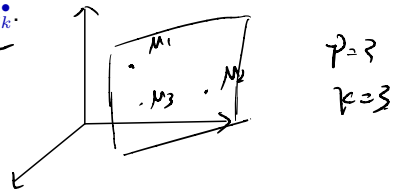
When looking at the closest class mean, one can ignore distances orthogonal to this subspace, as they contribute equally for each class.

We can therefore project the $x^\bullet$ onto this class-spanning subspace H_{K-1}.

This corresponds to finding the principal components subspace of the transformed means, or similarly, to find the eigendecomposition of the sphered between-class covariance

$$B^\bullet = \operatorname{cov}(\mathbb{E}[\Sigma^{-1/2}X|Y]) = \Sigma^{-1}\operatorname{cov}(\mathbb{E}[X|Y]) = \sum_{k=1}^K \pi_k(\mu_k^\bullet-\mu^\bullet)(\mu_k^\bullet-\mu^\bullet)^\intercal$$

where $\mu^\bullet = \sum_{k=1}^K \pi_k\mu_k^\bullet$.



Equal covariances: discriminant coordinates

Denote

$$B^\bullet = U^\bullet D^\bullet U^{\bullet\intercal}$$

the eigendecomposition of $B^\bullet$.

The columns $u_\ell^\bullet$ of $U^\bullet$ in sequence define the coordinates of the optimal subspaces.

We therefore define, for $\ell = 1,\dots,K-1$

$$z_\ell = (u_\ell^\bullet)^\intercal x^\bullet = (u_\ell^\bullet)^\intercal\Sigma^{-1/2}x$$

the ℓth discriminant coordinate of the vector x.

Equal covariances: discriminant coordinates

Fisher arrived at the same decomposition with a different route, without refering to Gaussian distributions.

Assume we want to find a linear combination $Z = a^\intercal X$ such that the between-class variance is maximised relative to the within-class variance.

That is, we want to maximise

between class variance of projection

$$\frac{\operatorname{cov}(\mathbb{E}[a^\intercal X|Y])}{\mathbb{E}[\operatorname{cov}(a^\intercal X|Y)]} = \frac{a^\intercal Ba}{a^\intercal\Sigma a}$$

Setting $u = \Sigma^{1/2}a$ yields

$$\frac{u^\intercal B^\bullet u}{u^\intercal u}$$

Maximisation over u is achieved by the first eigenvector of $B^\bullet$, which corresponds to $u_1^\bullet$ as above.

The next eigenvector $u_2^\bullet$ is obtained by finding the vector orthogonal to $u_1^\bullet$ that maximises $\frac{u_2^\intercal B^\bullet u_2}{u_2^\intercal u_2}$, etc.

Empirical Risk

For a prediction rule h and a dataset $d = (x_i, y_i)_{i=1,\dots,n}$, the empirical risk, or training error, is defined as

$$\widehat{R}_n(h) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i)).$$

Represents the average error over the dataset d used to train/learn the classifier.

The population risk $R(h)$, defined earlier, is also called the generalisation error or out-of-sample error.

Represents the expected error for a new observation out of the training set d, and is the quantity we aim to minimise.

The generalisation gap is defined as the difference between the generalisation error and the training error

$$R(h) - \widehat{R}_n(h).$$

Empirical Risk

Our objective is to find a prediction rule with small generalisation error.

The conditional distribution of $Y|X = x$ being unknown, we cannot calculate the optimal prediction rule h^*.

For a given function h, we cannot either compute its generalisation error $R(h)$ as this involves an expectation with respect to the true unknown distribution of (X, Y).

All we have is a realisation $d = (x_i, y_i)_{i=1,\dots,n}$ from a random sample $D = (X_i, Y_i)_{i=1,\dots,n}$.

Learning approaches

Bayes (optimal) prediction rule:

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

$$h^*(x) = T(F_x) \text{ with } T(F_x) = \arg\min_{y^* \in \mathcal{Y}} \int_{\mathcal{Y}} L(y, y^*) dF_x(y).$$

Two broad families of learning methods that we might consider

Empirical Risk Minimisation (ERM): Replace the population risk $R(h)$ by the empirical risk $\widehat{R}_n(h)$, and minimise this function over some set of functions $h \in \mathcal{H} \subset \mathcal{F}$

$$\widehat{h}^{(d)} = \arg\min_{h \in \mathcal{H}} \widehat{R}_n(h).$$

Plug-in methods: Obtain some estimate $\widehat{F}_x$ of the conditional distribution of Y given $X = x$, and set

$$\widehat{h}^{(d)}(x) = T(\widehat{F}_x) = \arg\min_{y^* \in \mathcal{Y}} \int_{\mathcal{Y}} L(y, y^*) d\widehat{F}_x(y).$$

Learning approaches

Bayes (optimal) prediction rule:

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

$$h^*(x) = T(F_x) \text{ with } T(F_x) = \arg\min_{y^* \in \mathcal{Y}} \int_{\mathcal{Y}} L(y, y^*) dF_x(y).$$

Two broad families of learning methods that we might consider

Empirical Risk Minimisation (ERM): Replace the population risk $R(h)$ by the empirical risk $\widehat{R}_n(h)$, and minimise this function over some set of functions $h \in \mathcal{H} \subset \mathcal{F}$

$$\widehat{h}^{(d)} = \arg\min_{h \in \mathcal{H}} \widehat{R}_n(h).$$

Plug-in methods: Obtain some estimate $\widehat{F}_x$ of the conditional distribution of Y given $X = x$, and set

$$\widehat{h}^{(d)}(x) = T(\widehat{F}_x) = \arg\min_{y^* \in \mathcal{Y}} \int_{\mathcal{Y}} L(y, y^*) d\widehat{F}_x(y).$$

Learning approaches

The two different approaches address the prediction problem by considering tasks of increasing difficulty:

ERM learns directly the map from $\mathcal{X}$ to $\mathcal{Y}$, without trying to estimate the conditional distribution of $Y|X = x$.
The plug-in approach aims at estimating the conditional distribution of $Y|X = x$ in order to derive the prediction rule.

Plug-in methods can themselves be separated between the conditional approach and the generative approach:

2a. The conditional plug-in method estimates directly the conditional distribution of Y given $X = x$
2b. The generative plug-in method first estimates the joint distribution of (X, Y), then derives an estimate of the conditional distribution of Y given $X = x$ via Bayes' theorem.

While the conditional approach leaves the marginal distribution of X unspecified, the generative approach aims at estimating the joint distribution, and hence tackles a more difficult problem, making more assumptions on the data generating process.

Generative vs discriminative

ERM and conditional plug-in approaches are commonly referred as discriminative learning, as one tries to directly learn the relation from X to Y.

The generative plug-in approach is known as generative learning, as one tries to learn the full joint distribution of (X, Y), hence the full generative model of the data.

Summary

Method	ERM	Conditional plug-in	Generative plug-in
Learns...	Map from X to Y	Conditional $Y\|X = x$	Joint dist. of (X, Y)
Type	Discriminative	Discriminative	Generative
Examples	Neural Nets	Logistic regression	LDA/QDA
	Boosting		Naive Bayes
	Nearest neighbours		
	Random Forests		

Empirical Risk minimisation

Empirical risk

$$\widehat{R}_n(h) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, h(x_i)).$$

Monte Carlo estimate of the true risk $R(h)$
Strong law of large numbers

$$\widehat{R}_n(h) \to R(h)$$

almost surely as $n \to \infty$.

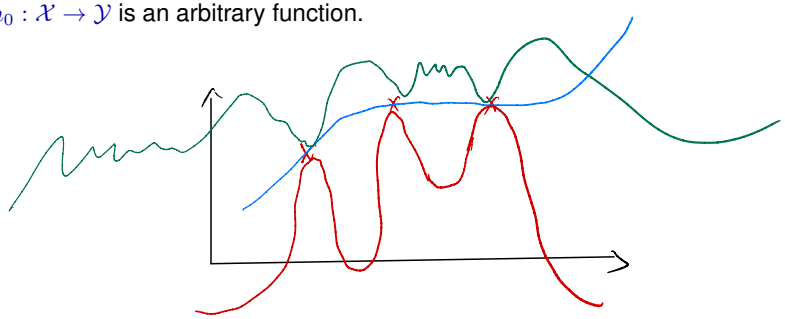
Empirical Risk minimisation

We cannot minimise the empirical risk over the whole set of prediction rules $\mathcal{F}$.

If all the x_i take different values, the minimal empirical risk $\widehat{R}_n(h) = 0$ is achieved for an infinite number of prediction rules:

$$\widehat{h}^{(d)}(x) = \begin{cases} y_i & \text{if } x \in \{x_1, \ldots, x_n\} \\ h_0(x) & \text{otherwise} \end{cases}$$

where $h_0 : \mathcal{X} \to \mathcal{Y}$ is an arbitrary function.



Empirical Risk minimisation

We therefore restrict the set of prediction functions to a specified function space $\mathcal{H} \subset \mathcal{F}$ (e.g. the set of linear classifiers).

Definition

For a function space $\mathcal{H}$, a loss function L, and a training set $d = (x_i, y_i)_{i=1,\ldots,n}$, the Empirical Risk Minimiser $\widehat{h}^{(d)}$ is defined as

$$\widehat{h}^{(d)} = \arg\min_{h \in \mathcal{H}} \widehat{R}_n(h)$$

where $\widehat{R}_n(h)$ is the empirical risk.

Empirical Risk minimisation

$h \in \mathcal{H}$ parameterised by a vector $\theta \in \Theta$, and we write h_θ.

For instance, if $\mathcal{H}$ is the class of linear functions on $\mathbb{R}$, then $h_\theta(x) = \beta_0 + \beta_1 x$ where $\theta = (\beta_0, \beta_1) \in \mathbb{R}^2$.

The ERM is therefore given by $\widehat{h}^{(d)} = h_{\widehat{\theta}}$ where

$$\widehat{\theta} = \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} L(y_i, h_\theta(x_i)).$$

Plug-in

Plug-in methods obtain some estimate $\widehat{F}_x$ of the conditional distribution of Y given $X = x$, and set

$$\widehat{h}^{(d)}(x) = T(\widehat{F}_x) = \arg\min_{y^* \in \mathcal{Y}} \int_{\mathcal{Y}} L(y, y^*) d\widehat{F}_x(y).$$

We will focus here on the case where the estimate is obtaining via maximum likelihood

Provides a connection between plug-in and ERM approaches.

Plug-in

Assume that the conditional probability mass/density function of Y given $X = x$ is $f_\theta(y|x)$ where the parametric form f_θ is known, but $\theta \in \Theta$ is some unknown parameter.
In the generative case, $f_\theta(y|x) \propto \pi_\theta(x, y)$, where π_θ is the known parametric form of the joint distribution
The maximum likelihood estimate of θ based on the training dataset d is

$$\widehat{\theta} = \arg\max_{\theta \in \Theta} \sum_{i=1}^{n} \log(f_\theta(y_i|x_i)) \quad \text{[Conditional]}$$

$$\widehat{\theta} = \arg\max_{\theta \in \Theta} \sum_{i=1}^{n} \log(\pi_\theta(x_i, y_i)) \quad \text{[Generative]}$$

The associated estimate of the pmf/pdf is therefore $f_{\widehat{\theta}}$, leading to the plug-in estimators

$$\widehat{h}^{(d)}(x) = \int_{\mathbb{R}} y f_{\widehat{\theta}}(y|x) dy \text{ for regression}$$

$$\widehat{h}^{(d)}(x) = \arg\max_{k=1,\ldots,K} f_{\widehat{\theta}}(k|x) \text{ for classification}$$

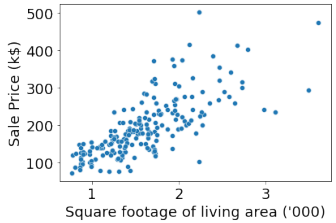
Example: Ordinary linear regression

Dataset consisting of house sale prices together with various attributes (square footage, number of rooms, etc)

Objective is to predict the sale price of an house based on its attributes.

We will only consider here predicting the sale price $y \in \mathbb{R}$ based on the square footage $x \in \mathbb{R}$ of the living area.

Training set of $n = 200$ observations (x_i, y_i), $i = 1, \ldots, n$.



Example: Ordinary linear regression

Empirical Risk minimisation.

Consider the set of linear prediction rules

$$\mathcal{H}_1 = \{h : \mathcal{X} \to \mathcal{Y} \mid h(x) = \beta_0 + \beta_1 x, \text{ with } \beta_0 \in \mathbb{R}, \beta_1 \in \mathbb{R}\}$$

For a prediction rule $h \in \mathcal{H}_1$, with $h(x) = \beta_0 + \beta_1 x$, the empirical risk under the squared error loss is given by

$$\widehat{R}_n(h) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x)^2.$$

Example: Ordinary linear regression

Empirical Risk minimisation.

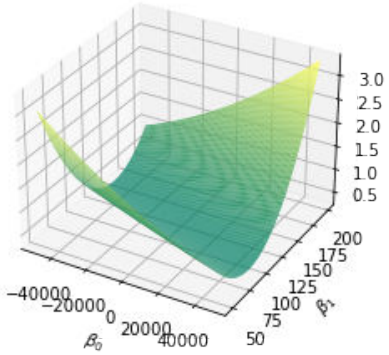


Figure: Empirical risk as a function of the model parameters

Example: Ordinary linear regression

Empirical Risk minimisation.

The ERM under the squared error loss, in the hypothesis set $\mathcal{H}_1$, is therefore given by

$$\widehat{h}^{(d)}(x) = \widehat{\beta}_0 + \widehat{\beta}_1 x$$

where

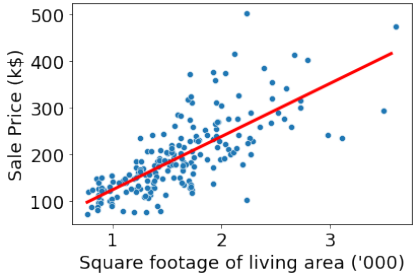
$$(\widehat{\beta}_0, \widehat{\beta}_1) = \underset{(\beta_0,\beta_1)\in\mathbb{R}^2}{\arg\min} \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x)^2$$

are the ordinary least squares estimates

$$\widehat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})y_i}{\sum_{i=1}^n (x_i - \bar{x})^2}, \quad \widehat{\beta}_0 = \bar{y} - \widehat{\beta}_1 \bar{x}.$$

Example: Ordinary linear regression

Empirical Risk minimisation.



Example: Ordinary linear regression

Plug-in approach

The prediction rule obtained via ERM may also be interpreted as a plug-in estimator, under additional assumptions.

Assume now that the conditional distribution of Y given $X = x$ is normal with mean $\beta_0 + \beta_1 x$ and variance σ^2, where σ^2 is assumed fixed.

Conditional pdf

$$f_\theta(y|x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\beta_0-\beta_1 x)^2}{2\sigma^2}}$$

where $\theta = (\beta_0, \beta_1) \in \mathbb{R}^2 \times \mathbb{R}_+$ are unknown parameters.

Example: Ordinary linear regression

Plug-in approach

Log-likelihood takes the form

$$\ell(\beta_0, \beta_1) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2.$$

Maximum likelihood estimates

$$(\widehat{\beta}_0, \widehat{\beta}_1) = \underset{(\beta_0,\beta_1)}{\arg\max} -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$$

$$= \underset{(\beta_0,\beta_1)}{\arg\min} \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2$$

Same as ordinary least square estimates.
Same learned prediction rule as ERM

$$\widehat{h}^{(d)}(x) = \int_{\mathbb{R}} y f_{(\widehat{\beta}_0,\widehat{\beta}_1)}(y|x)dy = \widehat{\beta}_0 + \widehat{\beta}_1 x$$

Summary

Quality of a given prediction is quantified by a loss function L

The risk $R(h)$ of a prediction rule h is the expected loss/error over the population, and the quantity we want to minimise

The Bayes prediction rule h^* is the optimal rule that minimises the population risk

Neither the risk nor the Bayes prediction rule can be calculated in practice

Empirical risk minimisation replaces the true risk by the empirical risk, and minimises it over a subset of prediction rules

Plug-in methods obtain some estimate of the conditional distribution Y given $X = x$ to derive a prediction rule

Statistical Machine Learning
Hilary Term 2021

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

Statistical Learning approaches

Empirical Risk Minimisation
Plug-in
- Conditional approach
- Generative approach

Statistical Learning approaches

Empirical Risk Minimisation
Plug-in
- Conditional approach
- Generative approach

The next two lectures

Generative Classifiers
- Linear Discriminant Analysis
- Quadratic Discriminant Analysis
- Naive Bayes

Generative Classifiers

Generative classifiers attempt to obtain some estimate of the joint distribution of the pair $(X, Y) \in \mathcal{X} \times \{1, \ldots, K\}$

Let

$\pi_k = \Pr(Y = k)$ be the class probability and
$g_k(x)$ be the conditional pdf or pmf of X, given $Y = k$.

Bayes' theorem

$$\Pr(Y = k | X = x) = \frac{\pi_k g_k(x)}{\sum_{\ell=1}^{K} \pi_\ell g_\ell(x)}.$$

Generative Classifiers

Under a $0 - 1$ loss, the optimal classifier is

$$\begin{aligned} h^*(x) &= \underset{k \in \{1,\ldots,K\}}{\arg\max} \Pr(Y = k | X = x) \\ &= \underset{k \in \{1,\ldots,K\}}{\arg\max} \pi_k g_k(x) \\ &= \underset{k \in \{1,\ldots,K\}}{\arg\max} \log \pi_k + \log g_k(x) \end{aligned}$$

Generative Classifiers

Definition (Generative classifier)

For $k = 1, \ldots, K$ and $x \in \mathcal{X}$, let $\widehat{\pi}_k$ and $\widehat{g}_k(x)$ be some estimates, using the training set d, of the class probabilities and of the conditional densities. The generative classifier is then given by

$$\widehat{h}^{(d)}(x) = \underset{k \in \{1,\ldots,K\}}{\arg\max} \log(\widehat{\pi}_k) + \log(\widehat{g}_k(x))$$

Generative Classifiers

How do we obtain these estimates?

For the class probabilities π_k, it is rather straightforward.

The class variables $Y_1, \ldots, Y_n$ are iid from a discrete distribution on $\{1, \ldots, K\}$ with probability mass function $(\pi_1, \ldots, \pi_K)$.

Use maximum likelihood

$$\ell(\pi_1, \ldots, \pi_K) = \sum_{k=1}^{K} m_k \log(\pi_k)$$

where $m_k = \#\{j : y_j = k\}$ is the number of observations in the class k.

Generative Classifiers

We aim at maximising

$$\ell(\pi_1, \ldots, \pi_K) = \sum_{k=1}^{K} m_k \log(\pi_k)$$

under the constraints $\pi_k \geq 0$ for all $k \geq 0$ and $\sum_{k'} \pi_{k'} = 1$.
Lagrangian

$$\mathcal{L}(\pi_1, \ldots, \pi_K, \gamma) = \sum_{k=1}^{K} m_k \log(\pi_k) - \gamma \left(\sum_{k=1}^{K} \pi_k - 1 \right).$$

Differentiating $\mathcal{L}$ with respect to π_k and setting to 0 gives

$$\pi_k = m_k / \gamma \text{ for all } k$$

Using $\sum_{k=1}^{K} \pi_k = 1$,

$$\widehat{\pi}_k = \frac{m_k}{n}.$$

Generative Classifiers

What about the conditional class densities $g_k(x)$?

Inputs may be of mixed type, potentially high-dimensional

Three different types of generative classifiers, corresponding to different assumptions

- Linear Discriminant Analysis
- Quadratic Discriminant Analysis
- Naive Bayes

Linear Discriminant Analysis

Let $\mathcal{X} = \mathbb{R}^p$

Linear discriminant analysis (LDA) assumes the class conditional densities $g_k(x)$ are normal pdfs, with a shared covariance matrix

$$g_k(x) = \varphi(x; \mu_k, \Sigma)$$

where μ_k, $k = 1, \ldots, p$ is a vector of size p, called the centroid of the class, and Σ is a $p \times p$ covariance matrix.

$(\mu_1, \ldots, \mu_K, \Sigma)$ are unknown

Σ is the within-class covariance of (X, Y)

Linear Discriminant Analysis

Parameters μ_k and Σ can be estimated via maximum likelihood
(x_i, y_i) are iid realisations of a random variable (X, Y) with
$\Pr(Y = k) = \pi_k$ and conditional Gaussian class densities $g_k(x)$
Log-likelihood

$$\begin{aligned}
&\ell(\pi_1, \dots, \pi_K, \mu_1, \dots, \mu_K, \Sigma) \\
&= \sum_{i=1}^{n} (\log(\pi_{y_i}) + \log \varphi(x_i; \mu_{y_i}, \Sigma)) \\
&= \sum_{k=1}^{K} \left(m_k \log(\pi_k) + \sum_{i|y_i=k} \log \varphi(x_i; \mu_k, \Sigma) \right) \\
&= \left(\sum_{k=1}^{K} m_k \log(\pi_k) \right) - \frac{1}{2} \sum_{k=1}^{K} \sum_{i|y_i=k} (x_i - \mu_k)^\intercal \Sigma^{-1} (x_i - \mu_k) \\
&\quad - \frac{n}{2} \log |\Sigma| + \text{const}
\end{aligned}$$

Linear Discriminant Analysis

Log-likelihood

$$\begin{aligned}
&\ell(\pi_1, \dots, \pi_K, \mu_1, \dots, \mu_K, \Sigma) \\
&= \left(\sum_{k=1}^{K} m_k \log(\pi_k) \right) - \frac{1}{2} \sum_{k=1}^{K} \sum_{i|y_i=k} (x_i - \mu_k)^\intercal \Sigma^{-1} (x_i - \mu_k) - \frac{n}{2} \log |\Sigma| + \text{const}
\end{aligned}$$

Maximising the log-likelihood under the constraint $\sum_{k=1}^{K} \pi_k = 1$ gives the maximum likelihood estimates

$$\begin{cases}
\widehat{\pi}_k = \dfrac{m_k}{n} \\
\widehat{\mu}_k = \dfrac{1}{m_k} \displaystyle\sum_{i:y_i=k} x_i \\
\widehat{\Sigma} = \dfrac{1}{n} \displaystyle\sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \widehat{\mu}_k)(x_i - \widehat{\mu}_k)^\intercal.
\end{cases}$$

Linear Discriminant Analysis

Proof

Already shown for π_k

For μ_k,

$$\frac{\partial \ell}{\partial \mu_k} = \sum_{i|y_i=k} \Sigma^{-1}(x_i - \mu_k)$$

Setting this to 0 yields

$$\widehat{\mu}_k = \frac{1}{m_k} \sum_{i:y_i=k} x_i.$$

Linear Discriminant Analysis

Proof

$$\begin{aligned}
&\ell(\pi_1, \dots, \pi_K, \mu_1, \dots, \mu_K, \Sigma) \\
&= \left(\sum_{k=1}^{K} m_k \log(\pi_k) \right) - \frac{1}{2} \sum_{k=1}^{K} \sum_{i|y_i=k} (x_i - \mu_k)^\intercal \Sigma^{-1} (x_i - \mu_k) - \frac{n}{2} \log |\Sigma| + \text{const}
\end{aligned}$$

Already shown for π_k

For Σ,

$$\frac{\partial \ell}{\partial \Sigma} = \frac{1}{2} \sum_{k=1}^{K} \sum_{i|y_i=k} \Sigma^{-1}(x_i - \mu_k)(x_i - \mu_k)^\intercal \Sigma^{-1} - \frac{n}{2} \Sigma^{-1}.$$

Setting this to 0,

$$\widehat{\Sigma} = \frac{1}{n} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \widehat{\mu}_k)(x_i - \widehat{\mu}_k)^\intercal$$

Linear Discriminant Analysis

Take $\widehat{\pi}_k \varphi(x; \widehat{\mu}_k, \widehat{\Sigma})$ as an approximation to $\pi_k g_k(x)$.

As the class densities are normal, this leads to linear discriminant functions and linear decision boundaries

$$\widehat{h}^{(d)}(x) = \underset{k=1,\dots,K}{\arg\max} \; \log(\widehat{\pi}_k) - \frac{1}{2} \underbrace{(x - \widehat{\mu}_k)^\intercal \widehat{\Sigma}^{-1} (x - \widehat{\mu}_k)}_{\text{squared Mahalanobis distance}}$$

$$= \underset{k=1,\dots,K}{\arg\max} \; \underbrace{\log(\widehat{\pi}_k) - \frac{1}{2} \widehat{\mu}_k^\intercal \widehat{\Sigma}^{-1} \widehat{\mu}_k}_{\widehat{a}_k} + \underbrace{\widehat{\mu}_k^\intercal \widehat{\Sigma}^{-1}}_{\widehat{b}_k^\intercal} x.$$

The scalar $\widehat{a}_k$ is the intercept of the linear prediction rule for class k, and the p-dimensional vector $\widehat{b}_k$ its coefficients (called slope if $p = 1$).

Computations for LDA

Assume $\widehat{\Sigma}$ has full rank

Let $\widehat{\Sigma} = V \Lambda V^\intercal$ be the eigenvalue decomposition of the matrix $\widehat{\Sigma}$, where V is a $p \times p$ orthogonal matrix and Λ is a diagonal $p \times p$ matrix.

Note that $\widehat{\Sigma}^{-1} = V \Lambda^{-1} V^\intercal$ and $\widehat{\Sigma}^{-1/2} = \Lambda^{-1/2} V^\intercal$.

We have

$$\begin{aligned}
&(x - \widehat{\mu}_k)^\intercal \widehat{\Sigma}^{-1} (x - \widehat{\mu}_k) \\
&= (\Lambda^{-1/2} V^\intercal x - \Lambda^{-1/2} V^\intercal \widehat{\mu}_k)^\intercal (\Lambda^{-1/2} V^\intercal x - \Lambda^{-1/2} V^\intercal \widehat{\mu}_k) \\
&= ||x^\bullet - \widehat{\mu}_k^\bullet||^2
\end{aligned}$$

that is the squared Euclidian distance between the transformed vectors $x^\bullet = \Lambda^{-1/2} V^\intercal x$ and $\widehat{\mu}_k^\bullet = \Lambda^{-1/2} V^\intercal \widehat{\mu}_k$.

Computations for LDA

Training

- Compute the MLE $\widehat{\pi}_k$, $\widehat{\mu}_k$ and $\widehat{\Sigma}$.
- Compute the eigendecomposition V, Λ of $\widehat{\Sigma}$
- For $k = 1, \dots, K$, set $\widehat{\mu}_k^\bullet = \Lambda^{-1/2} V^\intercal \widehat{\mu}_k$.

Prediction. For an input vector x

- Compute $x^\bullet = \Lambda^{-1/2} V^\intercal x$
- Classify to the closest class mean $\widehat{\mu}_k^\bullet$, modulo the effect of the estimated class proportions $\widehat{\pi}_k$

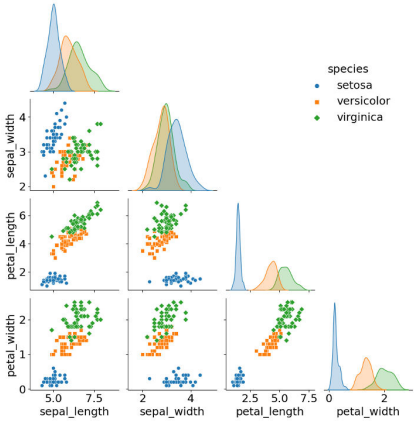
$$\widehat{h}^{(d)}(x) = \underset{k=1,\dots,K}{\arg\min} \; ||x^\bullet - \widehat{\mu}_k^\bullet||^2 - 2 \log(\widehat{\pi}_k)$$

.

Example: Iris dataset

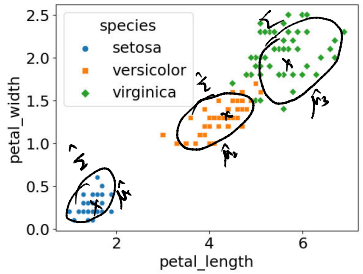
$p = 4$ measurements (sepal length, sepal width, petal length, petal width) for $n = 50$ Iris plants.

Each plant is from one of $K = 3$ species: Setosa, Versicolour and Virginica.

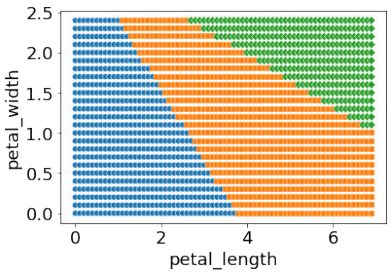


Example: Iris dataset

For visualisation purposes, we will first consider two dimensions, the petal width and length.



Training dataset



LDA prediction rule

LDA for dimensionality reduction

Let $\widehat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i$ be the sample mean and

$$\widehat{B} = \sum_{k=1}^{K} \widehat{\pi}_k (\widehat{\mu}_k - \widehat{\mu})(\widehat{\mu}_k - \widehat{\mu})^\intercal$$

be an estimate of the between class covariance.
Define

$$\widehat{B}^\bullet = \sum_{k=1}^{K} \widehat{\pi}_k (\widehat{\mu}_k^\bullet - \widehat{\mu}^\bullet)(\widehat{\mu}_k^\bullet - \widehat{\mu}^\bullet)^\intercal$$

where $\widehat{\mu}^\bullet = \sum_{k=1}^{K} \pi_k \mu_k^\bullet$.
Denote

$$\widehat{B}^\bullet = U^\bullet D^\bullet U^{\bullet\intercal}$$

the eigendecomposition of $\widehat{B}^\bullet$, and $u_\ell^\bullet$ the columns of $U^\bullet$.
The $\ell's$ discriminant coordinate of a vector x is defined as

$$\boxed{z_\ell = (u_\ell^\bullet)^\intercal x^\bullet = (u_\ell^\bullet)^\intercal \Lambda^{-1/2} V^\intercal x.}$$

LDA for dimensionality reduction

As mentioned when discussing the optimal Bayes classifier under Gaussian class distributions, the prediction $\widehat{h}^{(d)}(x)$ only depends on x through its projections onto the first $K-1$ discriminant coordinates.
The projections maximise the separation between the classes.
In particular, $a_1 = V\Lambda^{-1/2}u_1^\bullet$ minimises

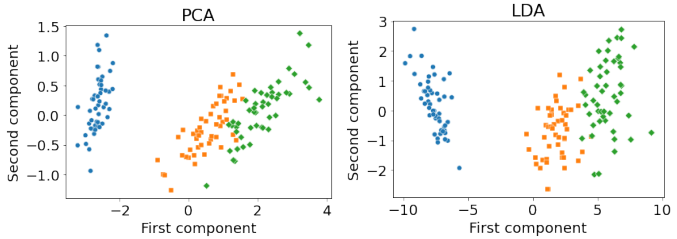
$$\frac{a_1^\intercal \widehat{B} a_1}{a_1^\intercal \widehat{\Sigma} a_1}$$

and $a_1^\intercal x$ is the one-dimensional projection of the data maximising the separation between classes.

LDA vs PCA

Example: Iris dataset (4D)

Consider now the full Iris dataset ($p = 4$)
$K = 3$ classes, so $K - 1 = 2$ discriminant coordinates



LDA and SVD

Let $\widetilde{\mathbf{X}}$ be the n-by-p matrix with rows $(x_i - \widehat{\mu}_{y_i})^\intercal$

Note that $\widehat{\Sigma} = \frac{1}{n}\widetilde{\mathbf{X}}^\intercal\widetilde{\mathbf{X}}$.

Similarly to the PCA case, one can use the singular value decomposition of $\widetilde{\mathbf{X}}$ instead of the eigendecomposition of $\widehat{\Sigma}$ to compute the different quantities of interest

More efficient when p is large

In the case $p > n$ the matrix $\widehat{\Sigma}$ is not invertible anymore; we can still compute the discriminant coordinates using the SVD decomposition

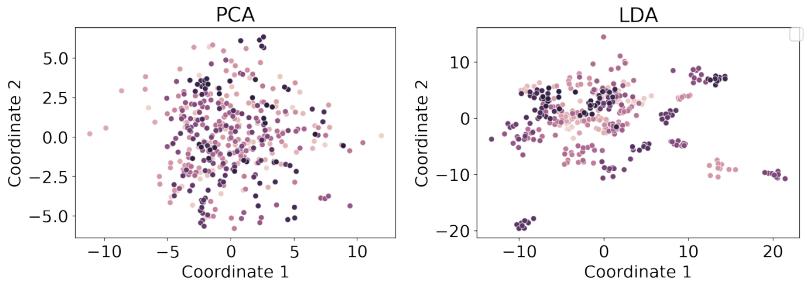
Olivetti face dataset

Image Dataset of $n = 400$ images from $K = 40$ different persons

Each image is 64×64 pixel in grey levels, hence $p = 4096$

Note that $p > n$

Olivetti face dataset



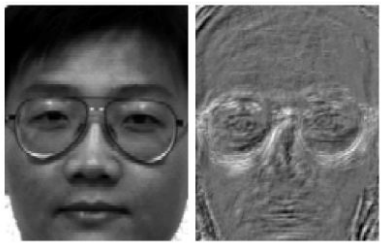
Eigenfaces vs Fisherfaces

Yale face dataset

Two classes: with or without glasses

First PC v_1 given by PCA and first discriminant vector a_1 given by LDA

Called Eigenface and Fisherface in this context



P.N. Belhumeur, J.P. Hespanha, D.J. Kriegman. Eigenfaces vs. Fisherfaces: recognition using class specific linear projection. IEEE PAMI, 1997.

Quadratic Discriminant Analysis

Quadratic discriminant analysis (QDA) assumes the class conditional densities $g_k(x)$ are normal pdfs, with unconstrained covariance matrices

$$g_k(x) = \varphi(x; \mu_k, \Sigma_k)$$

where μ_k, $k = 1, \ldots, p$ is a vector of size p, called the centroid of the class, and Σ_k is a $p \times p$ covariance matrix of the class k.

Quadratic Discriminant Analysis

The parameters μ_k and Σ_k can be estimated via maximum likelihood, as for the class frequencies π_k.
Log-likelihood

$$\begin{aligned}
&\ell(\pi_1, \ldots, \pi_K, \mu_1, \ldots, \mu_K, \Sigma_1, \ldots, \Sigma_K) \\
&= \sum_{i=1}^{n} (\log(\pi_{y_i}) + \log \varphi(x_i; \mu_{y_i}, \Sigma_{y_i})) \\
&= \sum_{k=1}^{K} \left(m_k \log(\pi_k) + \sum_{i|y_i=k} \log \varphi(x_i; \mu_k, \Sigma_k) \right) \\
&= \left(\sum_{k=1}^{K} m_k \log(\pi_k) \right) - \frac{1}{2} \sum_{k=1}^{K} \sum_{i|y_i=k} (x_i - \mu_k)^\intercal \Sigma_k^{-1} (x_i - \mu_k) - \sum_{k=1}^{K} \frac{m_k}{2} \log |\Sigma_k| + \text{const}
\end{aligned}$$

Quadratic Discriminant Analysis

MLE

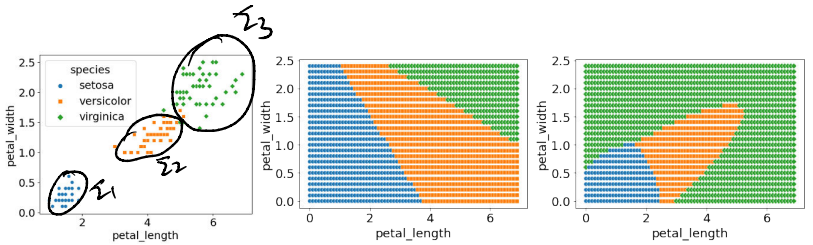
$$\begin{cases}
\widehat{\pi}_k = \dfrac{m_k}{n} \\
\widehat{\mu}_k = \dfrac{1}{m_k} \displaystyle\sum_{i:y_i=k} x_i \\
\widehat{\Sigma}_k = \dfrac{1}{m_k} \displaystyle\sum_{i:y_i=k} (x_i - \widehat{\mu}_k)(x_i - \widehat{\mu}_k)^\intercal.
\end{cases}$$

Quadratic Discriminant Analysis

Take $\widehat{\pi}_k \varphi(x; \widehat{\mu}_k, \widehat{\Sigma}_k)$ as an approximation to $\pi_k g_k(x)$.
Quadratic discriminant functions and quadratic decision boundaries

$$\begin{aligned}
\widehat{h}^{(d)}(x) &= \underset{k=1,\ldots,K}{\arg\max} \; \log(\widehat{\pi}_k) - \frac{1}{2} \underbrace{(x - \widehat{\mu}_k)^\intercal \widehat{\Sigma}_k^{-1} (x - \widehat{\mu}_k)}_{\text{squared Mahalanobis distance}} \\
&= \underset{k=1,\ldots,K}{\arg\max} \; \log(\widehat{\pi}_k) - \frac{1}{2} \widehat{\mu}_k^\intercal \widehat{\Sigma}_k^{-1} \widehat{\mu}_k + \widehat{\mu}_k^\intercal \widehat{\Sigma}^{-1} x - \frac{1}{2} x^\intercal \Sigma_k^{-1} x.
\end{aligned}$$

Example: Iris dataset (2D)



Training dataset LDA prediction rule QDA prediction rule

Quadratic Discriminant Analysis

QDA allows for more flexibility and can capture more complex distributions as it allows for different covariance matrices.

There is however a price to pay in terms of increased variance and potential overfitting.

The number of parameters to estimate is

- $(K-1)+Kp+p(p+1)/2$ for LDA,
- $(K-1)+Kp+Kp(p+1)/2$ for QDA.

Scales quadratically with p.

When p is large, this may lead to overfitting, in particular for QDA if some classes have few examples.

Regularised discriminant analysis

Regularised discriminant analysis combines LDA and QDA by considering the following estimate for the covariance within class k

$$\widehat{\Sigma}_k(\alpha) = \alpha\widehat{\Sigma}_k + (1-\alpha)\widehat{\Sigma}$$

where $\widehat{\Sigma}$ is the MLE for LDA, and $\widehat{\Sigma}_k$ is the MLE for QDA, for some $\alpha \in [0,1]$.

This introduces a new parameter α and allows for a continuum of models between LDA and QDA to be used.

The parameter α can be chosen by cross-validation for example.

LDA/QDA with shrinkage

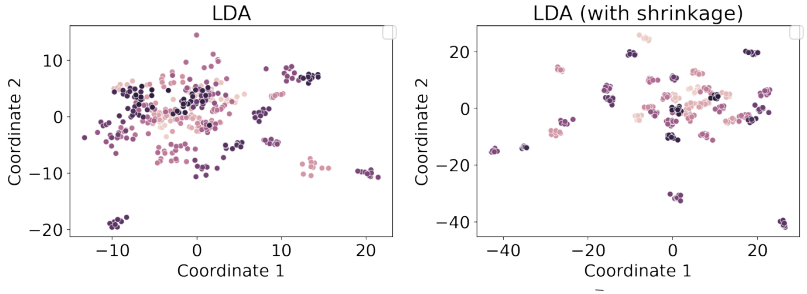
An alternative approach is to consider the shrinkage estimate (here for LDA)

$$\widehat{\Sigma}(\delta) = \delta I_p + (1-\delta)\widehat{\Sigma}$$

where $\delta \in (0,1)$ is a regularising parameter.

Note that $\widehat{\Sigma}(\delta)$ is always invertible, even if $\widehat{\Sigma}$ is not.

Olivetti face dataset



Naive Bayes classifier

5/2

The naive Bayes classifier is another generative plug-in classifier.

!!! Not to be confused with the Bayes classifer, which is the optimal classifier under the 0-1 loss !!!

It makes the assumption that the different univariate components of the p-dimensional input variable X are conditionally independent given the class Y.

Naive Bayes can easily handle inputs x of mixed type (categorical, binary, continuous), as well as missing data.

Naive Bayes classifier

For illustration consider this toy example

Imagine we want to predict the voter preferences for the US election based on some attributes

Voted in 2016?	Annual Income	State	Candidate Choice
Y	50K	OK	Biden
N	173K	CA	Biden
Y	80K	NJ	Trump
Y	150K	WA	Biden
Y	85K	IL	Trump
⋮	⋮	⋮	⋮
Y	1050K	NY	Biden
N	35K	CA	Trump
N	100K	NY	?

The input vector x is 3-dimensional and contains a mix of binary (Voted in 2016?), real (Annual Income) and categorical (State) variables.

The response variable y is binary (Biden/Trump).

Naive Bayes classifier

The naive Bayes classifier assumes that the different inputs coordinates are independent conditional on the class labels.

That is, for an input vector $x = (x_1, \ldots, x_p)^\intercal$, the conditional class distributions, for $k = 1, \ldots, K$, factorise as

x_j = j^{th} coordinate of vector x

$$g_k(x) = \prod_{j=1}^{p} g_{kj}(x_j; \theta_{kj})$$

where g_{kj} are conditional pmf/pdf on $\mathbb{R}$ of $X_j \mid Y = k$, parameterised by θ_{kj}.

Clearly, the independence assumption is “naive" and rarely satisfied.

Allows to significantly reduce the number of parameters to estimate and make inference much easier.

Although the generative model is quite simple, naive Bayes often works quite well in practice for predicting the class labels.

Naive Bayes classifier

In our voting example, we need to estimate
$\Pr(\text{person } x \text{ voted in 2016} \mid \text{Biden supporter})$, the probability that someone voted in 2016, given it is a Biden supporter.

Similarly, we have to estimate
$\Pr(\text{person } x \text{ voted in 2016} \mid \text{Trump supporter})$.

Same for the other variables “Annual income" and “State", for each candidate, resulting in the estimation of the parameters of 6 conditional univariate pmf or pdf.

Naive Bayes classifier

We need to define the parametric models $g_{kj}(x_j;\theta_{kj})$. There are standard choices depending on the type of variable.

Real-Valued. For real-valued variables, such as the annual income, a standard choice is to use a Gaussian model, with unknown mean μ_{kj} and variance σ^2_{kj}, that is

$$g_{kj}(x_j;\theta_{kj}) = \varphi(x_j;\mu_{kj},\sigma^2_{kj})$$

where $\theta_{kj} = (\mu_{kj},\sigma^2_{kj})$.

Binary If $x_j \in \{0,1\}$ (such as “Voted in 2016?"), we use a Bernoulli distribution with parameter $\theta_{kj} \in [0,1]$

$$g_{kj}(1;\theta_{kj}) = 1 - g_{kj}(0;\theta_{kj}) = \theta_{kj}.$$

Categorical If $x_j \in \{1,\dots,C\}$ is a categorical feature (such as the state), we can use the multinomial distribution, with parameters $\theta_{kj,1},\dots,\theta_{kj,C}$ where $\sum_{c=1}^{C}\theta_{kj,c} = 1$. For any $c = 1,\dots,C$

$$g_{kj}(c;\theta_{kj}) = \theta_{kj,c}$$

Naive Bayes classifier

Parameter estimation

We can estimate the parameters using maximum likelihood
Log-likelihood

$$\begin{aligned}
&\ell((\pi_k)_{k=1\dots,K},(\theta_{kj})_{k=1,\dots,K;j=1,\dots,p}) \\
&= \sum_{i=1}^{n}(\log(\pi_{y_i}) + \log g_{y_i}(x_i)) \\
&= \sum_{k=1}^{K} m_k \log(\pi_k) + \sum_{k=1}^{K}\sum_{i|y_i=k}\log g_k(x_i) \\
&= \sum_{k=1}^{K} m_k \log(\pi_k) + \sum_{k=1}^{K}\sum_{i|y_i=k}\sum_{j=1}^{p}\log g_{kj}(x_{ij};\theta_{kj}) \\
&= \underbrace{\sum_{k=1}^{K} m_k \log(\pi_k)}_{\ell_1((\pi_k))} + \sum_{j=1}^{p}\sum_{k=1}^{K}\underbrace{\sum_{i|y_i=k}\log g_{kj}(x_{ij};\theta_{kj})}_{\ell_{kj}(\theta_{kj})}
\end{aligned}$$

Naive Bayes classifier

Parameter estimation

The log-likelihood therefore factorises as $\ell_1((\pi_k)) + \sum_{j,k}\ell_{kj}(\theta_{kj})$, and each term of the sum can be optimised separately.

For π_k, the MLE is given as before:

$$\widehat{\pi}_k = \frac{m_k}{n}.$$

For the other parameters, the MLE is given by

$$\widehat{\theta}_{kj} = \arg\max_{\theta_{kj}} \sum_{i|y_i=k}\log g_{kj}(x_{ij};\theta_{kj})$$

and depends on the choice of the model.

Naive Bayes classifier

Parameter estimation

In particular, we have

Gaussian likelihood: $\widehat{\theta}_{kj} = (\widehat{\mu}_{kj}, \widehat{\sigma}^2_{kj})$ where

$$\widehat{\mu}_{kj} = \frac{1}{m_k} \sum_{i|y_i=k} x_{ij}, \quad \widehat{\sigma}^2_{kj} = \frac{1}{m_k} \sum_{i|y_i=k} (x_{ij} - \widehat{\mu}_{kj})^2$$

Bernoulli likelihood:

$$\widehat{\theta}_{kj} = \frac{\sum_{i|y_i=k} x_{ij}}{m_k}$$

Multinomial likelihood: For each $c = 1, \ldots, C$,

$$\widehat{\theta}_{kj,c} = \frac{\sum_{i|y_i=k} \mathbb{1}_{x_{ij}=c}}{m_k}$$

Naive Bayes classifier

Parameter estimation

Once the parameters are estimated, the conditional distribution $\Pr(Y = k \mid X = x)$ is approximated by

$$\frac{\widehat{\pi}_k \prod_{j=1}^p g_{kj}(x; \widehat{\theta}_{kj})}{\sum_{k'=1}^K \widehat{\pi}_{k'} \prod_{j=1}^p g_{k'j}(x; \widehat{\theta}_{k'j})}$$

and the naive Bayes classifier is

$$\widehat{h}^{(d)}(x) = \underset{k \in \{1,\ldots,K\}}{\arg\max} \log(\widehat{\pi}_k) + \sum_{j=1}^p \log(g_{kj}(x; \widehat{\theta}_{kj})).$$

Missing data

Going back to our example, consider that the voter did not reveal if they had voted in 2016:

Voted in 2016?	Annual Income	State	Candidate Choice
Y	50K	OK	Biden
N	173K	CA	Biden
Y	80K	NJ	Trump
Y	150K	WA	Biden
Y	85K	IL	Trump
⋮	⋮	⋮	⋮
Y	1050K	NY	Biden
N	35K	CA	Trump
?	100K	NY	?

For our new observation $\widetilde{x} = (\widetilde{x}_1, \widetilde{x}_2, \widetilde{x}_3)$, the value of the first variable $\widetilde{x}_1$ is missing, and we can thus only use $(\widetilde{x}_2, \widetilde{x}_3)$ to predict its candidate choice.

Missing data

Under the naive Bayes assumption,

$$g_k((\widetilde{x}_1, \widetilde{x}_2, \widetilde{x}_3)) = \prod_{j=1}^3 g_{kj}(\widetilde{x}_j; \theta_{kj})$$

Marginalising the above expression with respect to $\widetilde{x}_1$ gives the conditional distribution of $(\widetilde{X}_2, \widetilde{X}_3)$ given $\widetilde{Y} = k$, which is

$$\sum_{\widetilde{x}_1=0,1} g_k((\widetilde{x}_1, \widetilde{x}_2, \widetilde{x}_3)) = \prod_{j=2}^3 g_{kj}(\widetilde{x}_j; \theta_{kj}).$$

Hence, using Bayes rule,

$$\Pr(\widetilde{Y} = k \mid (\widetilde{X}_2, \widetilde{X}_3) = (\widetilde{x}_2, \widetilde{x}_3)) \propto \pi_k \prod_{j=2}^3 g_{kj}(\widetilde{x}_j; \theta_{kj}).$$

The naive Bayes prediction is therefore given by

$$\widehat{h}^{(d)}((\widetilde{x}_2, \widetilde{x}_3)) = \underset{k \in \{1,\ldots,K\}}{\arg\max} \log(\widehat{\pi}_k) + \sum_{j=2}^p \log(g_{kj}(x; \widehat{\theta}_{kj})).$$

Missing data

Note that missing data can be handled in a similar way for any generative approach, by marginalising the variable of interest in $g_k(x)$.

This is done without extra computation for naive Bayes due to the independence assumption.

Without the independence this would require summation/integration.

Missing data

Dealing with missing data in the training set is also quite easy.

Assume that some entries are missing for the variable “Voted in 2016":

Voted in 2016?	Annual Income	State	Candidate Choice
?	50K	OK	Biden
?	173K	CA	Biden
Y	80K	NJ	Trump
Y	150K	WA	Biden
?	85K	IL	Trump
⋮	⋮	⋮	⋮
Y	1050K	NY	Biden
N	35K	CA	Trump
?	100K	NY	?

For example, let’s say that for Biden voters, 103 had voted in 2016, 54 had not, and 25 didn’t answer.

We can simply estimate $\theta_{Biden,1}$ based on the non-missing answers, that is $\widehat{\theta}_{Biden,1} = 103/157$.

Naive Bayes and Curse of dimensionality

By making the independence assumption, naive Bayes breaks the curse of dimensionality and avoids overfitting

Assume for illustration that all the features x_j of the vector x are binary, hence $x \in \{0,1\}^p$, where $\{0,1\}^p$ has 2^p elements

If one were to parameterise $g_k(x)$, this would require $O(2^p)$ parameters

By contrast, using naive Bayes only requires $O(p)$ parameters (one for each dimension of the input vector x)

Example: Titanic data

Dataset consisting of attributes of 887 passengers of the Titanic, and whether or not they survived the tragedy.

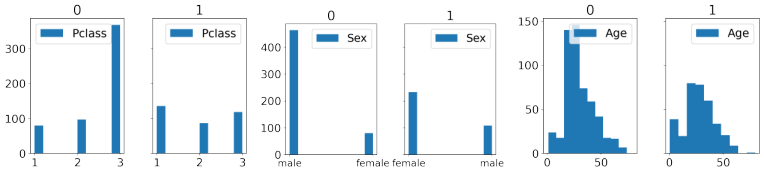
Class	Sex	Age	Survived
3	male	22	N
1	female	38	Y
3	female	26	Y
1	female	35	Y
3	male	35	N
⋮	⋮	⋮	⋮

Response variable y is binary (survived=1, did not survived=0)

Input vector x is three dimensional $x = (x_1, x_2, x_3)^\intercal$ where x_1 is the class of the passenger (1st,2nd or 3rd), x_2 is the sex (female/male), x_3 is the age.

Example: Titanic data

We aim at deriving a classifier to predict the survival based on the three attributes.



(a) Passenger class (b) Sex (c) Age

Figure: Histogram of the (a) Passenger class (b) Sex and (c) Age of the passenger amongst those who did not survive (left plot) and did survive (right plot).

Example: Titanic data

We use a multinomial model with $C = 3$ for g_{k1} (passenger class), a Bernoulli model for g_{k2} (sex) and a Gaussian model for g_{k3} (age).

Obviously the Gaussian distribution gives a poor fit to the data, and one could consider alternative models!

Classifier:

$$\widehat{h}^{(d)}(x) = \underset{k \in \{0,1\}}{\arg\max} \log(\widehat{\pi}_k) + \sum_{j=1}^{3} \log(g_{kj}(x; \widehat{\theta}_{kj})).$$

Estimated Parameters:

Name	Parameters	Did not Survived ($k = 0$)	Survived ($k = 1$)
Prior class proba	π_k	0.61	0.39
Pass. Class	$(\widehat{\theta}_{k1,1}, \ldots, \widehat{\theta}_{k1,3})$	$(0.15, 0.18, 0.67)$	$(0.40, 0.25, 0.35)$
Sex (F=1)	$\widehat{\theta}_{k2}$	0.15	0.32
Age	$(\widehat{\mu}_{k3}, \widehat{\sigma}_{k3})$	$(30.1, 13.9)$	$(28.4, 14.4)$

Example: Titanic data

Some predictions from Naive Bayes classifier

Class	Sex	Age	Predicted Survival	Estimated Probability of survival
3	female	30	Y	0.587
1	female	30	Y	0.882
3	male	20	N	0.114
1	male	20	N	0.403

Summary

We have seen three different classes of generative classifiers

- LDA and QDA assume that the input X is normally distributed given the class $Y = k$, with the same covariance (LDA) or different covariances (QDA)
- Naive Bayes assumes that the different dimensions of the input X are conditionally independent given the class $Y = k$

Advantages of the generative approaches are

- Assumptions made on the data generating process can be statistically tested (Gaussianity, independence)
- Even if the assumptions do not hold, the classifier may still provide good predictions in practice
- Can easily handle missing data in the input vector
- Interpretable predictions

Main drawback is that it makes stronger assumptions on the data generating process, that are rarely met in practice, and limit the flexibility of the approach

Statistical Machine Learning
Hilary Term 2021

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

16/2

The next two lectures

Further concepts/tools in statistical machine learning
- Nonlinear input transformation/expansion,
- Overfitting and bias-variance tradeoff
- Regularisation
- Cross-validation

Simple regression example for illustration, with empirical risk minimisation

Concepts/tools apply equally to classification tasks, and plug-in methods

Multivariate linear regression

Let $((x_1, y_1), \ldots, (x_n, y_n))$ be the input/target data
- $x_i \in \mathbb{R}^p$
- $y_i \in \mathbb{R}$

For an input variable $x \in \mathbb{R}^p$, let $\widetilde{x} = \begin{pmatrix} 1 \\ x \end{pmatrix} \in \mathbb{R}^{p+1}$.

Class of linear prediction rules

$$\mathcal{H} = \{h(x) = \widetilde{x}^\intercal \beta = (1 \;\; x^\intercal)\beta \mid \beta \in \mathbb{R}^{p+1}\}.$$

Multivariate linear regression

Empirical risk minimiser under a squared error loss for the class $\mathcal{H}$ is

$$\widehat{h}^{(d)}(x) = \widetilde{x}^\intercal \widehat{\beta}$$

where

$$\widehat{\beta} = \underset{\beta \in \mathbb{R}^{p+1}}{\arg\min} \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2$$

Multivariate linear regression

Matrix form

$$\begin{aligned}\widehat{\beta} &= \arg\min_{\beta\in\mathbb{R}^{p+1}} \sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \\ &= \arg\min_{\beta\in\mathbb{R}^{p+1}} \sum_{i=1}^{n}(y_i - \widetilde{x}_i^\intercal\beta)^2 \\ &= \arg\min_{\beta\in\mathbb{R}^{p+1}} \|\,\mathbf{y} - \widetilde{\mathbf{X}}\beta\|^2\end{aligned}$$

where $\mathbf{y} = (y_1, \ldots, y_n)^\intercal \in \mathbb{R}^n$ and $\widetilde{\mathbf{X}}$ is the n-by-$(p+1)$ design matrix defined by

$$\widetilde{\mathbf{X}} = \begin{pmatrix} 1 & x_{11} & \ldots & x_{1p} \\ 1 & x_{21} & \ldots & x_{2p} \\ \ldots & \ldots & \ldots & \ldots \\ 1 & x_{n1} & \ldots & x_{np} \end{pmatrix} = \begin{pmatrix} | & \widetilde{x}_1^\intercal \\ & | \\ | & \widetilde{x}_n^\intercal \end{pmatrix} = \begin{pmatrix} \widetilde{x}_1^\intercal \\ | \\ \widetilde{x}_n^\intercal \end{pmatrix}$$

Multivariate linear regression

We have

$$\begin{aligned}\|\,\mathbf{y} - \widetilde{\mathbf{X}}\beta\|^2 &= (\mathbf{y} - \widetilde{\mathbf{X}}\beta)^\intercal(\mathbf{y} - \widetilde{\mathbf{X}}\beta) \\ &= \beta^\intercal\widetilde{\mathbf{X}}^\intercal\widetilde{\mathbf{X}}\beta - 2(\widetilde{\mathbf{X}}^\intercal\mathbf{y})^\intercal\beta + \text{const.}\end{aligned}$$

Vector derivatives: for $x, a \in \mathbb{R}^p$, B p-by-p symmetric matrix

Recall: $\quad \frac{\partial(a^\intercal x)}{\partial x} = a, \quad \frac{\partial(x^\intercal Bx)}{\partial x} = 2Bx$

We obtain

$$\frac{\partial(\|\,\mathbf{y} - \widetilde{\mathbf{X}}\beta\|^2)}{\partial\beta} = 2\widetilde{\mathbf{X}}^\intercal\widetilde{\mathbf{X}}\beta - 2\widetilde{\mathbf{X}}^\intercal\mathbf{y}$$

Setting this to 0, and assuming that $\widetilde{\mathbf{X}}^\intercal\widetilde{\mathbf{X}}$ has full rank, we obtain the estimate

$$\widehat{\beta} = (\widetilde{\mathbf{X}}^\intercal\widetilde{\mathbf{X}})^{-1}\widetilde{\mathbf{X}}^\intercal\mathbf{y}\,.$$

Nonlinear basis expansion

Learned prediction rule $\widehat{h}^{(d)}(x) = \widetilde{x}^\intercal\widehat{\beta}$ obtained in closed-form

Can only capture linear relations between the input x and response y

Possible to obtain non-linear prediction rules by using a class of linear prediction rules on an extended input/feature space

Nonlinear basis expansion

Input space $\mathcal{X}$

Function $\phi : \mathcal{X} \to \mathcal{X}'$, where $\mathcal{X}'$ is the extended input (or feature) space

ϕ is a known, typically nonlinear function

Extended input space $\mathcal{X}'$ may be of

- Lower dimension (dimensionality reduction, e.g. PCA encoder)
- Same dimension
- Higher dimension (input expansion)

By using nonlinear mappings and/or by expanding the dimension of the input we can allow for more complex prediction rules

Can apply the learning method on the transformed inputs $\phi(x_1), \ldots, \phi(x_n)$ instead of the original inputs $x_1, \ldots, x_n$

Capture complex prediction rules using simple methods

Nonlinear basis expansion

Polynomial expansion

$x \in \mathbb{R}$, $\phi(x) = (1 \;\; x \;\; x^2 \;\; x^3)^\intercal$

With interactions

$x = (x_1, x_2, x_3)^\intercal \in \mathbb{R}^3$

$\phi(x) = (1 \;\; x_1 \;\; x_2 \;\; x_3 \;\; x_1x_2 \;\; x_1x_3 \;\; x_2x_3 \;\; x_1x_2x_3 \;\; x_1^2x_2 \;\; x_1^2x_3 \ldots)^\intercal$

Nonlinear basis expansion

Let $\phi : \mathcal{X} \to \mathbb{R}^{M+1}$ for some $M \geq 0$.

Class of linear prediction rules on the transformed inputs

$$\mathcal{H} = \{h(x) = \phi(x)^\intercal \beta \text{ where } \beta \in \mathbb{R}^{M+1}\}.$$

Linear functions in the transformed input space $\mathcal{X}' = \mathbb{R}^{M+1}$, but nonlinear function in the original input space $\mathcal{X}$ if ϕ is nonlinear.

ERM under a squared loss can be found in closed form

$$\widehat{\beta} = \underset{\beta \in \mathbb{R}^{M+1}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} (y_i - \phi(x_i)^\intercal \beta)^2 = (\mathbf{\Phi}^\intercal \mathbf{\Phi})^{-1} \mathbf{\Phi}^\intercal \mathbf{y}$$

where

$$\mathbf{\Phi} = (\phi(x_1), \ldots, \phi(x_n))^\intercal \in \mathbb{R}^{n \times (M+1)}, \quad \mathbf{y} = (y_1, \ldots, y_n)^\intercal \in \mathbb{R}^n .$$

Nonlinear basis expansion

Sinusoid example

Illustrative example

(X, Y) has a joint distribution

$$Y = \sin(2\pi X) + \epsilon, \quad X \sim \mathsf{U}(0,1), \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

where X is independent of ϵ, and $\sigma > 0$

Bayes prediction rule

$$h^*(x) = \mathbb{E}[Y|X = x] = \sin(2\pi x).$$

Risk for a prediction rule h

$$R(h) = \mathbb{E}[(Y - h(X))^2] = \mathbb{E}[(\sin(2\pi X) + \epsilon - h(X))^2]$$
$$= \sigma^2 + \int_0^1 (\sin(2\pi x) - h(x))^2 dx$$

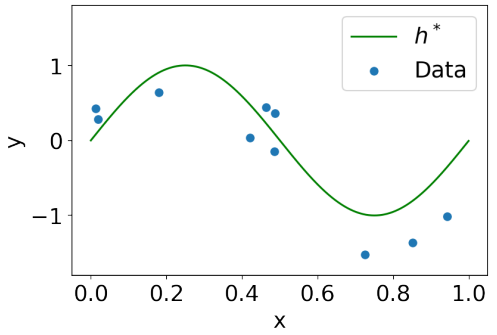
Bayes risk is $R(h^*) = \sigma^2$.

Excess risk of h: $R(h) - R(h^*)$

Nonlinear basis expansion

Sinusoid example

$n = 10$ realisations (x_i, y_i), $i = 1, \ldots, n$



Nonlinear basis expansion

Sinusoid example

For $M \geq 0$, let $\mathcal{H}_M$ be the set of polynomials of order M

$$\mathcal{H}_M = \{h : \mathbb{R} \to \mathbb{R} \mid h(x) = \sum_{j=0}^{M} \beta_j x^j, \beta_j \in \mathbb{R} \text{ for } j = 0 \ldots, M\}$$

Nonlinear input expansion $\phi(x) = (1 \;\; x \;\; x^2 \;\; \ldots \;\; x^M)^\intercal$

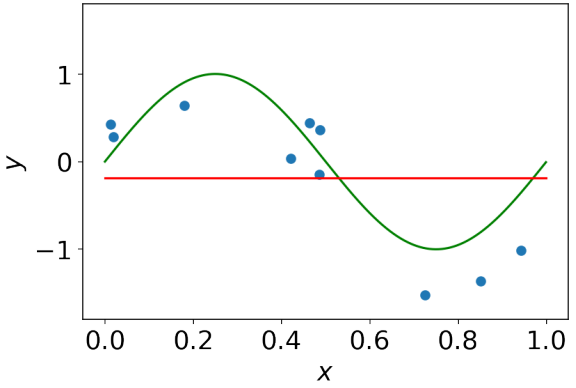
Hypothesis classes are nested $\mathcal{H}_0 \subset \mathcal{H}_1 \subset \ldots$

Bayes prediction rule $h^* \notin \mathcal{H}_M$, for all M

Nonlinear basis expansion

Sinusoid example

Find the Empirical risk minimiser over each class $\mathcal{H}_M$, for $M = 0, \ldots, 9$

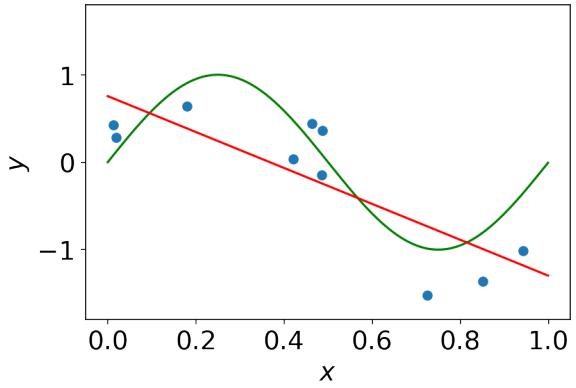


$M = 0$

Nonlinear basis expansion

Sinusoid example

Find the Empirical risk minimiser over each class $\mathcal{H}_M$, for $M = 0, \ldots, 9$

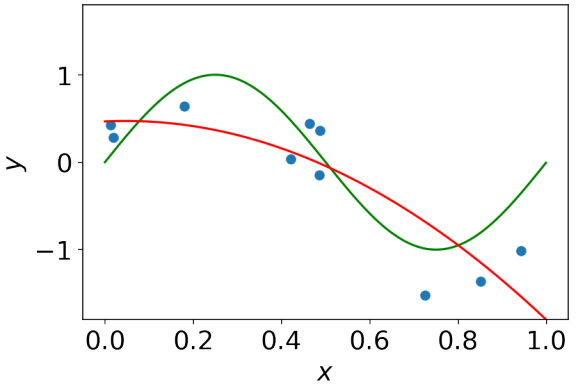


$M = 1$

Nonlinear basis expansion

Sinusoid example

Find the Empirical risk minimiser over each class $\mathcal{H}_M$, for $M = 0, \ldots, 9$

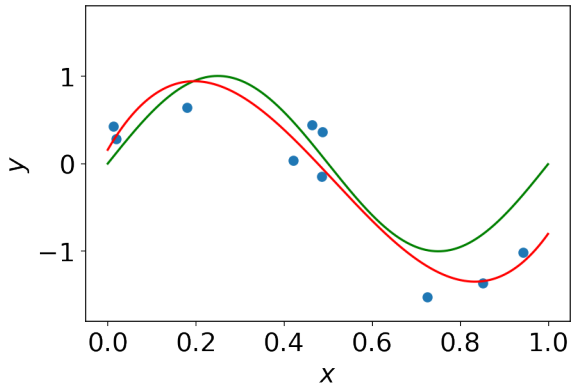


$M = 2$

Nonlinear basis expansion

Sinusoid example

Find the Empirical risk minimiser over each class $\mathcal{H}_M$, for $M = 0, \ldots, 9$

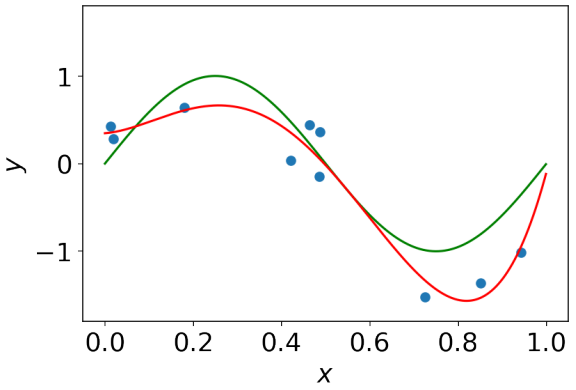


$M = 3$

Nonlinear basis expansion

Sinusoid example

Find the Empirical risk minimiser over each class $\mathcal{H}_M$, for $M = 0, \ldots, 9$

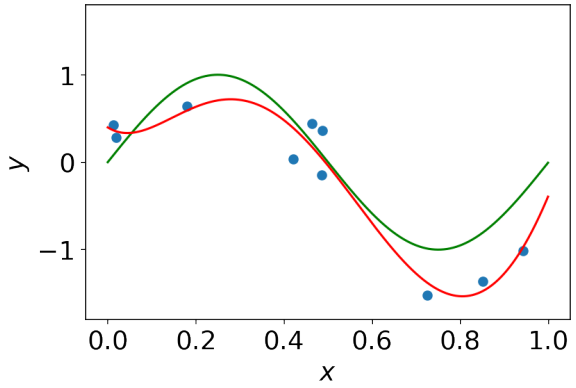


$M = 4$

Nonlinear basis expansion

Sinusoid example

Find the Empirical risk minimiser over each class $\mathcal{H}_M$, for $M = 0, \ldots, 9$

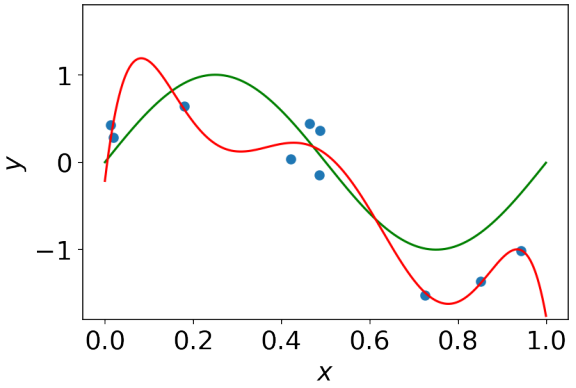


$M = 5$

Nonlinear basis expansion

Sinusoid example

Find the Empirical risk minimiser over each class $\mathcal{H}_M$, for $M = 0, \ldots, 9$

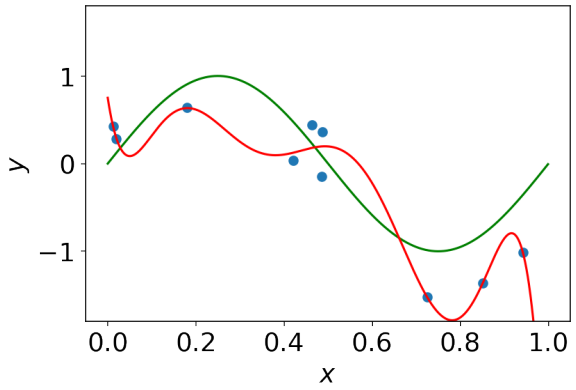


$M = 6$

Nonlinear basis expansion

Sinusoid example

Find the Empirical risk minimiser over each class $\mathcal{H}_M$, for $M = 0, \ldots, 9$

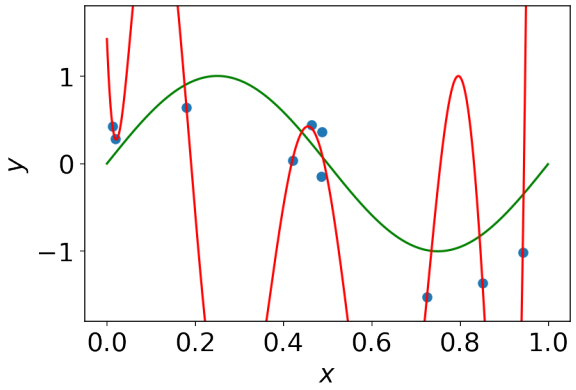


$M = 7$

Nonlinear basis expansion

Sinusoid example

Find the Empirical risk minimiser over each class $\mathcal{H}_M$, for $M = 0, \ldots, 9$



$M = 8$

Nonlinear basis expansion

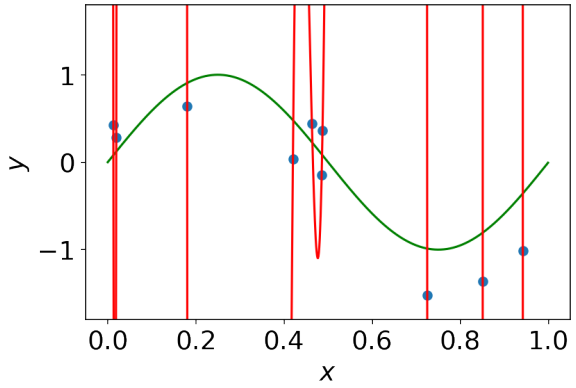
Sinusoid example

Find the Empirical risk minimiser over each class $\mathcal{H}_M$, for $M = 0, \ldots, 9$

$\hat{h}^{(d)}(x_i) = y_i$

$i = 1, \ldots, n$

$\widehat{R}_n(\hat{h}^{(d)}) = 0$



$M = 9$

Nonlinear basis expansion

Sinusoid example

For $M = 0$ and $M = 1$, the learned prediction rules are too simple.

We say that we are underfitting

For $M = 3$, the learned prediction rule is close to the Bayes prediction rule

For $M = 9$, the learned prediction interpolates the data ($\widehat{R}_n(\widehat{h}^{(d)}) = 0$)

But predictions are far from the Bayes prediction rule when x is not in the training set

Nonlinear basis expansion

Sinusoid example

Estimated coefficients take large values for $M = 9$, suggesting the estimator has large variance

	$M=0$	$M=1$	$M=3$	$M=9$
$\widehat{\beta}_0$	-0.19	0.76	0.16	171.61
$\widehat{\beta}_1$		-2.05	8.66	-24859.77
$\widehat{\beta}_2$			-27.29	1052225
$\widehat{\beta}_3$			17.67	-13060914
$\widehat{\beta}_4$				75449695
$\widehat{\beta}_5$				-239878655
$\widehat{\beta}_6$				446010421
$\widehat{\beta}_7$				-483353259
$\widehat{\beta}_8$				282706993
$\widehat{\beta}_9$				-68922625

Table: Estimates under for polynomial class of degree 0, 1, 3, and 9.

Nonlinear basis expansion

Sinusoid example

For large M we say that we are overfitting

Class of prediction rule is too complex for the amount of data available

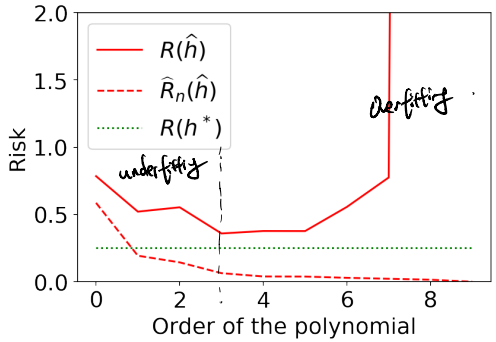
Fitting both the signal and the noise

Overfitting

Sinusoid example

Empirical risk decreases with the model complexity, to reach zero for $M = 9$

Population/Generalisation risk has a U shape: it first decreases (underfitting regime), then increases (overfitting regime)

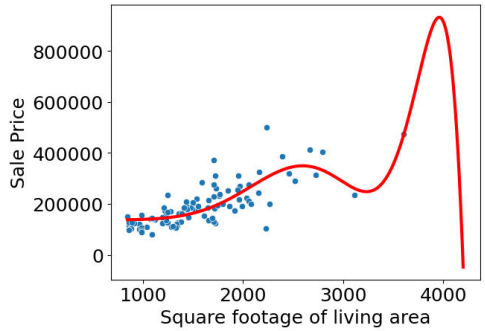


Overfitting

Overfitting can have disastrous effect

Polynomial of degree 11 on the house pricing data

Prediction rule decreases for large living area



Estimation-Approximation error

For a given set of prediction rules $\mathcal{H} \subset \mathcal{F}$, called an hypothesis class, denote

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

the best prediction rule, in terms of risk minimisation, amongst the set $\mathcal{H}$.

By definition,

$$R(\widehat{h}^{(d)}) \geq R(h^*_{\mathcal{H}}) \geq R(h^*)$$

where $\widehat{h}^{(d)}$ is the empirical risk minimiser

Excess risk can be decomposed as

$$\underbrace{R(\widehat{h}^{(d)}) - R(h^*)}_{\text{excess risk}} = \underbrace{R(\widehat{h}^{(d)}) - R(h^*_{\mathcal{H}})}_{\text{estimation error}} + \underbrace{R(h^*_{\mathcal{H}}) - R(h^*)}_{\text{approximation error}}$$

Estimation-Approximation error

The approximation error is the difference between the risk of the best prediction rule within the class and the Bayes risk.

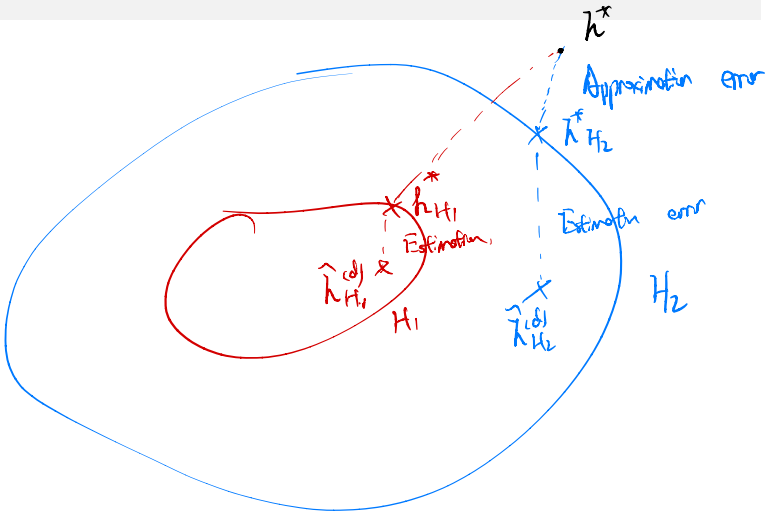
If the hypothesis class is large, more complex prediction rules can be captured, and the approximation error is thus smaller.

The estimation error is the difference between the risk of the learned prediction rule and of the best prediction rule in the class.

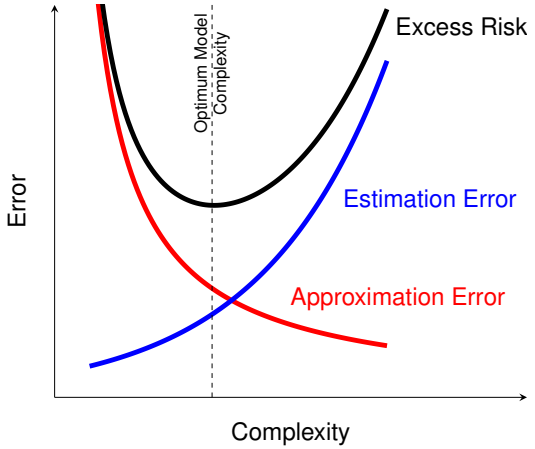
Larger classes have a large number of parameters to estimate, leading to a larger estimation error.

Estimation-Approximation error

Illustration



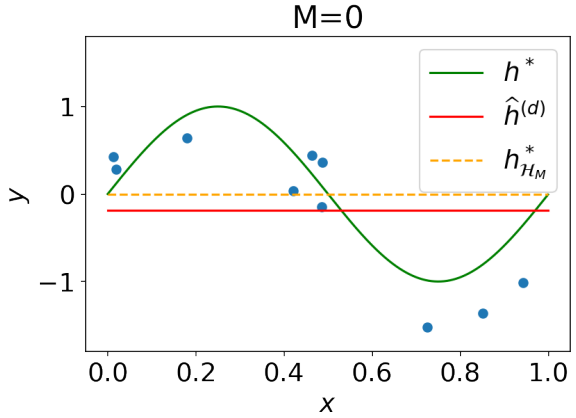
Estimation-Approximation error



Estimation-Approximation error

Sinusoid example

Complexity corresponds to the degree of the polynomial.
As the degree increases best prediction rule in the class becomes closer to the Bayes prediction rule, but the estimation error increases

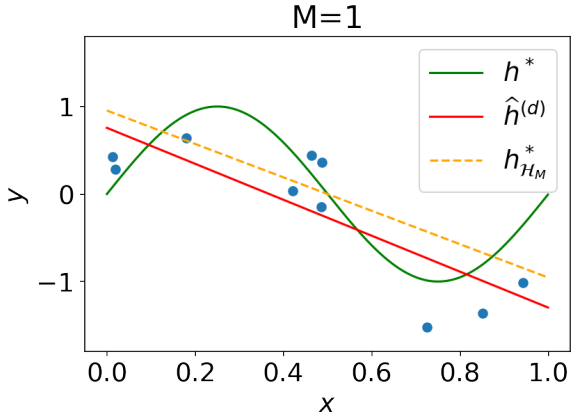


Estimation-Approximation and Bias-variance trade-off

Estimation-Approximation error

Sinusoid example

Complexity corresponds to the degree of the polynomial.
As the degree increases best prediction rule in the class becomes closer to the Bayes prediction rule, but the estimation error increases

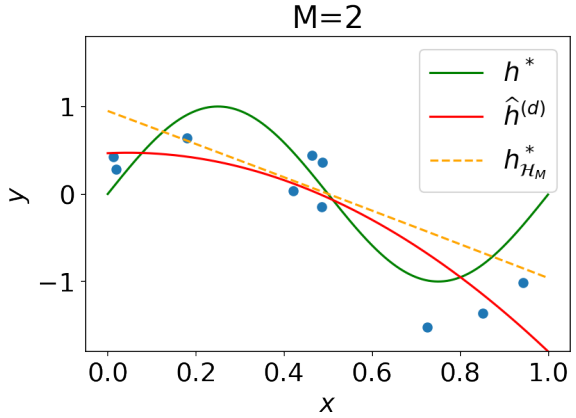


Estimation-Approximation and Bias-variance trade-off

Estimation-Approximation error

Sinusoid example

Complexity corresponds to the degree of the polynomial.
As the degree increases best prediction rule in the class becomes closer to the Bayes prediction rule, but the estimation error increases

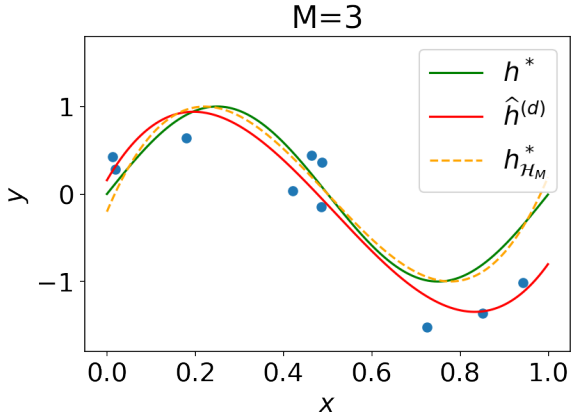


Estimation-Approximation and Bias-variance trade-off

Estimation-Approximation error

Sinusoid example

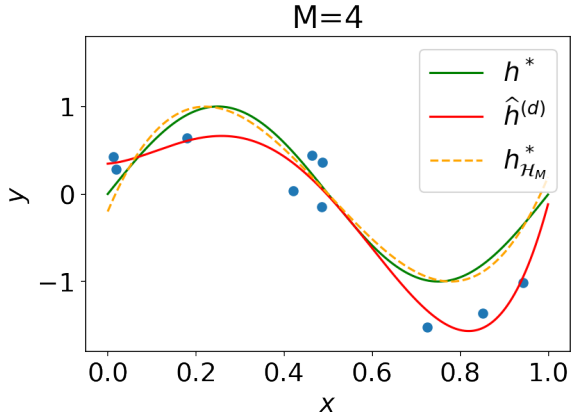
Complexity corresponds to the degree of the polynomial.
As the degree increases best prediction rule in the class becomes closer to the Bayes prediction rule, but the estimation error increases



Estimation-Approximation error

Sinusoid example

Complexity corresponds to the degree of the polynomial.
As the degree increases best prediction rule in the class becomes closer to the Bayes prediction rule, but the estimation error increases

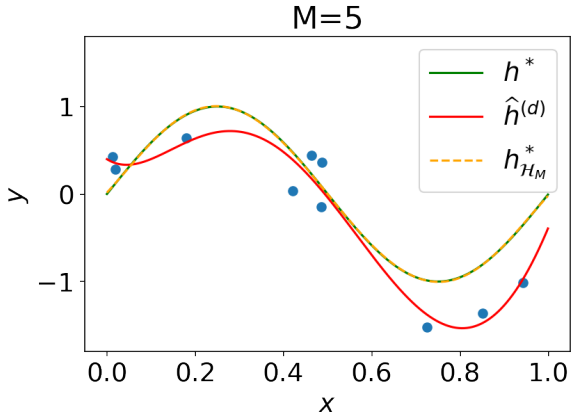


Estimation-Approximation and Bias-variance trade-off

Estimation-Approximation error

Sinusoid example

Complexity corresponds to the degree of the polynomial.
As the degree increases best prediction rule in the class becomes closer to the Bayes prediction rule, but the estimation error increases

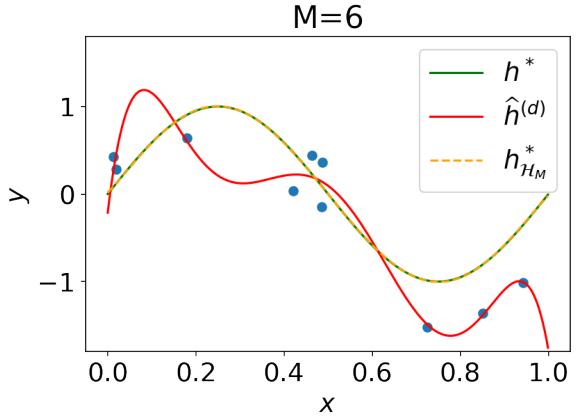


Estimation-Approximation and Bias-variance trade-off

Estimation-Approximation error

Sinusoid example

Complexity corresponds to the degree of the polynomial.
As the degree increases best prediction rule in the class becomes closer to the Bayes prediction rule, but the estimation error increases

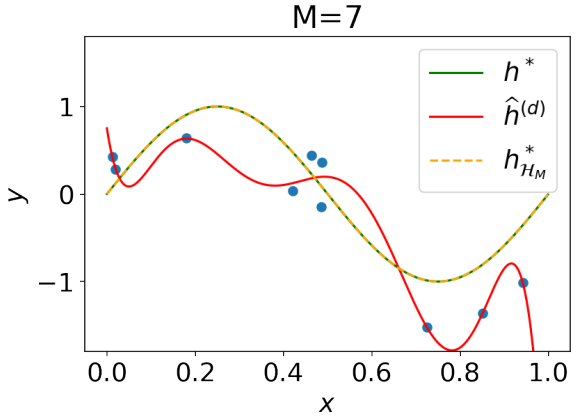


Estimation-Approximation and Bias-variance trade-off

Estimation-Approximation error

Sinusoid example

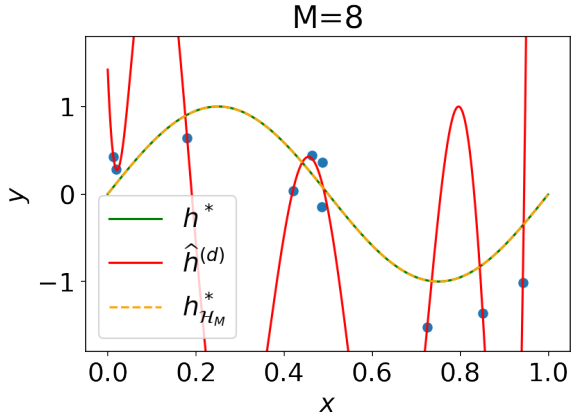
Complexity corresponds to the degree of the polynomial.
As the degree increases best prediction rule in the class becomes closer to the Bayes prediction rule, but the estimation error increases



Estimation-Approximation error

Sinusoid example

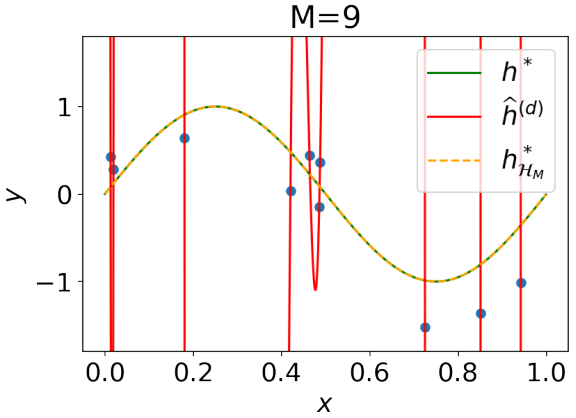
Complexity corresponds to the degree of the polynomial.
As the degree increases best prediction rule in the class becomes closer to the Bayes prediction rule, but the estimation error increases



Estimation-Approximation error

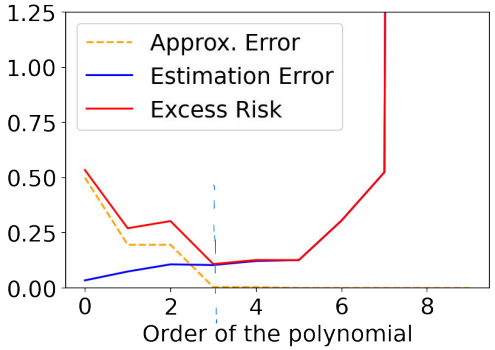
Sinusoid example

Complexity corresponds to the degree of the polynomial.
As the degree increases best prediction rule in the class becomes closer to the Bayes prediction rule, but the estimation error increases



Estimation-Approximation error

Sinusoid example



Approximation error and size of the dataset

Approximation error only depends on the class $\mathcal{H}$, not the dataset d

Under mild assumptions, the estimation error goes to 0 as the number of observations n goes to infinity

The excess risk converges to the approximation error or, equivalently, the risk $R(\widehat{h})$ converges to the risk $R(h^*_{\mathcal{H}})$ of the best prediction rule in the class as $n \to \infty$.

Empirical risk and population risk

Empirical risk of the learned rule $\widehat{R}_n(\widehat{h})$ typically underestimates the true risk $R(\widehat{h})$

By definition of the ERM in the class $\mathcal{H}$, we have $R(\widehat{h}^{(d)}) \geq R(h^*_{\mathcal{H}})$ for any dataset d.

But (see Problem Sheet)

$$\mathbb{E}[\widehat{R}_n^{(D)}(\widehat{h}^{(D)})] \leq R(h^*_{\mathcal{H}})$$

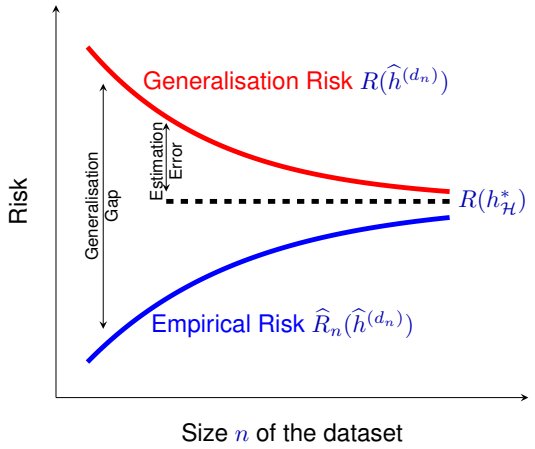
where the expectation is taken with respect to the random sample D.

Generalisation gap

$$R(\widehat{h}) - \widehat{R}_n(\widehat{h}).$$

Generalisation gap usually converges to 0 as the sample size n increases, as both $R(\widehat{h})$ and $\widehat{R}_n(\widehat{h})$ will converge to $R(h^*_{\mathcal{H}})$

Empirical risk and population risk

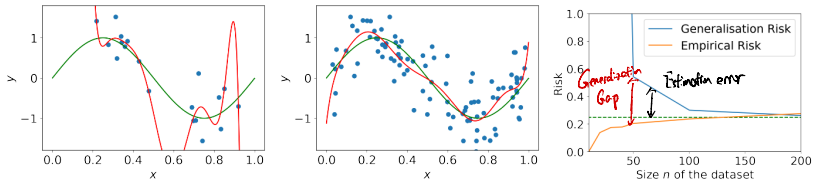


Approximation error and size of the dataset

Sinusoid example

ERM with hypothesis class the set $\mathcal{H}_9$ of polynomials of order 9

Different sample sizes $n = 20, 100$



Bias-variance decomposition

For the squared loss, another traditional interpretable decomposition of the excess risk is the bias-variance decomposition

Under the squared loss, the excess risk takes the form

$$R(\widehat{h}^{(D)}) - R(h^*) = \mathbb{E}_X[(\widehat{h}^{(D)}(X) - h^*(X))^2].$$

Random variable, as it depends on the random sample $D = ((X_1, Y_1), \ldots, (X_n, Y_n))$.

Bias-variance decomposition

Taking the expectation over the dataset D, we obtain the expected excess risk

$$\mathbb{E}_D[R(\widehat{h}^{(D)}) - R(h^*)] = \mathbb{E}_D\mathbb{E}_X[(\widehat{h}^{(D)}(X) - h^*(X))^2]$$
$$= \mathbb{E}_X\mathbb{E}_D[(\widehat{h}^{(D)}(X) - h^*(X))^2]$$

Excess risk

Bias-variance decomposition

Background material

Let $\widehat{\theta}$ be an estimator of a fixed parameter θ_0

Mean $\mathbb{E}[\widehat{\theta}]$, variance $\text{var}(\widehat{\theta})$ and bias $\text{bias}(\widehat{\theta}) = \mathbb{E}[\widehat{\theta}] - \theta_0$

The Mean Squared error of the estimator satisfies the following bias-variance decomposition

$$\mathbb{E}[(\widehat{\theta} - \theta_0)^2] = \mathbb{E}[((\widehat{\theta} - \mathbb{E}[\widehat{\theta}]) - (\theta_0 - \mathbb{E}[\widehat{\theta}]))^2]$$
$$= \mathbb{E}[(\widehat{\theta} - \mathbb{E}[\widehat{\theta}])^2] + (\theta_0 - \mathbb{E}[\widehat{\theta}])^2 - 2(\theta_0 - \mathbb{E}[\widehat{\theta}])\mathbb{E}[(\widehat{\theta} - \mathbb{E}[\widehat{\theta}])]$$
$$= \text{var}(\widehat{\theta}) + \text{bias}(\widehat{\theta})^2$$

0

Bias-variance decomposition

For any $x \in \mathcal{X}$, the random variable $\widehat{h}^{(D)}(x)$ can be seen as an estimator of the Bayes predictor $h^*(x)$

This estimator has mean $\bar{h}_{\mathcal{H},n}(x) = \mathbb{E}_D[\widehat{h}^{(D)}(x)]$ and bias and variance

$$\begin{cases} b_{\mathcal{H},n}(x) = \bar{h}_{\mathcal{H},n}(x) - h^*(x) \\ v_{\mathcal{H},n}(x) = \mathbb{E}_D[(\widehat{h}^{(D)}(x) - \bar{h}_{\mathcal{H},n}(x))^2] \end{cases}$$

For any x, we have the bias-variance decomposition of the estimator $\widehat{h}^{(D)}(x)$

$$\mathbb{E}_D[(\widehat{h}^{(D)}(x) - h^*(x))^2]$$
$$= \mathbb{E}_D[(\widehat{h}^{(D)}(x) - \bar{h}_{\mathcal{H},n}(x))^2 + (\bar{h}_{\mathcal{H},n}(x) - h^*(x))^2 + 2(\widehat{h}^{(D)}(x) - \bar{h}_{\mathcal{H},n}(x))(\bar{h}_{\mathcal{H},n}(x) - h^*(x))]$$
$$= v_{\mathcal{H},n}(x) + b_{\mathcal{H},n}(x)^2.$$

$\widehat{h}^{(D)}(x) - h^*(x) = (\widehat{h}^{(D)}(x) - \bar{h}(x)) - (h^*(x) - \bar{h}(x))$

Bias-variance decomposition

It follows that

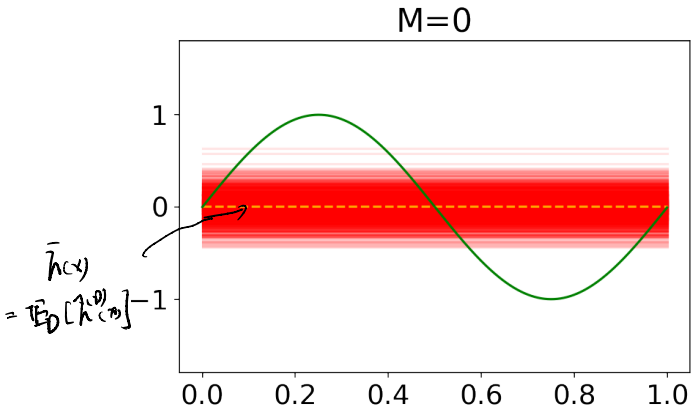
$$\underbrace{\mathbb{E}_D[R(\widehat{h}^{(D)}) - R(h^*)]}_{\text{expected excess risk}} = \underbrace{\mathbb{E}_X[v_{\mathcal{H},n}(X)]}_{\text{variance term}} + \underbrace{\mathbb{E}_X[b_{\mathcal{H},n}(X)^2]}_{\text{bias term}}$$

The variance term relates to the estimation error; it will typically be large for large hypothesis sets $\mathcal{H}$. This term usually vanishes as the number of observations grows.
The bias term relates to the approximation error; it will typically be large for small hypothesis sets $\mathcal{H}$. This term does not usually vanish as the number of observations grows.

Bias-variance decomposition

Sinusoid example

Generate multiple datasets of size $n = 30$, and plot the learned prediction rule

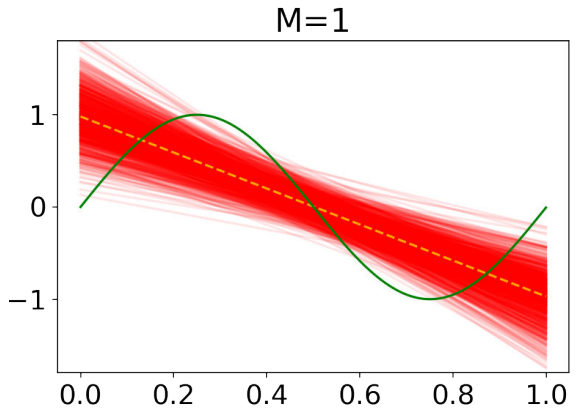


Estimation-Approximation and Bias-variance trade-off

Bias-variance decomposition

Sinusoid example

Generate multiple datasets of size $n = 30$, and plot the learned prediction rule

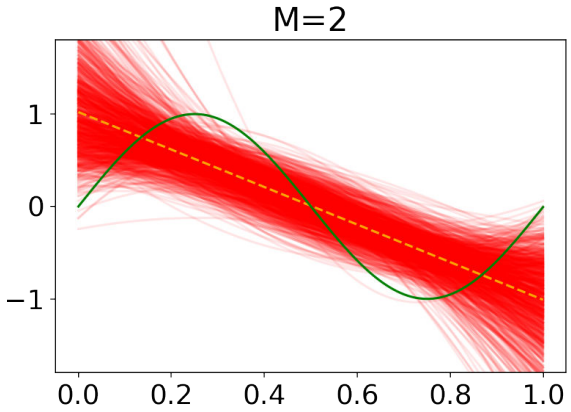


Estimation-Approximation and Bias-variance trade-off

Bias-variance decomposition

Sinusoid example

Generate multiple datasets of size $n = 30$, and plot the learned prediction rule

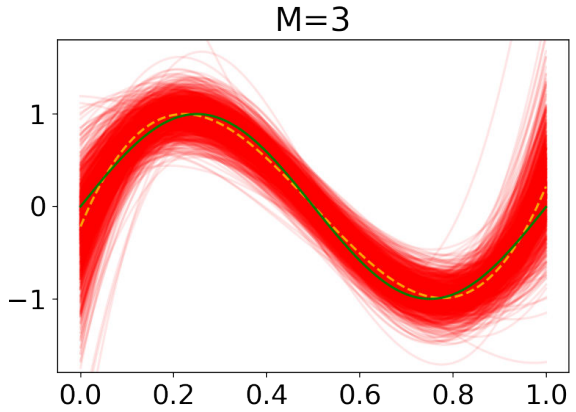


Estimation-Approximation and Bias-variance trade-off

Bias-variance decomposition

Sinusoid example

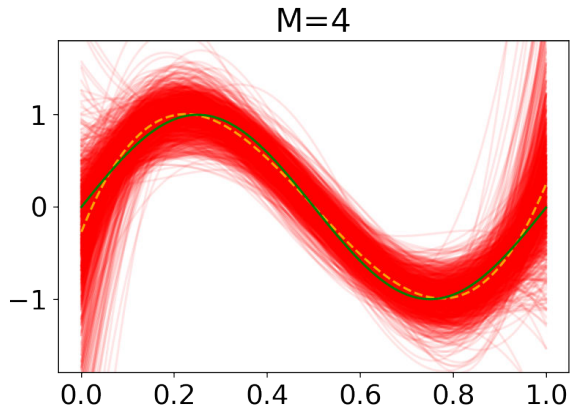
Generate multiple datasets of size $n = 30$, and plot the learned prediction rule



Bias-variance decomposition

Sinusoid example

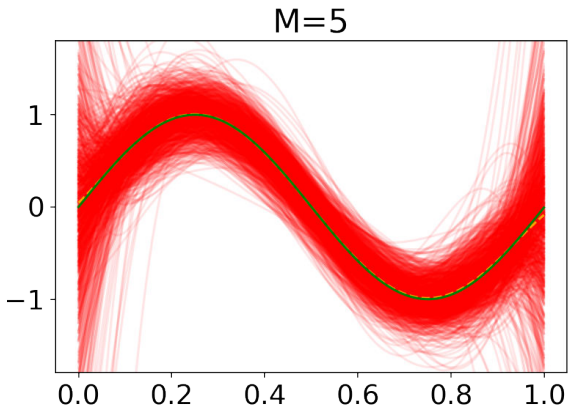
Generate multiple datasets of size $n = 30$, and plot the learned prediction rule



Bias-variance decomposition

Sinusoid example

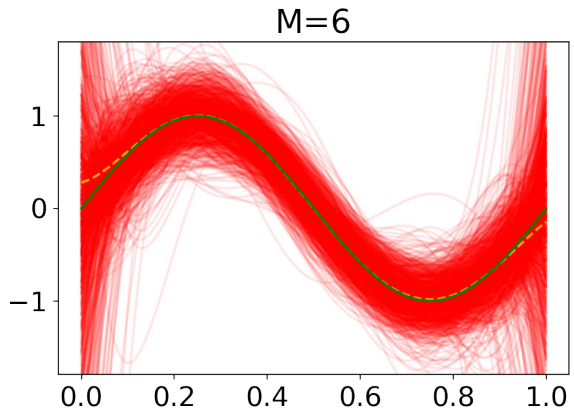
Generate multiple datasets of size $n = 30$, and plot the learned prediction rule



Bias-variance decomposition

Sinusoid example

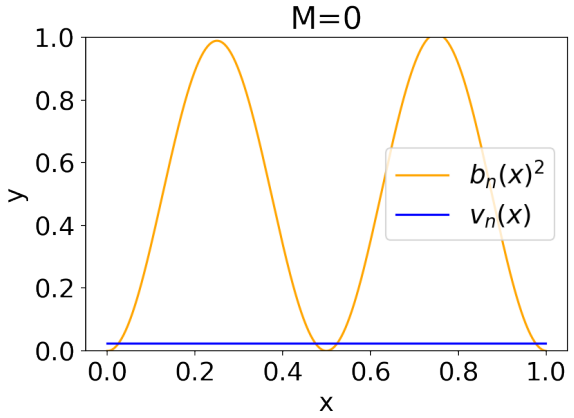
Generate multiple datasets of size $n = 30$, and plot the learned prediction rule



Bias-variance decomposition

Sinusoid example

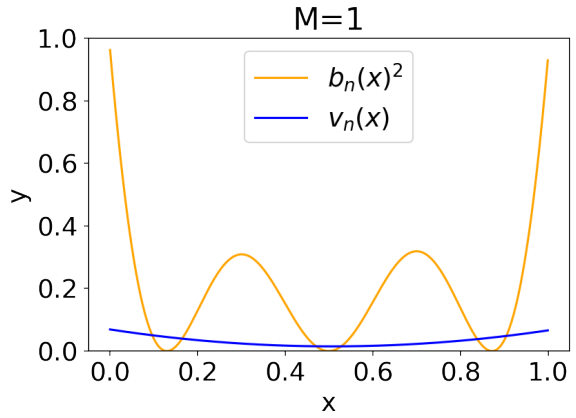
Bias-variance functions



Bias-variance decomposition

Sinusoid example

Bias-variance functions

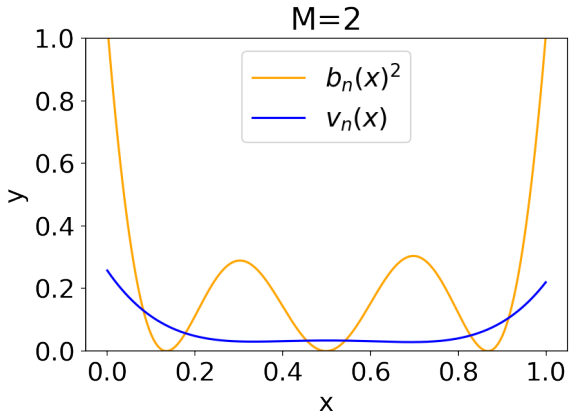


Estimation-Approximation and Bias-variance trade-off

Bias-variance decomposition

Sinusoid example

Bias-variance functions

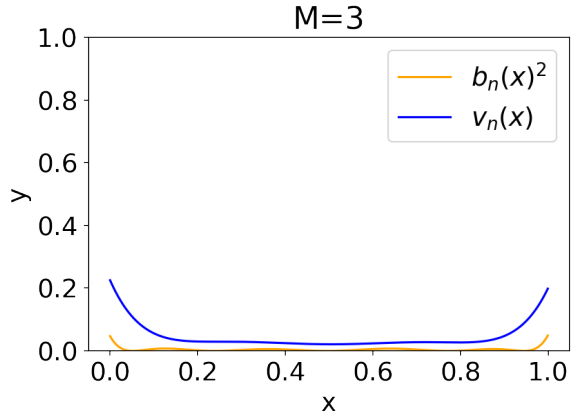


Estimation-Approximation and Bias-variance trade-off

Bias-variance decomposition

Sinusoid example

Bias-variance functions

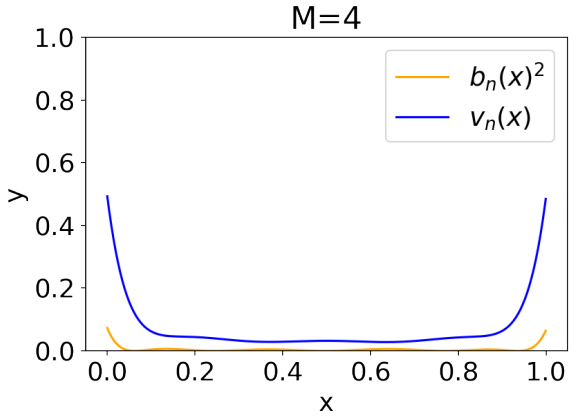


Estimation-Approximation and Bias-variance trade-off

Bias-variance decomposition

Sinusoid example

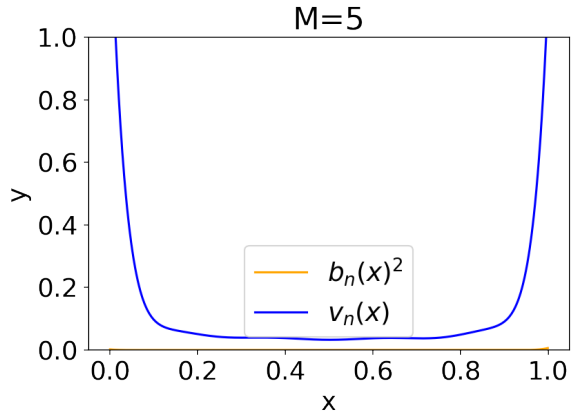
Bias-variance functions



Bias-variance decomposition

Sinusoid example

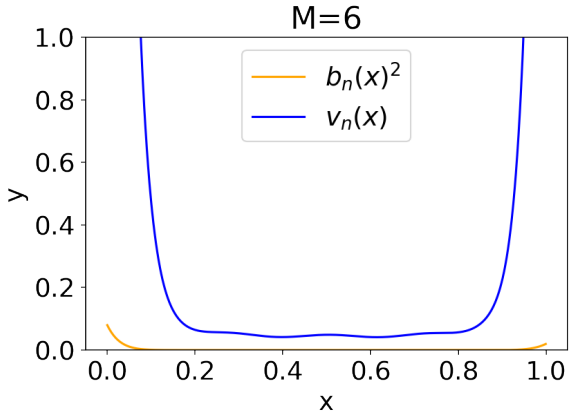
Bias-variance functions



Bias-variance decomposition

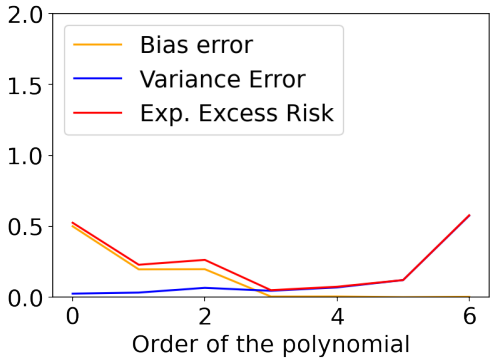
Sinusoid example

Bias-variance functions



Bias-variance decomposition

Sinusoid example



Regularisation

18/2

An alternative to considering classes of different complexity/capacity is to consider a single (large) class of prediction rules, and penalise more complex prediction rules within this class

Let $\mathrm{pen} : \mathcal{F} \to \mathbb{R}_+$ be a penalty or regularisation function

$\mathrm{pen}(h)$ is a measure of the complexity of the prediction rule h; larger values correspond to more complex prediction rules, while smaller values are less complex prediction rules.

Regularisation

Definition

Let d be a dataset. For a loss function L, a hypothesis class $\mathcal{H} \subset \mathcal{F}$, a penalty function pen and a regularisation parameter $\lambda \geq 0$, the regularised empirical risk minimiser is given by

$$\widehat{h}^{(d)} = \arg\min_{h \in \mathcal{H}} \left\{ \widehat{R}_n(h) + \frac{\lambda}{n} \text{pen}(h) \right\}.$$

Regularisation

The empirical risk $\widehat{R}_n(h)$ measures the fit of the prediction rule to the data

More complex prediction rules will typically give a better fit on the data

$\frac{\lambda}{n} \text{pen}(h)$ penalises the complexity of the prediction rule

We seek the best compromise between the fit to the data and the complexity

The regularisation parameter λ controls this trade-off

- If $\lambda = 0$, this corresponds to the ERM solution: there is no penalty for the complexity, and the learned prediction rule is likely to overfit if the class is large.
- As $\lambda \to \infty$, a very large penalty is enforced for the model's complexity, and simpler prediction rules will be chosen.

In terms of bias-variance decomposition, larger values of λ will lead to higher bias and smaller variance.

Regularisation

If the prediction rules $h \in \mathcal{H}$ are parameterised by a vector $\beta = (\beta_0, \ldots, \beta_M)^\intercal \in \mathbb{R}^{M+1}$, a standard measure of the complexity of the prediction rule is the squared L2 norm

$$||\beta||^2 = \beta^\intercal \beta = \sum_{j=0}^{M} \beta_j^2$$

The associated penalty, known as Tikhonov regularisation, is therefore

$$\text{pen}(h) = \|\beta\|^2 = \sum_{j=0}^{M} \beta_j^2.$$

Regularisation

For example, for regression under the squared loss, if $\mathcal{H}$ is the class of linear prediction rules on some transformed inputs $\phi(x)$,

$$\mathcal{H} = \{h(x) = \phi(x)^\intercal \beta \text{ where } \beta \in \mathbb{R}^{M+1}\}$$

the regularised ERM is $\widehat{h}^{(d)}(x) = \phi(x)^\intercal \widehat{\beta}$ where

$$\begin{aligned}\widehat{\beta} &= \arg\min_{\beta \in \mathbb{R}^{M+1}} \sum_{i=1}^{n} (y_i - \phi(x_i)^\intercal \beta)^2 + \lambda \|\beta\|^2 \\ &= \arg\min_{\beta \in \mathbb{R}^{M+1}} \|\mathbf{y} - \mathbf{\Phi}\beta\|^2 + \lambda \|\beta\|^2\end{aligned}$$

Regularisation

$$\|\mathbf{y} - \mathbf{\Phi}\beta\|^2 + \lambda\|\beta\|^2 = \beta^\intercal \mathbf{\Phi}^\intercal \mathbf{\Phi}\beta - 2(\mathbf{\Phi}^\intercal \mathbf{y})^\intercal \beta + \lambda\beta^\intercal\beta + \text{const}$$

Differentiating

$$\frac{\partial(\|\mathbf{y} - \mathbf{\Phi}\beta\|^2 + \lambda\|\beta\|^2)}{\partial\beta} = 2\mathbf{\Phi}^\intercal\mathbf{\Phi}\beta - 2\Phi^\intercal\mathbf{y} + 2\lambda\beta = 0$$

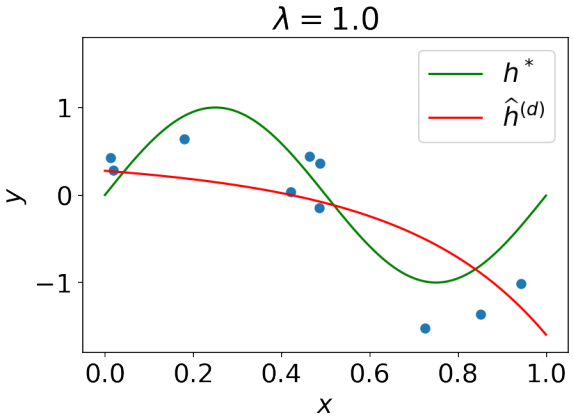
Ridge regression estimator

$$\widehat{\beta} = (\mathbf{\Phi}^\intercal\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^\intercal\mathbf{y}$$

Regularisation

Sinusoid example

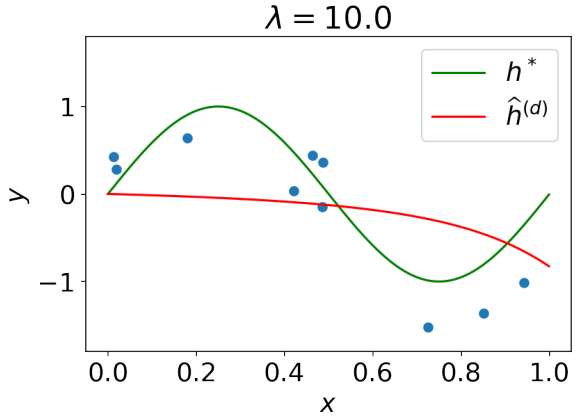
$n = 10$ observations
Class of linear models with polynomial expansion of order $M = 9$
Regularised ERM, with regularisation parameter λ



Regularisation

Sinusoid example

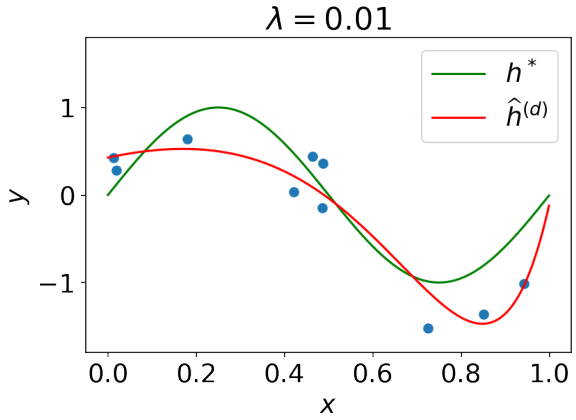
$n = 10$ observations
Class of linear models with polynomial expansion of order $M = 9$
Regularised ERM, with regularisation parameter λ



Regularisation

Sinusoid example

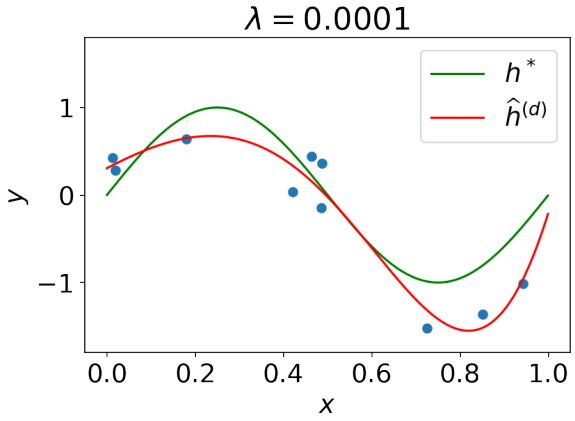
$n = 10$ observations
Class of linear models with polynomial expansion of order $M = 9$
Regularised ERM, with regularisation parameter λ



Regularisation

Sinusoid example

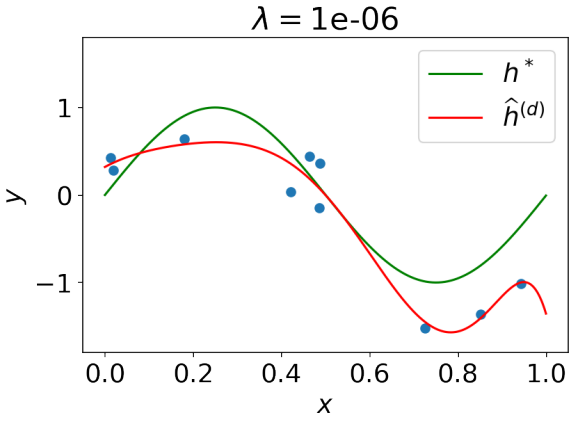
$n = 10$ observations
Class of linear models with polynomial expansion of order $M = 9$
Regularised ERM, with regularisation parameter λ



Regularisation

Sinusoid example

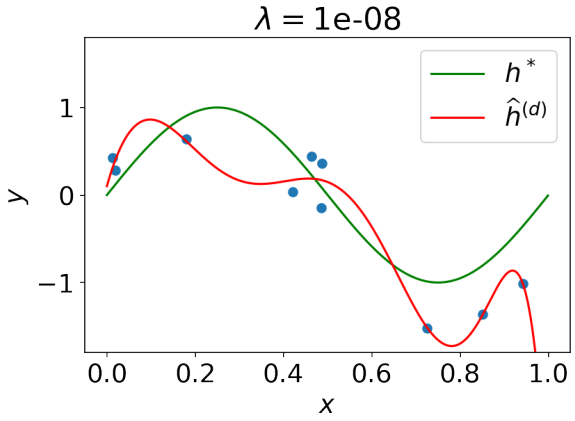
$n = 10$ observations
Class of linear models with polynomial expansion of order $M = 9$
Regularised ERM, with regularisation parameter λ



Regularisation

Sinusoid example

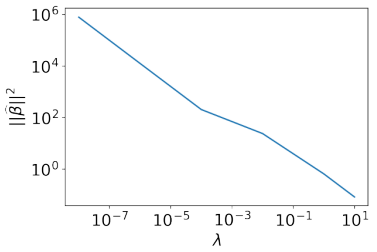
$n = 10$ observations
Class of linear models with polynomial expansion of order $M = 9$
Regularised ERM, with regularisation parameter λ

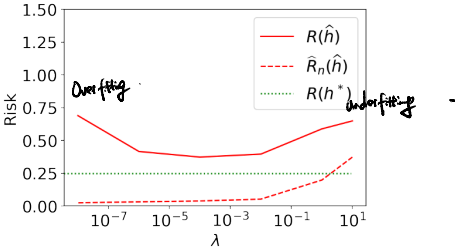


Regularisation

Sinusoid example

Norm of $\widehat{\beta}$ decreases with λ
Population Risk has a U shape: Overfitting for small values of λ (too complex), underfitting for large values of λ (too simple)
Empirical risk increases with λ

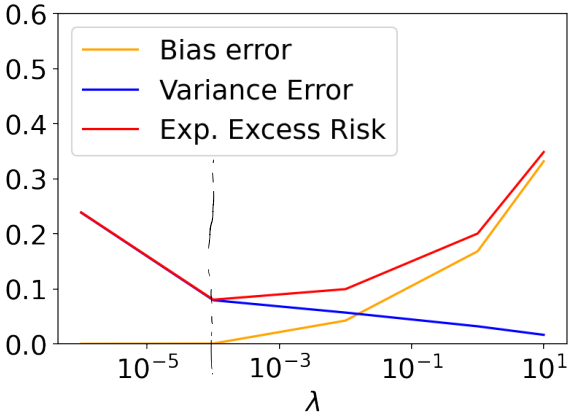




Regularisation

Sinusoid example

Bias-variance error as a function of the regularisation parameter λ ($n = 30$ here)



Regularisation and sample size

As the sample size $n \to \infty$, for any fixed h,

$$\frac{\lambda}{n}\,\mathrm{pen}(h) \to 0$$

$$\widehat{R}_n(h) \to R(h) \quad \text{almost surely}$$

Effect of the regularisation vanishes as we have more data

Regularisation

Other penalisation functions

L1 (factor 2 introduced for mathematical convenience)

$$\mathrm{pen}(h) = 2\|\beta\|_1 = 2\sum_{j=0}^{M} |\beta_j|$$

L1+L2 (elastic net)

$$\mathrm{pen}(h) = 2\delta\|\beta\|_1 + (1-\delta)\|\beta\|_2^2$$

for some $\delta \in [0, 1]$.

Test, validation and cross-validation

Assume that we obtain a set of M_{max} learned prediction rules $\widehat{h}_1^{(d)}, \ldots, \widehat{h}_{M_{\text{max}}}^{(d)}$

Learned Prediction rules may be obtained:

- Using ERM on hypothesis sets $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \ldots \subset \mathcal{H}_{M_{\text{max}}}$ of increasing complexity
- Using ERM on a large class $\mathcal{H}$, with different regularisation parameters $0 \leq \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_{M_{\text{max}}}$
- More generally, they may be obtained by using different hyperparameters, learning algorithms, feature expansions, etc.

Test, validation and cross-validation

As we have seen, some learned prediction rules will overfit or underfit.

We would like to

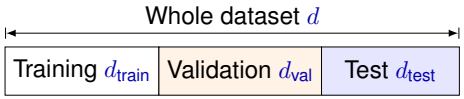
- Choose a prediction rule with small generalisation risk
- Estimate its generalisation risk (expected error for predicting new observations)

Two caveats:

- The true risk cannot be computed as the true distribution of the data is unknown;
- The empirical risk of the learned prediction rule can be computed but, as we have seen, it largely underestimates the true risk, in particular for complex models.

Training-Validation-Test procedure

The idea is to split the dataset in three parts: a training set, a validation set, and a test set.



Training-Validation-Test procedure

Denote d_{train}, d_{val} and d_{test} respectively the training, validation and test set

Assume that the training, validation and test set are realisations from D_{train}, D_{val} and D_{test}, iid random samples

Denote $\widehat{R}^{(d_{\text{train}})}(h)$, $\widehat{R}^{(d_{\text{val}})}(h)$ and $\widehat{R}^{(d_{\text{test}})}(h)$ the empirical risk of a prediction rule h on the training, validation and training sets

Training-Validation-Test procedure

Training: For each $j = 1, \ldots, M_{\text{max}}$, estimate the learned prediction rule $\widehat{h}_j^{(d_{\text{train}})}$ using the training set d_{train}. For instance, when considering the ERM on a class $\mathcal{H}_j$

$$\widehat{h}_j^{(d_{\text{train}})} = \underset{h \in \mathcal{H}_j}{\arg\min}\, \widehat{R}^{(d_{\text{train}})}(h).$$

Validation (model selection):

a) For each j, calculate $\widehat{R}^{(d_{\text{val}})}(\widehat{h}_j^{(d_{\text{train}})})$ and report the best learner $\widehat{M} = \arg\min_j \widehat{R}^{(d_{\text{val}})}(\widehat{h}_j^{(d_{\text{train}})})$

b) Re-train the prediction rule on both the training and validation set for the chosen model $\widehat{M}$. For instance, for ERM on a class $\mathcal{H}_j$

$$\widehat{h}^{(d_{\text{train}}, d_{\text{val}})} = \underset{h \in \mathcal{H}_{\widehat{M}}}{\arg\min}\, \widehat{R}^{(d_{\text{train}}, d_{\text{val}})}(h)$$

Test: Approximate the generalisation error with $\widehat{R}^{(d_{\text{test}})}(\widehat{h}^{(d_{\text{train}}, d_{\text{val}})})$.

Training-Validation-Test procedure

In Step 1, as we use the training set d_{train} to estimate the $\widehat{h}_j^{(d_{\text{train}})}$, we cannot use $\widehat{R}^{(d_{\text{train}})}(\widehat{h}_j^{(d_{\text{train}})})$ as an estimate for $R(\widehat{h}_j^{(d_{\text{train}})})$

We therefore use in step 2a) another dataset (the validation set), that has not been used for training

For any j, and any training set d_{train},

$$\mathbb{E}[\widehat{R}^{(D_{\text{val}})}(\widehat{h}_j^{(D_{\text{train}})}) \mid D_{\text{train}} = d_{\text{train}}] = R(\widehat{h}_j^{(d_{\text{train}})})$$

where the expectation is taken over the validation random sample.

Unbiased estimate of the true risk of the jth learned prediction rule.

Error of order $O(1/n_{\text{val}})$

Training-Validation-Test procedure

After selecting the best class $\widehat{M}$, we could use $\widehat{h}_{\widehat{M}}^{(d_{\text{train}})}$ as the estimated prediction rule.

However, to reduce the estimation error, it is better to re-train on both the training and validation sets for this model.

The training and validation datasets having been used to estimate $\widehat{h}^{(d_{\text{train}},d_{\text{val}})}$, we get an estimate of the risk on a third dataset (the test set)

We have

$$\mathbb{E}[\widehat{R}^{(D_{\text{test}})}(\widehat{h}^{(D_{\text{train}},D_{\text{val}})}) \mid D_{\text{train}} = d_{\text{train}}, D_{\text{val}} = d_{\text{val}}] = R(\widehat{h}^{(d_{\text{train}},d_{\text{val}})})$$

where the expectation is taken over the test set

Unbiased estimate of the true risk of the learned prediction rule

Error of order $O(1/n_{\text{test}})$

Training-Validation-Test procedure

How to split the dataset?

Any data in the validation and test set is not used to initially train the different models

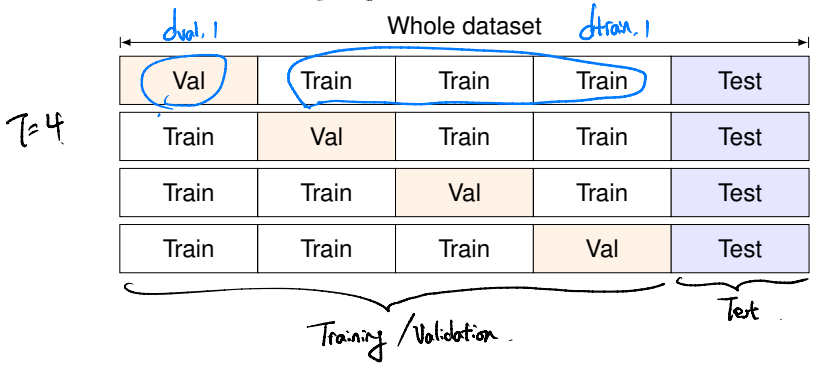
If the training set is too small, this may lead to large estimation errors for the learned prediction rules

If the validation set or test sets are small, this leads to a large error on the estimated risks

In the case where the dataset is small or the learners need a lot of data, an alternative is to use cross-validation.

Cross-validation

The idea of T-fold cross-validation is to split the dataset into a training/validation set and a test set, then to further split the training/validation set into T folds.



Cross-validation

Training: For each fold $t = 1, \ldots, T$

Use fold t as a validation set, and the rest as a training set to train each learner j; denote $\widehat{h}_j^{(d_{\text{train},t})}$ the jth learned prediction rule, and

$$\widehat{R}^{(d_{\text{val},t})}(\widehat{h}_j^{(d_{\text{train},t})})$$

its estimated risk

Validation:

a) Choose the learner that minimises the estimated risk averaged over the folds

$$\widehat{M} = \arg\min_j \frac{1}{T} \sum_{t=1}^{T} \widehat{R}^{(d_{\text{val},t})}(\widehat{h}_j^{(d_{\text{train},t})})$$

b) Re-train the prediction rule on the whole training/validation set for the chosen learner $\widehat{M}$.

Test: Approximate the generalisation error with $\widehat{R}^{(d_{\text{test}})}(\widehat{h}^{(d_{\text{train/val}})})$.

Cross-validation

A special case of cross-validation is leave-one-out cross-validation

One data item per fold, that is T equals the number of observations in training/validation set

Cross-validation can be computationally intensive, as the computational cost is multiplied by the number of folds T.

Performance evaluation for binary classification

We now consider binary classification, $\mathcal{Y} = \{-1, 1\}$

Performance evaluation of binary classifiers

Performance evaluation for binary classification

Let y be the true class, and $h(x)$ be the predicted class. The following table summarises the different configurations.

	$h(x) = -1$	$h(x) = 1$
$y = -1$	*true negative*	*false positive*
$y = 1$	*false negative*	*true positive*

The matrix counting the number of true/false positive/negative over a dataset d is called a confusion matrix

We denote TN, FN, FP, TP respectively the number of true negative, false negative, false positive and true positive over the dataset d for the classifier h.

$$TN = \sum_{i=1}^{n} \mathbb{1}_{y_i = h(x_i) = -1}$$

Performance evaluation for binary classification

So far, as a measure of the quality of a classifier h, we have focused on the (population) risk under a 0-1 loss, which is the misclassification error, or error rate

$$R(h) = \Pr(Y \neq h(X)).$$

Empirical misclassification error

$$\widehat{R}_n^{(d)}(h) = \frac{FN + FP}{FN + FP + TP + TN}.$$

Performance evaluation for binary classification

For binary classification, other quantities are often considered to assess the performance of a classifier

Name	Population	Empirical
Misclassification error/Error rate	$\Pr(Y \neq h(X))$	$\frac{FP+FN}{TP+TN+FP+FN}$
Accuracy (1-Error rate)	$\Pr(Y = h(X))$	$\frac{TP+TN}{TP+TN+FP+FN}$
Sensitivity/Recall/True positive rate	$\Pr(h(X) = 1 \mid Y = 1)$	$\frac{TP}{TP+FN}$
Specificity/True negative rate	$\Pr(h(X) = -1 \mid Y = -1)$	$\frac{TN}{TN+FP}$
False positive rate (1-specificity)	$\Pr(h(X) = 1 \mid Y = -1)$	$\frac{FP}{TN+FP}$
Precision	$\Pr(Y = 1 \mid h(X) = 1)$	$\frac{TP}{TP+FP}$

Weighted loss

Two possible types of error may occur:

- False positive ($y = -1, h(x) = 1$), also called Type I error,
- False negative ($y = 1, h(x) = -1$), also called Type II error.

The $0-1$ loss assigns an equal cost of 1 to each type of error

	$h(x) = -1$	$h(x) = 1$
$y = -1$	$L(y, h(x)) = 0$	$L(y, h(x)) = 1$
$y = 1$	$L(y, h(x)) = 1$	$L(y, h(x)) = 0$

Weighted loss

In some cases, it is more sensible to assign different costs to the different types of error.

For example, when predicting if a patient has a given disease, we may assign a higher cost to a false negative error than a false positive.

We consider in this case a loss

$$L(y, h(x)) = a\mathbb{1}_{y=-1,h(x)=1} + b\mathbb{1}_{y=1,h(x)=-1}$$

where $a, b > 0$ are respectively the type I and type II cost.

Bayes classifier under this loss function is (see problem sheet)

$$h_t^*(x) = \begin{cases} 1 & \text{if } \eta(x) \geq t := \frac{a}{a+b} \\ -1 & \text{otherwise} \end{cases}$$

where $\eta(x) = \Pr(Y = 1|X = x)$.

Weighted loss

Bayes classifier

$$h_t^*(x) = \begin{cases} 1 & \text{if } \eta(x) \geq t := \frac{a}{a+b} \\ -1 & \text{otherwise} \end{cases}$$

where $\eta(x) = \Pr(Y = 1|X = x)$

The Bayes classifier only depends on a and b through the ratio $t = a/(a + b)$

$t = 1/2$ corresponding to the Bayes classifier under the $0 - 1$ loss

A large value t will allow more false negative and less false positive; a small value allows for more false positive and less false negative.

ROC Curve and AUC

For a given weighted loss with ratio t, let $\widehat{h}_t$ be some plug-in (generative or discriminative) classifier, defined by

$$\widehat{h}_t(x) = \begin{cases} 1 & \text{if } \widehat{\eta}(x) \geq t \\ -1 & \text{otherwise} \end{cases}$$

where $\widehat{\eta}(x)$ is the estimate of $\eta(x) = \Pr(Y = 1 \mid X = x)$, and does not depend on the choice of the threshold t.

ROC Curve and AUC

How can we assess the performance of the family of plug-in classifiers $(\widehat{h}_t)$ for $t \in [0, 1]$?

A classical strategy is the ROC curve and the Area Under the Curve (AUC)

ROC Curve and AUC

For a threshold $t \in [0, 1]$, denote $\beta(t) = \Pr(\widehat{h}_t(X) = 1 \mid Y = 1)$ the true positive rate (TPR) of $\widehat{h}_t$ and $\alpha(t) = \Pr(\widehat{h}_t(X) = 1 \mid Y = -1)$ its false positive rate (FPR)

α and β are monotone functions of t, with $\alpha(t) = \beta(t) = 0$ if $t = 1$, and $\alpha(t) = \beta(t) = 1$ if $t = 0$

A good classifier will have a large true positive rate for a small false positive rate

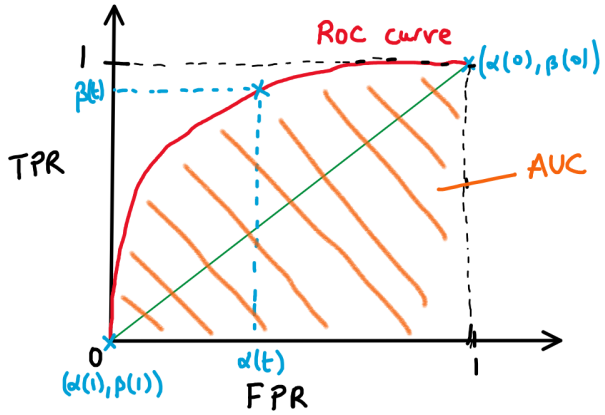
To quantify this over a range of values of t, the ROC curve plots the TPR $\beta(t)$ vs the FPR $\alpha(t)$ for t ranging from 1 to 0

Area under the ROC curve (AUC)

$$AUC = \int_1^0 \beta(t) d\alpha(t),$$

AUC measure of the performance of the family of plug-in classifiers over the whole range of loss functions.

ROC Curve and AUC



ROC Curve and AUC

In practice the population TPR and FPR cannot be computed analytical

Empirical approximation over a test set $d_{\text{test}} = (\widetilde{x}_i, \widetilde{y}_i)_{i=1,\dots,n_{\text{test}}}$

Let $\widetilde{m}_1$ and $\widetilde{m}_{-1}$ be resp. the number of positive and negatives examples in the test set, with $\widetilde{m}_1 + \widetilde{m}_{-1} = n_{\text{test}}$

For any $t \in [0, 1]$

$$\begin{cases} \widehat{\alpha}^{(d_{\text{test}})}(t) = \dfrac{\sum_{i|\widetilde{y}_i=-1} \mathbb{1}_{\widehat{\eta}(\widetilde{x}_i)\geq t}}{\widetilde{m}_{-1}} \\ \widehat{\beta}^{(d_{\text{test}})}(t) = \dfrac{\sum_{i|\widetilde{y}_i=1} \mathbb{1}_{\widehat{\eta}(\widetilde{x}_i)\geq t}}{\widetilde{m}_1} \end{cases}$$

Probabilistic interpretation of the AUC

Consider a plug-in classifier with estimated discriminant function $\widehat{\eta}(x)$, obtained from a dataset d, which is a realisation from a random sample D.

For two random variables $(\widetilde{X}_0, \widetilde{Y}_0)$ and $(\widetilde{X}_1, \widetilde{Y}_1)$, mutually independent and independent from D, with the same distribution as (X, Y), we have

$$AUC = \Pr\left(\widehat{\eta}(\widetilde{X}_1) \geq \widehat{\eta}(\widetilde{X}_0) \mid \widetilde{Y}_1 = 1, \widetilde{Y}_0 = -1, D = d\right).$$

That is the AUC is the probability that, for two independent samples from class 1 and -1, the discriminant score of the sample from class 1 is greater than the discriminant score of the sample from class -1.

Probabilistic interpretation of the AUC

Proof

The conditioning on $D = d$ is implicit in what follows.

Denote $F_{-1}(s)$ the cdf of the random variable $\widehat{\eta}(X) \in [0, 1]$ given $Y = -1$, and $F_1(s)$ the cdf of $\widehat{\eta}(X)$ given $Y = 1$:

$$\begin{cases} F_1(s) = \Pr(\widehat{\eta}(X) \leq s \mid Y = 1) \\ F_{-1}(s) = \Pr(\widehat{\eta}(X) \leq s \mid Y = -1). \end{cases}$$

Let $f_1 = F_1'$ and $f_{-1} = F_{-1}'$ be their pdf.

Probabilistic interpretation of the AUC

Proof

For a threshold $t = a/(a+b)$, the (population) true positive rate $\beta(t)$ and false positive rate $\alpha(t)$ of the plug-in classifier $\widehat{h}_t$ can be expressed as

$$\begin{cases} \beta(t) = \Pr(\widehat{\eta}(X) \geq t \mid Y = 1) = 1 - F_1(t) \\ \alpha(t) = \Pr(\widehat{\eta}(X) \geq t \mid Y = -1) = 1 - F_{-1}(t) \end{cases}$$

The AUC is

$$\begin{aligned} \int_1^0 \beta(t) d\alpha(t) &= \int_0^1 (1 - F_1(t)) f_{-1}(t) dt \\ &= \mathbb{E}\left[(1 - F_1(\widehat{\eta}(\widetilde{X}_0))) \mid \widetilde{Y}_0 = -1\right] \\ &= \mathbb{E}\left[\Pr\left(\widehat{\eta}(\widetilde{X}_1) \geq \widehat{\eta}(\widetilde{X}_0) \mid \widetilde{Y}_1 = 1\right) \mid \widetilde{Y}_0 = -1\right] \\ &= \Pr\left(\widehat{\eta}(\widetilde{X}_1) \geq \widehat{\eta}(\widetilde{X}_0) \mid \widetilde{Y}_1 = 1, \widetilde{Y}_0 = -1\right) \end{aligned}$$

where $(\widetilde{X}_0, \widetilde{Y}_0)$ and $(\widetilde{X}_1, \widetilde{Y}_1)$ are iid with the same distribution as (X, Y).

Statistical Machine Learning
Hilary Term 2021

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

Optimisation for machine learning

Let $\theta \in \mathbb{R}^p$

Consider either

- ERM with a class of prediction rules h_θ parameterised by a parameter θ or
- a conditional plug-in method, where the conditional distribution $f_\theta(y|x)$ is parameterised by θ or
- a generative method, where the joint distribution $\pi_\theta(x, y)$ is parameterised by θ.

For the plug-in approach, we assume that the parameter θ is fitted using maximum likelihood, potentially with the addition of a regularisation term.

Optimisation for machine learning

In all three cases, we aim at minimising an objective function $J : \mathbb{R}^p \to \mathbb{R}_+$ of the form

$$J(\theta) = \sum_{i=1}^{n} J_i(\theta) + J_0(\theta)$$

where J_0 is a penalisation/regularisation term and each term J_i is associated to an example (x_i, y_i):

- For empirical risk minimisation, $J_i(\theta) = \frac{1}{n} L(y_i, h_\theta(x_i))$
- For a conditional plug-in approach, $J_i(\theta) = \log(f_\theta(y_i|x_i))$
- For a generative plug-in approach, $J_i(\theta) = \log(\pi_\theta(x_i, y_i))$.

Optimisation for machine learning

In this lecture, we will use the notations of an ERM objective function but the algorithms apply equally to estimate the parameters of plug-in methods

For ERM, the objective function J is:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, h_\theta(x_i)) + \frac{\lambda}{n} \text{pen}(h_\theta).$$

Assuming it exists, the global minimum may not be available in closed-form and one has to resort to some numerical method.

Even if it is available in closed-form, computing it may be impractical for a large number n of observations and/or a large number of parameters p.

Optimisation for machine learning

For example, for empirical risk minimisation with the class of linear functions with input expansion $\phi : \mathcal{X} \to \mathbb{R}^p$, and L2 regularisation with regularisation parameter λ, the objective function is (ridge regression)

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \phi(x_i)^\intercal\beta)^2 + \frac{\lambda}{n}\|\beta\|^2.$$

Estimated parameters

$$\widehat{\beta} = (\mathbf{\Phi}^\intercal\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^\intercal\,\mathbf{y}$$

where $\mathbf{\Phi}$ is an $n \times p$ matrix, $\mathbf{y}$ is a vector of length n, and $\mathbf{\Phi}^\intercal\mathbf{\Phi} + \lambda I$ is a $p \times p$ matrix.

Number of computations of order

- $O(np^2)$ for computing $\mathbf{\Phi}^\intercal\mathbf{\Phi} + \lambda I$ and $\mathbf{\Phi}^\intercal\,\mathbf{y}$.
- $O(p^3)$ (using e.g. Gauss-Jordan elimination) for solving the linear system

$$(\mathbf{\Phi}^\intercal\mathbf{\Phi} + \lambda I)\beta = \mathbf{\Phi}^\intercal\,\mathbf{y}\,.$$

Prohibitive for large p!

Optimisation for machine learning

For other methods, such as logistic regression or neural networks, there is no closed-form solution

Need to resort to numerical methods that will scale well with both the dimension p of the inputs and the number n of data examples

Two algorithms in this lecture

- Gradient Descent
- Stochastic Gradient Descent

Notations

For a vector $\theta = (\theta_1, \ldots, \theta_p)^\intercal$, a twice differentiable function $J : \mathbb{R}^p \to \mathbb{R}_+$, let $\nabla_\theta J(\theta)$ be the gradient of J with respect to θ, and $\nabla^2_\theta J(\theta)$ be the Hessian of J with respect to θ, defined as

$$\nabla_\theta J(\theta) = \begin{pmatrix} \frac{\partial J(\theta)}{d\theta_1} \\ \cdots \\ \frac{\partial J(\theta)}{d\theta_p} \end{pmatrix}, \quad \nabla^2_\theta J(\theta) = \begin{pmatrix} \frac{\partial^2 J(\theta)}{d\theta_1 d\theta_1} & \cdots & \frac{\partial^2 J(\theta)}{d\theta_1 d\theta_p} \\ \cdots & \cdots & \cdots \\ \frac{\partial^2 J(\theta)}{d\theta_p d\theta_1} & \cdots & \frac{\partial^2 J(\theta)}{d\theta_p d\theta_p} \end{pmatrix}.$$

A symmetric real-valued p-by-p matrix H is said to be positive semi-definite iff

$$z^\intercal H z \geq 0$$

for all $z \in \mathbb{R}^p$.

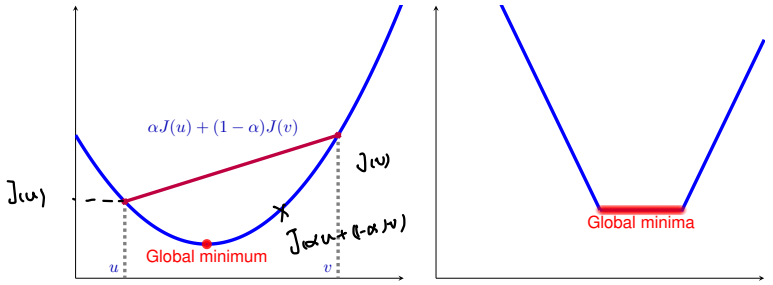
We use the notation $H \succeq 0$ if H is positive semi-definite.

Convex functions

Definition (Convex function)

A function $J : \mathbb{R}^p \to \mathbb{R}$ is said to be convex if, for all $u, v \in \mathbb{R}^p$, $\alpha \in [0, 1]$,

$$J(\alpha u + (1-\alpha)v) \leq \alpha J(u) + (1-\alpha)J(v)$$



Convex functions

Some examples of convex functions:

- Univariate: θ^2, $\exp(-\theta)$, $\log(1+\exp(-\theta))$, $\max(0, 1-\theta)$
- Affine functions: $A\theta + b$
- Quadratic functions: $\theta^\intercal H\theta$ where $H \succeq 0$

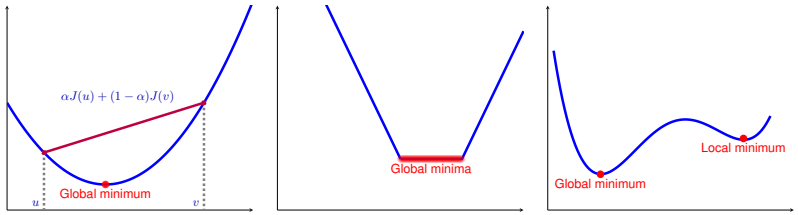
Convex functions

Key property of convex functions

Proposition

If the function J is convex, all local minima are also global minima.

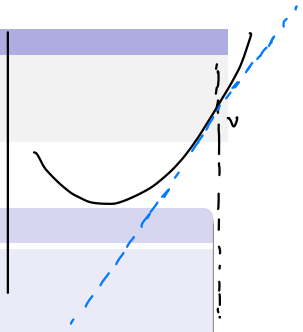
Convergence to a local minimum guarantees convergence to a global minimum



Convex Convex Non-Convex

Convex functions

First order characterisation



Definition (Differentiable convex function)

A differentiable function J is convex iff, for any u, v

$$J(u) \geq J(v) + \nabla J(v)^\intercal (u - v)$$

Proposition

Let J be a differentiable convex function. Any stationary point u of J, that is such that

$$\nabla_u J(u) = 0$$

is also a global minimum.

Convex functions

Second order characterisation

Definition (Twice differentiable convex function)

A twice differentiable function J is convex iff

$$\nabla_u^2 J(u) \succeq 0$$

for all $u \in \mathbb{R}^p$.

Convex functions

Convex functions can be combined in a number of ways:

Nonnegative linear combination of convex functions is convex. Let J_1 and J_2 be convex functions. Then

$$J = \alpha_1 J_1 + \alpha_2 J_2$$

is also convex for any $\alpha_1, \alpha_2 \geq 0$.

Affine composition of convex functions is convex. If g is convex, then $J(\theta) = g(A\theta + b)$ is convex.

If for any (x_i, y_i), the function $J_i : \theta \to L(y_i, h_\theta(x_i))$ is a convex function (of θ) and the regularisation function pen is convex, then the objective function J is also a convex function, and all local minima are global minima.

Objective functions of ridge regression, logistic regression, support vector machines are convex

Note that compositions of convex functions are NOT convex in general (in particular, the objective function of deep neural networks is non-convex)

Convex functions

The ridge regression objective function

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \phi(x_i)^\intercal \beta)^2 + \frac{\lambda}{n}\|\beta\|^2$$

is convex.

The gradient and Hessian are

$$\begin{cases} \nabla_\beta J(\beta) = \frac{1}{n}\left(2\mathbf{\Phi}^\intercal\mathbf{\Phi}\beta - 2\Phi^\intercal \mathbf{y} + 2\lambda\beta\right) \\ \nabla^2_\beta J(\beta) = \frac{2}{n}\left(\mathbf{\Phi}^\intercal\mathbf{\Phi} + \lambda I\right). \end{cases}$$

The Hessian does not depend on the parameter β, and is positive semi-definite for any $\lambda \geq 0$.

Gradient Descent

Objective function

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, h_\theta(x_i)) + \frac{\lambda}{n}\mathrm{pen}(h_\theta).$$

Gradient

$$\nabla_\theta J(\theta) = \frac{1}{n}\left(\sum_{i=1}^{n} \nabla_\theta L(y_i, h_\theta(x_i)) + \lambda \nabla_\theta \mathrm{pen}(h_\theta)\right)$$

Let $\eta > 0$ be the learning rate or step-size

Algorithm 1 Gradient Descent

Initialise $\theta^{(0)}$ and set $t = 0$;

Repeat until convergence

Compute the gradient $\nabla_\theta J(\theta^{(t)})$

Update the parameters

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

Set $t \leftarrow t + 1$

Gradient Descent

The negative gradient is going in the direction that decreases the function J

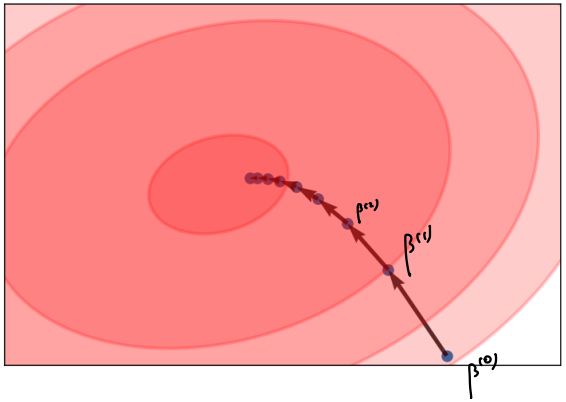
Large values of the gradient will induce large changes in the value of the parameter

As the parameter becomes closer to a stationary point of J, the gradient takes smaller values, and the parameter changes more slowly

For convex functions (under some additional assumptions omitted here), the algorithm will stop at a point close to the minimum solution, , independent of the starting point

For non-convex functions, the algorithm will attain different local minima depending on the initialisation.

Gradient Descent



Gradient Descent

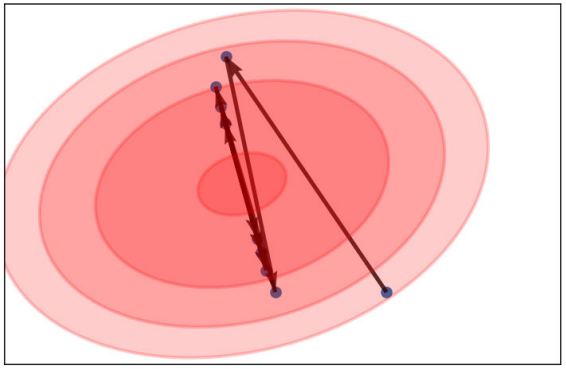
The parameter η is known as the step size, or learning rate.

Its value is critical to the good behaviour of the gradient descent algorithm.

Too large values may lead the algorithm to have a zigzaguing behaviour, or even to diverge, whereas too small values will lead to slow convergence.

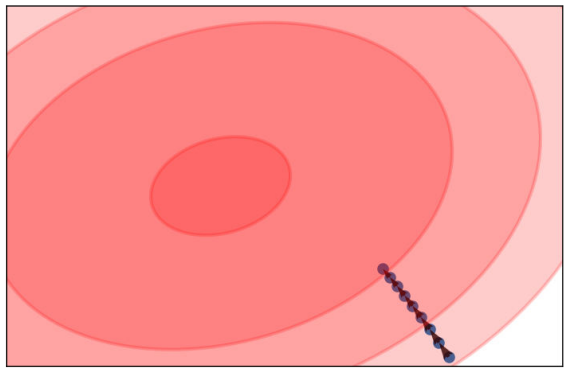
This parameter can also be updated adaptively.

Gradient Descent



η too large

Gradient Descent



η too small

Gradient Descent

Interpretation of gradient descent via a quadratic approximation, and Newton-Raphson algorithm

Assume that J is twice differentiable.
Second-order Taylor expansion of J around $\theta \in \mathbb{R}^p$

$$J(\theta + \delta) \simeq J(\theta) + \nabla_\theta J(\theta)^\intercal \delta + \frac{1}{2}\delta^\intercal \nabla^2_\theta J(\theta)\delta.$$

Replacing the Hessian matrix $\nabla^2_\theta J(\theta)$ with the diagonal p-by-p matrix $\frac{1}{\eta}I$ gives the approximation

$$J(\theta + \delta) \simeq J(\theta) + \nabla_\theta J(\theta)^\intercal \delta + \frac{1}{2\eta}\|\delta\|^2.$$

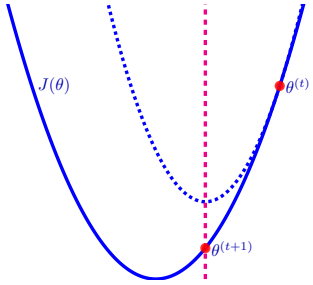
At θ, we use the quadratic function as an approximation to $J(\theta + \delta)$.
Taking the derivative with respect to δ, we obtain

$$\nabla_\theta J(\theta) + \frac{1}{\eta}\delta = 0,$$

Minimum is achieved for $\delta = -\eta \nabla_\theta J(\theta)$, which corresponds to the gradient descent update.

Gradient Descent

Interpretation of gradient descent via a quadratic approximation, and Newton-Raphson algorithm



Gradient Descent

Interpretation of gradient descent via a quadratic approximation, and Newton-Raphson algorithm

Second-order Taylor expansion of J around $\theta \in \mathbb{R}^p$

$$J(\theta + \delta) \simeq J(\theta) + \nabla_\theta J(\theta)^\intercal \delta + \frac{1}{2}\delta^\intercal \nabla^2_\theta J(\theta)\delta.$$

Instead of approximating the Hessian, use the second-order approximation.
Differentiating with respect to δ and setting to 0, we obtain

$$\nabla_\theta J(\theta) + \nabla^2_\theta J(\theta)\delta = 0$$

hence a minimum at

$$\delta = -(\nabla^2_\theta J(\theta))^{-1}\nabla_\theta J(\theta).$$

The corresponding update is therefore

$$\theta^{(t+1)} = \theta^{(t)} - (\nabla^2_\theta J(\theta^{(t)})^{-1}\nabla_\theta J(\theta^{(t)})$$

Iterative algorithm using this update is known as the Newton-Raphson gradient descent algorithm.
The Hessian is a p-by-p matrix, which may be computationally too expensive to compute for large p.

Algorithm 2 Newton-Raphson algorithm

Initialise $\theta^{(0)}$ and set $t = 0$;

Repeat until convergence

Update the parameters

$$\theta^{(t+1)} = \theta^{(t)} - (\nabla^2_\theta J(\theta^{(t)}))^{-1} \nabla_\theta J(\theta^{(t)})$$

Set $t \leftarrow t + 1$

Gradient Descent for linear regression

Objective function

$$J(\beta) = \frac{1}{n}(\mathbf{y} - \Phi\beta)^\intercal(\mathbf{y} - \Phi\beta).$$

Gradient

$$\nabla_\beta J(\beta) = \frac{2}{n}\Phi^\intercal(\Phi\beta - \mathbf{y})$$

Parameter update

$$\beta^{(t+1)} = \beta^{(t)} - \frac{2\eta}{n}\Phi^\intercal(\Phi\beta^{(t)} - \mathbf{y}).$$

The initial cost of gradient descent is $O(np^2)$ to compute $\Phi^\intercal\Phi$.

Then each iteration has a computational cost of $O(np)$.

If n and T, the total number of gradient descent steps, are both small compared to the number of parameters p, the overall computational cost of the gradient descent algorithm is lower than that of solving the linear system ($O(p^3)$)

Gradient Descent for linear regression

Input/feature normalisation

Gradient step

$$\begin{aligned}\beta^{(t+1)} &= \beta^{(t)} - \frac{2\eta}{n}(\Phi^\intercal\Phi\beta^{(t)} - \Phi^\intercal\mathbf{y}) \\ &= \beta^{(t)} + \frac{2\eta}{n}\Phi^\intercal\Phi(\widehat{\beta} - \beta^{(t)})\end{aligned}$$

where

$$\widehat{\beta} = (\Phi^\intercal\Phi)^{-1}\Phi^\intercal\mathbf{y}$$

is the minimiser of J.

If $\Phi^\intercal\Phi = c \times I$ for some constant $c > 0$, then a gradient descent update will go in the direction of $\widehat{\beta} - \beta^{(t)}$, and will converge quickly to $\widehat{\beta}$.

Otherwise, the update will have a zigzaguing behaviour.

A way to have a better conditioned problem is to apply some transformation to the data.

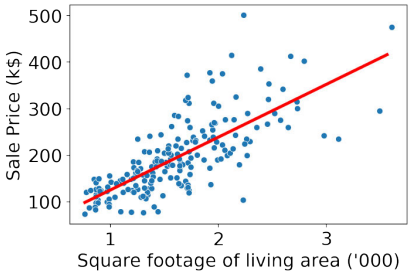
Standardising the different dimensions of the transformed inputs $\phi(x_i)$ so that they have zero mean and unit variance for example leads to a more spherical $\Phi^\intercal\Phi$ and a faster convergence of the GD algorithm.

Gradient Descent for linear regression

Example on house price dataset

Consider the house prices dataset ($n = 200$, $p = 1$)

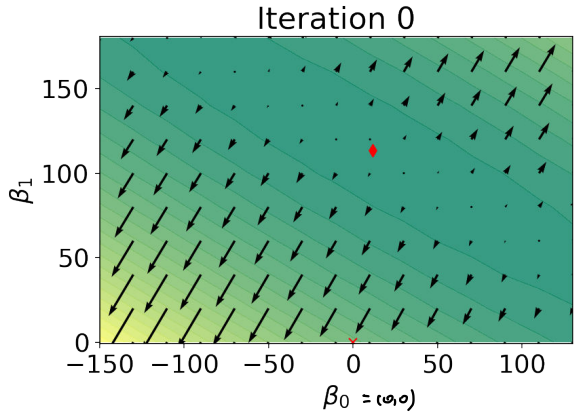
Ordinary linear regression



Apply gradient descent without standardising the data

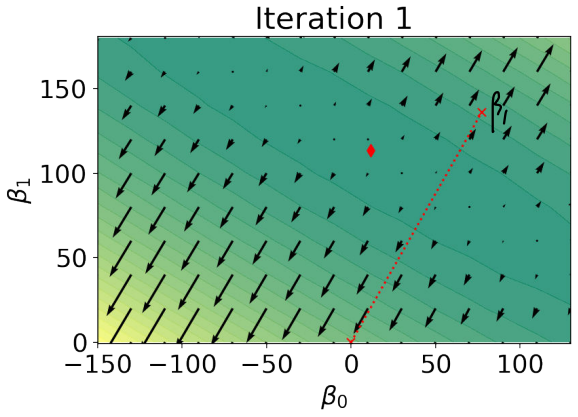
Gradient Descent for linear regression

Example on house price dataset



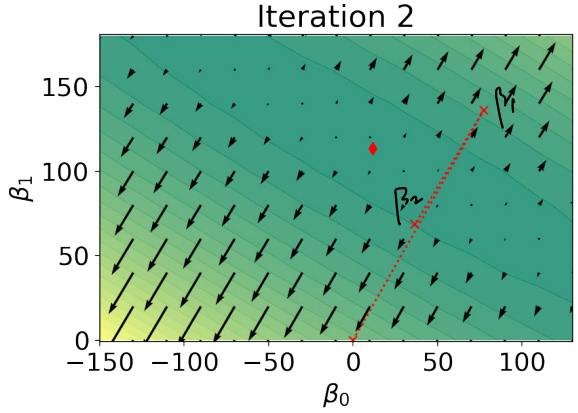
Gradient Descent for linear regression

Example on house price dataset



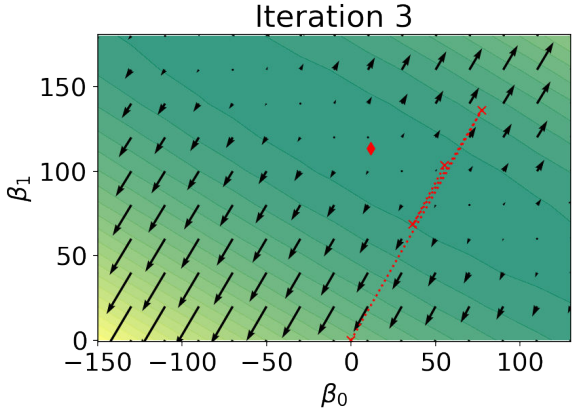
Gradient Descent for linear regression

Example on house price dataset



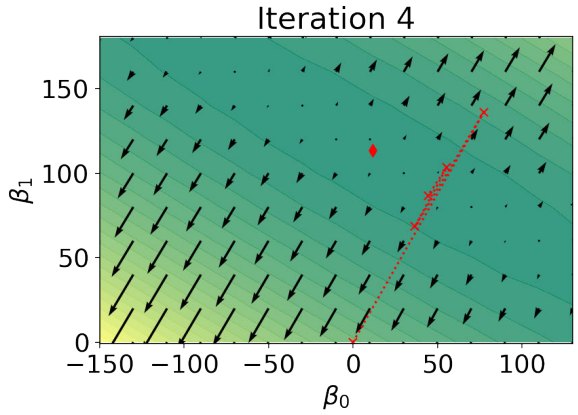
Gradient Descent for linear regression

Example on house price dataset



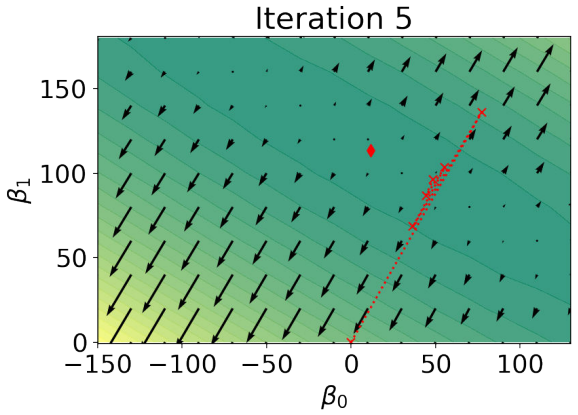
Gradient Descent for linear regression

Example on house price dataset



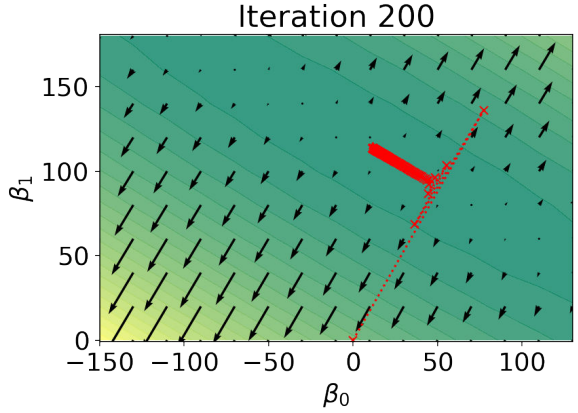
Gradient Descent for linear regression

Example on house price dataset



Gradient Descent for linear regression

Example on house price dataset



Gradient Descent for linear regression

Example on house price dataset

$\phi(x_i) = \begin{pmatrix} 1 \\ x_i \end{pmatrix}$ $\quad \phi(x_i) = \begin{pmatrix} 1 \\ \frac{x_i - \bar{x}}{\sigma} \end{pmatrix}$

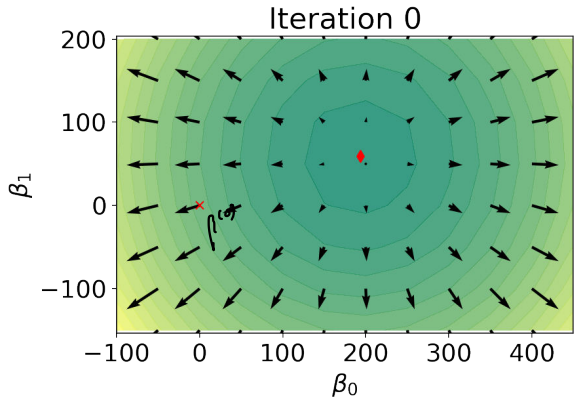
$\bar{x}$ sample mean

σ s.d.

Standardise the inputs x_i so that they have zero mean and unit variance

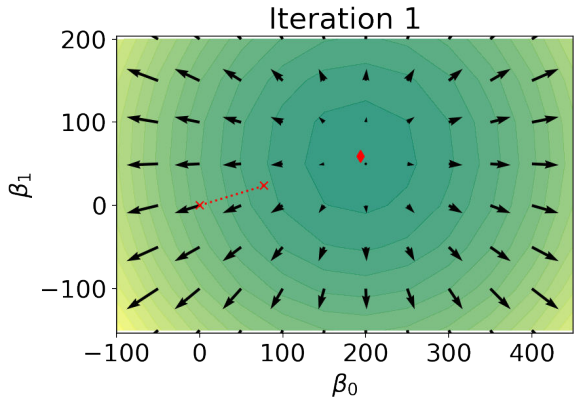
Gradient Descent for linear regression

Example on house price dataset



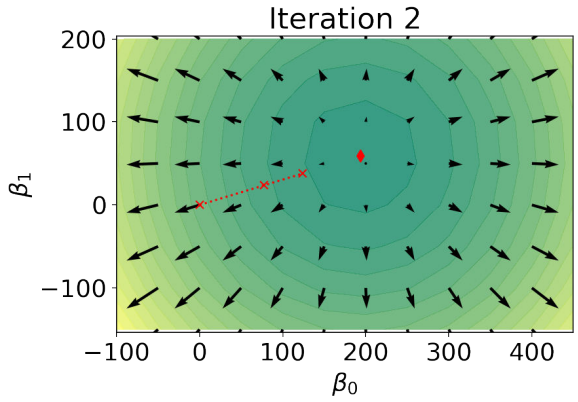
Gradient Descent for linear regression

Example on house price dataset



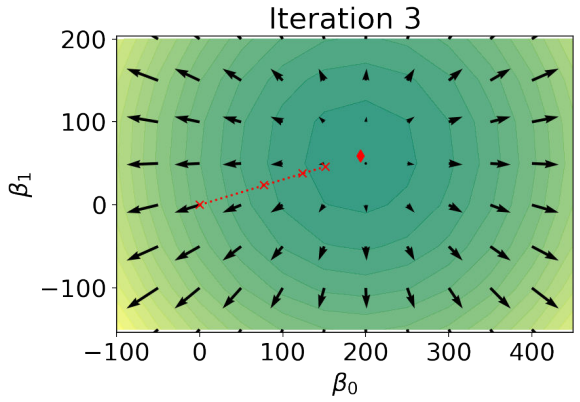
Gradient Descent for linear regression

Example on house price dataset



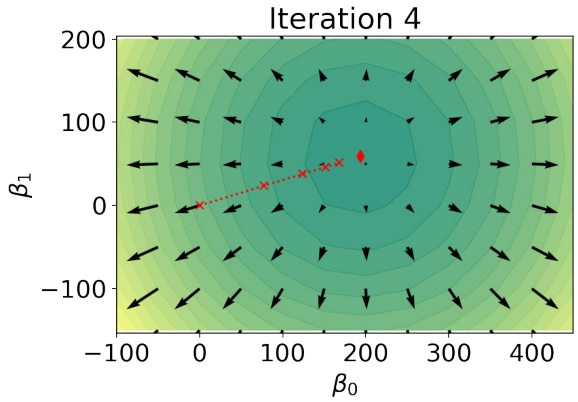
Gradient Descent for linear regression

Example on house price dataset



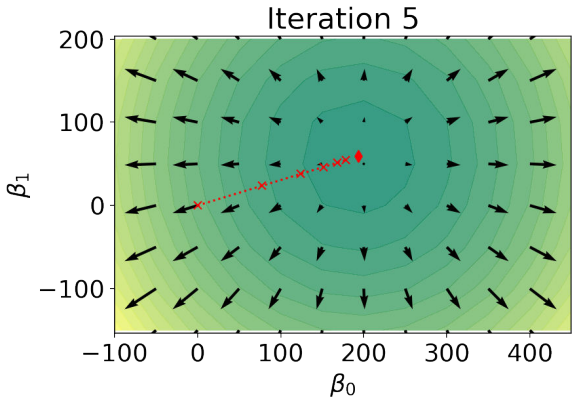
Gradient Descent for linear regression

Example on house price dataset



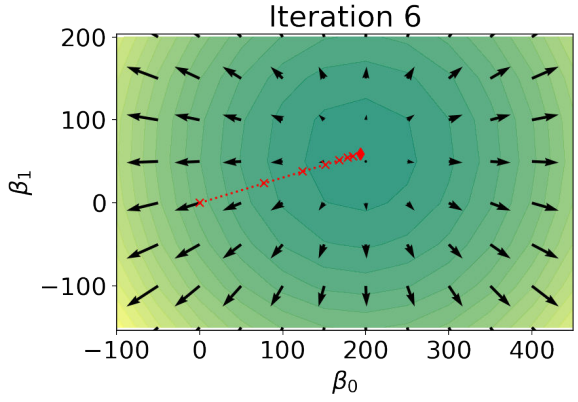
Gradient Descent for linear regression

Example on house price dataset



Gradient Descent for linear regression

Example on house price dataset



Stochastic Gradient Descent

Stochastic gradient descent (and its variants) is certainly the most popular algorithm for large-scale (large n and/or p) machine learning.

Default algorithm to learn deep learning models.

Gradient descent requires to evaluate the gradient

$$\nabla_\theta J(\theta) = \frac{1}{n}\left(\sum_{i=1}^{n} \nabla_\theta L(y_i, h_\theta(x_i)) + \lambda \nabla_\theta \operatorname{pen}(h_\theta)\right)$$

at each iteration, which typically scales linearly in the number n of observations.

Prohibitive for very large datasets.

Stochastic gradient descent replaces the gradient by an unbiased estimator of the gradient, obtained by using a subset of the data (called a mini-batch) at each iteration.

Stochastic Gradient Descent

Let $(\eta_t)_{t=0,1,\ldots}$ be a monotone decreasing sequence, and $1 \leq n_b \leq n$ be the mini-batch size.

Algorithm 3 Stochastic Gradient Descent

Initialise $\theta^{(0)}$ and set $t = 0$;

Repeat until convergence

Randomly select n_b observations from the dataset d; denote them $(\widetilde{x}_i^{(t)}, \widetilde{y}_i^{(t)})_{i=1,\ldots,n_b}$

Compute the gradient estimate $\nabla_\theta \widetilde{J}^{(t)}(\theta^{(t)})$ where

$$\nabla_\theta \widetilde{J}^{(t)}(\theta) = \frac{1}{n_b} \sum_{i=1}^{n_b} \nabla_\theta L(\widetilde{y}_i^{(t)}, h_\theta(\widetilde{x}_i^{(t)})) + \frac{\lambda}{n} \nabla_\theta \operatorname{pen}(h_\theta)$$

Update the parameters

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla_\theta \widetilde{J}^{(t)}(\theta^{(t)})$$

Set $t \leftarrow t + 1$

Stochastic Gradient Descent

For any θ, the mini-batch gradient $\nabla_\theta \widetilde{J}^{(t)}(\theta)$ is an unbiased estimator of the true gradient $\nabla_\theta J(\theta)$.

Proof.

Note that, for any $1 \leq i \leq n_b$, $(\widetilde{X}_i^{(t)}, \widetilde{Y}_i^{(t)})$ is (marginally) a uniform discrete random variable taking values in the set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. Hence, for any $1 \leq i \leq n_b$ and $1 \leq j \leq n$,

$$\Pr((\widetilde{X}_i^{(t)}, \widetilde{Y}_i^{(t)}) = (x_j, y_j)) = \frac{1}{n}.$$

It follows that, for any $1 \leq i \leq n_b$,

$$\mathbb{E}\left[L(\widetilde{Y}_i^{(t)}, h_\theta(\widetilde{X}_i^{(t)}))\right] = \frac{1}{n} \sum_{j=1}^{n} L(y_j, h_\theta(x_j)) = \widehat{R}_n(h_\theta)$$

and

$$\begin{aligned}
\mathbb{E}[\nabla_\theta \widetilde{J}^{(t)}(\theta)] &= \frac{1}{n_b} \sum_{i=1}^{n_b} \mathbb{E}\, \nabla_\theta L(\widetilde{Y}_i^{(t)}, h_\theta(\widetilde{X}_i^{(t)})) + \frac{\lambda}{n} \nabla_\theta \operatorname{pen}(h_\theta) \\
&= \frac{1}{n_b} \sum_{i=1}^{n_b} \nabla_\theta \mathbb{E}\, L(\widetilde{Y}_i^{(t)}, h_\theta(\widetilde{X}_i^{(t)})) + \frac{\lambda}{n} \nabla_\theta \operatorname{pen}(h_\theta) \\
&= \nabla_\theta \widehat{R}_n(h_\theta) + \frac{\lambda}{n} \nabla_\theta \operatorname{pen}(h_\theta) = \nabla_\theta J(\theta).
\end{aligned}$$

□

Stochastic Gradient Descent

For SGD to converge, the learning rate η_t should be such that $\eta_t \to 0$.
A standard parametrisation is

$$\eta_t = \frac{\eta_0}{(1 + t/t_0)^a}$$

where $\eta_0 > 0$, $t_0 > 0$ and $1/2 < a \leq 1$ are tuning parameters.
Alternatively, the learning rate may be tuned adaptively.

Stochastic Gradient Descent

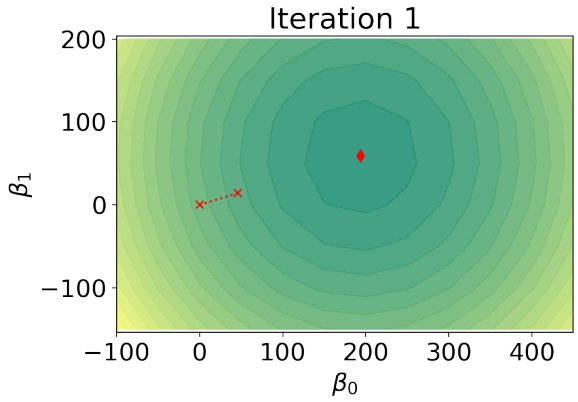
On large datasets, SGD typically converges faster than GD

Batch size: larger batch sizes lead to a better estimate of the gradient, but are more costly to compute

Randomness of the algorithm allows to escape local minima more easily

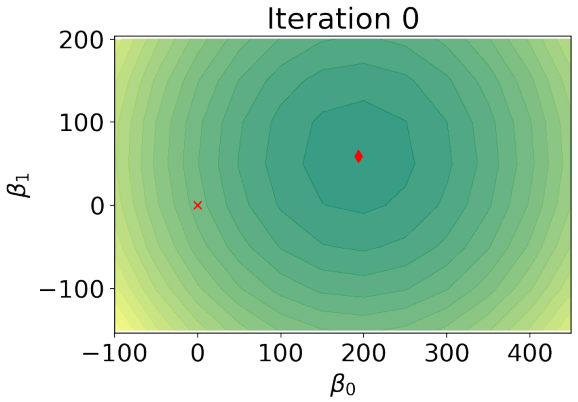
Stochastic Gradient Descent for linear regression

Example on house price dataset



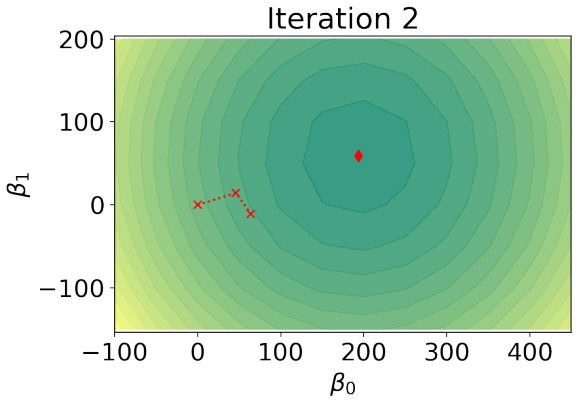
Stochastic Gradient Descent for linear regression

Example on house price dataset



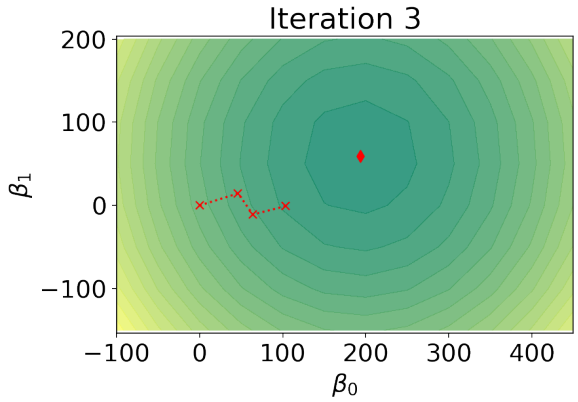
Stochastic Gradient Descent for linear regression

Example on house price dataset



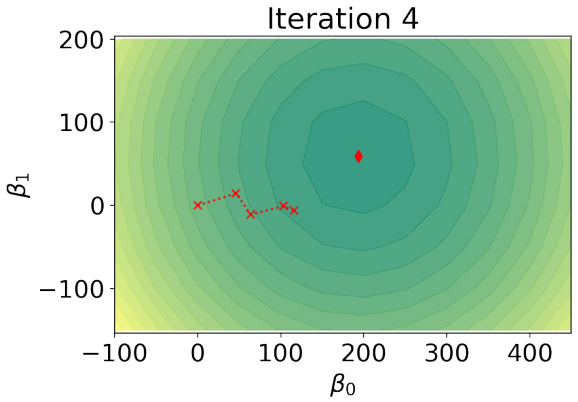
Stochastic Gradient Descent for linear regression

Example on house price dataset



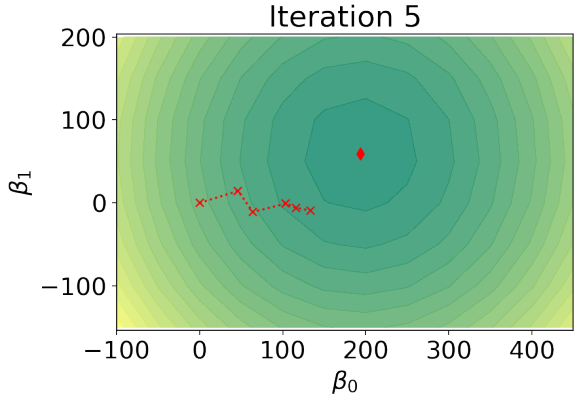
Stochastic Gradient Descent for linear regression

Example on house price dataset



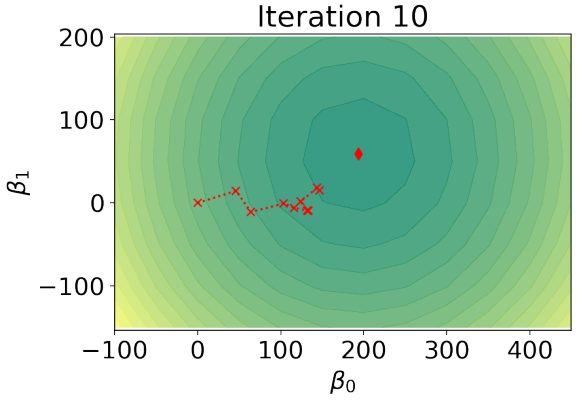
Stochastic Gradient Descent for linear regression

Example on house price dataset



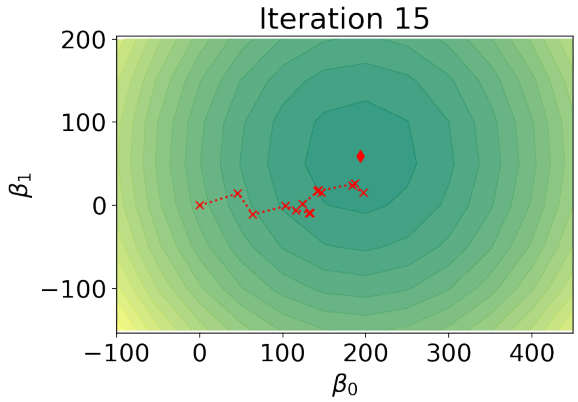
Stochastic Gradient Descent for linear regression

Example on house price dataset



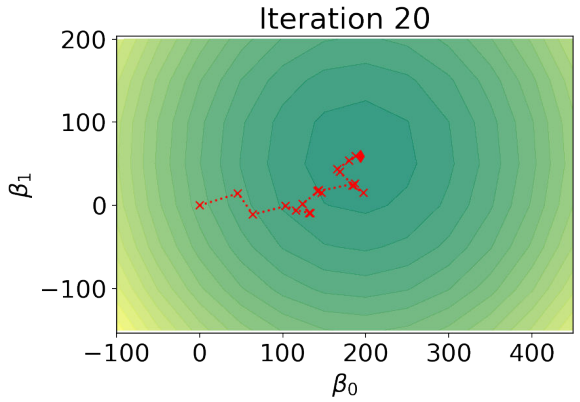
Stochastic Gradient Descent for linear regression

Example on house price dataset



Stochastic Gradient Descent for linear regression

Example on house price dataset



Early stopping and implicit regularisation

Consider using (stochastic) gradient descent to minimise the empirical risk $\widehat{R}_n(h_\theta)$ over a large class $\mathcal{H}$.

Without regularisation, the ERM will have a large estimation error, hence a large risk

Early stopping is a strategy that allows to implicitly regularise by stopping the algorithm before it reaches convergence.

Early stopping and implicit regularisation

Assume we start with an initial prediction rule $h_{\theta^{(0)}}$ with small complexity (hence large bias)

For example, take $\theta^{(0)} = 0$

Each step of the (stochastic) gradient descent tends to increase the complexity of the prediction rule, hence to decrease the bias and increase the variance

For small number of iterations t the bias part dominates, while for large t, the variance part dominates

One can find the optimal number of steps of the algorithm using a validation set

For each iteration $t = 0, 1, \ldots, T$ of the algorithm, we approximate the risk on the validation set, and return the value $\theta^{(t)}$ that minimises the validation risk.

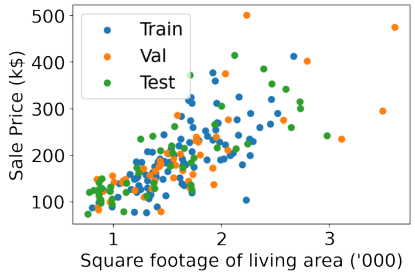
Early stopping and implicit regularisation

House price dataset

We use gradient descent to find the ERM for a linear model with polynomial expansion of order 12 (hence $\beta \in \mathbb{R}^{13}$)

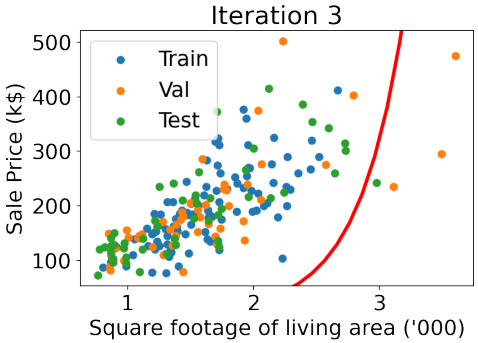
Start with $\beta^{(0)} = 0$

Split the dataset into a training/validation/test set, and approximate the population risk at each iteration using the validation set



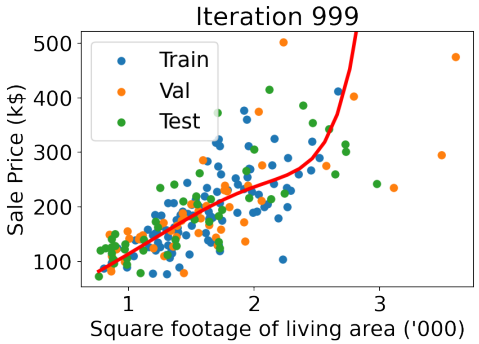
Early stopping and implicit regularisation

House price dataset



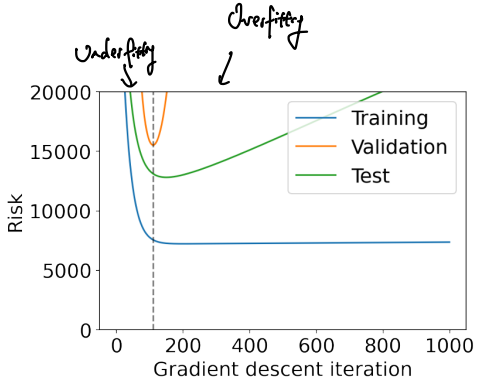
Early stopping and implicit regularisation

House price dataset



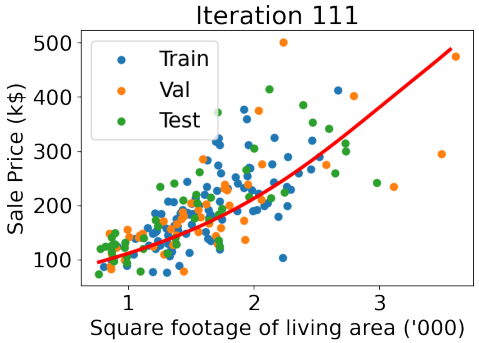
Early stopping and implicit regularisation

House price dataset



Early stopping and implicit regularisation

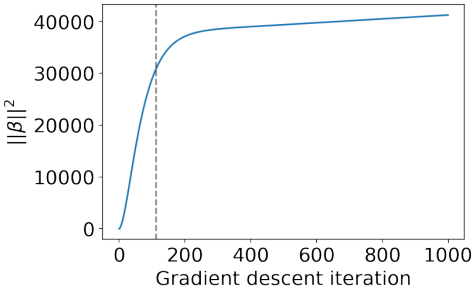
House price dataset



Early stopping in (stochastic) gradient descent and implicit regularisation

Early stopping and implicit regularisation

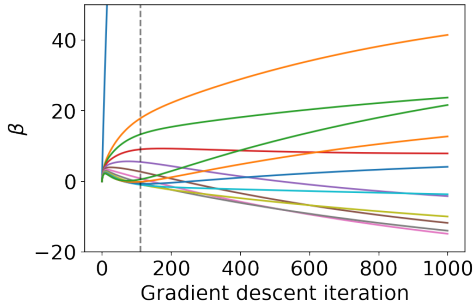
House price dataset



Early stopping in (stochastic) gradient descent and implicit regularisation

Early stopping and implicit regularisation

House price dataset



Statistical Machine Learning
Hilary Term 2021

22/2

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

Linear binary classifiers

Binary classification, with $Y \in \{-1, 1\}$

Prediction rule can be expressed in terms of a discriminant function $f : \mathcal{X} \to \mathbb{R}$, such that

$$h(x) = \begin{cases} 1 & \text{if } f(x) \geq 0 \\ -1 & \text{if } f(x) < 0 \end{cases}$$

Let $\phi : \mathcal{X} \to \mathbb{R}^p$ be some input expansion function

Let $\mathcal{H}$ be the set of classifiers h with a linear discriminant function on the transformed input $\phi(x)$

$$f(x) = \phi(x)^\intercal \beta$$

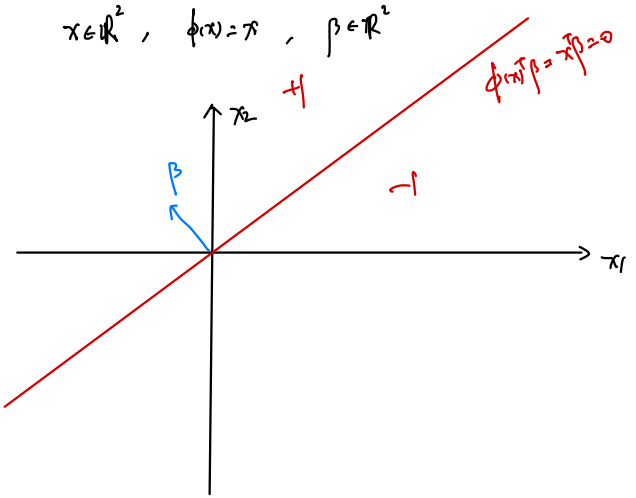
where f is parameterised by the vector $\beta \in \mathbb{R}^p$

Denote $\mathcal{H}_f = \{f : f(x) = \phi(x)^\intercal \beta\}$ the set of linear discriminant functions on the transformed inputs $\phi(x)$

f is a nonlinear function of x if ϕ is nonlinear

Linear binary classifiers

Plot



Linear binary classifiers

We have already seen a generative linear classifier: Linear discriminant analysis

This lecture: three other linear classifiers

- Least-squares
- Perceptron
- Logistic Regression

Linear binary classifiers

Natural estimator for β: Empirical risk minimiser under a 0-1 loss

$$\widehat{\beta} = \arg\min_{\beta} \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{y_i \neq \phi(x_i)^\intercal \beta}$$

As we will see, intractable for moderate/large n

Intractability of ERM under the 0-1 loss

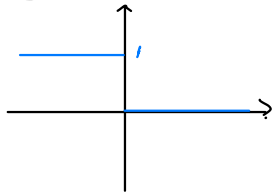
The 0-1 loss for binary classification is defined as

$$L(y, h(x)) = \mathbb{1}_{y \neq h(x)} = \mathbb{1}_{\text{sign}(yf(x))=-1} = \psi_{0-1}(yf(x))$$

where $\psi_{0-1} : \mathbb{R} \to \{0, 1\}$ is such that $\psi_{0-1}(z) = \mathbb{1}_{\text{sign}(z)=-1}$.

Under ERM, $\widehat{f}$ minimises the empirical risk

$$\frac{1}{n} \sum_{i=1}^{n} \psi_{0-1}(y_i f(x_i))$$



over some class of functions (e.g. the class of linear functions).

Intractability of ERM under the 0-1 loss

Objective function

$$\frac{1}{n} \sum_{i=1}^{n} \psi_{0-1}(y_i f(x_i))$$

The function ψ_{0-1} is piecewise constant, non-convex, with a discontinuity at 0.

Objective function cannot be minimised in closed-form, is piecewise constant, non-convex and discontinuous

Challenging optimisation problem, prevents the use of gradient-based algorithms

To resolve this issue, we typically use an alternative loss function, called a surrogate loss function, which will make the objective function easier to minimise.

Surrogate loss

Definition

A surrogate loss function is a continuous function such that

1. $\psi(x) \geq \psi_{0-1}(x)$ for all x
2. ψ is a convex function.

Surrogate loss

The discriminant function of the ERM under the surrogate loss function ψ is

$$\widehat{f} = \arg\min_{f \in \mathcal{H}_f} \frac{1}{n} \sum_{i=1}^{n} \psi(y_i f(x_i)).$$

If $f(x) = \phi(x)^\intercal \beta$, this leads to the estimate

$$\widehat{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \frac{1}{n} \sum_{i=1}^{n} \psi(y_i \phi(x_i)^\intercal \beta)$$

As ψ is convex, this is now a convex optimisation problem

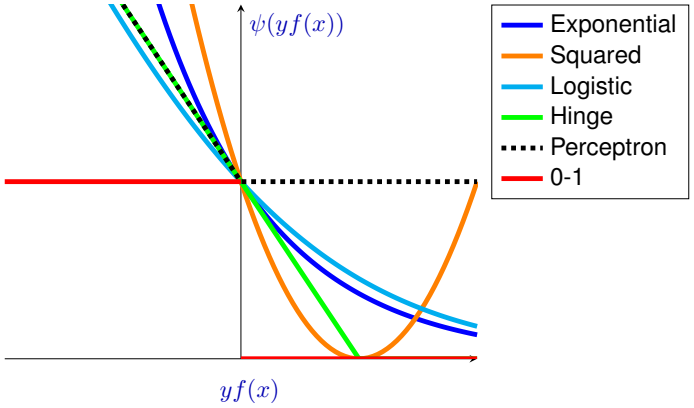
Surrogate loss

Standard surrogate loss functions

- Exponential: $\psi(z) = e^{-z}$
- Squared: $\psi(z) = (1-z)^2$
- Logistic: $\psi(z) = \log(1+e^{-z})/\log(2)$
- Perceptron: $\psi(z) = 1 + \max(0, -z)$
- Hinge: $\psi(z) = \max(0, 1-z)$

→ an upper bound for 0-1 loss

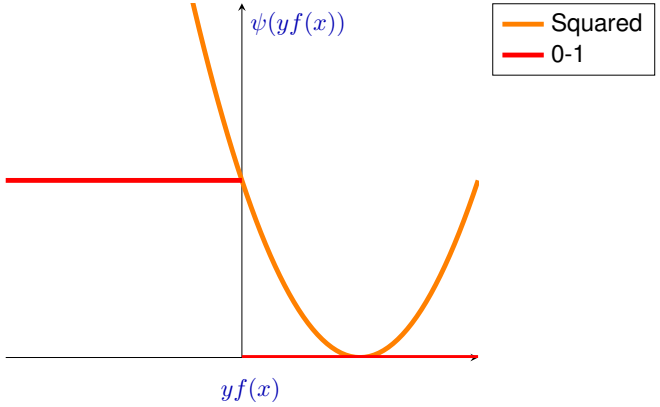
Surrogate loss



Least-squares classifier

Consider the squared surrogate loss function

$$\psi(z) = (1-z)^2. \quad (1)$$



Least-squares classifier

The ERM under this surrogate loss and the class of linear classifiers $\mathcal{H}$ is given by

$$\begin{aligned}\widehat{\beta} &= \arg\min_{\beta\in\mathbb{R}^p} \frac{1}{n}\sum_{i=1}^{n}(1 - y_i\phi(x_i)^\intercal\beta)^2 \\ &= \arg\min_{\beta\in\mathbb{R}^p} \frac{1}{n}\sum_{i=1}^{n} y_i^2(1 - y_i\phi(x_i)^\intercal\beta)^2 \\ &= \arg\min_{\beta\in\mathbb{R}^p} \frac{1}{n}\sum_{i=1}^{n}(y_i - \phi(x_i)^\intercal\beta)^2\end{aligned}$$

and $\widehat{\beta}$ is therefore the typical least square estimate

$$\widehat{\beta} = (\Phi^\intercal\Phi)^{-1}\Phi^\intercal\mathbf{y}.$$

Least-squares classifier

Advantages: Closed-form solution

Drawback: Poor proxy for the 0-1 loss; penalises large positive values (well classified points far from the boundary)

When the number of examples in each classes is the same, the least-square binary classifier is the same as the binary LDA classifier (see Problem Sheets)

Perceptron

The perceptron algorithm is a classic machine learning algorithm introduced in 1962 by Rosenblatt.

Surrogate perceptron loss

$$\psi(z) = 1 + \max(0, -z).$$

Objective function

$$J(\beta) = 1 + \frac{1}{n}\sum_{i=1}^{n}\max(0, -y_i\phi(x_i)^\intercal\beta).$$

Minimisation cannot be solved in closed form.

Classic perceptron algorithm resorts to a stochastic gradient descent type algorithm with a mini-batch size of 1, and a fixed step size η

Perceptron

Algorithm 1 Perceptron algorithm

Initialise $\beta^{(0)}$ and set $t = 0$;

For $t = 1, \ldots, n$

If $y_t\phi(x_t)^\intercal\beta^{(t)} \geq 0$, set $\beta^{(t+1)} = \beta^{(t)}$;
otherwise, set

$$\beta^{(t+1)} = \beta^{(t)} + \eta y_t\phi(x_t)$$

Note that instead of randomly selecting a batch observation, it just runs over the dataset.

The algorithm will converge if the transformed inputs are linearly separable (that is, there exists a function $h \in \mathcal{H}$ such that $\widehat{R}_n(h) = 0$)

Logistic regression as a plug-in classifier

Logistic regression is another linear classifier
Like standard linear regression, this classifier can either be interpreted as

a conditional plug-in classifier, or
an empirical risk minimiser with a given surrogate function.

Logistic regression as a plug-in classifier

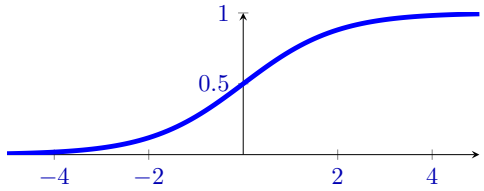
Assume that the conditional distribution of Y given $X = x$ takes the form

$$\Pr(Y = 1 \mid X = x) = \mathrm{sig}(\phi(x)^\intercal \beta)$$

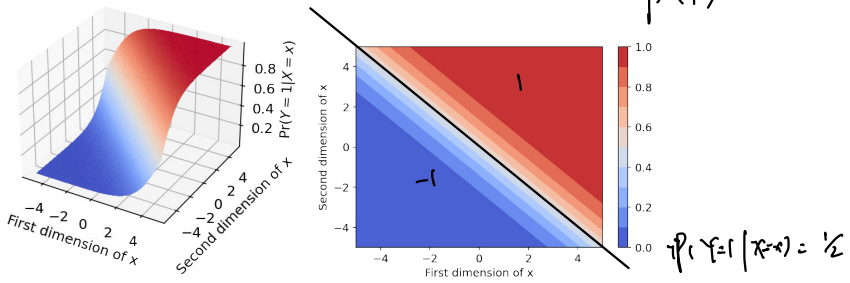
and $\Pr(Y = -1 \mid X = x) = 1 - \Pr(Y = 1 \mid X = x) = \mathrm{sig}(-\phi(x)^\intercal \beta)$, where $\beta \in \mathbb{R}^p$ is an unknown parameter and

$$\mathrm{sig}(x) = \frac{1}{1 + e^{-x}}$$

is the sigmoid function.



Logistic regression as a plug-in classifier



Logistic regression as a plug-in classifier

We have

$$\log \frac{\Pr(Y = 1 \mid X = x)}{\Pr(Y = -1 \mid X = x)} = \phi(x)^\intercal \beta$$

Bayes classifier under this model is linear and can be written as

$$h^*(x) = \begin{cases} 1 & \text{if } f(x) \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

where $f(x) = \phi(x)^\intercal \beta \geq 0$ is the linear discriminant function.
The (linear) plug-in classifier is therefore given by

$$\widehat{h}(x) = \begin{cases} 1 & \text{if } \phi(x)^\intercal \widehat{\beta} \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

where $\widehat{\beta}$ is some estimate of the parameter β.

Logistic regression as a plug-in classifier

Log-likelihood is

$$\ell(\beta) = \sum_{i=1}^{n} \log \operatorname{sig}(y_i \phi(x_i)^\intercal \beta)$$
$$= -\sum_{i=1}^{n} \log(1 + e^{-y_i \phi(x_i)^\intercal \beta})$$

MLE

$$\widehat{\beta} = \arg\max_{\beta \in \mathbb{R}^p} -\sum_{i=1}^{n} \log(1 + e^{-y_i \phi(x_i)^\intercal \beta}). \quad (2)$$

Logistic regression as ERM under a surrogate loss function

The MLE can be written as

$$\widehat{\beta} = \arg\min_{\beta \in \mathbb{R}^p} \frac{1}{n \log(2)} \sum_{i=1}^{n} \log(1 + e^{-y_i \phi(x_i)^\intercal \beta})$$

hence it can also be interpreted as the ERM under the surrogate logistic loss

$$\psi(z) = -\log(\operatorname{sig}(z))/\log(2) = \log(1 + e^{-z})/\log(2)$$

amongst the class $\mathcal{H}$ of linear classifiers.

Parameter estimation

Let (dropping the $\log 2$ factor)

$$J(\beta) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-y_i \phi(x_i)^\intercal \beta}) = -\frac{1}{n} \sum_{i=1}^{n} \log(\operatorname{sig}(y_i \phi(x_i)^\intercal \beta))$$

Properties of the sigmoid function:

$$\operatorname{sig}(-z) = 1 - \operatorname{sig}(z) \qquad \frac{\partial \operatorname{sig}(z)}{dz} = \operatorname{sig}(z) \operatorname{sig}(-z)$$
$$\frac{\partial \log \operatorname{sig}(z)}{dz} = \operatorname{sig}(-z) \qquad \frac{\partial^2 \log \operatorname{sig}(z)}{dz^2} = -\operatorname{sig}(z) \operatorname{sig}(-z)$$

J is differentiable, with gradient and Hessian

$$\nabla_\beta J(\beta) = -\frac{1}{n} \sum_{i=1}^{n} y_i \phi(x_i) \operatorname{sig}(-y_i \phi(x_i)^\intercal \beta)$$
$$\nabla^2_\beta J(\beta) = \frac{1}{n} \sum_{i=1}^{n} \operatorname{sig}(y_i \phi(x_i)^\intercal \beta) \operatorname{sig}(-y_i \phi(x_i)^\intercal \beta) \phi(x_i) \phi(x_i)^\intercal \succeq 0$$

Parameter estimation

No closed-form solution

The objective function J can be minimised using e.g.

Gradient descent, with update

$$\beta^{(t+1)} = \beta^{(t)} + \frac{\eta}{n} \sum_{i=1}^{n} y_i \phi(x_i) \operatorname{sig}(-y_i \phi(x_i)^\intercal \beta^{(t)})$$

where η is the learning rate.

Stochastic gradient descent, with update

$$\beta^{(t+1)} = \beta^{(t)} + \frac{\eta_t}{n_b} \sum_{i=1}^{n_b} \widetilde{y}_i^{(t)} \phi(\widetilde{x}_i^{(t)}) \operatorname{sig}(-\widetilde{y}_i^{(t)} \phi(\widetilde{x}_i^{(t)})^\intercal \beta^{(t)})$$

where $(\widetilde{x}_i^{(t)}, \widetilde{y}_i^{(t)})$ is the tth mini-batch of size n_b and (η_t) is a decreasing sequence of learning rates.

Newton-Raphson algorithm (known as iterative reweighted least square (IRLS) in this case), with update

$$\beta^{(t+1)} = \beta^{(t)} - (\nabla^2_\beta J(\beta^{(t)}))^{-1} \nabla_\beta J(\beta^{(t)}).$$

Iterative reweighted least squares

We can write the gradient and Hessian in a more compact form.
Let $\mu \in \mathbb{R}^n$ be such that $\mu_i = \text{sig}(\phi(x_i)^\intercal \beta)$ and S be a n-by-n diagonal entries with entries $\mu_i(1-\mu_i)$
Let c be the vector such that $c_i = \mathbb{1}_{y_i=+1}$
Then

$$\begin{aligned} n\nabla_\beta J(\beta) &= -\sum_{i=1}^n \frac{y_i \phi(x_i) e^{-y_i \phi(x_i)^\intercal \beta}}{1+e^{-y_i \phi(x_i)^\intercal \beta}} \\ &= \sum_{i=1}^n \phi(x_i)(\mu_i - c_i) \\ &= \boldsymbol{\Phi}^\intercal(\mu - c) \\ n\nabla^2_\beta J(\beta) &= \sum_{i=1}^n \text{sig}(y_i \phi(x_i)^\intercal \beta)\,\text{sig}(-y_i \phi(x_i)^\intercal \beta)\phi(x_i)\phi(x_i)^\intercal \\ &= \boldsymbol{\Phi}^\intercal S \boldsymbol{\Phi} \end{aligned}$$

Iterative reweighted least squares

Let $\beta^{(t)}$ be the parameter value after t iterations of the Newton-Raphson algorithm. Let $\mu^{(t)}$ and $S^{(t)}$ be the corresponding vectors and matrices.
Newton-Raphson update

$$\begin{aligned} \beta^{(t+1)} &= \beta^{(t)} - (\nabla^2_\beta J(\beta^{(t)}))^{-1} \nabla_\beta J(\beta^{(t)}) \\ &= \beta^{(t)} + (\boldsymbol{\Phi}^\intercal S^{(t)} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\intercal (c - \mu^{(t)}) \\ &= (\boldsymbol{\Phi}^\intercal S^{(t)} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\intercal S^{(t)} (\boldsymbol{\Phi}\beta^{(t)} + (S^{(t)})^{-1}(c - \mu^{(t)})) \\ &= (\boldsymbol{\Phi}^\intercal S^{(t)} \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\intercal S^{(t)} z^{(t)} \end{aligned}$$

where $z^{(t)} = \boldsymbol{\Phi}\beta^{(t)} + (S^{(t)})^{-1}(c - \mu^{(t)})$. Then $\beta^{(t+1)}$ is a solution of the weighted least squares problem

$$\begin{aligned} \beta^{(t+1)} &= \arg\min_{\beta \in \mathbb{R}^p} \sum_{i=1}^n S^{(t)}_{ii}(z^{(t)}_i - \phi(x_i)^\intercal \beta)^2 \\ &= \arg\min_{\beta \in \mathbb{R}^p} (z^{(t)} - \boldsymbol{\Phi}\beta)^\intercal S^{(t)} (z^{(t)} - \boldsymbol{\Phi}\beta). \end{aligned}$$

Linearly separable data

or plane perfectly separates data.

Assume that the data are linearly separable. That is, there exists a vector $\widetilde{\beta}$ such that $y_i(\phi(x_i)^\intercal \widetilde{\beta}) > 0$ for $i = 1, \ldots, n$.
For any $c > 0$, the objective function for $c\widetilde{\beta}$ is

$$J(c\widetilde{\beta}) = \frac{1}{n}\sum_{i=1}^n \log(1 + e^{-cy_i \phi(x_i)^\intercal \widetilde{\beta}})$$

which can be made arbitrarily close to 0 as $c \to \infty$.
The associated estimated discriminant function $\widehat{f}(x) = \lim_{c\to\infty} c\phi(x)^\intercal \widetilde{\beta}$ is therefore equal to $+/-\infty$, depending on the sign of $\phi(x)^\intercal \widetilde{\beta}$.
Similarly, the plug-in estimator of $\eta(x) = \Pr(Y = 1 \mid X = x)$ is $\widehat{\eta}(x) = 1$ or 0, yielding overconfident class predictions.
One way to address this problem is to add a regularisation term to the objective function.

Logistic regression for multi-class classification

Let $Y \in \{1, \ldots, K\}$.
Multi-class logistic regression uses the softmax function to model the conditional class probabilities.
For $k = 1, \ldots, K$

$$\Pr(Y = k \mid X = x) = \frac{\exp(\phi(x)^\intercal \beta_k)}{\sum_{k'=1}^K \exp(\phi(x)^\intercal \beta_{k'})}$$

where $\beta_k \in \mathbb{R}^p$ for $k = 1, \ldots, K$.
This yields linear decision boundaries as

$$\log \frac{\Pr(Y = k \mid X = x)}{\Pr(Y = \ell \mid X = x)} = \phi(x)^\intercal(\beta_k - \beta_\ell).$$

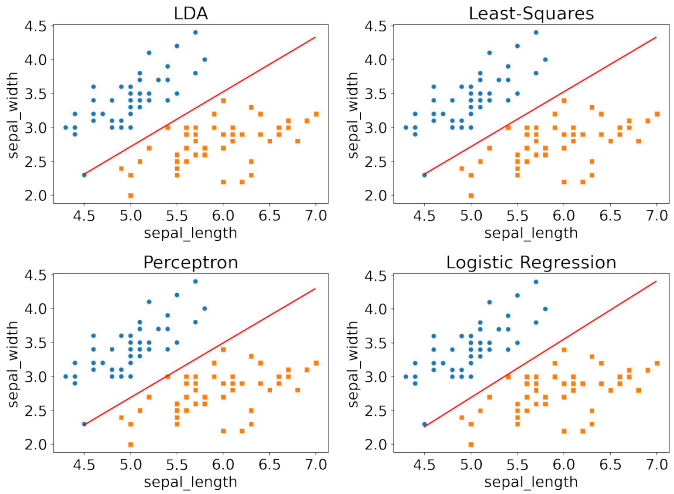
Example: Binary classification on Iris data

Iris dataset

Subset of $n = 100$ observations from the classes setosa and versicolor

Two dimensions: sepal length and sepal width

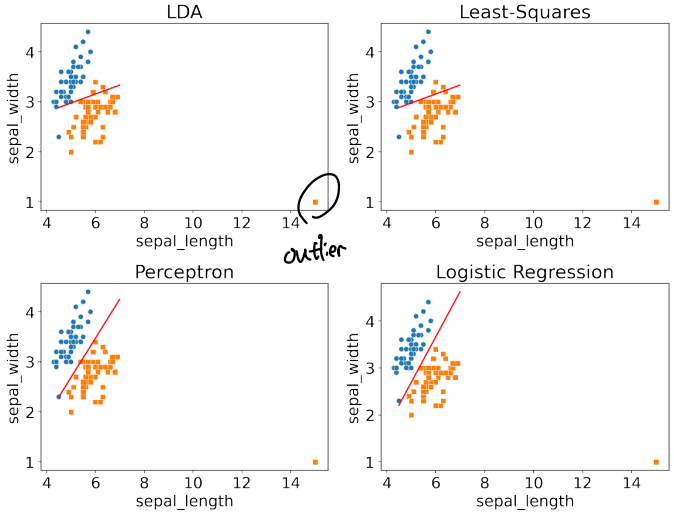
Example: Binary classification on Iris data



Example: Binary classification on Iris data

Add an artificial outlier example from the class versicolor, far from the boundary

Example: Binary classification on Iris data



Example: Multiclass classification on the Crabs data

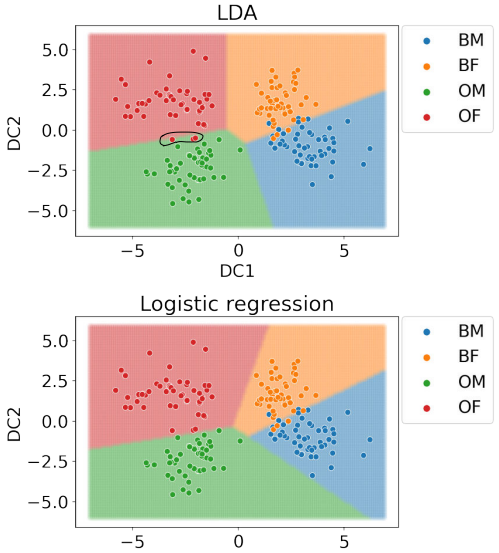
Crabs dataset ($K = 4$ classes)

For visualisation purposes, first extract the first two LDA discriminant coordinates, use these coordinates as our input data

Consider a simple linear model, that is a transformation function $\phi(x_i) = (1\ x_{i1}\ x_{i2})^\intercal$.

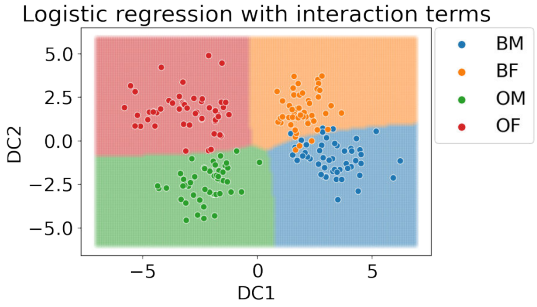
Compare LDA and logistic regression

Example: Multiclass classification on the Crabs data



Example: Multiclass classification on the Crabs data

Add interaction terms $\phi(x_i) = (1\ x_{i1}\ x_{i2}\ x_{i1}x_{i2})^\intercal$.



Statistical Machine Learning
Hilary Term 2021

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

Parametric vs Nonparametric

So far, the methods studied were parametric: prediction rule h_θ parameterised by some (finite-dimensional) parameter θ

θ is fitted using maximum likelihood or empirical risk minimisation using the training data, giving the learned prediction rule $h_{\widehat{\theta}}$

In this lecture: Model-free, Nonparametric approach

Number of parameters grows with the number of data

Nearest neighbor (NN) classification/regression

Let $\mathcal{X}$ be the input space

Let $\rho : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ be some distance/dissimilarity function

E.g., Euclidian distance (L2 norm)

$$\rho(x, x') = \|x - x'\|_2$$

Nearest neighbor (NN) classification/regression

Training: Store the entire training set $(x_1, y_1), \ldots, (x_n, y_n)$

Learned Prediction rule

$$\widehat{h}^{(d)}(x) = y_{\mathsf{nn}(x)}$$

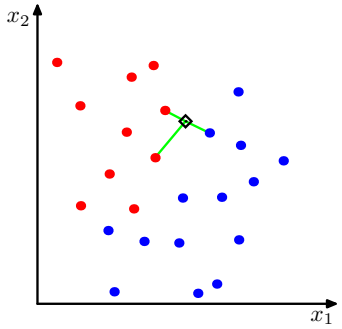
where $\mathsf{nn}(x) \in \{1, 2, \cdots, n\}$, is the index of the training example whose input is the closest to x

$$\mathsf{nn}(x) = \underset{i \in \{1,2,\cdots,n\}}{\arg\min} \|x - x_i\|_2^2$$

Inductive bias: Label of point is similar to the label of nearby points

Visual example

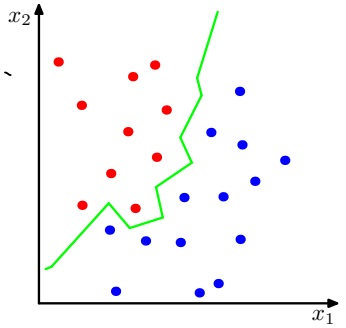
In this 2-dimensional example, the nearest point to x is a red training instance, thus, x will be labeled as red.



(a)

Decision boundary

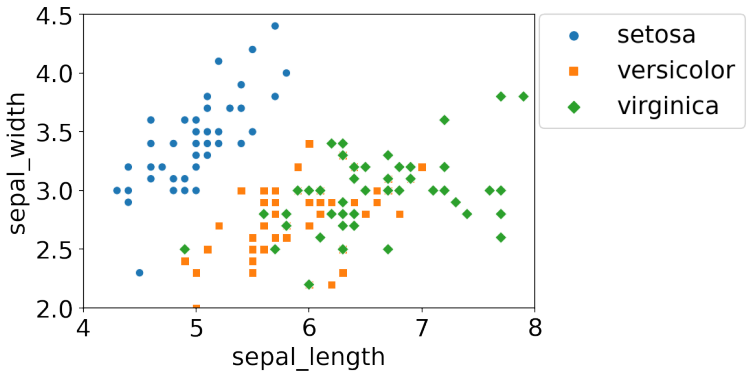
For every point in the space, we can determine its label using the NN classification rule. This gives rise to a decision boundary that partitions the space into different regions.



(b)

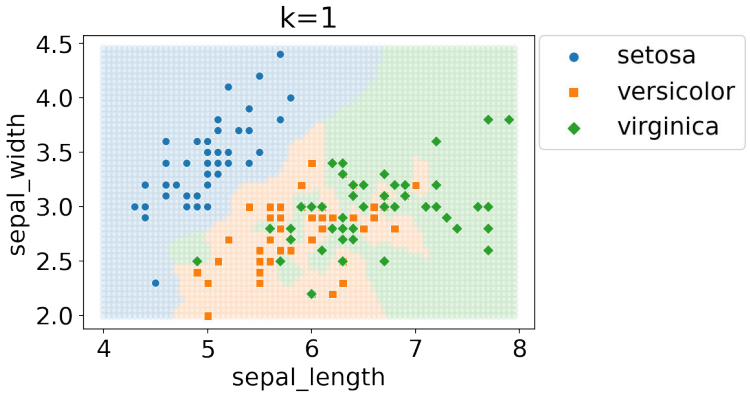
Nearest neighbor (NN) classification/regression

Example on Iris data (2D)



Nearest neighbor (NN) classification/regression

Example on Iris data (2D)



Nearest neighbor (NN) classification/regression

Theorem (Cover-Hart Inequality)

Let $\widehat{h}^{(d_n)}$ be the NN learned classifier for a dataset of size n. Under mild assumptions,

$$R(h^*) \leq \lim_{n\to\infty} \mathbb{E}[R(\widehat{h}^{(D_n)})] \leq 2R(h^*)\underbrace{(1 - R(h^*))}_{\leq 1} \leq 2R(h^*)$$

The risk is asymptotically at worst twice that of the Bayes optimal classifier.

How to measure nearness with other distances?

Previously, we used the Euclidean distance

$$\mathsf{nn}(x) = \underset{i\in\{1,\ldots,n\}}{\arg\min} \|x - x_i\|_2^2$$

We can also use alternative distances

$$\|x - x_n\|_q = \left(\sum_j |x_j - x_{ij}|^q\right)^{1/q}$$

for $q \geq 1$.
E.g., the L_1 distance for $q = 1$ (i.e., city block distance, or Manhattan distance)

$$\begin{aligned}\mathsf{nn}(x) &= \underset{i\in\{1,\ldots,n\}}{\arg\min} \|x - x_i\|_1 \\ &= \underset{i\in\{1,\ldots,n\}}{\arg\min} \sum_{j=1}^{p} |x_j - x_{ij}|\end{aligned}$$

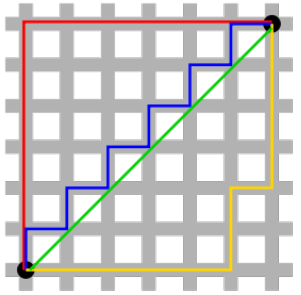


Figure: Green line is Euclidean distance. Red, Blue, and Yellow lines are L_1 distance

Regression/classification with k neighbours?

Denote $\mathsf{knn}(x) \subset \{1,\ldots,n\}$ the subset of cardinality k corresponding to the indices of the k nearest neighbours of x

Classification Rule ($y_i \in \{1,\ldots,K\}$)

Every neighbour votes: neighbour $i \in \mathsf{knn}(x)$ votes for class y_i
Aggregate everyone's vote to obtain the discriminant function; for each class $c = 1,\ldots,K$

$$\widehat{f}_c(x) = \sum_{i\in\mathsf{knn}(x)} \mathbb{1}_{y_i=c}$$

Label with the majority

$$\widehat{h}(x) = \underset{c=1,\ldots,K}{\arg\max}\, \widehat{f}_c(x)$$

Regression rule

Average across nearest neighbours:

$$\widehat{h}(x) = \frac{\sum_{i\in\mathsf{knn}(x)} y_i}{k}$$

kNN as a plug-in method

Can interpret kNN as a plug-in approach
For classification, $\Pr(Y = c \mid X = x)$ is approximated by

$$\frac{1}{k}\sum_{i\in\mathsf{knn}(x)} \mathbb{1}_{y_i=c}$$

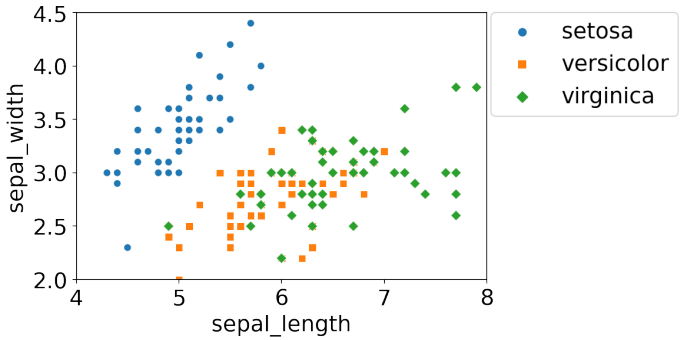
For regression, the conditional cdf $\Pr(Y \leq y \mid X = x)$ is approximated by

$$\frac{1}{k}\sum_{i\in\mathsf{knn}(x)} \mathbb{1}_{y_i\leq y}$$

Nonparametric estimators of the conditional distribution

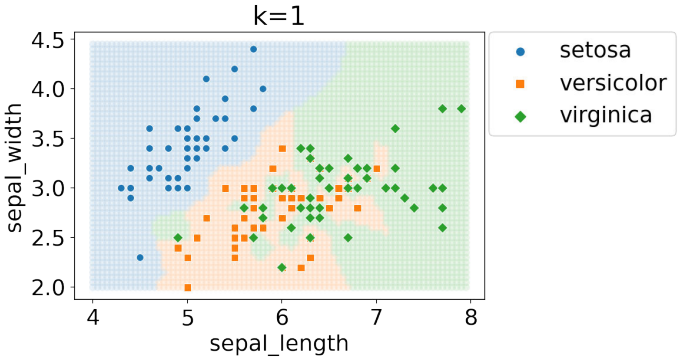
Regression/classification with k neighbours?

Example on Iris data (2D)



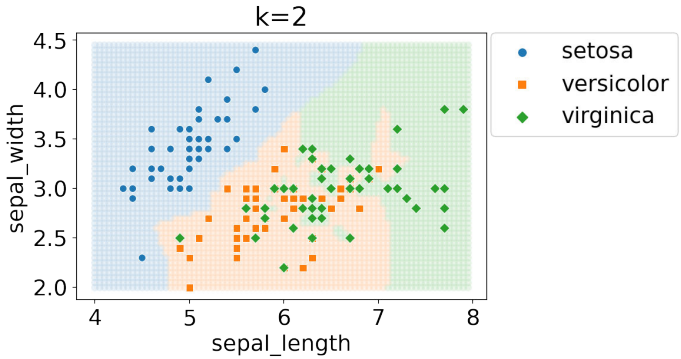
Regression/classification with k neighbours?

Example on Iris data (2D)



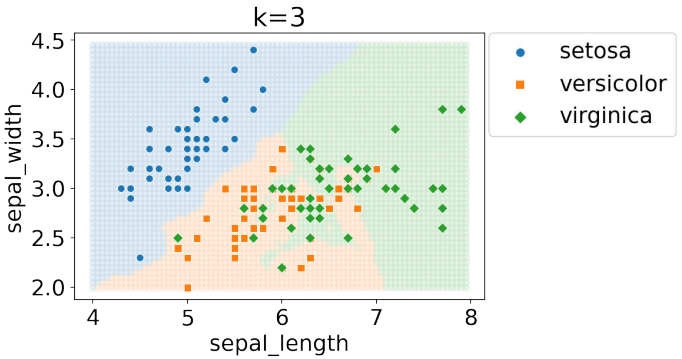
Regression/classification with k neighbours?

Example on Iris data (2D)



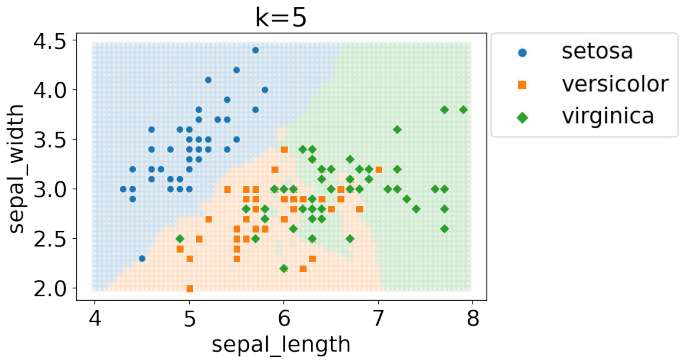
Regression/classification with k neighbours?

Example on Iris data (2D)



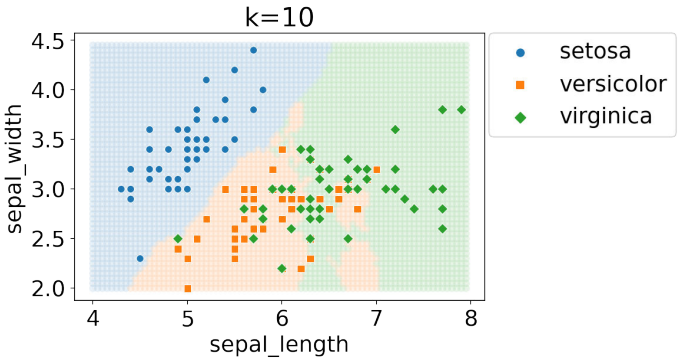
Regression/classification with k neighbours?

Example on Iris data (2D)



Regression/classification with k neighbours?

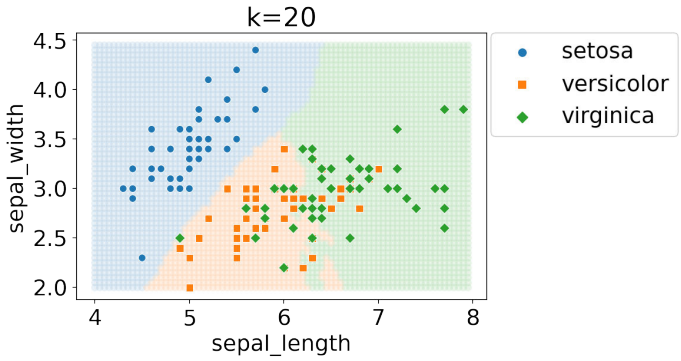
Example on Iris data (2D)



12

Regression/classification with k neighbours?

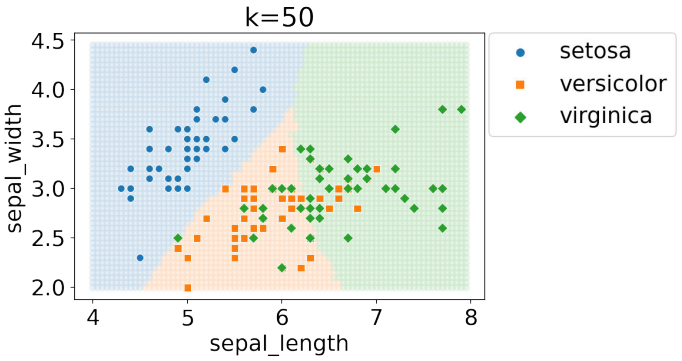
Example on Iris data (2D)



12

Regression/classification with k neighbours?

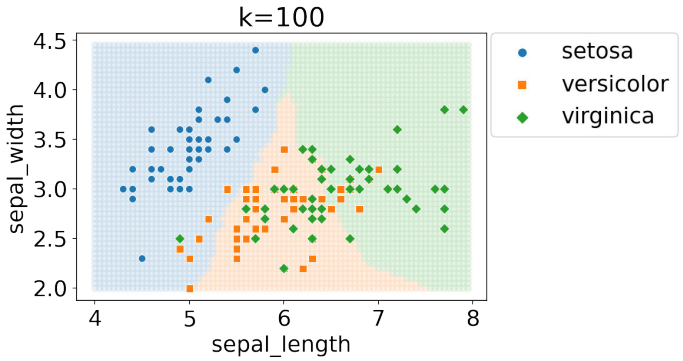
Example on Iris data (2D)



12

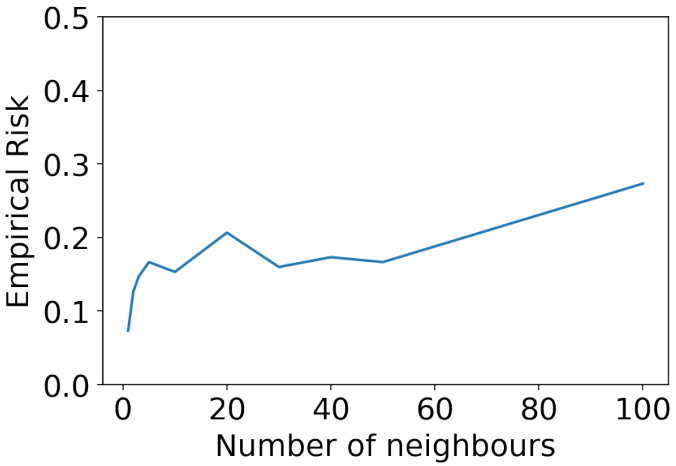
Regression/classification with k neighbours?

Example on Iris data (2D)



Regression/classification with k neighbours?

Example on Iris data (2D)



Regression/classification with k neighbours?

Bias/variance - Approximation/estimation tradeoff

- Small number of neighbours k: small bias, large variance
- Large number of neighbours k: large bias, small variance

k may be chosen by using a validation set, or cross-validation

Choosing the right distance is also important

Preprocess data

Assuming all features are equally important!

Distances depend on units of the features.

Normalise data to have zero mean and unit standard deviation in each dimension

- Compute the means and standard deviations in each dimension $j = 1, \ldots, p$

$$\bar{x}_j = \frac{1}{n}\sum_i x_{ij}, \qquad s_j^2 = \frac{1}{n-1}\sum_i (x_{ij} - \bar{x}_j)^2$$

- Scale the feature accordingly

$$x_{ij} \leftarrow \frac{x_{ij} - \bar{x}_j}{s_j}$$

Many other ways of normalising data

Summary

Advantages of k-NN

- Simple and easy to implement – just computing distance
- Theoretically, has strong guarantees of "doing the right thing"

Disadvantages of k-NN

- Computationally intensive: $O(np)$ for labeling a data point
- Curse of dimensionality: Need a lot of data for large p. May want to reduce dimensions first.
- Not useful for understanding relationships between attributes.
- We need to "carry" the training data around (nonparametric approach).
- Choosing the right distance measure and k can be involved.

Statistical Machine Learning
Hilary Term 2021

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

Neural Networks

Introduction

Linear prediction rules

Regression

$$h(x) = \phi(x)^\intercal \beta$$

Binary classification

$$h(x) = \begin{cases} 1 & \text{if } \phi(x)^\intercal \beta \geq 0 \\ -1 & \text{Otherwise} \end{cases}$$

Convex optimisation problem to estimate the parameters

Interpretable prediction rule

Introduction

Choice of the (fixed) input/feature transformation ϕ

Tailored to the problem (images, music, text, etc.)

Difficult and time-consuming to find good features, requires a lot of engineering

Default choice (e.g polynomial expansions with interactions)

$$\phi(x_i) = (1, x_i, x_i^2, x_i^3)$$

May require a large number of features to represent some sets of functions; e.g. the number of interaction terms of order 3 (of type $x_{i1}x_{i2}x_{i3}$) scales as p^3

Introduction

Alternative is to learn the feature transformations

Adaptive basis functions

For regression

$$h(x) = \beta_0 + \sum_{k=1}^{m} \beta_k \phi_{\theta_k}(x)$$

where θ_k are unknown parameters.

For classification, discriminant function of the form

$$f(x) = \beta_0 + \sum_{k=1}^{m} \beta_k \phi_{\theta_k}(x)$$

$\phi_{\theta_k}(x)$ is an adaptive basis function, parameterised by θ_k, learned from the data

Biological inspiration

Basic computational elements: neurons.

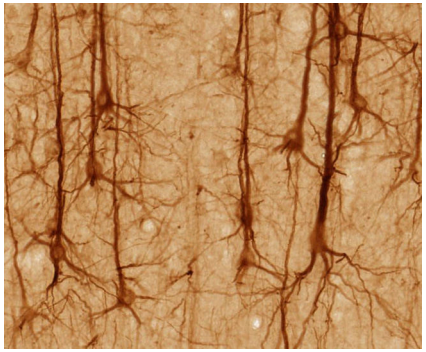
Receives signals from other neurons via dendrites.

Sends processed signals via axons.

Axon-dendrite interactions at synapses.

$10^{10} - 10^{11}$ neurons.

$10^{14} - 10^{15}$ synapses.



Neural network for regression

Let $x_i \in \mathbb{R}^p$ be an input example

Denote $m \geq 1$ the number of hidden units/neurons

Prediction rule of the form,

$$h(x_i) = b^{(o)} + \sum_{k=1}^{m} w_k^{(o)} z_{ik}$$

where for each neuron $k = 1, \ldots, m$

$$z_{ik} = s\left(b_k^{(h)} + \sum_{j=1}^{p} w_{jk}^{(h)} x_{ij}\right).$$

The nonlinear function s is known as the activation/transfer function

The vector $(z_{i1}, \ldots z_{im}) \in \mathbb{R}^m$, called the vector of hidden units, represents the latent features for the example x_i

Neural network for regression

Parameters that need to be estimated:

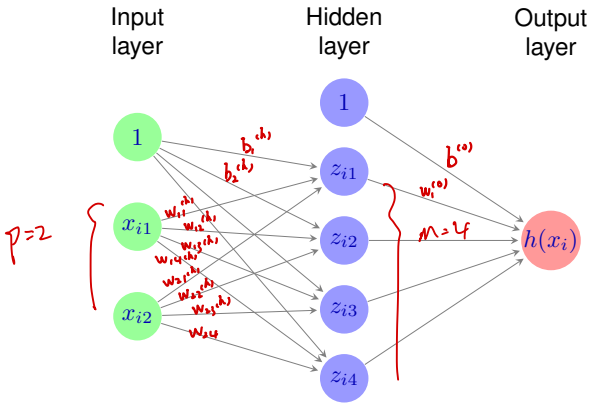
- Output layer: $(b^{(o)}, w_1^{(o)}, \ldots, w_m^{(o)}) \in \mathbb{R}^{m+1}$
- Hidden layer: $(b_k^{(h)}, w_{1k}^{(h)}, \ldots, w_{pk}^{(h)})_{k=1,\ldots,m} \in \mathbb{R}^{(p+1)m}$

The intercept terms $b^{(o)}$ and $b_k^{(h)}$ are often called the bias terms (not to be confused with the bias of an estimator!)

$w_k^{(o)}$ and $w_{jk}^{(h)}$ are called the weights

Neural network for regression

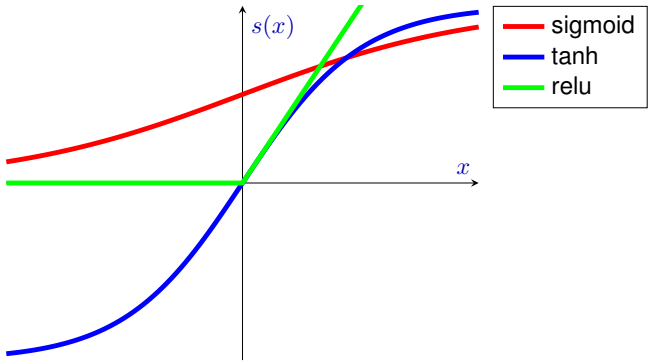
Illustration with $p = 2$ and $m = 4$ hidden inputs/neurons



Neural network for regression

Activation functions

- Rectified Linear Unit (ReLU): $s(x) = \max(0, x)$
- sigmoid: $s(x) = \text{sig}(x) = (1 + e^{-x})^{-1}$
- tanh: $s(x) = \tanh(x)$

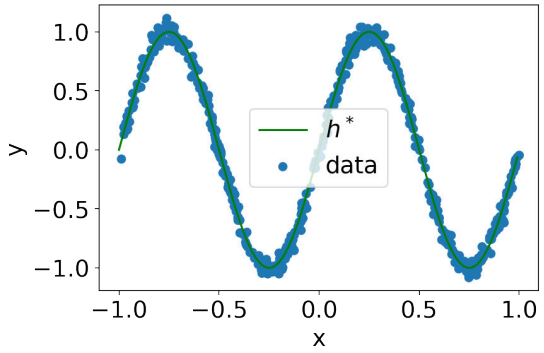


Neural network for regression

Example

Example: synthetic data $(x_i, y_i) \in \mathbb{R}^2$ $(n = 500)$ with Bayes prediction rule

$$h^*(x) = \sin(2\pi x)$$



Neural network for regression

Example

Neural network with $m = 1, \ldots, 5$ hidden units/neurons
tanh activation function

$$h(x_i) = b^{(o)} + \sum_{k=1}^{m} w_k^{(o)} z_{ik}$$

where for each neuron $k = 1, \ldots, m$

$$z_{ik} = \tanh\left(b_k^{(h)} + w_k^{(h)} x_i\right) := \phi_k(x_i).$$

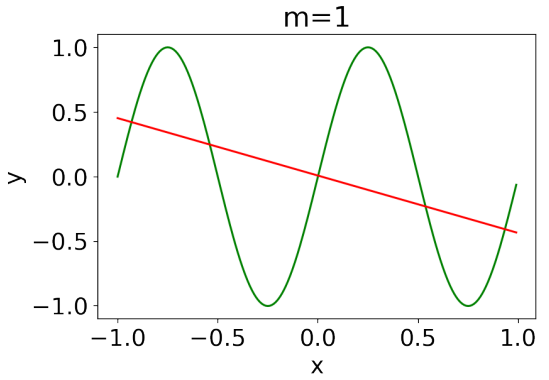
Parameters $\theta = (b^{(o)}, (w_k^{(o)}, b_k^{(h)}, w_k^{(h)})_{k=1,\ldots,m}) \in \mathbb{R}^{3m+1}$
ERM under the squared loss

$$\widehat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} (y_i - h(x_i))^2$$

(more on the estimation algorithm later)

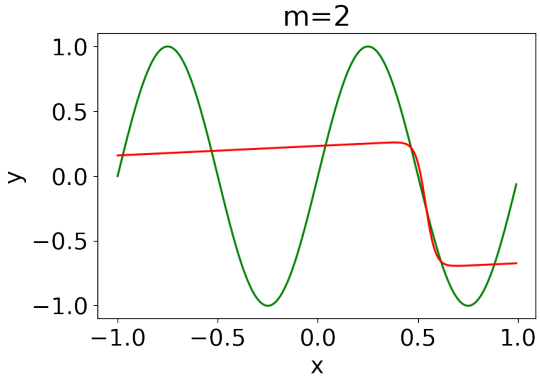
Neural network for regression

Example



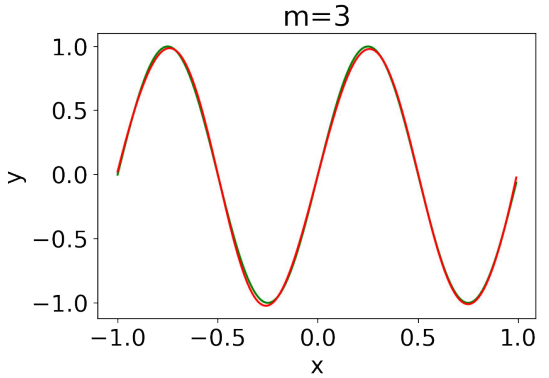
Neural network for regression

Example



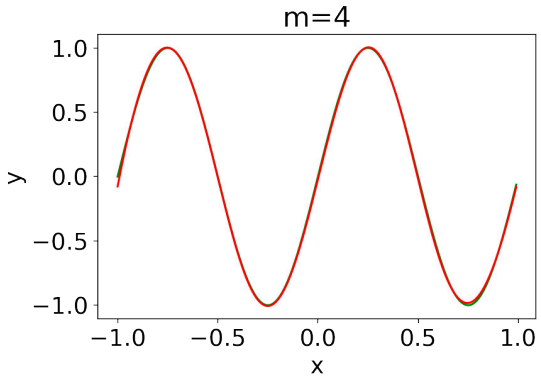
Neural network for regression

Example



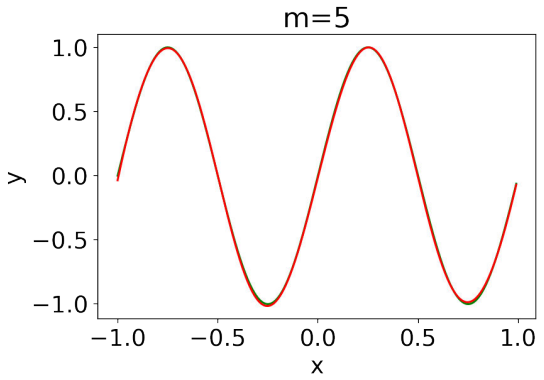
Neural network for regression

Example



Neural network for regression

Example



Neural network for regression

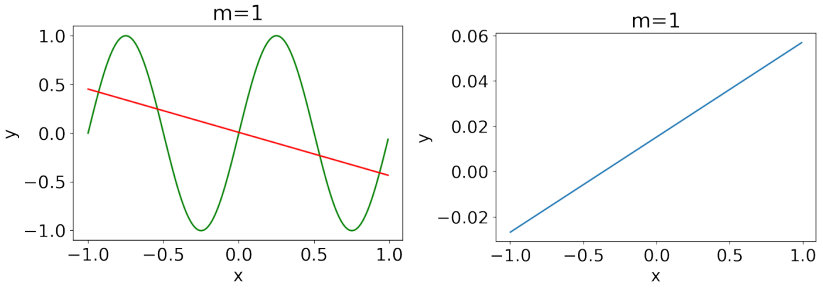
Example

Learned input transformations, for $k = 1, \ldots, m$

$$\widehat{\phi}_k(x) = \tanh\left(\widehat{b}_k^{(h)} + \widehat{w}_k^{(h)} x\right)$$

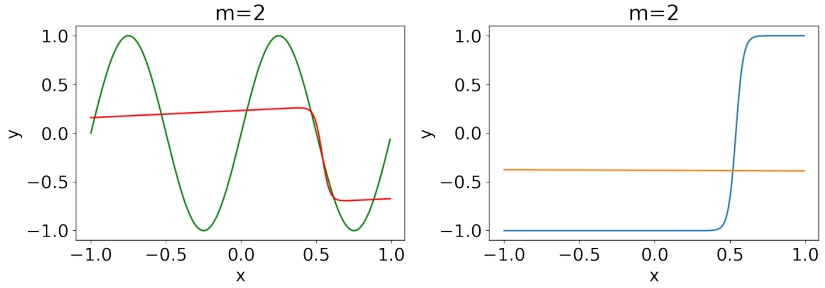
Neural network for regression

Example



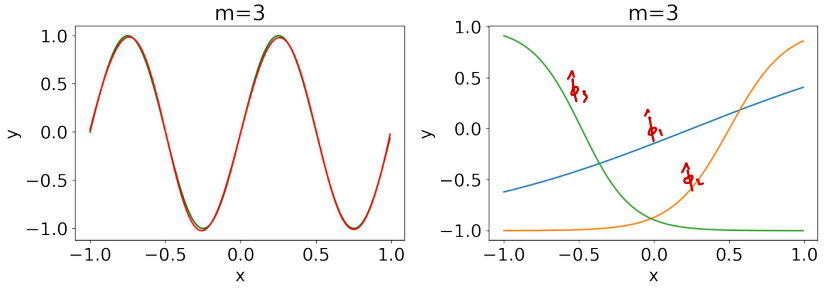
Neural network for regression

Example



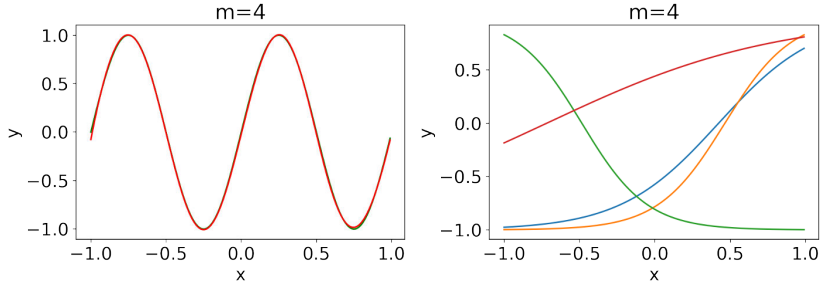
Neural network for regression

Example



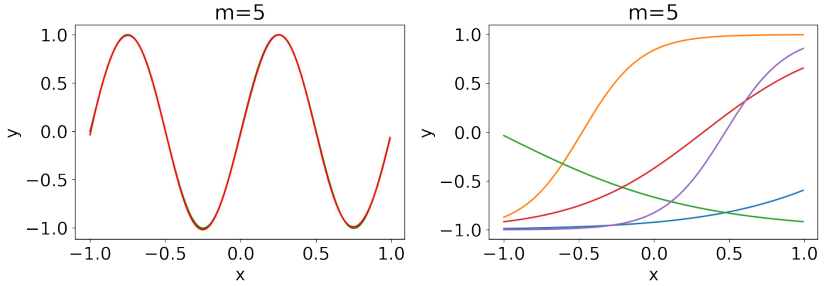
Neural network for regression

Example



Neural network for regression

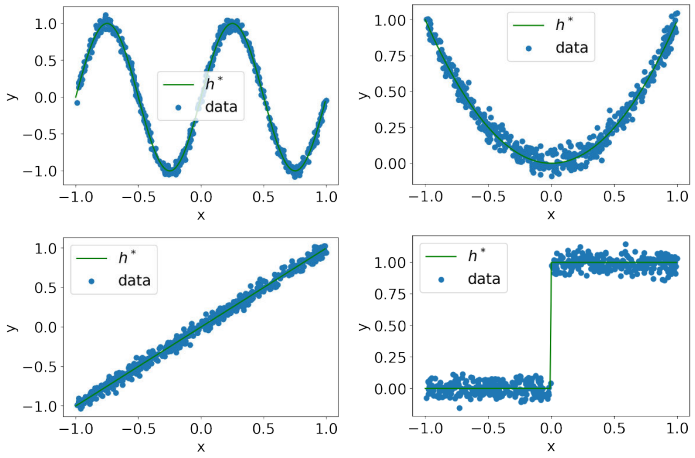
Example



Neural network for regression

Example

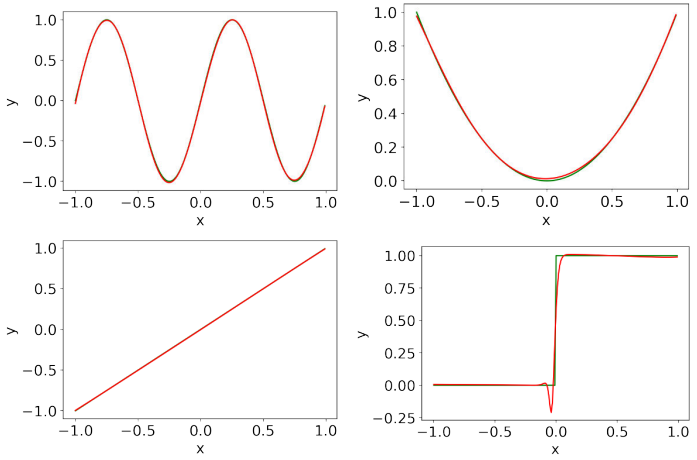
Different datasets



Neural network for regression

Example

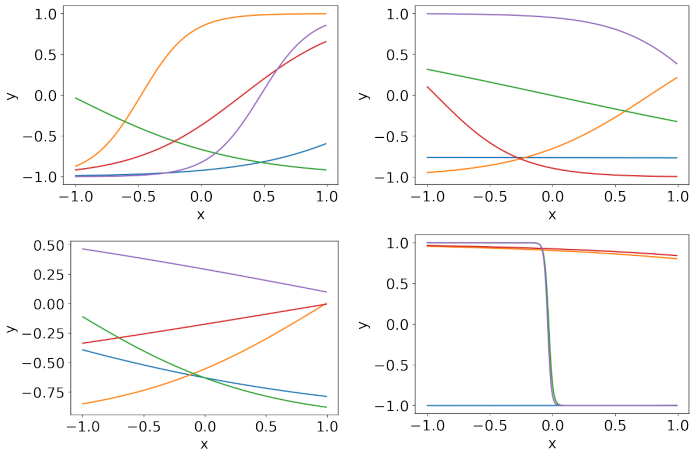
Learned prediction rules with $m = 5$ hidden neurons/units



Neural network for regression

Example

Estimated input transformations $\widehat{\phi}_k(x)$, for $k = 1, \ldots, m$



Neural network for regression

Universal approximation property on $\mathbb{R}$ (simplified)

Let s be the relu, tanh or sigmoid activation function. For $m \geq 1$ let $\mathcal{F}_{s,m}$ be the set of neural network prediction rules with activation s and m hidden units.

Theorem (Cybenko, 1989)

Let $[a, b] \subset \mathbb{R}$ and h^* be a continuous real-valued function on $[a, b]$. For any $\epsilon > 0$, there exists $m \geq 1$ and $h \in \mathcal{F}_{s,m}$ such that

$$|h(x) - h^*(x)| < \epsilon,$$

for all $x \in [a, b]$.

By increasing the number m of neurons in the hidden layer (width) we can approximate any continuous function on a closed interval at an arbitrary precision

Theorem applies more generally to compacts in $\mathbb{R}^p$, and activation functions satisfying some mild assumptions

Neural network for binary classification

Let $x_i \in \mathbb{R}^p$ be an input example

Denote $m \geq 1$ the number of hidden units/neurons

Prediction rule of the form,

$$h(x_i) = \begin{cases} 1 & \text{If } f(x_i) \geq 0 \\ -1 & \text{Otherwise} \end{cases}$$

with neural network discriminant function

$$f(x_i) = b^{(o)} + \sum_{k=1}^{m} w_k^{(o)} z_{ik}$$

where for each neuron $k = 1, \ldots, m$

$$z_{ik} = s\left(b_k^{(h)} + \sum_{j=1}^{p} w_{jk}^{(h)} x_{ij}\right).$$

Neural network for binary classification

Let $x_i \in \mathbb{R}^p$ be an input example

Denote $m \geq 1$ the number of hidden units/neurons

Prediction rule of the form,

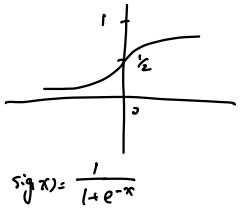
$$h(x_i) = \begin{cases} 1 & \text{If } \eta(x_i) \geq 1/2 \\ -1 & \text{Otherwise} \end{cases}$$

where $\eta(x_i) = \text{sig}(f(x_i)) \in [0, 1]$, with discriminant function

$$f(x_i) = b^{(o)} + \sum_{k=1}^{m} w_k^{(o)} z_{ik}$$

where for each neuron $k = 1, \ldots, m$

$$z_{ik} = s\left(b_k^{(h)} + \sum_{j=1}^{p} w_{jk}^{(h)} x_{ij}\right).$$



Neural network for binary classification

Let $x_i \in \mathbb{R}^p$ be an input example

Denote $m \geq 1$ the number of hidden units/neurons

Prediction rule of the form,

$$h(x_i) = \begin{cases} 1 & \text{If } \eta(x_i) \geq 1/2 \\ -1 & \text{Otherwise} \end{cases}$$

where

$$\eta(x_i) = \text{sig}\left(b^{(o)} + \sum_{k=1}^{m} w_k^{(o)} z_{ik}\right)$$

output layer

where for each neuron $k = 1, \ldots, m$

$$z_{ik} = s\left(b_k^{(h)} + \sum_{j=1}^{p} w_{jk}^{(h)} x_{ij}\right).$$

hidden layer

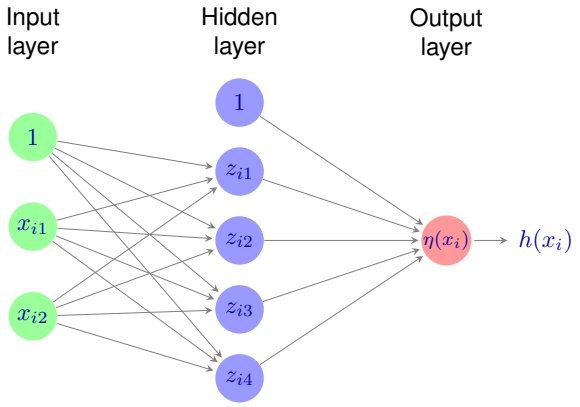
Neural network for binary classification

Activation function s (relu, tanh, sigmoid) for the hidden layer

Activation function sig (sigmoid) for the output layer

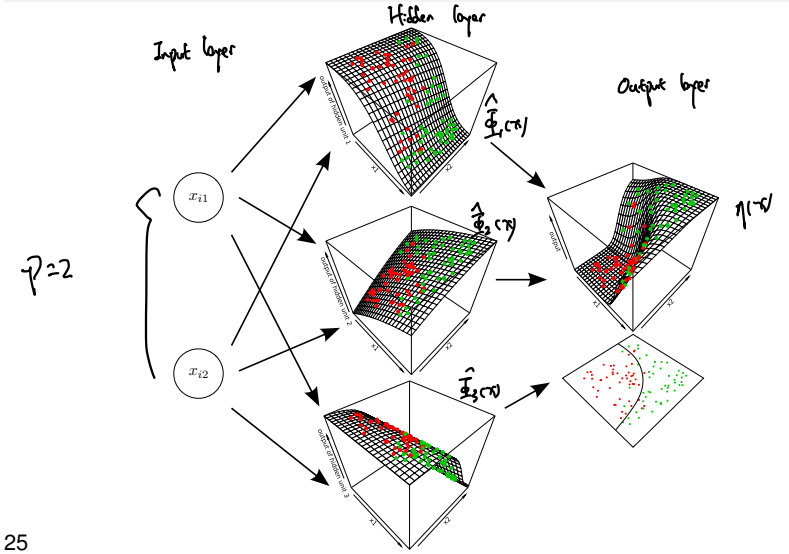
Neural network for binary classification

Illustration with $p = 2$ and $m = 4$ hidden inputs/neurons



Neural Network for classification

Illustration for $p = 2$, $m = 3$, with the activation function s the sigmoid function



Empirical risk minimisation and surrogate loss

Let θ be the parameters of the neural network

Under the 0-1 loss, the empirical risk takes the form

$$\widehat{R}_n(h) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}_{y_i \neq h_\theta(x_i)}$$

As we have seen earlier, this is not amenable to gradient descent

Surrogate logistic loss

$$\begin{aligned}\psi(y_i f_\theta(x_i)) &= \frac{\log(1 + e^{-y_i f_\theta(x_i)})}{\log(2)} \\ &= \frac{-\log \mathrm{sig}(y_i f_\theta(x_i))}{\log(2)} \\ &= \frac{-\log \eta_\theta(x_i)\mathbb{1}_{y_i=1} - \log(1-\eta_\theta(x_i))\mathbb{1}_{y_i=-1}}{\log(2)}\end{aligned}$$

Empirical risk minimisation and surrogate loss

Surrogate loss (dropping $\log 2$ factor)

$$\widetilde{L}(y_i, \eta_\theta(x_i)) = -\log \eta_\theta(x_i)\mathbb{1}_{y_i=1} - \log(1-\eta_\theta(x_i))\mathbb{1}_{y_i=-1}$$

known as the log-loss

ERM under the surrogate loss

$$\widehat{\theta} = \arg\min_\theta \frac{1}{n}\sum_{i=1}^{n} \widetilde{L}(y_i, \eta_\theta(x_i))$$

The learned classifier may equivalently be interpreted as a plug-in classifier under the model

$$\Pr(Y = 1 \mid X = x) = \eta_\theta(x) = \mathrm{sig}(f_\theta(x))$$

where the parameters θ are estimated using maximum-likelihood.

Regularisation for neural networks

Parameters

$$\theta = (b^{(o)}, (w_k^{(o)})_{k=1,\dots,m}, (b_k^{(h)})_{k=1,\dots,m}, (w_{jk}^{(h)})_{j=1,\dots,p,k=1,\dots,m}) \in \mathbb{R}^{(p+2)m+1}$$

Prone to over-fitting if m is large

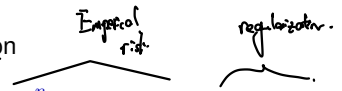
Tikhonov/L2 regularisation on the weights

$$\text{pen}(h_\theta) = \sum_{jk} (w_{jk}^{(h)})^2 + \sum_k (w_k^{(o)})^2$$

Often called weight decay

Regularisation for neural networks

Objective function



$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \widetilde{L}(y_i, \eta_\theta(x_i)) + \frac{\lambda}{n}\text{pen}(h_\theta)$$

$$= -\frac{1}{n}\sum_{i=1}^{n} (\log \eta_\theta(x_i)\mathbb{1}_{y_i=1} + \log(1-\eta_\theta(x_i))\mathbb{1}_{y_i=-1})$$

$$+ \frac{\lambda}{n}\left(\sum_{jk}(w_{jk}^{(h)})^2 + \sum_k (w_k^{(o)})^2\right)$$

where $\lambda \geq 0$ is is the regularisation parameter.

Illustration: different number of neurons/units

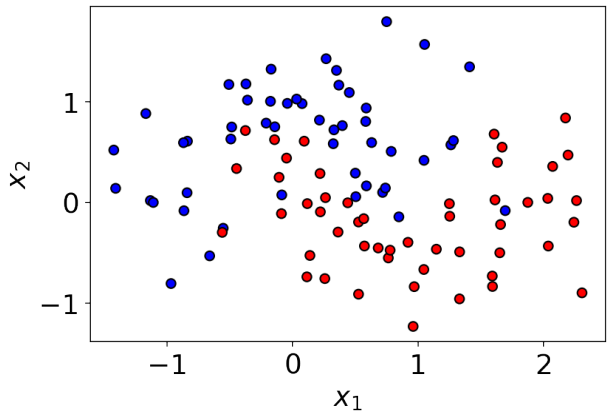


Illustration: different number of neurons/units

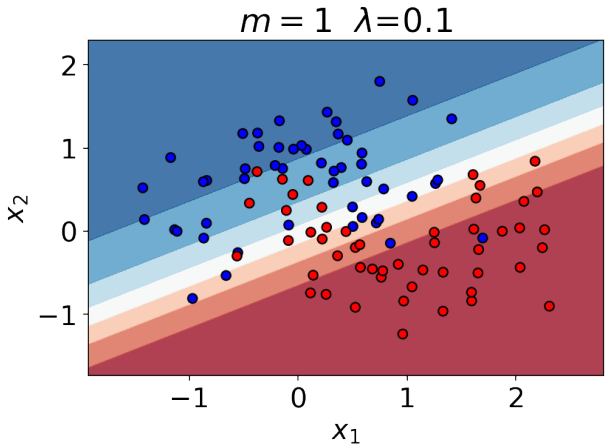
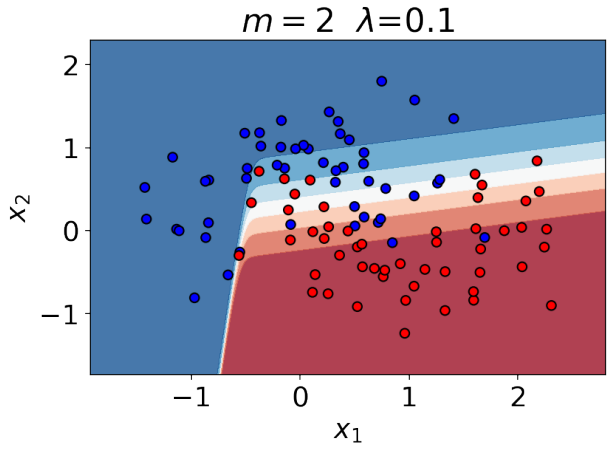
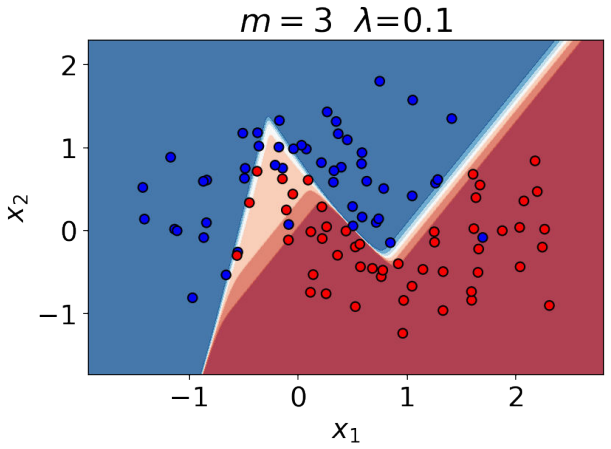


Illustration: different number of neurons/units



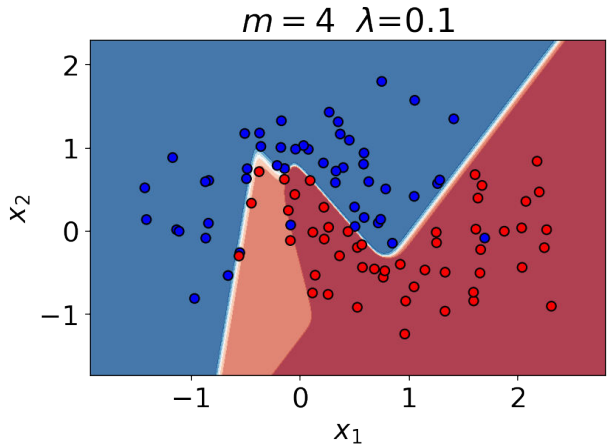
Neural Networks | Binary classification

Illustration: different number of neurons/units



Neural Networks | Binary classification

Illustration: different number of neurons/units



Neural Networks | Binary classification

Illustration: different number of neurons/units

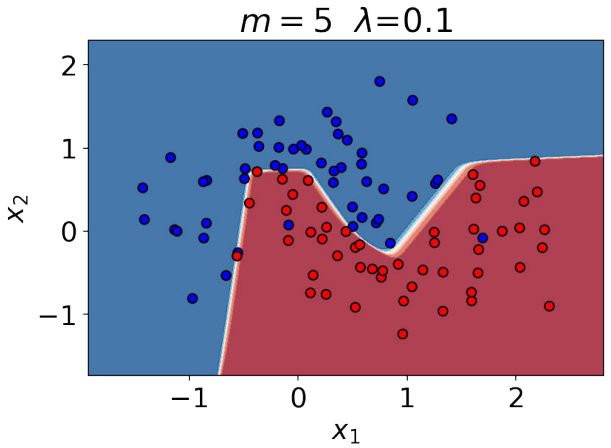


Illustration: different number of neurons/units

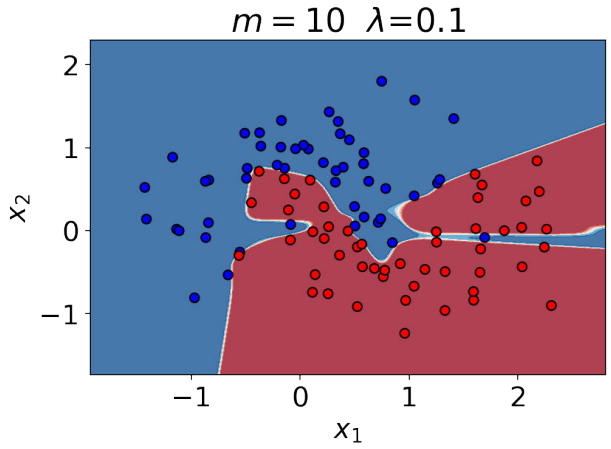


Illustration: different number of neurons/units

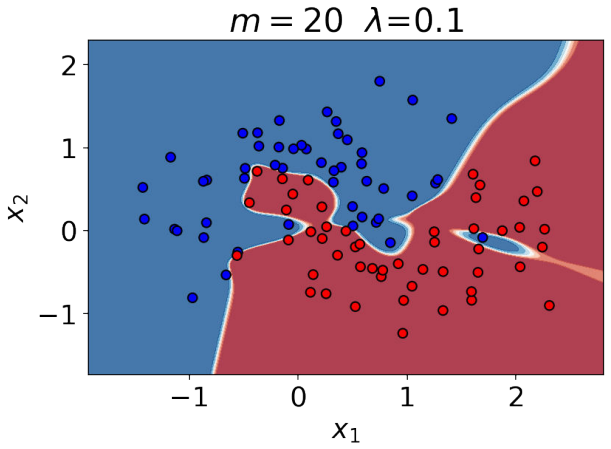


Illustration: different number of neurons/units

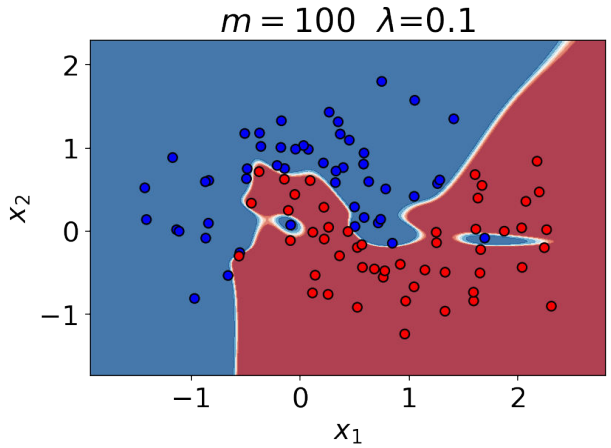


Illustration: different values of the regularisation parameter λ

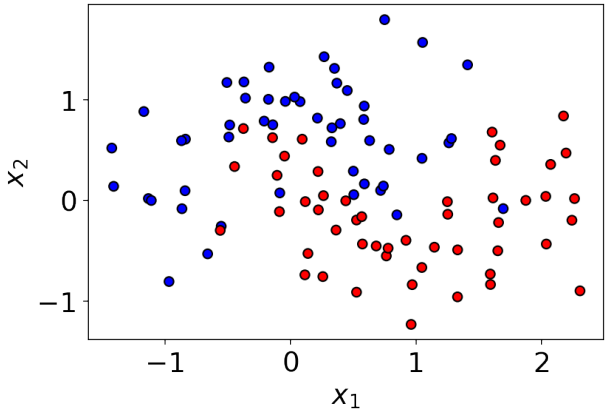
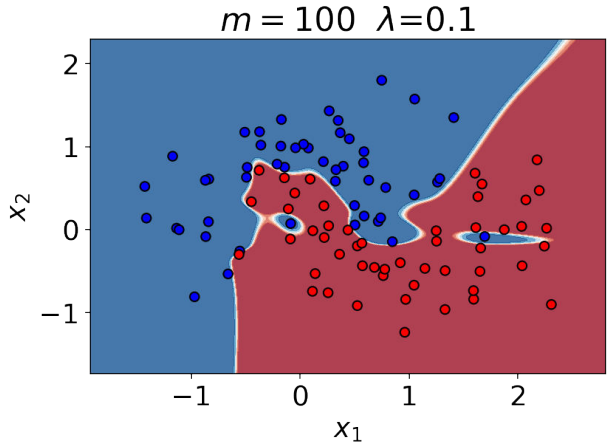
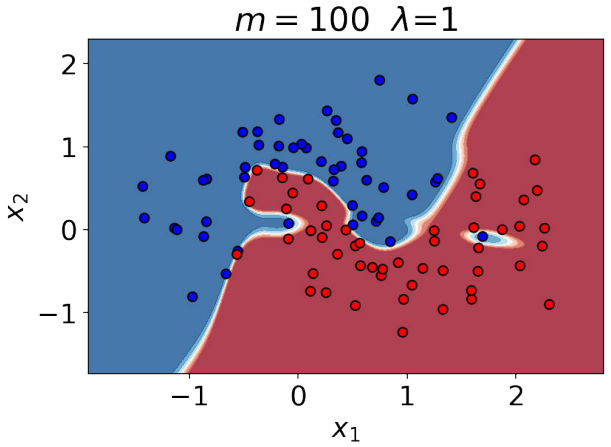


Illustration: different values of the regularisation parameter λ



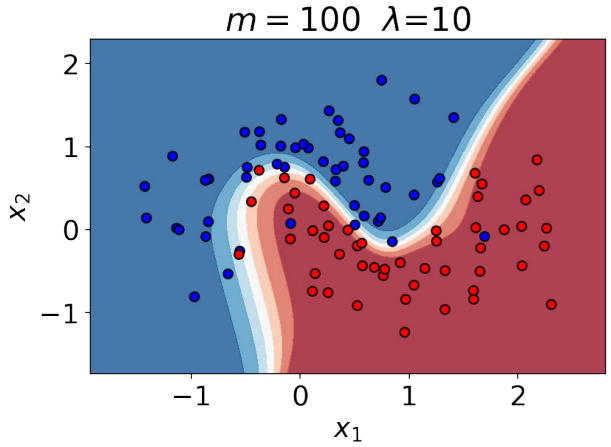
Neural Networks | Binary classification

Illustration: different values of the regularisation parameter λ



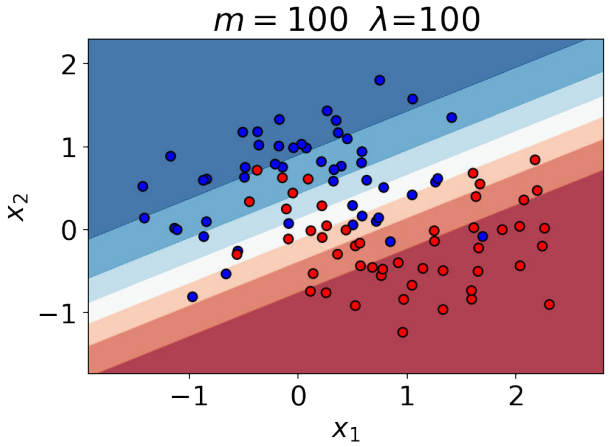
Neural Networks | Binary classification

Illustration: different values of the regularisation parameter λ



Neural Networks | Binary classification

Illustration: different values of the regularisation parameter λ



Training a Neural Network

Objective function $J(\theta)$

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\left(\log \eta_\theta(x_i)\mathbb{1}_{y_i=1} + \log(1-\eta_\theta(x_i))\mathbb{1}_{y_i=-1}\right) + \frac{\lambda}{n}\left(\sum_{jk}(w_{jk}^{(h)})^2 + \sum_{k}(w_k^{(o)})^2\right)$$

Does not admit a closed-form global minimum

Non-convex

Usually resort to (stochastic) gradient descent methods to find a local minimum

Training a Neural Network

Assume the activation function of the hidden layer is the sigmoid function
Using a 0-1 coding $y_i \in \{0, 1\}$ of the output
Objective function $J(\theta)$

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log \eta_\theta(x_i) + (1-y_i)\log(1-\eta_\theta(x_i))) + \frac{\lambda}{n}\left(\sum_{jk}(w_{jk}^{(h)})^2 + \sum_{k}(w_k^{(o)})^2\right)$$

where

$$\eta_\theta(x_i) = \text{sig}\left(b^{(o)} + \sum_{k=1}^{m} w_k^{(o)} z_{ik}\right) \qquad z_{ik} = \text{sig}\left(b_k^{(h)} + \sum_{j=1}^{p} w_{jk}^{(h)} x_{ij}\right)$$

Using the chain rule (for one variable, see Problem Sheets)

$$n\frac{\partial J}{\partial w_k^{(o)}} = 2\lambda w_k^{(o)} + \sum_{i=1}^{n}\frac{\partial J}{\partial \eta_\theta(x_i)}\frac{\partial \eta_\theta(x_i)}{\partial w_k^{(o)}} = 2\lambda w_k^{(o)} + \sum_{i=1}^{n}(\eta_\theta(x_i) - y_i)z_{ik},$$

$$n\frac{\partial J}{\partial w_{jk}^{(h)}} = 2\lambda w_{jk}^{(h)} + \sum_{i=1}^{n}\frac{\partial J}{\partial \eta_\theta(x_i)}\frac{\partial \eta_\theta(x_i)}{\partial z_{ik}}\frac{\partial z_{ik}}{\partial w_{jk}^{(h)}}$$

$$= 2\lambda w_{jk}^{(h)} + \sum_{i=1}^{n}(\eta_\theta(x_i) - y_i)w_k^{(o)} z_{ik}(1-z_{ik})x_{ij}.$$

Training a Neural Network

Can implement gradient descent or stochastic gradient descent

Due to the non-convexity, different initialisations may lead to different local minima

Early stopping can also be used as implicit regularisation

Neural Networks for Multiclass classification

$\mathcal{Y} = \{1, \ldots, K\}$

Use softmax output activations

$$h(x_i) = \underset{c=1,\ldots,K}{\arg\max}\ \eta_c(x_i)$$

$\eta_c(x_i) \in [0,1]$

$\sum_{c=1}^{K} \eta_c(x_i) = 1$

where, for $c = 1, \ldots, K$

$$\eta_c(x_i) = \frac{\exp(b_c^{(o)} + \sum_{k=1}^{m} w_{kc}^{(o)} z_{ik})}{\sum_{c'}\exp(b_{c'}^{(o)} + \sum_{k=1}^{m} w_{kc'}^{(o)} z_{ik})}$$

$K(m+1)$ parameters at the output layer

Surrogate log-loss ($\eta(x_i) = (\eta_1(x_i), \ldots, \eta_K(x_i))$)

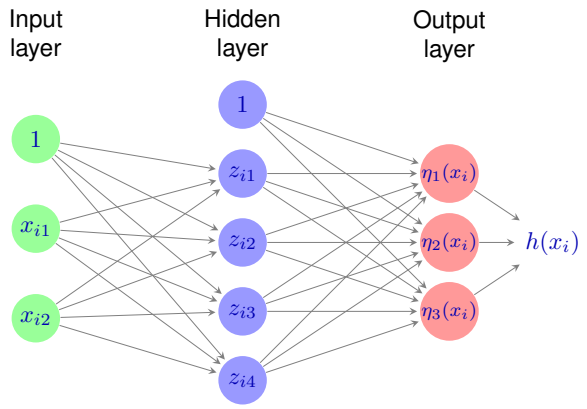
$$\widetilde{L}(y_i, \eta(x_i)) = -\sum_{c=1}^{K}\mathbb{1}_{y_i=c}\log \eta_c(x_i)$$

Can be interpreted as maximum likelihood estimation under the model

$$\Pr(Y = c \mid X = x) = \eta_c(x)$$

Neural network for multiclass classification

Illustration with $p = 2$, $m = 4$ hidden inputs/neurons and $K = 3$ classes



Deep learning

Shallow neural networks

Neural networks with one hidden layer are called shallow neural networks

Let $z_i = (z_{i1}, \ldots, z_{im})$ be the feature representation of example i

Can be written as (for regression)

$$h(x_i) = g^{(2)}(z_i), \quad z_i = g^{(1)}(x_i)$$

where $g^{(2)}$ and $g^{(1)}$ correspond to a linear transformation followed by some potentially nonlinear activation

Hence

$$h(x_i) = g^{(2)}(g^{(1)}(x_i))$$

Deep neural networks

Deep neural networks are neural networks with more than one hidden layer

Let $\ell \geq 2$ be the number of layers

$$h(x_i) = g^{(\ell+1)}(z_i^{(\ell)})$$

for $l = 2, \ldots, \ell$

$$z_i^{(l)} = g^{(l)}(z_i^{(l-1)})$$

and

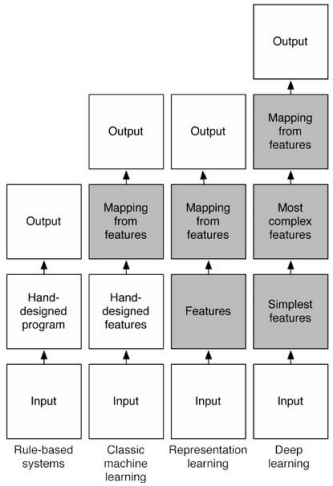
$$z_i^{(1)} = g^{(1)}(x_i)$$

Hence

$$h(x_i) = g^{(\ell+1)}\left(g^{(\ell)}\left(\ldots g^{(1)}(x_i)\right)\right)$$

$z_i^{(1)}, \ldots, z_i^{(\ell)}$ can be thought as forming a hierarchy from low-level to high-level representations of the example x_i

Deep learning intuition



Source: http://rinuboney.github.io/2015/10/18/theoretical-motivations-deep-learning.html

Deep neural networks

Why do we need deep networks?

As we have seen, a shallow network ($\ell = 1$) with a large number of neurons m can approximate any continuous function with compact support

Two issues

- m may need to be very large
- We may not be able to estimate the parameters

Deep networks often lead to more parsimonious representations, and models whose parameters are easier to learn

Feedforward neural network

A feedforward neural network is a neural network where each layer is fully connected to the layer above/below

Sometimes called multilayer perceptron, but we will not used this name

Let ℓ be the number of hidden layers, and $m^{(l)}$ be the number of hidden neurons/units at layer $l = 1, \ldots, \ell$

For binary classification

$$\eta(x_i) = \text{sig}\left(b^{(\ell+1)} + \sum_{k=1}^{m^{(\ell)}} w_k^{(\ell+1)} z_{ik}^{(\ell)}\right)$$

and for $l = 2, \ldots, \ell$, $k = 1, \ldots, m^{(l)}$

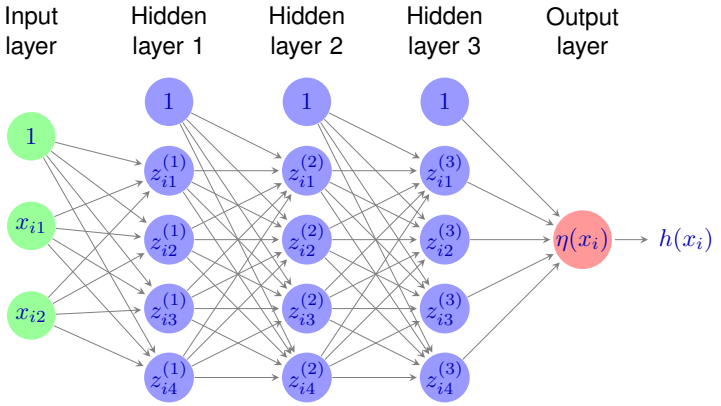
$$z_{ik}^{(l)} = s\left(b_k^{(l)} + \sum_{k'=1}^{m^{(l-1)}} w_{k'k}^{(l)} z_{ik'}^{(l-1)}\right)$$

and for $k = 1, \ldots, m^{(1)}$

$$z_{ik}^{(1)} = s\left(b_k^{(1)} + \sum_{j=1}^{p} w_{jk}^{(1)} x_{ij}\right)$$

Feedforward neural network

Illustration with $p = 2$, $\ell = 3$ layers and $m = 4$ hidden inputs/neurons per layer



Feedforward neural network

Matrix notation

For $l = 1, \ldots, \ell$, define $z_i^{(l)} = (z_{i1}^{(l)}, \ldots, z_{im^{(l)}}^{(l)})^\intercal \in \mathbb{R}^{m^{(l)}}$, and for $l = 1, \ldots, \ell + 1$ let $b^{(l)} = (b_1^{(l)}, \ldots, b_{m^{(l)}}^{(l)})$ and $W^{(l)}$ the $m^{(l-1)}$-by-$m^{(l)}$ matrix with entries $w_{kk'}^{(l)}$, with $m^{(\ell+1)} = 1$ and $m^{(0)} = p$

Denote $\underline{s}$ the multivariate function where the nonlinear activation s is applied elementwise

Then

$$\eta(x_i) = \text{sig}\left(b^{(\ell+1)} + (W^{(\ell+1)})^\intercal z_i^{(\ell)}\right)$$

and for $l = 2, \ldots, \ell$

$$z_i^{(l)} = \underline{s}\left(b^{(l)} + (W^{(l)})^\intercal z_i^{(l-1)}\right)$$

and

$$z_i^{(1)} = \underline{s}\left(b^{(1)} + (W^{(1)})^\intercal x_i\right)$$

Learning in feedforward neural networks

Objective function (ignoring the regularisation term for simplicity)

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} -\log(\eta_\theta(x_i))\mathbb{1}_{y_i=1} - \log(1-\eta_\theta(x_i))\mathbb{1}_{y_i=-1}$$

where θ is the set of parameters, consisting of the intercepts/biases terms and weights at each layer

No closed-form solution

Nonconvex: Objective functions can have many local minima

On large scale problems, usually use stochastic gradient descent, along with a whole host of techniques for optimization, regularization, and initialization.

Efficient computation of the (stochastic) gradient using backpropagation

Recap: Chain rule

Let

$$F(x, y) = f(u_1(x, y), \ldots, u_p(x, y)) = f(z_1, \ldots, z_p)$$

where $z_j = u_j(x, y)$ for $j = 1, \ldots, p$.

Chain rule

$$\frac{\partial F}{\partial x} = \sum_{j=1}^{p} \frac{\partial F}{\partial z_j}\frac{\partial z_j}{\partial x}$$

$$\frac{\partial F}{\partial y} = \sum_{j=1}^{p} \frac{\partial F}{\partial z_j}\frac{\partial z_j}{\partial y}$$

Backpropagation

Using 0-1 coding $y_i \in \{0, 1\}$

Set $z_i^{(\ell+1)} = \eta_\theta(x_i)$

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} -y_i \log(z_i^{(\ell+1)}) - (1-y_i)\log(1-z_i^{(\ell+1)})$$

Gradients wrt $z_{ik}^{(l)}$ computed by recursive applications of chain rule, and propagated through the network backwards.

$$\begin{aligned}
\frac{\partial J}{\partial z_i^{(\ell+1)}} &= -\frac{1}{n}\left(\frac{y_i}{z_i^{(\ell+1)}} - \frac{(1-y_i)}{1-z_i^{(\ell+1)}}\right) \\
\frac{\partial J}{\partial z_{ik}^{(l)}} &= \sum_{r=1}^{m^{(l+1)}} \frac{\partial J}{\partial z_{ir}^{(l+1)}}\frac{\partial z_{ir}^{(l+1)}}{\partial z_{ik}^{(l)}} \\
\frac{\partial J}{\partial w_{jk}^{(l)}} &= \sum_{i=1}^{n} \frac{\partial J}{\partial z_{ik}^{(l)}}\frac{\partial z_{ik}^{(l)}}{\partial w_{jk}^{(l)}}
\end{aligned}$$

Deep learning demo

http://playground.tensorflow.org/

Convolutional neural networks

Convolutional neural networks (CNN) are a specialised type of neural network for dealing with dataset with a grid-like representation

Applications to images, audio, video

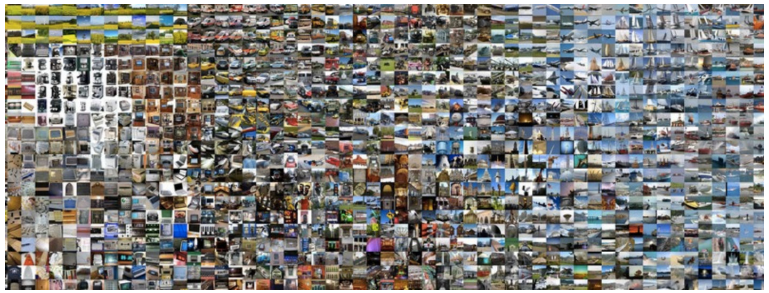
State-of-the-art results in the last ten years

Convolutional neural networks

Example: Imagenet[1]

Image classification task

1000 classes, 1.3M training, 100k test



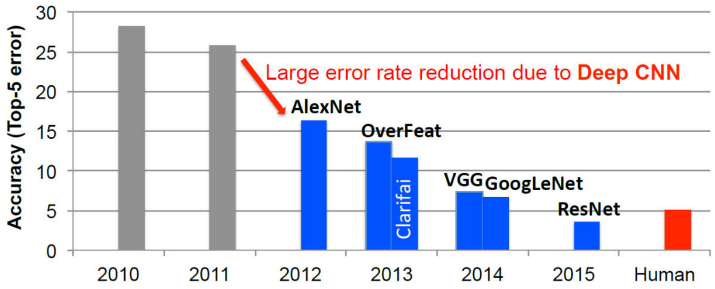
[1] [Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. IJCV. 2015]

Convolutional neural networks

Example: Imagenet[1]

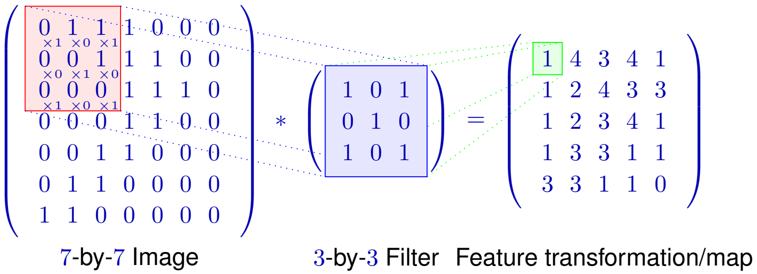
Image classification task

1000 classes, 1.3M training, 100k test

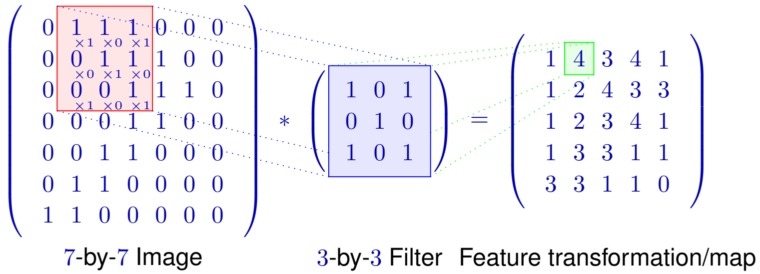


[1] [Russakovsky et al. ImageNet Large Scale Visual Recognition Challenge. IJCV. 2015]

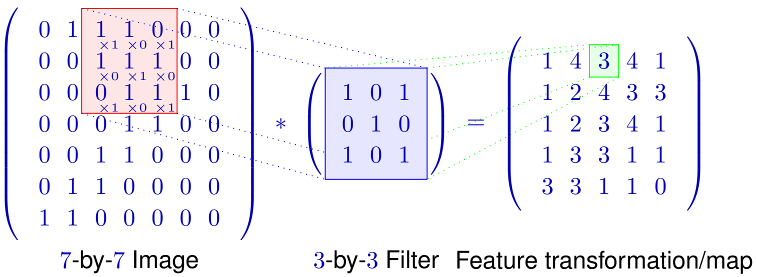
Convolutions



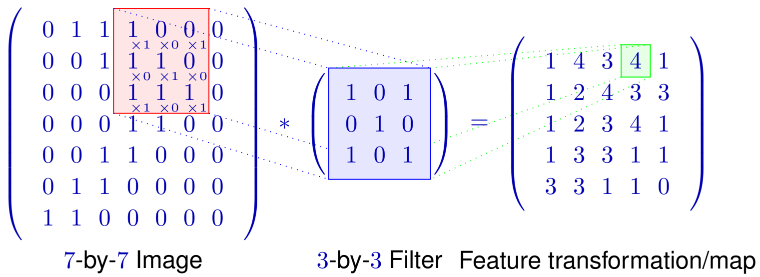
Convolutions



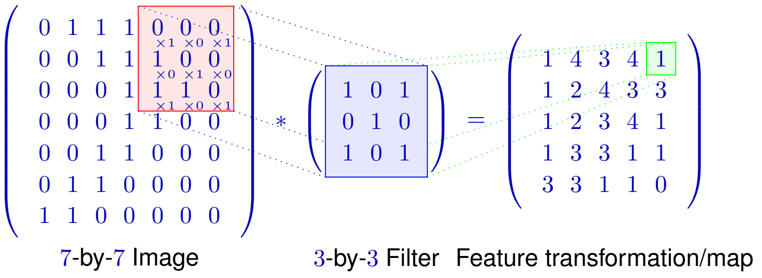
Convolutions



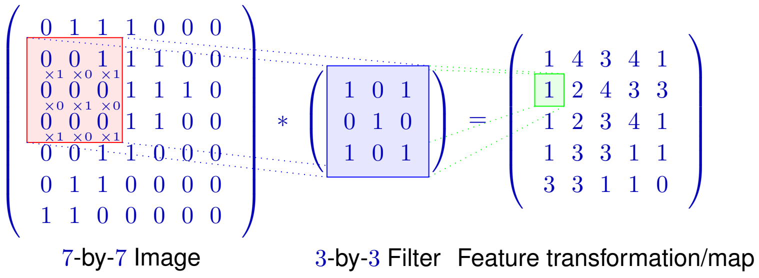
Convolutions



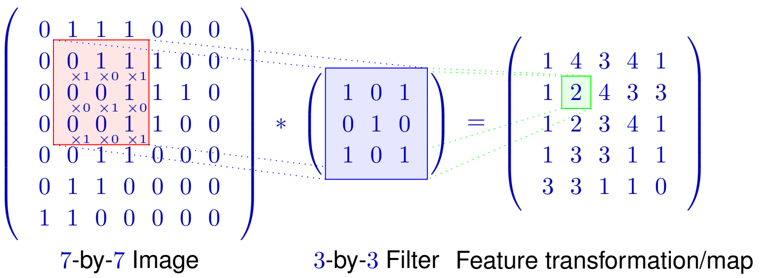
Convolutions



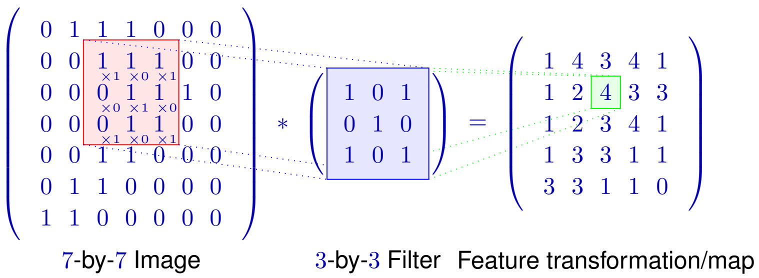
Convolutions



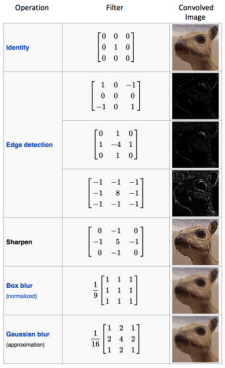
Convolutions



Convolutions



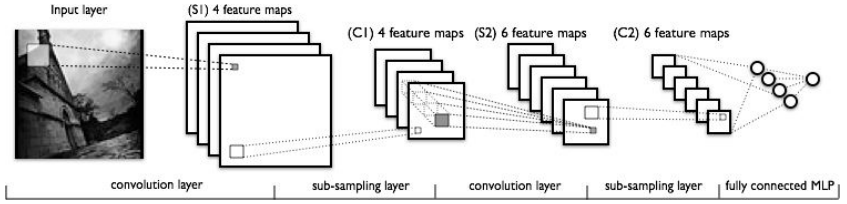
Convolutions



Click for animation.

Source: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Deep Convolutional Neural Networks



Input: 2D image.

Convolution layer

Applies different filters, whose weights are learned from the data, followed by a nonlinear activation

Can be seen as a neural network layer, with sparse connectivity (many weights are set to zero) and shared weights

Pooling and Sub-sampling: replace the output with a summary statistic of the nearby outputs, e.g. max-pooling (allows invariance to small translations in the input).

LeCun et al, Krizhevsky et al.

Summary

Deep neural networks are heavily parameterised models that learn hierarchical representations of the data

Model parameters are learned using stochastic gradient descent

Regularisation via L2 penalisation (weight decay), early stopping, or others (e.g. dropout, not covered here)

Architecture of the network is important: type of layers (fully connected, convolutional, etc.), number of hidden layers, number of neurons per layers; can be chosen using a validation set

Requires a lot of data and computing resources

State-of-the-art results for many applications in computer vision, natural language processing and others

Python libraries: pytorch, tensorflow, keras

Statistical Machine Learning
Hilary Term 2021

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

Decision Trees

Classification and Regression Trees (CART)

Denote input domain by $\mathcal{X}$ and let the output domain be $\mathcal{Y} = \{1, \ldots, K\}$ (classification) or $\mathcal{Y} = \mathbb{R}$ (regression).

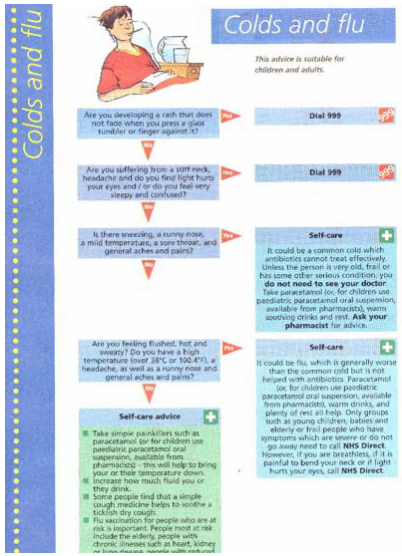
A decision tree gives a partition of $\mathcal{X}$ into R disjoint sets (regions) $\mathcal{P} = \{\mathcal{R}_1, \ldots, \mathcal{R}_R\}$, such that the fitted decision function is constant on each region $\mathcal{R}_j \subset \mathcal{X}$, $j = 1, \ldots, R$, i.e.

$$h(x) = \beta_j, \quad \forall x \in \mathcal{R}_j.$$

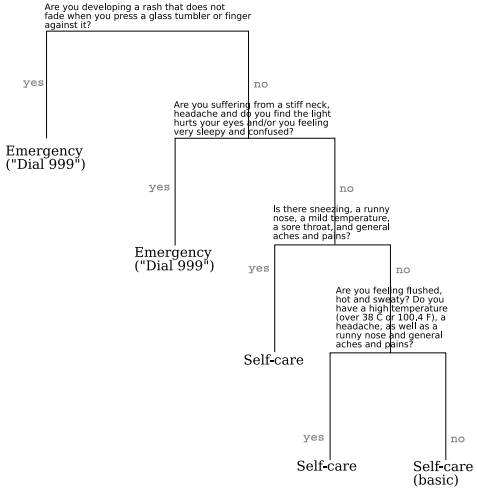
Main strengths: easy to use, easy to interpret.

Often serve as a starting point for powerful model combination and ensemble techniques: bagging, random forests

Example: NHS Direct Self-help Guide (pre-covid era)



Example: NHS Direct Self-help Guide (pre-covid era)



Decision Trees

A decision tree is a hierarchically organised structure, with each node splitting the data space into regions based on value of a single feature (attribute).

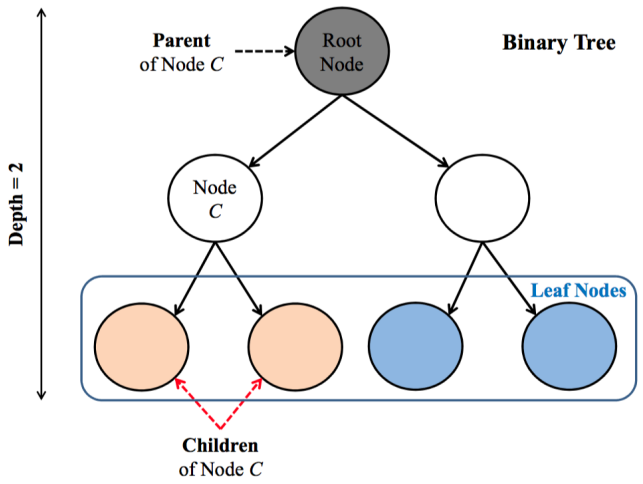
Some terminology:

Parent of a node c is the node with an arrow pointing into c.
Children of a node c are those nodes which have node c as a parent.
Root node is the top node of the tree; the only node without parents.
Leaf nodes are nodes which do not have children.
Stumps are trees with just the root node and two leaf nodes.
The **depth** of a tree is the maximal length of a path from the root node to a leaf node.

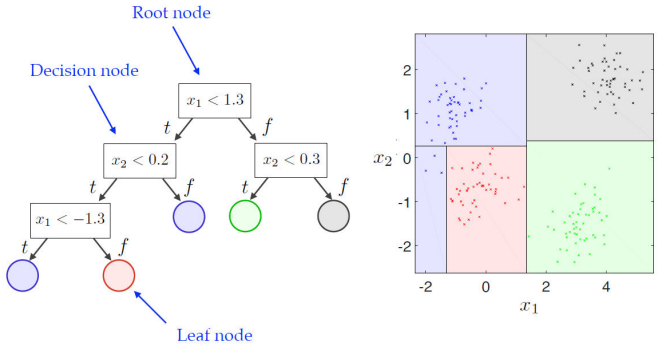
Partition of $\mathcal{X}$ into R disjoint sets $(\mathcal{R}_1, \ldots, \mathcal{R}_R)$ is determined by the leaves of the tree.

On each region $\mathcal{R}_j$ the same decision/prediction is made: $h(x) = \beta_j$ for all $x \in \mathcal{R}_j$ - typically as a majority vote of the data items associated to that leaf (classification under the 0-1 loss) or as their mean (regression under the squared loss)

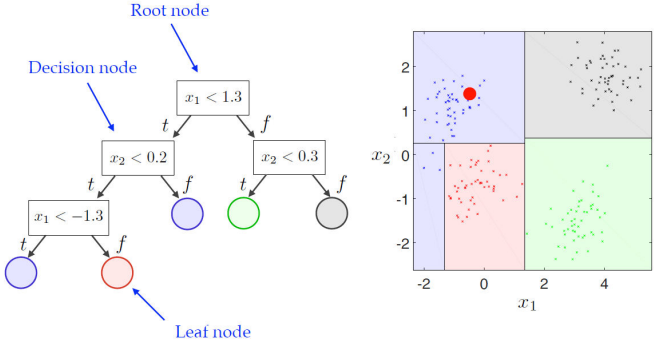
Terminology



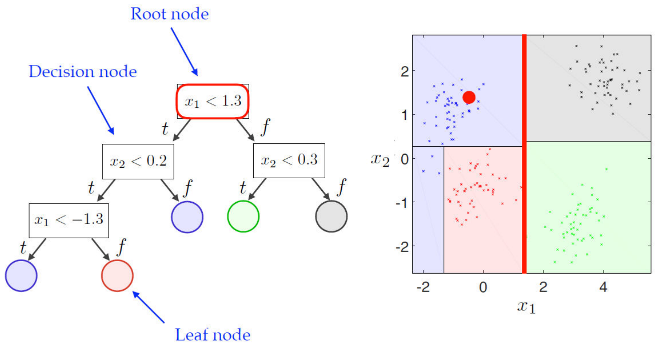
Decision tree



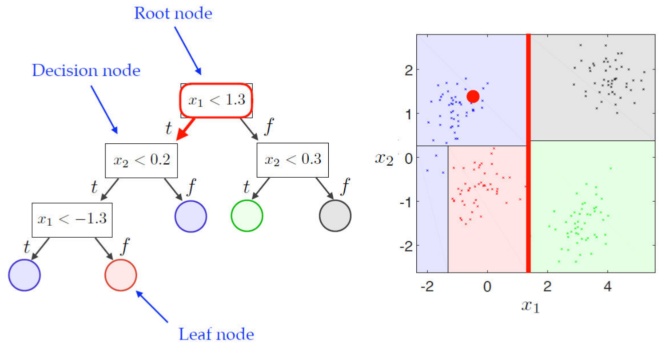
Decision tree



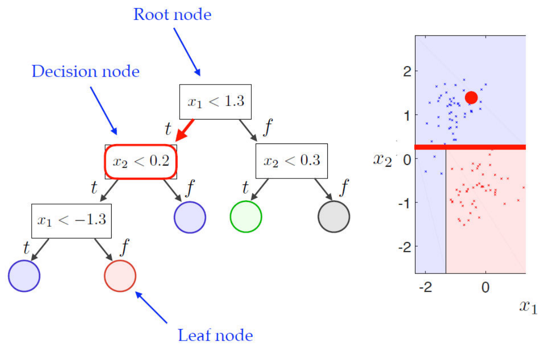
Decision tree



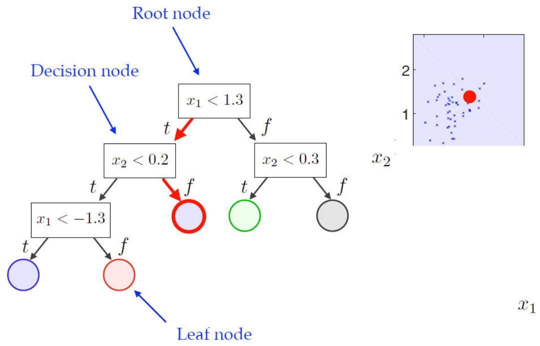
Decision tree



Decision tree



Decision tree

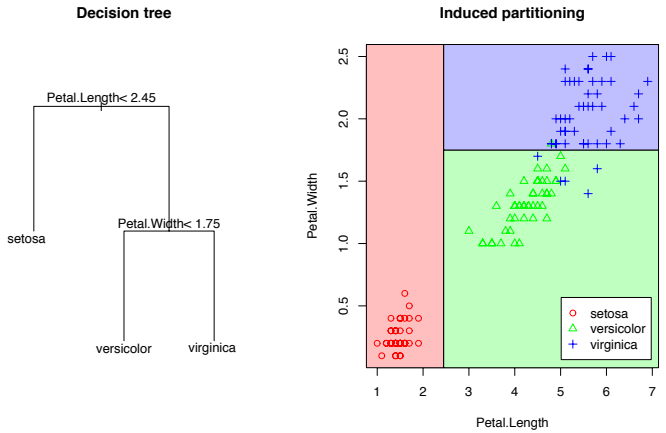


Example: Iris Data

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
4.4	3.2	1.3	0.2	setosa
5.9	3.0	5.1	1.8	virginica
6.3	3.3	6.0	2.5	virginica
5.3	3.7	1.5	0.2	setosa
5.5	2.5	4.0	1.3	versicolor
6.1	2.9	4.7	1.4	versicolor
6.1	3.0	4.9	1.8	virginica
5.7	2.8	4.5	1.3	versicolor
5.4	3.0	4.5	1.5	versicolor
4.8	3.4	1.6	0.2	setosa
4.6	3.1	1.5	0.2	setosa
4.9	3.1	1.5	0.2	setosa
6.4	2.9	4.3	1.3	versicolor
.....				

Previously seen Iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

Example: Iris Data



Partition of $\mathcal{X}$ into R disjoint sets $(\mathcal{R}_1, \ldots, \mathcal{R}_R)$ is determined by the leaves of the tree.

Parameter estimation

The prediction rule is of the form

$$h(x) = \sum_{j=1}^{R} \beta_j \mathbb{1}_{x \in \mathcal{R}_j},$$

Parameters: Partition $\{\mathcal{R}_1, \ldots \mathcal{R}_R\}$ of $\mathcal{X}$ and weights $\beta_1, \ldots, \beta_R$
Given the partition, estimating the parameters β_j via ERM is easy
For regression, under the squared loss, we have

$$\hat{\beta}_j = \frac{\sum_{i=1}^{n} y_i \mathbb{1}_{x_i \in \mathcal{R}_j}}{\sum_{i=1}^{n} \mathbb{1}_{x_i \in \mathcal{R}_j}}$$

For classification problems, under the 0-1 loss, we do a majority vote within the set $\mathcal{R}_j$

$$\hat{\beta}_j = \underset{k=1,\ldots,K}{\arg\max} \sum_{i=1}^{n} \mathbb{1}_{y_i = k} \mathbb{1}_{x_i \in \mathcal{R}_j}$$

These estimates can be regularised as well.

Partition Estimation

Ideally, would like to find partition that achieves minimal risk: lowest mean-squared error for regression or misclassification rate for classification.

Number of potential partitions is too large to search exhaustively.

'Greedy' search heuristics for a good partition:

- Start at root.
- Determine best feature and value to split.
- Recurse on children of node.
- Stop at some point.

Growth Heuristic for Regression Trees

Start with $\mathcal{X} = \mathbb{R}^p$.

For each feature $j = 1, \ldots, p$, and for each value $v \in \mathbb{R}$ that we can split on:

Split data set:

$$I_< = \{i : x_{ij} < v\} \qquad I_> = \{i : x_{ij} \geq v\}$$

Estimate parameters:

$$\beta_< = \frac{\sum_{i \in I_<} y_i}{|I_<|} \qquad \beta_> = \frac{\sum_{i \in I_>} y_i}{|I_>|}$$

Compute the quality of split, e.g., the square loss:

$$\sum_{i \in I_<} (y_i - \beta_<)^2 + \sum_{i \in I_>} (y_i - \beta_>)^2$$

Choose split, i.e., feature j and value v, with minimal loss.

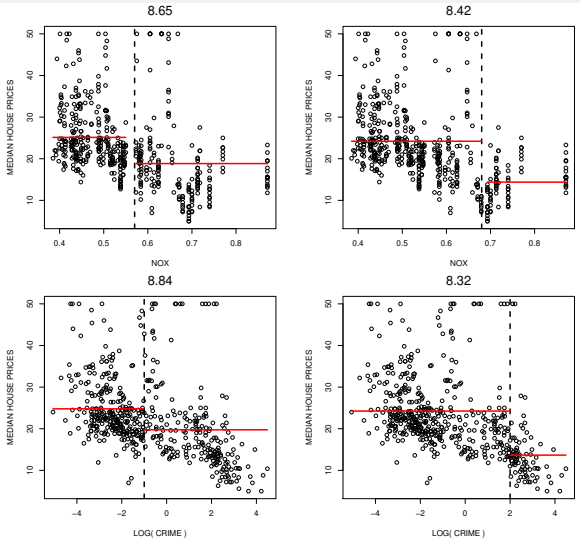
Recurse on both children, with datasets $(x_i, y_i)_{i \in I_<}$ and $(x_i, y_i)_{i \in I_>}$.

Boston Housing Data

crim	per capita crime rate by town
zn	proportion of residential land zoned for lots over 25,000 sq.ft
indus	proportion of non-retail business acres per town
chas	Charles River dummy variable
nox	nitric oxides concentration (parts per 10 million)
rm	average number of rooms per dwelling
age	proportion of owner-occupied units built prior to 1940
dis	weighted distances to five Boston employment centres
rad	index of accessibility to radial highways
tax	full-value property-tax rate per USD 10,000
ptratio	pupil-teacher ratio by town
b	1000(B - 0.63)^2 where B is the proportion of blacks by town
lstat	percentage of lower status of the population
medv	median value of owner-occupied homes in USD 1000's

Predict median house value.

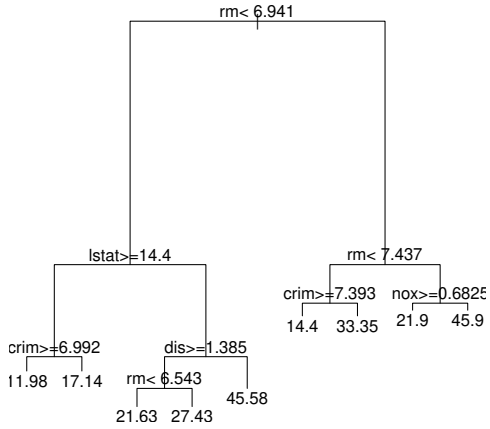
Boston Housing Data



Boston Housing Data

Overall, the best first split is on variable `rm`, average number of rooms per dwelling.

Final tree contains predictions in leaf nodes.



Growth Heuristics for Classification Trees

For binary classification, the proportion of class 1 items in a node corresponding to a region $\mathcal{R}$ is given by

$$\eta_1 = \frac{\sum_i \mathbb{1}_{y_i=1}\mathbb{1}_{x_i\in\mathcal{R}}}{\sum_i \mathbb{1}_{x_i\in\mathcal{R}}}$$

A split is good if both sides are more pure, i.e. η_1 is closer to 0 or 1.

Different measures of node impurity:

- Misclassification error: $1 - \max\{\eta_1, 1-\eta_1\}$.
- Gini impurity: $2\eta_1(1-\eta_1)$.
- Entropy: $-\eta_1 \log \eta_1 - (1-\eta_1)\log(1-\eta_1)$.

Gini and entropy preferred: differentiable and produce purer nodes.

Extension to multi-class:

- Misclassification error: $1 - \max_k \eta_k$.
- Gini impurity: $\sum_{k=1}^{K} \eta_k(1-\eta_k)$.
- Entropy: $-\sum_{k=1}^{K} \eta_k \log \eta_k$.

Stops once a node has insufficient number of items, or is pure.

Growth Heuristics for Classification Trees

Consider splitting $\mathcal{R}$ into two non-overlapping regions $\mathcal{R}_l$ and $\mathcal{R}_r$, where $\mathcal{R}_l \cup \mathcal{R}_r = \mathcal{R}$

Denote $q_l = \frac{\#\{i|x_i\in\mathcal{R}_l\}}{\#\{i|x_i\in\mathcal{R}\}}$ the proportion of the examples in $\mathcal{R}$ assigned to the left partition and $q_r = 1 - q_l$ the proportion of the examples in $\mathcal{R}$ assigned to the right partition

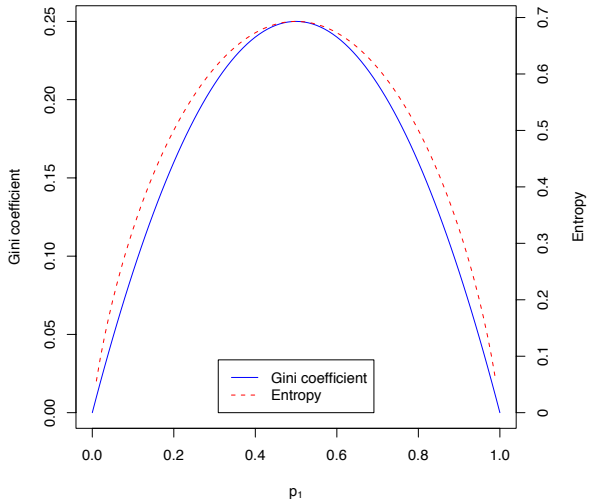
Denote η_1, $\eta_{1,l}$, $\eta_{1,r}$ the proportion of class 1 items respectively in regions $\mathcal{R}$, $\mathcal{R}_l$ and $\mathcal{R}_r$

Let $i(\eta_1)$, $i(\eta_{1,l})$ $i(\eta_{1,r})$ the corresponding impurity measures

We choose the split that maximises the change in the impurity measure

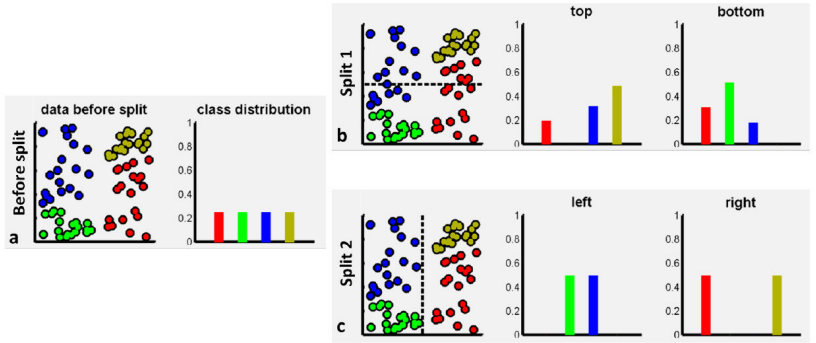
$$i(\eta_1) - q_l i(\eta_{1,l}) - q_r i(\eta_{1,r})$$

Growth Heuristics for Classification Trees



Misclassification error?

Growth Heuristics for Classification Trees



Misclassification error is 0.25 for both splits.

Adapted from Criminisi et al., Decision Forests, 2012.

Example: Pima Indians Diabetes Dataset

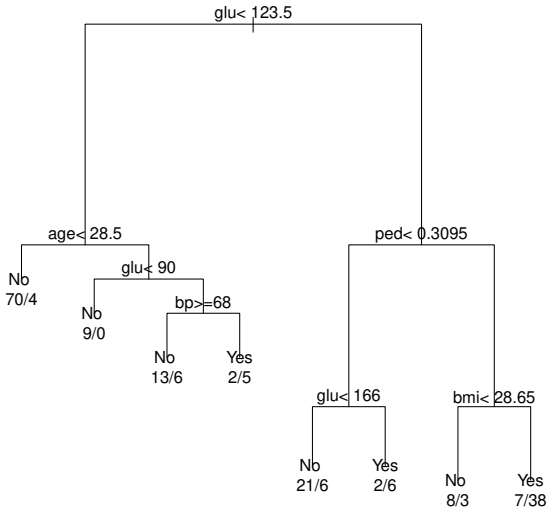
The subjects: women who were at least 21 years old, of Pima Indian heritage living near Phoenix, Arizona.

Tested for diabetes according to World Health Organisation criteria.

Features:

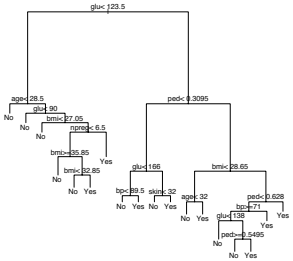
- number of pregnancies (npreg)
- plasma glucose concentration (glu)
- diastolic blood pressure (bp)
- tricep skin fold thickness (skin)
- body mass index(bbi)
- diabetes pedigree function (ped)
- age (age)

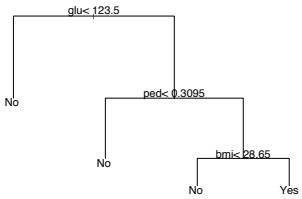
Example: Pima Indians Diabetes Dataset



Example: Pima Indians Diabetes Dataset

Two possible trees.





Model Complexity

When should tree growing be stopped?

Will need to control complexity to prevent overfitting, and in general find optimal tree size with best predictive performance.

A regularised objective

$$\widehat{R}_n(h_T) + \frac{\lambda}{n} \times \text{size}(T)$$

with size(T) the number of leaf nodes.

- Grow the tree from scratch and stop once the criterion objective starts to increase.
- First grow the full tree and prune nodes (starting at leaves), until the objective starts to increase.

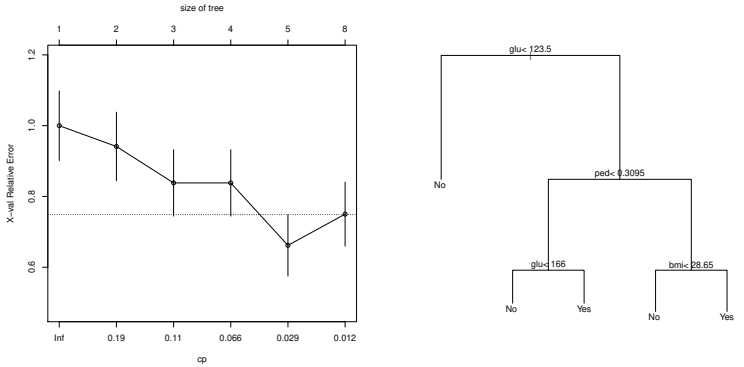
Second option is preferred as the choice of tree is less sensitive to "wrong" choices of split points and variables to split on in the first stages of tree fitting.

Use a validation set or cross-validation to determine optimal λ.

Model Complexity

Pima Indian dataset

Cross-validation risk versus regularisation parameter λ



Computational Considerations

Numerical input features x_{ij}

We could split on any feature, with any threshold
However, for a given feature, the only split points we need to consider are the n values in the training data for this feature.
If we sort each feature by these n values, we can quickly compute our metric of interest (squared loss or impurity)

This takes $O(pn \log n)$ time

Categorical input features x_{ij}

Assuming q distinct categories, there are $2^{q-1} - 1$ possible partitions we can consider.

Summary

Advantages

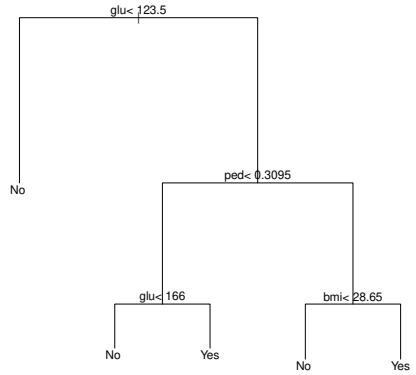
Interpretable predictions
Handles both numerical and categorical data

Disadvantages

Instability: a small change in the data may result in a very different series of splits
Lack of smoothness for regression
Often poor predictive performances

Bagging

Model Variability

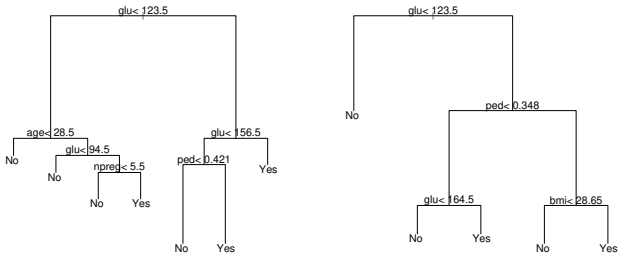


Is the tree 'stable' if training data were slightly different?

Bootstrap for Classification Trees

The bootstrap is a way to assess the variance of estimators.

Fit multiple trees, each on a bootstrapped sample. This is a data set obtained by sampling with replacement n times from training set.

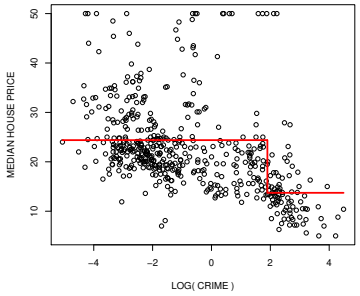


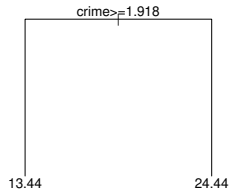
Bootstrap for Regression Trees

Regression for Boston housing data.

Predict median house prices based only on crime rate.

Use decision stump—the simplest tree with a single split at root.





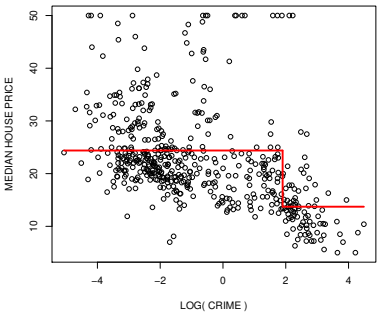
Bootstrap for Regression Trees

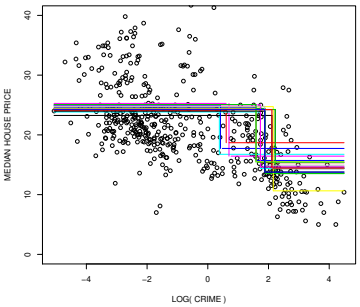
We fit a predictor $\hat{h}^{(d)}(x)$ on the data $d = \{(x_i, y_i)\}_{i=1}^n$.

Assess the variance of $\hat{h}^{(D)}(x)$ by taking $B = 20$ bootstrap samples of the original data, and obtaining bootstrap estimators

$$\hat{h}^b(x), \qquad b = 1, \ldots, B$$

Each tree $\hat{h}^b$ is fitted on the resampled data $(x_{j_i}, y_{j_i})_{i=1}^n$ where each j_i is chosen randomly from $\{1, \ldots, n\}$ with replacement.





Bagging

Bagging (**B**ootstrap **Agg**regation): average across all B trees fitted on different bootstrap samples.

For $b = 1, \ldots, B$:

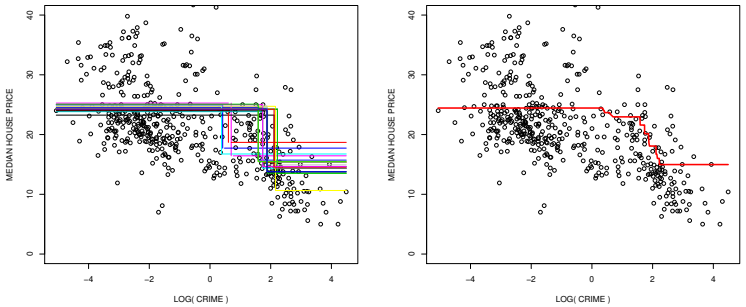
Draw indices $(j_1, \ldots, j_n)$ from the set $\{1, \ldots, n\}$ with replacement.
Fit the model, and form predictor $\hat{h}^b(x)$ based on bootstrap sample

$$(x_{j_1}, y_{j_1}), \ldots, (x_{j_n}, y_{j_n})$$

Form bagged estimator

$$\hat{h}_{Bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{h}^b(x)$$

Bagging



Bagging smooths out the drop in the estimate of median house prices.
Bagging reduces the variance of predictions, i.e. when taking expectations over a random dataset D:

$$\mathbb{E}_D\left[(\hat{h}(x) - \mathbb{E}_D[\hat{h}(x)])^2\right] \;\geq\; \mathbb{E}_D\left[(\hat{h}_{Bag}(x) - \mathbb{E}_D[\hat{h}_{Bag}(x)])^2\right]$$

Variance Reduction in Bagging

Suppose, in an ideal world, our estimators $\hat{h}^b$ are each based on different independent datasets of size n from the true joint distribution of X, Y.
The aggregated estimator would then be

$$\hat{h}_{ag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{h}^b(x) \to \bar{h}(x) = \mathbb{E}_D[\hat{h}(x)] \quad \text{a.s. as } B \to \infty$$

where expectation is with respect to datasets of size n.
The conditional risk under the squared-loss is:

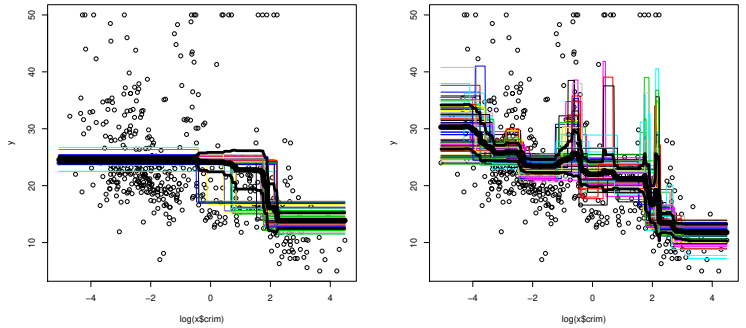
$$\begin{aligned}
&\mathbb{E}_D[(Y - \hat{h}_{ag}(X))^2 | X = x] \\
&= \mathbb{E}_D[(Y - \bar{h}(X))^2 | X = x] + \mathbb{E}_D[(\bar{h}(X) - \hat{h}_{ag}(X))^2 | X = x] \\
&\to \mathbb{E}_D[(Y - \bar{h}(X))^2 | X = x] \quad \text{as } B \to \infty.
\end{aligned}$$

Aggregation reduces the squared loss by eliminating variance of $\hat{h}(x)$.
In bagging, the bootstrap samples are not independent, but variance reduction still applies at the cost of a small increase in bias.
Bagging is most useful for flexible estimators with high variance (and low bias).

Variance Reduction in Bagging

Deeper trees have higher complexity and variance.

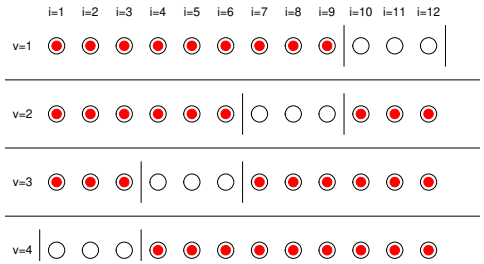
Compare bagging trees of depth 1 and 3.



Out-of-bag Test Error Estimation

How well does bagging to? Can we estimate generalisation performance, and tune hyperparameters?

Answer 1: cross-validation.



For each $v = 1, \ldots, V$,

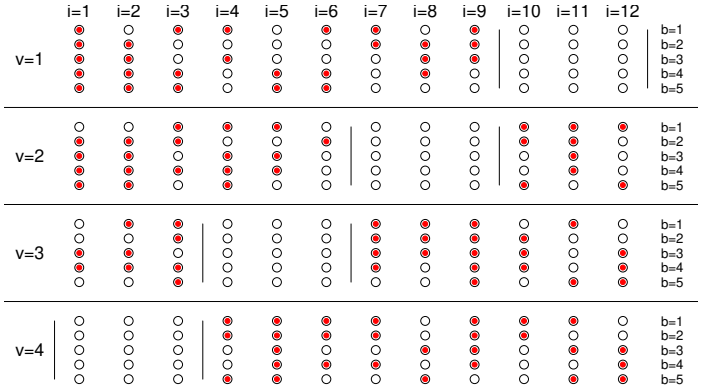
fit $\hat{h}_{Bag}$ on the training samples.

predict on validation set.

Compute the CV error by averaging the loss across all test observations.

Out-of-bag Test Error Estimation

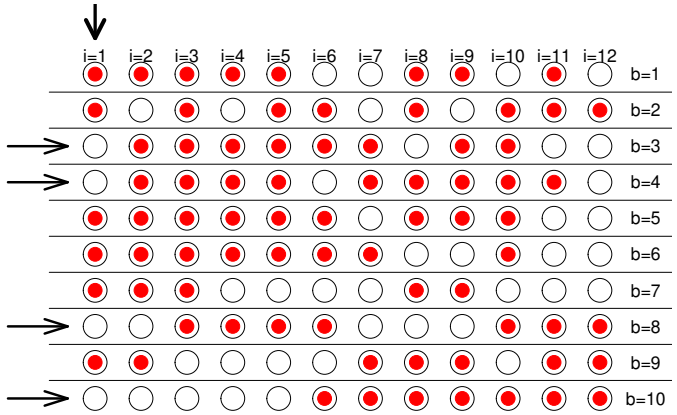
But to fit $\hat{h}_{Bag}$ on the training set for each $v = 1, \ldots, V$, we have to train on B bootstrap samples!



Answer 2: Out-of-bag test error estimation.

Out-of-bag Test Error Estimation

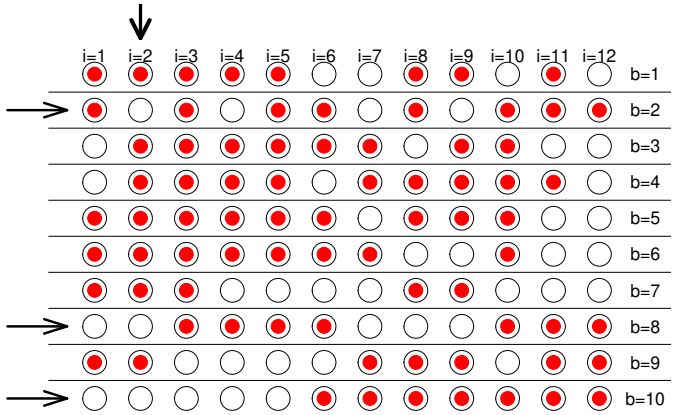
Idea: test on the “unused” data points in each bootstrap iteration to estimate the test error.



$$\hat{h}^{\text{oob}}(x_1) = \frac{1}{4} \sum_{b \in \{3,4,8,10\}} \hat{h}^b(x_1)$$

Out-of-bag Test Error Estimation

Idea: test on the “unused” data points in each bootstrap iteration to estimate the test error.



$$\hat{h}^{\text{oob}}(x_2) = \frac{1}{3} \sum_{b \in \{2,8,10\}} \hat{h}^b(x_2)$$

Out-of-bag Test Error Estimation

For each $i = 1, \ldots, n$, the out-of-bag sample is:

$$\tilde{B}_i = \{b : x_i \text{ is not in training set}\} \subseteq \{1, \ldots, B\}.$$

Construct the out-of-bag estimate at x_i:

$$\hat{h}^{\text{oob}}(x_i) \;=\; \frac{1}{|\tilde{B}_i|} \sum_{b \in \tilde{B}_i} \hat{h}^b(x_i)$$

Out-of-bag risk:

$$\widehat{R}^{\text{oob}} = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{h}^{\text{oob}}(x_i))$$

Out-of-bag Test Error Estimation

We need $|\tilde{B}_i|$ to be reasonably large for all $i = 1, \ldots, n$.

The probability π^{oob} of an observation NOT being included in a bootstrap sample $(j_1, \ldots, j_n)$ (and hence being 'out-of-bag') is:

$$\pi^{\text{oob}} = \prod_{i=1}^{n} \left(1 - \frac{1}{n}\right) \overset{n \to \infty}{\longrightarrow} \frac{1}{e} \approx 0.367.$$

Hence $\mathbb{E}[|\tilde{B}_i|] \approx 0.367B$

In practice, number of bootstrap samples B is typically between 200 and 1000, meaning that the number $|\tilde{B}_i|$ of out-of-bag samples will be approximately in the range $70 - 350$.

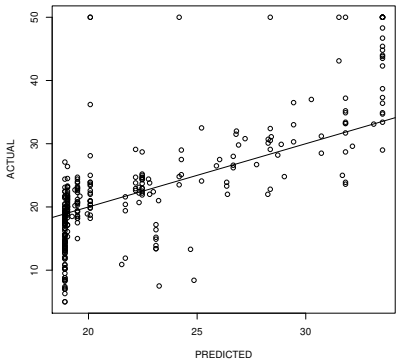
Example: Boston Housing Dataset

Apply out of bag test error estimation to select optimal tree depth and assess performance of bagged trees for Boston Housing data.

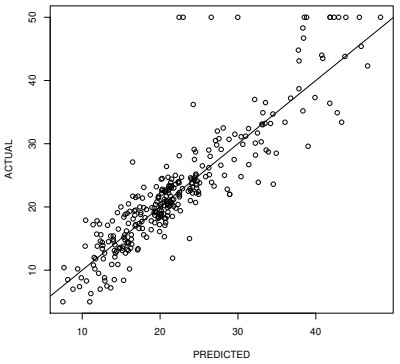
Use the entire dataset with $p = 13$ predictor variables.

Example: Boston Housing Dataset

For depth $d = 1$.



For depth $d = 10$.



Example: Boston Housing Dataset

Test error as a function of tree depth d:

tree depth d	1	2	3	4	5	10	30
single tree $\hat{h}$	60.7	44.8	32.8	31.2	27.7	26.5	27.3
bagged trees $\hat{h}_{Bag}$	43.4	27.0	22.8	21.5	20.7	20.1	20.1

Without bagging, the optimal tree depth seems to be $d = 10$.

With bagging, we could also take the depth up to $d = 30$.

Summary

Bagging reduces variance and prevents overfitting

Often improves accuracy in practice.

Bagged trees cannot be displayed as nicely as single trees and some of the interpretability of trees is lost.

Random Forests

Random Forests and Extremely Randomized Trees

Random forests are similar to bagged decision trees with a few key differences:

- For each split point, the search is not over all p variables but just over some $p_{\max} \leq p$ randomly chosen ones (where e.g. $p_{\max} = \lfloor \sqrt{p} \rfloor$)
- No pruning necessary. Trees can be grown until each node contains just very few observations (1 or 5).
- Random forests tend to produce better predictions than bagging.
- Results often not sensitive to the tuning parameter $p_{\max}$.

Even more random methods, e.g. extremely randomized trees:

- For each split point, sample $p_{\max}$ variables each with a random value to split on, and pick the best one.
- Often works even when $p_{\max}$ equals 1!

Often produce very accurate predictions, and amongst the top performing methods in machine learning competitions.

Breiman (2001), Geurts et al (2006)

Random Forests

TABLE 2
Test set misclassification error (%)

Data set	Forest	Single tree
Breast cancer	2.9	5.9
Ionosphere	5.5	11.2
Diabetes	24.2	25.3
Glass	22.0	30.4
Soybean	5.7	8.6
Letters	3.4	12.4
Satellite	8.6	14.8
Shuttle $\times 10^3$	7.0	62.0
DNA	3.9	6.2
Digit	6.2	17.1

From Breiman, Statistical Modelling: the two cultures, 2001.

Random Forests

Comparison of 179 classifiers on 121 datasets. Random forests come top with SVMs close behind.

Rank	Acc.	κ	Classifier
32.9	82.0	63.5	parRF_t (RF)
33.1	**82.3**	**63.6**	rf_t (RF)
36.8	81.8	62.2	svm_C (SVM)
38.0	81.2	60.1	svmPoly_t (SVM)
39.4	81.9	62.5	rforest_R (RF)
39.6	82.0	62.0	elm_kernel_m (NNET)
40.3	81.4	61.1	svmRadialCost_t (SVM)
42.5	81.0	60.0	svmRadial_t (SVM)
42.9	80.6	61.0	C5.0_t (BST)
44.1	79.4	60.5	avNNet_t (NNET)

From Delgado et al, 2014

Variable "Importance"

Tree ensembles have better performance, but decision trees are more interpretable.

How to interpret a forest of trees ?

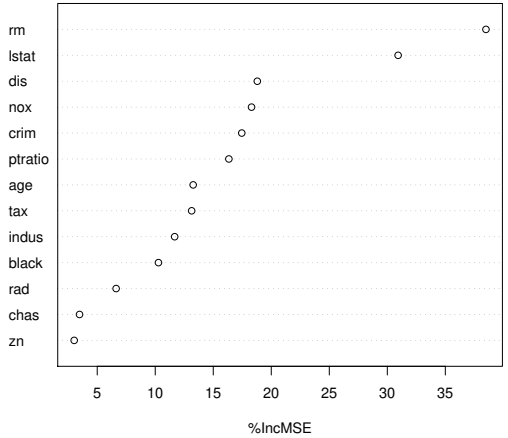
Idea: denote by $\hat{e}$ the out-of bag estimate of the loss when using the original data samples. For each variable $j \in \{1, \ldots, p\}$,

permute randomly the j-th predictor variable to generate a new set of samples $(\tilde{X}_1, Y_1), \ldots, (\tilde{X}_n, Y_n)$, i.e., $\tilde{X}_{ij} = X_{\tau(i)j}$, for a permutation τ.

compute the out-of-bag estimate $\hat{e}_j$ of the prediction error with these new samples.

A measure of importance of variable j is then $\hat{e}_j - \hat{e}$, the increase in error rate due to a random permutation of the j-th variable.

Example for Boston Housing data.



Ensemble Methods

Bagging and random forests are examples of ensemble methods, where predictions are based on an ensemble of many individual predictors.

Many other ensemble learning methods: boosting, stacking, mixture of experts, Bayesian model averaging etc.

Often gives significant boost to predictive performance.

Summary

Advantages of random forests

- Easy to use
- Fast
- State-of-the-art for many application
- Particularly good for small/medium size datasets
- Requires little tuning

Disadvantages

- Typically worse than deep learning on huge datasets
- Limited Interpretability

Statistical Machine Learning
Hilary Term 2021

François Caron
Department of Statistics
University of Oxford

Slide credits and other course material can be found at:
https://canvas.ox.ac.uk/courses/65441

Boosting

Boosting

Binary classification

Weak classifiers $h_t : \mathcal{X} \to \{-1, 1\}$ in some base hypothesis class $\mathcal{H}$, for $t = 1, \ldots, T$

Simple classifiers with few parameters to learn (e.g. decision stump), typically with high bias and small variance

Boosting prediction rule

$$h(x) = \text{sign}(f(x))$$

where

$$f(x) = \sum_{t=1}^{T} \beta_t h_t(x).$$

Weighted combination of weak classifiers

Parameters to estimate: $(\beta_t, h_t)_{t=1,\ldots,T}$

Adaboost Algorithm

Dataset $d = \{(x_i, y_i)\}$, where $y_i \in \{+1, -1\}$
Initialise weights $\overline{w}_{i,1} = \frac{1}{n}$ for every training sample $i = 1, \ldots, n$
For $t = 1$ to T

Train the weak classifier h_t using current weights $\overline{w}_{i,t}$, by minimising

$$\widehat{h}_t = \arg\min_{h_t \in \mathcal{H}} R_{\overline{w}_t}(h_t)$$

where $R_{\overline{w}_t}(h_t) = \sum_{i=1}^{n} \overline{w}_{i,t} \mathbb{1}_{y_i \neq h_t(x_i)}$ is the weighted classification error.

Compute contribution for this classifier: $\widehat{\beta}_t = \frac{1}{2} \ln \frac{1-\widehat{\epsilon}_t}{\widehat{\epsilon}_t}$ where $\widehat{\epsilon}_t = R_{\overline{w}_t}(\widehat{h}_t)$.

Update weights on training points

$$\overline{w}_{i,t+1} \propto \overline{w}_{i,t} e^{-\widehat{\beta}_t y_i \widehat{h}_t(x_i)}$$

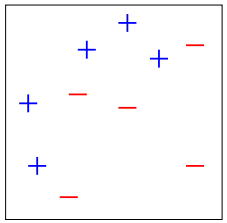
and normalise them such that $\sum_i \overline{w}_{i,t+1} = 1$.

Output the final classifier

$$\widehat{h}(x) = \text{sign}\left(\sum_{t=1}^{T} \widehat{\beta}_t \widehat{h}_t(x)\right)$$

Example

$n = 10$ data examples and $p = 2$ features

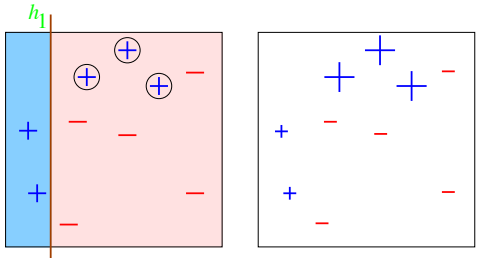


The data points are clearly not linearly separable

In the beginning, all data points have equal weights (weights are indicated by the size of the data markers “+” or “-”)

Base class $\mathcal{H}$ is the class of decision stumps: trees with a single split and two leaf nodes, classifying data based on a single attribute (x_{i1} or x_{i2})

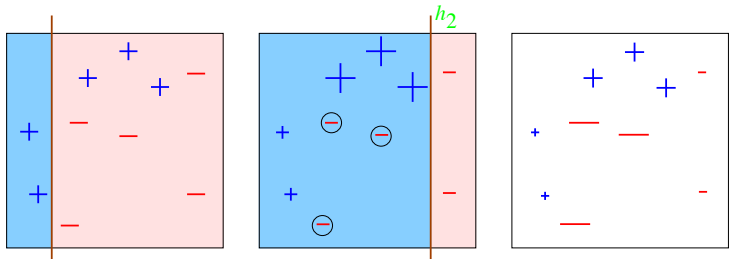
Round 1: $t = 1$



3 misclassified (with circles): $\widehat{\epsilon}_1 = 0.3 \to \widehat{\beta}_1 = 0.42$.

Weights recomputed; the 3 misclassified data points receive larger weights

Round 2: $t = 2$

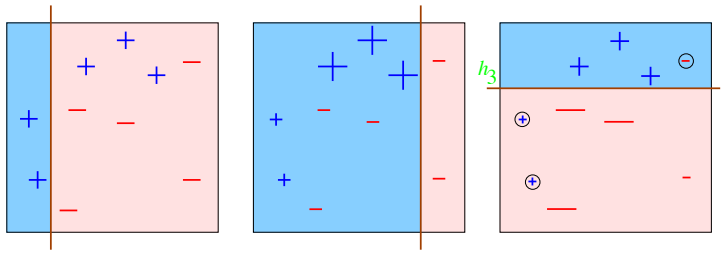


3 misclassified (with circles): $\widehat{\epsilon}_2 = 0.21 \rightarrow \widehat{\beta}_2 = 0.65$.
Note that $\widehat{\epsilon}_2 \neq 0.3$ as those 3 data points have weights less than $1/10$

3 misclassified data points get larger weights

Data points classified correctly in both rounds have small weights

Round 3: $t = 3$

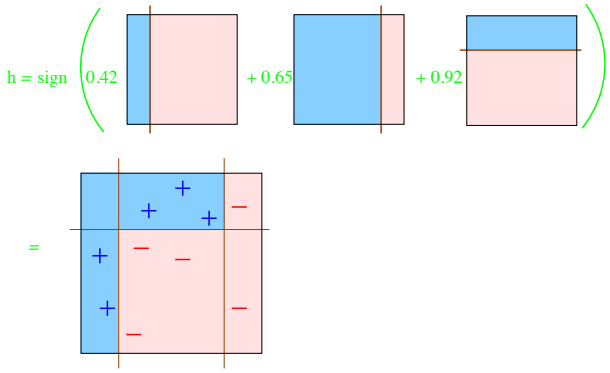


3 misclassified (with circles): $\widehat{\epsilon}_3 = 0.14 \rightarrow \widehat{\beta}_3 = 0.92$.

Previously correctly classified data points are now misclassified, hence our error is low; what's the intuition?

> Since they have been consistently classified correctly, this round's mistake will hopefully not have a huge impact on the overall prediction

Final classifier: combining 3 classifiers



All data points are now classified correctly!

Why AdaBoost works?

Classifier

$$h(x) = \text{sign}(f(x)) = \begin{cases} 1 & \text{if } f(x) > 0 \\ -1 & \text{if } f(x) < 0 \end{cases}$$

where f is the discriminant function.

Under the 0-1 loss

$$L(y, h(x)) = \begin{cases} 0 & \text{if } yf(x) > 0 \\ 1 & \text{if } yf(x) < 0 \end{cases}$$

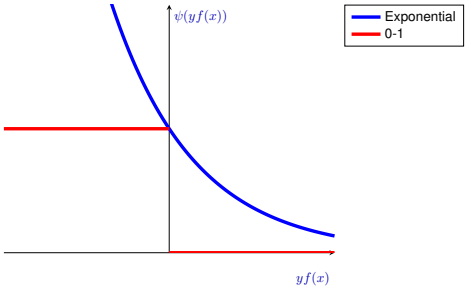
The discriminant function $f(x)$ and the target label y should have the same sign to avoid a loss of 1.

Surrogate Exponential loss

As we discussed in the lectures on linear models, the $0-1$ loss is non-convex and difficult to optimise.

Surrogate Exponential Loss

$$\widetilde{L}(y, h(x)) = \psi(yf(x)) = e^{-yf(x)}$$



ERM under Surrogate Exponential loss

Adaboost aims at minimising the empirical risk under the surrogate exponential loss

$$\frac{1}{n}\sum_{i=1}^{n} e^{-y_i f(x_i)} = \frac{1}{n}\sum_{i=1}^{n} e^{-y_i(\sum_{t=1}^{T}\beta_t h_t(x_i))}$$

Direct minimisation is not possible, and Adaboost uses a stagewise/greedy approach: (forward stagewise additive modelling)

Initialise $\widehat{f}_0(x) = 0$.
At iteration $t = 1, \ldots, T$, add a new weighted weak learner into the model:

$$\begin{aligned}(\widehat{\beta}_t, \widehat{h}_t) &= \underset{\beta_t\in\mathbb{R}, h_t\in\mathcal{H}}{\arg\min} \frac{1}{n}\sum_{i=1}^{n} \exp\left(-y_i\left(\widehat{f}_{t-1}(x) + \beta_t h_t(x)\right)\right)\\ \widehat{f}_t(x) &= \widehat{f}_{t-1}(x) + \widehat{\beta}_t\widehat{h}_t(x)\end{aligned}$$

Let's prove this is equivalent to the Adaboost algorithm update

We have

$$\sum_{i=1}^{n} \exp\left(-y_i\left(\widehat{f}_{t-1}(x_i) + \beta_t h_t(x_i)\right)\right) = \sum_{i=1}^{n} w_{i,t}\exp\left(-y_i\beta_t h_t(x_i)\right)$$

where $w_{i,t} = e^{-y_i\widehat{f}_{t-1}(x_i)}$.
Note that the Adaboost weights are $\overline{w}_{i,t} = w_{i,t}/(\sum_j w_{j,t})$.
We have $y_i h_t(x_i) \in \{1, -1\}$, hence

$$\begin{aligned}\sum_{i=1}^{n} w_{i,t}\exp\left(-y_i\beta_t h_t(x_i)\right) &= \sum_{i=1}^{n} w_{i,t}e^{\beta_t}\mathbb{1}_{y_i\neq h_t(x_i)} + \sum_{i=1}^{n} w_{i,t}e^{-\beta_t}\mathbb{1}_{y_i=h_t(x_i)}\\ &= (e^{\beta_t} - e^{-\beta_t})\sum_{i=1}^{n} w_{i,t}\mathbb{1}_{y_i\neq h_t(x_i)} + e^{-\beta_t}\sum_{i=1}^{n} w_{i,t}\\ &= \left(\sum_{i=1}^{n} w_{i,t}\right)\left((e^{\beta_t} - e^{-\beta_t})\sum_{i=1}^{n}\overline{w}_{i,t}\mathbb{1}_{y_i\neq h_t(x_i)} + e^{-\beta_t}\right)\end{aligned}$$

Minimising wrt h_t gives Step 1 of Adaboost

$$\widehat{h}_t = \underset{h_t\in\mathcal{H}}{\arg\min}\sum_{i=1}^{n}\overline{w}_{i,t}\mathbb{1}_{y_i\neq h_t(x_i)}$$

For β_t, we need to minimise

$$(e^{\beta_t} - e^{-\beta_t})\widehat{\epsilon}_t + e^{-\beta_t}$$

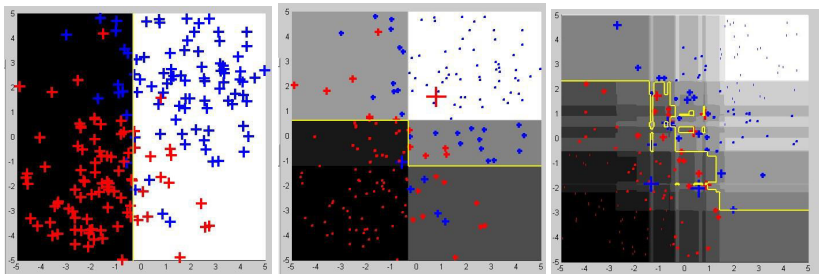
where $\widehat{\epsilon}_t = \sum_{i=1}^{n}\overline{w}_{i,t}\mathbb{1}_{y_i\neq\widehat{h}_t(x_i)}$.

Taking the derivative and setting to 0 gives

$$\widehat{\beta}_t = \frac{1}{2}\log\frac{1-\widehat{\epsilon}_t}{\widehat{\epsilon}_t}$$

which is exactly Step 2 of Adaboost.

AdaBoost with decision stumps



The degree of blackness represents the confidence in the red class. The size of datapoints represents their weight. Decision boundary in yellow.
Left: After 1 iteration, Middle: After 3 iterations, Right: After 120 iterations.

Example from Murphy, p.560; generating script written by R.Stapenhurst

Example: Netflix

The Netflix Prize

http://www.netflixprize.com

Training data is a set of users and past ratings (1 to 5 stars).

Construct a classifier that predicts user rating for unrated movies.

Winning team (BellKor's Pragmatic Chaos) employed boosting. They received 1M$ in 2009 for a 10.06% improvement.

Base learners

We can use other base learners than decision stumps, as long as we are minimising the weighted classification error

Typically use a simple, high bias classifier, whose parameters are easy to estimate

Other surrogate loss functions

Possible to use other surrogate loss functions ψ, aiming to minimise

$$\frac{1}{n}\sum_{i=1}^{n}\psi\left(y_i\left(\sum_{t=1}^{T}\beta_t h_t(x_i)\right)\right)$$

The ψ-boosting algorithm proceeds as follows

Initialise $\widehat{f}_0(x) = 0$.
At iteration $t = 1, \ldots, T$, add a new weighted weak learner into the model:

$$\begin{aligned}(\widehat{\beta}_t, \widehat{h}_t) &= \underset{\beta_t\in\mathbb{R}, h_t\in\mathcal{H}}{\arg\min}\ \frac{1}{n}\sum_{i=1}^{n}\psi\left(y_i\left(\widehat{f}_{t-1}(x) + \beta_t h_t(x)\right)\right)\\ \widehat{f}_t(x) &= \widehat{f}_{t-1}(x) + \widehat{\beta}_t\widehat{h}_t(x)\end{aligned}$$

If minimisation not tractable, perform a gradient descent step instead

If $\psi(x) = \log(1 + e^{-x})$, this is known as logit-boost.

L_2-Boosting

Can also use boosting for regression

Under the squared-loss, known as L_2-Boosting

For example, use regression trees as weak learner

New weak learners are obtained by fitting the residuals:

$$L\left(y_i, \widehat{f}_{t-1}(x_i) + \beta_t h_t(x_i)\right) = (y_i - \widehat{f}_{t-1}(x_i) - \beta_t h_t(x_i))^2 = L(\underbrace{y_i - \widehat{f}_{t-1}(x_i)}_{r_{i,t}}, \beta_t h_t(x_i))$$

L_2-Boosting algorithm

Initialise $\widehat{f}_0(x) = 0$.
For $t = 1, \ldots, T$, compute current residuals

$$r_{i,t} = y_i - \widehat{f}_{t-1}(x_i),$$

and fit the residuals $\{(x_i, r_{i,t})\}_{i=1}^n$ to obtain the term $\widehat{\beta}_t \widehat{h}_t(x)$ to be added to the expansion.

Boosting: Summary

Boosting is a bias-reduction technique, as opposed to bagging.

Resistant to overfitting (the testing error typically stays flat for a large number of iterations - but will eventually go up).

Can be understood as functional gradient descent, leading to a generic framework called gradient boosting.

Further reading: Hastie et al, Chapter 10; Murphy, Section 16.4.