# Introduction to Machine Learning

Matteo Amabili

matteo.amabili@unibo.it

# Course Outline

1. Introduction
2. Preprocessing Data
3. Linear Regression
4. Classification Model: Logistic Regression
5. How to Evaluate a Model
6. Tree Based Method
7. Unsupervised learning: Clustering Methods

**In every chapter we will have some example in python**

# Prerequisites

**Basic** knowledge of:

- **Statistic**
- **Probability**
- **Linear Algebra**: matrix calculus
- **Calculus**: derivatives, gradient, ecc…

**No prerequisites for programming**

# Recommended reading

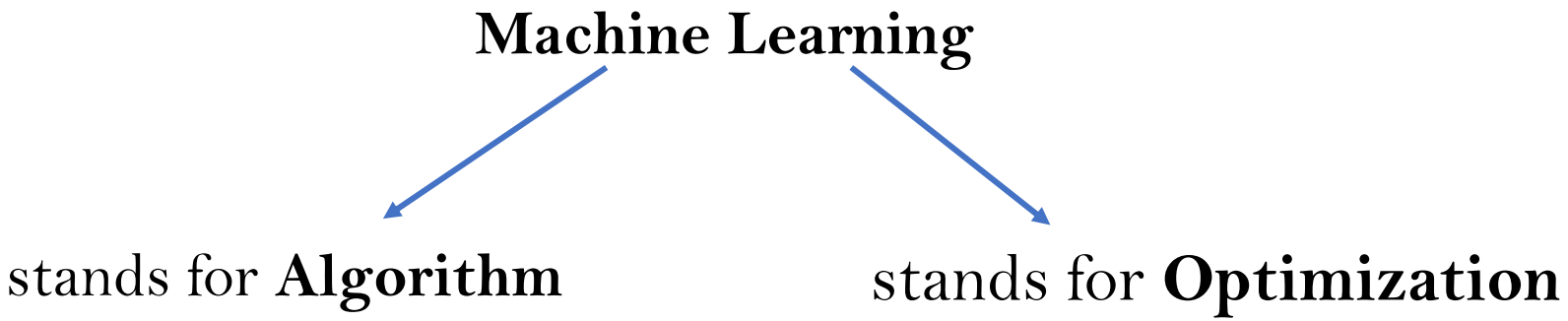
- **James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning* (Vol. 112, p. 18). New York: springer.**

- Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2, pp. 1-758). New York: springer.

- Rogers, S., & Girolami, M. (2016). *A first course in machine learning*. Chapman and Hall/CRC.

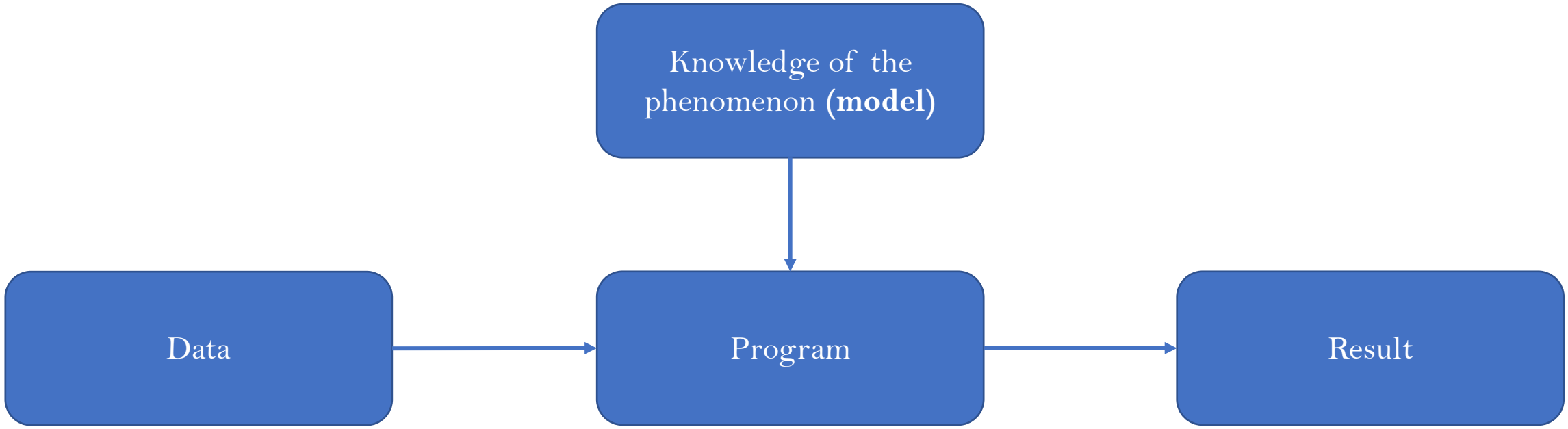
- Many resources available online, stackoverflow, standford and MIT courses etc…

# What is Machine Learning



# A paradigm shift: from knowledge to data

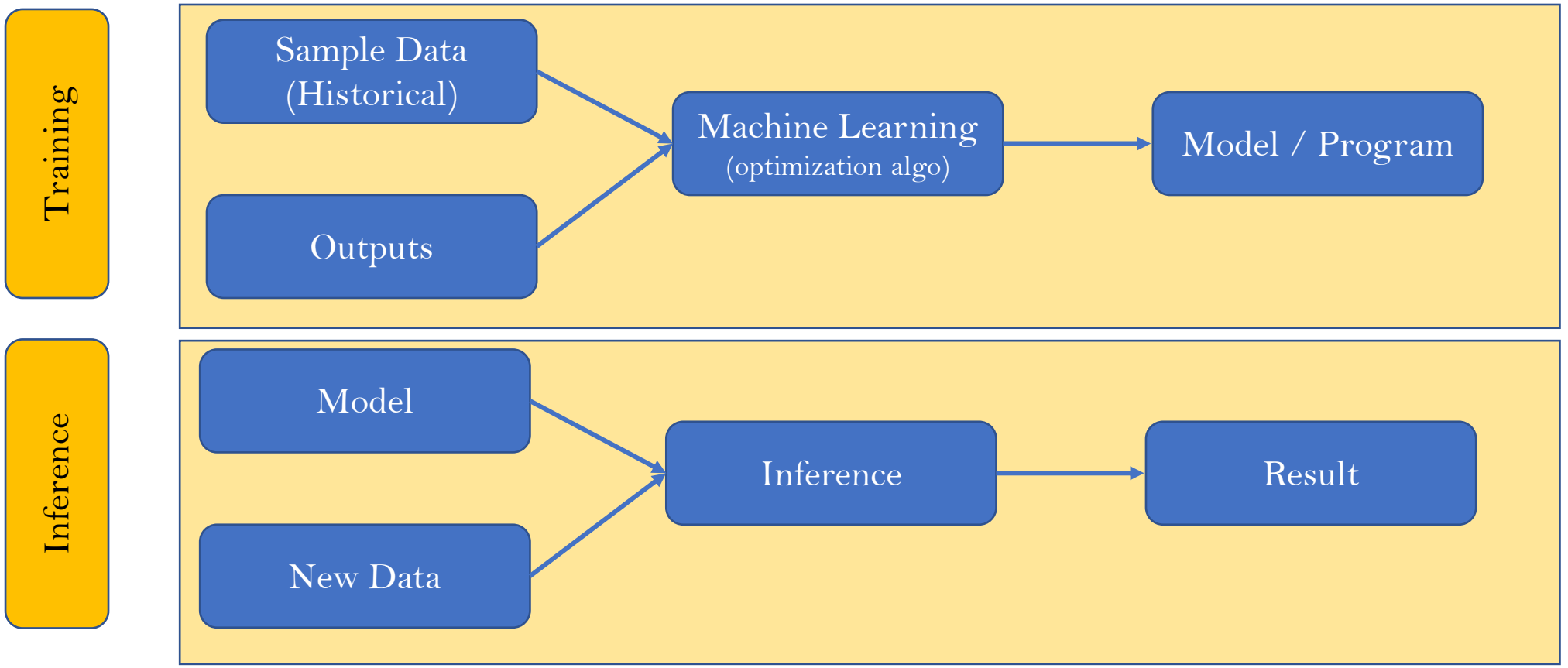
**Traditional Programming:**




**Data are used to validate/test assumption** of the pre-existing model

# A paradigm shift: from knowledge to data

**Machine Learning:**




Data are used to build (train) a model used to get results (predictions) during inference →

**where is the knowledge about the process ? (see next slides)**

# A paradigm shift: from knowledge to data

Write a program that output if a number is even or odd:

**Traditional Programming:**

A number is even if the remainder of the division by 2 is 0

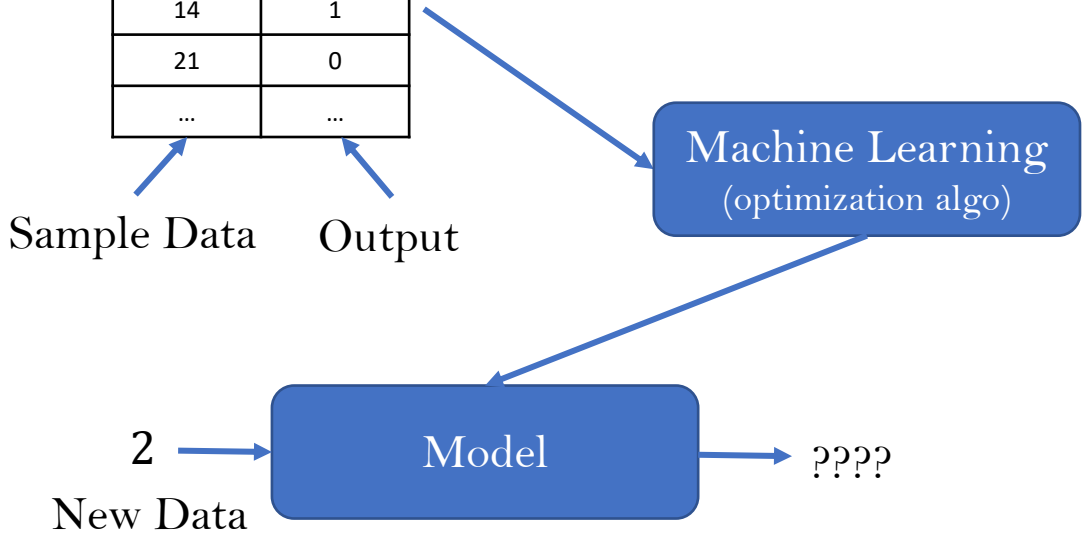
```
In [9]: def is_even(number):
   ...:     if(number%2==0):
   ...:         return True
   ...:     else:
   ...:         return False
   ...:
   ...:

In [10]: is_even(11)
Out[10]: False

In [11]: is_even(10)
Out[11]: True
```

**Machine Learning:**

| Number | Is_even |
|---|---|
| 11 | 0 |
| 14 | 1 |
| 21 | 0 |
| ... | ... |

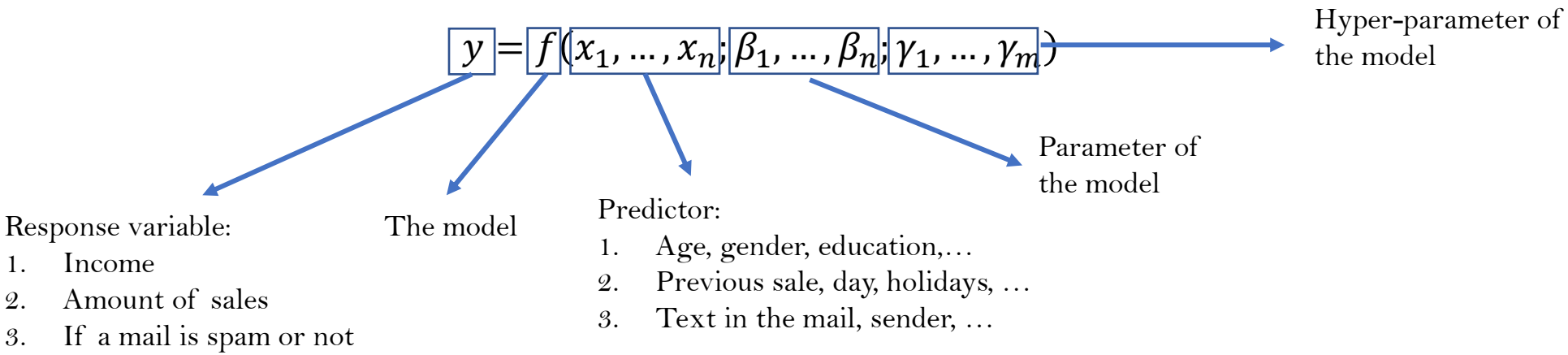



This approach will not work → **no 'simple' pattern in the data**

# Machine Learning Foundamentals

Which Problems do ML solves ?

ML Aims to learn (approximate) a model $f$, that relate the features $x_i$ to the response $y$



# Machine Learning Foundamentals

$$y = f(x_1, \dots, x_n; \beta_1, \dots, \beta_n; \gamma_1, \dots, \gamma_m)$$

How do we learn $f$?

given a dataset $D = \{ Y_i, \boldsymbol{X}_i \}$ $i = 1, \dots, n$ where $n$ are the rows of $D$, $\boldsymbol{X}_i$ is the matrix of the predictor variables and $Y_i$ is the response variable, **our aim is to find "the best" model *f***

How to formalize the expression **"the best"?** → **Optimization**

We fix the shape for the model (e.g. linear model, tree based model, ecc..) and solve the following optimization problem:

$\beta, \gamma = \ argmin_{\beta,\gamma}\ L(Y, f(X; \beta; \gamma))$ usually simplified* as $\beta = \ argmin_{\beta}\ L(Y, f(X; \beta; \gamma^*))$

$L$ is called the **Loss Function** and measures how well our model approximate the data

* Having fixed $\gamma$ with other methods, e.g. cross-validation (see chapter 5)

# Machine Learning Foundamentals

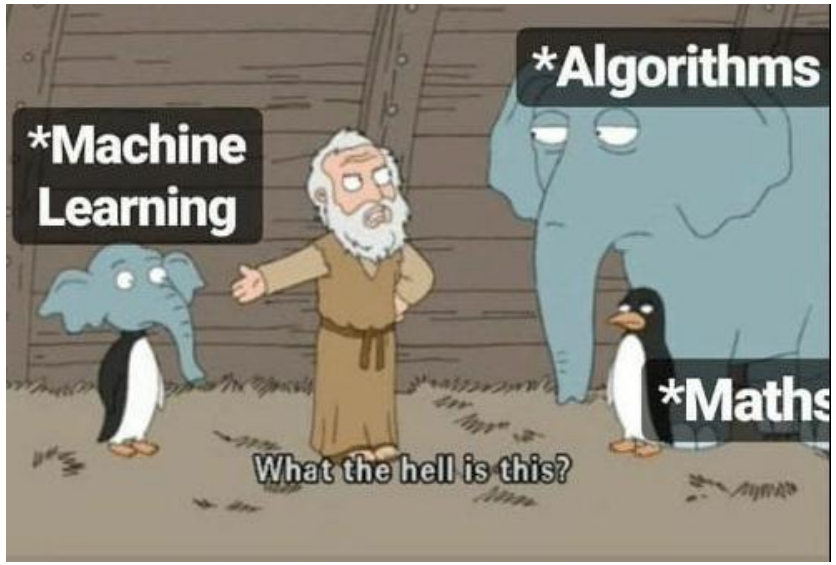
$$y = f(x_1, \dots, x_n; \beta_1, \dots, \beta_n; \gamma_1, \dots, \gamma_m)$$

How do we learn $f$?

$$\beta = \ argmin_\beta \ L(Y, f(X; \beta; \gamma^* )) \quad (1)$$

In this course we will:

- Study several shape for $f$
- Understand the parameter & hyper-parameter of the $f$
- Define suitable loss function L for the problem at hand
- Understand how to compute hyperparameter
- Understand how to solve the minimization problem in (1)
- Understand the result in term of business insight



# Type of Problem

## Supervised

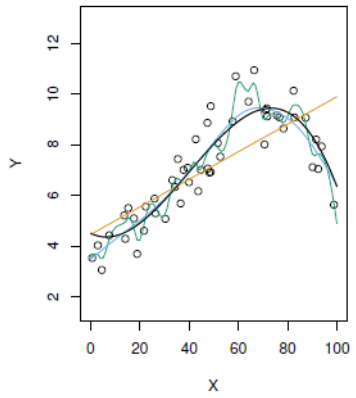
We observe both $x_i, y$ → fit model that relate predictor to response

## Unsupervised

We observe only $x_i$ → understand the relationships between the variables

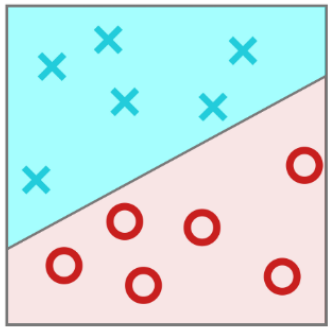
### Regression

$y \in R$

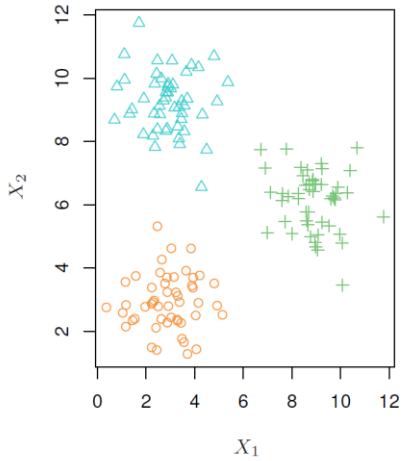



### Classification

$y \in \{0,1, \ldots, m\}$

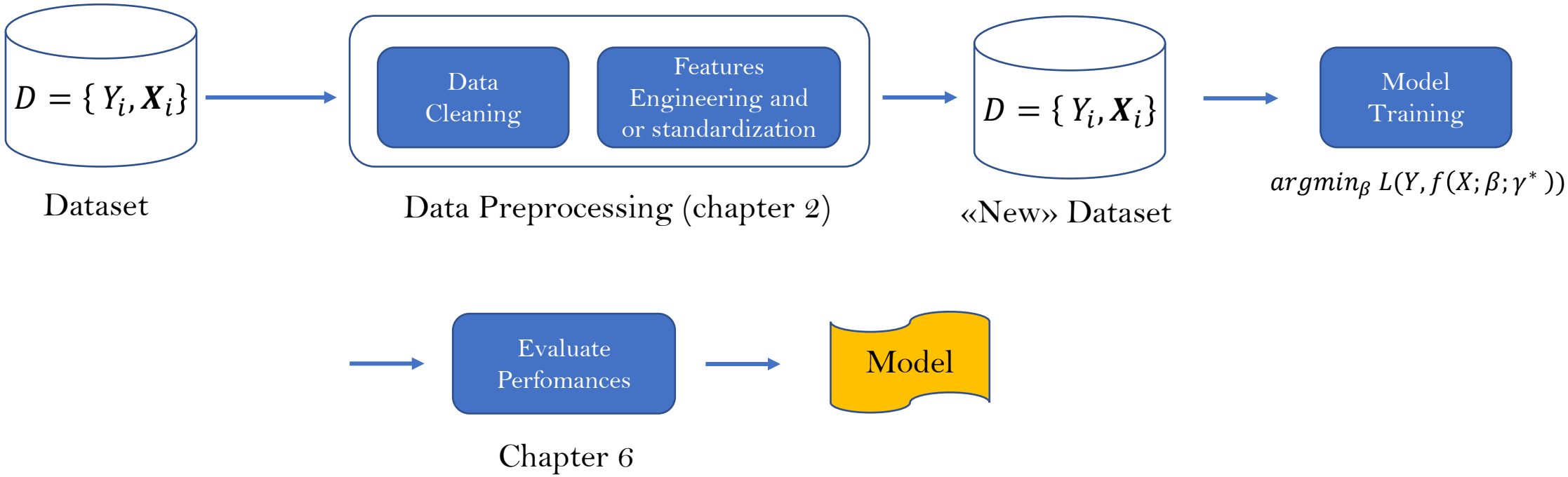


### Clustering Methods



# Machine Learning Pipeline

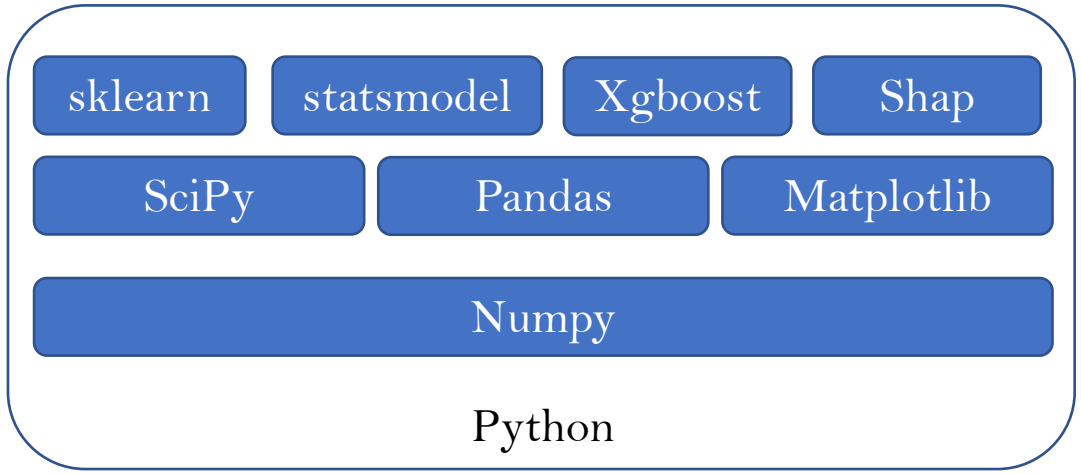
**Features Engineering is usually driven by business knownledge** (look slide 7)



# Python

Python is the most diffuse programming languages for ML applications

https://www.python.org/



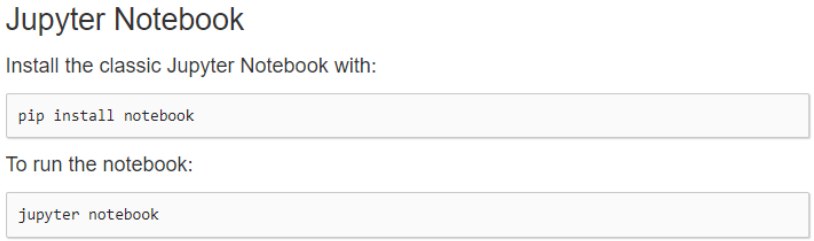

- **Numpy**: support for multidimensional array and mathematical function
- **SciPy**: scientific computing in python
- **Pandas**: data/table manipulation and analysis
- **Matplotlib**: Plotting graph
- **Sklearn, statsmodel, (Xgboost)**: training of ML model
- **(Shap)**: variables explanation for complex model

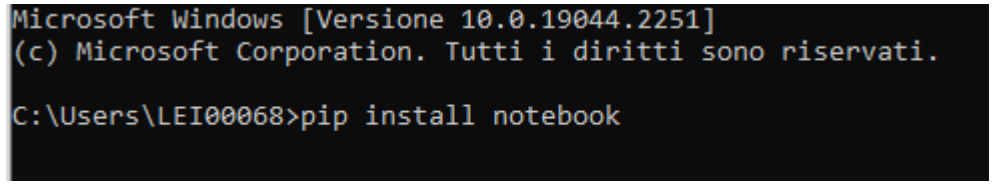
# Install python

- Go to the website https://www.python.org/downloads/



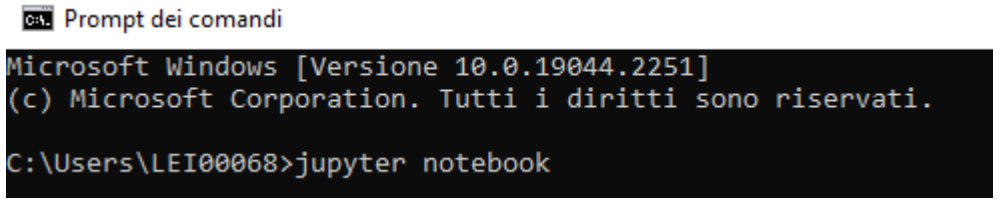
- I suggest to install version 3.10 or 3.9 (usually with the latest is more difficult to install package)
  - remember to add python to the $PATH env variables (it will ask)
    - Install Jupyter notebook/lab at https://jupyter.org/install




**pip is the packege manager of python**

Pip install <package_name> == <package_version>
Pip uninstall <package_name>



# Python hands on

Notebook **Python_basics**

Example of Datasets

# Chapter 2: Data Preprocessing

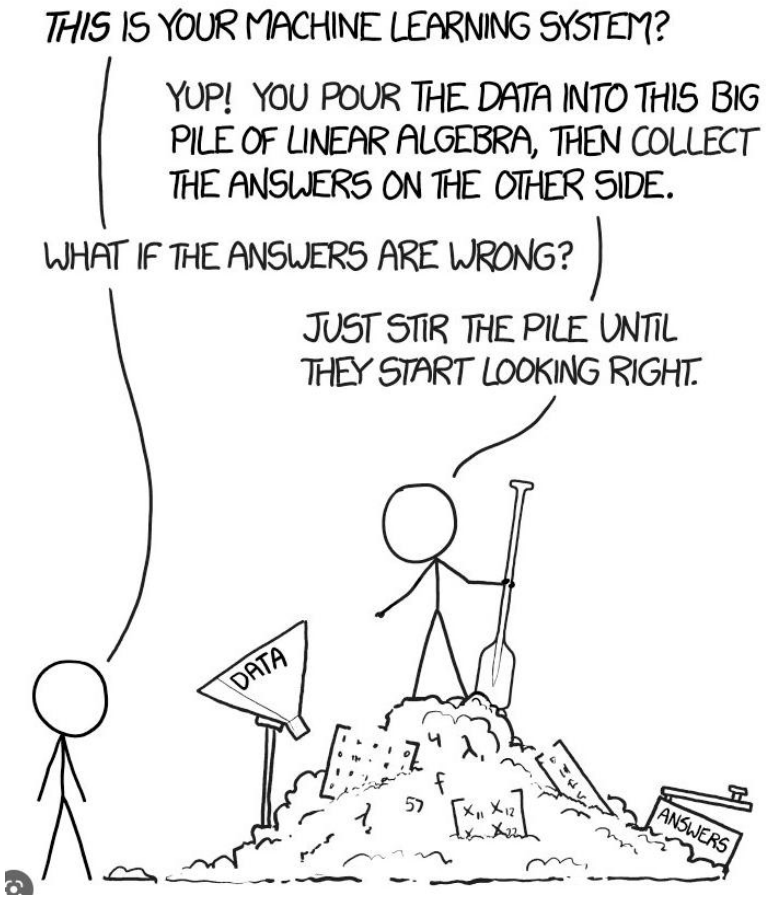
# Motivation

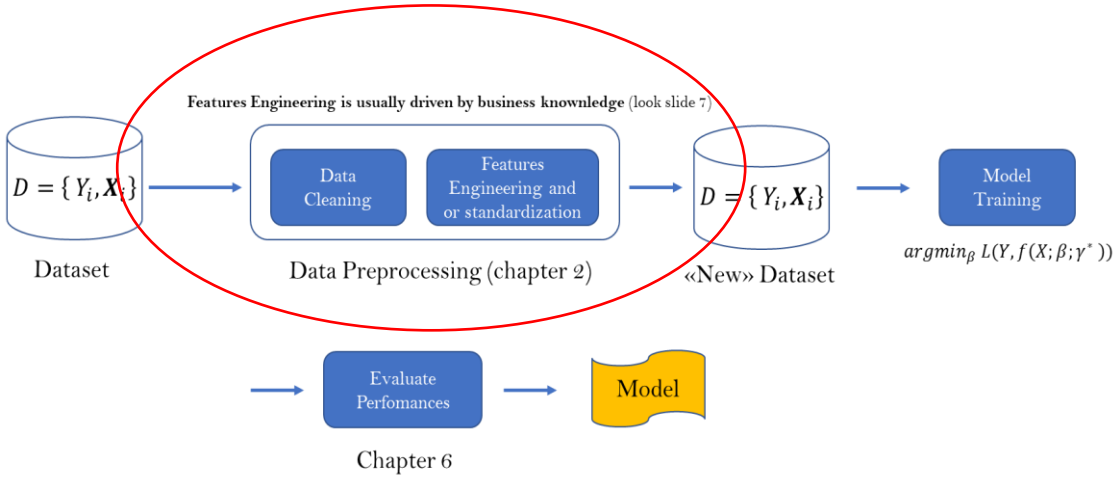
- Raw data rarely comes in the form and shape that is necessary for the optimal performance of a learning algorithm.
- Real-world data is often dirty containing outliers, missing values, wrong data types, irrelevant features, or non-standardized data.
- Field of Data Preprocessing is really problem depend, **these notes report only limited examples**

The success of a machine learning algorithm highly depends on the quality of the data

Transforming raw data into a useful format is an essential stage in the machine learning process.

# Motivation



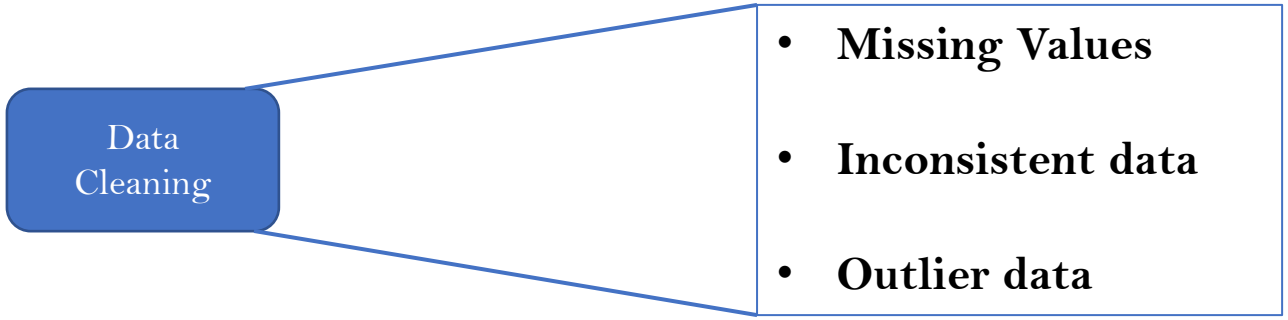




We are here and will talk about:

- Data Cleaning
- Featuere Eng. and/or Standardization

pandas

# Data Cleaning



# Data Cleaning : Missing values

Data often has a lot of missing values due to failure in the recording process

**Solution 1**

Deleting the rows or columns having **null** values:

- If a columns have more than a given % of the rows as null can be dropped.
- The rows having one value as null can also be dropped.

Pandas can be used to easily handle null values (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.dropna.html)

| Pros | Cons |
| --- | --- |
| • Create a robust model | • If null are common in the dataset, a lot of information are removed<br>• Removing a rows with a null cause the remove of all the good data of these rows |

# Data Cleaning : Missing values

Data often has a lot of missing values due to failure in the recording process

**Solution 2**

Imputing **null** values with statistics about that columns (e.g. mean/median)

Pandas can be used to easily handle to impute null values (https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html)

| Pros | Cons |
| --- | --- |
| • No data loss, use all rows and columns | • Introduce synthetic data that may reduce model generalization<br>• Imputing method does not consider relation between features |

# Data Cleaning : Missing values

Data often has a lot of missing values due to failure in the recording process

**Solution *3***

Use model that automatically handle missing values e.g. XGBoost

| Pros | Cons |
| --- | --- |
| • Null values are handled as “missing information” | • Only few model can handle null values |

# Data Cleaning : Inconsistent data

Data often shows inconsistent values:

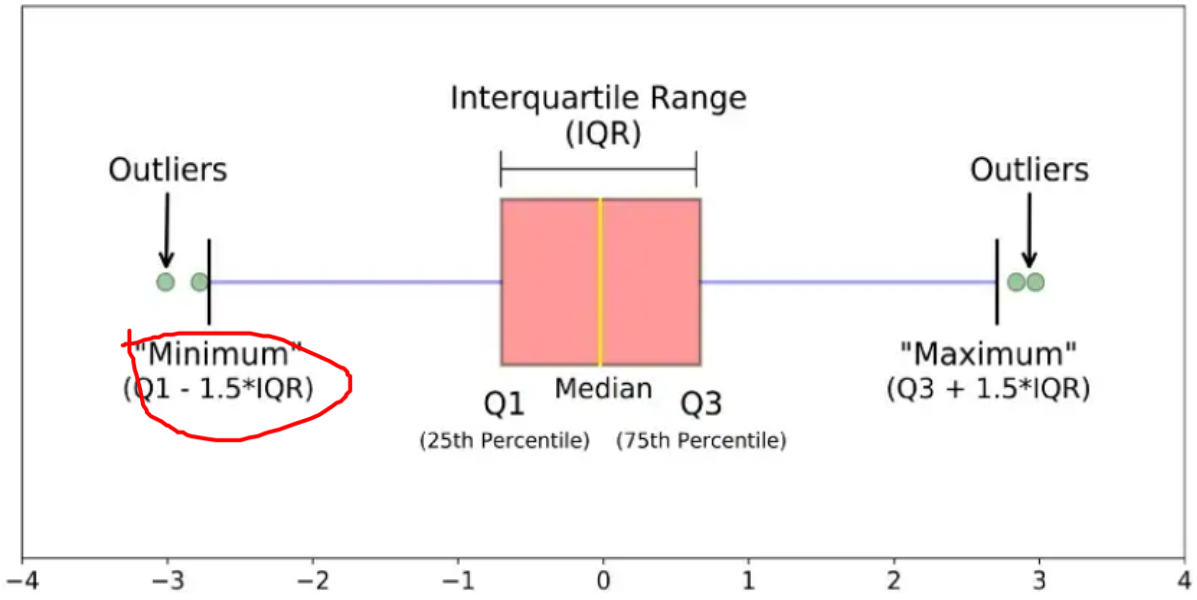
- Unfeasible data (too large or too low values ecc…)
- Data out of possible range (e.g. negative temperature in kelvin, negative ages, ecc…)

**Solution**

- **Remove rows** with inconsistent data
- **impute data** with suitable statistics
- **use model that handle null values**

# Data Cleaning : Outlier

**Data points that differ significantly from other observations**. Some ML models may be sensitive to outlier, so they are excluded from the dataset




Remove data outside a given range, e.g. all data above 99 percentile.

**Pros**

- Simple to implement

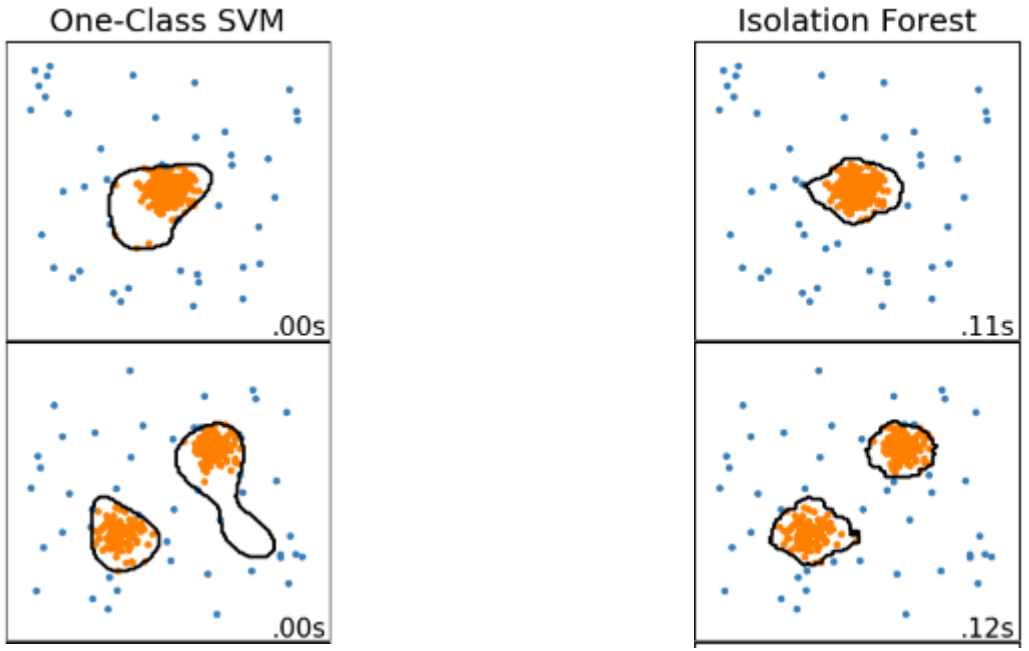
**Cons**

- **Univariate description of the data**

At the end of this chapter, we will introduce **Principal Component Analysis** (PCA) that can be used for outlier detection

# Data Cleaning : Outlier

There are sophisticated methods to detect outlier: here some example that are outside the scope of this course:

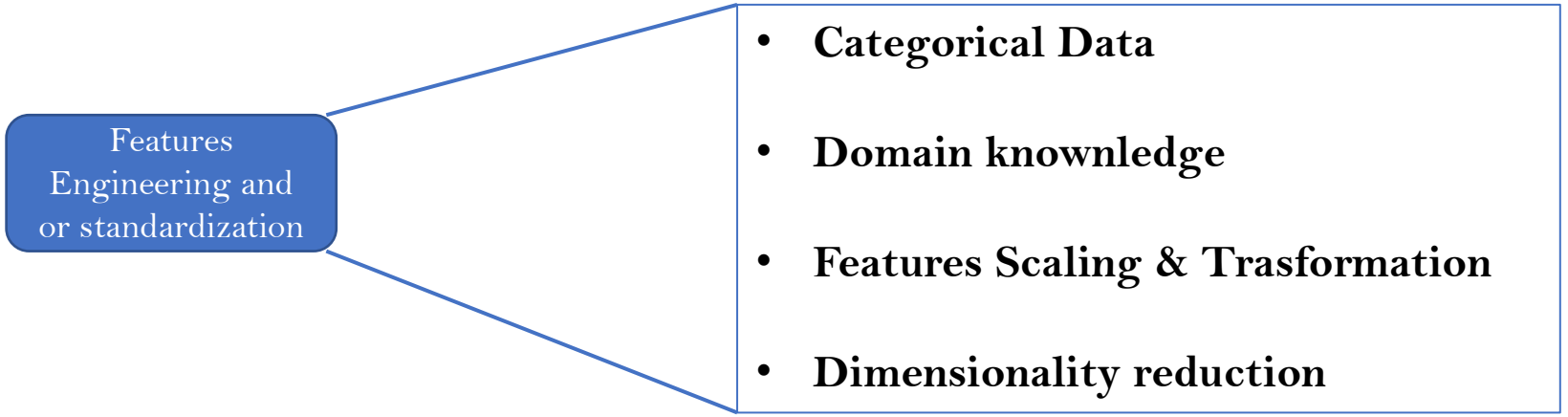



https://scikit-learn.org/stable/modules/outlier_detection.html

Another approach is to **use clustering methos**, see Chapter 7!

# Feature Engineering

Feat. Eng. is the process of using domain knowledge to extract additional features from raw data



# Feature Engineering: Categorical Data

Categorical data takes on values within a finite set of discrete classes

Some examples are:

- Province of birth or residence
- Gender
- Degree of education (phd, high school)

ML algorithms 'mainly'* work with numerical data → **convert/encode** Categorical data into numerical data

| Ordinal | Non ordinal |
| --- | --- |
| maintain an order in their class of values | do not maintain any order |
| ↓ | ↓ |
| Ordinal Encoding | One-Hot Encoding |

* Some models can handle categorical data without any encoding

# Feature Engineering: Categorical Data

## Ordinal

maintain an order in their class of values

**Ordinal Encoding**

Higher class are mapped to higher class

- Phd → 4
- Master Degree → 3
- Bachelor Degree → 2
- High School → 1

Implemented in sklern:
sklearn.preprocessing.OrdinalEncoder

## Non ordinal

do not maintain any order

**One-Hot Encoding**

Create **dummy** variables.
For N class category → give N binary (0/1) columns

- Red
- Blue
- Green

| Red | Blue | Green |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

Implemented in pandas & sklearn:

- sklearn.preprocessing.OneHotEncoder
- Pandas.get_dummy

One column may be eliminated → brings no info

# Feature Engineering: Domain knownledge

using domain knowledge to extract additional features from raw data

$$x_{new} = g(x_3, x_4, \dots)$$

Create a new feature $x_{new}$ as a function of other features

Improve the performances of ML model as compared to supplying only the raw data

The function $g$ is obtained from **domain knownledge**

For instance, consider a dataset with $x_1: gain,\ x_2: cost$

$$x_3 = \frac{(gain - cost)}{cost}$$

ROI: Return of Investement

# Feature Engineering: Scaling

Some ML algorithms may need to scale features so that they are comparable:

- Compare **relative importance of features on a model**, e.g. in linear and logistic regression
- ML Methods involving the **computing of a distance** (eg. nearest neighborn, not studied in this course)
- Make features **independent from unit of measure** (usefull for PCA, next argouments)

## Standardization

Average

$$\widehat{x\_i} = \frac{(x_i - m_i)^2}{\sigma_i^2}$$

Variance

sklearn.preprocessing.StandardScaler

## Min-Max Scaling

Feature min

$$\widehat{x\_i} = \frac{x_i - x_i^{min}}{x_i^{max} - x_i^{min}}$$

Feature max

sklearn.preprocessing.minmax_scale

**Other scaling methods are available on sklearn**

# Feature Engineering: Dimensionality Reduction

Consider a dataset $D$ with $n$ rows (observations) on a set of $p$ predictor, $x_1, \dots, x_p$ . When the values of p is large (e.g $p \gg 100$) it may be difficult to train a ML model from the fact that we need a **lot samples $n$** which, in principle, should **scale exponentially with $p$** (with $p$ feature one may need $\mathrm{n} = 10^p$ rows).

This problem is known as the **curse of dimensionality***

Need a model for **dimensionality reduction:** $p \to m$ with $m \ll p$

This technique should keep only the **relevant variables**

* This is a complex concept and sometimes we will forget about it ☺.

# Feature Engineering: Dimensionality Reduction

What are the **relevant variables**?

A lower-dimensional representation of $D$ that contains as much as possible of the **variance** of the data:
**if we have a constant column (0 variance) it is not useful to make predictions**
**Principal Component Analysis (PCA)** do exactly that!!

PCA aims to find a linear transfomation from $x_1, \dots, x_p$ to $z_1, \dots, z_p$ such that:

- The covariance matrix in the ***z* space is diagonal (1)**
- Keep only z **that explain a given percentage of the total variance (2)**

$x_1 \longrightarrow z_1 = \phi_{11}x_1 + \phi_{12}x_2 + \phi_{13}x_3$
$x_2 \longrightarrow z_2 = \phi_{21}x_1 + \dots.$
$x_3 \longrightarrow z_3 = \dots$

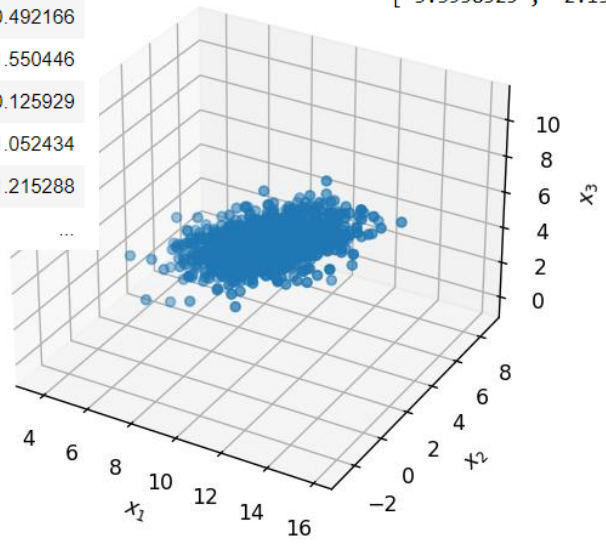
**3d example**

$Z = \Phi X$
Find the matrix $\phi$ that solve (1)
and then take (2)

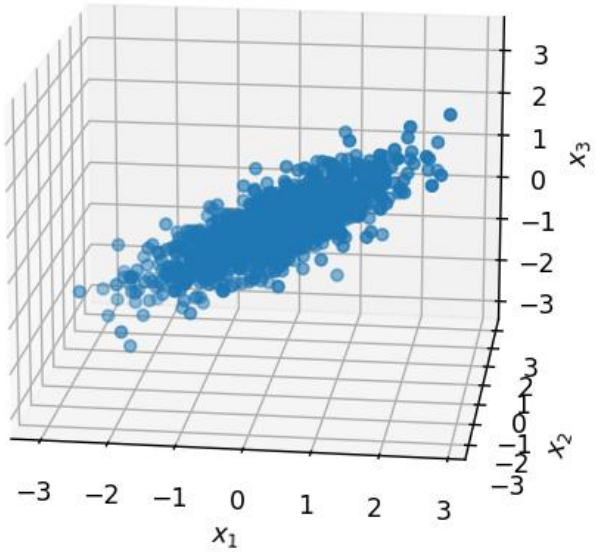
# Feature Engineering: Dimensionality Reduction

| | x1 | x2 | x3 |
|---|---|---|---|
| 0 | -0.713782 | 1.004033 | -0.492166 |
| 1 | -2.292310 | 0.634697 | -1.550446 |
| 2 | 0.324256 | 0.751750 | -0.125929 |
| 3 | 0.278542 | 1.101349 | -1.052434 |
| 4 | -0.858203 | 1.306537 | -1.215288 |
| ... | ... | ... | ... |

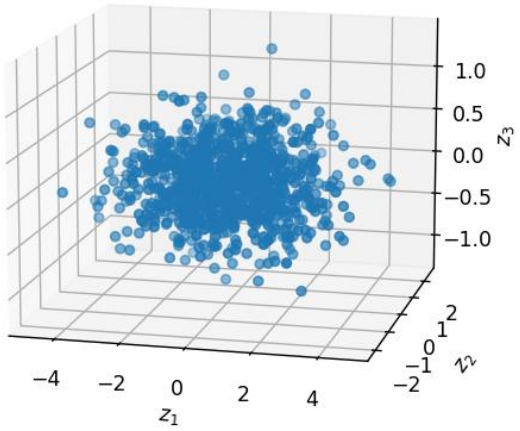
```
array([[ 4.84552459, -1.84664131,  3.3958525 ],
       [-1.84664131,  3.21845889, -2.13604753],
       [ 3.3958525 , -2.13604753,  3.37762347]])
```




Standard Scaling



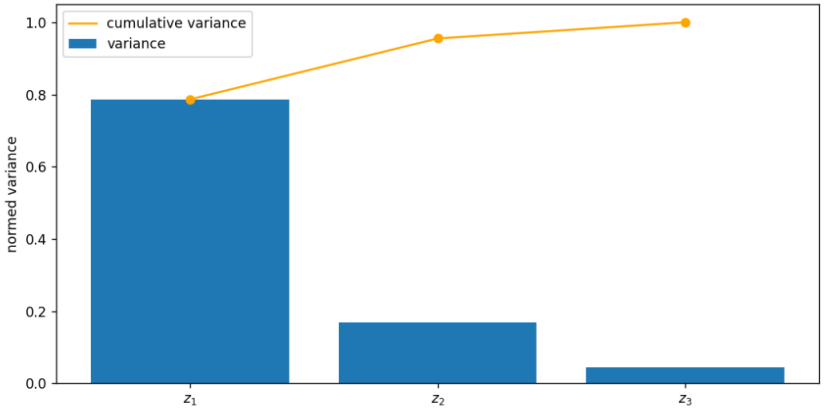

Diagonalizing



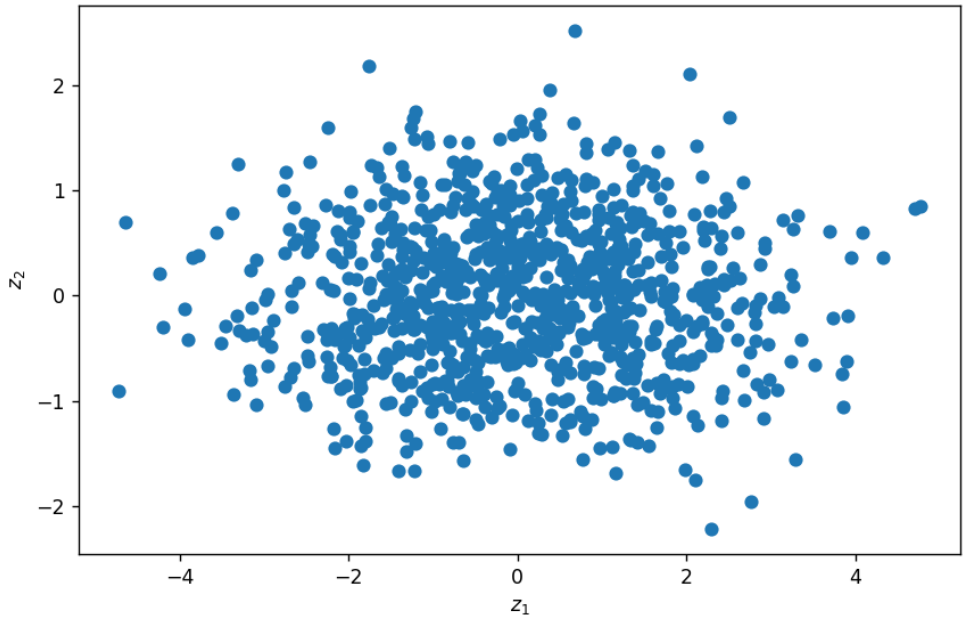

```
array([[ 2.37612477e+00, -9.60192886e-17, -6.36572321e-16],
       [-9.60192886e-17,  4.91899572e-01, -8.06828745e-17],
       [-6.36572321e-16, -8.06828745e-17,  1.34978657e-01]])
```

# Feature Engineering: Dimensionality Reduction

```
array([[ 2.37612477e+00, -9.60192886e-17, -6.36572321e-16],
       [-9.60192886e-17,  4.91899572e-01, -8.06828745e-17],
       [-6.36572321e-16, -8.06828745e-17,  1.34978657e-01]])
```

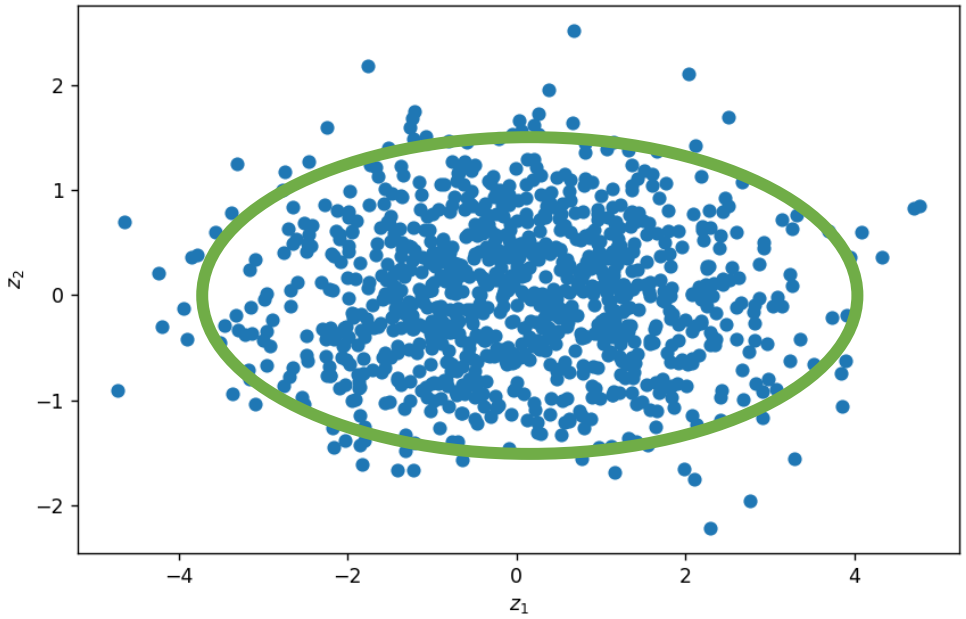



**2 variables explain > 90% variance**




**Reduction from *3* to *2* variables!!**

# PCA for outliers' detection




**HP:**

**Outliers are point far from the center in the z-space**

$$z_i = \sum_k^P z_i^2,\ t_i = \sqrt{\sum_i \frac{z_i^2}{\sigma_i^2}}$$ For each point

Mahalanobis distance: distance weighted with variance (covarince)

**Note: in the z-space the covarince is diagonal otherwise formula is more complex**

$$d_M(\vec{x}, Q) = \sqrt{(\vec{x} - \vec{\mu})^\top S^{-1} (\vec{x} - \vec{\mu})}.$$

Define a threshold $Z_{tht}$ , $T_{thr}$ for $z, t$ and label as outliers all points such that:

$z > Z_{thr}$ $OR$ $t > T_{thr}$

Choose $Z_{tht}$ , $T_{thr}$ based on some percentile of $z_i, t_i$

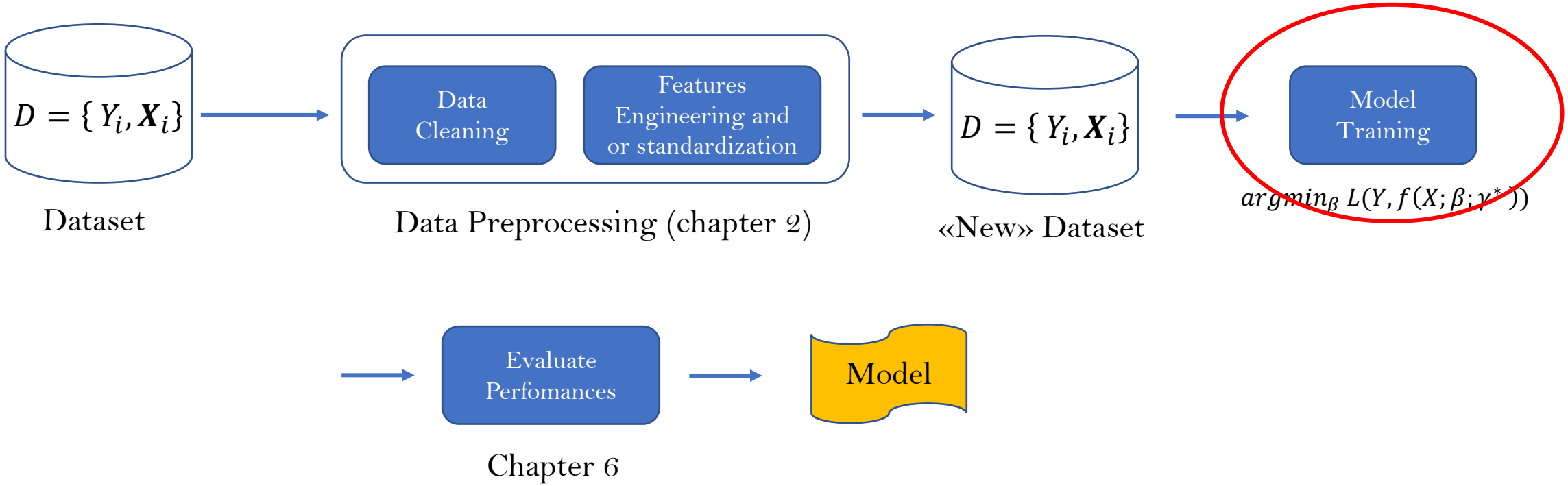
# Python hands on

Notebook on cleaning data chapter 2

PCA example

# Chapter 3: Linear Regression

# Where we are


$D = \{ Y_i, \boldsymbol{X}_i \}$
Dataset
Data Cleaning
Features Engineering and or standardization
Data Preprocessing (chapter 2)
$D = \{ Y_i, \boldsymbol{X}_i \}$
«New» Dataset
Model Training
$argmin_\beta \, L(Y, f(X; \beta; \gamma^*))$
Evaluate Perfomances
Chapter 6
Model

# Linear Regression (LR)

The first (very simple) model we want to study is Linear Regression

In Linear Regression we suppose a linear relation between $x_i$ and $y$

Consider a Dataset $D = \{x_i, y\}$ with $p$ features and $n$ rows **(training dataset)**

$$\hat{y} = f(x) = \beta_0 + \beta_1 x_1 + \ \ldots + \beta_p x_p$$

$\beta = [\beta_o, \ldots, \beta_n]$ are the parameter of the model
LR have no hyper-parameter

$$\beta, \gamma = argmin_{\beta,\gamma} L(Y, f(X; \beta; \gamma))$$

Which L do we use? Mean Squared Error !!

$$L = \frac{1}{n}\sum_{i=0}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=0}^{n}\left(y_i - (\beta_0 + \beta_1 x_{i,1} + \ \ldots + \beta_p x_{i,p})\right)^2$$

$y_i$ : *the actual values*

$\hat{y}_i$: the predicted values

$$\beta^* = argmin_\beta \frac{1}{n}\sum_{i=0}^{n}\left(y_i - (\beta_0 + \beta_1 x_{i,1} + \ \ldots + \beta_p x_{i,p})\right)^2$$

# Solve the optimization problem

$$\beta^* = argmin_\beta \frac{1}{n}\sum_{i=0}^{n}\left(y_i - \left(\beta_0 + \beta_1 x_{,1} + \ \ldots + \beta_p x_{i,p}\right)\right)^2$$

$$\beta^* = argmin_\beta \frac{1}{n}\ (Y - X\beta)^T(Y - X\beta)$$

$$\frac{d}{d\beta}\ L(\beta) = 0$$

Using Matrix Notation

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\mathsf{T} \\ \mathbf{x}_2^\mathsf{T} \\ \vdots \\ \mathbf{x}_n^\mathsf{T} \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix} \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_p \end{bmatrix}$$

The 1 are needed to fit the intercept $\beta_0$

$$\frac{d}{d\beta}(\ X\beta - Y\ )^T(X\beta - Y) = \frac{d}{d\beta}(\beta^T X^T X\beta - \beta^T X^T Y\ - Y^T X\beta + Y^T Y) =$$

$$= 2\ \beta^T X^T X\ - 2Y^T X = \ 2X^T X\beta\ - 2X^T Y$$

This is a scalar transpose
has no effect → we just prefer $\beta$ at the end

«Vector» derivatives

$$\alpha = \mathbf{y}^\mathsf{T}\mathbf{A}\mathbf{x} \qquad \frac{\partial \alpha}{\partial \mathbf{x}} = \mathbf{y}^\mathsf{T}\mathbf{A} \qquad \frac{\partial \alpha}{\partial \mathbf{y}} = \mathbf{x}^\mathsf{T}\mathbf{A}^\mathsf{T}$$

$$\alpha = \mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} \qquad \frac{\partial \alpha}{\partial \mathbf{x}} = \mathbf{x}^\mathsf{T}\left(\mathbf{A} + \mathbf{A}^\mathsf{T}\right)$$

$$2X^T X\beta\ - 2X^T Y = 0\ \rightarrow\ \boxed{\beta^* = (X^T X)^{-1}X^T Y\ \rightarrow \hat{Y} = X_{new}\beta = X_{new}\ (X^T X)^{-1}X^T Y}$$

Covariance Matrix

# Different perspective: max log-likehood

Let us suppose that:

- $y_i$ is a random variable distribuited as a gaussian $G(\mu(x_i); \sigma)$
- $y_i$ are independent
- There is a linear relation between $\mu_i \; and \; x$: $\mu_i(x) = \beta_0 + \beta_1 x_{i,1} + \; ... + \beta_p x_{i,p}$

The $\beta$ can be computed using max likehood as follows:

The probability of observing the sequence $\{y_i\}$ $P(\{y_i\}) = \prod_{i=0}^{n} G(\mu_i; \sigma) = \prod_i^n e^{-\frac{(y_i - \mu_i(x))^2}{2\sigma^2}}$

Extraction the log $logP(\{y_i\}) = -\sum_i^n \frac{(y_i - \mu_i(x))^2}{2\,\sigma^2}$

We want to compute $\beta$ such that the probability of observing the sequance is maximixed

$$\beta^* = argmax_\beta \log P = argmin_\beta \sum_{i=0}^{n} \left(y_i - \left(\beta_0 + \beta_1 x_1 + \; ... + \beta_p x_p\right)\right)^2$$

Same results, but different perspective!!

# Evaluate a Linear Regression: Metrics

| | |
|---|---|
| Mean squared error | $\text{MSE} = \frac{1}{n}\sum_{t=1}^{n} e_t^2$ |
| Root mean squared error | $\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$ |
| Mean absolute error | $\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} \lvert e_t \rvert$ |
| Mean absolute percentage error | $\text{MAPE} = \frac{100\%}{n}\sum_{t=1}^{n} \left\lvert \frac{e_t}{y_t} \right\rvert$ |

# Interpreting the coefficients

When the predictor **are scaled** the $\beta$ can be used to understand the importance of the $x_i$ over $y$

- **Which is the most important feature?** The $x_i$ with the **highest** $|\beta_i|$ is the **most important feature:**
  - **Positive** $\beta_i$ means that for positive values of $x_i$ the **y increase**
  - **Negative** $\beta_i$ means that for negative values of $x_i$ the **y increase**

- The $x_i$ with the **lowest** $|\beta_i|$ is the **less important** and (sometimes) **can be removed** from the dataset

Unluckly in real-world data this is often difficult to interpret variable importance. One problem is **collinearity** of predictors i.e., when one predictor can be expressed as a function (usually linear) of the other. In this case the estimated coefficients $\beta$ may be wrong:

Actual relation

$$\hat{y} = 1 + 2x_1 + 3x_2$$

Perfect collinearity

$$x_1 = 2x_2$$

$$\hat{y} = 1 + 7x_2$$

$$\hat{y} = 1 + \frac{7}{2}x_1$$

$$\hat{y} = \cdots$$

They all have the same performances of the actual model, but very different interpretation.

No more unique solution $(X^T X)^{-1}$ does not exist if there is collinearity :o !!!

# Ridge Regression

To avoid the collinearity and **improve performance and interpretability** of the model we need a **regolarizzation technique**

$\lambda$ is a **hyperparameter** of the model and should be fixed or computed in other ways (**see chapter 6**)

$$\hat{y} = f(x) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$$

$$\beta^* = argmin_\beta \frac{1}{n} \sum_{i=0}^{n} \left( y_i - \left( \beta_0 + \beta_1 x_{i,1} + \ldots + \beta_p x_{i,p} \right) \right)^2 + \lambda \sum_{j}^{p} \beta_j^2$$

This term try to keep the error between model and observation small

This term want to keep the $\beta$ close to 0

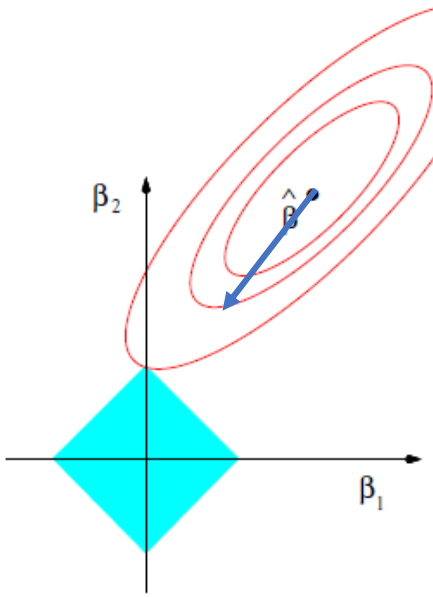
$$\beta^* = (X^T X + \lambda I)^{-1} X^T Y \rightarrow \hat{Y} = X_{new} \beta^* = X_{new} (X^T X + \lambda I)^{-1} X^T Y$$

This method **select the relavant** $\beta_i$ (those different from 0) solving the collinearity problem: this can be also interpreted as a variable selection method !!

# Other type of regolarizzation

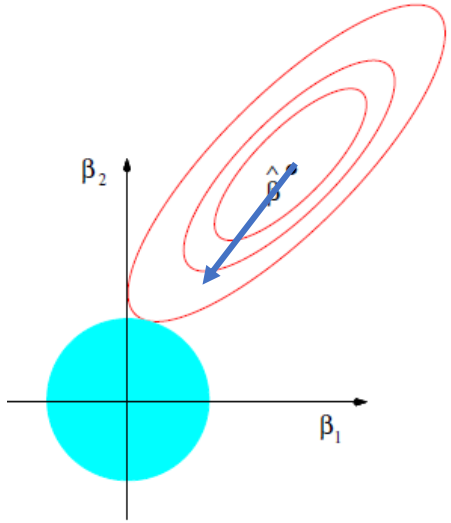
Lasso Regression

$$\beta^* = argmin_\beta \sum_{i=0}^{n} \left(y_i - (\beta_0 + \cdots + \beta_p x_p)\right)^2 + \lambda \sum_{j}^{p} |\beta_j|$$




Ridge Regression

$$\beta^* = argmin_\beta \sum_{i=0}^{n} \left(y_i - (\beta_0 + \cdots + \beta_p x_p)\right)^2 + \lambda \sum_{j}^{p} \beta_j^2$$




ElasticNet Regression

$$\beta^* = argmin_\beta \sum_{i=0}^{n} \left(y_i - (\beta_0 + \cdots + \beta_p x_p)\right)^2 + \lambda_1 \sum_{j}^{p} |\beta_j| + \lambda_2 \sum_{j}^{p} \beta_j^2$$

**Two hyperparameter**

**If $\beta^*$ is the solution of the non-regolarized problem, the regularization move the solotions toward the center of the axis (0) : higher is $\lambda$ more is close to 0**

# Polynomial Regression (PR)

What if the relation between $x$ and $y$ ?

For the sake of simplicity, Consider a Dataset $D = \{x_1, y\}$ with 1 features and $n$ rows and suppose:

$$\hat{y} = f(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_1^3 + \cdots$$

Polynomial Regression

| $x_1$ | $y$ |
|---|---|
| 1 | 12 |
| 4 | 17 |
| 3 | 45 |
| … | … |

Sklearn PolynomialFeatures →

| $x_1$ | $x_1^2$ | $x_1^3$ | $y$ |
|---|---|---|---|
| 1 | 1 | 1 | 12 |
| 4 | 16 | 64 | 17 |
| 3 | 9 | 27 | 45 |
| … | … | … | … |

After the transformation, Poly Reg. is equivalent to Liner Reg. → Poly Reg. is just a feature eng. ☺

$$\beta^* = argmin_\beta \frac{1}{n}\sum_{i=0}^{n}\left(y_i - \left(\beta_0 + \beta_1 x_1 + \ \ldots + \beta_p x_p\right)\right)^2 + \lambda \sum_{j}^{p} |\beta_j|$$

**LASSO/RIDGE must be used to improve model robustness (no overfitting, chapter 5)**

# Kernel Regression

Interesting! we used a linear regression on a transformation of $x$ to get a non-linear model with no effort!
Let us try to generalize this approach:

$x \in R^p \rightarrow \phi(x) \in R^m$

Define a function $\phi$ that map an element in the p-dim features space to a m-dim space (like feature eng.)

Apply all formulas for **Ridge*** Linear Regression on $\phi(x)$

$$\hat{Y}(X_{new}) = X_{new}\beta = X_{new}(X^TX + \lambda I)^{-1}X^TY = \boxed{Y^T(XX^T + \lambda I)^{-1}XX_{new}^T}$$

Used known relation about matrix

$$\hat{Y}(X_{new}) = Y^T(\phi\phi^T + \lambda I)^{-1}\phi\phi^T(X_{new})$$

$\phi$ appears always in the form $\phi(X)\phi^T(X)$ $\rightarrow$ We can define a function K (**kernel**) such as $K = \phi(X)\phi^T(X)$

In practice we do not need to explicit the function $\phi(x)$, but just `K(x,y)` (This is named the **kernel trick**)



* Here **regularization is foundamental** because of high (maybe infinite) dimensional $\phi(x)$ space.

# Kernel Regression

$$\hat{Y}(X_{new}) = Y^T(K + \lambda I)^{-1}k(x, X_{new})$$

$$K = \begin{pmatrix} \phi(x_1)\phi^T(x_1) & \cdots & \phi(x_n)\phi^T(x_1) \\ \vdots & \ddots & \vdots \\ \phi(x_1)\phi^T(x_n) & \cdots & \phi(x_n)\phi^T(x_n) \end{pmatrix} = \begin{pmatrix} K(x_1, x_1) & \cdots & K(x_1, x_n) \\ \vdots & \ddots & \vdots \\ K(x_n, x_1) & \cdots & K(x_n, x_n) \end{pmatrix}$$

## Example 1 : Polynomial Kernel

$$x = (x_1, x_2) \longrightarrow \phi(x) = (1, \sqrt{2}\,x_1, \sqrt{2}\,x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2)$$

$$K(x, y) = (1 + xy^T)^2$$

**Similar to 2-order polynomial regression, but with interaction between features**

## Example 2 : Gaussian Kernel

$$K(x, y) = e^{-\frac{|x-y|^2}{\sigma^2}}$$

$$\phi_{RBF}(x) = e^{-\gamma x^2}\left[1, \sqrt{\frac{2\gamma}{1!}}x, \sqrt{\frac{(2\gamma)^2}{2!}}x^2, \sqrt{\frac{(2\gamma)^3}{3!}}x^3, \ldots\right]^T,$$

**Here $\phi$ is difficult to write and map x to infinite dimensional space !!!! (hint: series expansion)**

# Kernel Regression

$$\hat{Y}(X_{new}) = Y^T(\phi\phi^T + \lambda I)^{-1}\phi\phi^T(X_{new}) = \sum_{i=1}^{N} \alpha_i k(x_i, x_{new})$$

$$(1, N) * (N, N) = (1, N)$$

$$\phi\phi^T = \begin{pmatrix} K(x_1, x_1) & \cdots & K(x_1, x_n) \\ \vdots & \ddots & \vdots \\ K(x_n, x_1) & \cdots & K(x_n, x_n) \end{pmatrix}$$

$\alpha_i$ is a weight for each rows of the dataset, it has Y inside !!

$k(x_i, x_{new})$ measures the similarity between $x_{new}$ and the points in the dataset $x_i$: it will be high when they will be «similar» (similar in the sense given by the kernel function)

# Python hands on

Notebook on regression

Bibliography:


http://web.eecs.umich.edu/~cscott/past_courses/eecs598w14/notes/13_kernel_methods.pdf
https://web2.qatar.cmu.edu/~gdicaro/10315-Fall19/additional/welling-notes-on-kernel-ridge.pdf
https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf
https://stats.stackexchange.com/questions/391692/is-a-polynomial-kernel-ridge-regression-really-equivalent-to-performing-a-linear

# Chapter 4: Classification Methods

# Logistic Regression (LR)

The first (very simple) classification model is Logistic Regression

Consider a Dataset $D = \{x_i, y\}$ with $p$ features and $n$ rows where $y \in \{0,1\}$

**Instead of modelling the response**, we model the probability $p(x)$ that the record belong to a category

In Logistic Regression we have that:

$$p(y = 1 \mid x) = \frac{e^{(\beta_0+\beta_1 x_1+ \ldots+\beta_p x_p)}}{1+e^{(\beta_0+\beta_1 x_1+ \ldots+\beta_p x_p)}}$$

$$p(y = 0 \mid x) = 1 - p(y = 1|x) = \frac{1}{1 + e^{(\beta_0+\beta_1 x_1+ \ldots+\beta_p x_p)}}$$

Known as Logistic Function

$$\log\frac{p(y = 1|x)}{p(y = 0|x)} = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$$

Linear relation on logit!

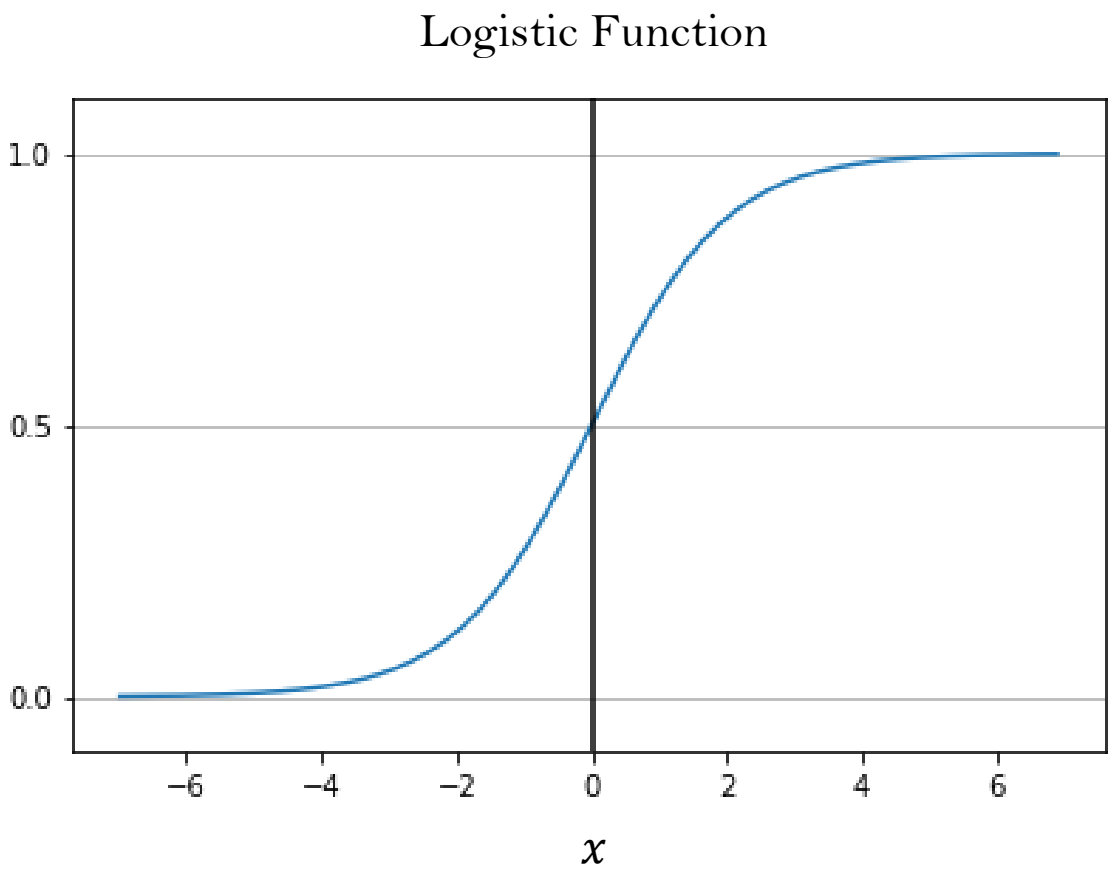
Why this "strange" function and not a simple linear relation for $p(x)$?

Because in this way $p(x) \in [0,1]$ which is "natural" for probabilities.

**Many functions may meet this criteria, but here Logistic Function is used!**

# Logistic Regression (LR)

$$\frac{e^{\beta x}}{1 + e^{\beta x}}$$



# Logistic Regression (LR)

Now we need to computed the coefficients $\beta$ → Use Max. likehood

The probability of observing the sequence $\{y_i\}$

$$P(\{y_i\}) = \prod_{y_i=1} p(y=1|x) \prod_{y_i=0} \big(1 - p(y=1|x)\big) = \prod_i p(y=1|\,x)^{y_i} \big(1 - p(y=1|\,x)\big)^{1-y_i}$$

$$\log P\,(\{y_i\}) = \sum_i y_i \ln p(y=1|x) + (1-y_i) ln(1 - p(y=1|x))$$

Substituting the form for $p(y=1|x)$ used in Logistic Regression, and **using scalar product** we obtain

$$\log P\,(\{y_i\}) = \sum_i y_i \boldsymbol{\beta} \cdot x_i - \log(1 + e^{\boldsymbol{\beta} \cdot x_i})$$

$$\beta^* = argmin_\beta - \sum_i y_i \boldsymbol{\beta} \cdot x_i - ln(1 + e^{\boldsymbol{\beta} \cdot x_i})$$

**Regolarizzation can be applyed**

Changed sign to write as a min-problem

$L(Y, f(X;\beta))$ Loss Function

# Logistic Regression (LR)

$$\beta^* = argmin_\beta L(Y, f(X;\beta)) = argmin_\beta(-\sum_i y_i\beta \cdot x_i - \ln\left(1 + e^{\beta \cdot x_i}\right))$$

$$\frac{d}{d\beta_j} L(Y, f(X;\beta)) = 0 \rightarrow \boxed{\sum_i -y_i x_i^j + x_i^j \, p(x_i;\beta) = 0}$$

$i$ is the row index
$j$ is the features index

This is a **non-linear system of p-equations** for all $\beta_i$ and can **not** be solved analytically

**Let's do it numerically:**
**Gradient Descent!**

# Gradient Descent

We want to solve the problem: $\beta^* = argmin_\beta L(\beta)$ or equivalently, find $\beta^*$ such as $\frac{d}{d\beta} L(\beta)|_{\beta^*} = 0$

Define an iterative (pseudo-dynamics) algorithm as follow:

$$\beta_0 = \beta_0$$

$$\beta_{t+1} = \beta_t - \gamma \frac{d}{d\beta} L(\beta)\Big|_{\beta_t}$$

$$\downarrow \quad t \to \infty$$

$$\beta_t \to \beta_{t+1} \to \beta^*$$

$$\cancel{\beta^*} = \cancel{\beta^*} - \gamma \frac{d}{d\beta} L(\beta)\Big|_{\beta^*}$$

$$\frac{d}{d\beta} L(\beta)\Big|_{\beta^*} = 0$$

$\gamma$ is **the learning rate (lr)** (timestep in physics):

- The correct value depend on the problem at hand, i.e. it depends on the magnitude of $\frac{d}{d\beta} L(\beta)$
- The convergence of the algorithm critically depends on $\gamma$:
  - too small cause the process to get stuck
  - too large may cause instability

This method converge to global minimum only if $L$ is a convex function of $\beta$. In the other cases it converges to local minima!

# Gradient Descent: an example

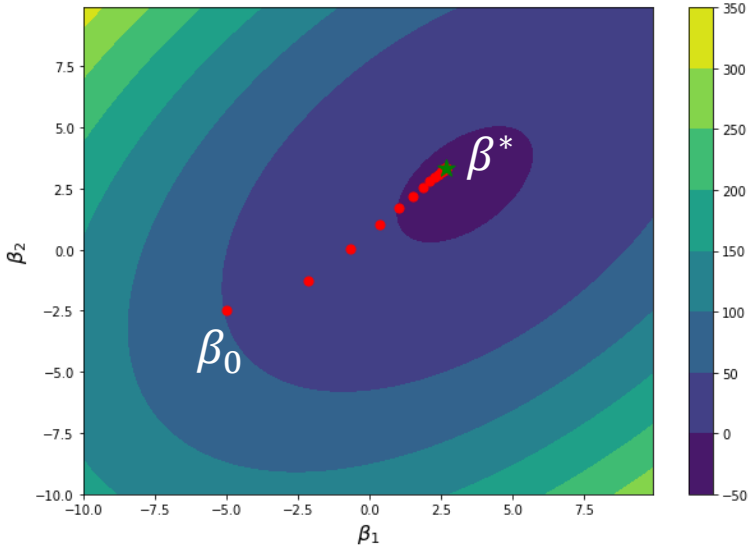
$$L = (\beta_1 - 1)^2 + (\beta_2 - 2)^2 - \beta_1\beta_2$$

$$\frac{dL}{d\beta_1} = 2(\beta_1 - 1) - \beta_2$$

$$\frac{dL}{d\beta_2} = 2(\beta_2 - 2) - \beta_1$$

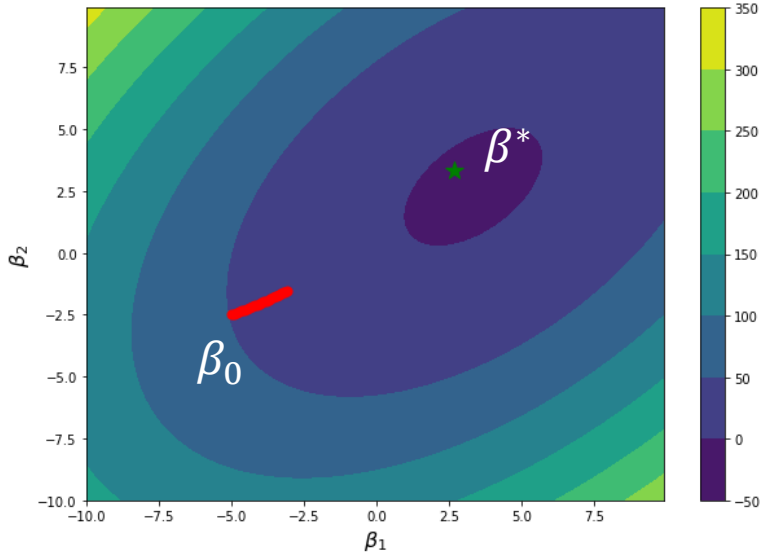
$$\beta_{t+1} = \beta_t - \gamma \frac{d}{d\beta} L(\beta)\Big|_{\beta_t}$$

$\gamma = 0.3$



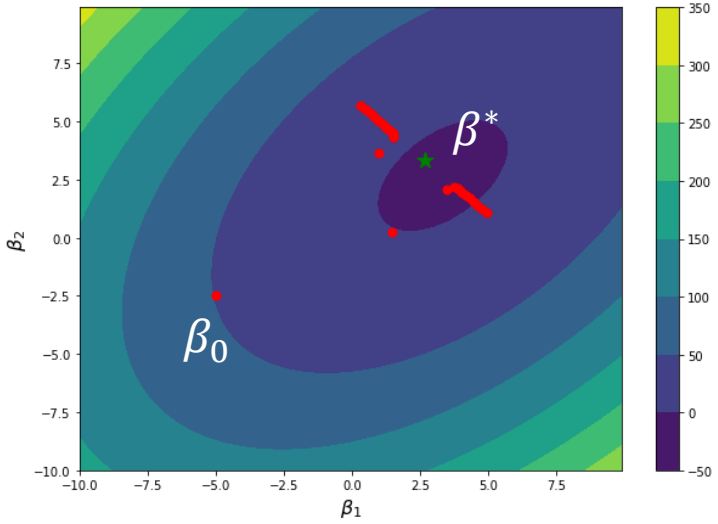

$\gamma$ is ok!

$\gamma = 0.01$




$\gamma$ is too small → slow convergence

$\gamma = 0.68$




$\gamma$ is too big → instability, bounce!

# Interpreting the coefficients

When the predictor **are scaled** the $\beta$ can be used to understand the importance of the $x_i$ over $y$

- The $x_i$ with the **highest** $|\beta_i|$ is the **most important:**
  - **Positive** $\beta_i$ means that for positive values of $x_i$ the $\beta x$ **increase , hence** $\boldsymbol{p(x)} \rightarrow 1$
  - **Negative** $\beta_i$ means that for positive values of $x_i$ the $\beta x$ **decrease, hence** $\boldsymbol{p(x)} \rightarrow 0$
- The $x_i$ with the **lowest** $|\beta_i|$ is the **less important** and **can be removed** from the dataset
- Instead of $p(x)$ you can interpret coefficients in term of odds i.e., $ln\left(\frac{p}{1-p}\right)$

Also, for logistic regression **collinearity** can be a problem for the estimation of $\beta$ coefficients.
This problem can be solved as well applying **reguralizzation techniques**.

# Prediction & Performances

Logistic regression, given $x_i$ output a probability $p(x)$ that the record belong to the class 1.
We could classify an observation based on some rules on $p(x)$: let us define a threshold $t$ such as:

- If $p(x) > t$ the obeservation belong to class 1
- If $p(x) \leq t$ the obeservation belong to class 0

One can be tempt to choose $t = 0.5$, but this value really depends on the problems at hand

Confusion Matrix

| | Predicted Negative | Predicted Positive |
|---|---|---|
| Outcome Negative | True Negative(TN) | False Positive (FP) |
| Oucome Positive | False Negative(FN) | True Positive (TP) |

$$Accuracy = \frac{TN + TP}{TN + FP + FN + TP}$$

Overall performance of the model: How many corrected classification over all observations

The value of this metric depend on $t$

# Prediction & Performances

Logistic regression, given $x_i$ output a probability $p(x)$ that the record belong to the class 1.
We could classify an observation based on some rules on $p(x)$: let us define a threshold $t$ such as:

- If $p(x) > t$ the obeservation belong to class 1
- If $p(x) \leq t$ the obeservation belong to class 0

One can be tempt to choose $t = 0.5$, but this value really depends on the problems at hand

Confusion Matrix

| | Predicted Negative | Predicted Positive |
|---|---|---|
| Outcome Negative | True Negative(TN) | False Positive (FP) |
| Oucome Positive | False Negative(FN) | True Positive (TP) |

$$recall = \frac{TP}{TP + FN}$$

Recall: How many good prediction over the total actual positive

The value of this metric depend on $t$

# Prediction & Performances

Logistic regression, given $x_i$ output a probability $p(x)$ that the record belong to the class 1.
We could classify an observation based on some rules on $p(x)$: let us define a threshold $t$ such as:

- If $p(x) > t$ the obeservation belong to class 1
- If $p(x) \leq t$ the obeservation belong to class 0

One can be tempt to choose $t = 0.5$, but this value really depends on the problems at hand

Confusion Matrix

| | Predicted Negative | Predicted Positive |
|---|---|---|
| Outcome Negative | True Negative(TN) | False Positive (FP) |
| Oucome Positive | False Negative(FN) | True Positive (TP) |

$$precision = \frac{TP}{TP + FP}$$

Precision: How many good prediction over the total predicted positive

The value of this metric depend on $t$

# Prediction & Performances

Logistic regression, given $x_i$ output a probability $p(x)$ that the record belong to the class 1.
We could classify an observation based on some rules on $p(x)$: let us define a threshold $t$ such as:

- If $p(x) > t$ the obeservation belong to class 1
- If $p(x) \leq t$ the obeservation belong to class 0

One can be tempt to choose $t = 0.5$, but this value really depends on the problems at hand

Confusion Matrix

| | Predicted Negative | Predicted Positive |
|---|---|---|
| Outcome Negative | True Negative(TN) | False Positive (FP) |
| Oucome Positive | False Negative(FN) | True Positive (TP) |

$$F_{beta}\ score = (1 + \beta^2)\frac{precision * recall}{\beta^2 precision + recall}$$

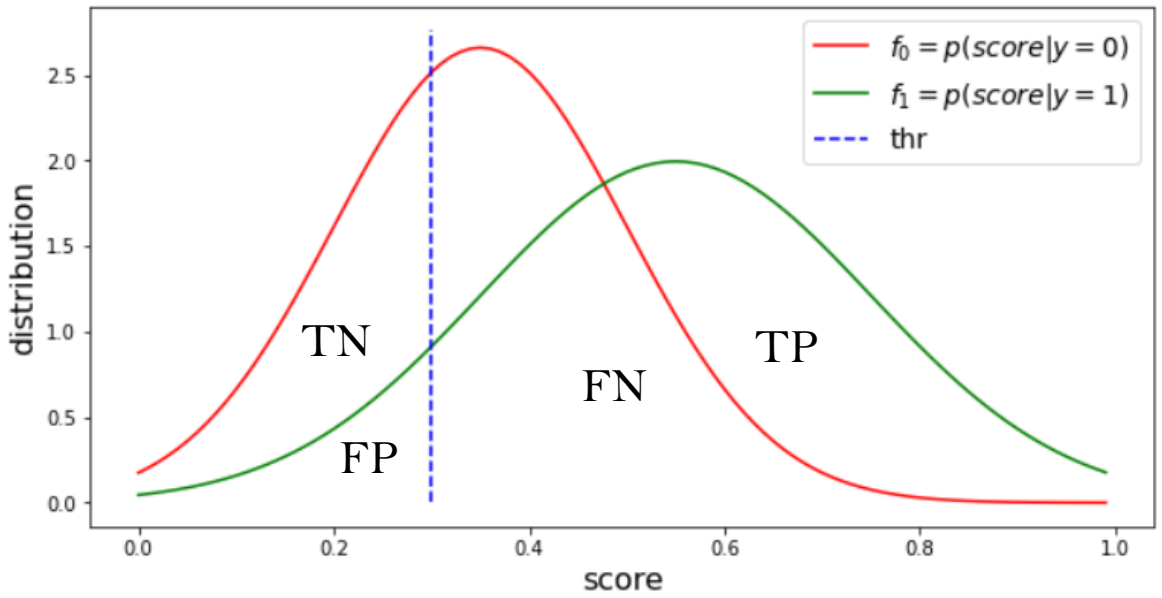
Precision: weight precision & recall
$F_1\ score$ is harmonic mean of prec & recall

The value of this metric depend on $t$. **Usually $t$ is chosen by maximaxing $F_1\ score$**

# ROC

All the metrics discussed above depend on the choice of the threshold $t$. Here we want to find define a metric independent of $t$.

| Actual Outcome | Score/Probability |
|---|---|
| 0 | 0.2 |
| 0 | 0.8 |
| 0 | 0.15 |
| 0 | 0.25 |
| … | … |
| 1 | 0.12 |
| 1 | 0.81 |
| 1 | 0.6 |

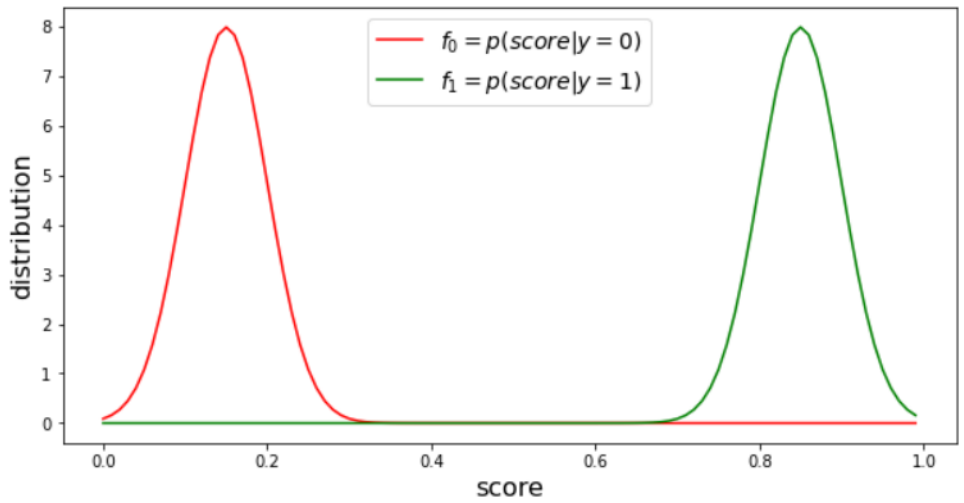


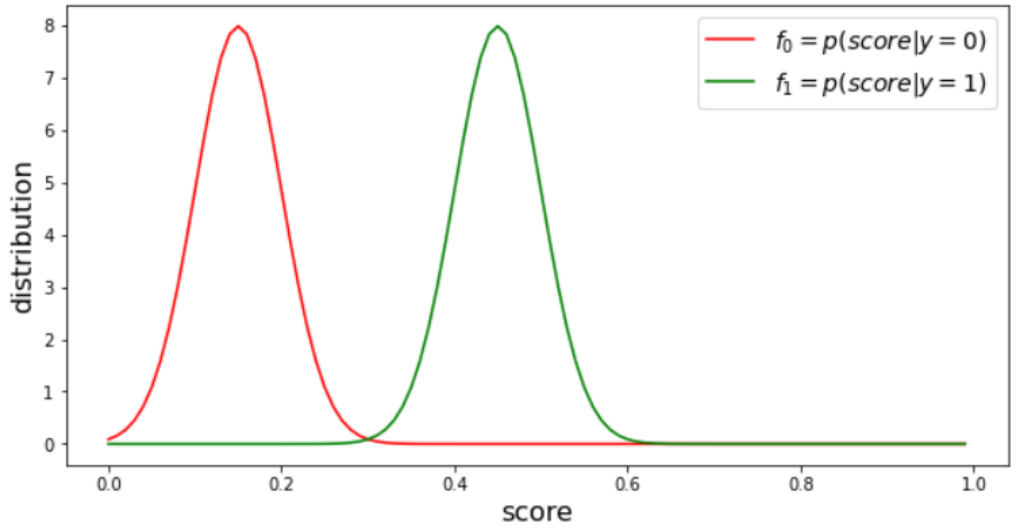

We can say that a good classifier output a score that can separate 0-class to 1-class

# ROC

All the metrics discussed above depend on the choice of the threshold $t$. Here we want to find define a metric independent of $t$

Which can be a good metric?



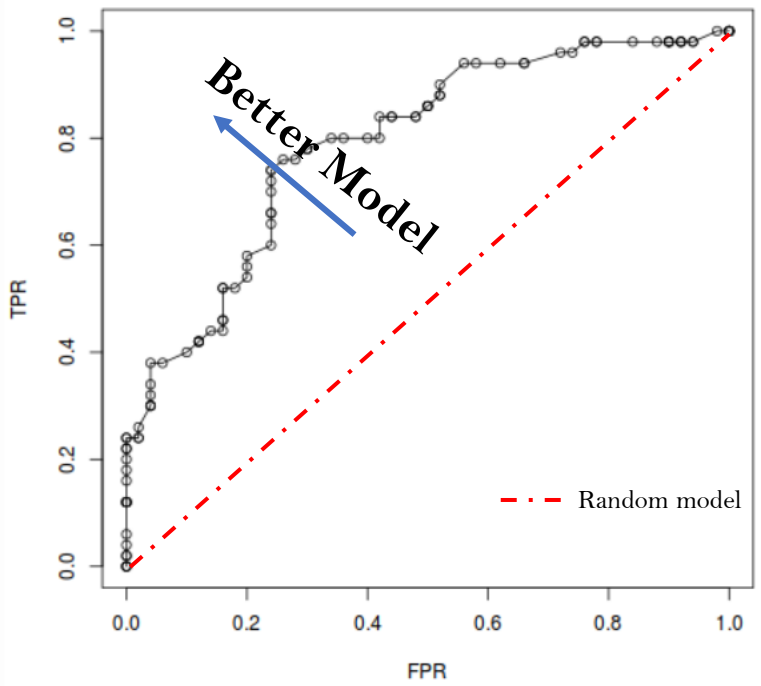

Both are very good classifier, only the scale of score/predictions change.

**We need a metrics that measure how much this two distribution overlap**

# ROC

To define this metrics, firstly we define the ROC curve (Receiver Operating Characteristic).
For each values of the threshold $t$, we compute: $TPR(t) = \frac{TP}{\#actual\ positive}$, $FPR(t) = \frac{FP}{\#actual\ negative}$

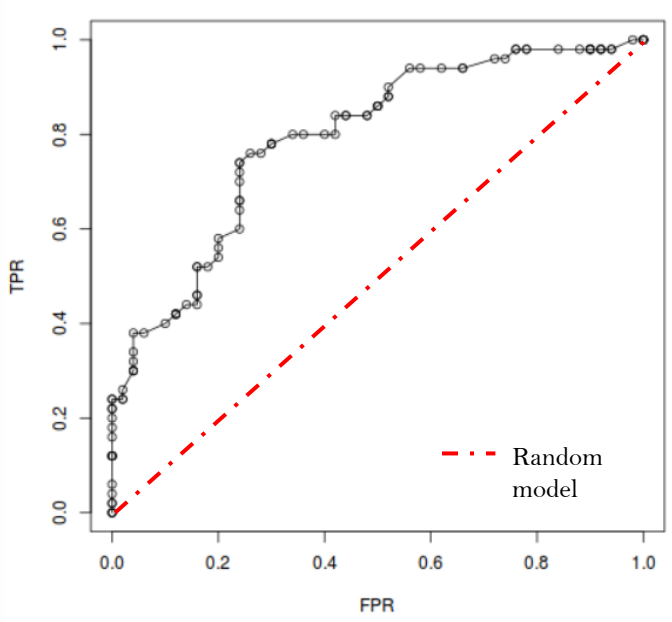
**ROC curve is the plot of TRP (recall) against FPR**

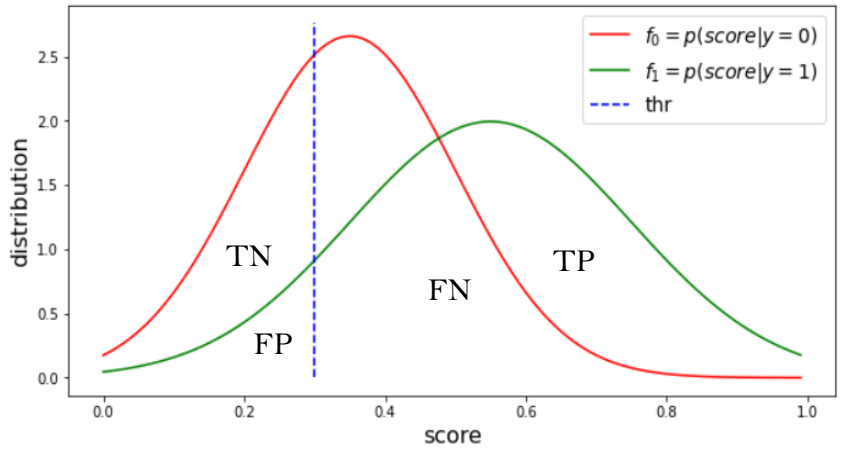



An example or ROC curve

- The point (0,1) is the perfect model!
- The red line is the random model line. Good model should have a ROC above this line (see proof in the upcoming slide)
- If a ROC is below the random model line, just inverting prediction 0 with 1 you obtain a better model above the red dashed line
- Given 2 models such as $ROC_1 \geq ROC_2$ for every FPR then **model 1 is better than model 2**

# Area under the ROC




$$TPR(t) = P[p(x) > t \mid y = 1] = \int_t^1 f1(t)dt$$

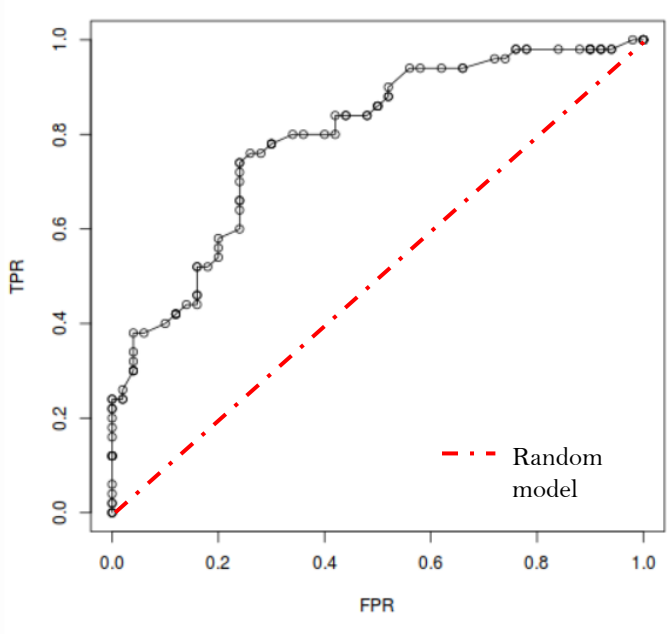
$$FPR(t) = P[p(x) < t \mid y = 0] = \int_0^t f_0(t)dt \ = F(t)$$

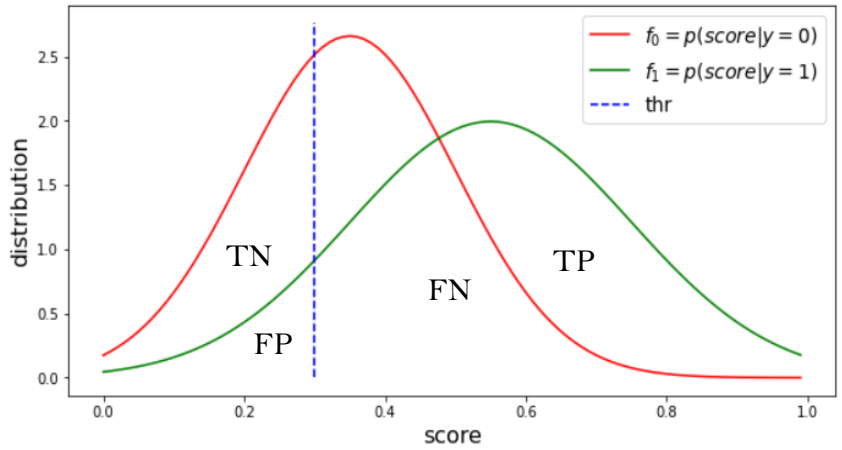
$p(x)$ is the score

**Area Under the curve**

$$\begin{aligned}
\mathrm{AUC} &= \int_0^1 T(F_0)\,\mathrm{d}F_0 \\
&= \int_0^1 P[\hat{p}(\mathbf{x}) > \quad t \quad \mid y(\mathbf{x}) = 1]\,\mathrm{d}F_0 \\
&= \int_1^0 P[\hat{p}(\mathbf{x}) > \quad t \quad \mid y(\mathbf{x}) = 1] \cdot \frac{\partial F(t)}{\partial t}\,\mathrm{d}t \\
&= \int_0^1 P[\hat{p}(\mathbf{x}) > t \mid y(\mathbf{x}) = 1] \cdot P[\hat{p}(\mathbf{x}') = t \mid y(\mathbf{x}') = 0]\,\mathrm{d}t \\
&= \int_0^1 P[\hat{p}(\mathbf{x}) > \hat{p}(\mathbf{x}') \,\&\, \hat{p}(\mathbf{x}') = t \mid y(\mathbf{x}) = 1 \,\&\, y(\mathbf{x}') = 0]\,\mathrm{d}t \\
&= P[\hat{p}(\mathbf{x}) > \hat{p}(\mathbf{x}') \mid y(\mathbf{x}) = 1 \,\&\, y(\mathbf{x}') = 0],
\end{aligned}$$

# Area under the ROC




$$TPR(t) = P[p(x) > t \mid y = 1] = \int_t^1 f1(t)dt$$

$$FPR(t) = P[p(x) < t \mid y = 0] = \int_0^t f_0(t)dt \ = F(t)$$

$$AUC = \int TPR(FPR)dFPR =$$
$$P[p(x) > p(x')|\, y(x) = 1 \,\&\, y(x') = 0]$$

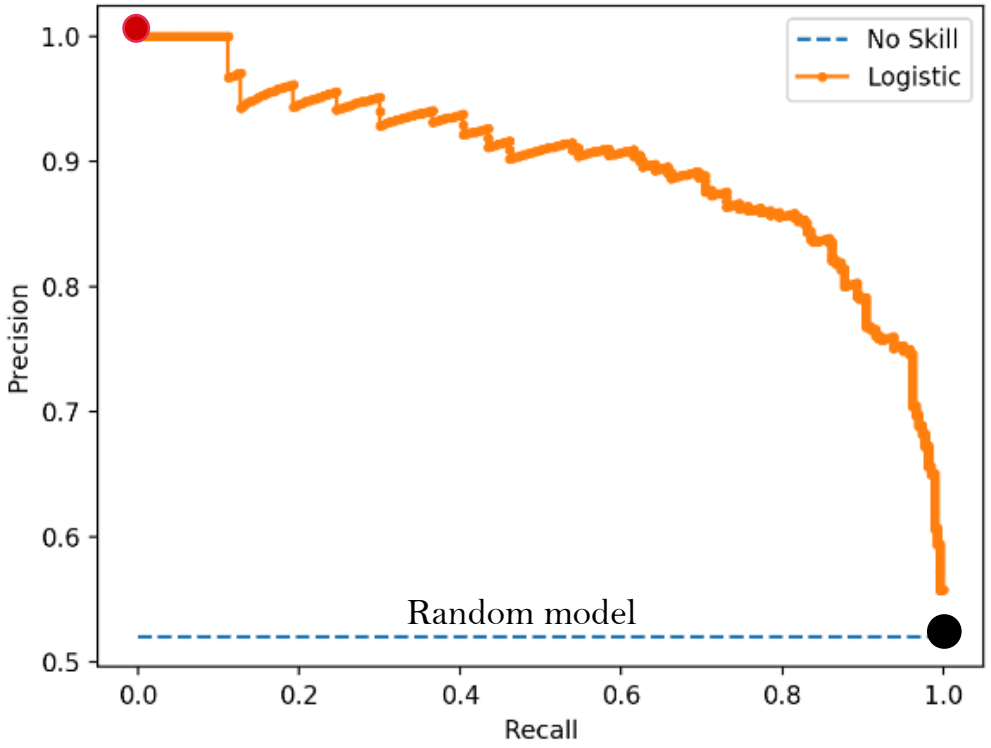
In other word, given a randomly observation $x$ belonging to *class 1*, and a randomly chosen observation $x'$ belonging to *class 0*, the AUC is the probability that the classification algorithm will assign a higher score to $x$ than to $x'$:

**This is equivalent to ask that the red and green distribution are separated. Higher is the AUC, higher is the separation!**

# Precision Recall curve

Another widley used curve is the **Precion-Recall PR curve**.
For each values of the threshold $t$, we compute: $precision(t),\ recall(t)$

**PR curve is the plot of precision vs recall**




An example or ROC curve

- The blue line is the random model line. Good model should have a PR above this line. Random model is the line $precision = \frac{\#\text{total positive}}{\#total\ case}$ fraction of positive case
- $\boldsymbol{AUC - PR}$ is a valid metric, but has not easy probabilistic interpretation

  - **Dummy model** predicts always class 1: recall = 1, precision = fraction of positive
  - **Dummy model** just one obs as class 1: recall = 1/N, precision = 100%

# ROC – PR: property of random model

The **random model** assign a score according to a uniform distribution between 0 and 1

$N$

| Actual Outcome | Score/Prediction |
|---|---|
| 0 | 0.2 |
| 0 | 0.8 |
| 0 | 0.15 |
| 0 | 0.25 |
| … | … |
| 1 | 0.12 |
| 1 | 0.81 |
| 1 | 0.6 |

$k$

$1-k$

$t \to 0_{class}$

$1-t \to 1_{class}$

$N: observation, \quad k: fraction\ of\ 0-class$
$t: threshold$

$$FPR = \frac{FP}{N} = \frac{Nk(1-t)}{Nk} = 1 - t$$

$$TPR = \frac{TP}{P} = \frac{N(1-k)(1-t)}{N(1-k)} = 1 - t$$

This is a straight line in
$FPR$ vs $TPR$ plane

# ROC – PR: property of random model

The **random model** assign a score according to a uniform distribution between 0 and 1

$N$

| Actual Outcome | Score/Prediction |
|---|---|
| 0 | 0.2 |
| 0 | 0.8 |
| 0 | 0.15 |
| 0 | 0.25 |
| … | … |
| 1 | 0.12 |
| 1 | 0.81 |
| 1 | 0.6 |

$k$

$1-k$

$t \to 0_{class}$

$1-t \to 1_{class}$

$N: observation, \quad k: fraction\ of\ 0-class$
$t: threshold$

$$precision = \frac{N(1-k)(1-t)}{Nk(1-t)+N(k-1)(1-t)} = (1-k)$$

$(1-k)$ is the fraction of positive case and is independent on recall and $t$

# More on precision & recall

$$recall = \frac{TP}{TP+FN}; \quad precision = \frac{TP}{TP+FP}$$

Confusion Matrix

| | Predicted Negative | Predicted Positive |
|---|---|---|
| Outcome Negative | True Negative(TN) | False Positive (FP) |
| Oucome Positive | False Negative(FN) | True Positive (TP) |

Some example:

- A Classification algorithm that suggest to control or not a luggage in an airport to detect a bomb, anything that doesn't account for **false-negatives** is a problem. **Recall** is a better measure than precision.
- When determining if a patient should receive a particular medication, especially considering its catastrophic effects when given to a healthy individual, it's crucial to avoid erroneously predicting a positive outcome for a negative instance. Since we want to **avoid FP** errors, we prefer **Precision**
- Usually, you can assign a business value to FP and FN (thus to prediction and recall) and just compute this metrics.

# Unbalanced Dataset

**Unbalanced Dataset** when the number of observation of 1 class is lower than those of 0 class (1% or 0.1%):

- Accuracy in not a good metrics to evaluate a model (see e.g. if the model predict all 0-class)
- Model prediction are often not calibrated, i.e. tends to be closer to 0
- **AUC-ROC** (but also tend to be too optimistic) and, **mainly**, **precision-recall** are the right way to measure performances (usually we are interested in 1-class related performaces)
- You can add weight on loss for 1 class to penalize more the 1 class term (there is an option in .fit() method)

$$\sum_i y_i \ln p(y=1|x) + w_i(1-y_i)\ln(1-p(y=1|x))$$

$w_i$ it become an hyper-parameter of the model

- Downsample 0-class observation
- Oversample 1-class observation (exists particular methods, e.g. SMOTE:out of the scope of this course)

# Generalized Linear Model

| Distribution | Target Domain | Model hp | Likehood | Loss |
|---|---|---|---|---|
| Bernoulli | $y \in \{0,1\}$ | $p(x) = \frac{1}{1+e^{\beta x}}$ ; $\log\frac{p}{1-p} = \beta x$ | $P(\{y_i\}) = \prod_{y_i=1} p \prod_{y_i=0} (1-p)$ | $L = \sum_i y_i \beta x_i - \ln(1+e^{\beta x_i})$ |
| Poisson / Count | $y \in \{0,1,..n\}$ | $\ln(\mu(x)) = \ln(E(y\|x)) = \beta x$ | $P(\{y_i\}) = \prod_i \frac{\mu^{y_i}}{y_i!} e^{-\mu}$ | $L = \sum_i y_i \beta x_i - e^{\beta x_i}$ |
| … | … | … | … | … |

This approach can be generalized to different distribution:

- Suppose a probability distribution for $y_i$
- Define a relation between $x$ and some moments of $y_i$
- Compute the probability of the sequences, extract the log and compute the Loss

# Python hands on

Notebook on regression

Bibliography:


http://web.eecs.umich.edu/~cscott/past_courses/eecs598w14/notes/13_kernel_methods.pdf
https://web2.qatar.cmu.edu/~gdicaro/10315-Fall19/additional/welling-notes-on-kernel-ridge.pdf
https://people.eecs.berkeley.edu/~jordan/courses/281B-spring04/lectures/lec3.pdf
https://stats.stackexchange.com/questions/391692/is-a-polynomial-kernel-ridge-regression-really-equivalent-to-performing-a-linear

# Chapter 5: Evaluate a Model

# Motivation

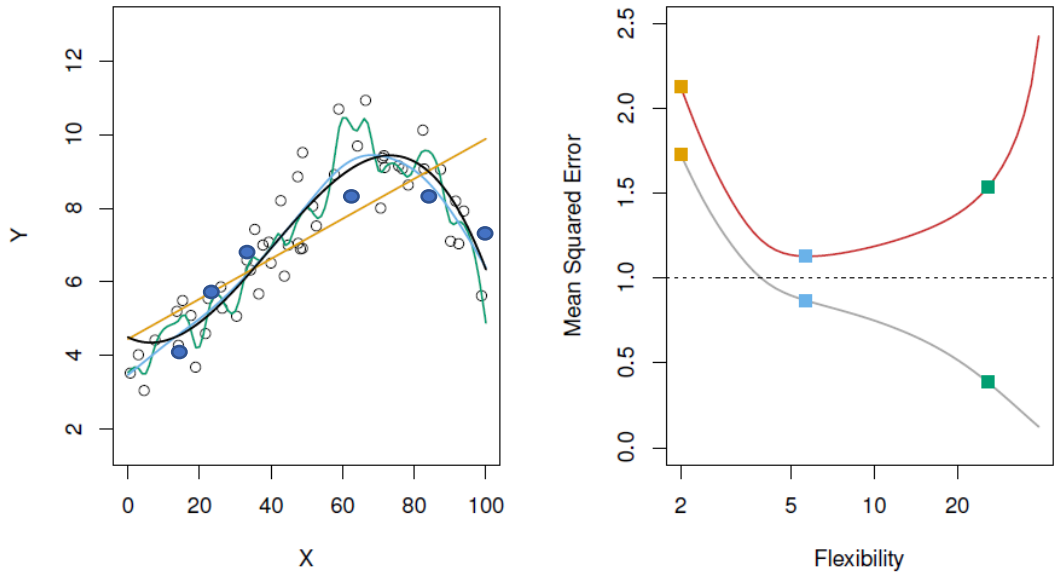
- In all the real-world situation, we do not really care how the model works well on the **training data**. Rather, we are interested in the performance of the model to previously **unseen test data**:
  - Performance on unseen test data are refered as the **generalization performance** of the model.
  - A model that perform well on training, but has low perfomance on test suffers from **overfitting**
  - Trade-off between generalization & overfitting is often referred as **bias-variance trade-off.**
- Moreover, we need also a procedure to compute the **hyper-parameters of a given model** (e.g., the value of $\lambda$ in LASSO/RIDGE regression, or the kernel for a regression)

In this chapter we will introduce statistical methods aimed at evaluate overfitting and compute model hyper-parameter

# Overfitting

Just consider two sets of data:

- Train data: used to build the model
- Test data: used to evaluate the model (must remain unseen)




Right) From linear regression, to high order polynomial regression. Training data are the empty dots.

Left) Performance on train data (gray line) improves as the model complexity increase while, performance on test data (red line) becomes lower. This is a fundamental property of statistical learning holding regardless of the datasets and of the model being used. Model with order equal to 20 **overfits.**

Idea: Seems reasonable to choose the model with polynomial order equal to 5: it show good performance for both sets (but we are selecting the model looking at data that should be unseen: **data leak**). **We are mixing together the generalization performance and the choose of the hyper-parameter of the model.**

# Bias – Variance trade-off

It can be shown that the expected error on a new unseen points $x_o$ can be written as a sum of 2 terms:

$$E\left(y_0 - \hat{f}(x_0)\right)^2 = \mathrm{Var}(\hat{f}(x_0)) + [\mathrm{Bias}(\hat{f}(x_0))]^2$$

Mean squared error case

The overall expected test MSE can be computed by averaging over all possible values of $x_0$ in the test set

**Variance** refers to the amount by which the estimated model $f$ would change if we use a different training data set. In general, more **complex/flexible** statistical methods have higher variance.

**Bias** refers to the error that is introduced by approximating a real-world problem, by a much simpler model.

- in order to minimize the expected test error, we need to select a statistical learning method that achieves low variance and low bias but…
- For more **complex/flexible** methods, the variance will increase, and the bias will decrease. **Should find a trade-off**.

# The simpler setup: train-test-validation splitting

If we are in a **data-rich situation**, the best approach is to randomly divide the dataset into three parts:


Used to fit the model (70% of data)

Used to evaluate the **model performances** (15% of data)

Used to evaluate the **model generalization** (15% of data)

In practice: define a grid for all the hyper-parameter e.g., $\lambda \in \{\lambda_1, \dots, \lambda_L\}$:

- For each value of $\lambda$, train the model on the train set
- Evaluate the performance of the model on the validation-set
- Choose the hyper-parameter $\lambda^*$ with the **best** validation performance (why not optimization* ??)
- Retrain the model with $\lambda^*$ and evaluate the generalizzation error on the test. **Finally compare performances between sets to adress possible overfitting!**

**This setup work well in a data-rich situation**

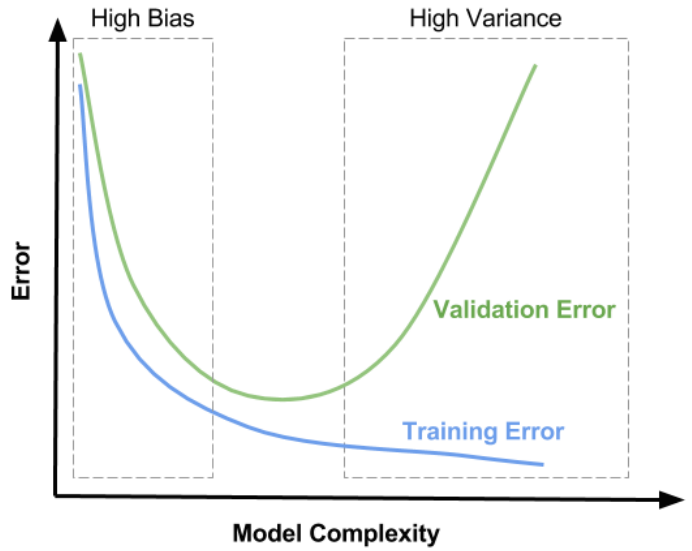


* One can also use optimization methods to find the best $\lambda^*$ (see e.g. Optuna python) but this method are beyond the scope of this course.

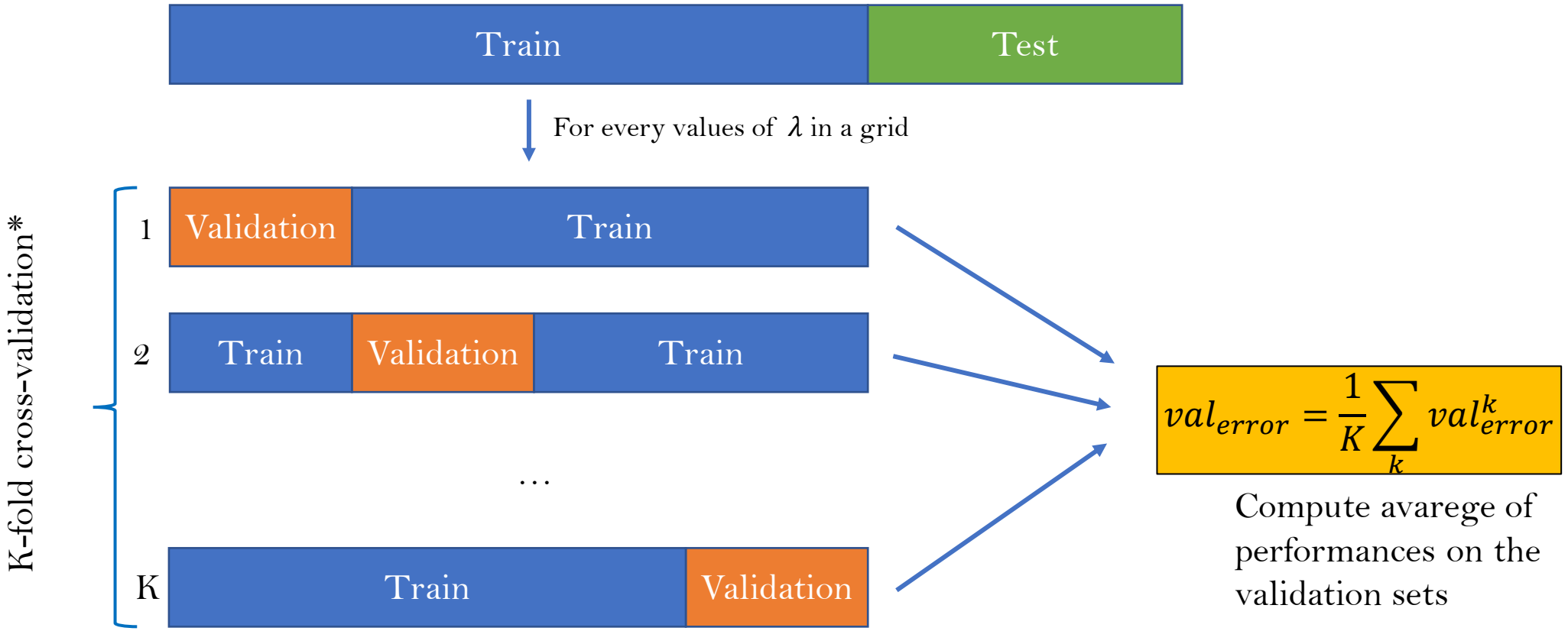
# The simpler setup: train-test-validation splitting

If we are in a **data-rich situation**, the best approach is to randomly divide the dataset into three parts:

| Training set | Validation set | Test set |
|---|---|---|
| Used to fit the model (70% of data) | Used to evaluate the **model performances** (15% of data) | Used to evaluate the **model generalization** (15% of data) |



# Cross-validation: less data-rich situation

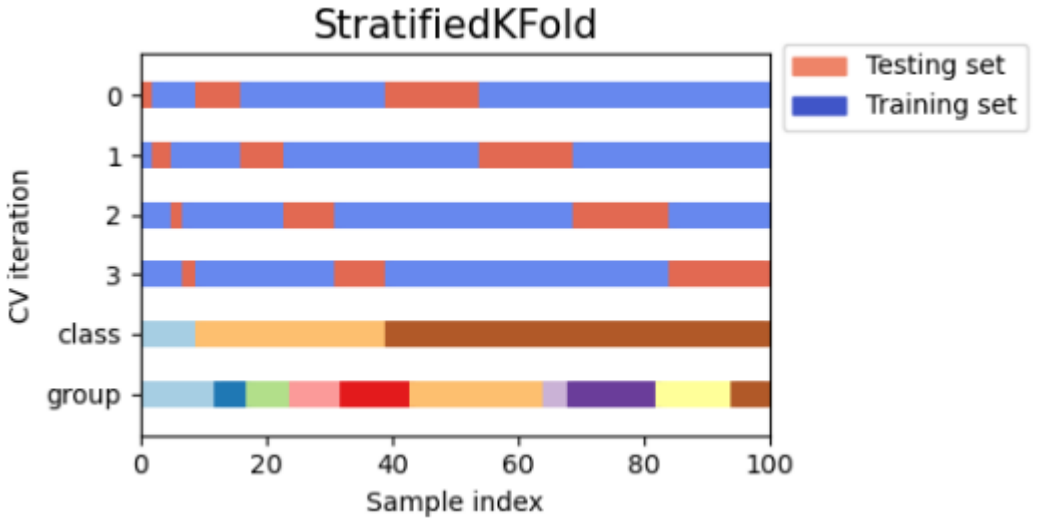


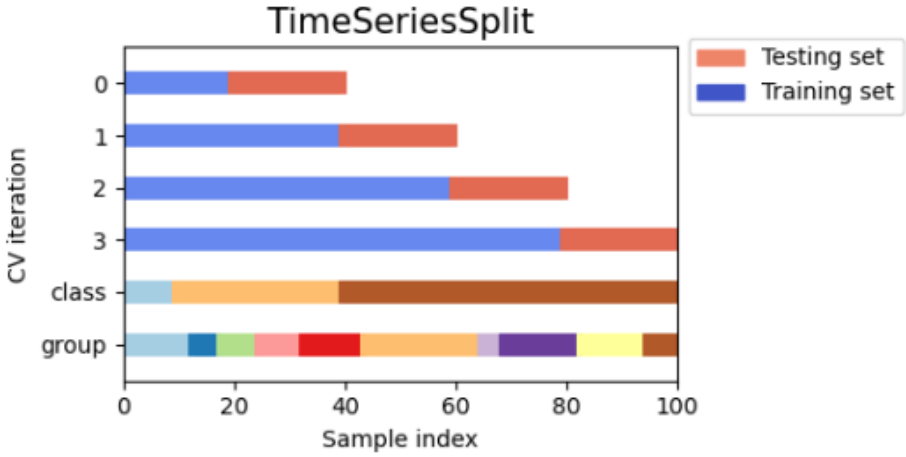

Choose the hyperparameter $\lambda^*$ with the **best** cross-validation error!
This approach is computationally expensive, but does not *waste* data (as the case for arbitrary validation)

# Cross-validation: special cases

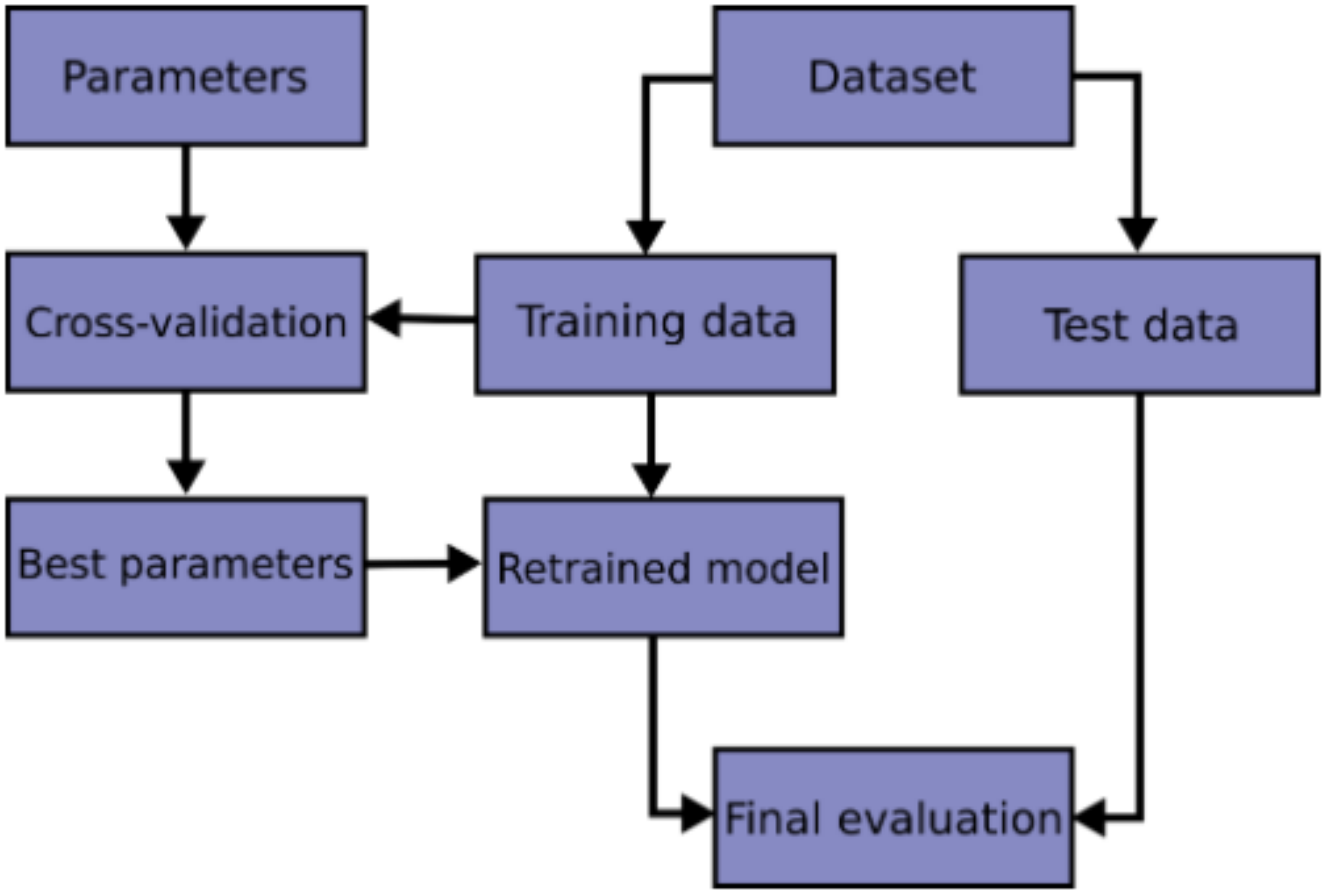
- When applying cross-validation on a classification problem one may want that all the dataset contains approximately the same percentage of samples of each target class as the complete set: this can be done using **Stratified k-fold**

- In this course we do not study **time series model**, but here we want to highlight that in some case cross-validation can be tricky, like in the time-series model setup.



# The general approach



# Conclusions

- There can be some setup in which cross-validation may not work well, overestimating validation error performance.
- There exists other statistical tool (boostrapping methods, montecarlo sampling) which aim to better estimate validation error (for instance by better estimating its variance), but these methods are below the scope of this course.
- **Scaling and any type of preprocessing should be fitted only on train dataset and the used on test data**

# Chapter 6: Tree-based Model

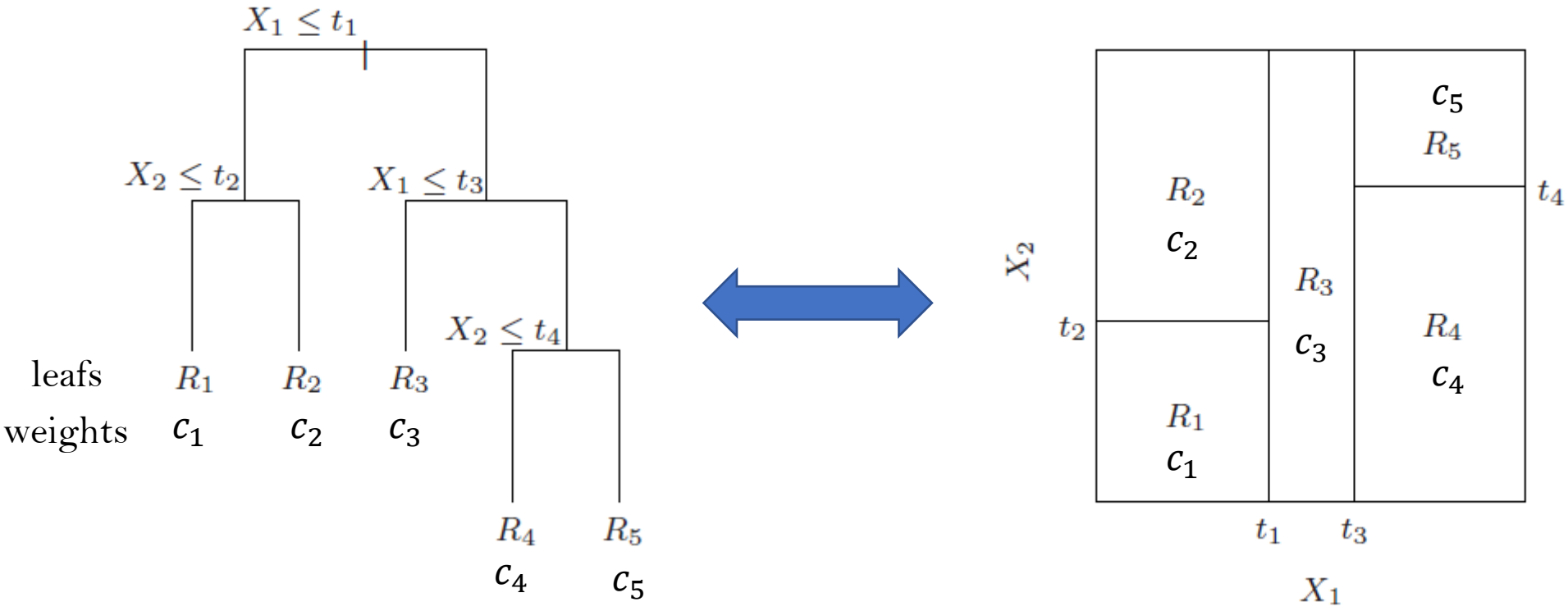
# Motivation

This chapter deals with a new class of model called tree-based model that are widley used both for regression and classification. We will study:

- Simple CART (Classification And Regression Tree)
- Random Forest
- Boosting tree
- Extreme Boosting Tree (XGBoost)

# CART - Regression

Tree-based methods partition the feature space into a set of rectangles (binary partitions), and then fit a simple model (like a constant) in each one

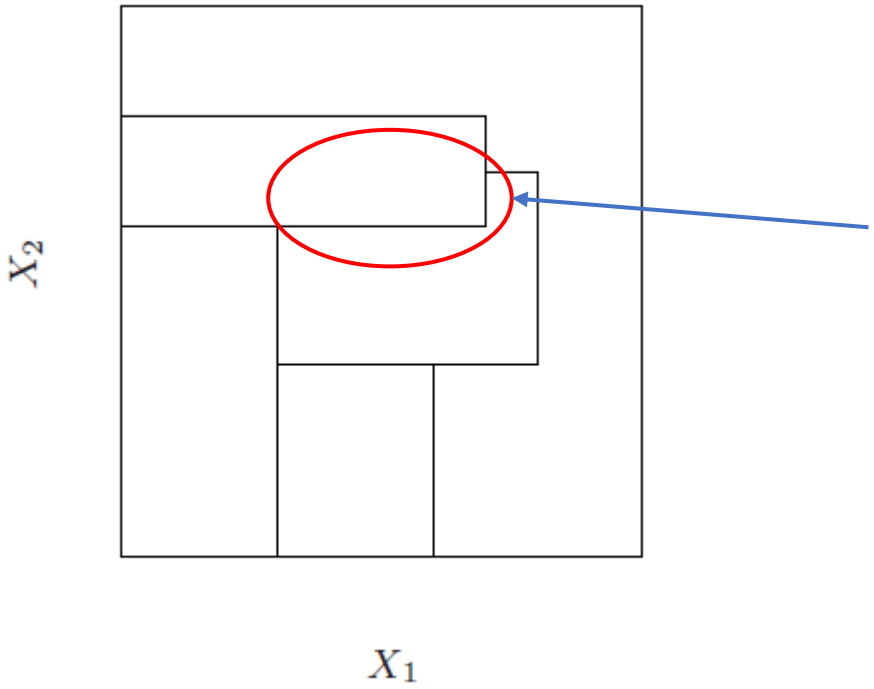



$$f(x) = \sum_{m=1}^{M} c_m \, I(x \in R_m)$$

$I$ is the identity function for the region $R_m$

# CART - Regression

Tree-based methods partition the feature space into a set of rectangles (binary partitions), and then fit a simple model (like a constant) in each one




**This kind of split is not possible because it is a non-binary split**

# CART - Regression

Consider a Dataset $D = \{x_i, y\}$ with $p$ features and $n$ rows **(training dataset)**

$$\beta, \gamma = argmin_{\beta,\gamma} L(Y, f(X; \beta; \gamma))$$

$$f(x) = \sum_{m=1}^{M} c_m \, I(x \in R_m) \qquad R_m^*, c_m^* = argmin_{R_m, c_m} \sum_{i=0}^{n} (y_i - f(x))^2 \quad (1)$$

For regression Mean Squared Error is a good Loss

Suppose first that we already have a partition into M regions $R_1^*, \dots R_M^*$ hence:

$$\frac{d}{dc_m} \sum_{i=0}^{n} (y_i - f(x))^2 = \sum_{i \in R_m^*} 2(y_i - c_m) = 0 \rightarrow \sum_{i \in R_m^*} y_i - \#m \, c_m = 0$$

The coeff $c_m$ compare only
In the sum for the observation $i \in R_m^*$

$$c_m^* = \frac{1}{\#m} \sum_{i \in R_m^*} y_i = ave(y_i | x \in R_m^*)$$

The optimal region $R_1^*, \dots R_M^*$ are computed following a *greedy optimization** algorithm that aim to solve (1)



* Greedy means that the convergence to the real minimum in not guaranteed

# CART – the greedy algorithm

The Region $R$ is defined by a set of two informations: the feature and the split point $\{j, s\}$

Starting with all the data consider a feature $j$ and a split point $s$ and compute:

Residual sum of square

$$RSS = \sum_{i \in R_1} (y_i - c_1)^2 + \sum_{i \in R_2} (y_i - c_2)^2 \quad where \;\; R_1 = \{x | x_j < s\} \; and \; R_2 = \{x | x_j \geq s\} \; and \; c_1 = ave(y_i | x \in R_1)$$

Cycling on all possible valus of $(j, s)$ find those minimizing $RSS \rightarrow (j^*, s^*)$

Then one can repeat recursively the splitting process on each of the two regions.

How large should we grow the tree? → **Regolarization and/or Cross Validation**

# CART – Regolarization

The process described above is likely to overfit the data, how to prevent this problem?

- **Regolarization**: Pruning the tree. This is tipically done by adding to the loss a term $+\lambda|T|$ where $|T|$ is the number of leafs of the tree (terminal nodes) and find the best value of $\lambda$ via cross-validation:

$$argmin_{R_m,c_m} \sum_{i=0}^{n} (y_i - f(x))^2 + \lambda|T|$$

- **Cross-validation**: the tree-depth is an hyperparameter of the model, it can be chosen via cross-validation

# CART – Hyperparameter

- **max_depth**: The maximum depth of the tree **(most important)**
- **min_samples_split:** The minimum number of samples required to split an internal node
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node

They can be chosen via cross-validation

**Example**: For instance, if min_samples_split = 5, and there are 7 samples at an internal node, then the split is allowed. But if the split results in two leaves, one with 1 sample, and another with 6 samples. If min_samples_leaf = 2, then the split won't be allowed (even if the internal node has 7 samples) because one of the leaves resulted will have less then the minimum number of samples required to be at a leaf node.

# CART – Classification

A classification tree is very similar to a regression one. For a classification tree, the leaf prediction is the **class proportions** among the training observations that fall into that region.

Use same growing algorithm as regression tree, but instead of RSS it consider a different split metrics:

Entropy (also called cross-entropy):

$$Entropy = -p_1 \log p_1 - (1 - p_1)\log(1 - p_1)$$

or:

Gini Index:

$$Gini = p_1(1 - p_1)$$

$$where\ \ p_1\ is\ the\ 1 - class\ proportion\ on\ the\ leafs$$

In essence, one have to compare the value of $Entropy$ or Gini between parent and child leaf: if such difference is positive, i.e. E decrease, we can split.

# CART – Classification

Entropy (also called cross-entropy):

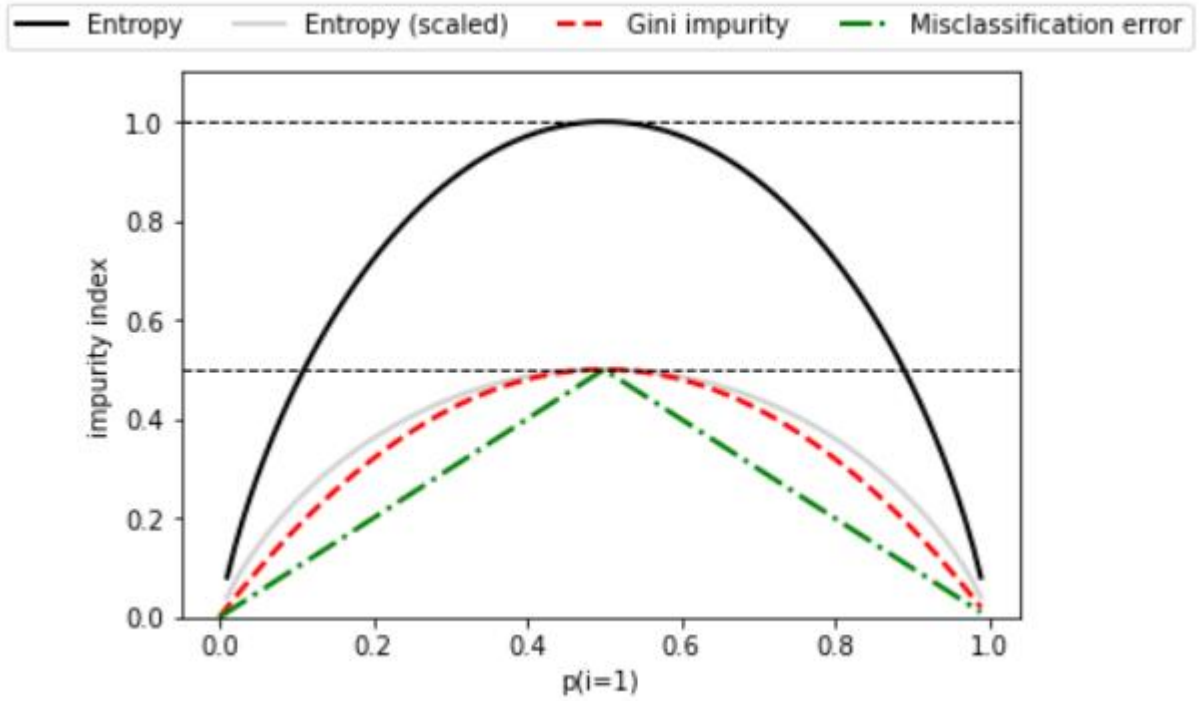
$$Entropy = -p_1 \log p_1 - (1 - p_1)\log(1 - p_1)$$

or:

Gini Index:

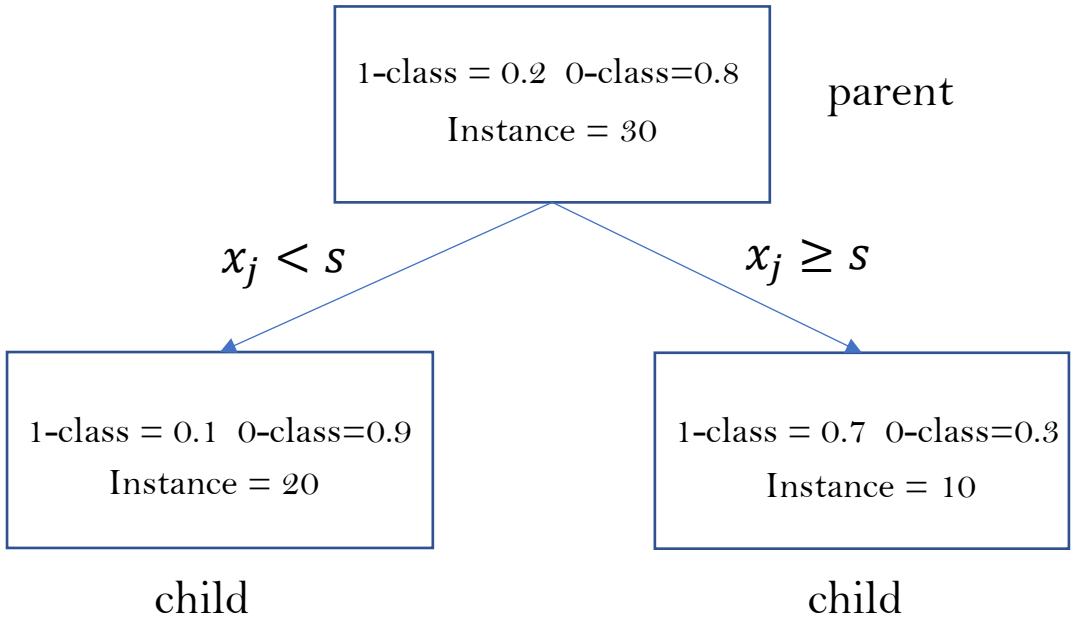
$$Gini = p_1(1 - p_1)$$

$$where\ \ p_1\ is\ the\ 1 - class\ proportion\ on\ the\ leafs$$



# CART – example

Introduce **gain** which **measure how much the entropy (or gini) increase or decrease after a split**




$$Gain = E_{parent} - E_{childs}$$

$$E_{parent} = -0.2\ log0.2 - 0.8 \log 0.8 = 0.50$$

$$E_{childs} = \frac{20}{30}(0.1\ log0.1 + 0.9 \log 0.9) + \frac{10}{30}(0.7log0.70 + 0.3log0.3) = 0.42$$

$$gain = E_{parent} - E_{child} = 0.08$$

Gain is positive thus spit!

$$IG(D_p, X) = I(D_p) - \frac{N_{\text{left}}}{N_p} I(D_{\text{left}}) - \frac{N_{\text{right}}}{N_p} I(D_{\text{right}})$$

If **entropy decrese (gain is positive)** as the tree growth means that the uncertain about the class attribution decrease!

# CART conclusions

## Pros

- Trees are very easy to explain
- Some researchers believe that decision trees more closely mirror human decision-making
- Trees can be displayed graphically and are easily interpreted even by a non-expert (especially if they are small).
- **The feature to put at the root node is the one with the most influence on the outcome. This is essential for business interpretation of the results**

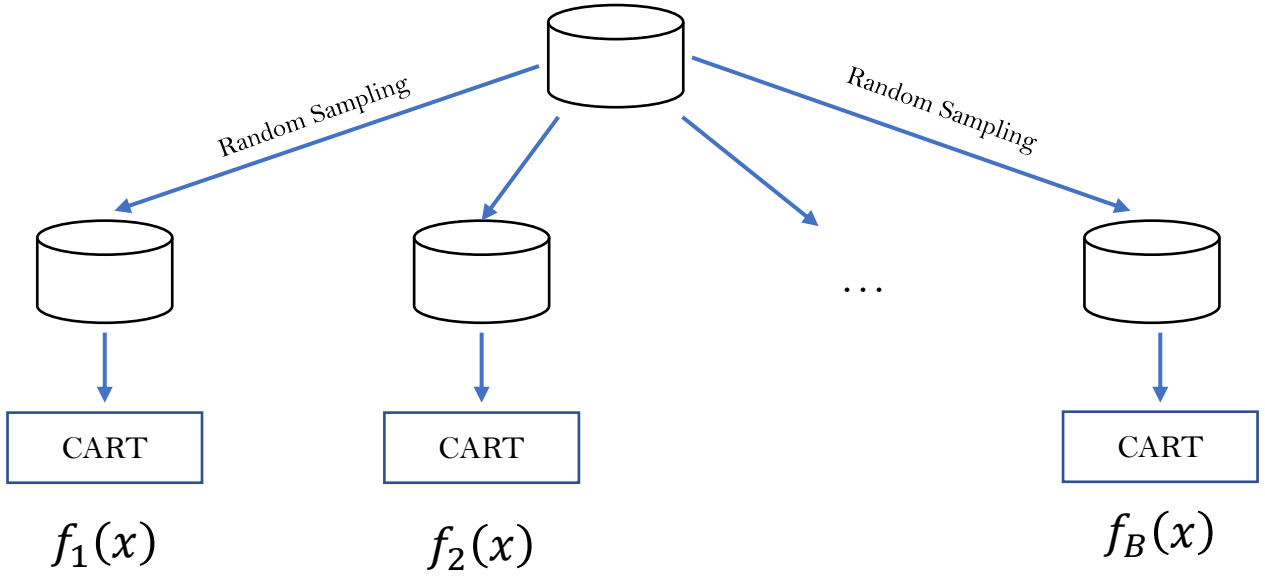
## Cons

- CART generally do not have the same level of predictive performances as some of the other regression and classification approaches seen in this course.
- CART can be non-robust: a small change in the data can cause a large change in the final estimated tree.

# Random Forest (RF)

"Why build one tree when you can make a forest?"

**The idea is to average many CART model to improve performance**




**To decorrelate the tree and avoid very similar CART** at each split in the tree the algorithm is allowed to consider **a random minority of the available predictors**! Indeed, averaging many highly correlated model does not lead to performance improvement and reduction in variance

$$f(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x)$$

For classification you can use:
- majority voting
- mean predicted class probabilities i.e the fraction of samples of the same class in a leaf.

# RF – Hyperparameter

- n_estimators: the number of tree in the forest
- max_features: the number of features to consider during a split
- max_samples: the number of sample for random sampling
- **max_depth**: The maximum depth of the tree
- **min_samples_split:** The minimum number of samples required to split an internal node
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node
- **ccp_alpha:** Complexity parameter used for Minimal Cost-Complexity Pruning.

They can be chosen via cross-validation

# Boosting tree

**Boosting**: build a model that combines the outputs of many **weak** model to have better result.

$$f(x) = \sum_m h_m(x; \gamma_m)$$

Where $h_m$ are a very **simple** function: here a **very shallow** tree is used

As usual we aim to find a solution $\gamma_m$ such as:

$$\gamma_m^* = argmin_{\gamma_m} \sum_i L\big(y_i, f(x)\big)$$

The idea is to solve this problem in a **gradient descent fashion**

# Boosting tree: gradient descent

$$f(x) = \sum_m h_m(x;\gamma_m) \rightarrow f_{m+1} = f_m + h_m$$

Iterative update like gradient descent

$$\beta_{t+1} = \beta_t - \gamma \frac{d}{d\beta} L(\beta)\Big|_{\beta_t}$$

$$h_m \rightarrow -\frac{d}{df_m} L(y, f_m)\Big|_{f_m}$$

Let us just build the model iteratively:

- $f_o$ is a constant ginven by $f_0 = argmin_{f_o} \sum_{i=0}^{n} (y_i - f_0)^2 \rightarrow f_0 = ave(y_i)$
- For each observation compute $r_1 = -\frac{d}{df_0} L(y, f_0)|_{f_0}$
- Fit a regression tree $h_1(x;\gamma_1)$ on $r_1$
- Update $f_1(x) = f_0(x) + \epsilon f_1(x)$ where $\epsilon$ is an input parameter
- Iterate until $m \rightarrow M$

$$L = \left(y_i - f(x)\right)^2$$

$$\frac{d}{df_0} L(y, f_0)|_{f_0} = \frac{d}{df_0} (y_i - f_0)^2$$

$$(y_i - f_0) \rightarrow r_1$$

residual of the previous step !!

Thus $h_1(x;\gamma_1)$ is fitted on the error made by previous predictor

# Boosting tree – Hyperparameter

- **n_estimators**: the number of tree in the forest
- **learning_rate** $\epsilon$ : shrinks the contribution of each tree by learning_rate : $f(x) = f_0 + \epsilon f_1(x) + \ ...$
  There is a trade-off between learning_rate and n_estimators
- **subsample**: The fraction of samples to be used for fitting the individual base learners
- **max_depth**: The maximum depth of the tree
- **min_samples_split:** The minimum number of samples required to split an internal node
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node
- **ccp_alpha:** Complexity parameter used for Minimal Cost-Complexity Pruning.

They can be chosen via cross-validation

# Variable importance

Like in any other ML model, for tree-based model is important to understand the impact of the features on the labels. CART are highly interpretable: the model can be represented by a simple two-dimensional graph. Linear combinations of trees lose this important feature and must therefore be interpreted in a different way.

**RF and Boosting critically improves prediction accuracy at the expense of interpretability.**

We can measure how each feature decrease the *impurity* of the split (the feature with highest decrease is selected for internal node). For each feature we can collect how on average it decreases the Loss, i.e. the splitting criterion. **The average over all trees in the model is the measure of the feature importance.**

$$I_V(N) = \frac{1}{|S|^2}\sum_{i\in S}\sum_{j\in S}\frac{1}{2}(y_i - y_j)^2 - \left(\frac{|S_t|^2}{|S|^2}\frac{1}{|S_t|^2}\sum_{i\in S_t}\sum_{j\in S_t}\frac{1}{2}(y_i - y_j)^2 + \frac{|S_f|^2}{|S|^2}\frac{1}{|S_f|^2}\sum_{i\in S_f}\sum_{j\in S_f}\frac{1}{2}(y_i - y_j)^2\right)$$

Another very popular library for variable importance is SHAP, but it is outside the scope of this course

# Chapter 7: Unsupervised Method : Clustering

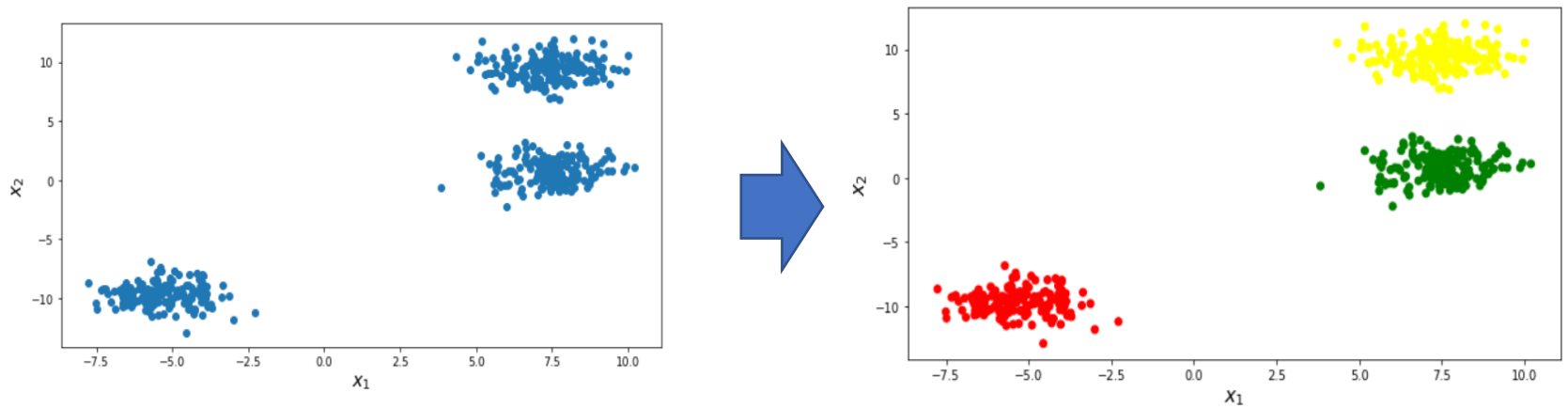
# Unsupervised Learning

- Unsupervised learning algorithms work with data that isn't labelled, i.e no $y$ is available

- It aims to partition data into distinct groups: observations within each group are *"similar"*, while observations in different groups are *"different"*: this is the definition of **clustering**.

**The heart of Clustering methods is to define what similar and different mean!**

- **Example**: Clustering Customers in an online shop:
  - You run a large online store and you would like to personalize the user's shopping experience
  - Each user is described by a list of features
  - Clustering will divide your user base in different group: within each group users will have similar shopping pattern, so they may have similar ads.



# K-means

Defimition of similarity

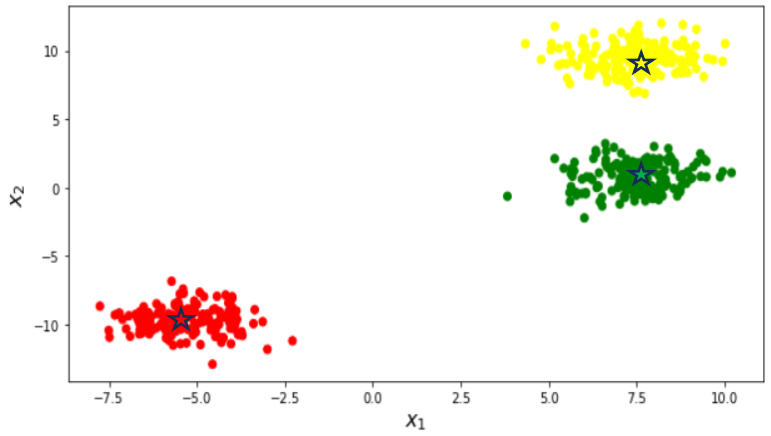
Consider a Dataset $D = \{x_i\}$ with $p$ features and $n$ rows
In the k-means alghorithm **"similarity"** is interpreted as **"distance".**
The simplest distance measure is the Euclidean distance, using vector notation:

$$d(x, y) = (x - y)(x - y)^T$$

$$x \text{ is similar to } y \rightarrow d(x, y) \sim 0 \; (d(x, x) = 0)$$

$$x \text{ not similar to } y \rightarrow d(x, y) \gg 1$$

Defimition of cluster




The clusterization procedure define:

- $\boldsymbol{c_i}$: the cluster assignment for each sample of the dataset (the colors in the Figure)
- $\boldsymbol{\mu_k}$: cluster center (the stars in the Figure)

# K-means

**The idea of K-means is that a good clustering is the one for which the within cluster distance is the smallest possible.** Cluster are found solving the following optimization problem:

$$c_i^*, \mu_k^* = \underset{c,\mu}{argmin} \sum_i \sum_k 1(c_i = k)(x_i - \mu_k)(x_i - \mu_k)^T$$

within-cluster distance

$$\beta, \gamma = \underset{\beta,\gamma}{argmin} L(Y, f(X; \beta; \gamma))$$

The output of k-means is to compute **a vector $c_i$ of cluster assignment for each sample of the dataset** and **k-cluster center $\mu_k$**

- The value of k is fixed (we will see later how to choose it)
- $\mu_k$ are the centroid of the cluster i.e., the "average" element that represent such cluster
- Need to **scale features**
- This suffer for the **curse of dimensionality**

# K-means

$$c_i^*, \mu_k^* = \ argmin_{c,\mu} \sum_i \sum_k 1(c_i = k)(x_i - \mu_k)(x_i - \mu_k)^T$$

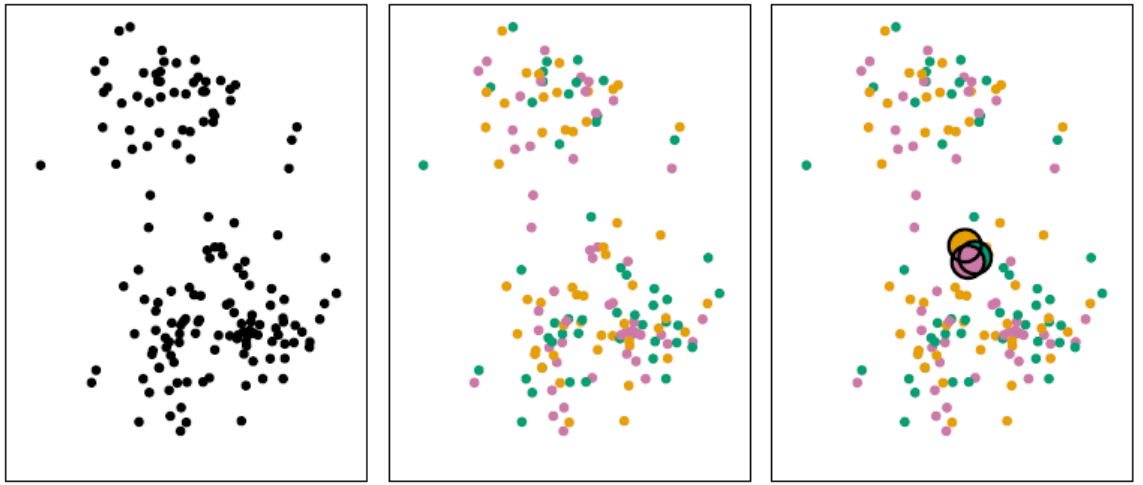
While this problem can be solved via gradient-descent, here we study a different algorithm that converge to the solutions of the problem:

1. Initialization: Randomly assign a cluster, from 1 to K, to each of the observations.
2. For each cluster, compute the cluster centroid as (center of mass, average):

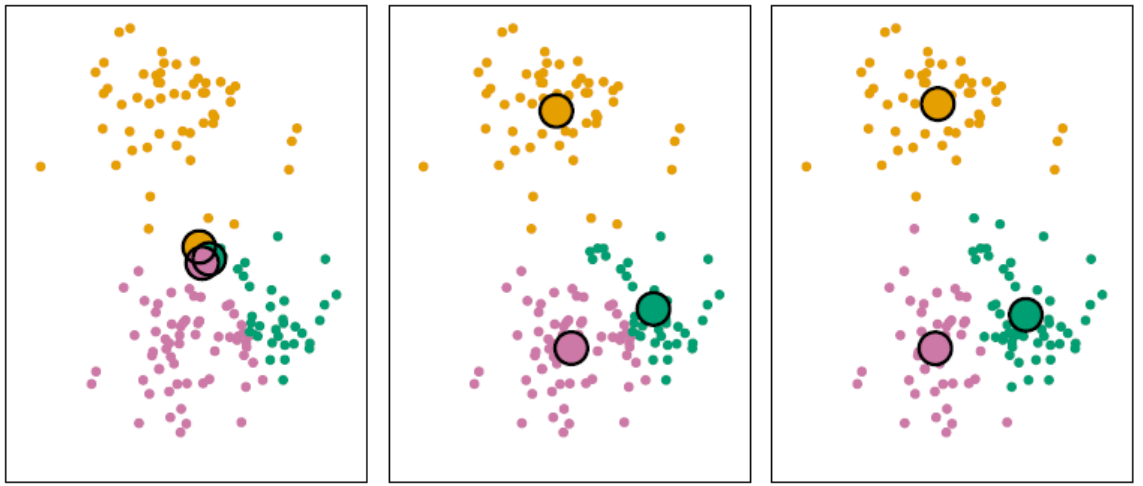
$$\mu_k = \frac{1}{\#l} \sum_{l \in k} x_l$$

3. Assign each observation to the cluster whose centroid is closest, thus computing $c_i$
4. Repeat 2), 3) until $c_i$ does not change. Output $\mu_k$, $c_i$.

# K-means



Iteration 1, Step 2b | Iteration 2, Step 2a | Final Results

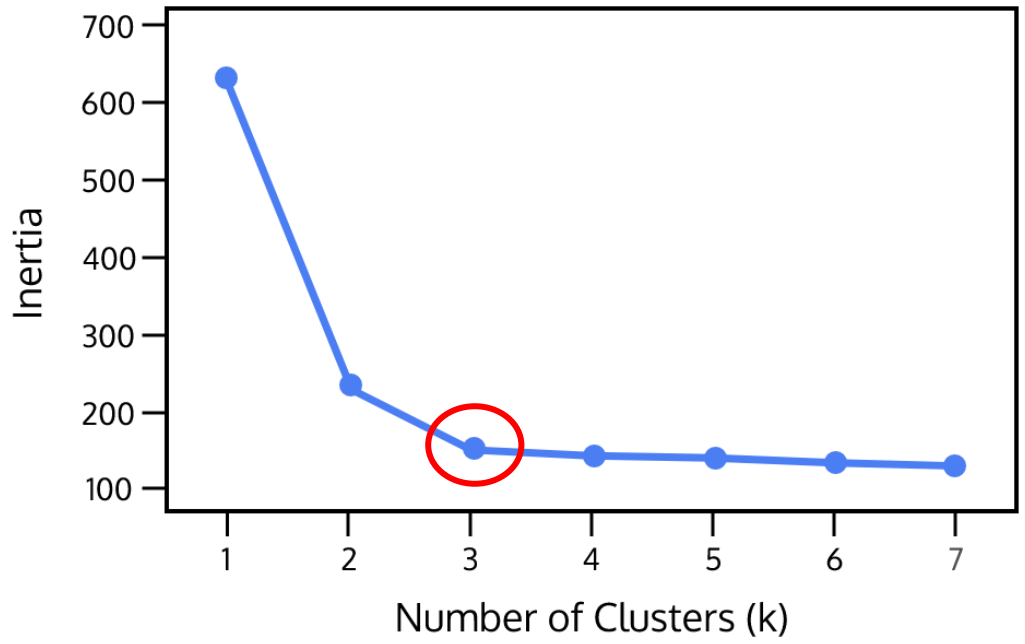


# K-means

How to choose the correct number of cluster?

$$Inertia = \sum_i d_i^2$$

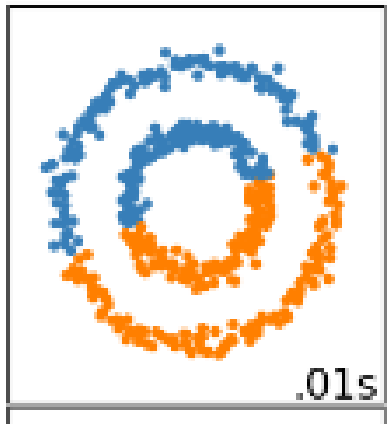
Compute the total distance to the cluster centroid for each value of k →
This is the same as the optimal Loss Value




$k = 3$ is the optimal value since the Inertia does not decrease anymore

**Inertia is also a performance metric**

# Kernelised K-means




With some dataset k-means could fail to extract the "correct" cluster. This is due to the definition of distance that has been used (for instance Euclidean).

**This can be solved used a kernelized k-means algorithm**.

$$c_i^*, \mu_k^* = argmin_{c,\mu} - \sum_i \sum_k 1(c_i = k) k(x_i, \mu_k)$$

Example o kernel $K(x, y) = e^{-\frac{|x-y|^2}{\sigma^2}}$

# Kernelised K-means

$$c_i^*, \mu_k^* = \ argmin_{c,\mu} \sum_i \sum_k 1(c_i = k) k(x_i, \mu_k)$$

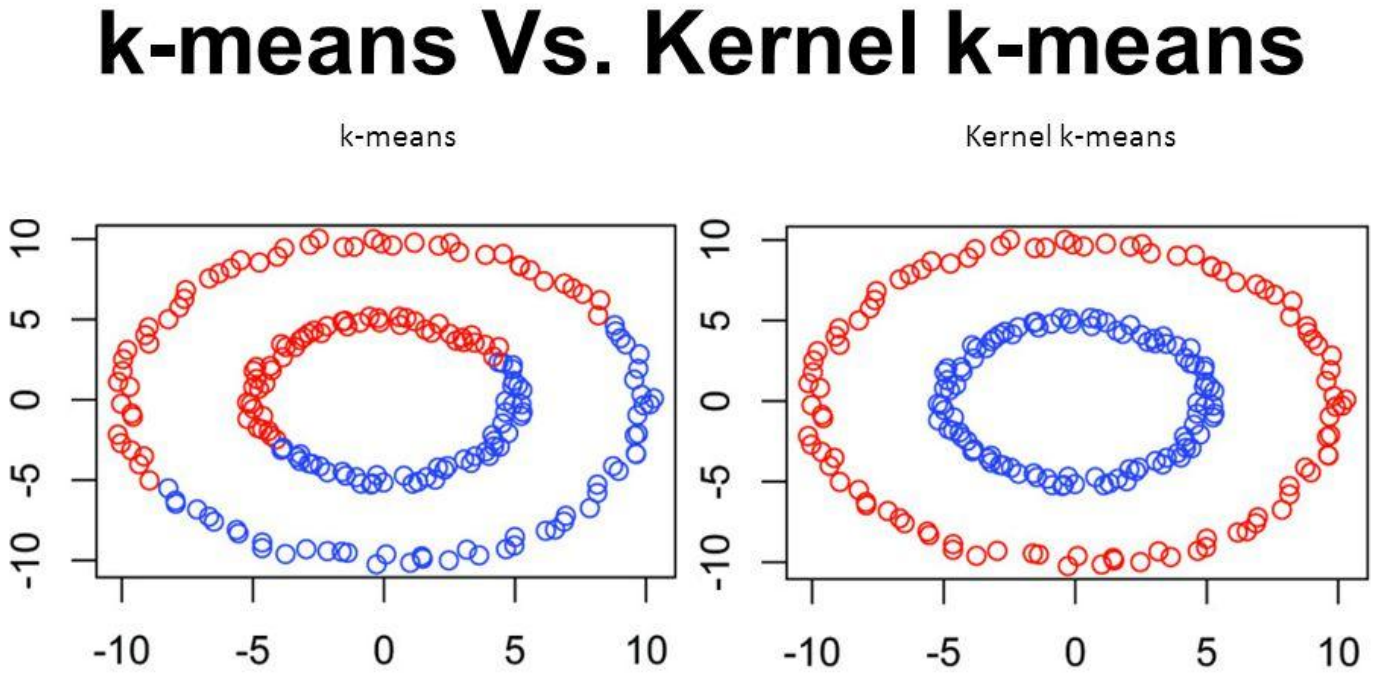
1. Initialization: Randomly assign a cluster, from 1 to K, to each of the observations.
2. For each cluster, compute the cluster centroid as (center of mass, average):

$$\mu_k = \frac{1}{\#l} \sum_{l \in k} \phi(x_l)$$

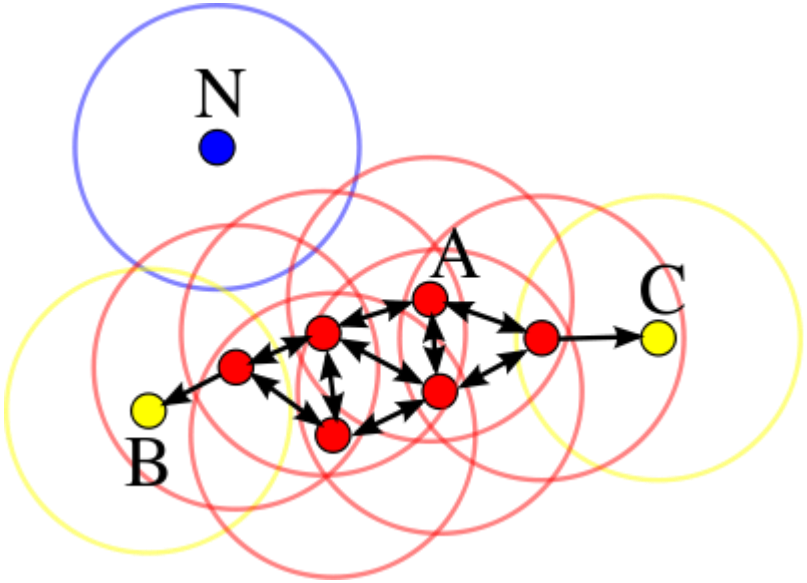
3. Assign each observation to the cluster whose centroid is closest, thus computing $c_i$
4. Repeat 2), 3) until $c_i$ does not change. Output $\mu_k$, $c_i$.

**For this algorithm we need to access to the feature $\boldsymbol{\phi(x_l)}$**

# Kernelised K-means



# DBSCAN




**Density-based spatial clustering (DBSCAN)** is a very popular clustering alghoritm

Some key idea are (defined two parameter MinPts and $\epsilon$):

- red points are core points, because circle of $\varepsilon$-radius contain at least MinPts points (including the point itself).
- Yellow points are frontier points, and they belong to the cluster
- Point N is a noise point that is neither a core point nor directly-reachable.

1.DBSCAN does not require one to specify the number of clusters as opposed to k-means
2.DBSCAN can find arbitrarily-shaped clusters like those on previous slide
3.DBSCAN can be used as an outliers-detection methods:N-points are the **outliers**

# Backup

Matteo Amabili

matteo.amabili@leitha.eu

# More on tree

# Extreme Boosting tree

Same principles as Boostin method→ write $f(x)$ as a sum of "weak" learner (shallow tree)

$$f(x) = \sum_m h_m(x; \gamma_m) \qquad f(x) = \sum_m h_m(x; \gamma_m) \rightarrow f_{m+1} = f_m + h_m$$

At the $T$ stage we have:

$$L_T = \sum_i L\big(y_i, f_T(x)\big) = \sum_i L\big(y_i, f_{T-1} + h_m(x_i)\big) \qquad f_{T-1} = \hat{y}_{T-1}(x_i)$$

Expanding the Loss-function to the second order via Taylor:

$$L_T = \sum_i L(y_i, y_{T-1}) + \frac{dL}{d\hat{y}} h_m(x_i) + \frac{1}{2}\frac{d^2L}{d^2\,\hat{y}} h_m^2(x_i)$$

Neglecting constant terms and using a more compact notation:

$$L_T = \sum_i g_T h_m(x_i) + \frac{1}{2} l_T h_m^2(x_i)$$

# Extreme Boosting tree

$$L_T = \sum_i g_T h_m(x_i) + \frac{1}{2} l_T h_m^2(x_i)$$

Suppose first that we already have a partition into M regions $R_1^*, \dots R_M^*$ hence:

$$h_m(x) = \sum_{m=1}^{M} w_i I(x \in R_i)$$

$$\frac{d}{dw_k} L_T = \sum_{i \in R_k^*} g_T + w_k l_T = 0 \rightarrow \quad w_k^* = \frac{\sum_{i \in R_k^*} g_T(x_i)}{\sum_{i \in R_k^*} l_T(x_i)}$$

The coeff $w_k$ compare only
In the sum for the observation $i \in R_k^*$

Then use CART algortihm to build the tree based on loss $L_T$ and optimal weight $w_k^*$

# Extreme Boosting tree: alternative alg.

$$L_T = \sum_i g_t h_m(x_i) + \frac{1}{2} l_t h_m^2(x_i)$$

Neglecting constant terms

$$L_T = \sum_i \frac{1}{2} l_t(x_i) \left( -\frac{g_t}{l_t} - h_m(x_i) \right)^2$$

- $f_o$ is a constant
- For each observation $x_i$ compute $g_i = \frac{d}{df_0} L(y, f_0)|_{f_0}$ and $l_i = \frac{d^2}{d^2 f_0} L(y, f_0)|_{f_0}$
- Fit a regression tree $h_1(x; \gamma_1)$ on the dataset $\left\{ x_i, -\frac{g_t}{l_t} \right\}$ minimizing $L_T = \sum_i \frac{1}{2} l_t(x_i) \left( -\frac{g_t}{l_t} - h_m(x_i) \right)^2$
- Update $f_1(x) = f_0(x) + \epsilon f_1(x)$ where $\epsilon$ is an input parameter
- Iterate until $m \rightarrow M$

$$L = (y_i - f(x))^2$$

$$\frac{d}{df_0} L(y, f_0)|_{f_0} = (y_i - f_0)$$

$$\frac{d^2}{d^2 f_0} L(y, f_0) \Big|_{f_0} = -1$$

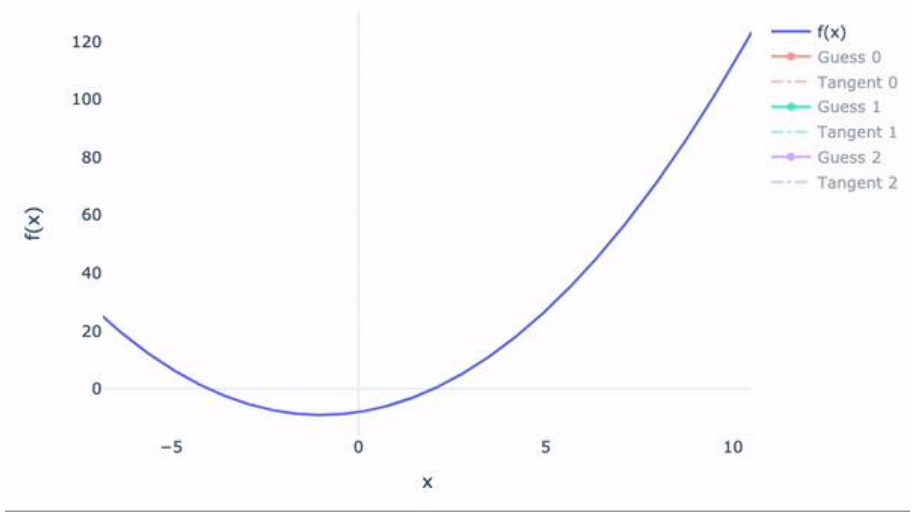
# Extreme Boosting and Newton methods

$$L_T = \sum_i \frac{1}{2} l_t(x_i) \left( -\frac{g_t}{l_t} - h_m(x_i) \right)^2$$

…why the term $-\frac{g_t}{l_t}$ is not a surprise?

Suppose we want to solve a minimization problem (equivalent to root finding):

$$\frac{d}{dx} f(x) = 0 \quad (1)$$




Newton Raphson Method:

$$x_0 = x_0$$

$$x_{t+1} = x_t - \frac{\frac{d}{dx} f(x)|_{x_t}}{\frac{d^2}{d^2 x} f(x)|_{x_t}}$$

At convergence $x_t$ is a solution of $(1)$

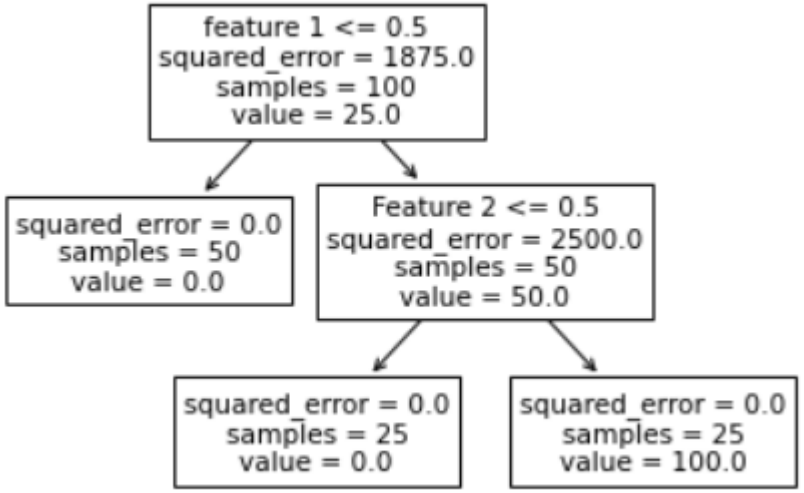
There is a sort of analogy between Newton Raphson and XGBoost algorithm!

# More on variable importance

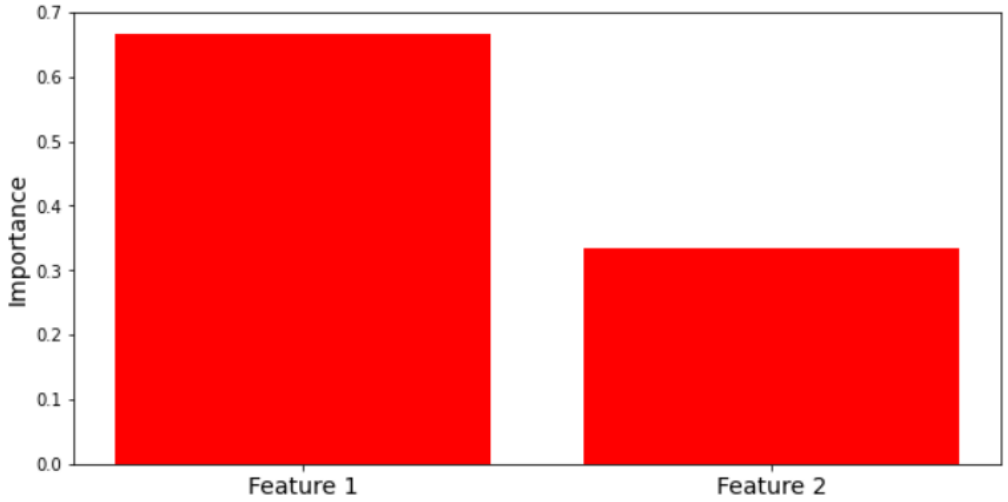
# Inconsistency of variable importance

feature attribution methods introduce above are often inconsistent, meaning they can assign higher importance to features with a lower impact on the model's output.

$$y = (feature_1 \,\&\, feature_2) * 100$$




We expect that feature 1 and feature 2 have same importance but..




Result of model.feature_importances_

Standard importance attribution methods gives **an average picture of the contribution of the features**, but we are more interested in **understanding why a model made a prediction on a given record**

# SHAP method

**IDEA: You do not understand the importance of the variables until you do not have it**

Let us introduce a simpler explanation model $g(z)$ used to explain the more complex model $f(x)$

$$g(z) = \phi_0 + \sum_P \phi_p z_p$$

$z \in \{0,1\}^M$ where $M$ is the total number of features.
It represents a feature being observed ($z_i = 1$) or unknown ($z_i = 0$)

$\phi_i$ are the features importance values

We want that given an obervation $\{x_i\} \rightarrow f(x_i) = g(z = [1, \dots 1])$. In other word $\phi_i$ are a function of $\{x_i\}$

# SHAP method

How to compute $\phi_i$? **Using Shap-values from game theory!**

All the possible model with the set $S$

Model considering the set S without the feature $i$

$$\phi_i^j = \sum_{S \leq M - \{i\}} \frac{|S|!\,(|M| - |S| - 1)!}{|M|!} \left(f_{S+\{i\}}(x_j) - f_S(x_j)\right)$$

$S$ are the subsets of all M features

Model considering the set S **and** the feature $i$

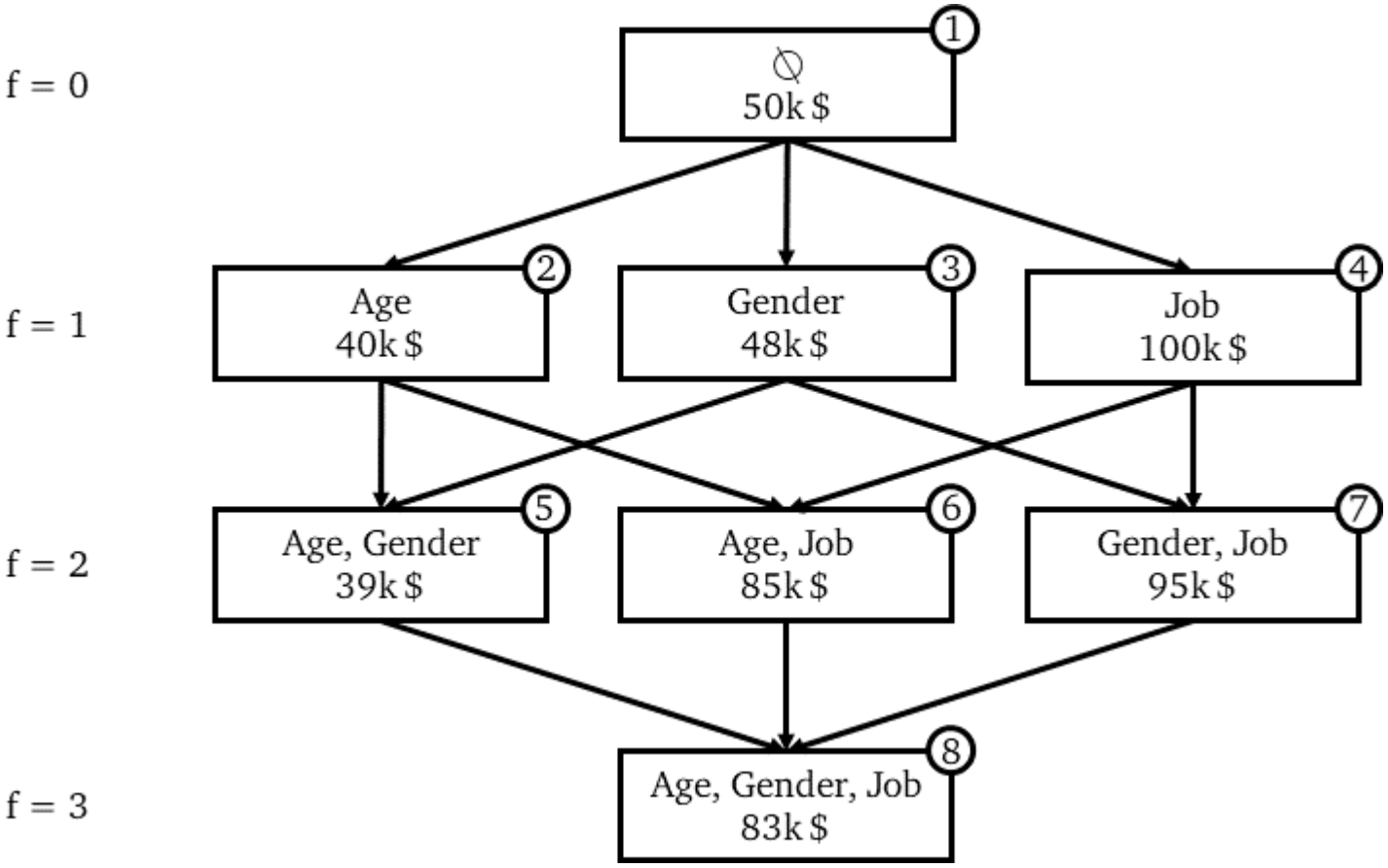
$\phi_0$ is the value predicted by the model without features, for regression is the average of $y$

**Naive Implementation**: Fit $2^M$ model with all possible subset from the set of features and then compute $\phi_i$

**Real Implementation**: For tree model shap values can be computed exactly (why?), for other model there are some approximations!!

# SHAP method: example

$M = \{age, gender, job\}$




We want the importance of feature the age!

| | without {age} | With {age} | difference | weight |
|---|---|---|---|---|
| ∅ | 50 | 40 | -10 | $\frac{2!}{3!} = \frac{1}{3}$ |
| {Job} | 100 | 85 | -15 | $\frac{1!}{3!} = \frac{1}{6}$ |
| {Gender} | 48 | 39 | -9 | $\frac{1!}{3!} = \frac{1}{6}$ |
| {Job, Gender} | 95 | 83 | -12 | $\frac{2!}{3!} = \frac{1}{3}$ |

$$\phi_{age} = -11.33$$

# SHAP method for classification

SHAP algorithm computes the SHAP values with respect to the margin not the transformed probability, the values you obtain are **log odds values!**

$$\log\frac{p}{1-p} = g(z) = \phi_0 + \sum_P \phi_p z_p$$