Index of contents

Chapter 1 – Introduction to Machine Learning

Machine learning was born because of the need of an automatic analysis of data which leads to human-like decisions. It is a set of methods that can automatically detect patterns in data to predict the future.

It has got many definitions during the years. The most general and accurate engineering definition of Machine Learning might be:

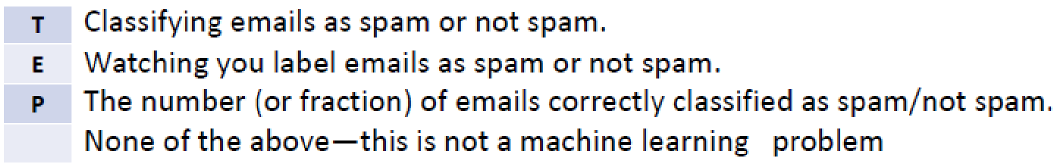
> "A computer program is said to learn from experience (E) with some class of tasks (T) and a performance measure (P) if its performance at tasks in T as measured by P improves with E" (Tom Mitchell)

So, we have a computer program that must do a class of tasks T and a set of experience E The purpose of the computer program is to "solve" a task, which means measuring the *performance* P to evaluate the learning process.

Basically, learning means improving with experience (a task, with respect to performance measure, based on experience).

Consider the following example:

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam.
What is the task T in this setting? The experience E? the performance P?



T	Classifying emails as spam or not spam.
E	Watching you label emails as spam or not spam.
P	The number (or fraction) of emails correctly classified as spam/not spam.
	None of the above—this is not a machine learning problem

1st classification – Type of Learning

Supervised Learning

Supervised Learning (also called **predictive learning**) is defined as when a model gets trained on a "Labelled Dataset", which have both input and output parameters. So, these algorithms learn to map points between inputs x and correct outputs y.
Hence, we define the dataset $\boldsymbol{D}$:

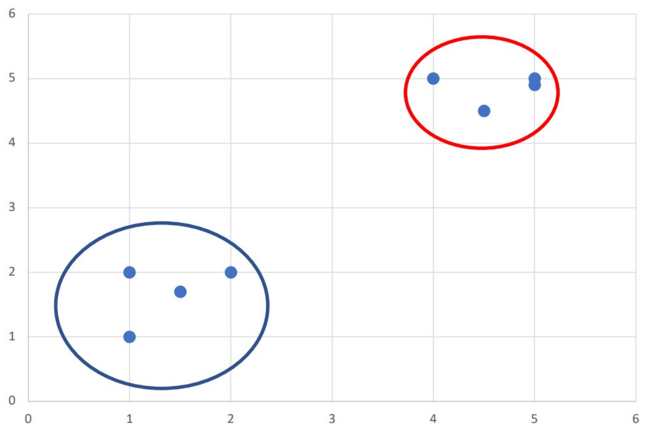
$$\boldsymbol{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$$

where:

- $\boldsymbol{N}$ is the number of pairs of the training set;
- $\boldsymbol{x}_i$ is the vector of inputs where each component is called **feature**, **attribute** or **covariates**;
- $\boldsymbol{y}_i$ is the vector of the outcomes and can be, theoretically, everything.
  In general, if the outcome is a categorial question, we talk about a **classification problem**. Instead, if $\boldsymbol{y}_i$ is a real value it's a **regression problem**.

Unsupervised Learning

Unsupervised learning is a type of machine learning technique in which an algorithm discovers patterns and relationships using unlabelled data. Unlike supervised learning, unsupervised learning doesn't involve providing the algorithm with labeled target outputs. The primary goal of Unsupervised learning is often to discover hidden patterns, similarities, or clusters within the data.



Reinforcement Learning

Reinforcement Learning is a learning method that interacts with the environment by producing actions and discovering errors. This method allows machines to automatically determine the ideal behaviour within specific context in order to maximize performance, and it is crucial for applications that involve decision-making in unpredictable environments.
It aims on act or behave when given an occasional reward or a punishment signal. So, for example, if we want to train a robot to go from point A to point B, programmers will incentivise it to move in a way rather than another.

Recommender Systems

These systems are based on the collaborative behaviour. So, users with the same behaviour will move in the same way in the future.
These are used for applications used to recommend movies, articles on the web, song, video and so on.

2nd classification – Type of Models

Discriminative models

These models are trained over a training set. When they take an input, they estimate the **most probable output**. The purpose of these models is to estimate the conditional probability $\boldsymbol{p(y|x)}$ (probability of y given x).

Generative models

The purpose of these models is to estimate the joint probability $\boldsymbol{p(y,x)}$. These are probabilistic models that produce both input and output. After the model is trained, the conditional probability can be inferred.

Chapter 2 – Supervised Learning

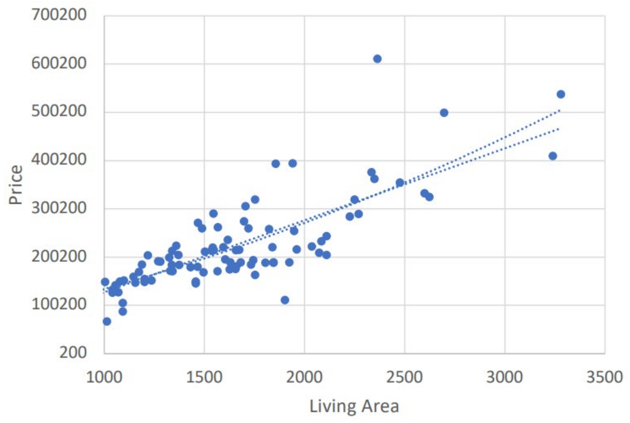
Let's consider the **housing cost prediction problem**, using the following *training set*:

Living area (feet2)	Price (1000$s)
1656	215
896	105
1329	172
2110	244
…	…

The notation we'll use is:

- $\boldsymbol{m}$: dimension of the training set
- $\boldsymbol{x}$: input variable
- $\boldsymbol{y}$: output variable
- $(\boldsymbol{x}, \boldsymbol{y})$: one single training sample
- $\left(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\right) = \boldsymbol{i}^{th}$: training sample

In the following graph there are shown data points obtained from the training set:

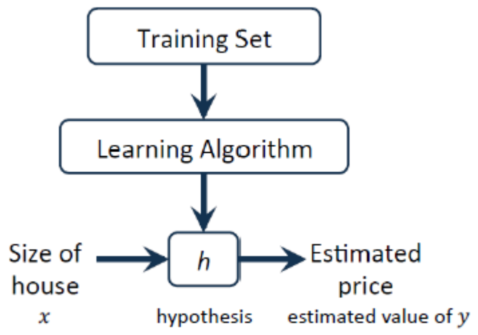


We want to build a function $\boldsymbol{h}$ that, given a new input, returns a proper output. This function $\boldsymbol{h}$ is named **hypothesis function**. It includes a set of parameters θ and then our goal will be how to find values for these parameters θ that make:

$$\boldsymbol{h_\theta(x_i) \approx y_i}$$

In general, the training process consists in:

1) collecting data (ex: tabular format);
2) filtering and "cleaning" data;
3) once we have data we can trust, training the model by defining the mathematic function $\boldsymbol{h}$.



The purpose is to realize a function that, given a new value of x, it will return the estimated value of y.

To achieve this, we need something to measure the error that the hypothesis commits when making predictions. Hence, it is defined the **general cost function** across all the training samples as:

$$\boldsymbol{J(\theta) = \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2}$$

Univariate Linear Regression

The simplest hypothesis is the **univariate linear regression** (with one instance of $\boldsymbol{x}$), defined as following:

$$\boldsymbol{h_\theta\left(x^{(i)}\right) = \theta_0 + \theta_1 \cdot x^{(i)}}$$

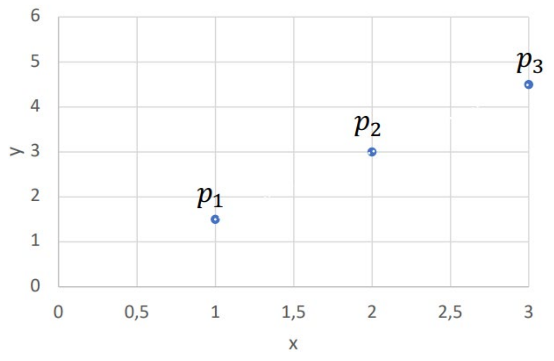
Our goal is to find the **parameters** θs (also called **weights**) such that the hypothesis will result as close as possible to $\boldsymbol{y}$ for our training samples.
First of all, the cost function is defined as:

$$\boldsymbol{J(\theta_0, \theta_1) = \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2}$$

So, the problem is defined as follows:

$$\theta_{min} = \underset{\theta_0,\theta_1}{\operatorname{argmin}} J(\theta_0, \theta_1) = \underset{\theta_0,\theta_1}{\operatorname{argmin}} \left(\frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2\right)$$

Consider the following example where we have only 3 data points (p_1, p_2, p_3):

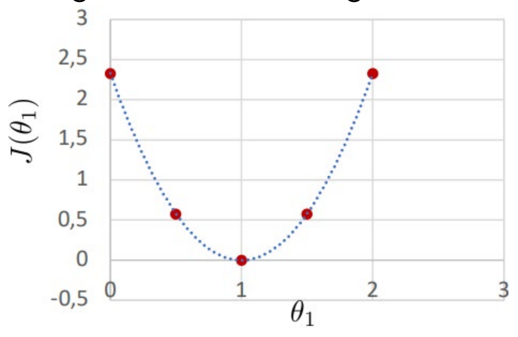


$$p_1 = (1, 1.5)$$
$$p_2 = (2, 3)$$
$$p_3 = (3, 4.5)$$

The cost function is:

$$J(\theta) = \frac{1}{2 \cdot \underset{\underset{m}{\uparrow}}{3}} \left(\left(\overbrace{\theta_0 + \underset{\underset{x^{(1)}}{\uparrow}}{1} \cdot \theta_1}^{h_\theta\left(x^{(1)}\right)} - \underset{\underset{y^{(1)}}{\uparrow}}{1.5}\right)^2 + (\theta_0 + 2 \cdot \theta_1 - 3)^2 + (\theta_0 + 3 \cdot \theta_1 - 4.5)^2\right)$$

For simplicity, we'll consider in this case $\theta_0 = 0$ while we'll consider different values of θ_1, such as $\{0, 0.5, 1\}$. What we get is the following function:



We can see that the graph of the cost function is a convex curve with only <u>one absolute minimum value</u>. Our goal is to minimize the cost function, so we'll find this minimum:

function:	$J(\theta_0, \theta_1)$	$J(\theta_0, \dots, \theta_n)$
goal:	$min\left(J(\theta_0, \theta_1)\right)$	$min\left(J(\theta_0, \dots, \theta_n)\right)$

Algorithm and How it works: Gradient Descent (GD)

In the end, we're looking for a couple of θ_0^* and θ_1^* so that the cost function is:

$$\boldsymbol{J(\theta_0^*, \theta_1^*) \approx 0}$$

The procedure for finding θ_{min} starts by initializing θ_0, θ_1 to some random values and then keep changing them to reduce the function $J(\theta_0, \theta_1)$ to get closer and reach the minimum.

One way to do this is using the ***Gradient Descent algorithm***, which works in the following way:

```
Repeat *until convergence* {
    $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$
    $(forj = 0 \quad and \quad j = 1)$
}
```

Considerations

The Gradient Descent algorithm uses the derivatives in order to find the direction to the **nearest minimum**, which is not exactly equal to finding the *global minimum* unless the function is convex (ex: quadratic function).

It is also used the parameter $\boldsymbol{\alpha}$, named as **learning rate**, which is defined as $\boldsymbol{\alpha > 0}$ and it indicates how much we should move from the starting point to the next point, presumably nearer to a minimum.

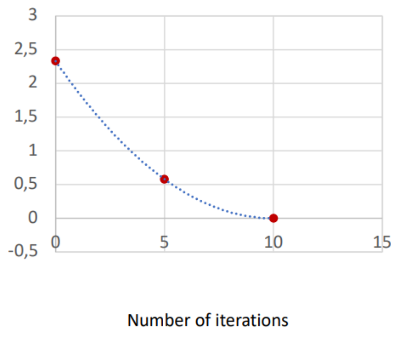
N.B. By the point of view of the algorithm, **converging** means that we get closer and closer to the minimum. It is possible that we never reach the exact minimum but practically there will be some **stopping conditions**, such as:

- **max iteration**: it stops after a fixed number of iterations;
- **absolute tolerance**: it stops when a fixed value of the cost function is reached;
- **relative tolerance**: it stops when the decreasing of the cost function, with respect to the previous step, is below a fixed rate;
- **gradient norm tolerance**: it stops when the norm of the gradient is lower than a fixed value.

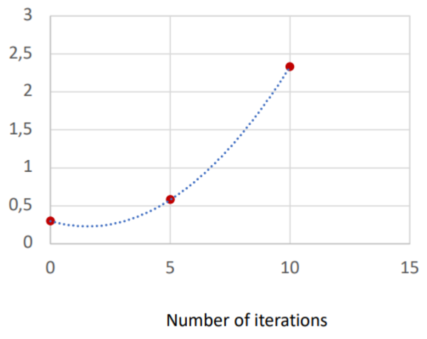
So, depending on the values of $\boldsymbol{\alpha}$:

- if $\boldsymbol{\alpha}$ is too small, gradient descent will converge slowly;
- if $\boldsymbol{\alpha}$ is too large, gradient descent will diverge.

In some cases, it may even cause the GD to bounce.

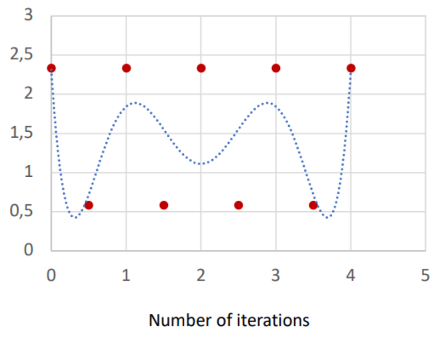


Gradient descent is working



Gradient descent is **NOT** working

Use smaller alpha



Gradient descent is **NOT** working

use smaller alpha

Moreover, in the formula we can see that the second term is negated:

$$-\alpha \frac{\partial\, J(\theta)}{\partial\, \theta_j}$$

and it is because if the derivative part is:

- a **negative number**: the new $\boldsymbol{\theta}$ will move from left to right.
- a **positive number**: the new $\boldsymbol{\theta}$ will move from right to left.

Deriving the formula (Univariate LR)

So, we have defined the cost function as:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

As we previously said, we have to compute the partial derivatives with respect to θ_0 and θ_1. Thus:

$$\frac{\partial\, J(\theta_0, \theta_1)}{\partial\, \theta_j} = \frac{\partial}{\partial\, \theta_j}\left(\frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2\right) =$$

$$= \frac{1}{2m} \cdot \sum_{i=1}^{m} \frac{\partial}{\partial\, \theta_j} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 =$$

$$= \frac{1}{m} \cdot \sum_{i=1}^{m} \frac{\partial}{\partial\, \theta_j} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot \frac{\partial\, h_\theta\left(x^{(i)}\right)}{\partial\, \theta_j}$$

Now, the two partial derivatives over the hypothesis are:

$$\boldsymbol{j = 0}: \quad \frac{\partial\, h_\theta\left(x^{(i)}\right)}{\partial\, \theta_0} = \frac{\partial\left(\theta_0 + \cancel{\theta_1 \cdot x^{(i)}}\right)}{\partial\, \theta_0} = \mathbf{1}$$

$$\boldsymbol{j = 1}: \quad \frac{\partial\, h_\theta\left(x^{(i)}\right)}{\partial\, \theta_1} = \frac{\partial\left(\cancel{\theta_0} + \theta_1 \cdot x^{(i)}\right)}{\partial\, \theta_1} = \boldsymbol{x^{(i)}}$$

In the end, we get:

$$\boldsymbol{j = 0}: \quad \frac{\partial\, J(\theta_0, \theta_1)}{\partial\, \theta_0} = \frac{1}{m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)$$

$$\boldsymbol{j = 1}: \quad \frac{\partial J(\theta_0, \theta_1)}{\partial\, \theta_1} = \frac{1}{m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot x^{(i)}$$

The gradient descent algorithm becomes like this:

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(\mathbf{x}^{(\mathbf{i})}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(\mathbf{x}^{(\mathbf{i})}) - y^{(i)})\mathbf{x}^{(\mathbf{i})}$$

}

Multivariate Linear regression

Let us consider now the **multivariate linear regression** (with multiple variables). We'll face a slight change in the notation, such that:

- $\boldsymbol{m}$: number of training samples
- $\boldsymbol{n}$: number of features
- $\boldsymbol{x}^{(i)}$: input features of the $\boldsymbol{i}^{th}$ training sample
- $\boldsymbol{y}^{(i)}$: output variable
- $\boldsymbol{x}_j^{(i)}$: value of feature $\boldsymbol{j}$ in $\boldsymbol{i}^{th}$ training sample

So, the linear hypothesis now considered is:

$$\boldsymbol{h_\theta\left(x^{(i)}\right) = \theta_0 \cdot x_0^{(i)} + \theta_1 \cdot x_1^{(i)} + \cdots + \theta_n \cdot x_n^{(i)}}$$

We can write all the inputs and parameters in a vectorial form, such that:

$$\boldsymbol{x}^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \dots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

Since a matrix multiplication is a *cross product*, such that $X^{n\cdot m} \times Y^{m\cdot t} = Z^{n\cdot t}$, we'll write the hypothesis $h_\theta\left(x^{(i)}\right)$ in a vectorial form as follows:

$$\boldsymbol{h_\theta\left(x^{(i)}\right) = \theta^T x^{(i)}}$$

where $\boldsymbol{\theta^T}$ is defined as the transposed matrix of $\boldsymbol{\theta}$ (horizontal vector).

For convenience, we'll set $\boldsymbol{x_0^{(i)}} = \mathbf{1}$ and then generalize the formula for $j = 0, 1, \dots, n$:

$$\frac{\partial\, J(\theta)}{\partial\, \theta_j} = \frac{\partial\, J(\theta_0, \theta_1, \dots, \theta_n)}{\partial\, \theta_j} = \frac{\partial}{\partial\, \theta_j}\left(\frac{1}{2m} \cdot \sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2\right) =$$

$$= \frac{1}{2m} \cdot \sum_{i=1}^{m} \frac{\partial}{\partial\, \theta_j}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 =$$

$$= \frac{1}{m} \cdot \sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot \frac{\partial\, h_\theta\left(x^{(i)}\right)}{\partial\, \theta_j}$$

As done before, we can calculate the partial derivatives of $h_\theta\left(x^{(i)}\right)$ over θ_j:

$$\frac{\partial\, h_\theta\left(x^{(i)}\right)}{\partial\, \theta_j} = \frac{\partial\left(\theta_0 + \theta_1 \cdot x_1^{(i)} + \cdots + \theta_n \cdot x_n^{(i)}\right)}{\partial\, \theta_j} = \boldsymbol{x_j^{(i)}} \qquad \Longrightarrow$$

$$\Longrightarrow \qquad \boldsymbol{\frac{\partial\, J(\theta)}{\partial\, \theta_j} = \frac{1}{m} \cdot \sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot x_j^{(i)}}$$

Gradient Descent – Batch, Stochastic, Mini-Batch

Full-Batch Gradient Descent

1) Randomly initialize parameters
2) Repeat until convergence:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(\mathbf{x}^{(\mathbf{i})}) - y^{(i)}) \mathbf{x}_\mathbf{j}^{(\mathbf{i})} \quad (for \quad j = 0, 1, ..., n)$$

In the **Full-Batch GD** we find each single optimal θ over the whole dataset (single batch). When we finish one iteration, we move on to the next parameter, again over the whole dataset.

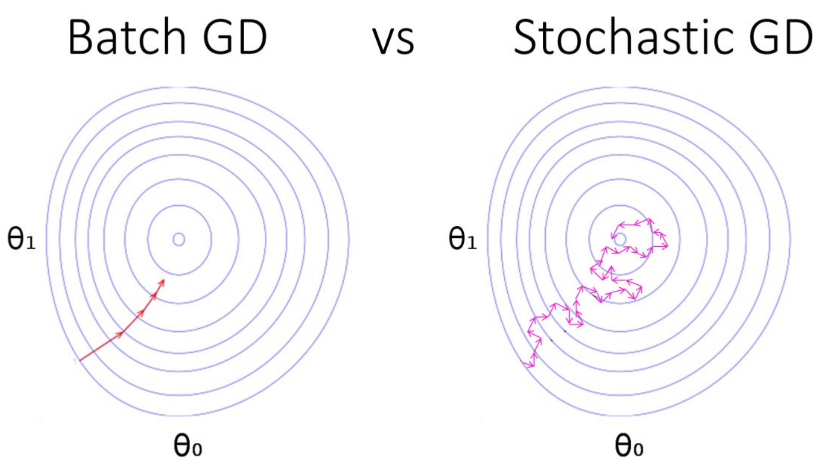
Stochastic Gradient Descent

1) Randomly initialize parameters
2) Repeat until convergence:

$$\theta_j := \theta_j - \alpha (h_\theta(\mathbf{x}^{(\mathbf{i})}) - y^{(i)}) \mathbf{x}_\mathbf{j}^{(\mathbf{i})} \quad (for \quad j = 0, 1, ..., n)$$

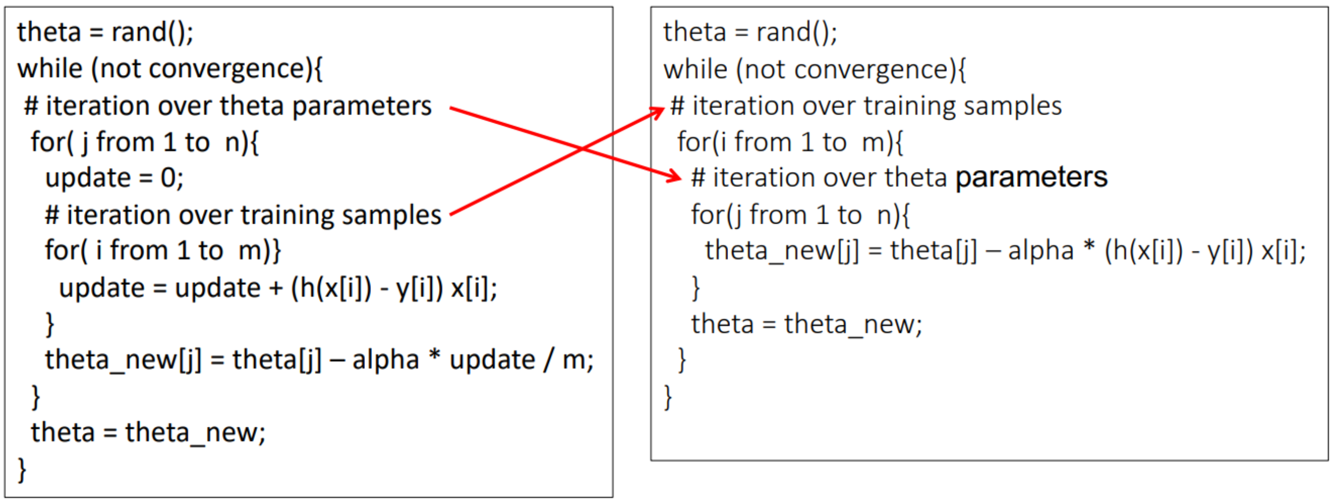
In the **Stochastic** (or **incremental**) **GD** we update all the parameters for every single training sample. Therefore, the gradient is a "stochastic approximation" of the true cost gradient.
By applying the gradient descent one sample at a time the calculations are speeded up, whereas the algorithm might swing around without never converge to the solution (*zig-zag problem*).
So, we'll accept a solution as close as possible to the optimal one and this is why it may be preferred over the Batch one in presence of large datasets.

Full-Batch GD *vs.* Stochastic GD



We can see that in the **Full-Batch GD** each jump is direct toward the minimum, but the relative update step goes through all the samples. It will converge for sure, but it can be really slow.
Instead, with the **Stochastic GD** each jump may go potentially everywhere but it is relative to just one training example. So, it is much faster, but the exact convergence is not guaranteed.

In terms of algorithms, the difference between these two methods stands in the position of the *for loops*.



```
theta = rand();
while (not convergence){
 # iteration over theta parameters
  for( j from 1 to  n){
    update = 0;
    # iteration over training samples
    for( i from 1 to  m)}
      update = update + (h(x[i]) - y[i]) x[i];
    }
    theta_new[j] = theta[j] – alpha * update / m;
  }
  theta = theta_new;
}
```

```
theta = rand();
while (not convergence){
 # iteration over training samples
  for(i from 1 to  m){
   # iteration over theta parameters
    for(j from 1 to  n){
      theta_new[j] = theta[j] – alpha * (h(x[i]) - y[i]) x[i];
    }
    theta = theta_new;
  }
}
```

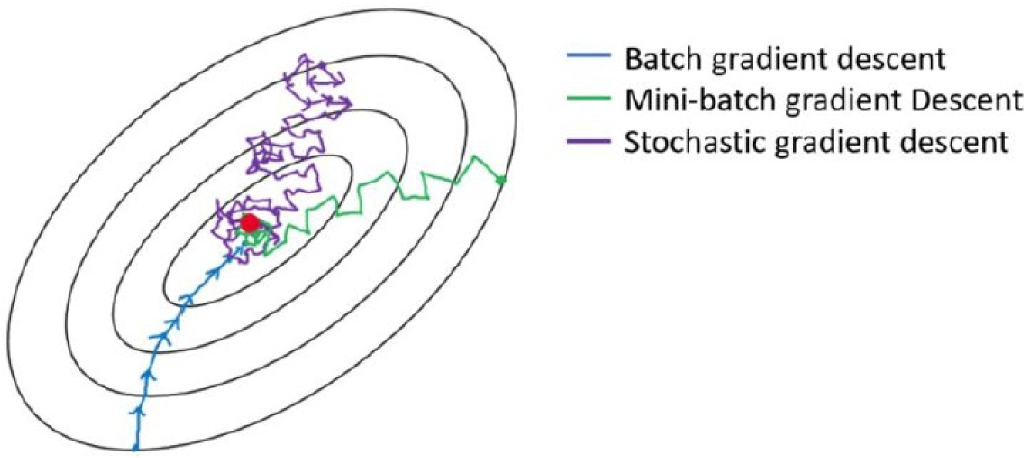
Mini-Batch Gradient Descent

The **Mini-Batch GD** represents a compromise between the Stochastic and the Batch GD algorithms, by taking the advantages of both.
For every step we take a number **b** of training samples $(1 \leq b \ll m)$ and on these we perform an update of all the parameters θs.

So, the update rule is now:

$$\theta_j := \theta_j - \alpha \frac{1}{b} \cdot \sum_{i=1}^{b} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot x_j^{(i)}$$

In this way, it reduces the variance of the parameters' updates, leads to a more robust convergence, and allows the usage of vectorization libraries rather than computing each step separately.



Adding features

Basically, when we have a training sample with $x^{(i)}$ as input, we can define our prediction function as:

$$h_\theta\left(x^{(i)}\right) = \theta^T \cdot x^{(i)} = \theta_0 + \theta_1 \cdot x_1^{(i)} + \dots + \theta_n \cdot x_n^{(i)}$$

However, there's always the possibility of adding new features (new $x^{(i)}$) which could derive or even substitute the former features.

Example

For instance, let us consider the following hypothesis and features related to a house:

$$h_\theta\left(x^{(i)}\right) = \theta_0 + \theta_1 \cdot front^{(i)} + \theta_2 \cdot side^{(i)}$$

By using the information given from $front^{(i)}$ and $side^{(i)}$, we can calculate:

$$front^{(i)} \cdot side^{(i)} = area^{(i)} \quad \Longrightarrow \quad h_\theta\left(x^{(i)}\right) = \theta_0 + \theta_1 \cdot area^{(i)}$$

In this way we have:

- possibly added useful information for our hypothesis, and
- reduced the complexity of the problem because we are now using less features for the prediction $h_\theta\left(x^{(i)}\right)$.

To sum up, given an input $x^{(i)}$, we can introduce a new feature $x_{n+(k+1)}^{(i)}$, with $k \in \mathbb{N}$, which is a transformation of already existing features such that:

$$x_{n+(k+1)}^{(i)} = f\left(x^{(i)}\right)$$

By doing this we define a new hypothesis $h'_\theta\left(x^{(i)}\right)$.

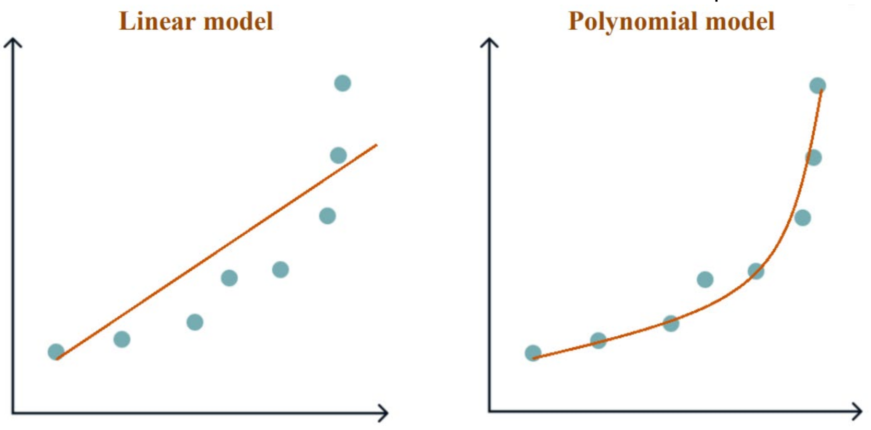
Polynomial Regression

Until now, we have used the linear regression which had only one feature and two parameters (θ_0, θ_1).

Introduce now the **Polynomial Regression** which involves the use of a polynomial of grade $\boldsymbol{t}$. The polynomial hypothesis is defined as follows:

$$h_\theta\left(x^{(i)}\right) = \theta_0 + \theta_1 \cdot x^1 + \dots + \theta_n \cdot x^n$$

Our goal is always the same as LR with one feature, but now we have more weights θs to determine. This will allow us to have a much more flexible prediction.



Feature scaling

When multiple features differ in scale (in terms of numerical orders), we can normalize the values to make sure that computations aren't affected by this difference. Consider, for instance, the difference between the number of rooms in a house and the living area:

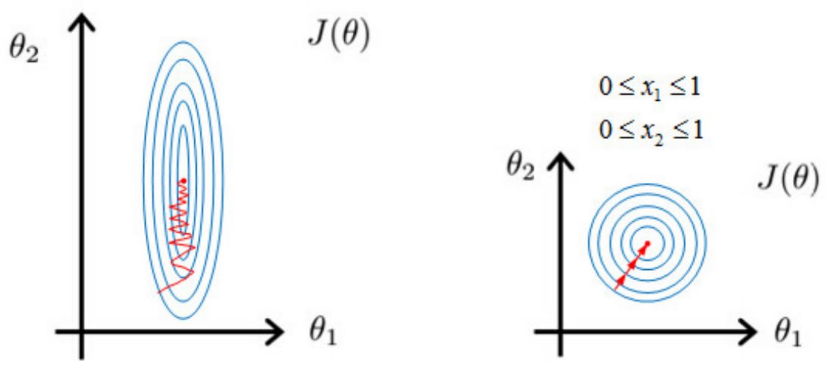
E.g. x_1= Living Area (0-2110)

x_2= number of bedrooms (1-5)

Idea:

$$x_1 = \frac{area(feet^2)}{2110}$$

$$x_2 = \frac{number of bedrooms}{5}$$



So, our $x^{(i)}$ is defined as:

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \end{bmatrix}$$

Since we know the max values for both our inputs, we can scale these two feature by dividing them for the max value:

$$x_1 = \frac{area^{(i)}}{2110} \qquad x_2 = \frac{bedrooms^{(i)}}{5}$$

In this way, all our parameters will be bounded in the specific interval $x_1, x_2 \in [0,1]$. This could improve the computing performance.

There are different techniques of **feature scaling**, such as the ***Min-max normalization*** and the ***Z-score normalization***.

Normal Equations

As previously seen, the cost function is defined as:

$$J(\theta) = \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Try to define this in a <u>vectorial way</u>.

Let's define the matrix $\mathbf{X} \in \mathbb{R}^{m \cdot n}$ of the input's vectors of all the m samples and the matrix $\mathbf{y} \in \mathbb{R}^{m}$ of the outputs of all the m samples, as:

$$\mathbf{X} = \begin{bmatrix} x^{(1)T} \\ x^{(2)T} \\ \cdots \\ x^{(i)T} \\ \cdots \\ x^{(m)T} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \cdots \\ y^{(i)} \\ \cdots \\ y^{(m)} \end{bmatrix}$$

where the matrix $\mathbf{X}$ can be also written as:

$$\mathbf{X} = \begin{bmatrix} x^{(1)T} \\ x^{(2)T} \\ \cdots \\ x^{(i)T} \\ \cdots \\ x^{(m)T} \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & \cdots & x_n^{(1)} \\ 1 & x_1^{(2)} & \cdots & x_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(m)} & \cdots & x_n^{(m)} \end{bmatrix}$$

Determine now the error in a vectorial way as the difference $\mathbf{X}\theta - \mathbf{y}$:

$$\boldsymbol{X\theta} - \boldsymbol{y} = \begin{bmatrix} 1 & \boldsymbol{x}_1^{(1)} & \cdots & \boldsymbol{x}_n^{(1)} \\ 1 & \boldsymbol{x}_1^{(2)} & \cdots & \boldsymbol{x}_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \boldsymbol{x}_1^{(m)} & \cdots & \boldsymbol{x}_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(i)} \\ \vdots \\ y^{(m)} \end{bmatrix} = \begin{bmatrix} h_\theta\left(\boldsymbol{x}_1^{(1)}\right) - y^{(1)} \\ h_\theta\left(\boldsymbol{x}_1^{(2)}\right) - y^{(2)} \\ \vdots \\ h_\theta\left(\boldsymbol{x}_1^{(m)}\right) - y^{(m)} \end{bmatrix}$$

Considering the transposed matrix of the error:

$$(\mathbf{X}\theta - \mathbf{y})^{\mathrm{T}} = \left[h_\theta\left(\mathbf{x}_1^{(1)}\right) - \mathrm{y}^{(1)} \quad h_\theta\left(\mathbf{x}_1^{(2)}\right) - \mathrm{y}^{(2)} \quad \ldots \quad h_\theta\left(\mathbf{x}_1^{(m)}\right) - \mathrm{y}^{(m)}\right]$$

hence, the product between the matrix found and its transposed is:

$$(\boldsymbol{X\theta} - \boldsymbol{y})^T \cdot (\boldsymbol{X\theta} - \boldsymbol{y}) = \left[h_\theta\left(\boldsymbol{x}_1^{(1)}\right) - y^{(1)} \quad \ldots \quad h_\theta\left(\boldsymbol{x}_1^{(m)}\right) - y^{(m)}\right] \cdot \begin{bmatrix} h_\theta\left(\boldsymbol{x}_1^{(1)}\right) - y^{(1)} \\ h_\theta\left(\boldsymbol{x}_1^{(2)}\right) - y^{(2)} \\ \vdots \\ h_\theta\left(\boldsymbol{x}_1^{(m)}\right) - y^{(m)} \end{bmatrix} =$$

$$= \left(\left(h_\theta\left(\mathbf{x}^{(1)}\right) - \mathrm{y}^{(m)}\right)^2 + \left(h_\theta\left(\mathbf{x}^{(2)}\right) - \mathrm{y}^{(m)}\right)^2 + \cdots + \left(h_\theta\left(\mathbf{x}^{(m)}\right) - \mathrm{y}^{(m)}\right)^2\right) =$$

$$= \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \left(\boldsymbol{h_\theta}\left(\mathbf{x}^{(i)}\right) - \mathbf{y}^{(i)}\right)^{\mathbf{2}}$$

So, we have proved that the cost function $J(\theta)$ can be written, in a vectorial way, as:

$$\boldsymbol{J(\theta)} = \frac{\mathbf{1}}{\mathbf{2}} \cdot (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^{\boldsymbol{T}} \cdot (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

Optimal parameters $\widehat{\theta}$

Let's proceed now to find the optimal parameters θs, written as the vector $\widehat{\boldsymbol{\theta}}$.
We can prove that, given the cost function:

$$J(\theta) = \frac{1}{2} \cdot (\mathrm{X}\theta - \mathrm{y})^T \cdot (\mathrm{X}\theta - \mathrm{y})$$

the vector of the optimal parameters θs can be defined as:

$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^{\boldsymbol{T}}\mathbf{X})^{-\mathbf{1}} \cdot \mathbf{X}^{\mathbf{T}} \cdot \mathbf{y}$$

Proof

Given the cost function $J(\theta)$:

$$J(\theta) = \frac{1}{2} \cdot (\mathbf{X}\theta - \mathbf{y})^T \cdot (\mathbf{X}\theta - \mathbf{y})$$

we want to find the minimum of $J(\theta)$ by calculating the point in which the gradient has null value:

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} = \mathbf{0}$$

The partial derivative of $J(\theta)$ is calculated as:

$$\frac{\partial J(\theta)}{\partial \theta_k} = \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_k^{(i)} = \left[\left(h_\theta(\mathbf{x}^{(1)}) - y^{(1)}\right) \ \dots \ \left(h_\theta(\mathbf{x}^{(m)}) - y^{(m)}\right)\right] \cdot \begin{bmatrix} 1 \\ x_1^{(1)} \\ \vdots \\ x_n^{(m)} \end{bmatrix}$$

Hence, we can re-write $\nabla J(\theta)$ as the matrix:

$$\nabla J(\theta) = \begin{bmatrix} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \\ \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_1^{(i)} \\ \vdots \\ \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x_n^{(i)} \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \boldsymbol{x}_1^{(1)} & \boldsymbol{x}_1^{(2)} & \cdots & \boldsymbol{x}_1^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x}_n^{(1)} & \boldsymbol{x}_n^{(2)} & \cdots & \boldsymbol{x}_n^{(m)} \end{bmatrix} \cdot \begin{bmatrix} h_\theta\left(\boldsymbol{x}_1^{(1)}\right) - y^{(1)} \\ h_\theta\left(\boldsymbol{x}_1^{(2)}\right) - y^{(2)} \\ \vdots \\ h_\theta\left(\boldsymbol{x}_1^{(m)}\right) - y^{(m)} \end{bmatrix} =$$

$$= \mathbf{X}^{\mathrm{T}} \cdot (\mathbf{X}\theta - \mathbf{y}) =$$

$$= \mathbf{X}^{\mathrm{T}}\mathbf{X}\theta - \mathbf{X}^{\mathrm{T}}\mathbf{y}$$

Then:

$$\nabla J(\theta) = 0 \quad \Longrightarrow \quad \mathbf{X}^{\mathrm{T}}\mathbf{X}\theta - \mathbf{X}^{\mathrm{T}}\mathbf{y} = 0$$

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}\theta = \mathbf{X}^{\mathrm{T}}\mathbf{y}$$

$$\widehat{\boldsymbol{\theta}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} \cdot \mathbf{X}^{\mathrm{T}}\mathbf{y} \quad \longleftarrow$$

Vector of optimal parameters θs. It is also called $\boldsymbol{\theta_{min}}$

Probabilistic Interpretation of Linear Regression

As previously said, we are approximating the output $y^{(i)}$ with our hypothesis $h_\theta\left(x^{(i)}\right)$, which is a function of the inputs using a set of parameters θ.
Let's consider now the $\boldsymbol{m}$ instances $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$ of the training set and the hypothesis:

$$h_\theta\left(\mathbf{x}^{(i)}\right) = \theta^T \cdot \mathbf{x}^{(i)}$$

that approximates the relation between features' vectors and output. The *output* can be computed as a predicted value added to the error:

$$y^{(i)} = \theta^T \mathbf{x}^{(\boldsymbol{i})} + e^{(i)}$$

where $e^{(i)}$ is the errors' vector between the actual and the predicted values.

By assuming that each sample is independent, we can model the error as an exponential random variable and it is assumed that the variables show all a normal distribution, with mean 0 and variance σ^2:

$$p\left(e^{(i)}\right) = N(0, \sigma^2) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot e^{-\frac{\left(e^{(i)}\right)^2}{2\sigma^2}} \qquad i = 1 \dots m$$

Expressing the error as:

$$e^{(i)} = y^{(i)} - \theta^T \mathbf{x}^{(\boldsymbol{i})}$$

we can say that the probability $p\left(e^{(i)}\right)$ is the same as considering how likely we will obtain an output $y^{(i)}$ given the input $x^{(i)}$ and parametrized by θ. Substituting the error definition in $p\left(e^{(i)}\right)$, we get:

$$p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \theta\right) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot e^{-\frac{\left(y^{(i)} - \theta^T \mathbf{x}^{(i)}\right)^2}{2\sigma^2}} \qquad i = 1 \dots m$$

The closer the prediction $h_\theta\left(\mathbf{x}^{(i)}\right)$ gets to the actual value $y^{(i)}$, the bigger the value of this probability gets with maximum value when $y^{(i)} = \theta^T \cdot x^{(i)}$.

Likelihood function (LHF)

Previously we have defined the form of the probability for a generic sample as:

$$p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \theta\right) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot e^{-\frac{\left(y^{(i)} - \theta^T \mathbf{x}^{(i)}\right)^2}{2\sigma^2}} \qquad i = 1 \dots m$$

Now we want to find a function of the parameters that, given the input, returns the output for each training case. This probability is named as the ***Likelihood function***, and it is defined as the joint probability of all the samples:

$$L(\theta) = L(\theta; \mathbf{X}; \mathbf{y}) = p(\mathbf{y} \mid \mathbf{X}; \theta)$$

where $\mathbf{X}$ is the matrix of all the $\mathbf{x}^{(i)}$ $inputs$ and $\mathbf{y}$ is the vector of all the $y^{(i)}$ $outputs$.

Since the independence assumption between the samples, the *likelihood function* can be defined as:

$$L(\theta) = p(\mathbf{y} \mid \mathbf{X}; \theta) = \prod_{i=1}^{m} p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \theta\right) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot e^{-\frac{\left(y^{(i)} - \theta^T \mathbf{x}^{(i)}\right)^2}{2\sigma^2}}$$

Now, we need to find the vector $\widehat{\boldsymbol{\theta}}$ of the parameters that maximize the $L(\theta)$. So, we'll look for the θs that have the maximum probability to return the output given the input. This is named **maximum likelihood criterion** and consists in:

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\boldsymbol{argmax}}\left(\boldsymbol{L}(\boldsymbol{\theta})\right) \quad \Rightarrow \quad \widehat{\boldsymbol{\theta}} = \underset{\theta}{argmax} \ln\left(L(\theta)\right) = \underset{\boldsymbol{\theta}}{\boldsymbol{argmax}}\ \boldsymbol{l}(\boldsymbol{\theta})$$

Why we use the logarithmic function?

When we work with products between values smaller than zero, it's much easier to work with the logarithm of the product. In fact, thanks to the properties of logarithms we can simplify our problem by transforming the production in a summatory.

We have defined the **log-likelihood**, as $\mathbf{ln}\left(\boldsymbol{L}(\boldsymbol{\theta})\right) = \boldsymbol{l}(\boldsymbol{\theta})$, such that:

$$l(\theta) = \ln\left(L(\theta)\right) = \ln\left(\prod_{i=1}^{m} p\left(y^{(i)} \mid \boldsymbol{x}^{(i)}; \theta\right)\right) = \ln\left(\prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\cdot\sigma} \cdot e^{-\frac{\left(y^{(i)}-\theta^T\mathbf{x}^{(i)}\right)^2}{2\sigma^2}}\right) =$$

Using the property $\mathbf{log}(\boldsymbol{AB}) = \mathbf{log}(\boldsymbol{A}) + \mathbf{log}(\boldsymbol{B})$:

$$= \sum_{i=1}^{m} \ln\left(\frac{1}{\sqrt{2\pi}\cdot\sigma} \cdot e^{-\frac{\left(y^{(i)}-\theta^T\mathbf{x}^{(i)}\right)^2}{2\sigma^2}}\right) =$$

$$= \sum_{i=1}^{m}\left(\ln\left(\frac{1}{\sqrt{2\pi}\cdot\sigma}\right) + \ln\left(e^{-\frac{\left(y^{(i)}-\theta^T\mathbf{x}^{(i)}\right)^2}{2\sigma^2}}\right)\right) =$$

$$= m\cdot\ln\left(\frac{1}{\sqrt{2\pi}\cdot\sigma}\right) - \sum_{i=1}^{m}\frac{\left(y^{(i)}-\theta^T\mathbf{x}^{(i)}\right)^2}{2\sigma^2} =$$

$$= m\cdot\ln\left(\frac{1}{\sqrt{2\pi}\cdot\sigma}\right) - \frac{1}{2\sigma^2}\cdot\sum_{i=1}^{m}\left(y^{(i)}-\theta^T\mathbf{x}^{(i)}\right)^2$$

Then, the goal is to maximize the function:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, l(\theta) = \underset{\theta}{\operatorname{argmax}}\left(m\cdot\ln\left(\frac{1}{\sqrt{2\pi}\cdot\sigma}\right) - \frac{1}{2\sigma^2}\cdot\sum_{i=1}^{m}\left(y^{(i)}-\theta^T\mathbf{x}^{(i)}\right)^2\right) =$$

Since the first term is a constant, it doesn't matter in the maximization problem; same consideration for the term $1/_{\sigma^2}$. Hence:

$$= \underset{\theta}{\operatorname{argmax}}\left(-\frac{1}{2}\cdot\sum_{i=1}^{m}\left(y^{(i)}-\theta^T\mathbf{x}^{(i)}\right)^2\right) =$$

Finding the maximum of a negative function (the negative sign in the argument) is the same as changing the sign of the function and then finding the minimum of it. Moreover, since it's all elevated to the power of 2, changing the sign doesn't affect the result.

$$= \underset{\theta}{\operatorname{argmax}}\left(-\frac{1}{2}\cdot\sum_{i=1}^{m}\left(y^{(i)}-\theta^T\mathbf{x}^{(i)}\right)^2\right) = \underset{\theta}{\operatorname{argmin}}\left(\frac{1}{2}\cdot\sum_{i=1}^{m}\left(y^{(i)}-\theta^T\mathbf{x}^{(i)}\right)^2\right)$$

Solved the problem in terms of minimization, we can safely say now that minimizing the probability of getting the right output with respect to a certain θ is the same as finding the θ that minimizes the cost function:

$$\underset{\boldsymbol{\theta}}{\mathbf{argmax}}\, \boldsymbol{l}(\boldsymbol{\theta}) = \underset{\theta}{\operatorname{argmin}} \left(\frac{1}{2} \cdot \sum_{i=1}^{m} \left(y^{(i)} - \theta^T \mathbf{x}^{(i)} \right)^2 \right) = \underset{\boldsymbol{\theta}}{\mathbf{argmin}}\, \boldsymbol{J}(\boldsymbol{\theta})$$

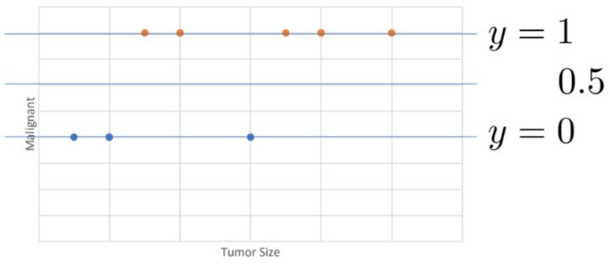
Chapter 3 – Classification problem

The **classification problem** is a particular problem which will give us a class as output, instead of a real number approximation. For instance, let's consider the classification example of tumours:

Tumor: Malignant/Benign?

So, given the input $x^{(i)}$ we'd like to predict if a tumour is benign or malignant as an output $y^{(i)}$. So, we can define:

$$y^{(i)} \in \begin{cases} 0 : \textit{negative class} \rightarrow \textbf{benign} \\ 1 : \textit{positive class} \rightarrow \textbf{malignant} \end{cases}$$



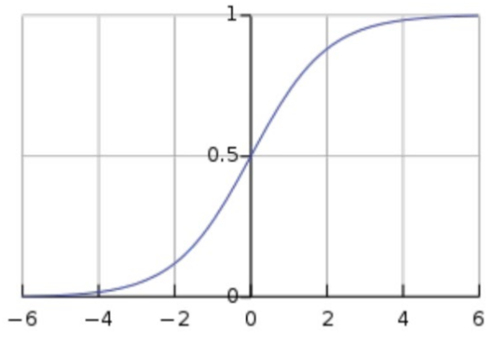
As we can see, we have drawn a line for the value 0.5 which will be our **threshold classifier** such that, given the hypothesis $0 \leq h_\theta(\mathbf{x}) \leq 1$, we'll consider:

- if $h_\theta(\mathbf{x}) \geq 0.5 \rightarrow \text{predict} \quad y = 1$
- if $h_\theta(\mathbf{x}) < 0.5 \rightarrow \text{predict} \quad y = 0$

Logistic Regression model

Thus, in classification problems we need to divide the output domain in parts based on the classes' characteristics. To normalize such set, we transform the hypothesis function already seen $h_\theta(\mathbf{x}) = \theta^T\mathbf{x}$ using the **Sigmoid function**, defined as:

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



So, the hypothesis becomes:

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

Our function, which is sigmoid-mapped, has minimum in 0 for $-\infty$ and the maximum in 1 for $+\infty$, just like the sigmoid itself is bounded from 0 to 1. So, every input will produce a bounded output $y^{(i)} \in [0, 1]$.

Probabilistic Interpretation and the Cost function

Our prediction $h_\theta(x)$ is basically the probability of that input being of a specific class. Since we are considering only 2 classes $(y = \{0, 1\})$, we can express the hypothesis as the probability:

$$\boldsymbol{h_\theta(x) = p(y = 1 \mid x; \theta)}$$

which is the estimated probability that y is equals to 1, given input x and parametrized by θ.

So, it is possible to assume that:

$$p(y = 0 \mid x; \theta) + p(y = 1 \mid x; \theta) = 1 \quad \Longrightarrow \quad p(y = 0 \mid x; \theta) = 1 - p(y = 1 \mid x; \theta)$$

Therefore, our goal is to maximize the probability of the output being 1 given a specific input and parameter. As previously seen, this could be done by using the **likelihood function**. Under the same assumptions of being independent and identically distributed (i.i.d.), we have:

$$L(\theta) = p(\mathbf{y} \mid \mathbf{X}; \theta) = \prod_{i=1}^{m} p\left(y^{(i)} \mid x^{(i)}; \theta\right)$$

Thinking of the hypothesis as the probability of $x^{(i)}$ belonging to the *positive class* $(y^{(i)} = 1)$, we can write that:

$$h_\theta(x) = p\left(y^{(i)} = 1 \mid x^{(i)}; \theta\right) \quad \Longrightarrow \quad p\left(y^{(i)} = 0 \mid x^{(i)}; \theta\right) = 1 - h_\theta\left(x^{(i)}\right)$$

By putting everything together, we get:

$$\boldsymbol{p\left(y^{(i)} \mid x^{(i)}; \theta\right) = h_\theta\left(x^{(i)}\right)^{y^{(i)}} \cdot \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}}}$$

Since $y^{(i)}$ can be only equal to 1 or 0, depending on its value we get that

$$\begin{cases} y^{(i)} = 0 & \Longrightarrow \quad p(y^{(i)} = 0 \mid x^{(i)}; \theta) = \boldsymbol{1 - h_\theta(x^{(i)})} \\ y^{(i)} = 1 & \Longrightarrow \quad p(y^{(i)} = 1 \mid x^{(i)}; \theta) = \boldsymbol{h_\theta(x^{(i)})} \end{cases}$$

Hence, it is like a Bernoulli's distribution.

Then we can substitute this result in the *likelihood function*, obtaining that:

$$L(\theta) = p(\mathbf{y} \mid \mathbf{X}; \theta) = \prod_{i=1}^{m} \left[h_\theta\left(x^{(i)}\right)^{y^{(i)}} \cdot \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}} \right]$$

As we did before for the LiRe, we can use the logarithm of $L(\theta)$ to simplify the problem:

$$\begin{aligned} \boldsymbol{l(\theta)} = \ln\left(L(\theta)\right) &= \ln\left(\prod_{i=1}^{m} \left[h_\theta\left(x^{(i)}\right)^{y^{(i)}} \cdot \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}} \right] \right) = \\ &= \sum_{i=1}^{m} \left[\ln\left(h_\theta\left(x^{(i)}\right)^{y^{(i)}} \cdot \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}} \right) \right] = \\ &= \sum_{i=1}^{m} \left[\ln\left(h_\theta\left(x^{(i)}\right)^{y^{(i)}} \right) + \ln\left(\left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}} \right) \right] = \\ &= \boldsymbol{\sum_{i=1}^{m} \left[y^{(i)} \cdot \ln\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \cdot \ln\left(1 - h_\theta\left(x^{(i)}\right)\right) \right]} \end{aligned}$$

Now, we can shift from a *maximization problem* to a *minimization one* and so:

$$\begin{aligned} \boldsymbol{J(\theta)} &= \boldsymbol{-\frac{1}{m} \cdot \sum_{i=1}^{m} \left[y^{(i)} \cdot ln\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \cdot ln\left(1 - h_\theta\left(x^{(i)}\right)\right) \right]} = \\ &= \boldsymbol{-\frac{1}{m} \cdot l(\theta)} \qquad \textbf{Cross} - \textbf{Entropy Error Function} \end{aligned}$$

Finally, we can set the whole problem, as previously done, as a parameter fitting problem. So, we'll have to find the parameters θs such that:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\boldsymbol{argmin}}\ \boldsymbol{J}(\boldsymbol{\theta})$$

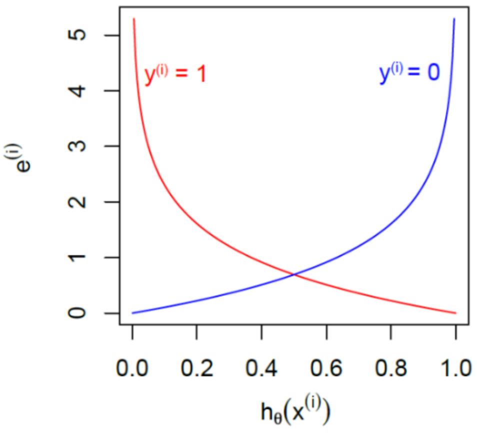
Errors

Since the logistic regression function cannot be easily studied analytically, we can get an intuition using the error contribution of each sample which is:

$$e^{(i)} = -y^{(i)} \cdot \ln\left(h_\theta\left(\mathbf{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \cdot \ln\left(1 - h_\theta\left(\mathbf{x}^{(i)}\right)\right)$$

It's easy to see that:

- if $\boldsymbol{y^{(i)} = 1} \quad \Rightarrow \quad e^{(i)} = -\log\left(h_\theta\left(\mathbf{x}^{(i)}\right)\right)$
  - $\boldsymbol{h_\theta\left(\mathbf{x}^{(i)}\right) = 0}$ then $\boldsymbol{e^{(i)} \to \infty}$
  - $\boldsymbol{h_\theta\left(\mathbf{x}^{(i)}\right) = 1}$ then $\boldsymbol{e^{(i)} \to 0}$
- if $\boldsymbol{y^{(i)} = 0} \quad \Rightarrow \quad e^{(i)} = -\log\left(1 - h_\theta\left(\mathbf{x}^{(i)}\right)\right)$
  - $\boldsymbol{h_\theta\left(\mathbf{x}^{(i)}\right) = 0}$ then $\boldsymbol{e^{(i)} \to 0}$
  - $\boldsymbol{h_\theta\left(\mathbf{x}^{(i)}\right) = 1}$ then $\boldsymbol{e^{(i)} \to \infty}$



Gradient Descent in Logistic Regression

Just like in linear regression, we use the **gradient descent algorithm** to find our weights $\hat{\boldsymbol{\theta}}$, such that $\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\boldsymbol{argmin}}\ \boldsymbol{J}(\boldsymbol{\theta})$.

We update the weights using the relation:

$$\theta_k \coloneqq \theta_k - \alpha \cdot \frac{\partial\, J(\theta)}{\partial\, \theta_k}$$

where the partial derivative is defined as:

$$\frac{\partial\, J(\theta)}{\partial\, \theta_k} = \frac{\partial}{\partial\, \theta_k}\left(-\frac{1}{m} \cdot \sum_{i=1}^{m}\left[y^{(i)} \cdot \left(\ln\left(h_\theta\left(x^{(i)}\right)\right)\right) + \left(1 - y^{(i)}\right) \cdot \left(\ln\left(1 - h_\theta\left(x^{(i)}\right)\right)\right)\right]\right) =$$

$$= -\frac{1}{m} \cdot \sum_{i=1}^{m}\left[y^{(i)} \cdot \frac{\partial}{\partial\, \theta_k}\left(\ln\left(h_\theta\left(x^{(i)}\right)\right)\right) + \left(1 - y^{(i)}\right) \cdot \frac{\partial}{\partial\, \theta_k}\left(\ln\left(1 - h_\theta\left(x^{(i)}\right)\right)\right)\right]$$

Let's compute the two separate derivatives:

$$\frac{\partial}{\partial\, \theta_k}\ln\left(h_\theta\left(x^{(i)}\right)\right) = \frac{1}{h_\theta\left(x^{(i)}\right)} \cdot \frac{\partial\, h_\theta\left(x^{(i)}\right)}{\partial\, \theta_k}$$

$$\frac{\partial}{\partial\, \theta_k}\ln\left(1 - h_\theta\left(x^{(i)}\right)\right) = -\frac{1}{1 - h_\theta\left(x^{(i)}\right)} \cdot \frac{\partial\, h_\theta\left(x^{(i)}\right)}{\partial\, \theta_k}$$

Then, determine the derivative of the hypothesis:

$$\frac{\partial\, h_\theta\left(x^{(i)}\right)}{\partial\, \theta_k} = \frac{\partial}{\partial\, \theta_k}\left(\frac{1}{1 + e^{-\theta^T \cdot x^{(i)}}}\right) = -\frac{e^{-\theta^T \cdot x^{(i)}}}{\left(1 + e^{-\theta^T \cdot x^{(i)}}\right)^2} \cdot \left(-x_k^{(i)}\right) =$$

$$= \frac{1}{1+e^{-\theta^T\cdot x^{(i)}}} \cdot \frac{e^{-\theta^T\cdot x^{(i)}}}{1+e^{-\theta^T\cdot x^{(i)}}} \cdot x_k^{(i)} =$$

$$= h_\theta\left(x^{(i)}\right) \cdot \frac{e^{-\theta^T\cdot x^{(i)}}}{1+e^{-\theta^T\cdot x^{(i)}}} \cdot x_k^{(i)} =$$

By summing and subtracting the element $h_\theta\left(x^{(i)}\right)$ we can get:

$$= h_\theta\left(x^{(i)}\right) \cdot \left[\frac{e^{-\theta^T\cdot x^{(i)}}}{1+e^{-\theta^T\cdot x^{(i)}}} + \boldsymbol{h_\theta\left(x^{(i)}\right)} - \boldsymbol{h_\theta\left(x^{(i)}\right)}\right] \cdot x_k^{(i)} =$$

$$= h_\theta\left(x^{(i)}\right) \cdot \left[\frac{e^{-\theta^T\cdot x^{(i)}}}{1+e^{-\theta^T\cdot x^{(i)}}} + \frac{1}{1+e^{-\theta^T\cdot x^{(i)}}} - h_\theta\left(x^{(i)}\right)\right] \cdot x_k^{(i)} =$$

$$= h_\theta\left(x^{(i)}\right) \cdot \left[\frac{\cancel{e^{-\theta^T\cdot x^{(i)}}+1}}{\cancel{1+e^{-\theta^T\cdot x^{(i)}}}} - h_\theta\left(x^{(i)}\right)\right] \cdot x_k^{(i)} =$$

$$= \boldsymbol{h_\theta\left(x^{(i)}\right) \cdot \left(1 - h_\theta\left(x^{(i)}\right)\right) \cdot x_k^{(i)}}$$

Now, we can use this result in the two previous terms:

$$\frac{\partial}{\partial\,\theta_k}\left(\ln\left(h_\theta\left(x^{(i)}\right)\right)\right) = \frac{1}{\cancel{h_\theta\left(x^{(i)}\right)}} \cdot \cancel{h_\theta\left(x^{(i)}\right)} \cdot \left(1 - h_\theta\left(x^{(i)}\right)\right) \cdot x_k^{(i)} =$$

$$= \boldsymbol{\left(1 - h_\theta\left(x^{(i)}\right)\right) \cdot x_k^{(i)}}$$

$$\frac{\partial}{\partial\,\theta_k}\left(\ln\left(1 - h_\theta\left(x^{(i)}\right)\right)\right) = -\frac{1}{\cancel{1 - h_\theta\left(x^{(i)}\right)}} \cdot h_\theta\left(x^{(i)}\right) \cdot \left(\cancel{1 - h_\theta\left(x^{(i)}\right)}\right) \cdot x_k^{(i)} =$$

$$= \boldsymbol{-h_\theta\left(x^{(i)}\right) \cdot x_k^{(i)}}$$

Hence:

$$\frac{\boldsymbol{\partial\,J(\theta)}}{\boldsymbol{\partial\,\theta_k}} = -\frac{1}{m} \cdot \sum_{i=1}^{m}\left[y^{(i)} \cdot \left(1 - h_\theta\left(x^{(i)}\right)\right) \cdot x_k^{(i)} + \left(1 - y^{(i)}\right) \cdot -h_\theta\left(x^{(i)}\right) \cdot x_k^{(i)}\right] =$$

$$= -\frac{1}{m} \cdot \sum_{i=1}^{m}\left[y^{(i)} - \cancel{y^{(i)} \cdot h_\theta\left(x^{(i)}\right)} - h_\theta\left(x^{(i)}\right) + \cancel{y^{(i)} \cdot h_\theta\left(x^{(i)}\right)}\right] \cdot x_k^{(i)} =$$

$$= -\frac{1}{m} \cdot \sum_{i=1}^{m}\left[y^{(i)} - h_\theta\left(x^{(i)}\right)\right] \cdot x_k^{(i)}$$

$$= \boldsymbol{\frac{1}{m} \cdot \sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) \cdot x_k^{(i)}}$$

So, we can write the result obtained in a vectorial form as:

$$\boldsymbol{\nabla J = \frac{1}{m} \cdot X^T \cdot \left(h_\theta(X) - y\right)}$$

Multi-class classification

In this study we have always considered having only 2 classes (positive/negative). However, it is obviously possible to have more than two classes and we can use the **one *vs.* all** approach.

Basically, every multi-class can be reconducted to a binary classification to determine whether the data belongs to that class or any other class.

Let us consider, for instance, the following cases:

Email tagging: Work, Friends, Family, Hobby

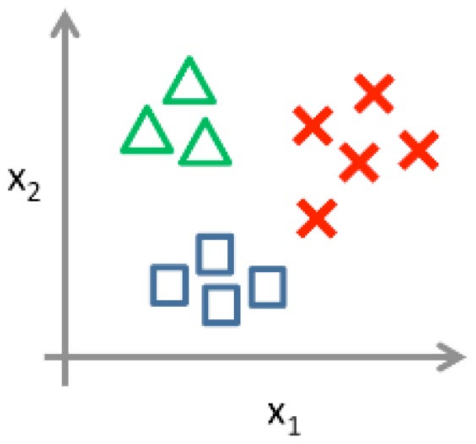
y= [1 , 2 , 3 , 4]

Medical diagrams: Not ill, Cold, Flu

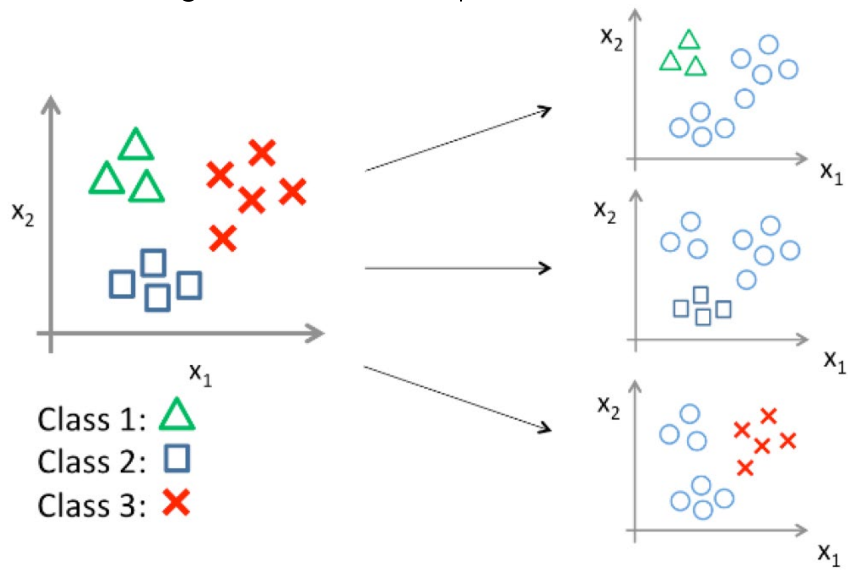
y= [1 , 2 , 3]

Weather: Sunny, Cloudy, Rain, Snow

y= [1 , 2 , 3 , 4]

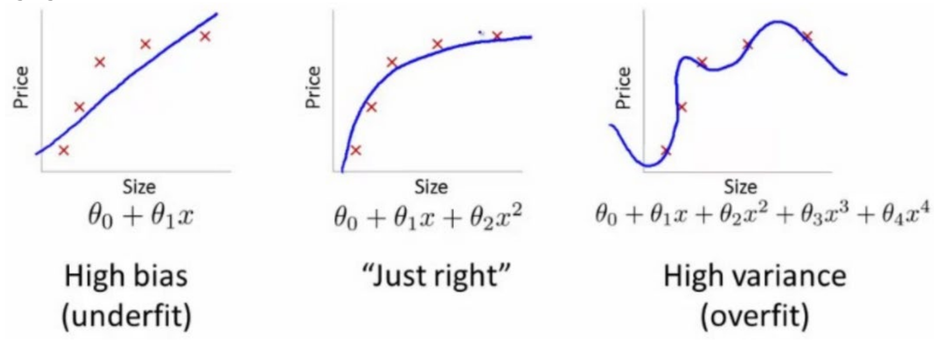


Considering now the technique **one *vs.* all**, we'll have:



Chapter 4 – Fitting (26/10/2023)

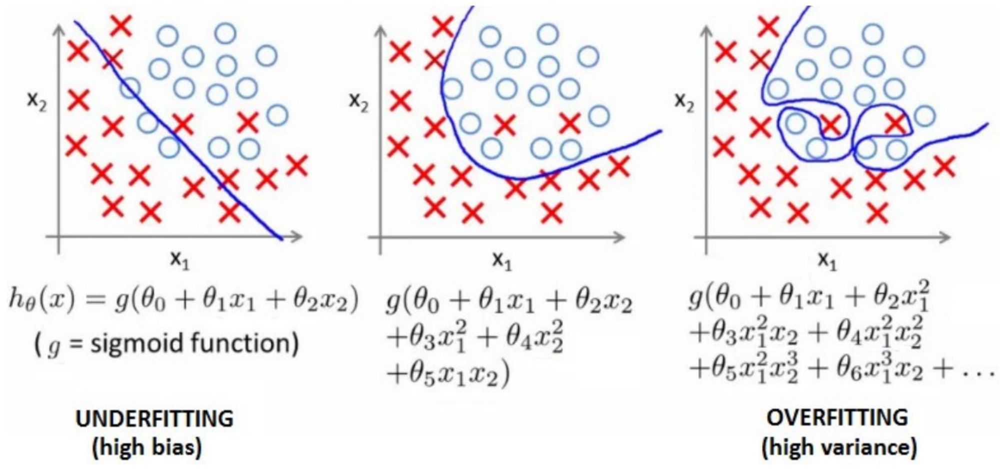
In the following graphs we can see 3 different cases:



These graphs were created with a polynomial regression, but they show different behaviours depending on the grade:

- The first graph shows a situation called **underfit**, where the error is too high (*high bias*).
- The third graph shows the situation called **overfit**, where the error is exactly or close to zero (*high variance*). However, this means that the model did not learn how to generalize, but it "describes" the training points.
- The second graph is the best one because it describes generally the behaviour of the samples without being too dependent on them. The error is not too high, and the model is correctly generalized.

These situations are valid also for a classification problem:



Underfitting and Overfitting

When we build a model, the main goal is making it able to generalize the experience taken from the training data set and then predict future situations. One thing that strongly influences the model's performance is its complexity which depends on the number of features used.

We define a model as **underfitting** when it does not fit the training data because it is too simple and then it's not able to learn. This means that the number of features chosen are not enough, so it will perform bad both in training and test sets.

Instead, we define a model as **overfitting** when it performs well on training data but not on test (unseen, new) data. It is so complex that it memorizes training data rather than *learning* from those. So, it is not able to generalize the experience gained. In general, this means that we have chosen too many features.

Bias and Variance

Let $\boldsymbol{X}$ be the input and $\boldsymbol{Y}$ be the output in a problem we want to model:

$$\boldsymbol{Y = f(X) + e}$$

where $\boldsymbol{f}$ represents the unknown general relationship between X and Y.
Assuming that the hypothesis $\boldsymbol{h(X)}$ approximates the $f(X)$, the expected squared error, named ***Mean Squared Error*** (**MSE**), is defined as:

$$\boldsymbol{MSE\big(f(X)\big) = E\left[\big(Y - h(X)\big)^2\right]}$$

Suppose we have an infinite number of datasets $\boldsymbol{D_i}$, each of size m, sampled from the same data distribution D. Each sample of the dataset D_i is denoted as $< \boldsymbol{x^{(i)}, y^{(i)}} >$, where:

$$y^{(i)} = f\left(x^{(i)}\right) + e^{(i)} \quad \Rightarrow \quad D_i = \left\{< x^{(i)}, y^{(i)} >\right\}$$

where $e^{(i)} \approx N(0, \sigma_e)$ is our gaussian error.

Hence, we train a model each time with a different dataset D_i, obtaining a set of hypotheses $h^{(D_i)}(x)$, each one affected by an error between the predicted values and the actual values.
This error can be defined in the form of the **MSE**:

$$\boldsymbol{MSE\left(h^{(D_i)}(x)\right) = E_x\left[\left(h^{(D_i)}(x) - f(x)\right)^2\right]}$$

Now, we want to compute the expectation of the MSE with respect to all the possible D_i and this means computing the MSE of each hypothesis and then compute the overall mean.
This value is called **Generalization Error** (**GER**) and it's defined as:

$$\boldsymbol{GER} = E_D\left[MSE\left(h^{(D_i)}(x)\right)\right] =$$
$$= E_D\left[E_x\left[\left(h^{(D_i)}(x) - f(x)\right)^2\right]\right] =$$

Knowing that the expectation $E[x]$ is a linear operator, we can swap E_x and E_D:

$$= E_x\left[E_D\left[\left(h^{(D)}\left(x^{(i)}\right) - f\left(x^{(i)}\right)\right)^2\right]\right]$$

Now, we can define the **best estimation** of $f\left(x^{(i)}\right)$ by feeding each hypothesis with the same training sample $x^{(i)}$ and then compute the overall mean:

$$\boldsymbol{\bar{h}\left(x^{(i)}\right) = E_D\left[h^{(D)}\left(x^{(i)}\right)\right]}$$

Focus now on the E_D term of the GER, by adding and subtracting (*sum zero operation*) the best estimation inside of it:

$$
\begin{aligned}
&E_D\left[\left(h^{(D)}(x^{(i)}) - f(x^{(i)})\right)^2\right] = \\
&\quad = E_D\left[\left(h^{(D)}(x^{(i)}) - \bar{\boldsymbol{h}}(\boldsymbol{x}^{(i)}) + \bar{\boldsymbol{h}}(\boldsymbol{x}^{(i)}) - f(x^{(i)})\right)^2\right] = \\
&\quad = E_D\left[\left(h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)})\right)^2\right] + E_D\left[\left(\bar{h}(x^{(i)}) - f(x^{(i)})\right)^2\right] \\
&\qquad + E_D\left[2 \cdot \left(h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)})\right) \cdot \left(\bar{h}(x^{(i)}) - f(x^{(i)})\right)\right]
\end{aligned}
$$

By considering each term of this result, we identify:

- $E_D\left[\left(h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)})\right)^2\right] = \boldsymbol{Var}\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}^{(i)})\right)$
- $E_D\left[\left(\bar{h}(x^{(i)}) - f(x^{(i)})\right)^2\right] = \left(\bar{h}(x^{(i)}) - f(x^{(i)})\right)^2 = \boldsymbol{Bias}^2\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}^{(i)})\right)$

Finally, for the third term we have:

$$
\begin{aligned}
&E_D\left[2 \cdot \left(h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)})\right) \cdot \left(\bar{h}(x^{(i)}) - f(x^{(i)})\right)\right] = \\
&\quad = 2 \cdot \left(\bar{h}(x^{(i)}) - f(x^{(i)})\right) \cdot E_D\left[\left(h^{(D)}(x^{(i)}) - \bar{h}(x^{(i)})\right)\right] = \\
&\quad = 2 \cdot \left(\bar{h}(x^{(i)}) - f(x^{(i)})\right) \cdot \left(E_D\left[\left(h^{(D)}(x^{(i)})\right)\right] - E_D[\bar{h}(x^{(i)})]\right) = \\
&\quad = 2 \cdot \left(\bar{h}(x^{(i)}) - f(x^{(i)})\right) \cdot \left(\cancel{\bar{h}(x^{(i)})} - \cancel{\bar{h}(x^{(i)})}\right) = \\
&\quad = \boldsymbol{0}
\end{aligned}
$$

In the end, we have obtained that:

$$
\boldsymbol{MSE}\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}^{(i)})\right) = \boldsymbol{Var}\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}^{(i)})\right) + \boldsymbol{Bias}^2\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x}^{(i)})\right) \quad \Longrightarrow
$$

$$
\boldsymbol{GER} = \boldsymbol{E_x}\left[\boldsymbol{MSE}\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x})\right)\right] = \boldsymbol{E_x}\left[\boldsymbol{Var}\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x})\right) + \boldsymbol{Bias}^2\left(\boldsymbol{h}^{(\boldsymbol{D})}(\boldsymbol{x})\right)\right]
$$

Bias: the systematic deviation of the estimator, and it is defined as the difference between the expected predicted values and the actual values:

$$
\boldsymbol{Bias}\left(\boldsymbol{h}(\boldsymbol{X}), \boldsymbol{f}(\boldsymbol{X})\right) = \boldsymbol{E}[\boldsymbol{h}(\boldsymbol{X})] - \boldsymbol{f}(\boldsymbol{X})
$$

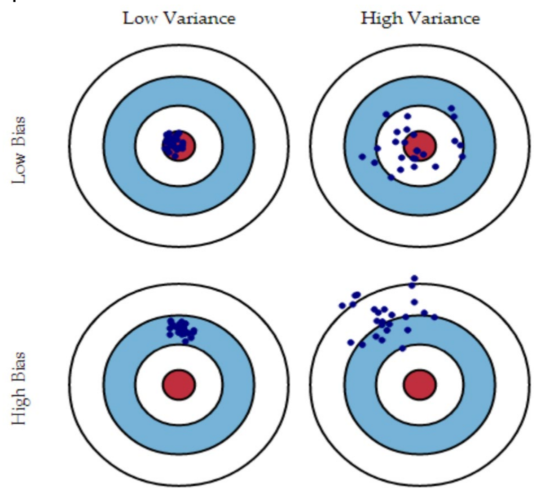
Variance: the mean of the variations of the predicted values with respect to the mean of the predicted values. It gives an idea about the stability of the model in response to new training samples.

$$
\boldsymbol{Var}\left(\boldsymbol{h}(\boldsymbol{X}), \boldsymbol{f}(\boldsymbol{X})\right) = \boldsymbol{E}[(\boldsymbol{h}(\boldsymbol{X}) - \boldsymbol{E}[\boldsymbol{h}(\boldsymbol{X})])^2]
$$

Bias/Variance trade-off

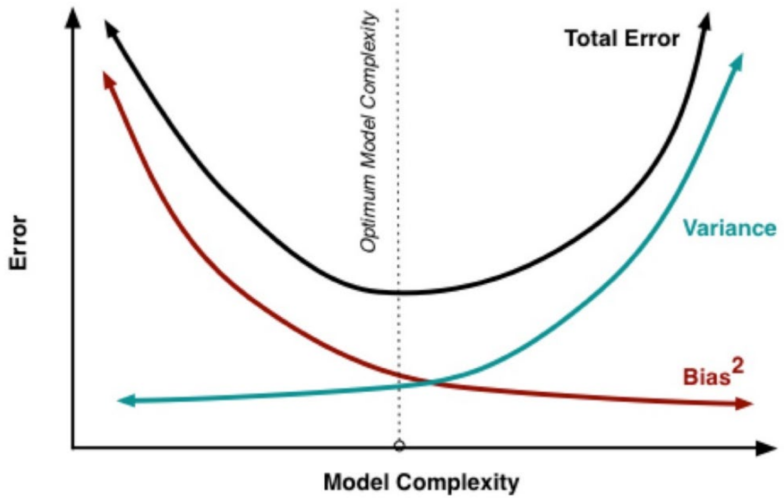
Since we want to **minimize the GER**, this means finding the minimum values for both *Bias* and *Variance* to have an error close to 0. However, these two parameters "fight" each other because they are two *symmetrical curves*. So, finding the *lowest Bias* means having a *high Variance*, and vice versa.

We can see the relationship between *Bias* and *Variance* in the following graph:



So, to reduce the GER we should consider a **trade-off** between these two values:

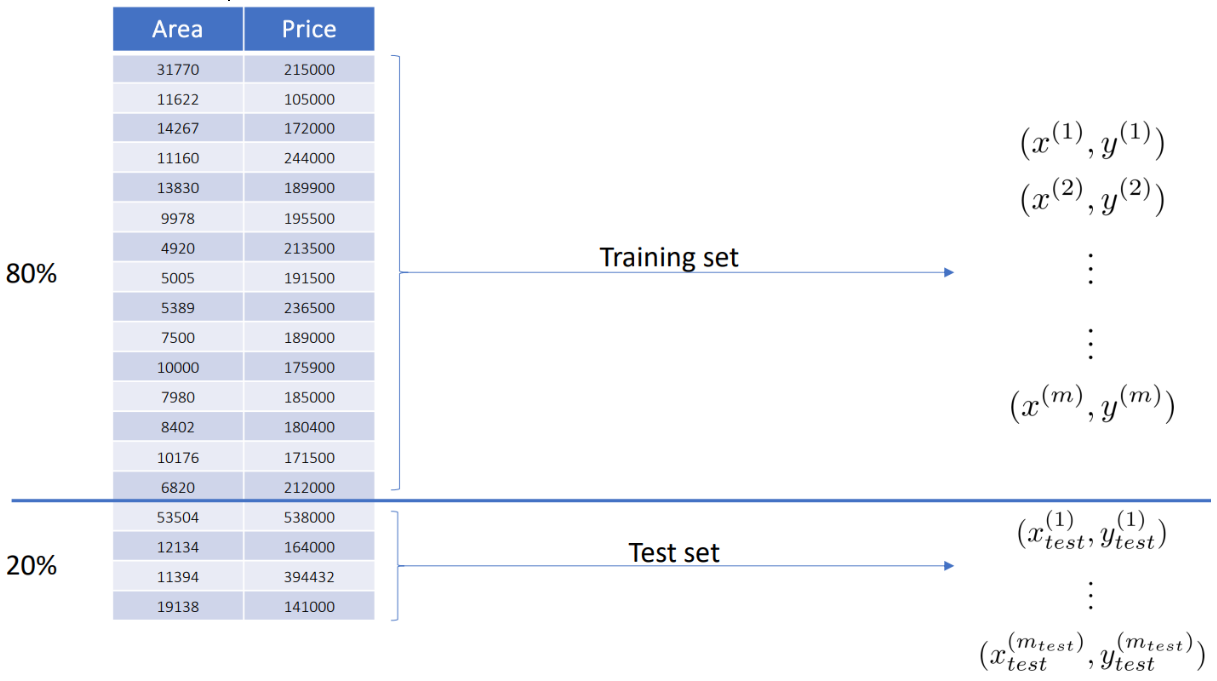
- Bias represents how much the best estimation is able to get close to the ground truth. A high bias denotes a too-simple model that isn't able to predict.
- Variance represents how much a single hypothesis can be different from the *best estimation*. A high variance means that the same model, trained with different datasets, generates different hypothesis. Consequently, very complex hypothesis means a model prone to overfit (not able to generalize).



It is obvious that the **best trade-off** is choosing the model complexity that shows the minimum sum between bias and variance.

Evaluating the hypothesis

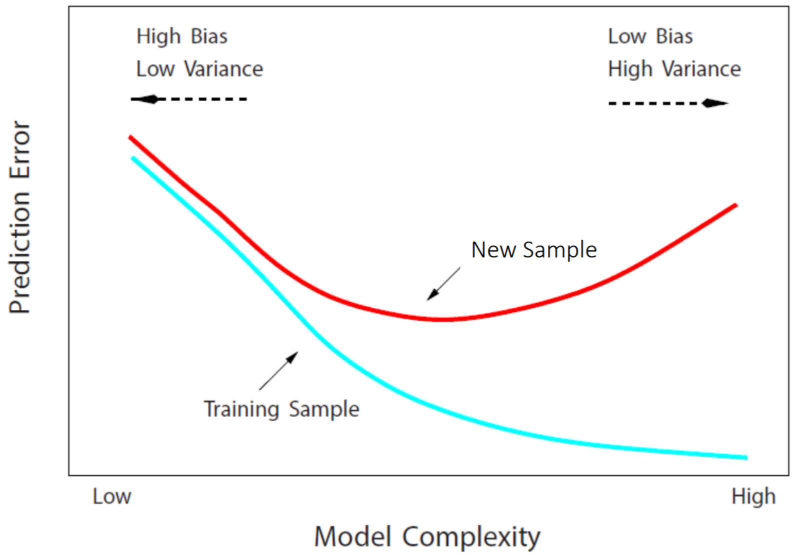
When we train a new model with a data set, usually we don't use the whole set as the ***training set***. In fact, usually it is divided into two pieces: the largest part is used for the training phase, while the remaining part becomes the ***test set,*** which is used to check the behaviour of the model.
To sum up, in the test phase we check if the model is able to generalize the experience gained and how it predicts the future on unseen data.



Area	Price
31770	215000
11622	105000
14267	172000
11160	244000
13830	189900
9978	195500
4920	213500
5005	191500
5389	236500
7500	189000
10000	175900
7980	185000
8402	180400
10176	171500
6820	212000
53504	538000
12134	164000
11394	394432
19138	141000

So, what we'll see regarding to Bias and Variance?

Given the curve of the training sample and comparing it to the behaviour of the model, we can evaluate if its complexity is right, too low, or too high.



Regularization

Then, how can we resolve the overfitting case?

We could reduce the complexity of the model by removing features (manually or with a model selection algorithm). However, this approach is not optimal for an automatic implementation and also leads to a **data loss**. So, it is not always acceptable.

A better solution to address the overfitting is by using the **Regularization**. It consists in a set of techniques whose goal is to keep all the features but, in the meantime, reducing the magnitude of the parameters.
In fact, usually during an overfitting situation the coefficients beyond a certain point influence a lot the function because of their features grade. So, what we want to do now is considering the contribution of the weights while minimizing the cost function.

So, let us consider the following cost function $J(\theta)$:

$$J(\theta) = \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

The *Regularization* technique consists in adding the **squared L2-norm $\boldsymbol{\ell^2}$** (*Euclidean norm*) multiplied for the **regularization hyperparameter $\boldsymbol{\lambda}$** which controls the influence of the regularization with respect to the hypothesis:

$$J(\theta) = \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\boldsymbol{\lambda}}{\boldsymbol{2m}} \cdot \|\boldsymbol{\theta}\|_2^2 =$$
$$= \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\boldsymbol{\lambda}}{\boldsymbol{2m}} \cdot \sum_{j=1}^{n} \boldsymbol{\theta_j^2}$$

It is important to note that the summation over the θs doesn't start from 0 but from 1, since we always consider $x_0^{(i)} = 1$. In fact, what we want to do is to act on the features by controlling the parameters, not to control the parameters.

However, $\boldsymbol{\lambda}$ must be chosen accurately because the bigger it is, the smaller will be the theta and the smoother will be the curve. The model would result in an *underfitting* situation because the minimization focuses on the second term of $J(\theta)$.

Regularized Linear Regression

The regularized cost function for the **linear regression** is:

$$\boldsymbol{J(\theta) = \frac{1}{2m} \cdot \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m} \cdot \sum_{j=1}^{n} \theta_j^2}$$

Instead, for the **logistic regression** the regularized cost function is:

$$\boldsymbol{J(\theta) = -\frac{1}{m} \cdot \sum_{i=1}^{m} \left[y^{(i)} \cdot \ln\left(h_\theta\left(x^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \cdot \ln\left(1 + h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m} \cdot \sum_{j=1}^{n} \theta_j^2}$$

Now, we need to modify the update rule in the gradient descent. By deriving the cost function, we can re-write it as:

$$\boldsymbol{\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)\cdot x_j^{(i)} + \frac{\lambda}{m}\theta_j\right]} \qquad \text{with } j = 1,2,\ldots,n$$

Since we don't want to completely change the behaviour of the GD, θs are updated moving from a ratio of the old theta smaller than 1:

$$\underbrace{\left(1-\underbrace{\overset{\text{much smaller than 1}}{\alpha}\frac{\overset{\text{large}}{\lambda}}{\underset{\text{large}}{m}}}_{0.1}\right)}_{0.9}$$

Regularization with L1-norm

It is also possible to do the regularization by using the **L1-norm** instead of L2, which is defined as the sum of the absolute values of all parameters θ:

$$\|\theta\|_1 = \ell^1 = \sum_{j=1}^{n}|\theta_j| \qquad \textbf{L1 norm}$$

But L1-norm is no more derivable and then the gradient descent cannot be used anymore.

<u>So, when is it usable and why?</u>

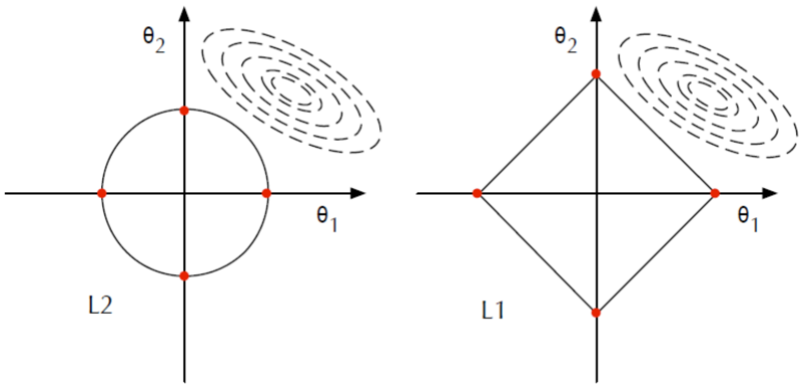
Let us analyse and compare the minimization of the norms, by considering just two parameters:

$$\text{Norm } \ell^2 : \quad \min \sum_{i=1}^{m}(\ldots)^2 \qquad \text{s.t.} \quad \theta_1^2 + \theta_2^2 < t$$

$$\text{Norm } \ell^1 : \quad \min \sum_{i=1}^{m}(\ldots) \qquad \text{s.t.} \quad |\theta_1| + |\theta_2| < t$$

For the ℓ^2, we obtain after the minimization $\boldsymbol{\theta_1^2 + \theta_2^2 < t}$ which denotes the internal area of a circle.

On the other hand, from the minimization of ℓ^1 we obtain $\boldsymbol{|\theta_1| + |\theta_2| < t}$, which denotes the internal area of a rhombus.



When we perform the minimization, we want to minimize that parabolic curve and lie inside the circle (or the rhombus).
It can be proved that it's easier to reach the intercept with the axis in the ℓ^1 case and hence to set that parameter to zero. If the point is located on the y-axis so $\theta_1 = 0$; vice versa, if it is on x-axis then $\theta_2 = 0$.

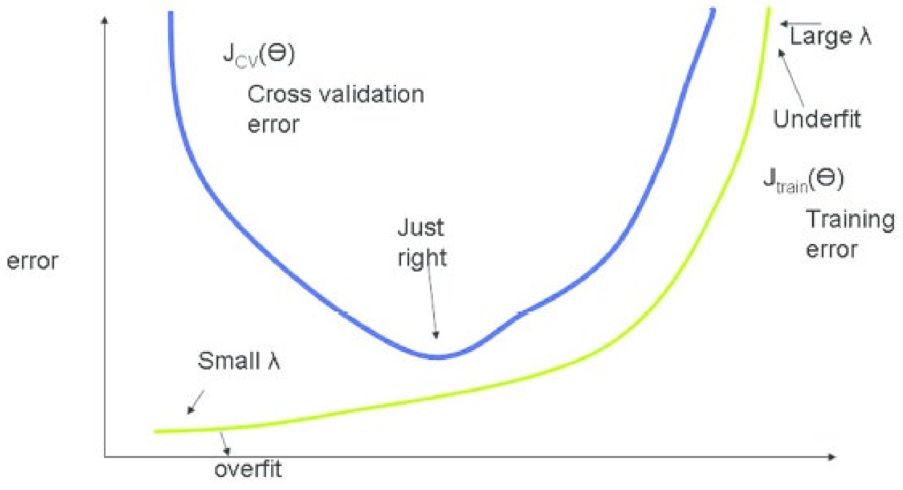
So, in case we need to reduce features the *L1-norm* could be useful because it leads to a simple but effective **feature selection**.

Behaviour in the training with Regularization

We've seen before that depending on the values of *Bias* and *Variance*, and their correlation with the **GER**, we could have situations of underfitting and overfitting.

But now, by using the new regularized cost functions $J(\theta)$ we can train our models to consider the weight of the θs directly correlated to the **regularization parameter λ**.
So, it's necessary to set it accurately in order to get as close as possible to the best approximation. In general:

- if λ is too large, the model is in *underfitting*,
- if λ is too small, the model is in *overfitting*.



Chapter 5 – Build a Machine Learning system

A Machine Learning system grounds on 3 parts:

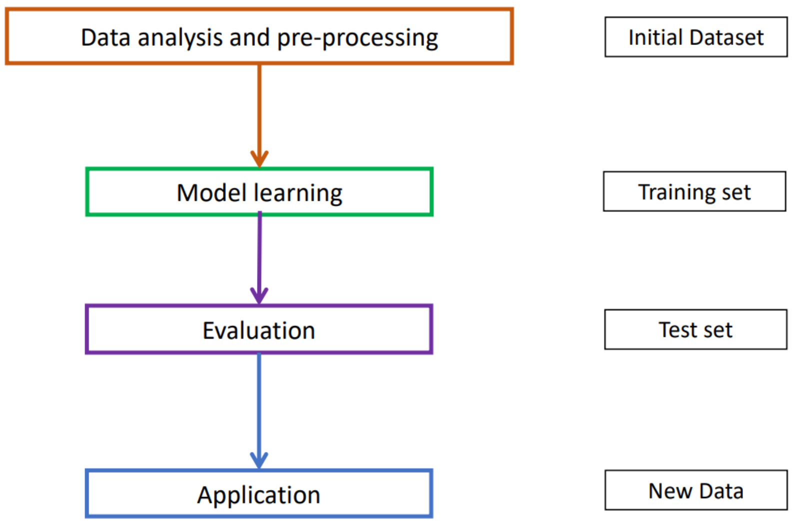
- **Representation** – identify the hypothesis space of the learner and decide which features to use to represent data. We can use different tools during this phase to select the most relevant features to use.
- **Optimization** – choose the method to train the learner (e.g., Gradient descent, greedy search).
- **Evaluation** – using an evaluation function (scoring/objective function) to determine the model's performance and find *good* and *bad* learners.

The general **Goal** is to generalize the model from the training set on unseen samples.

However, even if it's common that the first step is a bit ignored, it is one of the most important one because if we train our model with bad data, we'll get bad performances.

ML system life cycle

Let us see a first review of the so-called **life cycle** of a Machine Learning system:



Data analysis and Pre-processing

We must always consider that real word data are:

- **Incomplete**: the value of some attributes is missing, or some attributes are completely missing.
- **Inaccurate**: data may contain wrong values deriving from inaccurate or partial observations.

The basic and essential principle is: ***Garbage In-Garbage Out***.
If we let in unusable, incorrect, corrupted data, we'll get a bad output as a result.

Thus, it's necessary an accurate data analysis and cleaning, but this phase usually takes a lot of time. This phase is named as **pre-processing,** and it consists in:

- Removing outliers, noise, duplicates.
- Discretize, aggregate, normalize and re-scaling data.
- Creating new attributes.

Outlier removal – Boxplot

Given a data set, we can use *quartiles*, *deciles* and *percentiles* to divide it in equal parts (4 parts, 10 parts, 100 parts). Let us consider for our purpose the **quartiles**:

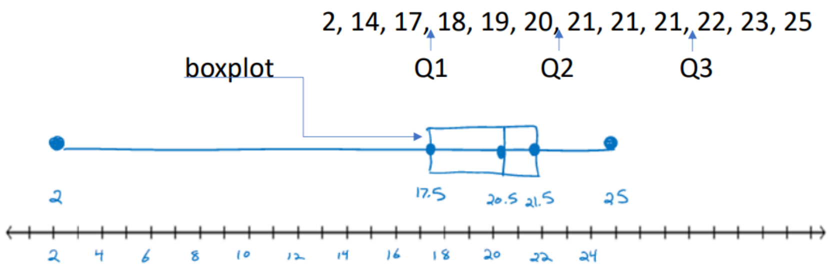
- $\boldsymbol{Q2}$: the median of the data set
- $\boldsymbol{Q1}$: the median of the values that are below Q2
- $\boldsymbol{Q3}$: the median of the values that are above Q2
- $\boldsymbol{IQR = Q3 - Q1}$: it's the interquartile range that tells us the spread of the middle 50% of the data.

So, we compute the interquartile $\boldsymbol{IQR}$ of the data and we'll use ONLY the samples that falls within the range:

$$[\boldsymbol{Q1 - 1.5 \cdot IQR},\quad \boldsymbol{Q3 + 1.5 \cdot IQR}]$$

The samples that are not included are called **outliers** and will no longer be considered for our learning process.

Example



$$Q_2 = \frac{20+21}{2} = 20.5$$
$$Q_1 = \frac{17+18}{2} = 17.5$$
$$Q_3 = \frac{21+22}{2} = 21.5$$

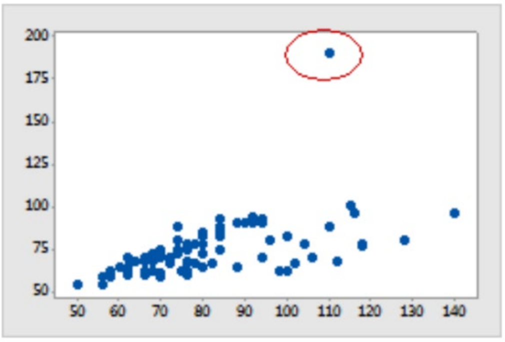
However, an outlier can represent the presence of noise in the training set, or it can indicate that something in particular is happening and it must be considered.

Normalization

Min-max normalization

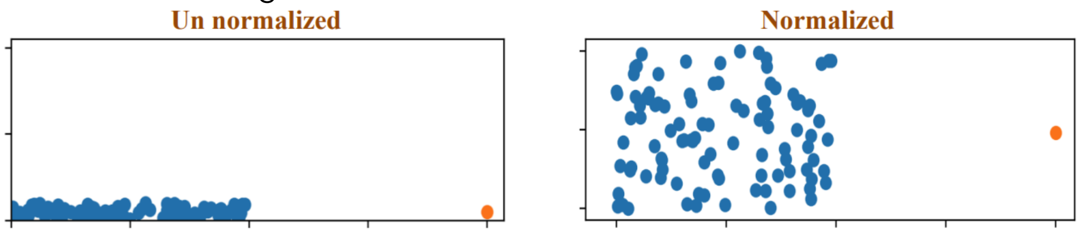
This is a process that forces the values in the interval $[a, b]$. This approach is negatively influenced by **outliers**, values that are way far from the usual behaviour of the model.

What we have to do is setting up *min* and *max values* and in the prediction phase you could meet a new input greater than max:



$$\boldsymbol{x' = \frac{x - min}{max - min} \cdot (b - a) + a}$$

So, the result is something like this:



This technique shifts the distribution to a smaller scale while preserving the relationship with the original data.
However, it doesn't handle really well outliers since we are using the *min* and *max values* and could ruin the normalization process.

Z-score normalization

This normalization process produces a ***zero mean distribution*** through subtraction of the mean and dividing by the standard deviation.
Defined the **mean $\boldsymbol{\mu}$** as the average value of x and the **standard deviation $\boldsymbol{\sigma}$** of x, the new element $\boldsymbol{x'}$ will be defined as:

$$\boldsymbol{x' = \frac{x - \mu}{\sigma}}$$

This normalization technique is globally less affected by outliers given the fact that we don't need to set *min* and *max* values.
However, it might lead to wrong conclusions due to the absence of considering the skewness and the kurtosis of the distribution.

REMAINDER The **skewness** measures the lack of symmetry in a distribution.
Instead, the **kurtosis** is statistical measure that quantifies the shape of a probability distribution, providing information about the tails and peakiness of the distribution compared to a normal distribution.

Feature Selection

Sometimes there are many features in a data set and some of them could be redundant or not important for us. Hence, we need to select the **relevant features** for the learning task without losing relevant information.

This could be done through specific tools:

- **Filter**: measure the importance of each feature to discriminate across the classes, such as *information gain*, *entropy*, etc…
- **Wrappers**: iterative method to find a good subset of features using a subset.
- **Dimensionality reduction**: PCA, SVD.

Hypothesis Evaluation

When we train our model, we use only a subset of the data set (~80%) for the training process. The rest of it (~20%) is used to verify and evaluate the hypothesis initially done. So, the **training/testing procedure** consists in:

1) Obtaining parameters θs from training data by minimizing the cost function and the training error.
2) Then, compute the cost over the test set, depending on if we are considering a regression or a classification problem.

Model Selection: Cross Validation

Imagine comparing different hypothesis in which we change the degree of the polynomial:

$$\begin{array}{lll}
d1 & 1. & h_\theta(x) = \theta_0 + \theta_1 x \quad \rightarrow \theta^{(1)} \rightarrow J_{test}(\theta^{(1)}) \\
d2 & 2. & h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 \quad \rightarrow \theta^{(2)} \rightarrow J_{test}(\theta^{(2)}) \\
d3 & 3. & h_\theta(x) = \theta_0 + \ldots + \theta_3 x^3 \quad \rightarrow \theta^{(3)} \rightarrow J_{test}(\theta^{(3)}) \\
 & \vdots & \vdots \\
d10 & 10. & h_\theta(x) = \theta_0 + \ldots + \theta_{10} x^{10} \quad \rightarrow \theta^{(10)} \rightarrow J_{test}(\theta^{(10)})
\end{array}$$

Theoretically, the hypothesis we'll choose is the one that guarantees the best performance <u>on that specific test set</u>. For this reason, it is likely to be an optimistic estimation of the generalization error.

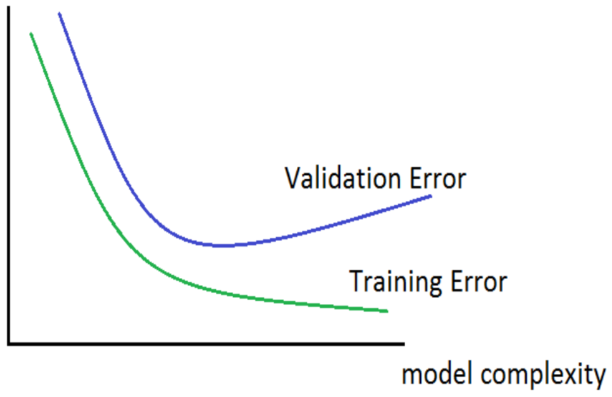
<u>So, how to choose the best model?</u>

We need a new way to find the best hypothesis and we do this by adding the **Cross Validation phase**. Basically, it consists in splitting now the data set in 3 parts:

- **Training set**: to train our model ($\sim 60\%$).
- **Cross validation set**: to validate the trained model. It is also useful for the feature selection and set the hyper parameters.
- **Test set**: useful to evaluate our models and to find the best option among all of them, i.e. which is the best combination of the parameters that gives the best performance.

Our main goal is to find the model with the best performance.
This evaluation cannot be done over the *training set* because we'd risk overfitting. This would mean that the model is behaving too well with respect to the training data such that it is memorizing them, and it won't be able to generalize on unseen data.
Instead, the correct way is to choose the model that has the best performance on the ***validation set***. In this way, we validate it and we evaluate the generalization error on unseen samples.

To finally evaluate the model, we use the **Test set** whose data are not included in the other two sets. This is necessary to see if the model can generalize well enough the experience gained.



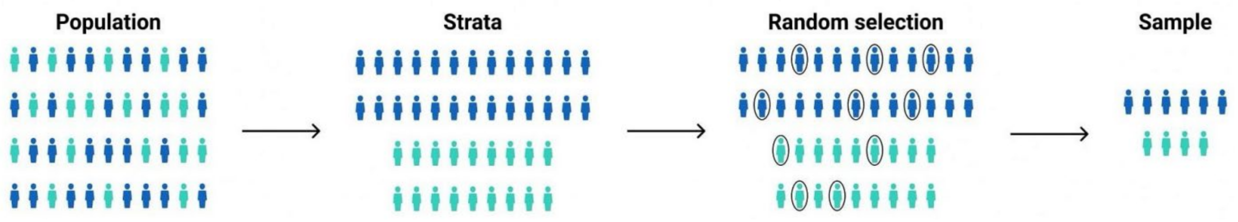
So, two are the main ways to split the data set in *Training* and *Test* sets:

- **Holdout**
- **K-fold cross validation**

Holdout and Stratification

The mechanism of choosing the training and test tests we have described is called **Holdout** and it consists in putting aside the same fraction of dataset over the different models.
However, by doing this we are not sure if that is the best possible partition in order to get the best model. In fact, if in our partitions 60-20-20% we have divergent samples only inside the test set, it means that they're not in the training one and that the model is not trained for that specific situation.
So, the choice of the two sets can strongly influence the training and the test phases as a class might be not represented equally in the two sets. In the end, this could lead to a distortion.

A possible solution in logistic problems is the **Stratification**. It consists in stratifying our different classes and then, by randomly picking from each stratum, we'll distribute the classes equally in the sets. This will maintain the same ratio:



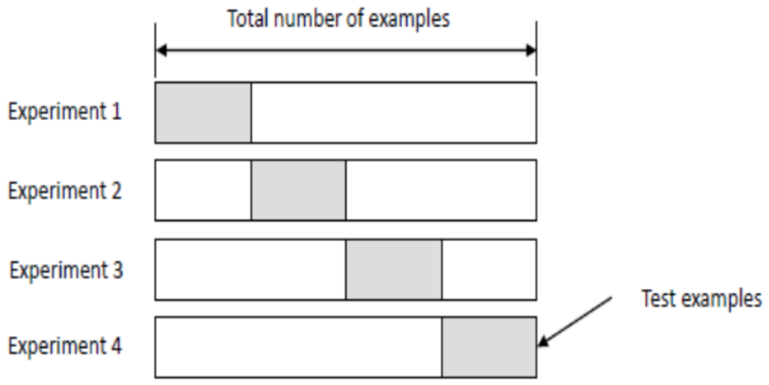
But there are also other problems:

1) We may have **not enough data** to make sufficiently large training and test sets. Giving the priority to a larger test would give us a more reliable estimate of accuracy; on the other hand, a larger training set allows the model to be more representative of the data we have.

2) A single training set doesn't tell us how much the accuracy is sensitive to a particular training sample. So, we're not considering enough patterns within the data set to evaluate the best hypothesis.

K-folds Cross Validation

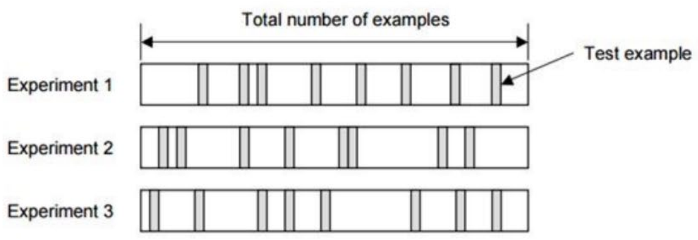
A possible solution to the first problem is represented by the **K-folds Cross Validation**. It consists in randomly partition the original dataset into $\boldsymbol{K}$ equal sized subsets, called **folds**. Of the K folds, one of them is retained as the test set while the others are used together as the training set.
The cross-validation process is repeated K times, using every time a different fold for the test phase and each one is called **experiment**. So, we'll have K different results of which can be calculated the average and then produce a single ***estimation***.

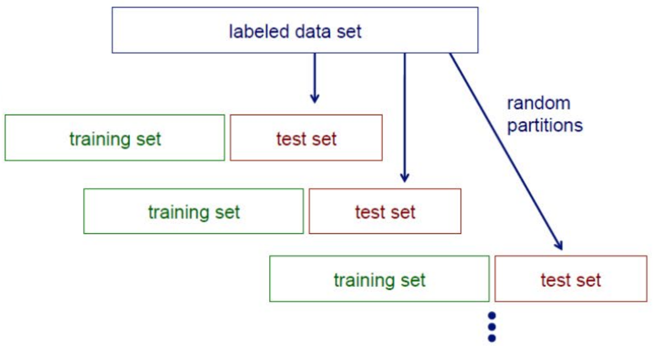


By doing this, we are considering more pattern inside the original data set, and this allows to get closer to the best hypothesis. Also, this technique doesn't require anymore to extract a validation set from the data set.

Random subsampling

A possible approach to address the second issue is represented by the **Random subsampling** technique, which consists in randomly partitioning the available data into *training* and *test sets*.
So, we perform $\boldsymbol{K}$ **data splits** of the dataset and each one randomly selects a fixed number (or percentage) of samples. For each data split, we retrain the classifier from scratch with the training samples and estimate the error over the test samples.



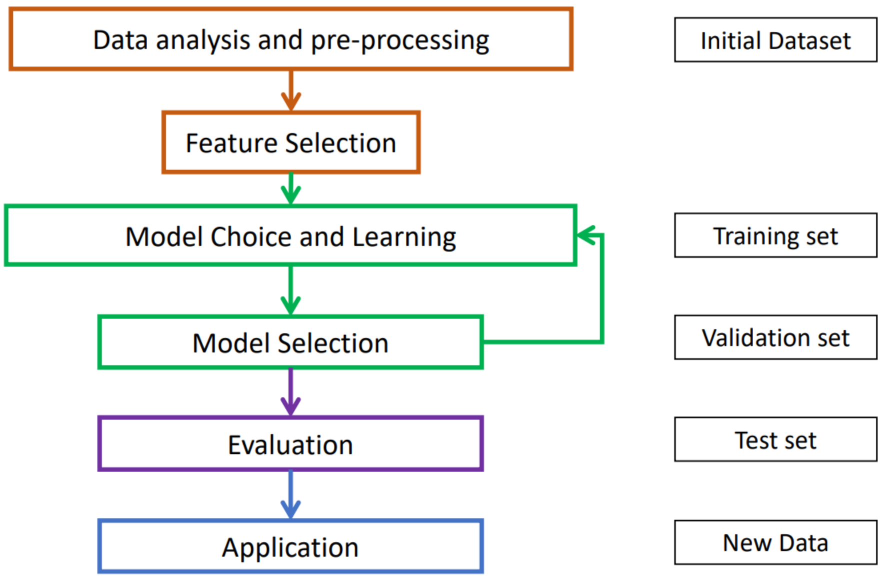


The overall error will be calculated as the summatory of the errors calculated over each fold:

Case i	Train on					Test on	Error
Case1		F2	F3	F4	F5	F1	1.5
Case2	F1		F3	F4	F5	F2	0.5
Case3	F1	F2		F4	F5	F3	0.3
Case4	F1	F2	F3		F5	F4	0.9
Case5	F1	F2	F3	F4		F5	1.1

$$Error = \frac{1}{k}\sum_{i=1}^{k} Error_i$$

ML system life cycle – Revisited



Diagnostics

After doing all the steps in the ML system life cycle, we could have a model with performances that are not satisfying. So, we need to use techniques to guide us to understand why a model doesn't work properly and how to improve it.
This phase is named as **Diagnostics**.

Learning curves

We can use the **Learning curves** to gain insights about what's happening and correctly decide what to do. It is a good technique to:

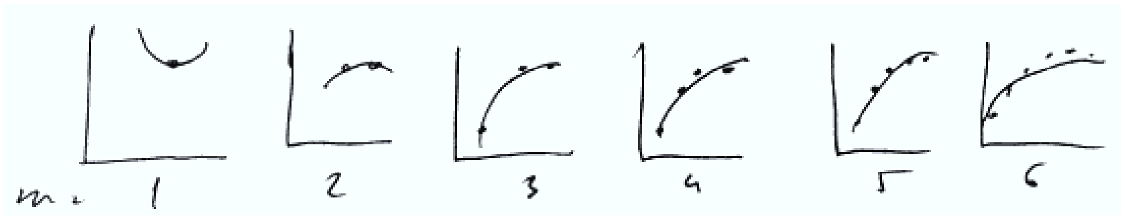
- sanity-check a model,
- improve performance.

They're basically the plot of the two following cost functions with respect to the size of the dataset $\boldsymbol{m}$, which are:

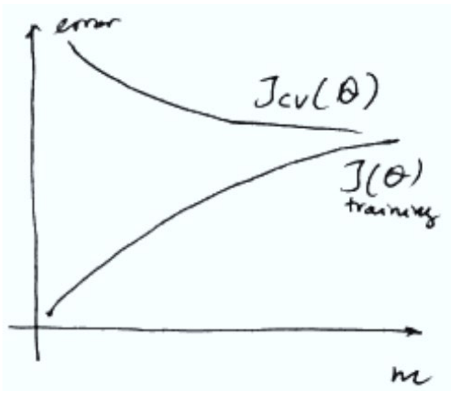
- $\boldsymbol{J_{train}(\theta)}$: training set error
- $\boldsymbol{J_{cv}(\theta)}$: cross-validation error

Suppose now to have a model with the following hypothesis $h_\theta(x)$:

$$\boldsymbol{h_\theta(x) = \theta_0 + \theta_1 \cdot x + \theta_2 \cdot x^2}$$



For each $\boldsymbol{m}$ we compute $\boldsymbol{J_{train}(\theta)}$ and $\boldsymbol{J_{cv}(\theta)}$ and plot the values.
We start from $m = 1$ sample and we compute the values of the 2 functions; so, $m = 2$ and compute the values; etc… The result of this process will be the plot of the learning curve:



As we've seen before, there could be two situations:

- **High Bias** (***Underfitting***) – The model converges to a high cost for both the training and test sets $J(\theta)$. So, the model has a too low complexity.
- **High Variance** (***Overfitting***) – The model converges to a high cost for the test set and to a low cost for the training set. So, with a high $\boldsymbol{m}$ there's a huge gap ΔJ between the two. The model is too complex.

So, based on this analysis we could conclude that:

- **high bias**, we can:
  - increase the complexity by adding features,
  - decrease λ,
  - decrease the amount of data.
- **high variance**, we can:
  - decrease the complexity by removing features,
  - increase λ,
  - increase the amount of data.

Evaluation of a ML system

The evaluation of a Machine Learning system depends on the type of the problem, so if it is a **regression** or a **classification problem**.

Regression – Evaluation

In case of a regression problem, we should consider the difference between values predicted and actual value. So, we have:

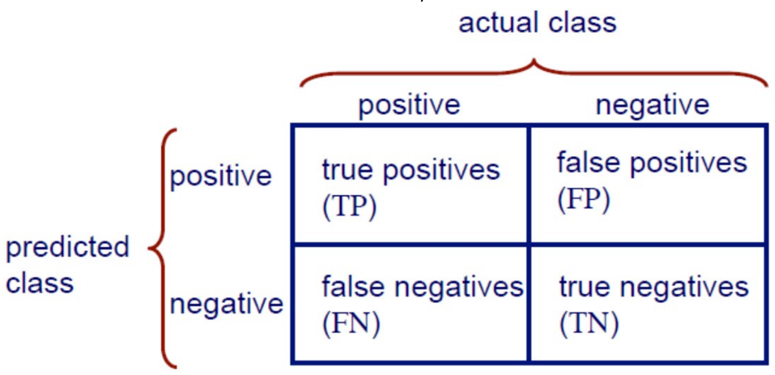
Mean Absolute Error
$$MAE = \frac{1}{m} \cdot \sum_{i=1}^{m} |h_\theta(x_i) - y_i|$$

Mean Squared Error
$$MSE = \frac{1}{m} \cdot \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

Root Mean Squared Error
$$RMSE = \sqrt{\frac{1}{m} \cdot \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2}$$

Classification – Evaluation

We generate the so-called **confusion matrix**, which is defined as:



		actual class	
		positive	negative
predicted class	positive	true positives (TP)	false positives (FP)
	negative	false negatives (FN)	true negatives (TN)

Basically, we associate all the pair of possible combinations of the **actual class** with the **predicted class**.

We can define a lot of **metrics**, such that:

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

Precision $$\frac{TP}{TP + FP}$$

Recall (TP Rate) $$\frac{TP}{TP + FN}$$

Specificity (TN Rate) $$\frac{TN}{TN + FP}$$

Error Rate $$\frac{FP + FN}{TP + TN + FP + FN}$$

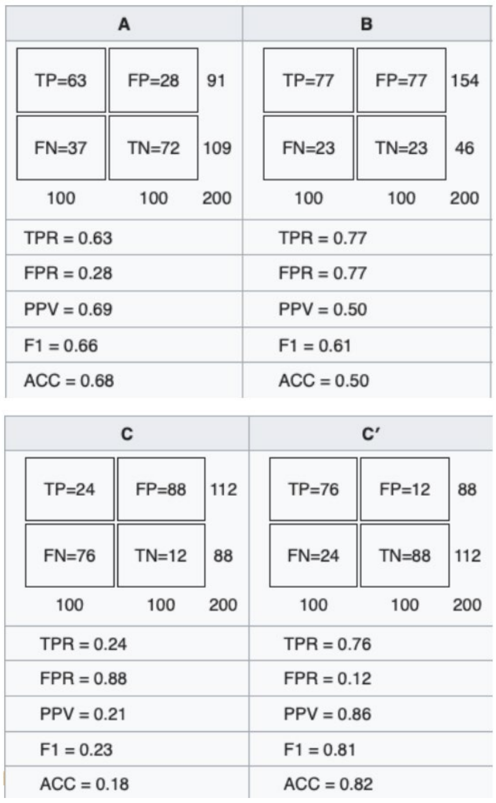
F-Measure $$\frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

FP Rate $$\frac{FP}{FP + TN}$$

Receiver Operating Characteristic (ROC) Space

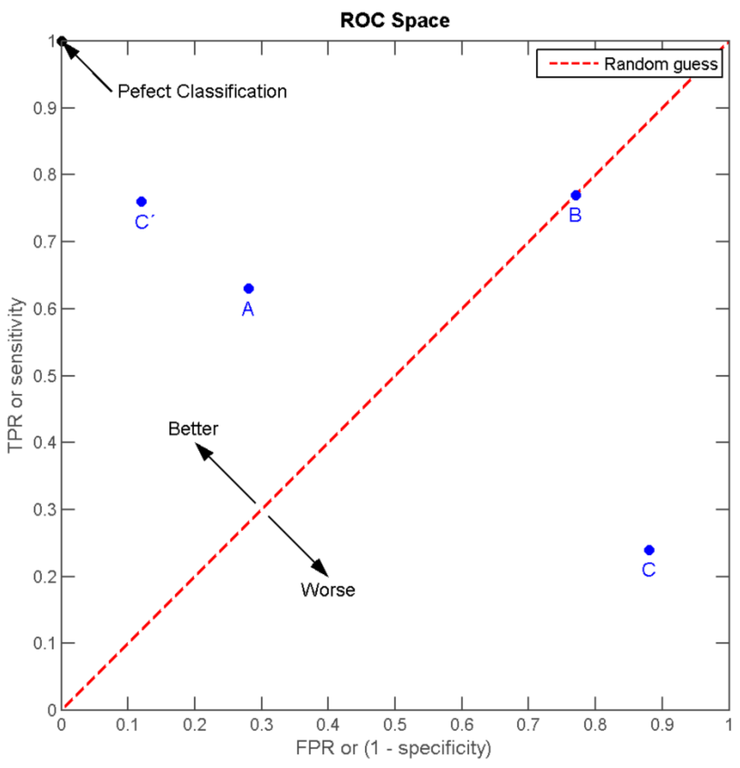
So, how to use those metrics? Which to use to best evaluate the system?

Suppose to have the following situation:



A			B		
TP=63	FP=28	91	TP=77	FP=77	154
FN=37	TN=72	109	FN=23	TN=23	46
100	100	200	100	100	200
TPR = 0.63			TPR = 0.77		
FPR = 0.28			FPR = 0.77		
PPV = 0.69			PPV = 0.50		
F1 = 0.66			F1 = 0.61		
ACC = 0.68			ACC = 0.50		

C			C′		
TP=24	FP=88	112	TP=76	FP=12	88
FN=76	TN=12	88	FN=24	TN=88	112
100	100	200	100	100	200
TPR = 0.24			TPR = 0.76		
FPR = 0.88			FPR = 0.12		
PPV = 0.21			PPV = 0.86		
F1 = 0.23			F1 = 0.81		
ACC = 0.18			ACC = 0.82		

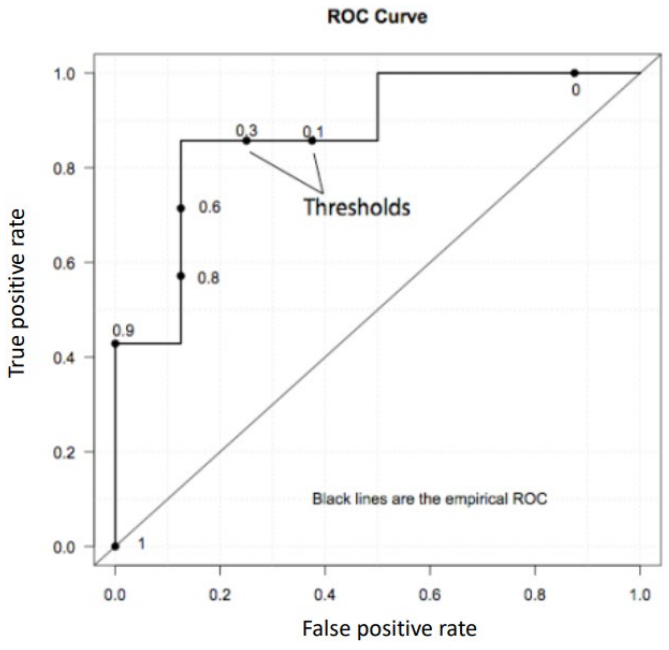


To compare different classification models, we can use the **Receiver Operating Characteristics** (**ROC**) **Space** which is a scatter plot with the **FPR** (False Positive Rate) on the x-axis and the **TPR** (True Positive Rate) on the y-axis.
The optimal solution is represented by the point $(\mathbf{0}, \mathbf{1})$ at the top-left of the space. So, we can assume that the nearer the point, the better the corresponding model. To evaluate our model, we measure the distance of every point to the optimal point and the smaller it is the better is the hypothesis. For instance, in the graph above the optimal point is represented by **C'**.

Moreover, we can see a diagonal in the middle of the space where $\text{TPR} = \text{FPR}$ and that basically is like a coin toss: being under this diagonal means that our hypothesis has less probability than a coin toss.

ROC Curve

Within the ROC Space, we can also evaluate the dependence between the threshold used for classifying the sigmoidal output and the FPR/TPR. So, we can draw a curve that basically indicates how the performance change by changing the threshold of the binary estimator.
We start by graphing from a value of the threshold of $T = 0$ to $T = 1$, which means drawing the first dot on $(\mathbf{1}, \mathbf{1})$ and the last dot on $(\mathbf{0}, \mathbf{0})$. The threshold that gives the closest dot to the optimal point $(\mathbf{0}, \mathbf{1})$ will be chosen.



Comparing systems

An important problem is understanding how much the results of our experiments are due to actual learning or bad engineering of the problem or to random fluctuations of the data.

So, to compare two systems and evaluate which one is better we use the **paired sample t-test**, which is a statistical procedure used to determine whether the mean difference between two sets of observation is zero and so are identical. It takes this name because each entity is measured twice, resulting in pair of observations.

The *paired t-test* is based on the hypotheses:

Null Hypothesis: the two ML systems have the same accuracy

Alternative Hypothesis: one of the two systems is more accurate than the other

So, the **Hypothesis test** uses the ***paired sample t-test*** to determine the probability $\boldsymbol{p}$ that the mean difference supports the null hypothesis. If p is sufficiently small $(p < 0.05)$, then we are rejecting the null hypothesis.

In our case, let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two models and let us measure the accuracy $\boldsymbol{y}$ (varying in $\boldsymbol{n}$ values), computed by varying the threshold:

$$\overrightarrow{\boldsymbol{y^a}} = \{y_1^a, y_2^a, \dots, y_n^a\} \qquad \overrightarrow{\boldsymbol{y^b}} = \{y_1^b, y_2^b, \dots, y_n^b\}$$

The test consists in:

1) given the vector $\vec{\boldsymbol{\delta}} = \{y_1^a - y_1^b,\ \ y_2^a - y_2^b, \dots,\ \ y_n^a - y_n^b\}$, compute:

$$\textbf{sample mean} \qquad \bar{\boldsymbol{\delta}} = \frac{\boldsymbol{1}}{\boldsymbol{n}} \sum_{i=1}^{n} \boldsymbol{\delta_i}$$

2) calculate the **t-statistic**:

$$\boldsymbol{t} = \frac{\bar{\boldsymbol{\delta}}}{\sqrt{\frac{\boldsymbol{1}}{\boldsymbol{n \cdot (n-1)}} \cdot \sum_{i=1}^{n} (\boldsymbol{\delta_i} - \bar{\boldsymbol{\delta}})^2}}$$

3) now find the corresponding **p-value** by looking up $\boldsymbol{t}$ in **Student's t-distribution**, with $n - 1$ degrees of freedom.

It is important to note that we can model δ as a *gaussian probability* because of the independence between the variables due to the randomness of the observations.

So, obtained the value of $\boldsymbol{p}$, depending on the relation between $\boldsymbol{t}$ and $\boldsymbol{p}$ we get a different information about $\boldsymbol{a}$ and $\boldsymbol{b}$:

- $\boldsymbol{p = 2 \cdot P_r(T > |t|)} \longrightarrow$ **Two-tailed**: *Is the accuracy of the two systems different?*
- $\boldsymbol{p = P_r(T > t)} \longrightarrow$ **Upper-tailed**: *Is a better than b?*
- $\boldsymbol{p = P_r(T < t)} \longrightarrow$ **Lower-tailed**: *Is b better than a?*

where $\boldsymbol{P_r}$ is the *Student density function*.

Coefficient of determination – R^2

$\boldsymbol{R^2}$ is a statistical measure that provides an estimate of the proportion of the variation in the dependent variable that is explained by the independent variable in the model. It is defined as:

$$\boldsymbol{R^2 = 1 - \frac{RSS}{TSS}}$$

where:

$$RSS = \sum_{i=1}^{m} \left(y^{(i)} - h\left(x^{(i)}\right)\right)^2$$ **Residual Sum of Squares**

$$TSS = \sum_{i=1}^{m} \left(y^{(i)} - \bar{y}\right)^2$$ **Total Sum of Squares**

Hence, $\boldsymbol{R^2} \in [0,1]$ and we can say that:

- $\boldsymbol{R^2}$ close to $\mathbf{1}$ suggests that the model fits the data well, i.e. the model is able to explain a large portion of the data variation.
- $\boldsymbol{R^2}$ close to $\mathbf{0}$ suggests that the model does not fit the data well, i.e. the model is not able to explain the variation in the dependent variable.