Summary

Disclaimer – LEGGI

Queste dispense potrebbero contenere errori sparsi che in genere sono segnati come ad esempio “vai a vedere dalle slide” o cose del genere, ciò non toglie che non sono perfette.

Nel caso ti siano piaciute davvero tanto guarda

IT19X0538741351000043108995

questo è l’IBAN, invia a Giuseppe Murgolo e grazie del regalo di Natale.

A prescindere da tutto, invio queste dispense per un po’ di positive vibes quindi se ti hanno aiutato sono davvero contento.

Capitolo 1 Lezione 2022-10-03

1.1 Machine learning (ML)

Machine learning was born because of the need of an automatically analysis of data which leads to human like decision. It is a set of methods that can automatically detect patterns in data in order to predict the future (kinda).
Since ML includes a lot of shades of uncertainty, probabilistic and statistic models' rules can be used in order to make the best prediction about the future by taking decisions based on data.

1.1.1 Some of many definitions

Herbert Alexander Simon

Machine Learning is concerned with computer programs that automatically improve their performance through experience.

Tom Mitchell (The one that He considers the most complete)

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

Generally

Learning means improving with experience at some tasks so improving over task T with respect to performance P based on experience E.

1.2 Supervised and unsupervised learning (1st classification)

Supervised learning (SL)

Also called predictive learning, it aims to mapping from inputs x to outputs y given a labeled set of (In, Out) pairs. $D = \{(x_i, y_i)\}_{i=1}^{N}$ where D is the training set of N pairs.
Usually x_i is a vector where each component is called **feature**, **attribute,** or **covariates**.
Since the outcome y_i can be everything theoretically, if it is a categorial question (Type of film ex.) the problem is known as **classification** or **pattern recognition** else, if y_i is a real value (Price of a house) it is a **regression**.

Unsupervised learning (USL)

In this approach we are given **only** inputs $D = \{x_i\}_{i=1}^{N}$ where the goal is to find **interesting pattern**. Since we don't have y_i like SL, we don't have to worry about a error matric to use.

Reinforcement Learning (RL) – Just a note

There is also **reinforcement learning,** which is not explained completely in the book, but it aims on act or behave when given an occasional reward or punishment signal (Like teaching a dog to sit OR not to do something)

1.3 Discriminative and generative models (2nd classification)

Discriminative model

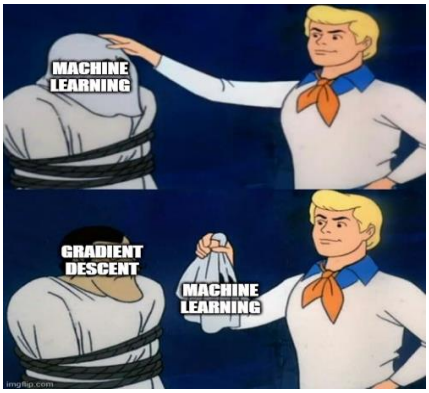
This is a type or probabilistic classifier in order to estimate the most probable outcome by taking an input $p(y|x)$ (Conditional probability)

Generative model

This is a type or probabilistic classifier in order to estimate a joint model and form the probability $p(x, y)$

1.4 General notation

m = Number of training examples
n = Number of features
x = "**Input**" variable/ feature
y = "**Output**" variable / "**target**" variable
(x, y) = tuple rappresenting one training example
$\left(x^{(i)}, y^{(i)}\right)$ = training example i^{th}
$x_j^{(i)}$ = the training example i^{th}, the j^{th} feature



1.5 Linear regression (LR) with one variable

One important aspect of the **regression** is that given an input (discrete) the output will be **continuous**; in this case the output will be a **straight line**.

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

or sometimes just $h(x)$.

1.5.1 LR Cost function and some considerations

Defining a cost function is important because we need to estimate how much accurate are our prediction $h_\theta(x) = \theta_0 + \theta_1 x$ compared to the real value y.

Coefficients θ randomly choosed

The easiest approach consist of choosing random coefficients θ hoping to find a good approximation.

Intuition of LR

We search for the minimum parameters such that the average prediction $h_\theta\left(x^{(i)}\right)$ is as much close to the real values $y^{(i)}$. The problem is defined as it follows:

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

Where every prediction is defined as $h_\theta\left(x^{(i)}\right) = \theta_0 + \theta_1 x^{(i)}$.
The cost function is defined as:

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

Note: this form or error is called **Mean Square Error** (**MSE**).

1.6 Gradient Descent (GD) (or Steepest Descent)

Note: in the book this topic is introduced as **"Steepest Descent"**, but "**Gradient Descent**" is practically a synonym.

We generalize the problem of the LR with *one* variable with n variables because problems concerning this kind of analysis usually contains more than one feature.

$$\begin{matrix} Before & & After \\ J(\theta_0, \theta_1) & \rightarrow & J(\theta_0, \dots, \theta_n) \end{matrix}$$

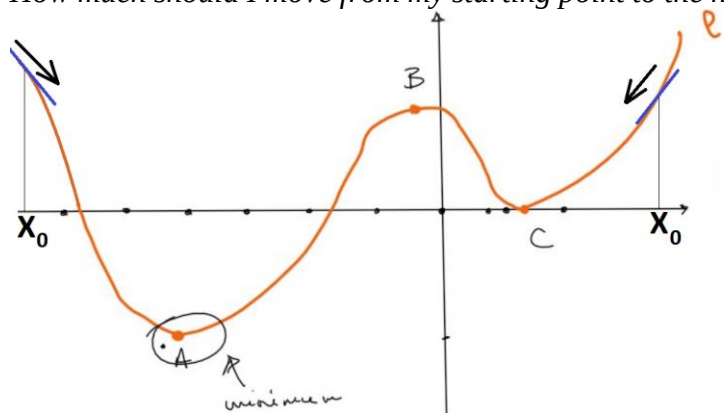
So the general goal is to minimize this new cost function:

$$\begin{matrix} Before & & After \\ \underset{\theta_0,\theta_1}{\operatorname{argmin}} J(\theta_0, \theta_1) & \rightarrow & \underset{\theta_0,\dots,\theta_n}{\operatorname{argmin}} J(\theta_0, \dots, \theta_n) \end{matrix}$$

Since this kind of algorithm (Or method) is equivalent for n features like for just *one*, we will proceed by using $J(\theta_0, \theta_1)$ instead of the "n *features*" one.

1.6.1 Algorithm and explanation

The GD uses the derivatives in order to find the direction to the nearest minimum (This does not mean to the **global minimum** unless the function is **convex**) and a parameter α ,called **learning rate** (yet again, the slides uses this name, but it is also called **step size** on the book) is used as a value of "*How much should I move from my starting point to the next point presumably nearer to a minimum?*".



Example: by watching at this random function, we can see that in the points X_0, knowing their derivatives (Which are the blue slopes), we can determine how to aim through a minimum.
This means that knowing the derivative of a point we can move through its nearest minimum, and we do that at steps of "α".

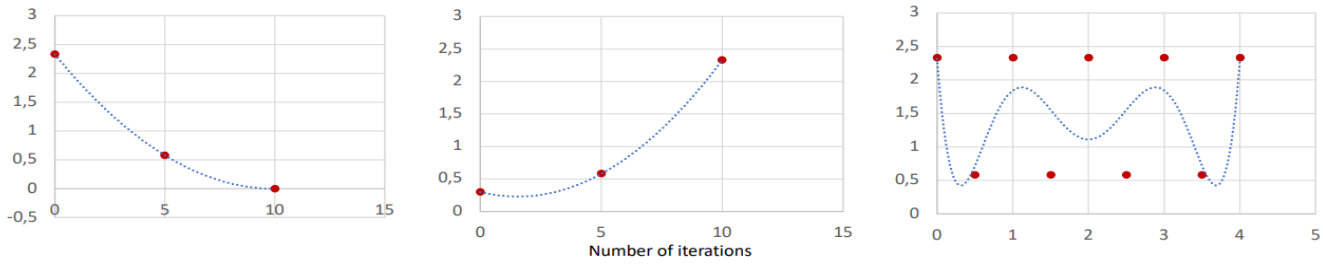
The problem it is defined as:

$$\theta_j := \theta_j - \alpha \frac{\partial J(\theta_0, \theta_1)}{\partial \theta_j} \mid \alpha > 0$$

Note: before changing the value of θ_j through this formula we first need to calculate all θ_js by using the formula and then, after we have all the new $\theta_j s$, we can change the old values with the new ones.

Considerations

1) Choosing a too small α will cause to slow down the calculations, otherwise choosing a too big one will cause the algorithm to diverge. One problem is to find a well α such that **converges** with a reasonable amount of calculations ("*A reasonable amount of time*" is not considered because the same calculations can be done at different speed e.g. computing on two different computers result in a different amount of time)
2) By the point of view of an algorithm, "**converging**" means that we will get closer and closer to the minimum without reaching it completely (It converge with an infinite number of iteration so to speak) and practically we need to set some "**ending conditions**" like a limited number of iteration and/or a tolerance somewhere (like if $[J(\theta_{new}) - J(\theta_{old})]$ is less than the set tolerance then stop).
3) Bad values of α may result in **converging slowly**, **diverging** or even **bouncing** (Like a sinusoidal function).



4) In the formula the sign of the partial derivative is negated $\left(-\alpha \frac{\partial J(\theta)}{\partial \theta_j}\right)$ because if the derivative is **negative** it means that the function is decreasing **left-to-right,** so the minimum is to the **right,** and it is needed to increase the value of the point X_0 (Negative derivative negated equals a positive number).
On the other hand, if the derivative is **positive** it means that the function is increasing **left-to-right** and that means that it decreases in the opposite direction, so we need to move to the **left** and that means decreasing the value of X_0 (Positive derivate negated equals a negative value).

1.6.2 Deriving the formula (Mathematics)

We have defined the cost function as:

$$J(\theta_0,\theta_1)=\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2 \;|\rightarrow|\; h_\theta\left(x^{(i)}\right)=\theta_0+\theta_1x^{(i)}$$

Since we introduced the partial derivatives in respect of θ_0 and $\theta_1$it follows that:

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

Due to the continuity of the summation, we can derive its augment:

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

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

And now we analyze the cause of $\boldsymbol{j=0}$ and $\boldsymbol{j=1}$:

$$\boldsymbol{j=0}\;\rightarrow\;\frac{\partial h_\theta\left(x^{(i)}\right)}{\partial\boldsymbol{\theta_0}}=\frac{\partial h_\theta\left(\boldsymbol{\theta_0}+\theta_1x^{(i)}\right)}{\partial\boldsymbol{\theta_0}}=\boldsymbol{1}$$

$$\boldsymbol{j=1}\;\rightarrow\;\frac{\partial h_\theta\left(x^{(i)}\right)}{\partial\boldsymbol{\theta_1}}=\frac{\partial h_\theta\left(\theta_0+\boldsymbol{\theta_1x^{(i)}}\right)}{\partial\boldsymbol{\theta_1}}=\boldsymbol{x^{(i)}}$$

And so, in the end:

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

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

Multiple features analysis

if we consider the following definition of the hypothesis

$$h_\theta\left(x^{(i)}\right)=\theta_0x_0^{(i)}+\theta_1x_1^{(i)}+\cdots+\theta_nx_n^{(i)}$$

with $x_0^{(i)}=1$ we can generalize the formula for all $j=0,1,\dots,n$:

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

Since we have $h_\theta\left(x^{(i)}\right)$ we can compute its partial derivative:

$$\frac{\partial h_\theta\left(x^{(i)}\right)}{\partial\theta_j}=\frac{\partial}{\partial\theta_j}\left(\theta_0x_0^{(i)}+\theta_1x_1^{(i)}+\cdots+\theta_nx_n^{(i)}\right)=x_j^{(i)}$$

In the end we get the following formula:

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

Capitolo 2 Lezione 2022-10-06

2.1.1.1.1 1_intro_to_ml_linear_regression.pdf (STILL THIS SLIDE)

2.2 Linear Regression with multiple features

As we told, generally speaking we will face multiple features with $n \geq 1$ so comparing to the previous LR with one variable we change the hypothesis:

$$Before \rightarrow h_\theta(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$$

$$After \rightarrow h_\theta(x^{(i)}) = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_n x_n^{(i)}$$

If we consider $x_0^{(i)} = 1$ for every $i = 1, \dots, m$ we can compact the formula even more:

$$h_\theta(x^{(i)}) = \theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \cdots + \theta_n x_n^{(i)} = \sum_{j=0}^{n} \theta_j x_j^{(i)}$$

As a point of view of a matrix representation:|

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} = 1 \\ x_1^{(i)} \\ \dots \\ x_n^{(i)} \end{bmatrix} \in R^{n+1} \quad | \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{bmatrix} \in R^{n+1}$$

So in the end we get

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

Little reminder (matrices product)

$\theta^T \in R^{(1)*(n+1)}$ and $x^{(i)} \in R^{(n+1)*(1)}$ so $h_\theta(x^{(i)}) = \theta^T x^{(i)} \in R^{(1)*(1)}$ (R).

1) $x^{(i)} \in R^{(n+1)} = R^{(n+1)*(1)}$ because the dimension $(n+1)$ implies $(n+1) * (1)$

2) Since this is now a generalized "**matrix multiplication**" (*Prodotto riga per colonna*) the end result is a $R^{(1)*(1)} = R$, so just a real number since they were vectors (**cross product**).

Since vector are "matrices" with dimension for the "rows" equal to one, we can generalize the product between 2 matrices:

$$R^{(a)*(c)} * R^{(c)*(b)} = R^{(a)*(b)}$$

If the first matrix was a vector $c = 1$, if the second one was a vector $b = 1$ but in the end the final dimension would still be $R^{(a)*(b)}$.

2.3 Batch Gradient Descent vs Stochastic Gradient Descent

Batch Gradient Descent (BGD)

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

Uses **all** training samples into consideration in order to take a **single step**.

Advantage: Moves directly into the minimum and convergence is sure (Depending on α).

Disadvantage: Very slow to compute since all samples are taken into consideration.

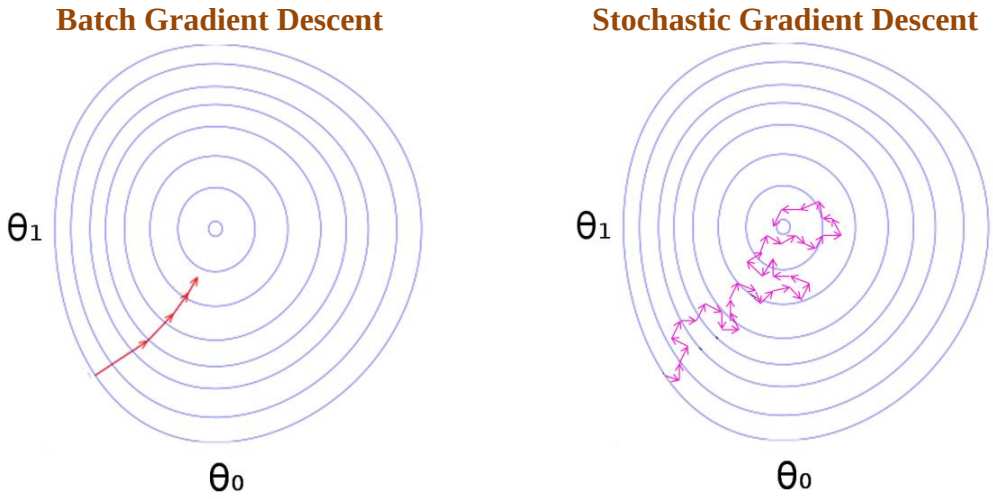
Stochastic Gradient Descent (SGD)

$$\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

One random sample is taken every step.

Advantage: Faster than **BGD** so it is useful when there are a lot of samples to analyze.

Disadvantage: Convergence in not sure (Even with a well α).



2.3.1 Mini-Batch Gradient Descent

This algorithm is a compromise between a **BGD** and a **SGD**. For every step we take b number of training samples into consideration ($1 \leq b \leq m$). Usually is taken a number $b \ll m$. We can analyze that when $b = 1$ this algorithm responds like an **SGD** and when $b = m$ like a **BGD** but usually b ~100.

Note: The algorithm is faster than a **BGD** and has a more stable level of convergence compared to **SGD** (Also allows the use of vectorization libraries rather than computing each step separately)

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

The differences from the **BGD** are:

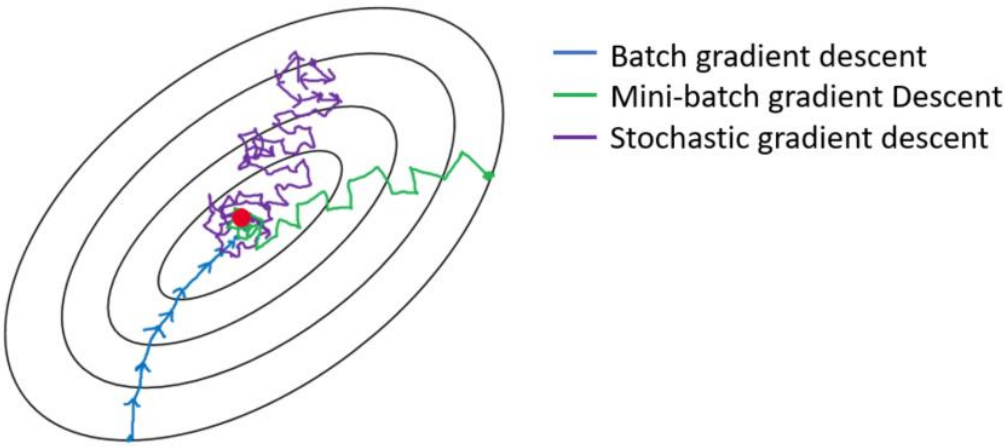
First) Each iteration we are considering just b samples instead of m in the BGD.

Second) We randomly pick a batch of b samples from the whole m dataset, but we need to be careful about considering all the samples inside the dataset.

How?

Each iteration we must be sure that among the b samples we only took elements that have never been taken into consideration before.

Approximately at the m/b iteration we will have considered all the samples inside the m dataset and this whole process is repeated.



2.4 *LR – adding features and polynomial regression*

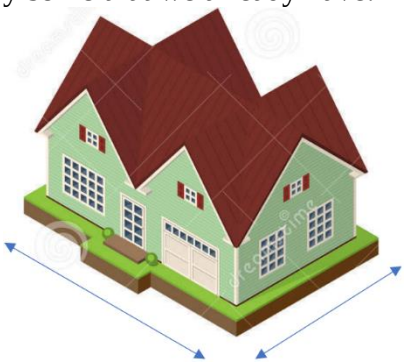
Adding feature

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

$$h_\theta\left(x^{(i)}\right) = \theta^T x^{(i)} = \theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_n x_n^{(i)}$$

But nothing stops us from creating new features which are constructed by some that we already have.

Example: Assuming that we have a house that we want to sell, and we have the training samples involving information about other houses. Our $x^{(i)}$ is defined as:



$$x^{(i)} = \begin{bmatrix} front^{(i)} \\ side^{(i)} \end{bmatrix}$$

At first glance we may think to predict the price with our function:

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

But we can also use the information about the area of the house which is calculated as: $area^{(i)} = front^{(i)} * side^{(i)}$

And so we can define a new hypothesis:

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

Considerations

1) By using the $area^{(i)}$ we have reduced the complexity of the problem because we are now using less features for the prediction $h_\theta\left(x^{(i)}\right)$.

2) The area of the house is a much more useful as information for our hypothesis (price) in this case.

Generally speaking

Supposing we have an input $x^{(i)}$, we can introduce a new feature $x_{n+(k+1)}^{(i)}$ $|\rightarrow|$ $k \in N$, which is a transformation of already existing features:

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

By doing so we can define a new prediction function $h'_\theta(x)$ which involves the use of this new feature.

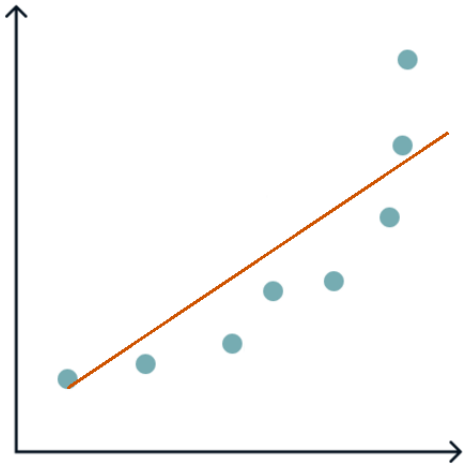
Polynomial regression

A polynomial regression involves the use of a polynomial of grade t, in our case defined as:

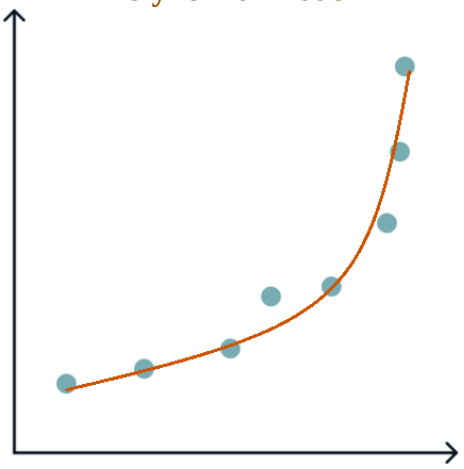
$$h_\theta(x) = \theta_0 x^0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_t x^t$$

We are using a $x^{(i)}$ composed just by one feature (simply called x) and our goal is the same as **LR** with one feature but right now we have more coefficients θs to find resulting in a much flexible prediction

Linear model

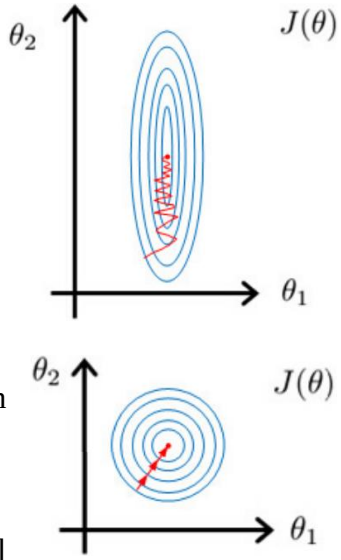


Polynomial model



2.5 *Feature scaling and normalizations technique*

Feature scaling



When analyzing a problem it could be useful to reduce the size of the features we are possessing

Example: supporting we have information about houses (area) and the number of bedrooms:

$x_1 =$ area, $x_1 \in [0,2110]$

$x_2 =$ bedrooms, $x_2 \in [1,5]$

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

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

Since we know the max area and the max number of bedrooms we can scaling these two factors:

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

By doing so we resized our parameters in a bounding region of $[0,1]$ ($x_1, x_2 \in [0,1]$) (Hopefully improving the computing performance)

2.5.1 Min-max and Z-scope normalization

Note (Notation): x' is the resized version of the element x

Min-max normalization

The min-max normalization is a method to scale the parameters . The general formula is:

$$x_j' = \frac{x_j - min}{max - min} \; |\rightarrow| \; x_j \in [min, max]$$

Using this formula we bound the scaled parameters between $[0,1]$ so $x_j' \in [0,1]$

Note: Since $x_j \in [min, max]$ if you substitute x_j with min and max you'll get the new boundary $x_j' \in [0,1]$

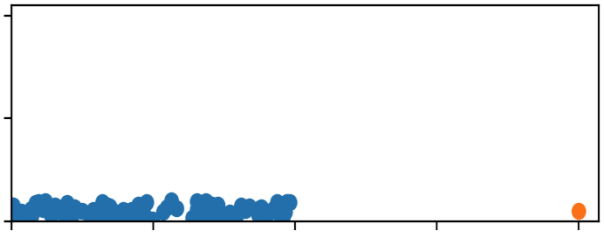
In the slides it uses the formula for resizing the parameters into a bound of $[a, b]$ with the formula:

$$x_j' = \frac{x_j - min}{max - min}(b - a) + a \; |\rightarrow| x_j' \in [a, b]$$

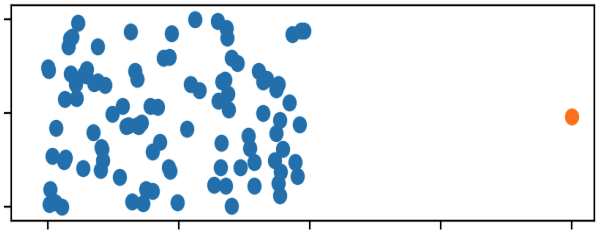
Advantage: shifts the distribution to a smaller scale preserving the relationship with the original data.

Disadvantage: doesn't handle well outliers (Since using both the formulas an outlier will be consider as a min or max and therefore "corrupting" the normalization)

Un normalized



Normalized



Z-scope normalization

By defining $mean$ (μ) as the average value of x and $devstd$ (σ) as the standard deviation of x, the new element will be defined as:

$$x' = \frac{x - mean}{devstd} = \frac{x - \mu}{\sigma}$$

Advantage: Less affected to outliers compared to min-max normalization

Disadvantage: Might lead to wrong conclusion due to the absence of considering skewness and kurtosis of the distribution.

2.6 *Normal equations*

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

After we define the matrix of the input $X \in R^{m*n}$ and the vector of the output $y \in R^m$ as:

$$X = \begin{bmatrix} x^{(1)T} \\ \dots \\ x^{(i)T} \\ \dots \\ x^{(m)T} \end{bmatrix} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_j^{(1)} & \dots & x_n^{(1)} \\ \dots & & & & & \\ 1 & x_1^{(i)} & \dots & x_j^{(i)} & \dots & x_n^{(i)} \\ \dots & & & & & \\ 1 & x_1^{(m)} & \dots & x_j^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \quad y = \begin{bmatrix} y^{(1)} \\ \dots \\ y^{(i)} \\ \dots \\ y^{(m)} \end{bmatrix}$$

The new form of our cost function $J(\theta)$ is:

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

Proof: The original cost function is $J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

$$X\theta = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_j^{(1)} & \dots & x_n^{(1)} \\ \dots & & & & & \\ 1 & x_1^{(i)} & \dots & x_j^{(i)} & \dots & x_n^{(i)} \\ \dots & & & & & \\ 1 & x_1^{(m)} & \dots & x_j^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_j \\ \dots \\ \theta_n \end{bmatrix} = \begin{bmatrix} h_\theta\left(x^{(1)}\right) \\ \dots \\ h_\theta\left(x^{(i)}\right) \\ \dots \\ h_\theta\left(x^{(m)}\right) \end{bmatrix}$$

So we can explicity write the difference $X\theta - y$ as:

$$X\theta - y = \begin{bmatrix} h_\theta\left(x^{(1)}\right) \\ \dots \\ h_\theta\left(x^{(i)}\right) \\ \dots \\ h_\theta\left(x^{(m)}\right) \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \dots \\ y^{(i)} \\ \dots \\ y^{(m)} \end{bmatrix} = \begin{bmatrix} h_\theta\left(x^{(1)}\right) - y^{(1)} \\ \dots \\ h_\theta\left(x^{(i)}\right) - y^{(i)} \\ \dots \\ h_\theta\left(x^{(m)}\right) - y^{(m)} \end{bmatrix}$$

So it follows that the product $(X\theta - y)^T(X\theta - y)$ can be written as:

$$(X\theta - y)^T(X\theta - y) = \begin{bmatrix} h_\theta\left(x^{(1)}\right) - y^{(1)} & \dots & h_\theta\left(x^{(m)}\right) - y^{(m)} \end{bmatrix} \begin{bmatrix} h_\theta\left(x^{(1)}\right) - y^{(1)} \\ \dots \\ h_\theta\left(x^{(i)}\right) - y^{(i)} \\ \dots \\ h_\theta\left(x^{(m)}\right) - y^{(m)} \end{bmatrix} =$$

$$= \left(h_\theta\left(x^{(1)}\right) - y^{(1)}\right)^2 + \dots + \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \dots + \left(h_\theta\left(x^{(m)}\right) - y^{(m)}\right)^2$$

Which is just the original summation:

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

Drawback:

1) Need to compute inverse matrix (Expensive and not always possible)
2) Need of invertible matrix (Only for square matrix with indipendent features)

Also for (2) we may have not enough training samples or too many features (It means not square matrix or even if it was it would still be a possibility of a linearity problem between the rows $x^{(i)}$ making not possible to calculate the inverse matrix)

Optimal parameters $\hat{\theta}$ (Proof)

We want to proof that we get the optimal $\hat{\theta}$ when $\nabla J(\theta) = X^T X\theta - X^T y = 0$ having that:

$$\hat{\theta} = (X^T X)^{-1} X^T y$$

With the same assumption of the previous proof:

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \dots \\ \frac{\partial J(\theta)}{\partial \theta_j} \\ \dots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

Where $\frac{\partial J(\theta)}{\partial \theta_k}$ was already defined as:

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

Which means:

$$\frac{\partial J(\theta)}{\partial \theta_k} = \begin{bmatrix} h_\theta\left(x^{(1)}\right) - y^{(1)} & \dots & h_\theta\left(x^{(i)}\right) - y^{(i)} & \dots & h_\theta\left(x^{(m)}\right) - y^{(m)} \end{bmatrix} \begin{bmatrix} x^{(1)} \\ \dots \\ x^{(i)} \\ \dots \\ x^{(m)} \end{bmatrix}$$

$$\frac{\partial J(\theta)}{\partial \theta_k} = \left(X\hat{\theta} - y\right) X^T = X^T X\hat{\theta} - X^T y = 0$$

So by isolating the θ we get:

$$X^T X\hat{\theta} = X^T y \rightarrow \hat{\theta} = (X^T X)^{-1} X^T y$$

2.7 Probabilistic interpretation of linear regression

As we told, we are approximating the output $y^{(i)}$ with our hypotethis $h\left(x^{(i)}\right)$ which is a function of the input using parameters θs in order to "calibrate" our hypothesis ($h\left(x^{(i)}\right) = \theta^T x^{(i)}$).

Our output can be expressed in the following form:

$$y^{(i)} = h\left(x^{(i)}\right) + e^{(i)}$$

Where $e^{(i)}$ is the error between the real value $y^{(i)}$ and the predicted value $h(x)$ ($e^{(i)} = y^{(i)} - h\left(x^{(i)}\right)$)

By assuming independent samples we can model the error as an exponential random variable. With an assumed normal distribution for the variables we get the following probability:

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

Due to the fact that $e^{(i)} = y^{(i)} - h\left(x^{(i)}\right)$, the probability $p\left(e^{(i)}\right)$ is the same as considering how likely we will obtain an output $y^{(i)}$ given the parameters $x^{(i)}$ and θ as input.

$$p\left(y^{(i)} | x^{(i)}; \theta\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}}$$

Important: The closer the prediction gets to the actual value the bigger the value of this probability gets with maximum value when $y^{(i)} = \theta^T x^{(i)}$.

2.8 Likelihood function (LHF)

We want a function which explains how well a set of samples $(x^{(i)}, y^{(i)})$ with given θs reacts to the system (How well is our prediction compared to the actual output?). This particular function takes name of "**likelihood function**". Basically, we are now considering the problem as an optimization one by seeking the best set or parameters θs which results in the best fit for the joint probability of our samples. Earlier we defined the form of the probability for a generic sample:

$$p\left(y^{(i)}|x^{(i)};\theta\right) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{\left(y^{(i)} - \theta^T x^{(i)}\right)^2}{2\sigma^2}}$$

What we want now is to define the "**likelihood function**" as the joint probability of all the samples (Remember that by using X and y we refer to respectively all the $x^{(i)}$ (Inputs) and all the output $y^{(i)}$):

$$L(\theta) = L(\theta; X; y) = p(y|X;\theta)$$

Since we assumed independence between samples our function $L(\theta)$ can be defined as:

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

2.8.1 LHF maximize parameters

As we told at the end of paragraph **3.7**, since this is the joint probability of all our samples in order to get the best fit we need to find the maximum of the $L(\theta)$ in the following form:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, L(\theta)$$

In order to simplifying this formula, we can consider the logarithmic of this formula:

$$\theta = L(\theta) \qquad l(\theta) = \ln\left(L(\theta)\right)$$
$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, L(\theta) \rightarrow \max_{\theta} l\left(\hat{\theta}\right) = \underset{\theta}{\operatorname{argmax}} \ln\left(L(\theta)\right)$$

Note: In the slides he wrote $\log\left(L(\theta)\right)$ but it should be technically $\ln\left(L(\theta)\right)$ because in the following steps if it had been the case it would have showed up the term $\log(e)$ which obviously doesn't. (QED)
Note 2: We can use $\log(x)$ and searching for the $\operatorname{argmax}(\log(x))$ because it is a monotonous increasing function.

Why do we use logarithmic function

Thanks to the different properties of logarithms we can simplify our problem:

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

1) $\log(AB) = \log(A) + \log(B)$

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

Applying once more this property:

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

After some simplification we can came up with:

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

Optimal $l(\theta)$

We shifted the problem from optimizing $L(\theta)$ to optimizing $\ln(ln(\theta))$ which is entirely possible due to the nature of the two functions ($L(\theta)$ and $\ln(x)$) and so the problem is described as it follows:

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

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

Since the first term is a constant, it doesn't contribute to the problem of maximizing the function:

$$\max_{\theta} l(\theta) = \max_{\theta} -\frac{1}{2\sigma^2}\sum_{i=1}^{m}\left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

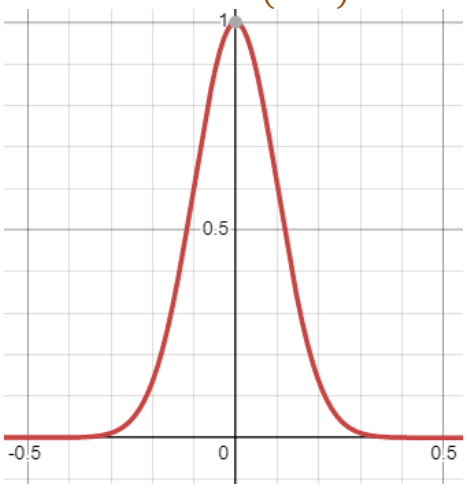
We can also remove this second constant due to the fact that is not useful for our analysis:

$$\max_{\theta} -\frac{1}{2\sigma^2}\sum_{i=1}^{m}\left(y^{(i)} - \theta^T x^{(i)}\right)^2 = \max_{\theta} -\frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

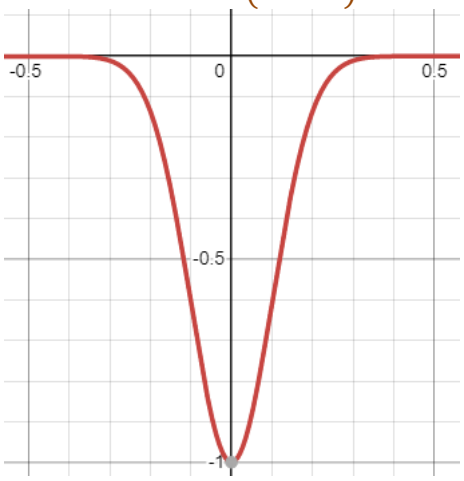
Due to the negative sign the problem would be the same as a minimizing and changing signs:

$$\max_{\theta} -\frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)} - \theta^T x^{(i)}\right)^2 = \min_{\theta} \frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)} - \theta^T x^{(i)}\right)^2$$

Maximize $\left(e^{-x^2}\right)$



Minimize $\left(-e^{-x^2}\right)$



Note: As we can see from those two graphs, the function e^{-x} has a maximum point at $x = 0$ but when it is gets negated it becomes a minimum point always at $x = 0$

In the end we get the final form of our problem:

$$\max_{\theta} l(\theta) = \min_{\theta} \frac{1}{2}\sum_{i=1}^{m}\left(y^{(i)} - \theta^T x^{(i)}\right)^2 = \min_{\theta} \frac{1}{2}\sum_{i=1}^{m}\left(\theta^T x^{(i)} - y^{(i)}\right)^2$$

Note: the last step can be done since it's all elevated to the second power, by changing the sign the result doesn't change.

2.8.2 Comparing likelihood and Mean Square Error

As we discussed in paragraph (**2.5.1**), we have already used the formula for **MSE**:

$$[(1)] \rightarrow \min\left(L(\theta)\right) = \min\left(\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2\right)$$

In this case, using our assumptions we got is:

$$[(2)] \rightarrow \min\left(L(\theta)\right) = \min\left(\frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2\right)$$

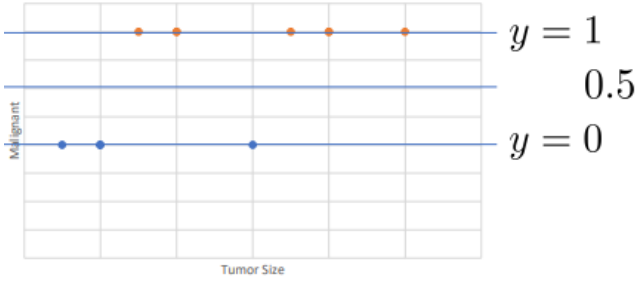
Remember: $h_\theta\left(x^{(i)}\right) = \theta^T x^{(i)}$ assuming $x_0^{(i)} = 1$ for all the samples (*bias*).

Conclusion: virtually identical problems but the regularization constant that has no influence in the minimization problem (At least I think m is the regularization constant).

Solving the Least Square Problem (1) means finding the best parameters θ that given the input x will produce the output y, basically the definition of the problem (2).

2.9 *Classification problem*

The classification problem is a particular problem which will give us a class as output (Instead of a real number approximation).



Example: given the input $x^{(i)}$ we would like to predict if a tumor is benign or malignant as an output $y^{(i)} \in \{0, 1\}$. Our new hypothesis will be classified as 1 for value of $h_\theta(x) \geq 0.5$ and every 0 for $h_\theta(x) \leq 0.5$
(If $h_\theta(x) = 0.5$ just do a coin toss).

As we can see it is drawn a middle line for the $h_\theta(x) = 0.5$ because that line will be our **threshold classifier**.

2.9.1 Logistic regression model

We are using a Sigmoid (logistic) function as hypothesis which is expressed as:

$$h_\theta(x) = g(\theta^T x) = g(z)$$

We define our $g(z)$ as:

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

So in the end we have our hypothesis:

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

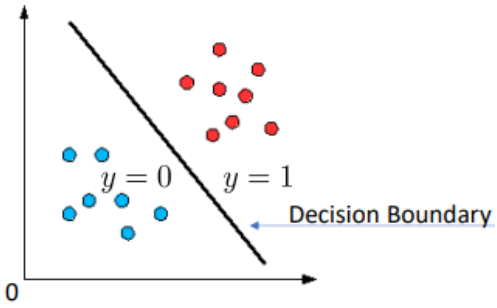
The minimum and maximum of this function are 0 and 1 at $-\infty$ and $+\infty$ and every input will produce a bounded output $y^{(i)} \in [0,1]$.

2.9.2 Decision boundary intuition

Given our prediction

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

We would like to predict our output $y^{(i)}$ that will get 0 on one side of the boundary and 1 on the other 1 (Simple enough?)



2.9.3 Probabilistic Interpretation

Our prediction $h_\theta(x)$ is basically a probability of that input of being a specific category.
We can express this probability as $\boldsymbol{h_\theta(x) = p(y = 1|x;\theta)}$ (Or $y = 0$ since they are 2 categories). The condition probability of y being 1 (or 0 if we like) given x and θ.
Since the problem has just 2 categories we can say that $p(y = 1|x;\theta) + p(y = 0|x;\theta) = 1$
So, we need to define just one of these two probabilities:

$$p(y = 0|x;\theta) = 1 - p(y = 1|x;\theta)$$

2.9.4 Cost function

We would like to maximize the probability of the output of being 1 given that specific input and parameters and we can do it so by using the "**Likelihood function**" previously shown some paragraphs ago:

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

As defined earlier, we can write the formulas for the two probabilities as:

$$\left\{\begin{matrix} p\left(y^{(i)} = 1\middle|x^{(i)};\theta\right) = h_\theta\left(x^{(i)}\right) \\ p\left(y^{(i)} = 0\middle|x^{(i)};\theta\right) = 1 - h_\theta\left(x^{(i)}\right) \end{matrix}\right\}$$

And in the end we can express the final probability as:

$$p\left(y^{(i)}\middle|x^{(i)};\theta\right) = h_\theta\left(x^{(i)}\right)^{y^{(i)}} * \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}}$$

Note: since we assume that $y^{(i)}$ can be just 0 or 1 we can substitute its value in the previous formula:

$$p\left(y^{(i)} = 1\middle|x^{(i)};\theta\right) = h_\theta\left(x^{(i)}\right)^{1} * \boldsymbol{\left(1 - h_\theta\left(x^{(i)}\right)\right)^{0}} = h_\theta\left(x^{(i)}\right)$$

Or on the other hand:

$$p\left(y^{(i)} = 0\middle|x^{(i)};\theta\right) = \boldsymbol{h_\theta\left(x^{(i)}\right)^{0}} * \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1} = 1 - h_\theta\left(x^{(i)}\right)$$

In the end we get the same result.
Back to the likelihood: if we substitute the formula for the condition probability we get:

$$L(\theta) = \prod_{i=1}^{m} h_\theta\left(x^{(i)}\right)^{y^{(i)}} \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}}$$

We can also use the logarithm trying to simplify the $L(\theta)$ as we did before:

$$l(\theta) = \ln\left(\prod_{i=1}^{m} h_\theta\left(x^{(i)}\right)^{y^{(i)}} \left(1 - h_\theta\left(x^{(i)}\right)\right)^{1-y^{(i)}}\right) =$$

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

Cross Entropy Error Function

We can also shift from maximization to minimization and so we define:

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

So, in the end we can set the whole problem as a parameter fitting problem $\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\left(J(\theta)\right)$

2.9.5 Errors

Since the logistic function cannot be easily studied analytically we can get an intuition using the error contribution plot (**Da capire se bisogna aggiungere qualcosa**):

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

Case 1: $y^{(i)} = 1 \rightarrow e^{(i)} = -\ln\left(h_\theta\left(x^{(i)}\right)\right)$

$h_\theta\left(x^{(i)}\right) = 0$ means $e^{(i)} \rightarrow \infty$

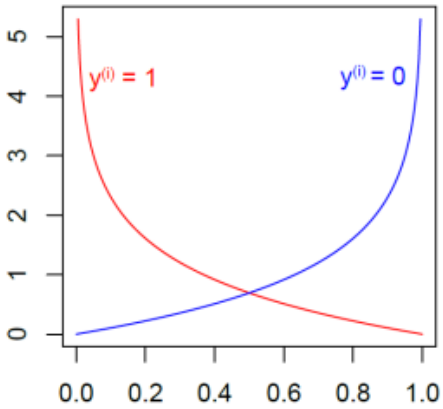
$h_\theta\left(x^{(i)}\right) = 1$ means $e^{(i)} \rightarrow 0$

Case 2: $y^{(i)} = 0 \rightarrow e^{(i)} = -\ln\left(1 - h_\theta\left(x^{(i)}\right)\right)$

$h_\theta\left(x^{(i)}\right) = 0$ means $e^{(i)} \rightarrow 0$

$h_\theta\left(x^{(i)}\right) = 1$ means $e^{(i)} \rightarrow \infty$

Analysis: The closer $h_\theta\left(x^{(i)}\right)$ aims through a class ($y = 1$ or $y = 0$) the less the error $e^{(i)}$ became to that class increasing the error to the other one.



2.9.6 Logistic regression Gradient Descent

We are aiming to $\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\left(J(\theta)\right)$ and we can use the **GD** in order to reach a minimum

As for the

$$\theta_k := \theta_k - \alpha \frac{\partial J(\theta)}{\partial \theta_k}$$

Basically a lot of mathematics starting from (watch the slide *1_intro_to_ml_linear_regression.pdf* for the full explanation)**:**

1) $g'(z) = g(z)\left(1 - g(z)\right)z'$

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

3) $\frac{\partial(\theta^T x)}{\partial \theta_k} = x_k$

4) $h'_\theta\left(x^{(i)}\right) = h_\theta\left(x^{(i)}\right)\left(1 - h_\theta\left(x^{(i)}\right)\right)x_k$

In order to get:

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

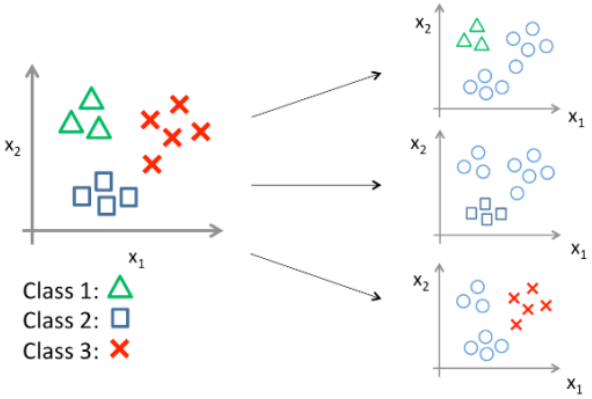
2.10 Multi-class classification

Basically every multi-class can be reconducted to a binary classification (2 class).

Example: let's say that we want to classify triangles, squares, and crosses.

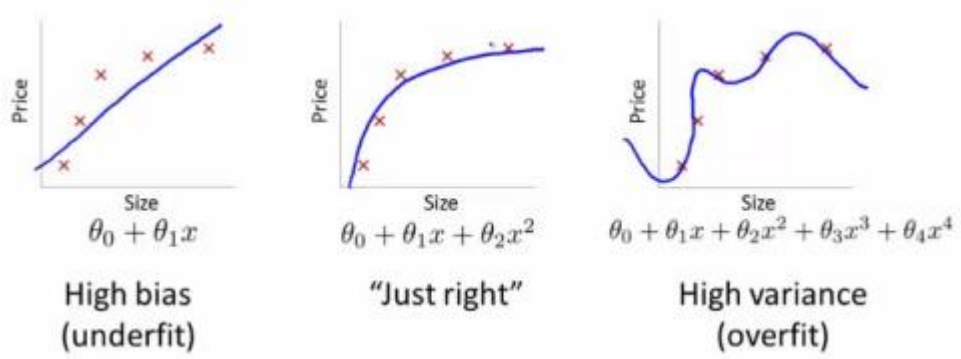
1) IF you are triangle or not
2) IF you are a square or not
3) IF you are a cross or not

Each case can be analyzed as a binary classification.



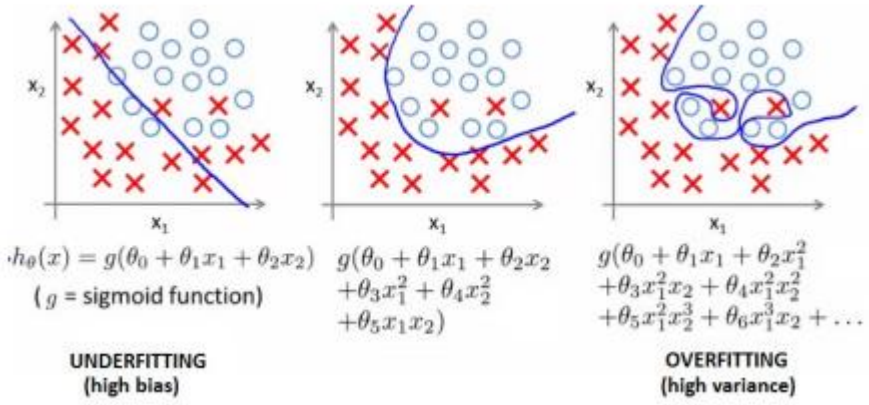
3.1.1.1.1 3_bias_and_variance.pdf

3.2 Fitting regression (Problem)



This kind of graphs were created with a polynomial regression but as we can see these graphs show different behaviors depending on the grade.
The **left** (**LX**) one has a quite good approximation of the problem (with also a decent error)
The **right** (**RX**) one has apparently the best solution since the prediction passes through every point (Error on those point is 0)
The **center** (**CX**) one is actually the best one because it **describe** generally the behavior of the samples without being dependent on them like the right graph.
What is happening on the right graph is that our prediction is no more learning but instead is **describing** the training points so this graph will probably give the **worse** error for new samples.
Note: we define this problem as **overfitting** or **underfitting**.

As for classification



The same problem may happen for classification type problems.

3.2.1 Overfitting and underfitting

Overfitting

The model performs well on the training data but not on unseen data because 1) is too complex and it's unable to generalize the behavior of the model and 2) it is memorizing the training data instead of "learning" from them by generalizing the problem

Underfitting

The model is too simple, and it can't perform well neither training samples nor new data

3.3 *Bias and Variance*

Bias

Systematic deviation of the estimator which represents the mean of the error of the predicted values

$$Bias\big(h(X), f(X)\big) = E[h(x)] - f(X)$$

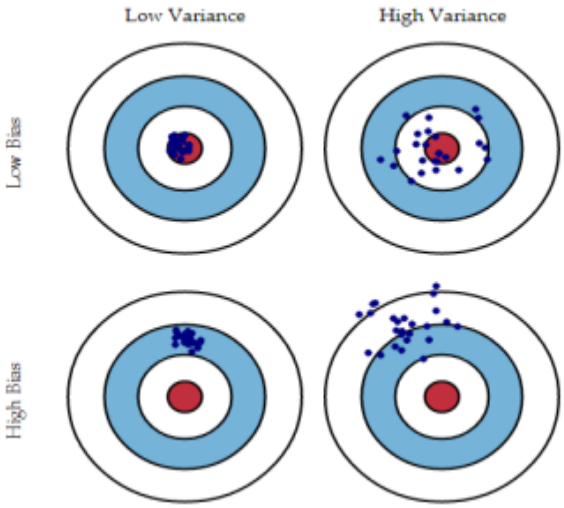
How much the real value differs from the hypothesis basically.

Variance

It represents the mean value of the variation of predicted values with respect to the mean of the predicted values (Basically a self-analysis)

$$Var\big(h(X)\big) = E\big[h(X) - E[h(x)]\big]^2$$

Main difference: Bias is related to the **actual value**, variance to the **hypothesis itself**.



As those graphs show, depending on the values of variance and bias we will end up with different models for our system. The best one is the problem with low variance and low bias which means a model able to generalize new data based on training samples (Which are not simply memorized).
Low (L) Bias / L Variance: perfectly cooked
High (H) Bias / L Variance: data compacted but different from what they should be
L Bias / H Variance: data are scattered means rough approximation
H Bias / H Variance: Rough approximation with scattered prediction

3.3.1 Bias and Variance trade-off

First, we define the expected value of a random variable X:

$$E[X] = \sum_{i=1}^{m} p_i x_i$$

If we let X be the input and Y be the output we want to model:

$$Y = f(X) + e$$

Assuming our hypothesis $h(x)$ able to approximate $f(X)$ then we will get a Mean Square Error (**MSE**) (or expected square error):

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

First, we assume an infinite number of **datasets** D_i which rappresents all the possible combination of the training samples that could feed our model (With all possible subsets of the corresponding random variable).
- We train our model and fix it with different D_i and we will get an infinite amount of hypothesis $h^{(D_i)}(x)$
- Each training sample is in the form $< x^{(i)}, y^{(i)} >$ where $y^{(i)} = f\big(x^{(i)}\big) + e^{(i)}$ and $e^{(i)}$ is our gaussian error characterized with $\mu = 0$ (mean value) and σ_e (variance).

- Each hypothesis $h^{(D_i)}(x)$ will be affected by an error in predicting values with respect to the ground truth (Actual values). It can be model with **MSE** as:

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

We want to estimate the expectation of error for all the possible D_i by computing **MSE** on each hypothesis and an overall mean at the end.
Since we can't perform over all the infinite dataset we opt to compute the expected value of the mean as the expectation of the MSE over all the datasets (**Generalization Error**):

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

Due to the linearity of $E[x]$ we can swap:

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

We define the best estimator of $f\left(x^{(i)}\right)$ by feeding each hypothesis with the same training sample $x^{(i)}$ and then we compute the mean of all the values:

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

We can now perform a sum zero operation:

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

We define A as the first term and B as the second one inside the $E_D[x]$:

$$E_D[(A+B)^2] = E_D[A^2 + B^2 + 2AB]$$

Due to the linearity again:

$$E_D[A^2 + B^2 + 2AB] = E_D[A^2] + E_D[B^2] + E_D[2AB]$$

If we focus separately on the terms:

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

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

The second term with B doesn't depend on the variable D so by definition $E_D[C] = C$.
For the third one:

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

Analyzing just that sub-term:

$$E_D\left[h^{(D_i)}(x) - \bar{h}\left(x^{(i)}\right)\right] = E_D\left[h^{(D_i)}(x)\right] - E_D\left[\bar{h}\left(x^{(i)}\right)\right] = \bar{h}\left(x^{(i)}\right) - \bar{h}\left(x^{(i)}\right) = 0$$

It all follow due to linearity, so we can say that:

$$E_D\left[2\left(h^{(D_i)}(x) - \bar{h}\left(x^{(i)}\right)\right)\left(\bar{h}\left(x^{(i)}\right) - f\left(x^{(i)}\right)\right)\right] = 0$$

So in the end we get:

$$MSE\left(h^{(D_i)}(x)\right) = Var\left(h^{(D)}(x)\right) + Bias^2\left(h^{(D)}\left(x^{(i)}\right), f\left(x^{(i)}\right)\right)$$

3.3.2 Formula analysis (Manca una cosa alla fine)

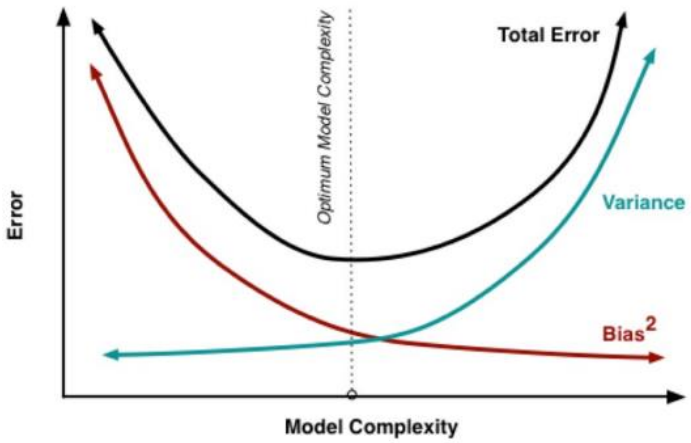
As we can see, in order to reduce the MSE we should reduce $Bias$ and Var but in reality we can't.

Bias

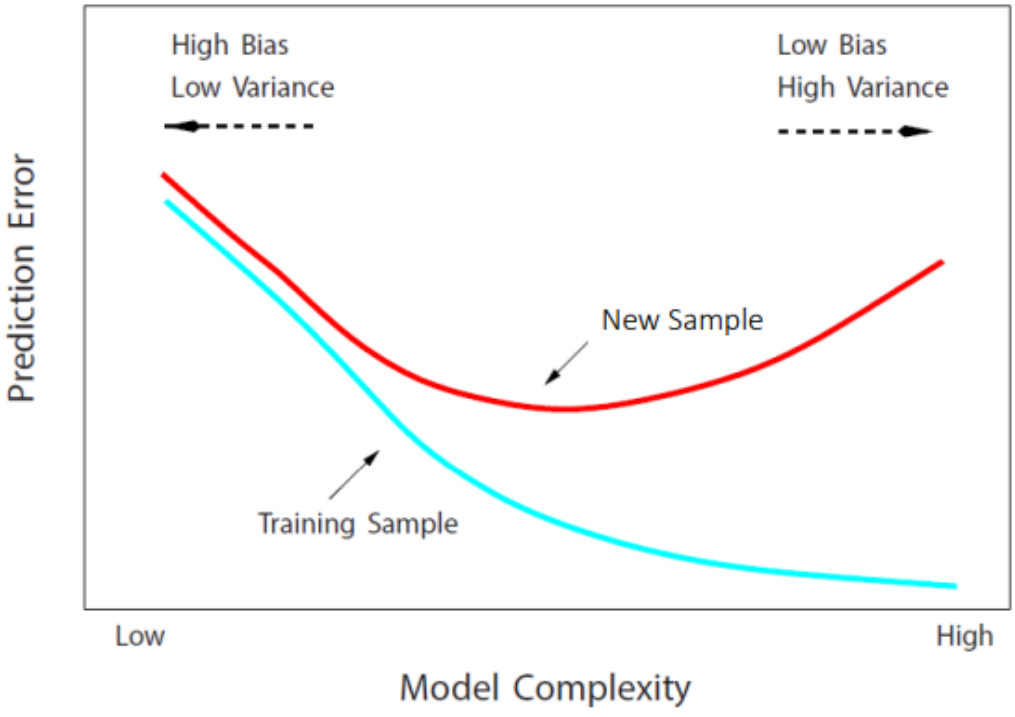
It represent how much the estimation is close to the ground truth (basically $y^{(i)}$) but high $Bias$ produce a too simple model unable to predict whatever dataset in input.

Variance

It represent how much the hypothesis calculated differs from the best estimation. A high variance means that every dataset generate very difficult and different hypothesis which are too much data-dependent from the training samples (So unable to generalize for new data)



Graph analysis: as we can see with a rough approximation, we need to trade-off between them.



Vorrei capire il discorso che ha fatto su questa slide

3.4 Regularization

When we are modelling a hypothesis, and we want to choose the "**right**" grade t for the polynomial regression we will need to find a way of doing this.

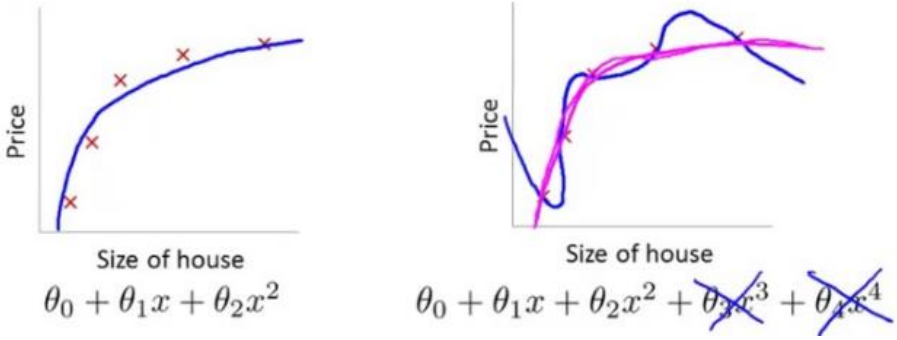
Reducing the feature

Manually select which features to keep
Model selection algorithm

Regularization

Keep all the features, but reduce the magnitude of the parameters
Advantage: works well with a lot of features since each contributes a bit to the prediction

3.4.1 Regularization Intuition



As we can see from those two graphs, the LX one is the one that we aim to obtain since the grade ($t = 2$) is the best grade in order to avoid **underfitting** (Enough complex to predict the data) and **overfitting** (Not too complex to get large error on new data).
We can say that the LX graph can be obtained by the RX graph (which is obviously in an **overfitting** situation) by reducing the grade of the polynomial regression ($t = 2$ from 4)
Consideration (Overfitting): usually during overfitting the coefficients θs beyond a certain point will assume very high values so that they will **influence** the function more due to their value.
Solution: during the previous chapters we considered a function to minimize the cost that didn't consider the "**weight**" of the coefficients θs so what we want is to minimize the following:

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

We also use the notation $|\theta|_2^2$ (L2 norm without considering θ_0) in order to represent $\sum_{j=1}^{n}\theta_j^2$

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

The term λ is called **regularization parameter**.
Note: θ_0 is excluded because there is no feature that multiplies it, so it doesn't count

3.4.2 Regularization parameter analysis

Since now we can define how much the λ weight on the model, we can assume different scenarios:

Lamba too high → Underfitting

Since λ is high in order to minimize $J(\theta)$ we have to reduce a lot the grade of the polynomial regression.

Lamba too low → Overfitting

Since λ is low, $|\theta|_2^2$ will increase its value, it means increasing the grade of the polynomial regression.

Lamba well defined → Fitting

Since λ is properly weight with the contribution of the hypothesis $h_\theta\left(x^{(i)}\right)$ in the expression of $J(\theta)$ it will result in a balance between the two.

Capitolo 4 Lezione 2022-10-13

4.1 Regularization

4.1.1 Regularization for linear regression

If we want to apply the gradient descent:

$$\begin{cases} \theta_0 \coloneqq \theta_0 - \alpha \frac{1}{m} \sum_{j=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)} & j = 0 \\ \theta_j \coloneqq \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)} + \frac{\lambda}{m} \theta_j\right] & j = 1, \dots, n \end{cases}$$

Remember that θ_0 doesn't count when minimizing $\lambda|\theta|_2^2$ due to the fact that it is not multiplied by a feature. If we focalize on the θ_j formula we can derive:

$$\theta_j \coloneqq \theta_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$$

Focusing on the first term, we want it to be as close as possible to 1 because by doing that we will not change che behaviour of the **GR** too much and we do that by putting the ratio λ/m close to 0.

4.1.2 Regularization for logistic regression

Basically we just need to adapt the formula of the cost function as we have already done:

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

Obtaining a behavior similar to the previous one

$$\begin{cases} \theta_0 \coloneqq \theta_0 - \alpha \frac{1}{m} \sum_{j=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_0^{(i)} & j = 0 \\ \theta_j \coloneqq \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)} + \frac{\lambda}{m} \theta_j\right] & j = 1, \dots, n \end{cases}$$

4.1.3 Regularization with L1-norm

If we remember the formula for the L1 norm:

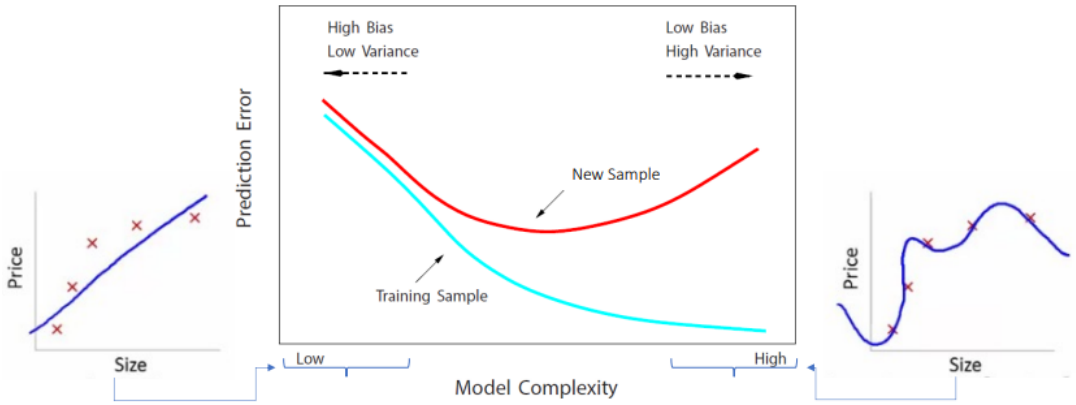
$$l^1 = \sum_{j=1}^{n} |\theta_j|$$

Due to the fact that is not possible to compute the partial derivative for the **GR** we can't use directly so in this case it is better to use l^2 norm.

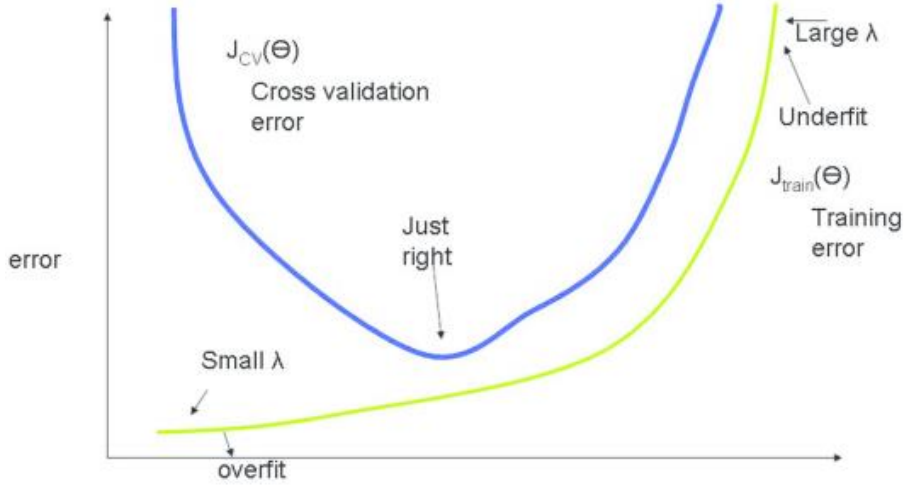
When is this norm useful? **Faster feature selection**

Hypothetically speaking, it is not impossible to have thousands and thousands of features, so it is useful to have an algorithm which is able to detect the most relevant ones.
Basically there is a theorem which explains that using L1 norm it helps us faster in an effective feature selection compared to the L2 norm. Since the prof. doesn't go into details too much, here it is the **slide**.
To remember: L1 in this case of selection is faster that L2 in order to get the most relevant features

4.1.4 Regularization vs Bias/Variance



What happened earlier is that depending on the values of the $Bias$ and Var, which we proved to be correlated in the MSE error, we could get events like **underfitting** and **overfitting** situation if we didn't consider those scenarios.



What we learned is that by using our new cost functions $J(\theta)$ we can train our models to consider the weight of the coefficients θs directly correlated to a variable λ called **regularization parameter** that must be set accurately in order to get as much close as possible to the best approximation.

4.1.4.1.1 4_how_to_build_a_ML_system.pdf

4.2 How to build a ML system

Now we know the tools in order to create a ML system because we know algorithms for **regression** and **classification** models associated with techniques such as **regularization** in order to prevent **overfitting** and **underfitting** situations.
First of all, we define three steps that we will follow during the realization of the system:

Representation

We first need to identify the hypothesis of the learner (Learner since our model will "learn") space and decide the features to use to represent data. During this selection phase (which is very important) we can use different tools in order to identify the most relevant features to use.

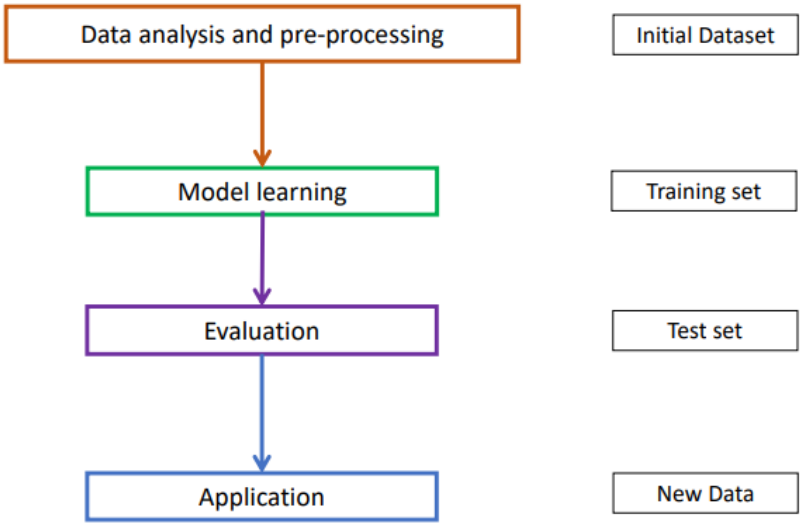
Optimization

Basically consist of choosing the method to train the learner

Evaluation

Maybe the most fundamental step of all because consist of finding bad and good learners (After we create out model is important to understand If it is capable of generalize the data)

4.3 ML system life cycle (First review)



Data analysis and pre-processing: we first start from the data analysis and a **pre-processing phase**, it is important to clean, adapt and being in control of the data we are using for our model.
Golden rule: "**GIGO**" Garbage IN Garbage OUT. It means if we put unusable, useless, or corrupted input we will get a bad output as result **MEANING** pre-processing is, in fact, important even if it takes a lot of time and it is boring (I'm sorry).
Model learning: this phase is the one we have already talked about, and it is the part where the model is actually trained by the **training samples** $(x^{(i)}, y^{(i)})$ and we estimate the error using the **cost function** $J(\theta)$ (Cost function considering **overfitting** and **underfitting** situation, so with λ).
Evaluation: it is a phase where we test our model with a **test set**. Since they are not training samples they are basically new unseen data for the model, so we are able to estimate a new error in order to understand how well the model is responding.
Application: and of course, we want our model to be actually used by putting new and unseen data.

4.4 Pre-processing

During this phase we are expected to clean our data enough because the end is to have the clearer data to insert inside the model. We acknowledge that real word data are "**dirty**" because they are incomplete (missing attributes or missing interesting attributes) and they can be also **inaccurate** (wrong reading of data like sensor reading, data were being observer partially, so they are no accurate etc.)

What to do list

Remove duplicates means the same exact information can be excluded from our dataset
Remove outliers: abnormal behavior which is usually just a wrong reading etc.
Discretize: transformation of the data (Usually from continuous to discrete)
Normalization and **re-scaling**: for a better representation of data
Creating new attributes: which means create new attributes from previous attributes and/or values in order to increase the information that we can use to represent data (Usually this helps for **classification problems**)

4.4.1 Outlier removal (boxplot)

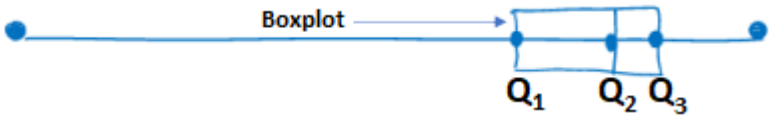
We can use information such as **quartiles**, **deciles** and **percentile** which will divide the data into equals set. What changes is that:

Quartiles	4 parts	**Deciles**	10 parts	**Percentiles**	100 parts

Using the **quartiles** method we divide 3 dividers (Q_1, Q_2, Q_3) defined such that:
1) Q_2 is defined as the meridian
2) Q_1 is defined as the meridian between 0 and Q_2
3) Q_3 is the mediant between Q_2 and the **END**.
So we have respectfully 4 distinct equal part:

0	Samples	$\boldsymbol{Q_1}$	Samples	$\boldsymbol{Q_2}$	Samples	$\boldsymbol{Q_3}$	Samples	**End**

Boxplot: part between Q_1 and Q_3



Interquartile: defined as $IQR = Q_3 - Q_1$
We consider **outliers** all the data which are:
1) Less than $Q_1 - IQR * 1.5$ and 2) more than $Q_3 + IQR * 1.5$

4.4.2 Min-max and Z-score normalization

Min-Max normalization

$$x' = \frac{x - min}{max - min}(b - a) + a \;|\rightarrow|_{\substack{x' \in [a,b] \\ x \in [min,max]}}$$

If we proceed by removing the outliers, like we discussed in the previous paragraph, this normalization will work fine **but** it will be still a little trouble to values external to $[a, b]$.

Z-score

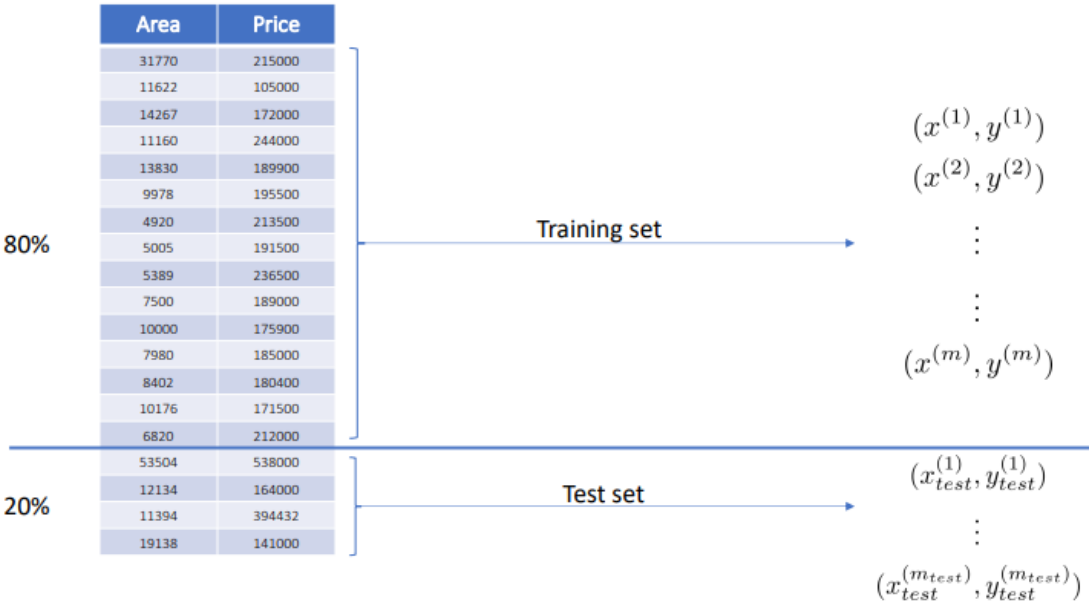
$$x' = \frac{x - mean(x)}{std(x)}$$

(Ha detto qualcosa a riguardo)

4.4.3 Features selection

We need to select the relevant feature for the learning task maybe because we are interested in reducing the useless features (Like name and surname or **redundant** features) or because we need to reduce the **dimensionality** of the data without reducing the amount of information (As a result we may be able to compact the data points due to the reduced dimensionality and keeping the information).
Note: we have tools that are able to find/detect the importance of features.
Filter: measure the importance of each feature to discriminate across the classes
Wrappers: it is an iterative method that aims to find a good subset of features using a subset.
Dimensionality reduction: PCA, SVD

4.5 Hypothesis evaluation



Since we have just one dataset we have to divide it into two parts: **training set** and **test set** and we do this because we will see the actual performance of our model.
Note: since real data are **dirty** you still want to split the data set after cleaning (So you are in control of the data), but it may be useful sometime to evaluate the model on dirty data.

4.5.1 Linear regression and Logistic regression

Linear regression

For linear regression we use the training samples for the parameters θs. We use a new formula for the error:

$$J_{test}(\theta)\frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$$

Because the $J(\theta)$ with the **regularization** is something specifically for the **training set** and we expect $J(\theta)$ to be small on training samples but for the **test set** is different because is a particular set of data which the system doesn't know so the cost function $J_{test}(\theta)$ is an **actual indicator**.
Note: The test set is created by choosing **randomly** or **selected** 20% from the dataset, the other 80% is for training.

Logistic regression

$$J_{test}(\theta)=-\frac{1}{m_{test}}\sum_{i=1}^{m_{test}}y^{(i)}\ln\left(h_\theta\left(x^{(i)}\right)\right)+\left(1-y^{(i)}\right)\ln\left(1-h_\theta\left(x^{(i)}\right)\right)$$

For classification we may have problems: the 20% is composed by the same category so **be careful** (Some tricks are needed at same point)

4.5.2 Optimal hypothesis from same training set and test set

Since we studied the **overfitting** and **underfitting** scenarios, we want a method in order to evaluate the best hypothesis among other obtained by using the same **training set** and **test set** but with different grade of the polynomial regression.

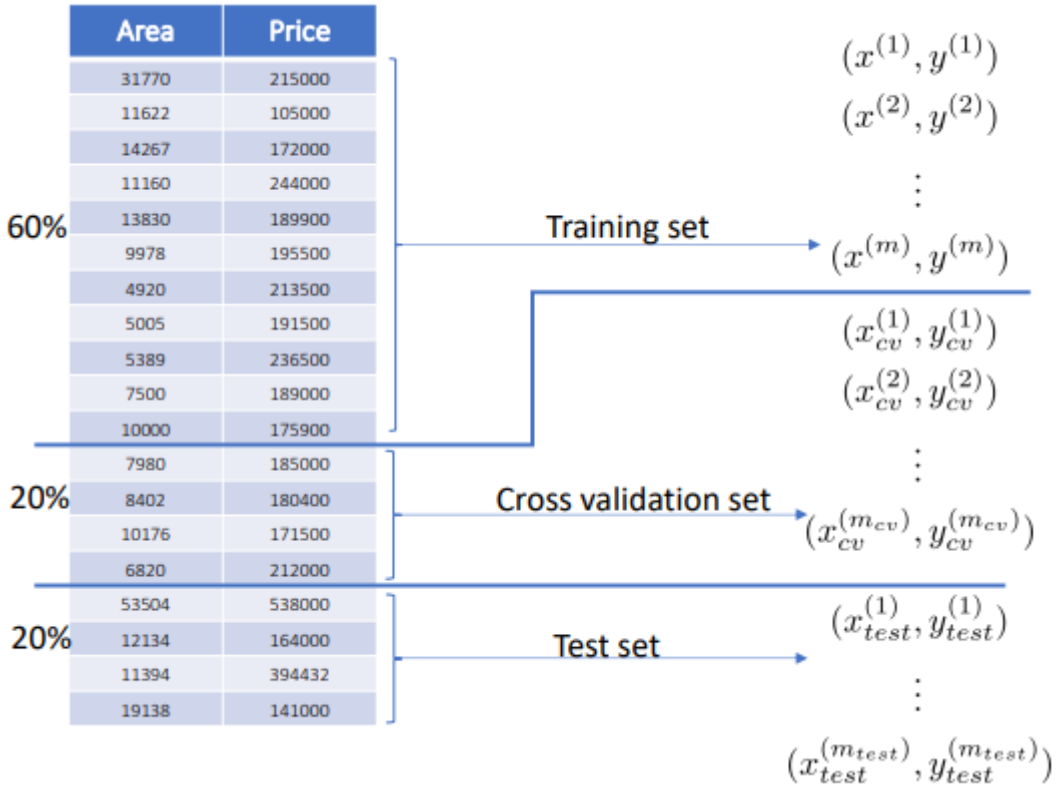
Example

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

We have 10 different polynomial with have all different grade (Starting from 1 to 10) that are obtained from the same dataset, same training set and test set.

Hypothetically, we image one test cost function to get the lowest $J_{test}(\theta^{(k)})$ **but we are not sure if even by doing this we will be able to get the best hypothesis from a dataset**.

4.5.3 Cross-validation phase



According to the previous **example**, we need a new way of evaluating the best hypothesis and we do this by adding a **cross validation phase**. Basically we create partitions of 60 training, 20 cross-validation, 20 test **sets**. The **test set** is the fundamental one because it will be used at the end.

How to evaluate the best hypothesis

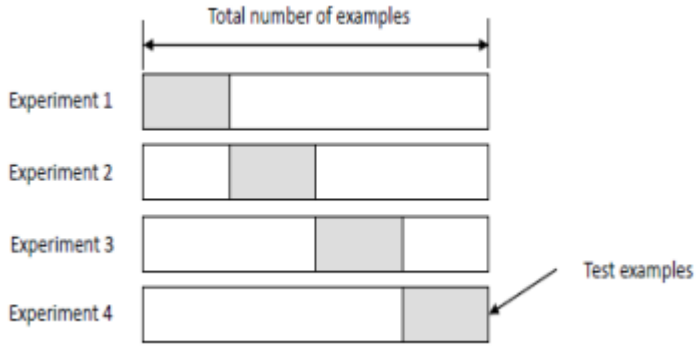
Goal: we aim to get the best performance on the **validation set**, instead of the training one.
1) We train the model as we usually do with the **training set**, we define our cost function $J(\theta)$.
2) We choose the best hypothesis taken with the same training set and cross validations set $J_{cv}(\theta)$
3) We create a new model using training and cross validation samples as training set.
4) We evaluate the performance of the model using the test set $J_{test}(\theta)$.

4.6 K-folds cross validation and hold-out cross validation

Basically, what we explain in **5.5.3** is the **Hold-out**, which uses the same partition in order to get the best hypothesis but by doing this we are not sure if that is the best possible partition to use in order to get the best model from that dataset. In order to generalize and optimize the way of searching for the best model we can use the **k-fold**.
Hold-out drawbacks: If for example in our partition 60 20 20 we have **divergent samples** inside the **test set** only (Not standard ones) it means that they ARE NOT inside the training set, and this means that if the model is not trained for a particular situation he will not be able to generalize that situation.
Solution: **Stratification** means that if there are 99% standard samples and 1% divergent ones we will have the same percentages inside the training set. **But** there are other problems like not considering enough pattern inside the dataset in order to evaluate the best hypothesis.

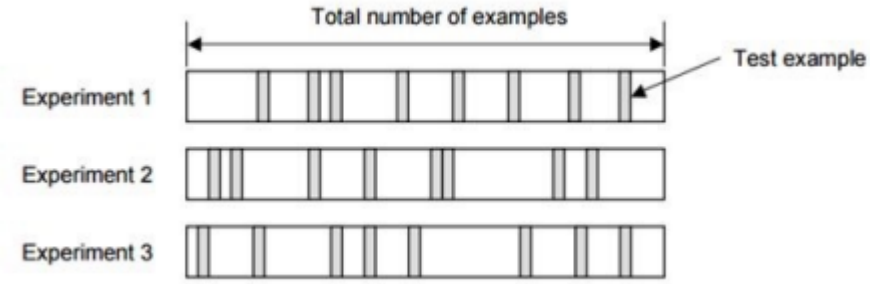
K-folds cross validation

Every dataset is randomly partitioned and then we generate different "**best hypothesis**" (We say "experiment" for saying that we take a different partition)
Advantage: by doing this we are considering more pattern inside our original dataset in order get closer to the best hypothesis that we can get from that particular set.



In the end: after taking the best hypothesis for every partition we evaluate the best one by computing a simple **average error** of all samples on every hypothesis and we get the minimum one.

4.6.1 Random subsampling



Basically we expect an enormous amount of data from particular ML system so in order to decide the test set we decide specific % (Like 1%) and we randomly sample that % from the dataset as **test set** (In principle is smaller and is it like 1% of the entire dataset) and we do this step n time (Like 10 times).
So after this step we use the **k-fold cross validation** with smaller test set n times (10 times).

4.6.2 Error estimate on k-folds

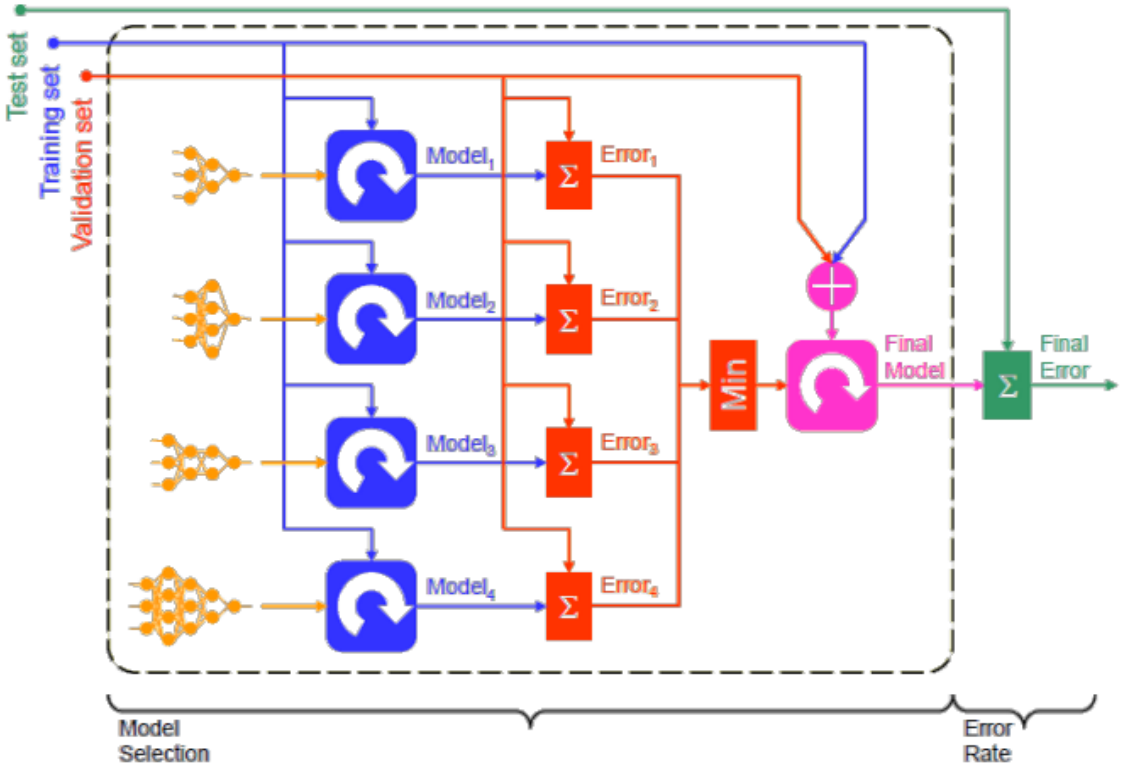
Case i	Train on					Test on	Error
Case1		F2	F3	F4	F5	F1	1.5
Case2	F1		F3	F4	F5	F2	0.5
Case3	F1	F2		F4	F5	F3	0.3
Case4	F1	F2	F3		F5	F4	0.9
Case5	F1	F2	F3	F4		F5	1.1

We simply partitioned our dataset and e perform evaluation for each fold case (**leave-one-out**)

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

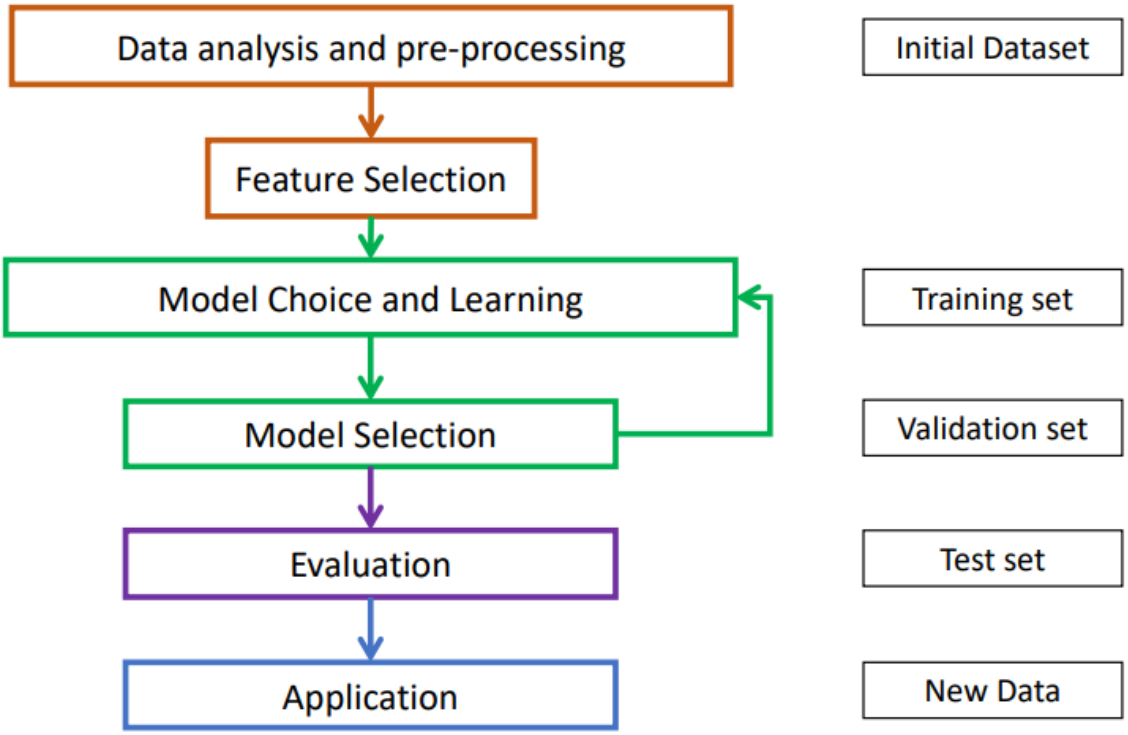
4.7 Cross validation overview

We simply partitioned our dataset and e perform evaluation for each fold case (**leave-one-out**)



A small recap of the steps we need to do:
1) **Training** → It trains our model by learning from the training set
2) **Validation** → We get the smaller error from different model taken by the same training set
3) **Merging** → After finding the best hypothesis from a particular training set we merge again the training set and the cross validation set and we compute a new model
4) **Evaluation** → we evaluate the model using the test set, we also check the performance.

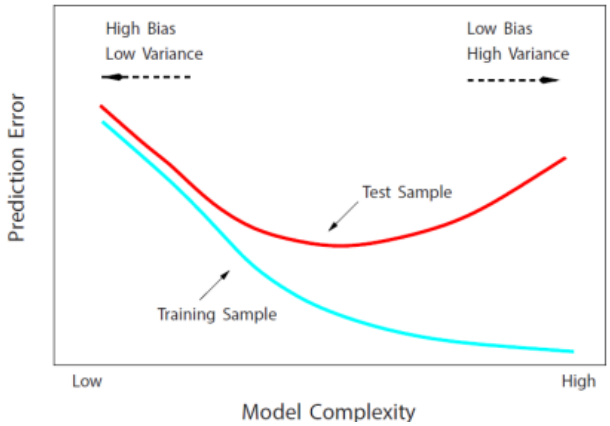
4.8 *ML system life cycle (Revisited review)*



Recal of the new actual model that we need to follow.

4.9 *Diagnostic and debugging*

After doing all the step in the previous model (**ML system life cycle**) we still get an unusable model because it doesn't work. We need to describe techniques responsible to guide us in order to understand why a model doesn't work so we can learn how to improve it; this process is called **diagnostic**.



Now that we defined new cost functions $J_{cv}(\theta)$ and $J_{test}(\theta)$ we need to understand different scenarios:

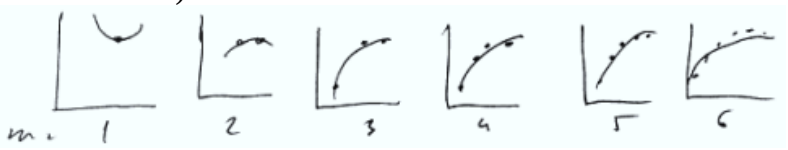
Bias (**underfitting**)
$J_{train}(\theta)$ high
$J_{cv}(\theta) \cong J_{test}(\theta)$

Variance (**Overfitting**)
$J_{train}(\theta)$ low
$J_{cv}(\theta) \gg J_{test}(\theta)$

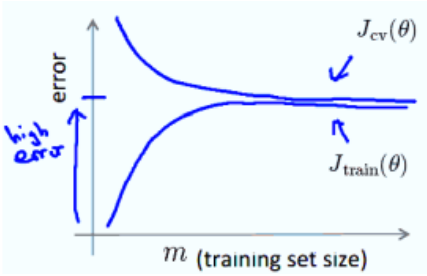
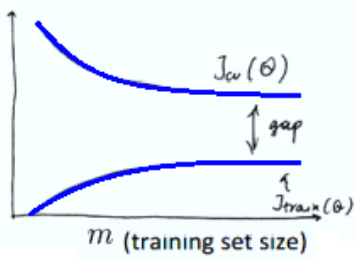
In order to evaluate the best hypothesis we can change λ inside $J_{train}(\theta)$ (Depending on **overfit** or **underfit**) or we can use the method of the **learning curves**.

4.9.1 Learning curves

We plot the values of two functions (Respectevely $J_{train}(\theta)$ and $J_{cv}(\theta)$ depending on how much samples m we consider for our set)



We start from $m = 1$ sample and we compute the values of the two function, then $m = 2$ and so on until $m = n$ which is a sufficient large number. The result of this process will be the plot of the learning curves:

Scenario	
Underfitting	**Overfitting**
	
Characteristics	
The errors of $J_{train}(\theta)$ and $J_{cv}(\theta)$ are very high and very similar	$J_{train}(\theta)$ is small but $J_{cv}(\theta)$ is high with a large gap
Solutions	
Descrease λ Increase complexity of the model Decrease the amount of data	Increase λ Decrease complexity Increase the amount of data

4.10 ML system check how well they works

The problem changes depending on **regression** or **classification**:

Regression

We need to compute evaluation metrics for regression (Which are basically formulas of errors):

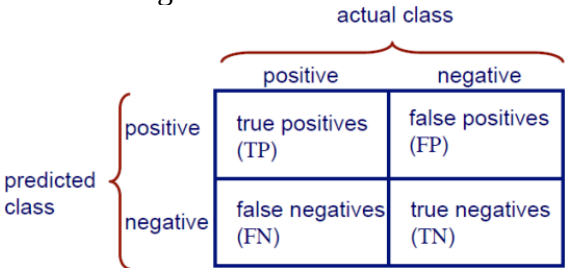
$$MAE = \frac{\sum_{i=1}^{m}|y_i^* - y_i|}{m} \qquad Mean\ Absolute\ Error$$

$$MSE = \frac{\sum_{i=1}^{m}(y_i^* - y_i)^2}{m} \qquad Mean\ Squared\ Error$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{m}(y_i^* - y_i)^2}{m}} \qquad Root\ Mean\ Squared\ Error$$

Classification

We need to generate our **confusion matrix**



Basically we associate all the pair of possible combinations (Actual class with predicted class) and then there are a lot of **parameters** that we can calculate.

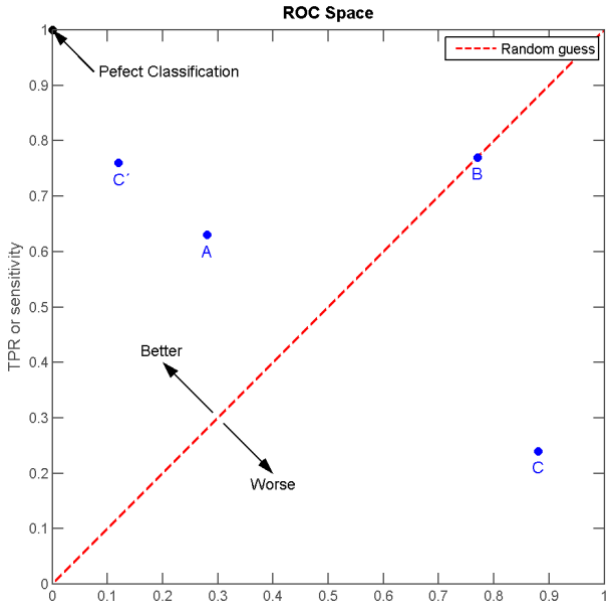
Capitolo 5 Lezione 2022-10-25 RIGUARDA

Lectures before this date were the **PYTHON** ones.

5.1.1.1.1 4_how_to_build_a_ML_system.pdf (Slide 48)

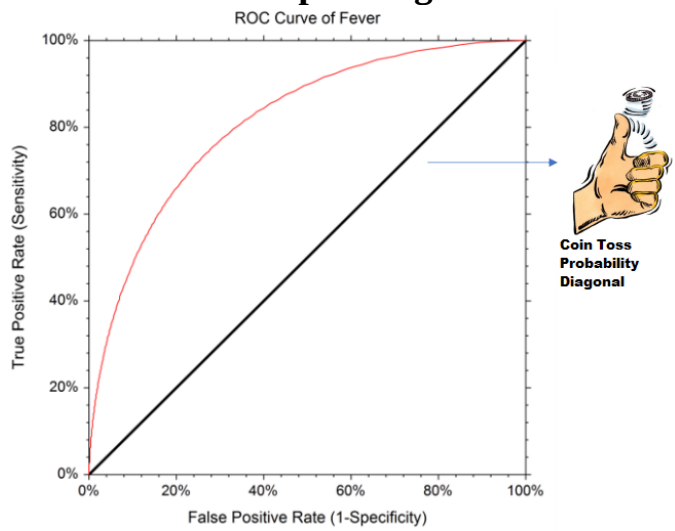
5.1 Most useful matrices (Confusion matrix analysis)

Remember: the metrics inside the table are just the most important ones but there are plenty of them which are useful, depending on what we are focused on. The metrics we will use the most are:
1) **Recall** (or **True Positive Rate**) which measure the amount of true positive predictions.
2) **False Positive Rate** which is similar but we are measuring **false positive** instead of true ones. .



What we usually do is to **plot** the values obtained by different hypothesis (A, B, C, C' in this case).
Optimal case: The best solution would be a **TPR** = 1 and a **FPR** = 0 (The top-left point for shorter) so we can assume that the nearest point to the optimal should be considered the best hypothesis (C').
Bad hypothesis: if you watch carefully there is a dashed diagonal which measure when **TRP=FRP** (In principle being on this line means the same probability of a coin toss) so being under this line (Like C) means that a coin toss is better than our hypothesis.
In general: we measure the distance of every point to the (0,1) optimal point in order to get an overall idea of how well a specific hypothesis is.

5.1.1 Receiver Operating Characteristic (ROC) curve



Briefly:

$$TPR = \frac{TP}{(TP + FN)}$$

And

$$FPR = \frac{FP}{(TN + FP)}$$

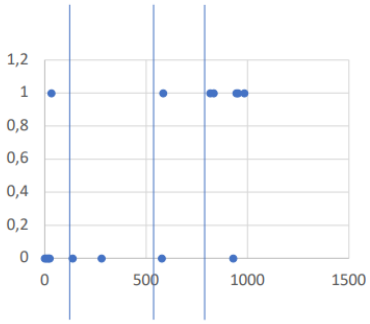
What we are graphing here is a rate of: "***How do performances change by changing the threshold of the binary estimator?***"
We start by graphing from threshold of 0 all al to 1 (Usually, a threshold of 0.5 is chosen).

Results

The point characterized by a given threshold which is the closest to the point $(0,1)$ is chosen.
$FPR = 0$ which means that among the **Negative values** there has been no misclassification
$TPR = 1$ which means that among the **Positive values** they have been classified correctly.
Also remember: T and F stands for $True$ and $False$, P and N stands for $Positive$ and $Negative$.
Example: $TP = True - Positive, FN = False - Negative$

ROC curve example

	Year	Observed event (1) or non-event(0)	Forecast Probability	T=0.1	T=0.5	T=0.8
Correct positive	1994	1	0.984	1	1	1
	1995	1	0.952	1	1	1
	1984	1	0.944	1	1	1
	1981	0	0.928	1	1	1
	1985	1	0.832	1	1	1
	1986	1	0.816	1	1	1
	1988	1	0.584	1	1	0
	1982	0	0.576	1	1	0
	1991	0	0.28	1	0	0
	1987	0	0.136	1	0	0
	1989	1	0.032	0	0	0
Correct negative	1992	0	0.024	0	0	0
	1990	0	0.016	0	0	0
	1983	0	0.008	0	0	0
	1993	0	0	0	0	0



Goal: given a year we want to predict if it was a rainy year (1) or not.

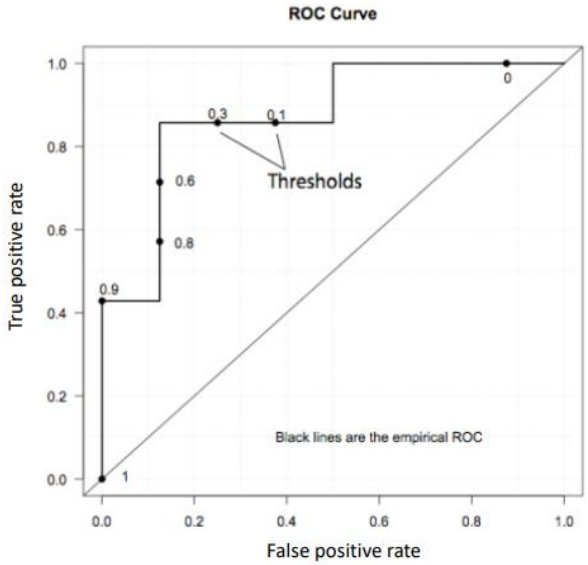
As we can see, we choose three different thresholds $\left(T = \begin{Bmatrix} 0.1 \\ 0.5 \\ 0.8 \end{Bmatrix}\right)$ which means that if the probability is above the threshold value, it is considered as positive (negative otherwise). Also note that results are being sorted by the **forecast probability**.

We obtain the **ROC curve** by changing the threshold limit.

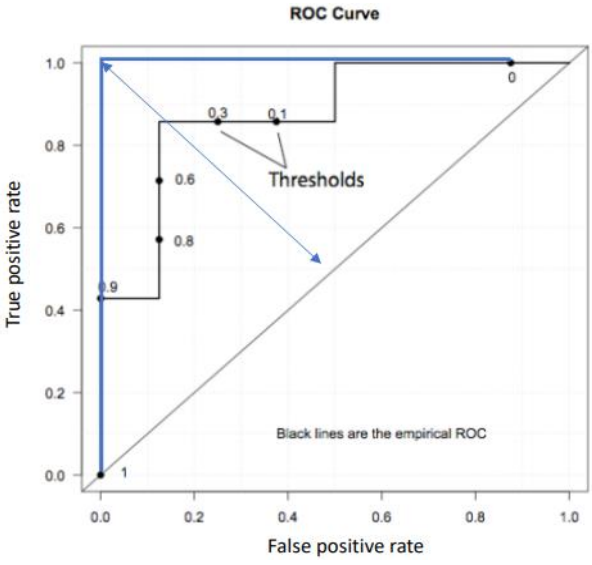
Note: we are using the same hypothesis but just by changing the threshold we get different curves, we connect the dots on the graph in order to plot the curve (which represents a different value of the threshold).

Area of the curve is a qualitative measure of how good our estimation is (Less the curve, better the estimation).

Corresponding ROC curve



Area of the ROC curve



5.2 Comparing systems

In order to understand which system between two, respectively called A and B, is better, we need a way of evaluating them by a procedure called **t-test**.

5.2.1 t-test procedure

This is a statistical procedure used to determine whatever the mean difference between two sets of observations is zero (So identical).
In order to do that we measure twice the subject (or entity) resulting in pairs of observations.
During the course of the paragraphs we will call them:

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

Case: we are changing the threshold so that we will get different values of accuracy etc.
Alternative hypothesis: one of the systems results to be more accurate than the other one.
Null hypothesis: they seems to have the same accuracy. If we get a **null hypothesis** it may seem that all observable differences are explained by random variation.
Goal: understand if there is a statistical importance comparing these two systems.

5.2.2 Hypothesis test

It is a test which uses paired t-test to determine the probability p that the mean difference supports the null hypothesis. If p is sufficiently small ($p < 0.05$ usually) then we are rejecting the **null hypothesis**. This means that we are trying to evaluate how much different these two systems are using this term p.
We define the following vector $\delta = \{y_1^a - y_1^b, \dots, y_i^a - y_i^b, \dots, y_n^a - y_n^b\}$ which represents the difference between each component of the vectors y^a and y^b ($\delta = y_a - y_b$).
1) We can calculate **mean**:

$$\bar{\delta} = \frac{1}{n}\sum_{i=1}^{n} \delta_i$$

2) Which allows us to calculate the **t-statistic** (t):

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

Note: if you want to remember it:

$$t = \frac{\bar{\delta}}{\sqrt{\frac{Var(\delta)}{n-1}}}$$

3) Determine the corresponding $p - value$, by looking up t in a table of values for the **student's distribution** with $n - 1$ degrees of freedom (The p value is called **probability mass value**)
Note: we can model δ (difference between the y_i^a element with the y_i^b element) as a **gaussian probability** because it meets the requirements to do it (Independence between the variable due to the randomness of the observation)

Different p test value

1) $p = 2\ P_r(T > |t|)$ two-tailed

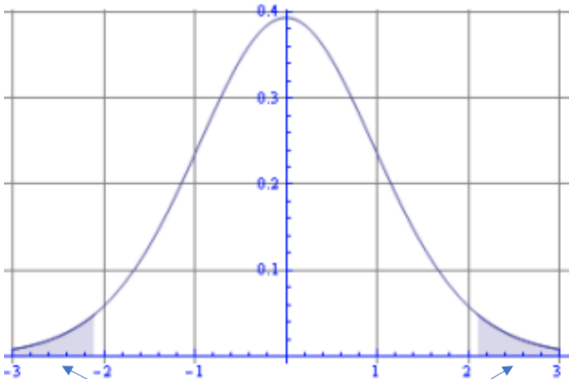
Is the accuracy of the two systems different?

2) $p = P_r(T > t)$ upper-tailed

Is A better than B?

3) $p = P_r(T < t)$ lower-tailed

Is B better than A?



Problem (one-tailed): if the question **fails** (A better than B? no) then we need to run the test **again** (B is better than A?) in order to be sure we need to do that.
Lazy solution: run the **two-tailed** test (Are the statistics different?)

5.2.3 Coefficient of determination – R^2

Thanks to R^2 we are able to quantify how much of the total deviation can be **explained** with the regression. We measure the proportion of the variance in the dependent variable (**predictable**) from the independent variable.
First we define:
1) $\bar{y}$ → mean observed values (mean of the **real values**)
2) y^*→ predicted value
3) y → observed value (**real values**)
So that we can define:

Total deviation

$$Det(T) = \sum_{i=1}^{m} \left(y^{(i)} - \bar{y}\right)^2$$

Which is the sum of the distances between a **real value** and the **real mean**

Regression Deviation

$$Det(R) = \sum_{i=1}^{m} \left(y^{(i)*} - \bar{y}\right)^2$$

Which is the sum of the distances between our **predicted value** compared to the **real mean**.

Residual Deviation

$$Det(E) = \sum_{i=1}^{m} \left(y^{(i)} - y^{(i)*}\right)^2$$

Which is basically the sum of the absolute errors squared $\left(\left(y^{(i)} - y^{(i)*}\right)^2 = \left(e^{(i)}\right)^2\right)$

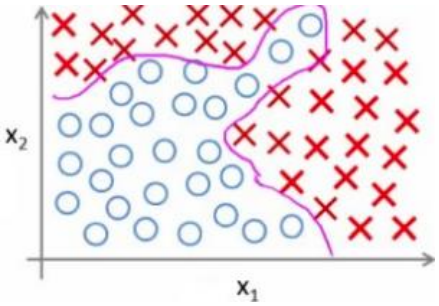
We can compute then:

$$R^2 = \frac{Dev(R)}{Det(T)} = 1 - \frac{Det(E)}{Det(T)} = \frac{cov(X,Y)^2}{Dev(X)Dev(Y)}$$

How to work with R^2:
First, $R^2 \in [0,1]$ and the closer it gets to 1 the better the data fit the model.
Depending on this value we can observe how well observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model.

5.2.3.1.1 0. lezione 4.3.pdf

5.3 Neural Networks



Supposing we have a problem that given an input x with a number of features n and for example let say that $n = 100$.
Hypothetically speaking, if we want to improve the predictions of our model we may choose to combine the features.
If we try to solve this problem with a **linear regression strategy** we will find out that the magnitude of features we generate by combining the initial one or by creating quadratic combination will reach magnitude of 5000 or even 170000 features
to analyze (basically an impracticable road because we are increasing too much the complexity.

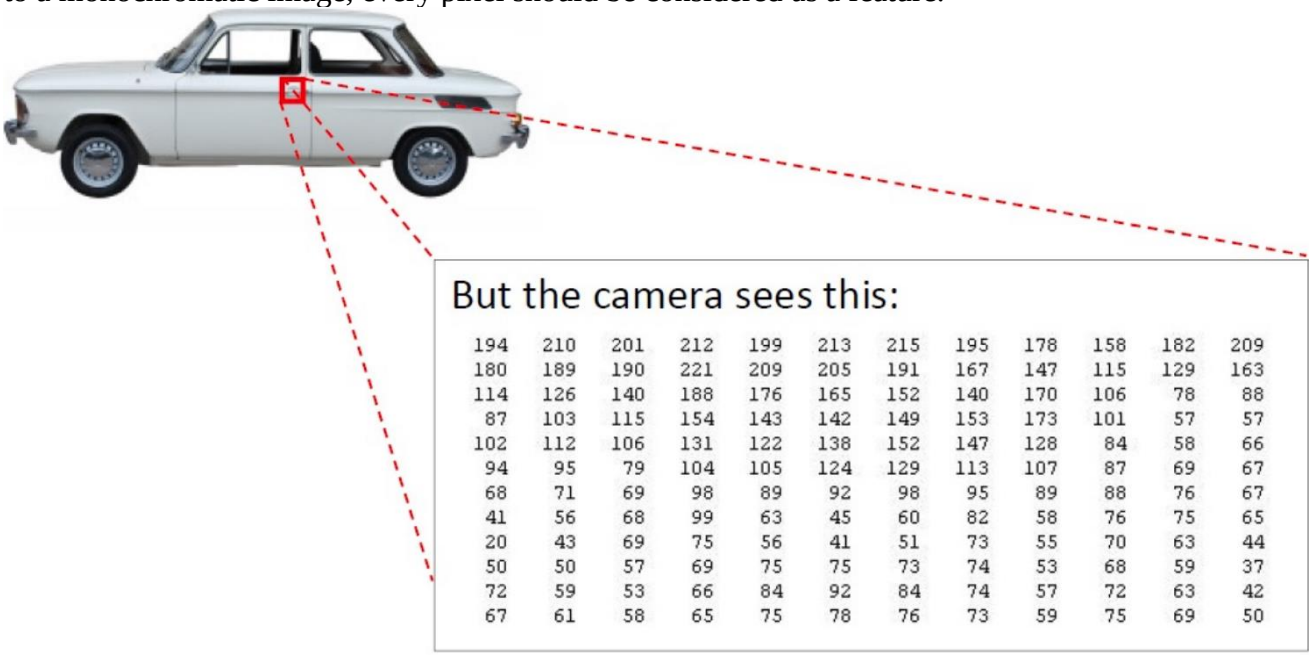
x_1 = size
x_2 = #bedrooms
x_3 = #floors
x_4 = age
...
x_{100}

n=100

$$x_1^2, x_2^2, x_3^2, \dots$$
$$x_1x_2, x_1x_3, x_1x_4, \dots, x_1x_{100}$$
$$\approx \quad 5000 features \quad \mathcal{O}(n^2)$$

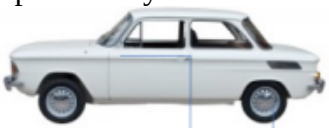
$$x_1^3, x_2^3, x_3^3, \dots$$
$$x_1^2, x_2^2, x_3^2, \dots$$
$$x_1^2x_2, x_1^2x_3, x_1^2x_4, \dots, x_1^2x_{100}$$
$$x_1x_2, x_1x_3, x_1x_4, \dots, x_1x_{100}$$
$$\approx \quad 170000 features \quad \mathcal{O}(n^2)$$

Just by thinking about an image, every pixel must be handled as a number. If we simplify the problem to a monochromatic image, every pixel should be considered as a feature.

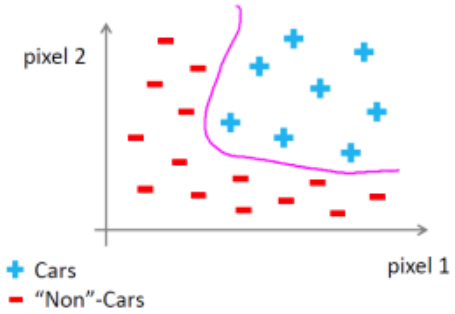


We are just analyzing the small red square and just by looking at this table, every number should be considered as a feature (remember, we are still considering the monochromatic case)

So the problem is simple to understand:
We want a way to compute handle huge amounts of features (like the one in an image) without increasing the magnitude of the problem exponentially



Simple 50 x 50 pixels images imply 2500 features

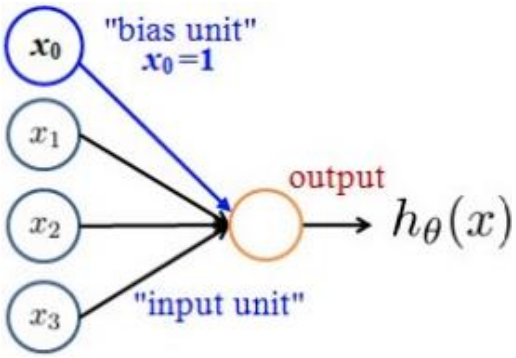


$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

Just by considering the quadratic features we would get (like in the example) a magnitude of 6m features just by 2500 pixel.

5.4 Neural model (RIGUARDA)

Different model compared to the previous one. Here we have the combination of our features which will enter the "**neuron**" which is basically a **gate** (A function called **activation function** determine if the neuron will active or not)

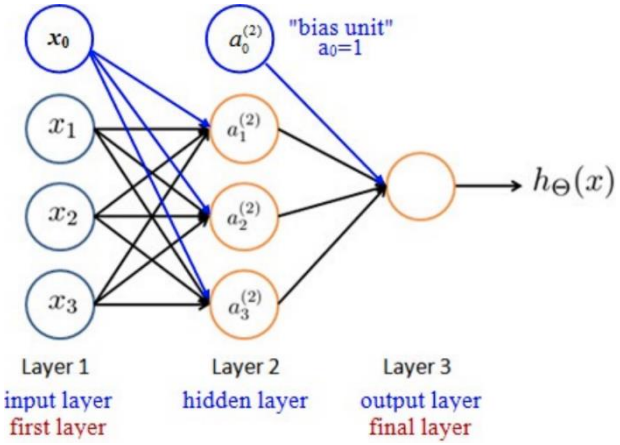


Activation function (Sigmoid [logistic])

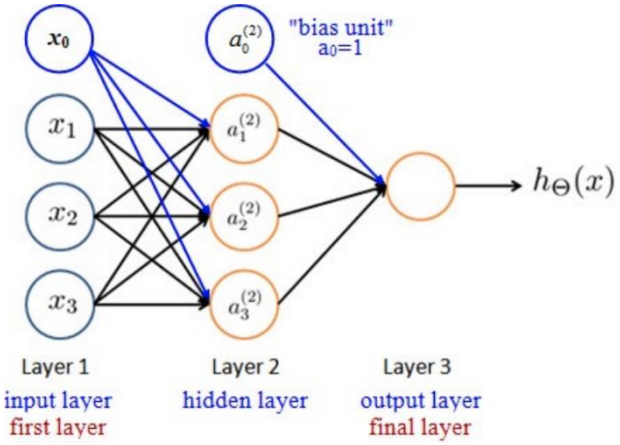
$$h_\theta(x) = \frac{1}{1 + e^{-\theta T}}$$

So **activation functions** works like gates and determine if an information should pass or not through the neuron.

5.4.1 Neural network



What we are doing now is to increase the number of layers where each layer is a combination of the previous one (Which can be considered as features and hypothesis in some sense).



If we consider how the layer 1 interact with layer 2, every feature reaches every neuron of the next layer and by doing that different weight will be assign:

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right)$$
$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right)$$
$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right)$$

So every x_i has his own weight inside a generic $a_i^{(j)}$ (Layer j, "**activation unit**" i)
In the end:

$$h_\Theta(x) = a_1^{(3)} = g\left(\Theta_{10}^{(2)}a_0 + \Theta_{11}^{(2)}a_1 + \Theta_{12}^{(2)}a_2 + \Theta_{13}^{(2)}a_3\right)$$

Basically, just because our network is composed by 3 layers, we have that our hypothesis is situated at the 3rd layer so is equivalent to the element $a_1^{(3)}$ as it is showed (The last neuron so to speak).

Why have we represented like this the layers?

$$\Theta^{(1)} = \begin{bmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} & \Theta_{13}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} & \Theta_{23}^{(1)} \\ \Theta_{30}^{(1)} & \Theta_{31}^{(1)} & \Theta_{32}^{(1)} & \Theta_{33}^{(1)} \end{bmatrix} \quad \Theta^{(2)} = \begin{bmatrix} \Theta_{10}^{(2)} & \Theta_{11}^{(2)} & \Theta_{12}^{(2)} & \Theta_{13}^{(2)} \end{bmatrix}$$

Because know we can represent our model as a matrix, by changing the weights we change the behavior of the entire system.

Overall

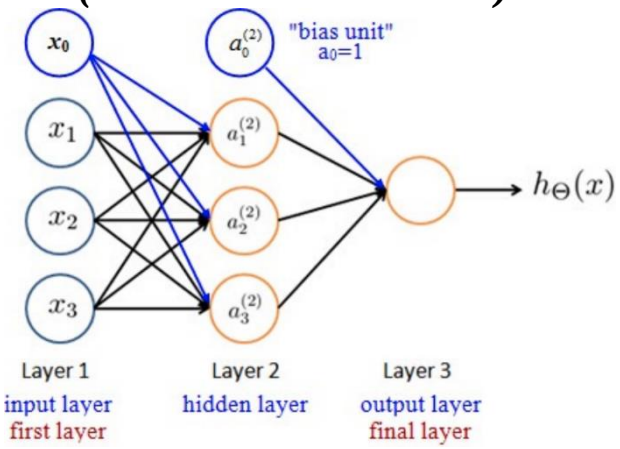
The **first layer** will enter inside each **neuron** of the **second layer** and every feature will get is own **weight** in the process.
Each **second layer neuron** will act as a **gate** using the **sigmoid** (The **activation function**) and by doing that we will **activate the neuron** by letting the information passing through it.
After this, each neuron of layer 2 will enter the last neuron in the layer 3 (in this case the same as the **hypothesis**) and with the same principle, it will assign each **second layer neuron** a weight.
Depending on the final result, the last neuron will activate or not by showing the final result

Capitolo 6 Lezione 2022-11-14

Lectures before this date were the **PYTHON** ones.

6.1.1.1.1 0. lezione 4.3.pdf (Slide 8)

6.1 Neural networks (Notations and remarks)



$$\Theta^{(1)} = \begin{bmatrix} \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} & \Theta_{13}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} & \Theta_{23}^{(1)} \\ \Theta_{30}^{(1)} & \Theta_{31}^{(1)} & \Theta_{32}^{(1)} & \Theta_{33}^{(1)} \end{bmatrix} \quad \Theta^{(2)} = \begin{bmatrix} \Theta_{10}^{(2)} & \Theta_{11}^{(2)} & \Theta_{12}^{(2)} & \Theta_{13}^{(2)} \end{bmatrix}$$

Note: just a reminder about what it has been discussed in the previous lecture.

1st: we have a matrix of weights associated with each layer and of course.
Exception: the final layer (**output** or **hypothesis**) is not a matrix but rather a vector
2nd: each layer we add a **bias unit** that adds also new weights to consider
3rd: with "j" we usually denote the **layer** and with "i" the **unit**, so it follows that $a_i^{(j)}$ is an element belonging to the layer j and denoted as unit i.
Each $a_i^{(j)}$ is associated with weights $\Theta^{(j)}$ which generate it passing through a **sigmoid** $g(z)$:

$$a_i^{(j)} = g\left(a^{(j-1)}\Theta_i^{(j-1)}\right) = g\left(z_i^{(j)}\right)$$

In principle, we associate $\left[a_i^{(j)}, z_i^{(j)}\right]$ through the simoid $a_i^{(j)} = g\left(z_i^{(j)}\right)$
Notation:

$$x = \begin{bmatrix} x_0 = 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(j)} = \begin{bmatrix} z_1^{(j)} \\ z_2^{(j)} \\ z_3^{(j)} \end{bmatrix} \quad a^{(j)} = \begin{bmatrix} a_1^{(j)} \\ a_2^{(j)} \\ a_3^{(j)} \end{bmatrix} \quad \Theta^{(j)} = \begin{bmatrix} \Theta_1^{(j)} \\ \Theta_2^{(j)} \\ \Theta_3^{(j)} \end{bmatrix}$$

$Input$ $Generic\ z$ $Generic\ a$ $Generic\ \Theta$

Exception: $a_0^{(j)} = 1$ every layer and it does not depend on weights, but it will contribute to create new parameters $a_i^{(j+1)}$ with weights $\Theta_i^{(j+1)}$.

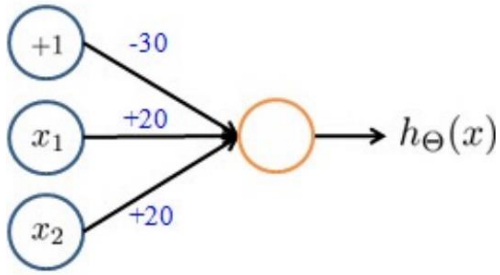
6.2 *Example: XOR with a Neural Network*

One of the most notorious not linear function to be represented is a XOR, where clearly a regression is not possible

In order to solve this problem, we will analyze the AND, NOT and OR operations

6.2.1 AND OR NOT analysis

AND



With the right hypothesis an **AND** can be approximated like:

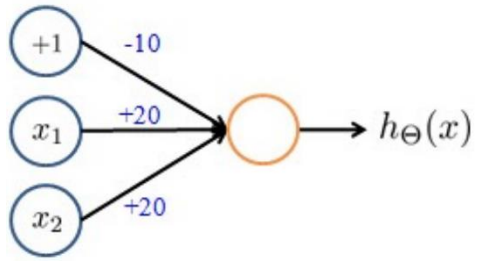
$$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$$

Since $x_1, x_2 \in \{0,1\}$ and $x_0 = 1$ we can observe that:

x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \cong 0$
0	1	$g(-10) \cong 0$
1	0	$g(-10) \cong 0$
1	1	$g(+10) \cong 1$

We have approximated an **AND** with a neural network

OR



Same concept but we want to approximate an **OR**:

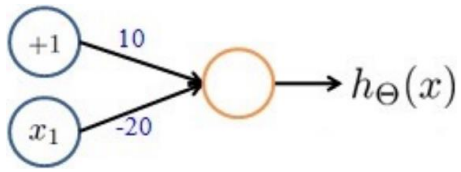
$$h_\Theta(x) = g(-10 + 20x_1 + 20x_2)$$

Since $x_1, x_2 \in \{0,1\}$ we can observe that:

x_1	x_2	$h_\Theta(x)$
0	0	$g(-10) \cong 0$
0	1	$g(+10) \cong 1$
1	0	$g(+10) \cong 1$
1	1	$g(+30) \cong 1$

We have approximated an **OR** with a neural network

NOT



Last but not least, a **NOT**:

$$h_\Theta(x) = g(-10 + 20x_1)$$

Since $x_1 \in \{0,1\}$ we can observe that:

x_1	$h_\Theta(x)$
0	$g(-10) \cong 0$
1	$g(+10) \cong 1$

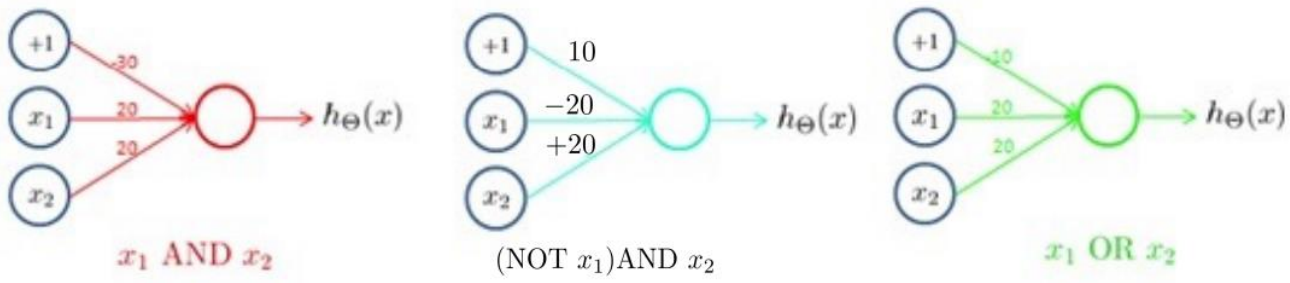
We have approximated a **NOT** with a neural network

6.2.2 XOR analysis

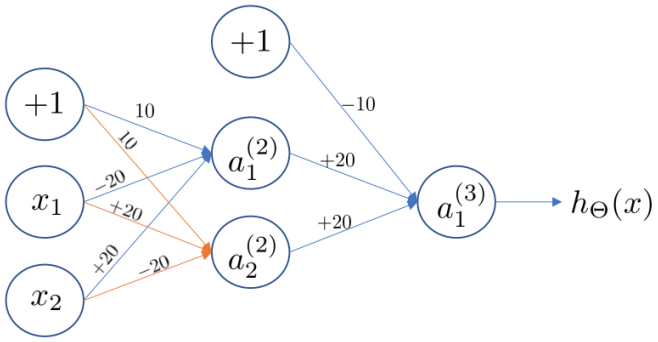
Now that we know the basic operations **AND OR NOT**, we can express our **XOR**:

$$x_1 XOR x_2 = (x_1 AND (NOT x_2)) OR (x_2 AND (NOT x_1))$$

We are not analyzing anymore the problem of the XOR, instead we got an equivalent expression in terms of AND, OR and NOT.



We can combine the behavior in order to obtain our final result:



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\Theta(x)$
0	0	0	0	0
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

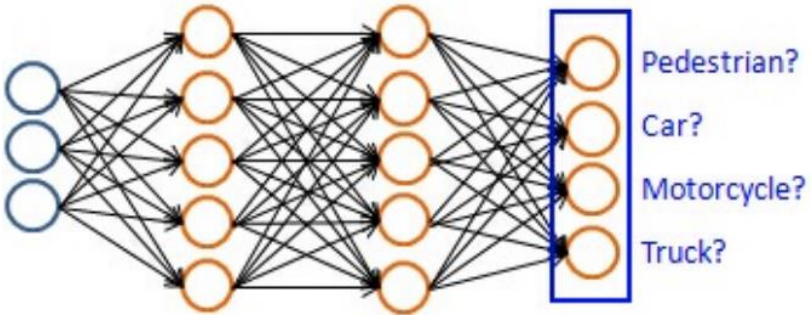
$$x_1 \text{XOR } x_2 = (x_1\ AND\ (NOT\ x_2)) OR\ (x_2\ AND\ NOT\ x_1)$$

Property (Not proven yet): combining neural network by combining their behaviors **IMPORTANT**

6.3 Neural Network Advantage (Multi-class classification)

If we compare this characteristic with a **logistic regression** we will quickly find out that the logistics was bounded by a "binary classification" (If we have A, B, C class we ask the model if it is class A or not; if it is class B or not and if it is class C or not. With logistic we are simply applying binary classification multiple time)

With a neuronal network, depending on the output, we can predict multiple outcomes:



$$h_\Theta(x) \in \mathbb{R}^4$$

So for example:

$$h_\Theta(x) \cong \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \quad h_\Theta(x) \cong \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \quad h_\Theta(x) \cong \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \quad h_\Theta(x) \cong \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \quad Etc$$

$$Pedestian \qquad Car \qquad Motorcycle \qquad Truck$$

6.4 Cost function

As for the **logistic** and **regression** problems, the neural network will optimize its parameters by minimizing a proper cost function (Depending on the problem we are facing)

Classification

Logistic regression

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

This is the known formula to minimize (regularization one)

Neural network

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K}\left(y_k^{(i)}\left(\log h_\Theta\left(x^{(i)}\right)\right)_k + \left(1-y_k^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\Theta_{ji}^{(l)}$$

$h_\Theta(x) \in R^K \rightarrow \left(h_\Theta(x)\right)_k = k^{th}$, basically the k^{th} output.

Note 1: the regularization term may seem difficult to understand but it is just the sum of every weight of every layer (as it was for the logistic regression)

Regression

Linear regression

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

With regularization term.

Neural network

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(h_\Theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\Theta_{ji}^{(l)}$$

Note 1: the regression in general must have just one neuron as an output because it should predict a single output (continuous one) and not a class of output (like a classification)

6.5 Gradient computation (Classification and regression)

Classification

What we aim for is to use an algorithm in order to minimize the error of:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K}\left(y_k^{(i)}\left(\log h_\Theta\left(x^{(i)}\right)\right)_k + \left(1-y_k^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)_k\right)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\Theta_{ji}^{(l)}$$

So the problem is defined as $J(\Theta) = \min_{\Theta} J(\Theta)$

In order to do that we need to compute $J(\Theta)$ and the partial derivatives:

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)$$

Regression

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\left(h_\Theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}\Theta_{ji}^{(l)}$$

It changes the form, but it is the nature of what you have to do that is important:

$$J(\Theta)\ \Big|\ \Big|\ \frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta)$$

We should calculate the same terms, but it is just the formula that is different.

6.5.1 General notation

Given a training example (x, y) we can assume that:

1) $a^{(1)} = x$, the first layer of our network are the features x

2) by multiplying the neurons of a layer $a^{(j)}$ with the weight of the edges $\Theta^{(j)}$ we define our new term:

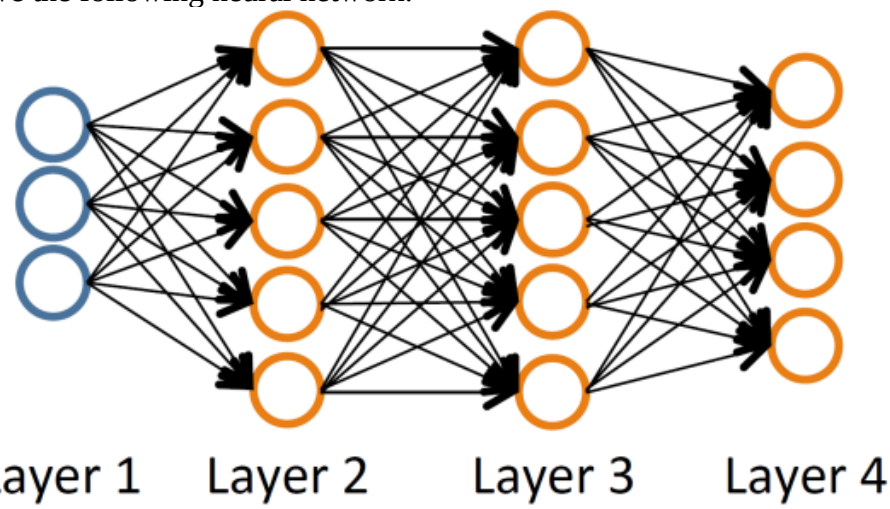
$$z^{(j+1)} = \Theta^{(j)} a^{(j)}$$

The next neurons will be calculated, as we told, $a^{(j+1)} = g\left(z^{(j+1)}\right) = g\left(\Theta^{(j)} a^{(j)}\right)$.

3) Each layer we add the term $a_0^{(j)} = 1$ (Bias) except for the layer for the hypothesis (Supposing that L is the max number of layers) $a^{(L)} = h_\Theta(x)$ which doesn't need the $a_0^{(L)} = 1$

Example

Supposing we have the following neural network:



1) $a^{(1)} = x$

2) $z^{(2)} = \Theta^{(1)} a^{(1)}$

3) $a^{(2)} = g\left(z^{(2)}\right)$ (Remember to add $a_0^{(2)} = 1$)

4) $z^{(3)} = \Theta^{(2)} a^{(2)}$

5) $a^{(3)} = g\left(z^{(3)}\right)$ (Remember to add $a_0^{(3)} = 1$)

6) $z^{(4)} = \Theta^{(3)} a^{(3)}$

7) $a^{(4)} = g\left(z^{(4)}\right) = h_\Theta(x)$ (The term $a_0^{(4)} = 1$ is not need, also $L = 4$)

Notes: just remember that each layer is just a transformation of the previous back with means of weight. Thanks to this example we should understand that if we want to move from layer *A* to *B* we need to consider all of these parameters in the middle.

6.5.2 Gradient computation – Regression Loss

Thanks to the previous example, the analysis of the partial derivative should be practicable relatively easy:

$$\frac{\partial}{\partial \Theta^{(j)}} J(\Theta)$$

Procedure with example – Hypothesis layer (network with $L = 4$ layers) → $\left(\boldsymbol{\Theta^{(3)}}\right)$

We want to compute the following partial derivative of $J(\Theta)$:

$$\frac{\partial J}{\partial \Theta^{(3)}}$$

And we know the form of $J(\Theta)$:

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

So it follows that:

$$\frac{\partial J}{\partial \Theta^{(3)}} = \frac{\partial\left(\frac{1}{2m}\sum_{i=1}^{m}\left(h_\Theta\left(x^{(i)}\right) - y^{(i)}\right)^2\right)}{\partial \Theta^{(3)}} = \frac{\cancel{2}}{\cancel{2}m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) * \frac{\partial\left(h_\theta\left(x^{(i)}\right)\right)}{\partial \Theta^{(3)}}$$

We can simplify the following formula because:

$$\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) = \left(a^{(4)} - y\right)$$

$$\frac{\partial\left(a^{(4)}\right)}{\partial \Theta^{(3)}} = \frac{\partial\left(g\left(z^{(4)}\right)\right)}{\partial \Theta^{(3)}} = g'\left(z^{(4)}\right)\frac{\partial\left(z^{(4)}\right)}{\partial \Theta^{(3)}} = g'\left(z^{(4)}\right)a^{(3)} \;\Big|\Big|\; z^{(4)} = a^{(3)}\Theta^{(3)}$$

We get the following result:

$$\frac{\partial J}{\partial \Theta^{(3)}} = \left(a^{(4)} - y\right)g'\left(z^{(4)}\right)a^{(3)}$$

We define a new term $\delta^{(4)} = \left(a^{(4)} - y\right)g'\left(z^{(4)}\right)$ so the final form:

$$\frac{\partial J}{\partial \Theta^{(3)}} = \delta^{(4)}a^{(3)}$$

Note: the parameter $\delta^{(4)}$ and $a^{(3)}$ are all known information so we can compute them in order to compute all the derivatives.

Generic next layer derivative → $(\boldsymbol{\Theta^{(2)}})$

$$\frac{\partial J}{\partial \Theta^{(2)}} = \delta^{(4)}\frac{\partial a^{(3)}}{\partial \Theta^{(2)}}$$

Without the need of computing $\delta^{(4)}$ (Because we should have already computed for the previous derivative) we can compute this derivative with the same principle:

$$\frac{\partial a^{(3)}}{\partial \Theta^{(2)}} = \frac{\partial g\left(z^{(3)}\right)}{\partial \Theta^{(2)}} = g'\left(z^{(3)}\right)\frac{\partial z^{(3)}}{\partial \Theta^{(2)}} = g'\left(z^{(3)}\right)a^{(2)} \;\Big|\Big|\; z^{(3)} = a^{(2)}\Theta^{(2)}$$

So in the end we get:

$$\frac{\partial J}{\partial \Theta^{(2)}} = \delta^{(4)}g'\left(z^{(3)}\right)a^{(2)} = \delta^{(3)}a^{(2)}$$

Last layer → $\left(\boldsymbol{\Theta^{(1)}}\right)$

$$\frac{\partial J}{\partial \Theta^{(1)}} = \delta^{(3)}\frac{\partial a^{(2)}}{\partial \Theta^{(1)}} = \delta^{(3)}g'\left(z^{(2)}\right)a^{(1)} = \delta^{(2)}a^{(1)}$$

Note: same operations, same principle if you remember that $z^{(2)} = a^{(1)}\Theta^{(1)}$

Capitolo 7 Lezione 2022-11-17 (RIVEDI)

7.1.1.1.1 0. lezione 4.3.pdf (Slide 30)

7.1 Gradient computation – Regression Loss (Rivisitare)

It is possible to compare a **logistic regression gradient** to a neural network one:

$$\frac{\partial}{\partial \theta_k} J(\theta) = \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x^{(i)}$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta^{(l+1)}$$

Remember: L is the number of layers inside the network

We can consider $\delta_j^{(4)} = a_j^{(4)} - y_j$ where $h_\Theta(x) = a_j^{(4)}$ so that it is possible to generalize each layer:

$$\delta^{(3)} = \left(\Theta^{(3)}\right)^T \delta^{(4)} .* \boldsymbol{g'}\left(\boldsymbol{z^{(3)}}\right) \rightarrow a^{(3)} .* \left(1 - a^{(3)}\right)$$

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* \boldsymbol{g'}\left(\boldsymbol{z^{(2)}}\right) \rightarrow a^{(2)} .* \left(1 - a^{(2)}\right)$$

7.2 Backpropagation algorithm

1) Training set $\left\{\left(x^{(1)}, y^{(1)}\right), \dots, \left(x^{(m)}, y^{(m)}\right)\right\}$

2) Set $\Delta_{ij}^{(l)} = 0$ for all l, i, j

3) Cycle the following operations (For $i = 1$ in $range(m)$ if you seek for a python implementation)

3.1) Set $a^{(1)} = x^{(1)}$

3.2) **forward propagation** to compute $a^{(l)}$ for $l = 2, \dots, L$

3.3) compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

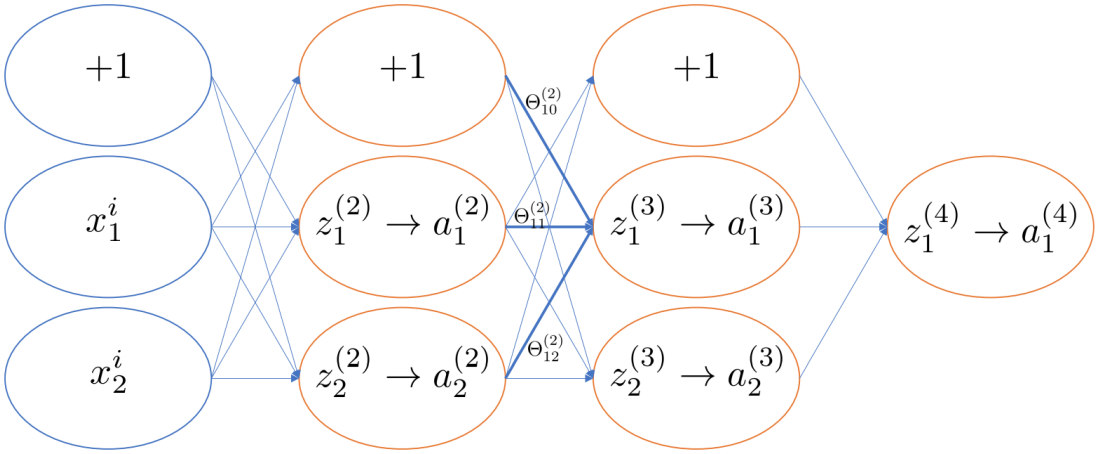
3.4) compute $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

We then define:

$$\left.\begin{array}{ll} D_{il}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} + \lambda\Theta_{ij}^{(l)} & if\ j \neq 0 \\ D_{il}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} & if\ j = 0 \end{array}\right\| \frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

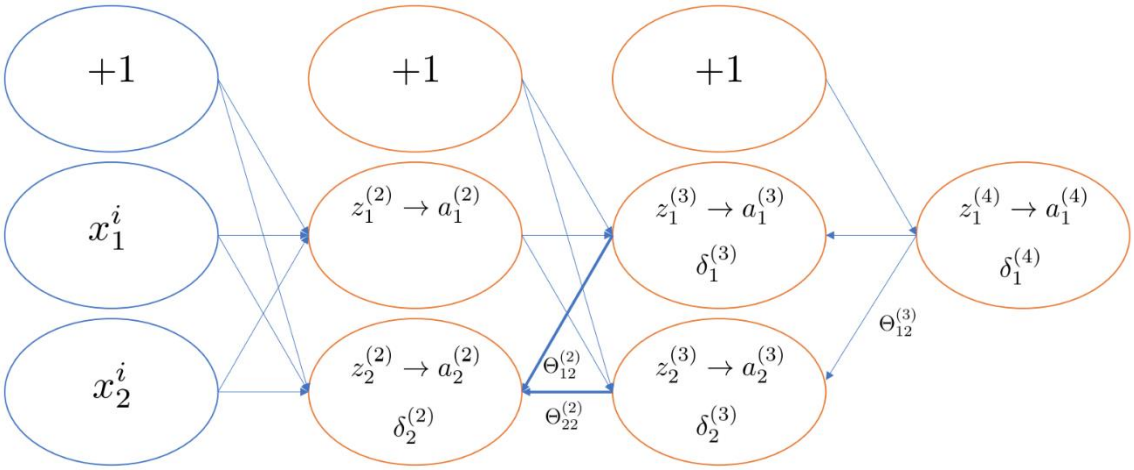
7.2.1 Backpropagation intuition

Forward propagation



Forward means that we move from the input layer to the output layer (hypothesis).
Through the input, we want to generate an output (In the process we generate the parameters (a, z) with specific weights Θ)

Backward propagation



Backward means that we move from the output layer (**hypothesis**) to the input layer.
We calculate the different error contributions $\delta_j^{(l)}$ (error contribution for $a_j^{(l)}$ in layer l) and calibrate the weights Θ in the process.

$$\delta_1^{(4)} = y^{(i)} - a_1^{(i)}$$
$$\delta_2^{(3)} = \Theta_{12}^{(3)} \delta_1^{(4)}$$
$$\delta_2^{(2)} = \Theta_{12}^{(2)} \delta_1^{(3)} + \Theta_{22}^{(2)} \delta_2^{(3)}$$

7.2.2 Zero initialization

For the first iteration of the algorithm we should try to set the weights Θ to a value.

First guess: If we set all the Θs to 0 $\left(\Theta_{ij}^{(l)} = 0\right)$ will cause that all parameters will be equal:

$$a_1^{(2)} = a_2^{(2)}$$
$$\delta_1^{(2)} = \delta_2^{(2)}$$
$$\frac{\partial}{\partial \Theta_{01}^1} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^1} J(\Theta)$$
$$\Theta_{01}^1 = \Theta_{02}^1$$

Each update will result in the same parameter no matter how many iterations.

7.2.3 Random initialization

In order to solve the problem of the **zero initialization**, we set the weights randomly inside a small interval $\Theta_{ij}^{(l)} \in [-\varepsilon, +\varepsilon]$

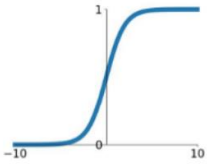
Other rules in order to increase the stability

Problem: gradients of the network's activations might vanish or explode
1) The mean of the activations should be zero
2) The variance of the activations should stay the same across every layer
Solution
1) Standard normal
2) Uniform
3) Xavier (The most effective)

7.3 Activation Functions

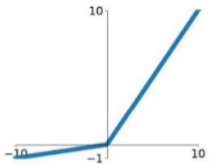
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



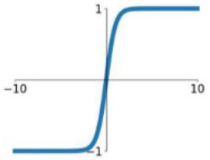
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

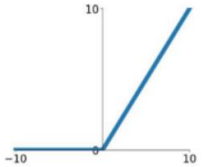


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

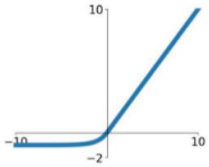
ReLU

$$\max(0, x)$$

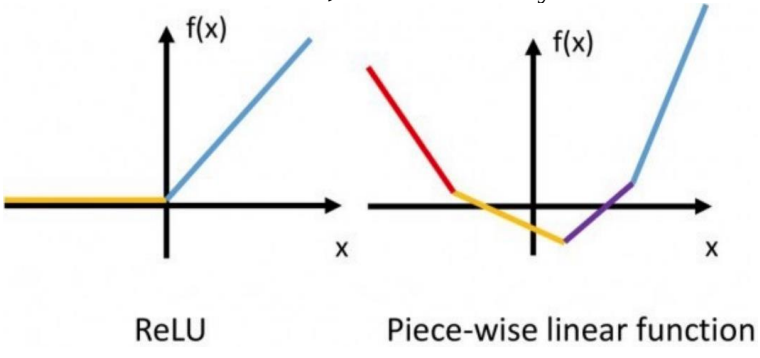


ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



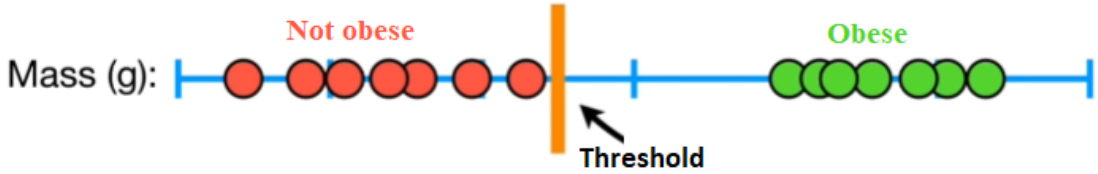
Notes:
1) **ReLU** is not used (Discontinuity in 0) compared to **ELU**.
2) **Maxout** is a combination of different **ReLU,** and it is not a symmetrical function



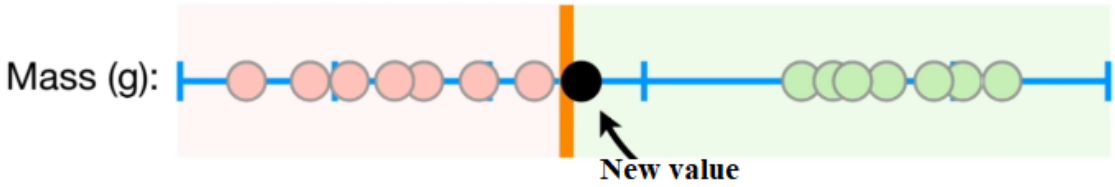
Capitolo 8 Lezione 2022-11-17

8.1 SVM (Intuition – Not professor related)

Note: this is an extra part I added which explains how SVM works in general.
Paragraph 8.4 starts with the professor's slide.

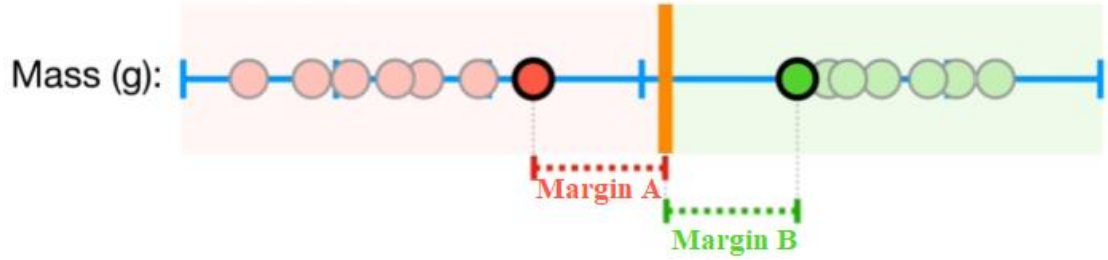


Let's say we are measuring if a mouse is **obese** depending on its weight (Expressed in grams)
We put a threshold based on the results obtained.
Threshold: depending on the value you want to predict; we will classify it as one of these two classes.



Supposing that we want to predict a new value, based on its position we would have classified it as **obese** because it crosses the threshold, but **it does not make any sense** because it is much closer to the **not obese** class.
Problem: the threshold is not well positioned so we need to optimize this

8.1.1 Margin (Definition)

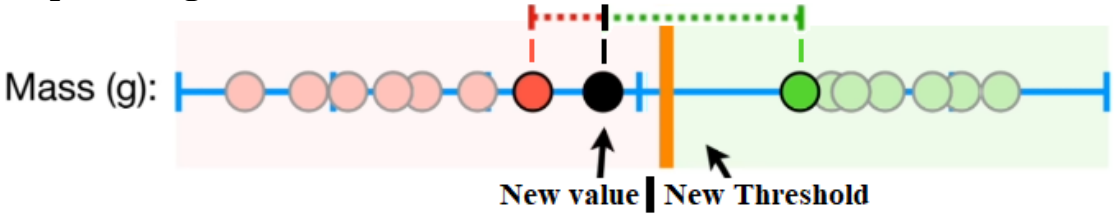


It is the distance between the nearest point to the threshold which are the edges of each *cluster*[1]
Generally: we will get the smallest margin among the two:

$$Margin = \min(Margin\ A, Margin\ B)$$

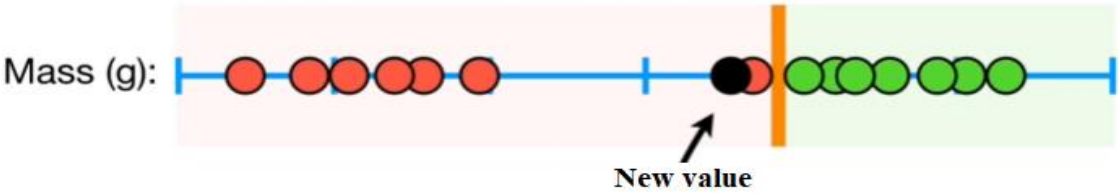
[1] **Cluster**: items with similar properties (if you watch **obese** and **not obese** you can recognize them as two clusters and **geometrically** are near each other)

8.1.2 Optimizing threshold: mean distance



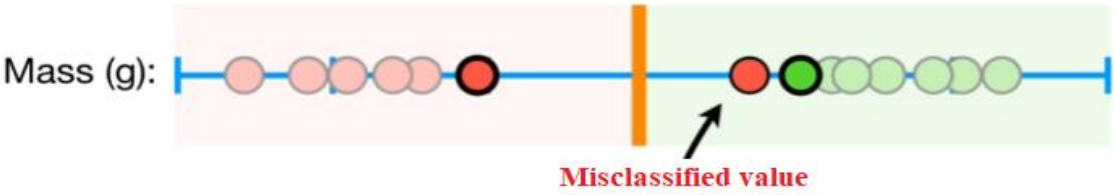
We get the nearest points (in terms of distance) from the two classes, and we position the new threshold in the middle of that. The new value will be classified as a **not obese** right now and it makes more sense since it was nearer to that class.
Margin: since the threshold is positioned in the center of the two marginal points, the margin is the same both sizes (this is called **maximal margin classifier** because it is the largest margin we will ever get).

Problem (outliers)



Due to an outlier inside **not obese** the margin (still using a **maximal margin classifier**) got really small than before. The threshold has been positioned still in the center of the marginal points and the new value will be classified as **not obese** instead of **obese** even if it is much closer to the last one
Consideration: maximal margin classifier is susceptible to outliers.

8.1.3 Optimizing threshold by misclassification



We misclassified a value (The one which creates the maximal margin classifier **sensitive** to errors)

Bias/Variance Trade-off

This is clearly a trade-off which plagues the ML world because in the previous example, we picked a **threshold** based too much on the training data (Low Bias) and it performed poorly on new data (High variance). What we are doing right now is allowing misclassification (Higher Bias) in order to improve the performance of the model (Low variance)

8.1.4 Soft margin (With problem)

It is any margin that doesn't consider the edges of the clusters

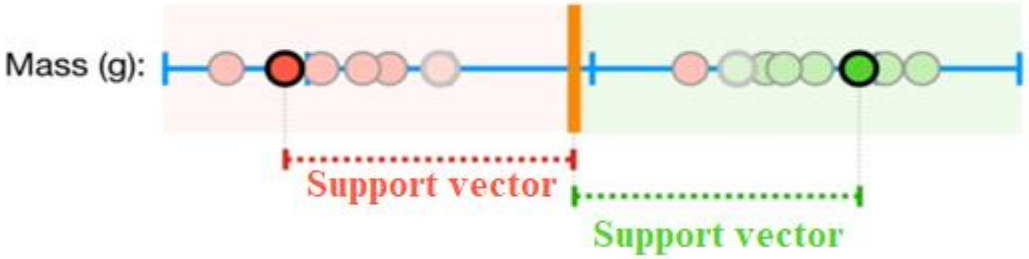
Problem: which **soft margin** is better in order to maximize the accuracy?
Solution: Cross-validation by changing the points that determine the threshold position and soft margin dimension.
Consideration: In this case there is **one misclassified value** and many **correctly classified** values (The points correctly classified which are nearer the threshold than the points used for the threshold)

8.2 Support Vector Classifier (Soft Margin Classifier)

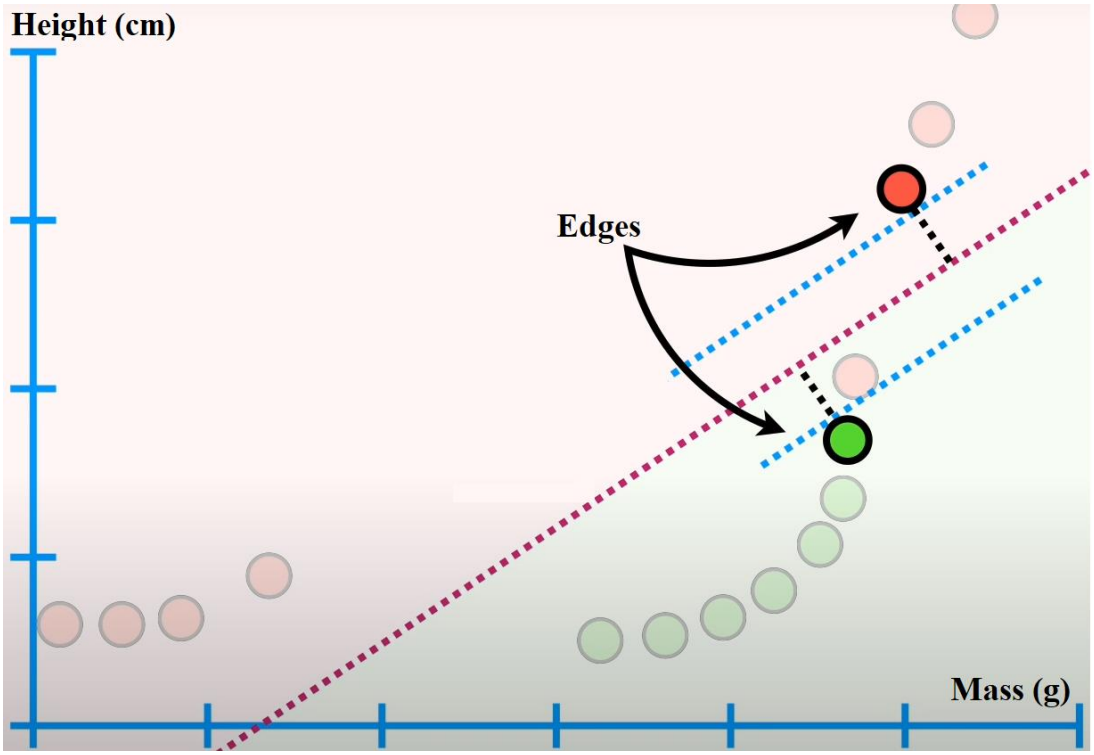
When we use a **soft margin** in order to determine the location of a threshold then we are using a **Support Vector Classifier** to classify these observations.
Note: when we talk about the **support vector classifier,** we refer to the **threshold**

8.2.1 Support Vector (Definition)

The points of the training set which are exactly on the margin are called **support vectors**.

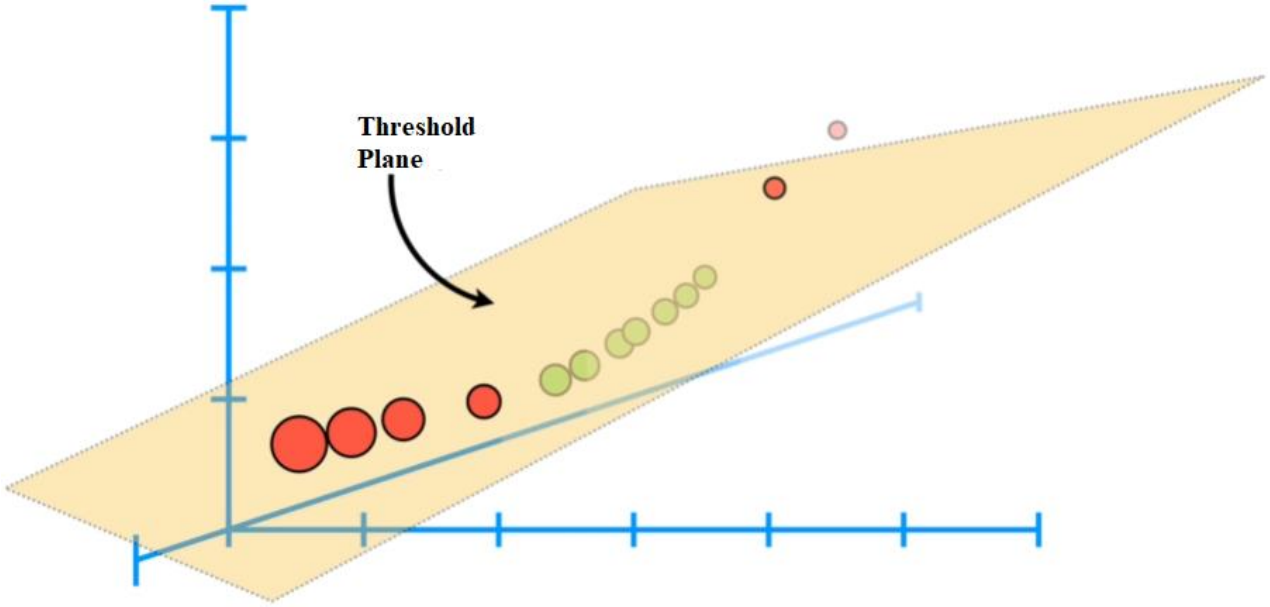


8.2.2 2D Support Vector Classifier (Example)



Supposing that for the same example of the mouses, we add a new feature (height of the mice expressed in centimeters) we would obtain a line as a threshold.
Notes: There is a miss classified value and the blue lines visually shows the margin.
Definition: A support vector classifier is a line when the data are 2-Dimensional.
Soft margin: it is calculated by the edges (The points which determine the threshold)
Consideration: in order to obtain the best threshold, we use **cross-validation**

8.2.3 3D Support Vectors Classifier (Example)



If we add a new feature to a 2D example, we will get a 3D Support Vector Classifier.
In this case the **threshold** is a plane (2D space)

8.2.4 N dimension Support Vector Classifier

If we want to generalize this problem, with N dimensions we would get a threshold of $(N-1)$ dimensions.

Hyperplanes: A hyperplane is a "flat affine subspace".

1D: a line

2D: a plane

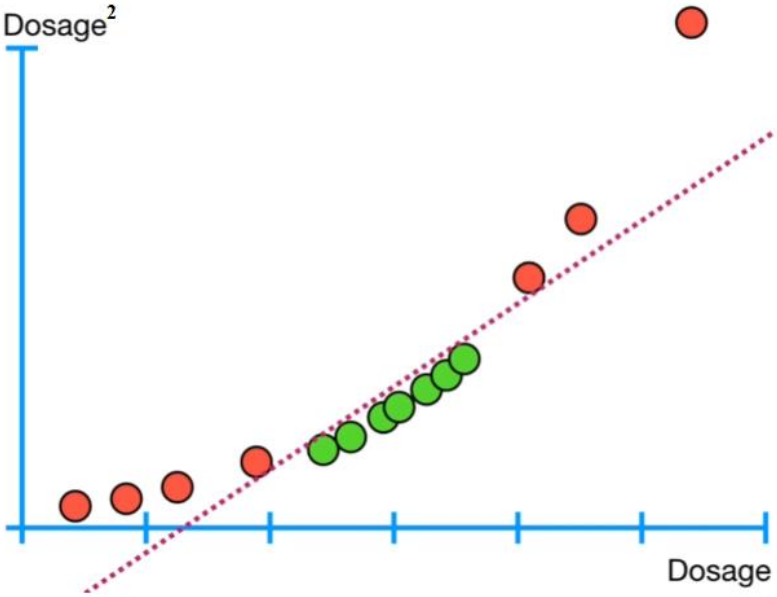
Threshold: technically, a threshold above or equal to 2D is a hyperplane (even a line). but practically, we use the term "hyperplane" when we can really represent it (So 3D and above)

8.3 Support Vector Machine (Intuition)

In this example, we are measuring the people **cured** and **not cured** by a drug and we are measuring the **dosage** (mg) given to the patient.
Note: The drug does not work for small or high dosage
Problem: no matter where we put the threshold, we would always get errors.

Support vector machine (Solution)

We have a problem with a dimension (In this case 1D), and we decide to increase its dimension.
How do we increase it? we increase the dimensions by transforming the original data. In this example, we have a 1D problem which cannot be solved so we decide to rely on a transformation of the data ($\boldsymbol{dosage}$), we increase the dimension by considering the square ($\boldsymbol{dosage^2}$)
Important note: the "**square**" operation is not the only one used, we could in fact use other transformations, but this topic will be explained better below (**Polynomial kernel** with $d = 2$)



As we can see, with the same problem increased from 1D to 2D, we can find a **support vector classifier** suitable enough to recognize **cured** and **not cured** people.

8.3.1.1.1 6_SVM.pdf

8.4 SVM Large Margin Classifier (Professor slides)

In general, we want to define the best threshold such that is able to generalize the problem by reducing error by **overfitting** and we want to intensify the robustness of the classification and therefore being able to classify better our data.
Theoretically speaking if we have two class "+" and "−", our **threshold** (hyperplane or line etc.) is defined as:

$$w * x + b$$

Where w is the **vector of weights**, x **our data** and b **the bias**, what we want is that:

$$w * x_+ + b \geq 0 \rightarrow y = +1$$
$$w * x_- + b \leq 0 \rightarrow y = -1$$

This is our **Decision rule**.
Goal: we want that every point of the two classes is assigned correctly to its class.

8.4.1 Large Margin Classifier (Criterion)

Using the criterion of the **Large Margin Classifier**, we reduce the risk of **overfitting** and increases the level of **generalization** of the problem by maximizing the **margin** and by setting a minimum distance from the threshold and each point of the dataset, respectively this distance is $+1$ for class "+" and -1 for class "−":

$$(1)\ w * x_+ + b \geq +1$$
$$(2)\ w * x_- + b \leq -1$$

If we consider the formula (1) we realize that multiplying by y_+ is like multiplying by $+1$:

$$\boldsymbol{y_+(w * x_+ + b) \geq 1}$$

But for the formula (2) multiplying by y_- is like multiplying by -1 which changes the inequality:

$$w * x_- + b \leq -1 \ \Big|\Big|\ -(w * x_- + b) \geq +1 \ \Big|\Big|\ \boldsymbol{y_-}(w * x_- + b) \geq +1$$

So, in the end we obtain:

$$\boldsymbol{y_-(w * x_- + b) \geq 1}$$

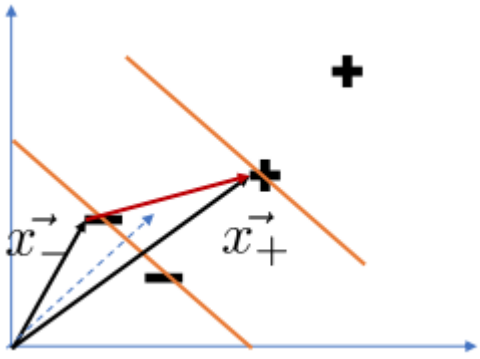
Since the formulas are the same, we can generalize that every point i must respect the following:

$$\boldsymbol{y_i(w * x_i + b) \geq 1}$$

8.4.2 Maximizing the margin (Development)

The points which $y_i(w * x_i + b) = 1$ are the **support vectors** that are the ones responsible for determining the **threshold** and therefore minimizing the error.
Before evaluating the margin, we are interested in the distance between x_+ and x_-:

$$(x_+ - x_-)$$

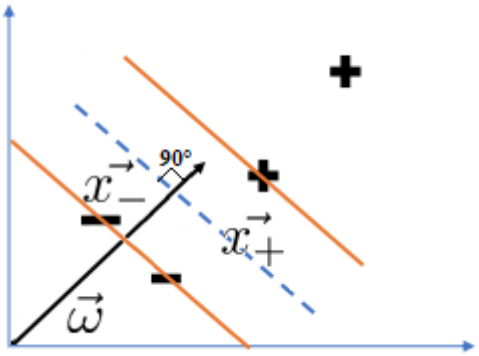


The points x_+ and x_- are **support vectors** for the classes $+$ and $-$ defined by $y_i(w * x_i + b) = 1$.

We are then interested in the projection of the difference of those two points on a direction perpendicular to the **threshold**:

$$margin = (x_+ - x_-)\frac{w}{|w|}$$

Where $w/|w|$ is a **unit vector** perpendicular to the **threshold**.



Since we projected $(x_+ - x_-)$ on $\frac{w}{|w|}$ we actually evaluated the margin.

All the points such that $y_i(w * x_i + b) = 1$ where:

$$w * x_+ + b = +1$$
$$w * x_- + b = -1$$

So it follows that:

$$w * x_+ = 1 - b$$
$$w * x_- = -1 - b$$

Obtaining the following formula for the margin:

$$margin = (x_+ - x_-)\frac{w}{|w|} = \big((1-b) - (-1-b)\big)\frac{1}{|w|} = \frac{2}{|w|}$$

The maximization problem is done on the $margin$:

$$\max margin = \max \frac{2}{|w|}$$

We then switch the maximization problem to a minimization one.

$$\max \frac{2}{|w|} = \min \frac{1}{2}|w|$$

Note: yes, this is equivalent

Last step

Evaluating $\min \frac{1}{2}|w|$ is equivalent to $\min \frac{1}{2}|w|^2$ and the SVM prefers the second one.

Important: Since the term to minimize is **quadratic**, it will not have local minimums (It is a parabola with just a global minimum with a **U** like scape)

$$\min \frac{1}{2}|w|^2$$

8.4.3 Lagrange method

In order to minimize the **margin**, we use the **Lagrange method** which consists of adding to the original problem the Lagrange function:

$$L = \frac{1}{2}|w|^2 - \sum \alpha_i [y_i(w * x_i + b) - 1]$$

Since we are interested in the **minimum** of w which minimizes the problem, it is the equivalent of finding the partial derivative of L with respect to w all equal to 0:

$$\frac{\partial L}{\partial w} = w - \sum_i \alpha_i * y_i * x_i = 0$$

We can conclude that:

$$w = \sum_i \alpha_i * y_i * x_i$$

Note: this is a great result because the vector w is a linear sum of the samples which have a Lagrange coefficient $a_i \neq 0$.

We do the same with the coefficient α:

$$\frac{\partial L}{\partial \alpha_i} = y_i(w * x_i + b) - 1 = 0$$

The bias can be evaluated just by considering one of the support vectors.

We evaluate the last derivative:

$$\frac{\partial L}{\partial b} = \sum_i \alpha_i * y_i = 0$$

This value will be useful in the next development.

We now consider the original $L(w, \alpha, b)$ by changing what has been obtained:

$$\frac{1}{2}|w|^2 = \frac{1}{2}\left(\sum_i \alpha_i * y_i * x_i\right)\left(\sum_j \alpha_j * y_j * x_j\right) = \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i x_j$$

We change the vector w with the formula obtained.

Note: the summation have i and j as indexes so be careful not to mix them.

$$\sum_i \alpha_i [y_i(w * x_i + b) - 1] = \sum_i (\alpha_i y_i w * x_i + \alpha_i y_i b - \alpha_i)$$

We divide the original summation into three different summations:

$$\sum_i (\alpha_i y_i w * x_i + \alpha_i y_i b - \alpha_i) = \sum_i \alpha_i y_i x_i w + \sum_i \boldsymbol{\alpha_i y_i b} - \sum_i \alpha_i$$

First) We realize that the **second term** is equal to 0 ($\partial L / \partial b$):

Second) we substitute the value of w:

$$\sum_i \alpha_i y_i x_i w = \sum_i \alpha_i y_i x_i \sum_j \alpha_j y_j x_j = \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i x_j$$

Third) the formula of L:

$$L = \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i x_j - \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i x_j + \sum_i \alpha_i$$

Last version of the formula

$$L = \sum_i^m \alpha_i - \frac{1}{2}\sum_i^m \sum_j^m \alpha_i \alpha_j y_i y_j \boldsymbol{x_i x_j}$$

Important: the optimization only depends on couples of samples $x_i x_j$ (their **dot product**)

Important: Due to the **equalities**, we are just working on **support vectors**; so, every x_i and x_j are **support vectors**.

8.4.4 Decision rule updated

If we put the w evaluated during the **Lagrange Method** into the **decision rule** we obtain:

$$\sum_{i}^{m} \alpha_i y_i x_i * u + b \geq 0$$

An unknown sample u will be classified as class "+" or class "−" depending on this formula which considers only the **support vectors**.

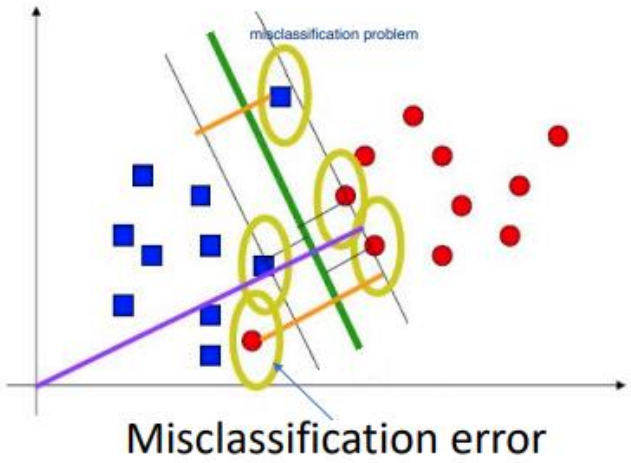
8.4.5 Cost function (MANCA!)

Some things are missing from the original slides

Lezione 2022-11-21

8.4.5.1.1 6_SVM.pdf (Slide 27)

8.1 SVM with error tolerance



Introducing non-linear separable date, the model may wrongly classify one point from a specified class to the other one. We must accept that some constraints are violated.
We introduce a **slack variable** $\xi^{(i)}$, so it is associated with each point $x^{(i)}$ such that an **error tolerance** is introduce in the model:

$$y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \xi^{(i)}$$

Objective: finding the best parameters of w and b by introducing an **error tolerance** in order to make a trade-off between accuracy and complexity of the model.
A **penalty** is added to the cost which increases it every misclassification done:

$$\frac{1}{2}|w|^2 + C\sum_{i=1}^{m}\xi^{(i)}$$

Note: the summation added to the cost is called **regularization term** preventing w to increase too much C is called **regularization parameter** (Like λ in GD basically)

Trade-off using C

By increasing C we penalize the model by increasing **precision** and **complexity** of the problem.

How does $\xi^{(i)}$ work

$\xi^{(i)}$ is proportional to how fare the misclassified pattern is from the hyperplane and if a misclassification error has been done, $\xi^{(i)} > 1$.

8.1.1 New optimization problem

Depending on C, we will penalize misclassification errors with a determined magnitude, consequentially the trade-off between complexity and admissible errors is determined by this parameter. The new **optimization problem** is defined as:

$$\min\frac{1}{2}|w|^2 + C\sum_{i=1}^{m}\xi^{(i)}$$

Such that $y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \xi^{(i)}$ where $\xi^{(i)} \geq 0$.

Lagrange coefficients (Dual Problem)

Due to the **regularization term**, the Lagrange coefficients will follow new conditions and defines a new **dual problem**:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i}^{m} \sum_{j}^{m} \alpha_i \alpha_j y_i y_j x_i^T x_j$$

Such that $0 \leq \alpha_i \leq C$ and $\sum_i^m \alpha_i y_i = 0$ bounding the variables to the parameter C.
Important: this solution may not guarantee good performance since the it may not be able to generalize well new data since the hyperplane may be too much restrictive (**dichotomy**) not being able to correctly represent the complexity of the data.

8.2 Kernel Operation

These are operations used by **Support Vector Machines** in order to systematically find **Support Vector Classifiers** in higher dimensions without doing the transformation into the higher dimension.

8.2.1 The Kernel trick

Basically, the **kernel operations** calculate the **geometrical relationships** like if the data were in a higher dimension, but in reality, they don't actually transform the data.
The Kernel Trick: it means evaluating the geometrical relationships among the observations, like we were on a higher dimension, without actually perform the transformation.
Given the right **kernel function** (or kernel trick) it's only implicit to project into a higher dimension.
If a **kernel function** is defined, it is a function such that:

$$K(x_i, x_j) = \Phi(x_i)\Phi(x_j) = \sum_{k=1}^{m} \Phi(x_i)\Phi(x_j)$$

Note: what we are really interested in is the product $\Phi(x_i)\Phi(x_j)$ but it doesn't mean we need to compute $\Phi(x_i)$ and $\Phi(x_j)$; so, just a function $k(x_i, x_j)$ which gives the same result is what we need.
Problem with the actual transformation: computational issues
Advantages
1) we reduce the amount of computation required for **Support Vector Machines** by avoiding the transformation from a low to a higher dimension.
2) It makes possible to perform the **Radial Kernel** which evaluates on infinite dimensions.

8.2.2 Cover's theorem

It has been proven that a **complex pattern-classification problem** (Which is non-linear in the **original** dimension) is likely to be separable if casted into a higher dimension.
Note: it is not guaranteed but the probability of a linear separation is increased.
Kernel trick: we won't actually transform every data, but it will be like we did

8.2.3 SVM with kernel

Given the assumptions of the **SVM with error tolerance**:

$$\max_{\alpha} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i}^{m} \sum_{j}^{m} \alpha_i \alpha_j y_i y_j k(x_i, x_j)$$

Where $0 \leq \alpha_i \leq C$ and $\sum_i^m \alpha_i y_i = 0$
Note: the product in the previous formula has been replaced by a proper **kernel function**

Parameter tuning

First) define the proper **kernel function**
Second) define the potential parameters of the **kernel function**
Third) define the value of the regularization parameter C

Advantages

First) no local minimums due to the quadratic relation
Second) The optimal solution can be found in a polynomial time (which is great considering, usually algorithms can also resolve in an exponential time)
Third) Few parameters to set up (C, **kernel** and **its parameters**)
Forth) Stable solution (No problem if initialization is random like **neural network**)
Fifth) Solution is **sparse** (Just involves support vectors)

8.2.4 Kernel functions – Examples

Linear kernel

$$(u * v)$$

Polynomial Kernel

$$(u * v + r)^d$$

Multi-Layer Perceptron tanh

$$\tanh(b(u * v) - c)$$

Radial basis function (RBF) kernel

$$e^{-\frac{|x_i - x_j|}{\sigma}}$$

Radial basis function (RBF) kernel

$$e^{-\frac{|x_i - x_j|^2}{2\sigma^2}}$$

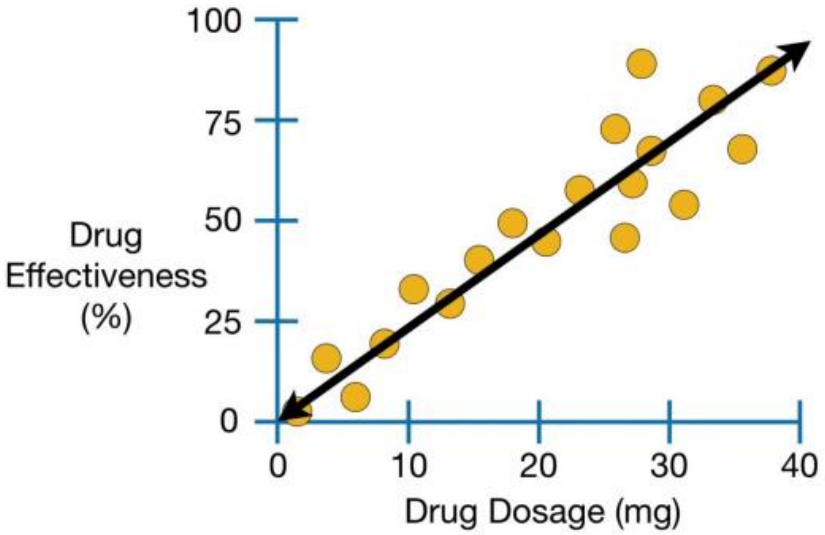
8.2.5 Mancano le slide dopo

Capitolo 9 Lezione 2022-11-28

9.1.1.1.1 7_Regression Trees.pdf
9.1.1.1.2 8_Classification Trees.pdf
9.1.1.1.3 10_(Random_Forest)_Missing Values.pdf

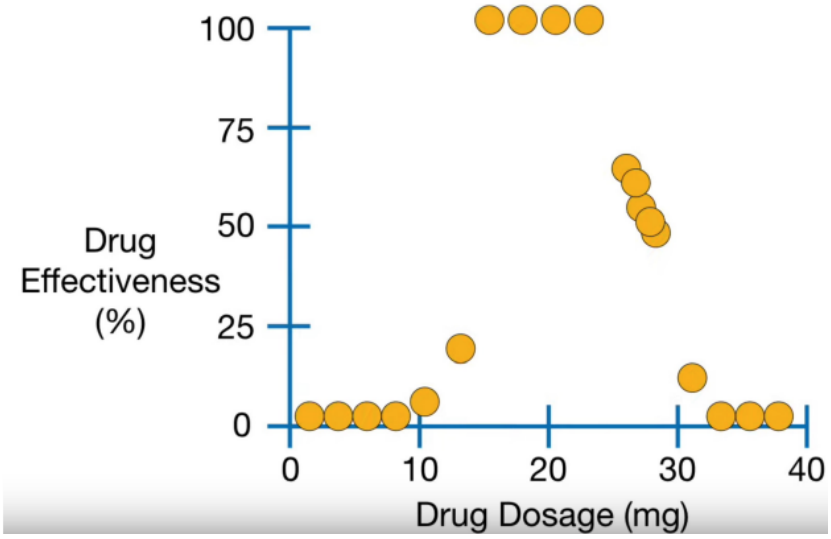
"***9_Regression Trees Pruning.pdf***" non l'ho messo perché nella lezione non ne parla

9.2 Regression tree (Intuition)



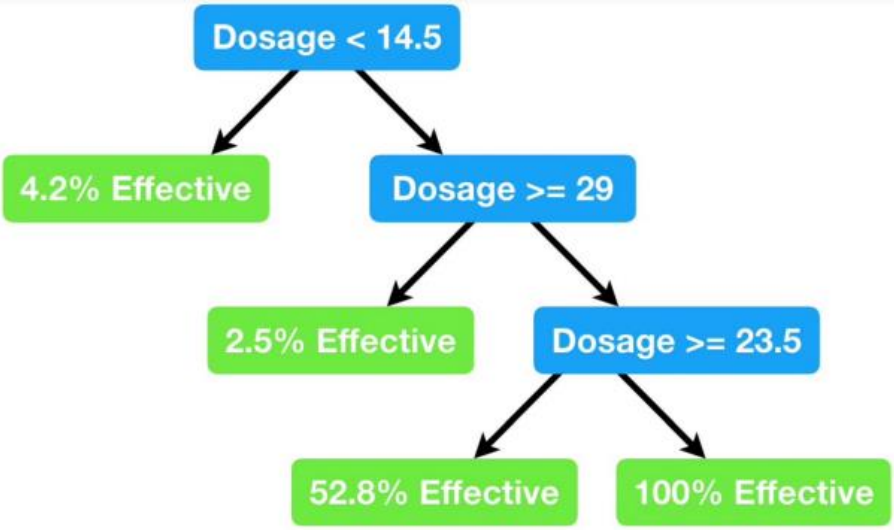
Suppose we have a dataset that looks like this where given an amount of **Drug Dosage** it is associated an **effectiveness** of the drug. Since the data seem to follow a linear relationship, **Linear Regression** can be used in order to predict new values of **Dosage**.

What if it was not the case?

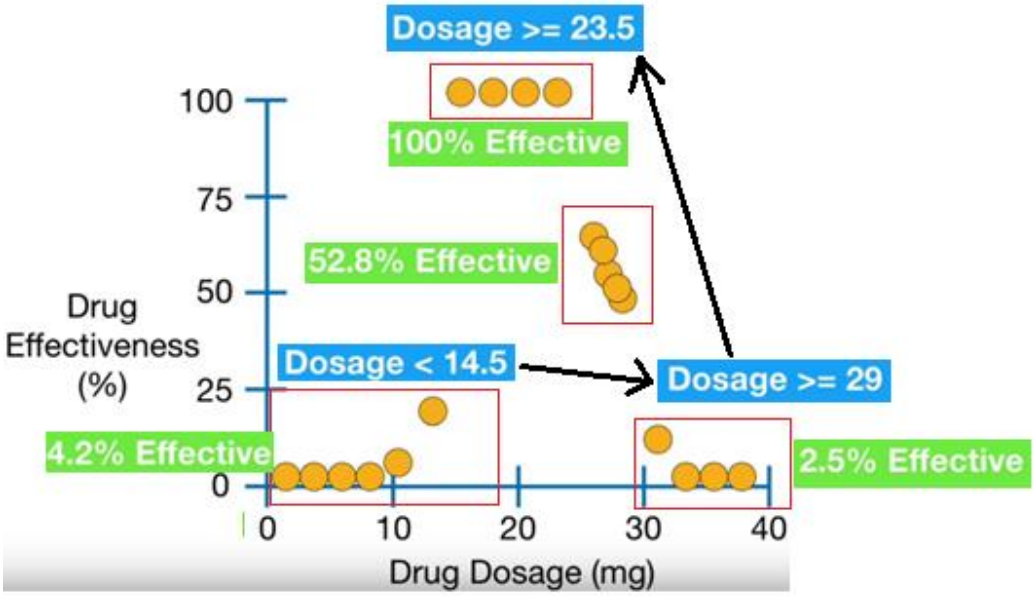


Suppose now that the data looks like this (It is not possible to predict the values using a **linear regression**)

9.3 Regression Tree (One variable)

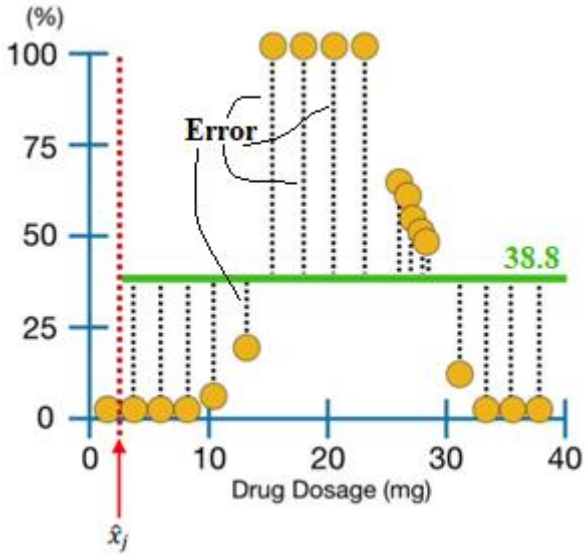


How to construct a regression tree

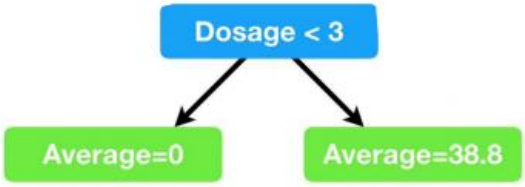


Regression trees use **mean values** of subsets of the original sub-dataset (called **split**).

9.3.1 Choosing the thresholds



We start by making partition of the graph (**Red line**), in the first iteration we take the average value of x_0 and x_1 as the threshold (It resulted $dosage = 3$ as the mean):
The **green line** is the average value of the partition, and each dot has a specific distance (which determines the error *aka* the dashed black line) from it.



We evaluate the **squared residual** (Called **Residual Sum of Squares** (**RSS**))

$$(0-0)^2 = 0$$

This is the value obtained by the **left partition**, since it is composed by one value, we expected a null impurity.

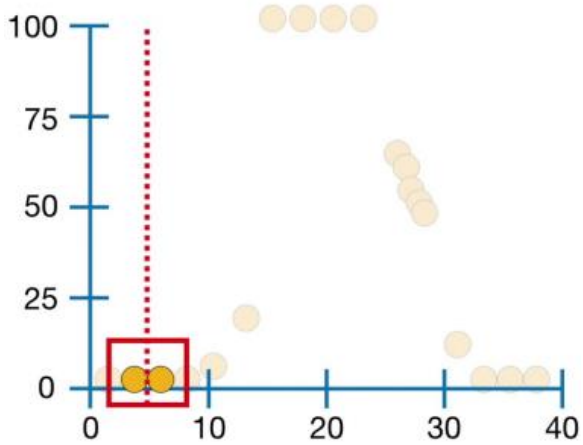
$$(0-38.8)^2 + (5-38.8)^2 - (20-38.8)^2 + (100-38.8)^2 + \cdots + (0-38.8)^2$$

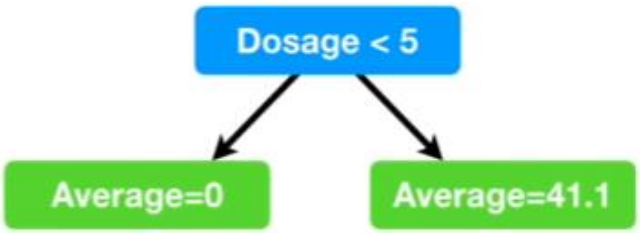
We are executing the **sum** of all the **right partition**.
The **RSS** is the whole sum:

$$RSS_1 = (0-0)^2 + (0-38.8)^2 + (5-38.8)^2 + \cdots + (0-38.8)^2 = 27468.5$$

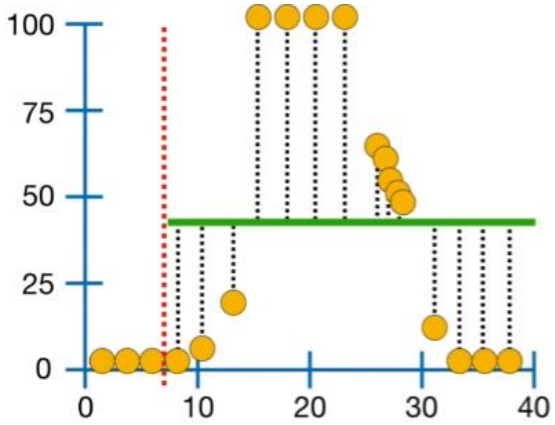
Next step

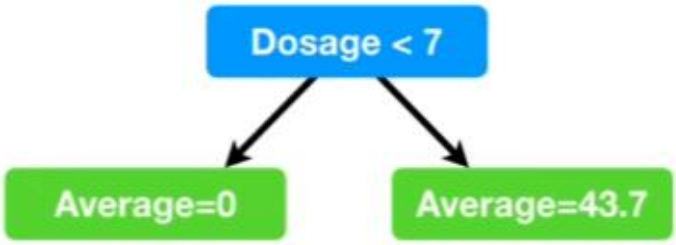
We repeat the same thing but by increasing the number of samples of the **left partition** by one (the **right partition** gets one sample smaller)





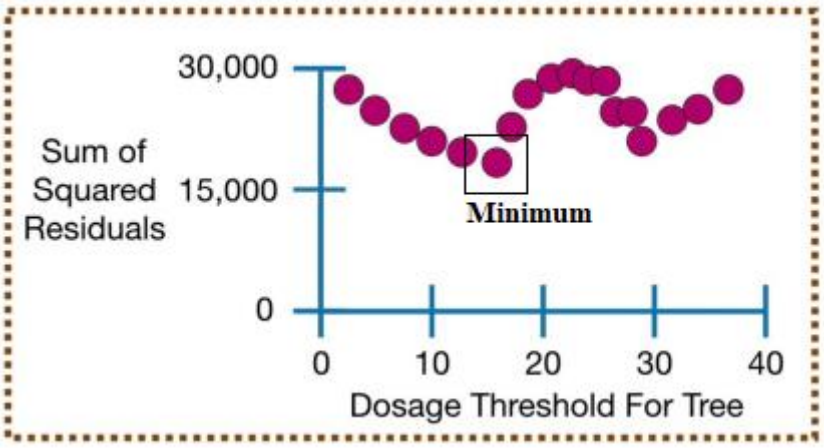
$$RSS_2 = 25222.7$$





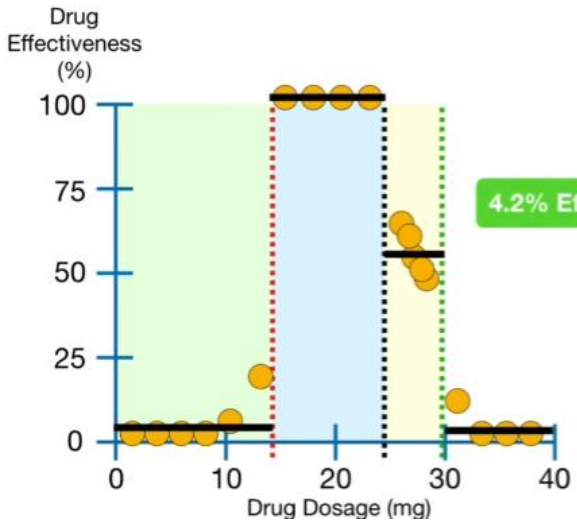
$RSS_3 = 25222.7$

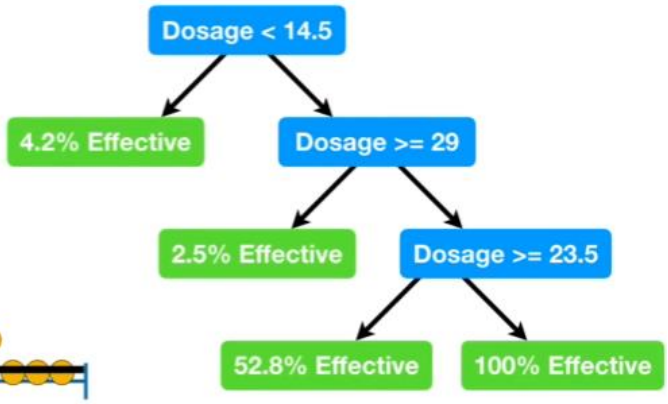
Graphing the RSS values



We find out that a minimum was based at $Dosage = 14.5$ and it will become **the root**.

We repeat the process for smaller partition



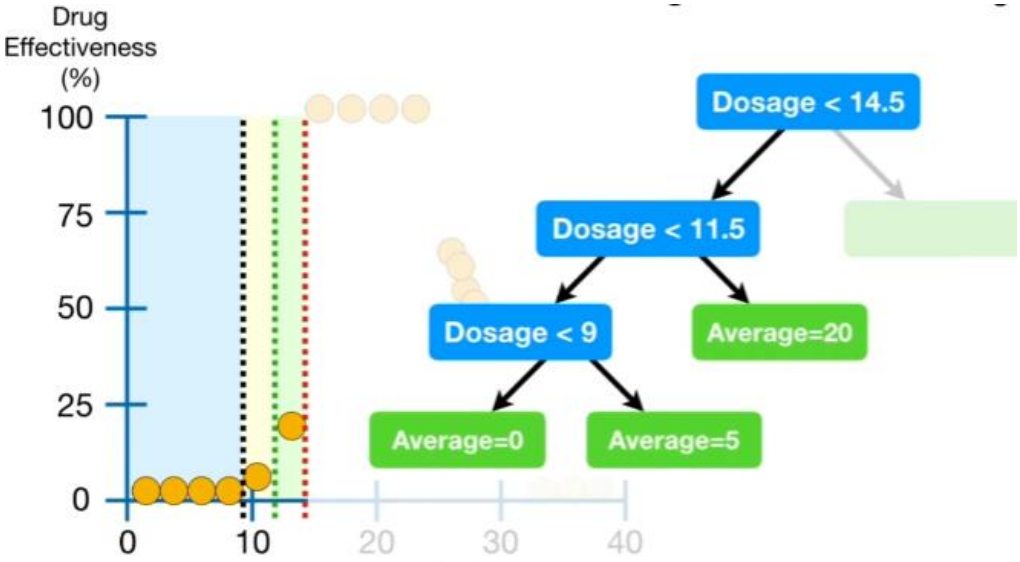


9.3.2 Prediction, RSS formula and overfit problem (rivedi)

$$h(x^{(i)}) = \sum_{\hat{x}_{j-1} < x^{(i)} < \hat{x}_j} \frac{y^{(i)}}{\left|\{x^{(i)} : \hat{x}_{j-1} < x^{(i)} < \hat{x}_j\}\right|}$$

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

Overfit problem

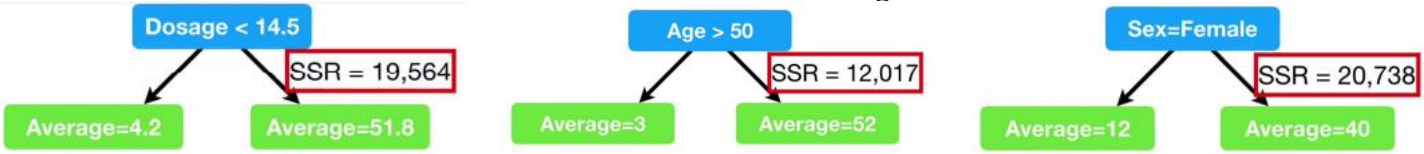


We should avoid making too small splits because we would face an **overfitting** problem.
Solution: set a minimum number of samples per split (Usually 20 but this number changes depending on the scenario)

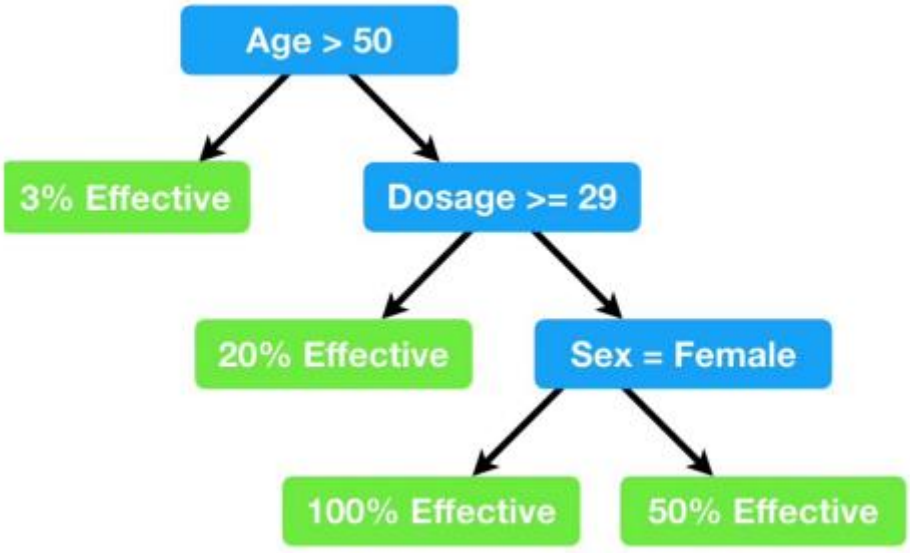
9.3.3 Multivariable regression tree

Dosage	Age	Sex	Drug Effect.
10	25	Female	98
20	73	Male	0
35	54	Female	6
5	12	Male	44
etc…	etc…	etc…	etc…

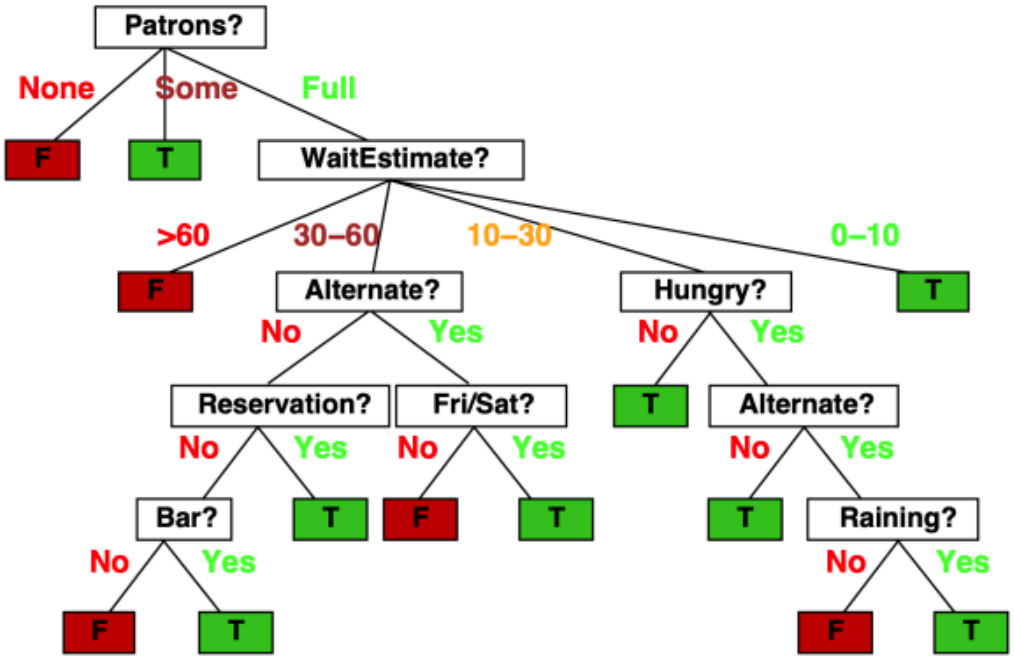
As we did before, we evaluate the best RSS for each of the categories



As we can see, we find out a value of $RSS = 12.017$ for $age = 50$ which will become the root of the tree.
The tree obtained by organizing the feature in a decreasing order of RSS



9.4 Decision Tree (Classification)



As we did for the regression tree, we are interested in a combination of conditions in order to obtain a model able to predict an output with a reasonable amount of precision.

Expressiveness

Decision trees describe a logical relationship between the hypothesis and the logical combination of the attributes. Any function of the input attributes, such as Booleans ones, can be expressed as a decision tree from the truth table row to the path to leaf.

Hypothesis space

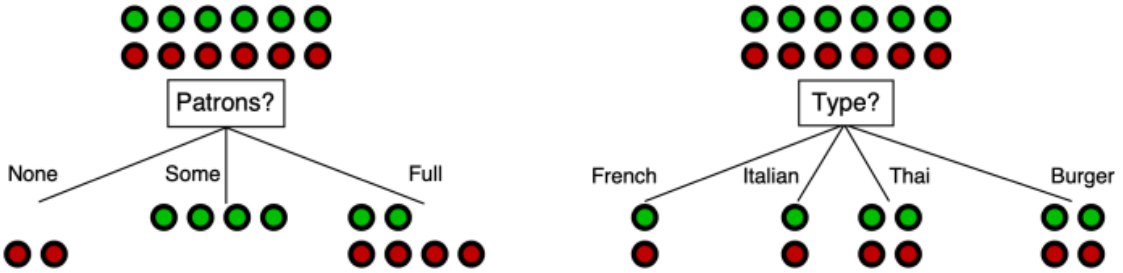
The number of possible decision trees with n attributes is equal to the number of possible Boolean functions with n attributes.

Given n attributes we have a truth table with 2^n rows and given that, we have 2^{2^n} possible Boolean functions.

9.4.1 Informativeness (Intuition)

Since the **hypothesis space** is composed by an enormous number of trees the question is if there is a way to optimize the search for the best solution using mathematics.

Important: we want the features which are the most significant ones in terms of information



As we can see in the example, **Patrons** seems to be a better feature in terms of information compared to **Type** because the input is divided into less class (**More general** which it is exactly what we search inside a ML model because it gives **information** about the classification problem)

We use the definition of entropy of signal processing in order to determine the best information:

$$\log_2 n = -\log_2 \frac{1}{n}$$

Entropy is evaluated by the number of bits (if we have n events they need to be represented by $\log_2 n$ bits.

$-\log_2 \frac{1}{n}$ is the information needed to represent the events:

$$-\log_2 \frac{1}{n} = -\sum_{(n\ events)} \frac{1}{n} * \log_2 \frac{1}{n}$$

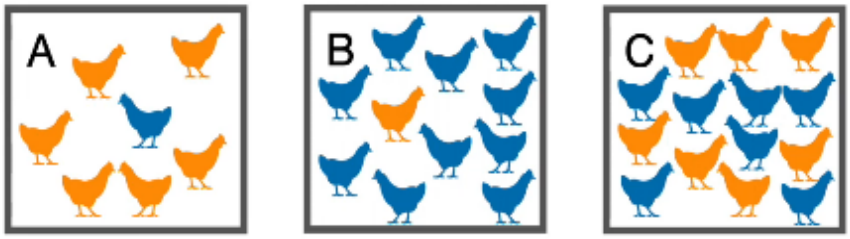
In case we had a uniform distribution, the probability of the event would be $1/n$.

Generalized form:

$$-\sum_{(n\ events)} \frac{1}{n} * \log_2 \frac{1}{n} \rightarrow -\sum_{i=1}^{n} p_i * \log_2 p_i$$

9.4.2 Surprise (Intuition) (All is extra till 9.4.3 paragraph)

Imagine observing at these chicken groups:



Group A) if we pick a blue chicken, we would be surprised since they are mostly orange

Group B) if we pick a blue chicken, we would not be surprised because the majority of them are blue, we would have been surprised if we had picked the orange one.

Group C) either blue or orange, we will be relatively surprised since there is an equal number of blue and orange chickens.

Consideration: there is a sort of inverse proportionality between probability and surprise because if we consider **group A**, we will be surprised to pick a blue one which has the lowest probability, vice versa we would not be surprised to pick an orange one which has the highest probability.

$$Surprise = \frac{1}{probability}$$

This is the first guess for the surprise.

Why this is not the form of surprise?

Suppose we have a coin which has a probability of heads of 100%

$$S = \frac{1}{P(heads)} = \frac{1}{1} = 1$$

But we would like to have 0 instead since we are not surprised if the result would be head again.

$$S_H = \log_2 \left(\frac{1}{P(heads)}\right) = \log_2 \left(\frac{1}{1}\right) = 0$$

By taking the $\log(x)$, we obtain what we expected, 0 surprise.

If we want to evaluate the surprise of a tail:

$$S_T = \log_2 \left(\frac{1}{P(tails)}\right) = \log_2 \left(\frac{1}{0}\right) = Undefined$$

We cannot define the surprise of a given event that has never happened, and this makes sense

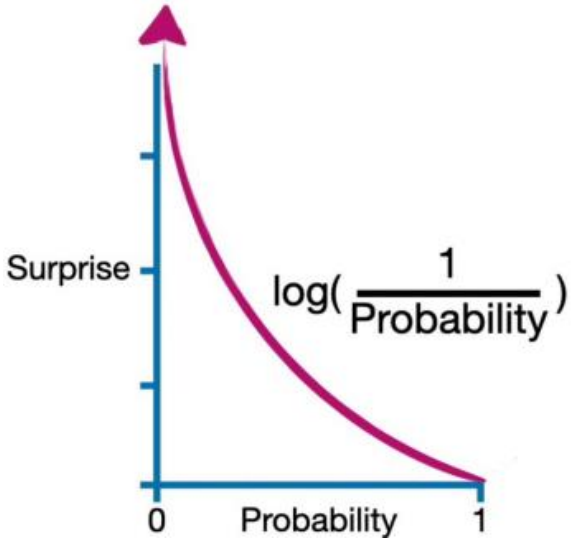
Surprise

After some considerations, we defined the formula for **surprise**:

$$Surprise = \log_2\left(\frac{1}{P(X=x)}\right)$$

$$Surprise = -\log_2\big(P(X=x)\big)$$

The last one follows by using $\log(x)$ properties



Total Surprise (surprise of a sequence of events)

Supposing that we want to evaluate the surprise of the following sequence:

The probability of $P(heads) = 0.9$ and $P(tails) = 0.1$

Since the throws were independent, we know that the probability of this event is:

$$P_E = P_H * P_H * P_T = 0.9 * 0.9 * 0.1 = 0.018$$

In order to evaluate the **surprise** of this event, we can determine it by using the formula:

$$S_E = -\log_2(P_E) = -\log_2(P_H * P_H * P_T)$$

Thanks to $\log(x)$ properties, we can rewrite the whole surprise as:

$$S_E = -\log_2(P_H * P_H * P_T) = -2\log_2(P_H) - \log_2(P_T) = 2S_H + S_T$$

$$S_H = -\log_2(P_H) = 0.15$$

$$S_T = -\log_2(P_T) = 3.32$$

In the end we obtain:

$$S_E = 2S_H + S_T = 2(0.15) + 3.32 = 3.62$$

Property: the **total surprise** of a sequence of events is the sum of the surprises of each event.

9.4.3 Entropy

We want to evaluate how much surprise we would get on average given a n number of events, we know that each surprise S_i has a probability associated $P_i(X = x) = p_i(x)$.
The entropy is then defined as the estimated value of the **surprise** (a mean value):

$$E[S] = -\sum_{i=1}^{n} p_i(x)\log_2\big(p_i(x)\big)$$

Useless note: this is the equation for **entropy** first published in 1948 by Clause Shannon.

Notation

$$H\left(\frac{t}{t+f},\frac{f}{t+f}\right) = -\frac{t}{t+f}\log_2\left(\frac{t}{t+f}\right) - \frac{f}{t+f}\log_2\left(\frac{f}{t+f}\right)$$

This is the notation used by the professor where H refers to the entropy (In this case f and t are the F and T of the **target feature**).
The dataset is composed by 12 rows $(t + f = 12)$ and $t = 6, f = 6$ so they are evenly distributed.

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Supposing that we are evaluating the **entropy** for the variable **Pat** (Patrons)
As we can see we have divided the dataset into ($\boldsymbol{Some}, \boldsymbol{None}, \boldsymbol{Full}$)
We need to evaluate the entropy for each subset:

$$R(\boldsymbol{Some}) = \frac{t_S + f_S}{t+f} H\left(\frac{t_S}{t_S+f_S},\frac{f_S}{t_S+f_S}\right)$$

$$R(\boldsymbol{None}) = \frac{t_N + f_N}{t+f} H\left(\frac{t_N}{t_N+f_N},\frac{f_N}{t_N+f_N}\right)$$

$$R(\boldsymbol{Full}) = \frac{t_F + f_F}{t+f} H\left(\frac{t_F}{t_F+f_F},\frac{f_F}{t_F+f_F}\right)$$

What do these formulas mean?

We know that for a given subclass (Such as $\boldsymbol{None}$) we have a specific number of given results t_N, f_N and a total number of events which is their sum $t_N + f_N$.
The entropy inside the **subset** $\boldsymbol{None}$ is evaluated by:

$$H\left(\frac{t_N}{t_N+f_N},\frac{f_N}{t_N+f_N}\right)$$

But since the **main goal** is to evaluate the entropy of **patrons** ($\boldsymbol{None}$ is a subclass of this feature) we need to perform a **weighted sum** of the entropy of each subclass ($\boldsymbol{None}$, $\boldsymbol{Some}$ and $\boldsymbol{Full}$)
$\frac{t_N+f_N}{t+f}$ is the associated weight for the entropy of $\boldsymbol{None}$ (total number of events of $\boldsymbol{None}$ $[t_N + f_N]$ divided by the total number of rows inside the dataset $[t + f]$)

Procedure

1) We evaluate the entropy of subclasses inside a feature (in this case **Patrons**):

$$H(\boldsymbol{Some}) \,||\, H(\boldsymbol{None}) \,||\, H(\boldsymbol{Full})$$

2) We perform the weighted average in order to obtain the entropy of the original variable:

$$H(\boldsymbol{Patrons}) = \frac{t_S + f_S}{t+f} H(\boldsymbol{Some}) + \frac{t_N + f_N}{t+f} H(\boldsymbol{None}) + \frac{t_F + f_F}{t+f} H(\boldsymbol{Full})$$

Which can be also expressed as:

$$H(\boldsymbol{Patrons}) = R(\boldsymbol{Some}) + R(\boldsymbol{None}) + R(\boldsymbol{Full})$$

3) Repeat this process for each feature

Note for notation: the last formula for $H(\boldsymbol{Patrons})$ is the one used by the professor.

9.4.4 Information gain

If we reach this point, we should have evaluated every entropy of every variable.
Recall that the entropy of the **target variable**:

$$H(\boldsymbol{target}) = H\left(\frac{6}{12}, \frac{6}{12}\right) = H\left(\frac{1}{2}, \frac{1}{2}\right) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1$$

So $H(\boldsymbol{target}) = 1$.
The information gain is defined as:

$$I_G(\boldsymbol{Feature}) = H(\boldsymbol{Target}) - H(\boldsymbol{Feature})$$

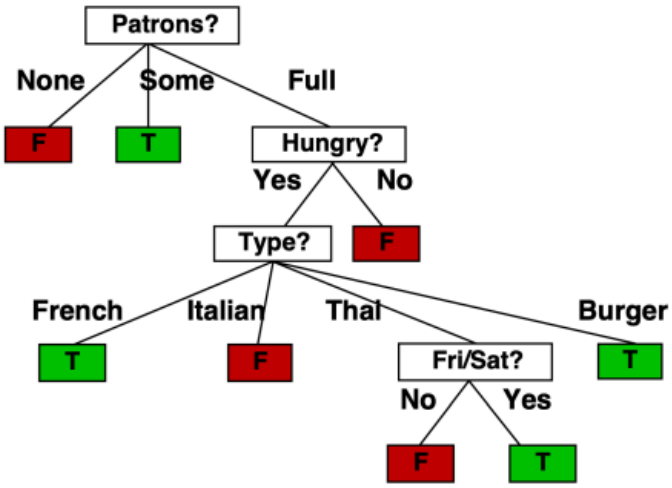
We choose the variable with the highest information gain given by this result:
Supposing that $I_G(\boldsymbol{Patrons}) = 0.541$ and $I_G(\boldsymbol{Type}) = 0$
We would pick $\boldsymbol{Patrons}$ instead of $\boldsymbol{Type}$.

What branch do we choose?

Since we have chosen $\boldsymbol{Patrons}$ as the main root, we would like to decide which is the best branch among the subclasses $\boldsymbol{Some}, \boldsymbol{None}, \boldsymbol{Full}$.
In order to do that we compare the entropy of each subclass inside **Patrons,** and we choose the one with the highest information gain:

$$I_G(\boldsymbol{Some}) = H(\boldsymbol{Patrons}) - R(\boldsymbol{Some})$$
$$I_G(\boldsymbol{None}) = H(\boldsymbol{Patrons}) - R(\boldsymbol{None})$$
$$I_G(\boldsymbol{Full}) = H(\boldsymbol{Patrons}) - R(\boldsymbol{Full})$$

Possible decision tree



Note: we choose the most informative features (We want to avoid **overfitting** which is something that usually occurs when you use too much features that are not enough informative)

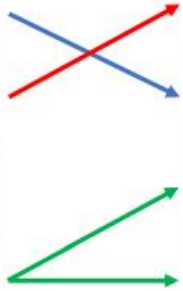
9.5 Random Forest (and how to generate it)

The random forest is a "*countermeasure*" for the **inaccuracy** generated by decision trees since they work well on the data that generate them (not flexible for new data).
The idea of random forest is to combine the simplicity of decision trees by adding flexibility to the model resulting in a better accuracy.

1st step – **Generate a bootstrapped dataset**

Suppose we start from a dataset of a given dimension, in order to create the **bootstrapped** version of our dataset we randomly select samples from the original dataset allowing to pick the same sample multiple time

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes



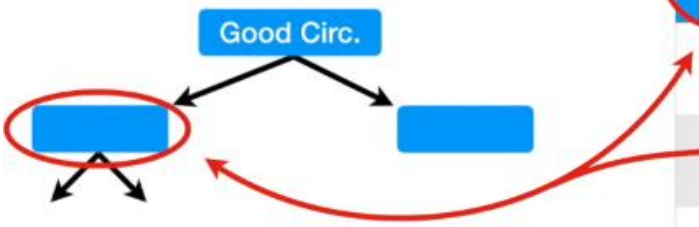
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

2nd step – **Create a decision tree based on the bootstrap dataset**

We will use random subset of variables (columns) at each step.
Note: this pick will later be optimized (Number of columns and which ones)

For the example, suppose that we randomly pick **Good Blood Circ.** and **Blocked Artesies** competing for the **main root**. We also find that **Good Blood Circ.** did the best job into separating the data.

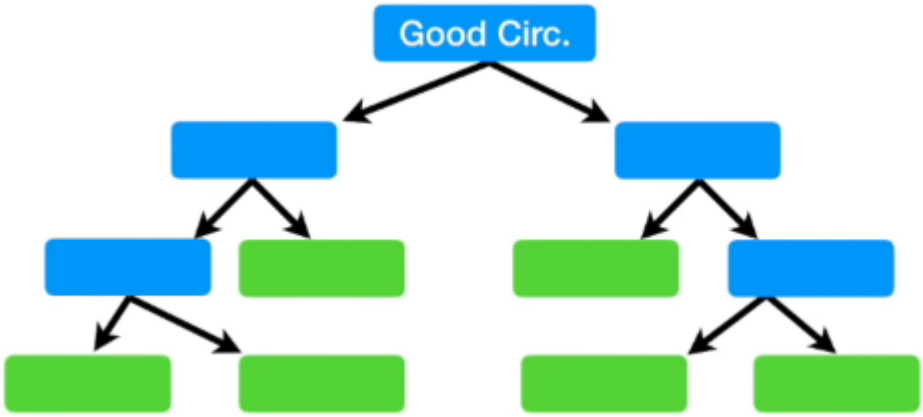
3rd step – **Going futher into the tree**



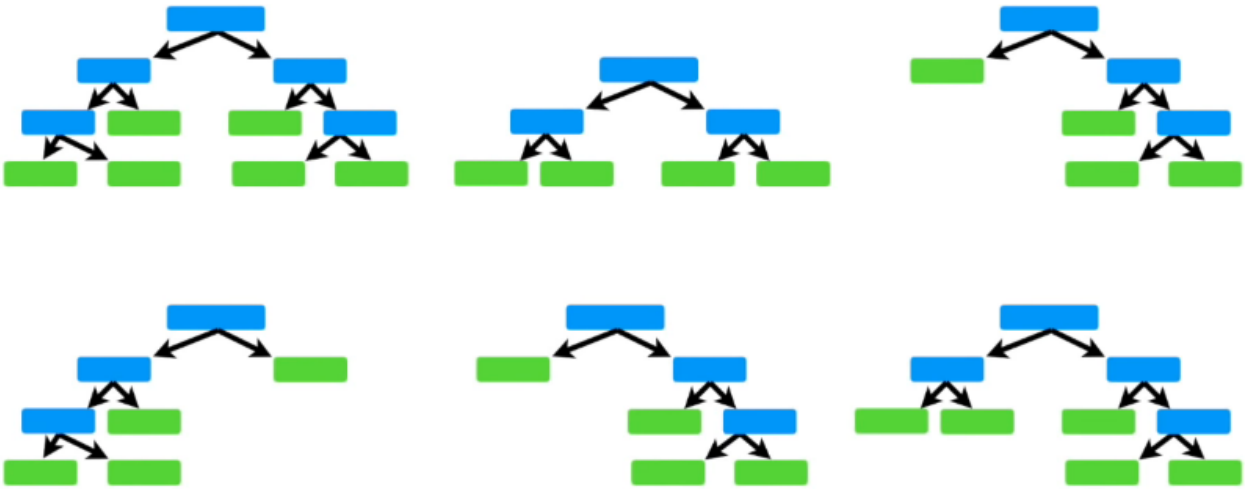
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

We would like to know how to continue this three and in order to do that we randomly select 2 other columns (**Good Blood Circ.** is excluded because it has been already chosen as the **main root**)

4th step – **Continue building the tree with the same idea of random selection of features**



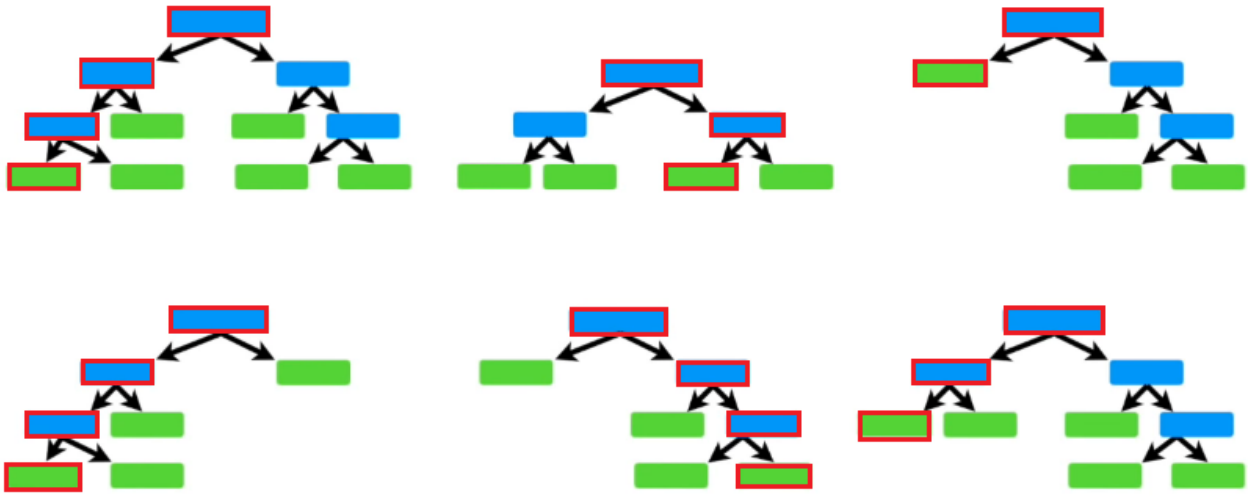
5th step – **Go back to the 1st step and generate new of trees**



Note: the number of trees that we can generate vary depending on the scenario (dataset size, etc) but in general this process is done hundreds of time.

9.5.1 How to use a random forest (Bagging)

We take a new sample and we ran in into each decision tree inside the random forest



After running the data we observe the most common output from each tree.

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	No	No	168	YES

Since **Yes** was the most common result we conclude that data associated with the sample have been categorized with an **Head Disease** with value of **Yes**.

Bagging: is the action of **bootstrapping** the data and **aggregate** them to make a decision

9.5.2 Evaluation of the accuracy (Out-of-Bag Error)

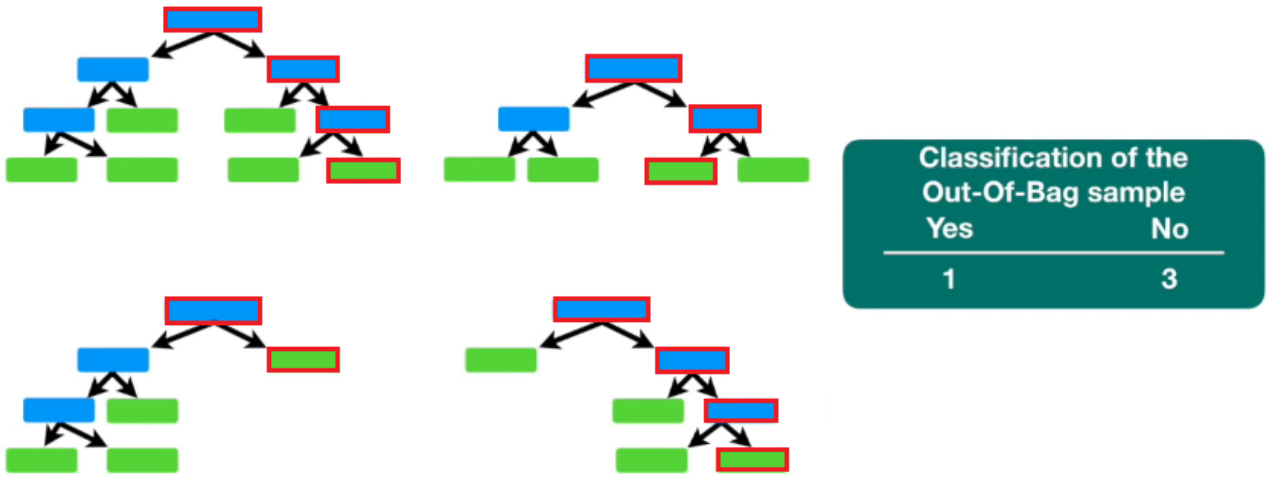
At the start of the example we haven't considered some samples inside the dataset.

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	Yes	167	Yes

Note: if the dataset had been larger, we would have got more that just a sample but usually the samples that are not been considered are around $1/3$ of the original dataset**.**
Important: the not considered samples create the **out-of-bag dataset**.

Evaluation of the error

Since the samples inside the **out-of-bag** have not been considered during the training, they can be used in order to evaluate the error.



We repeat this process for each **Out-of-Bag sample** and we evaluate the error based on the number of misclassification.

9.5.3 General scheme for a random forest

1) Build a random forest
2) Estimate the accuracy of the Random forest
3) Change the setting of the random forest
4) Return to (1) and generate different random forestes until you we are satisfied
5) Choose the most accurate random forest with the smallest **Out-of-Bag error**

9.6 Missing values inside a random forest

The random forest recognizes two types of missing data

9.6.1 Missing data in the original dataset (Iterative method)

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	???	???	No

Most common value

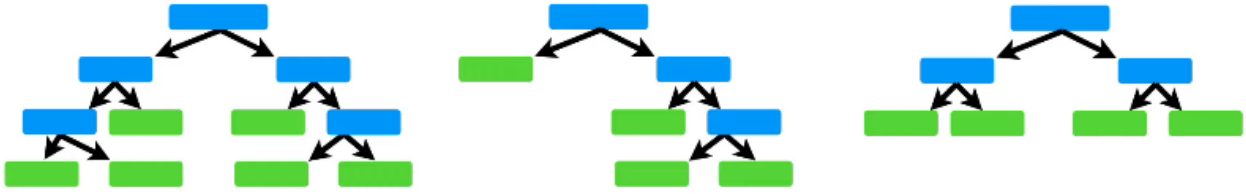
Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	167.5	No

Blocked arteries is binary, so the most common class will be chosen
Weight is a value, the average of the column will be chosen

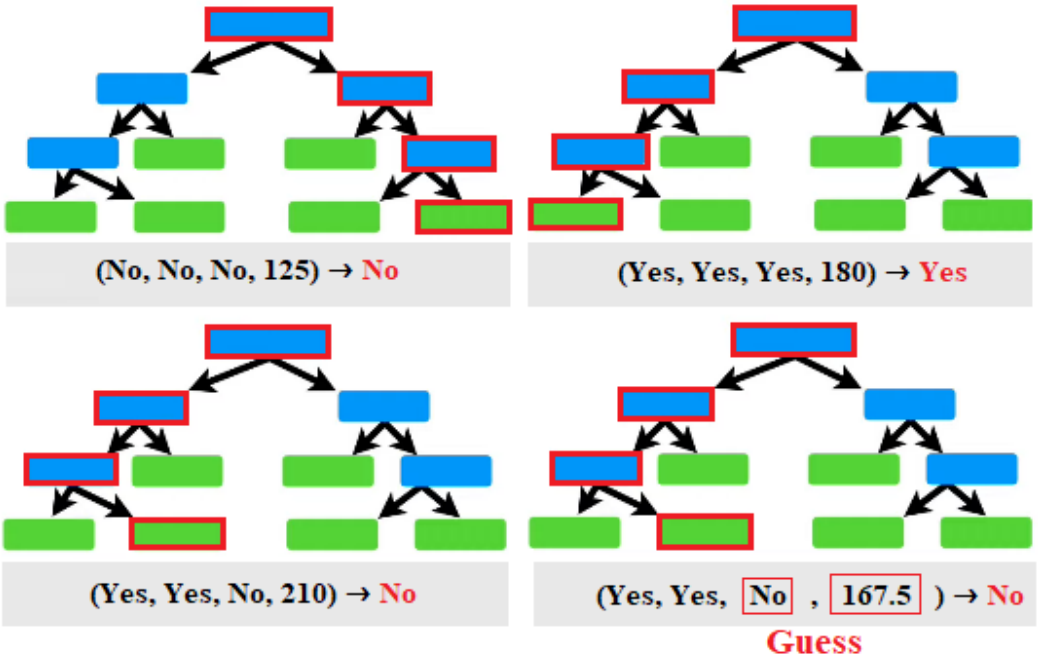
Check similarities among samples

We want to refine our guesses by checking similarity of the "**missing value sample**" with other samples.

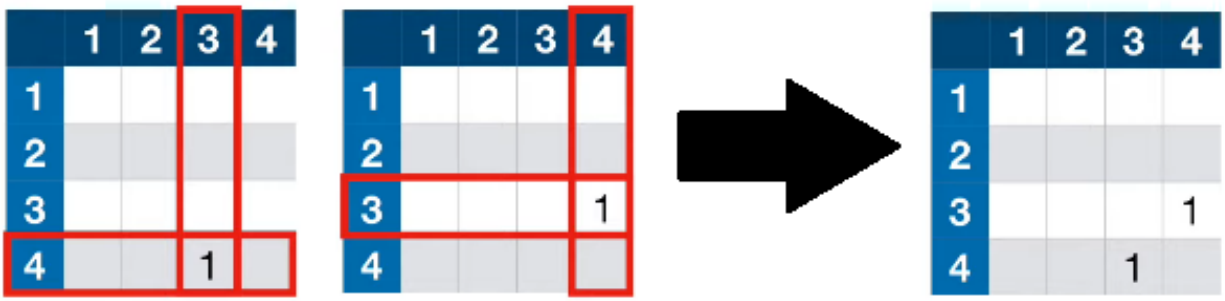
First) we create a random forest:



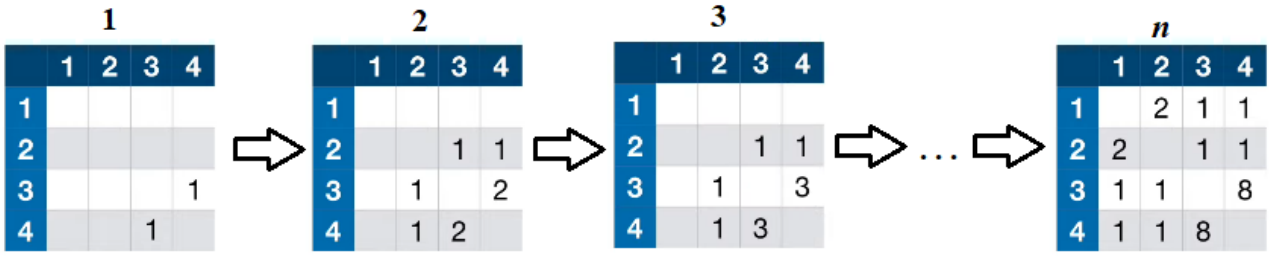
Second) we run all the data on the first tree



Consideration: both sample **3** and **4** landed in the same leaf, so they are "**similar**"
We keep track of similarities by using a proximity matrix
This matrix has a row for each sample and the same for each column.



Since $i = 3$ and $j = 4$ are "**similar**" we track by adding a 1 (the same happens if we consider $i = 4$ and $j = 3$ since the matrix is symmetric) and we end up with the matrix on the right.
Third) we continue with the next decision tree inside the random forest and we update the proximity matrix each iteration:



Value refinement

We divide the numbers inside the **proximity matrix** with n (for the example $n = 10$)

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

Regression value case

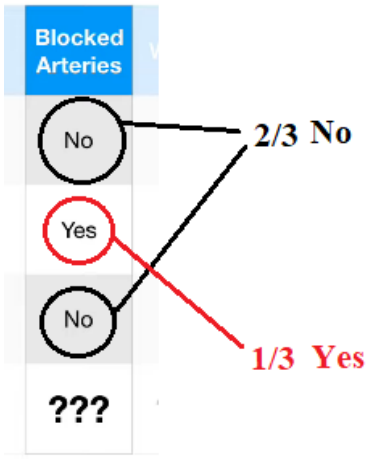
For the regression case, we can consider every weight of each sample and we perform a **weighted average**:

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	125	No
Yes	Yes	Yes	180	Yes
Yes	Yes	No	210	No
Yes	No	???	???	No

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

$$Weight = 125\frac{\mathbf{0.1}}{0.1 + 0.1 + 0.8} + 180\frac{\mathbf{0.1}}{0.1 + 0.1 + 0.8} + 210\frac{\mathbf{0.8}}{0.1 + 0.1 + 0.8} = 198.5$$

Binary case



The frequency of "**Yes**" and "**No**" can be determined by the number of occurances divided by the total number of rows. (It is like the evaluation of a probability, watch the image above)
Then, we need to multiply the frequency with their corrisponding weight of proximity:

$$Yes = F_{Yes} * W_{Yes}$$
$$No = F_{No} * W_{No}$$

In order to evaluate the weights W_{Yes} and W_{No} we use the **proximity matrix**:

Yes case)

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

As we can see from the **original dataset**, the only sample who has a $\boldsymbol{Yes}$ value is the number 2

$$W_{Yes} = \frac{0.1}{0.1 + 0.1 + 0.8} = 0.1$$

No case)

	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

From **original dataset**, the samples who have a $\boldsymbol{No}$ value are the numbers 1 and 3:

$$W_{No} = \frac{0.1 + 0.8}{0.1 + 0.1 + 0.8} = 0.9$$

Comparison)

$$\boldsymbol{Yes} = F_{Yes} * W_{Yes} = \frac{1}{3} * 0.1 = 0.03\overline{3}$$
$$\boldsymbol{No} = F_{No} * W_{No} = \frac{2}{3} * 0.9 = 0.6$$

Since the $\boldsymbol{No}$ has a bigger value than $\boldsymbol{Yes}$ our initial prediction seems to be fine, so we confirm No

Update the missing value with the refined ones

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	No	198.5	No

9.6.2 Important note about value refining

In order to obtain a **good prediction** of the missing values, we repeat this process with new random forestes until a convergence is reached (Usually 6 – 7 times is enough)

9.6.3 Missing data in a new sample (Not categorized) (ERROR IMG)

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
No	No	No	???	

First) we imagine that we have a **trained random forest**.
We make two copies of the same sample where we put two different outcome of the **heard disease**.

	Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
1	No	No	No	???	YES
2	No	No	No	???	NO

Second) we use the **iterative method** for **missing data inside the original dataset** in order to obtain good guesses.

	Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
1	No	No	No	YES	YES
2	No	No	No	NO	NO

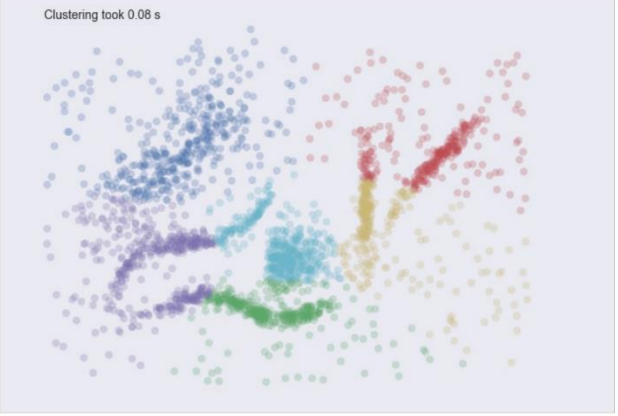
Third) we run these samples inside the trees in the forest and we take **the most corretly label**.
Following the example:
Yes: The sample (1) has been correctly labelled 3 times
No: The sample (2) has been corrected labeled 1 time
Due to this result, we consider the sample (1) as the winner

Capitolo 10 Lezione 2022-12-05

10.1.1.1.1 11_Clustering.pdf

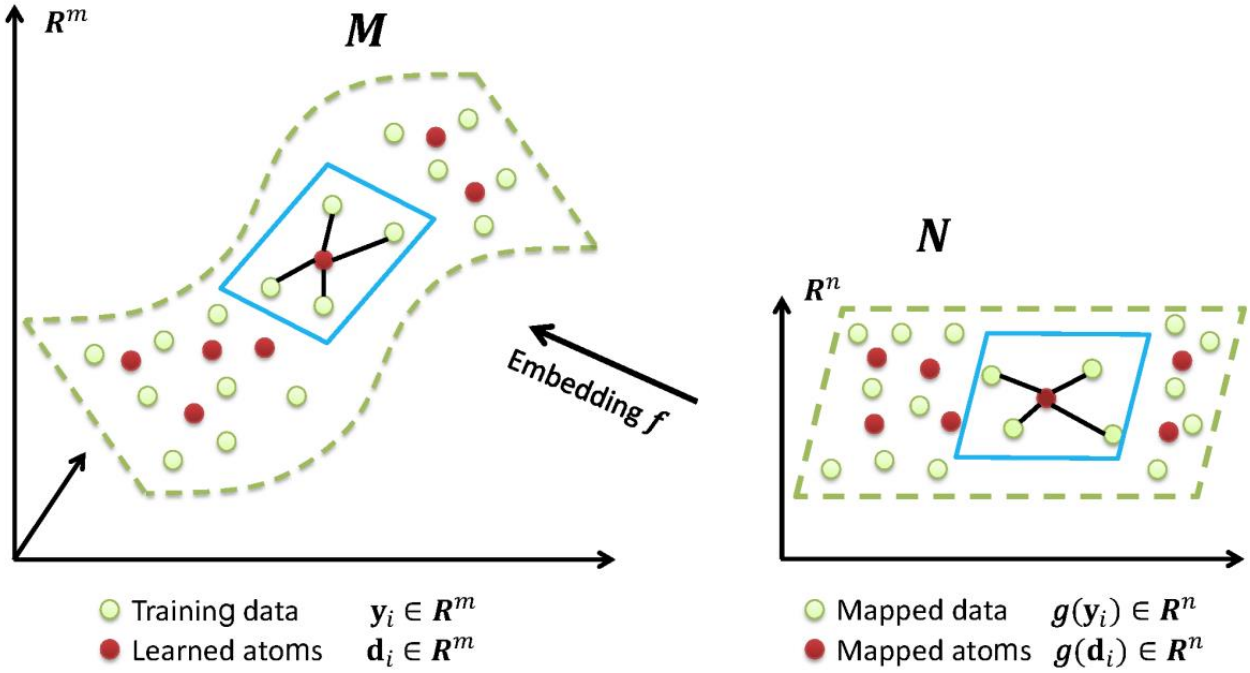
10.1 Unsupervised learning

10.1.1 Clustering



In the unsupervised learning, we want to classify similar data and the way we are doing this is by **clustering** data; by that we mean group similar items (data) with similarities.
So essentially clusters are groups of items which are similar (or close) to each other.
As you can see from the pictures, clusters may appear different from what we expect and this is clear by watching at the yellow and red clusters.
They are not predeterministic (different algorithms mean different results, even with the same algorithm the result may be different)
Important: computers are not able to recognize clusters by themselves so we need to create algorithms able to do this.

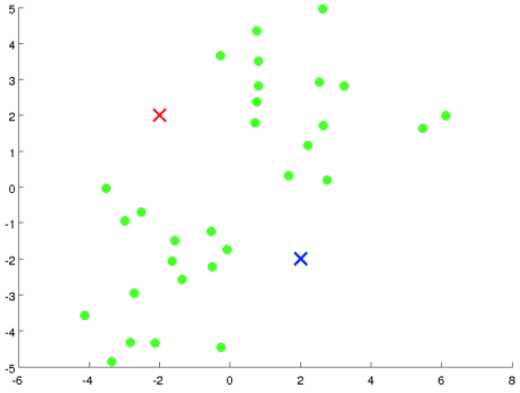
10.1.2 Dimensionality reduction



By dimensionality reduction we mean the action of reducing some of the features of the original problem in order to reduce the dimension of the original one by considering the most important features. the complexity of the overall problem is reduced at the cost of losing information on the features we are not considering.

Compressing: this is different compared to discarding features because we are trying to elaborate the original features in order to reduce the dimension and obtain a similar information of the problem we started with. It is different from "**discarding**" because we are not completely ignoring some features.

10.2 K-means (By iterations)



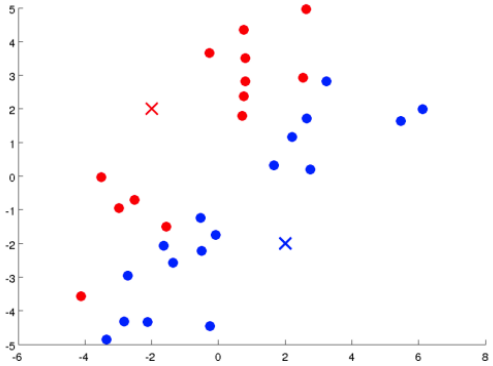
The **k-means** is an algorithm which has the purpose of creating clusters of a set of data; the main idea is to set a hypothetical number of clusters k, guess the initial center of the clusters (**centroids**) and then iterate with new centroids that are the new geometrical centers of the clusters obtained.

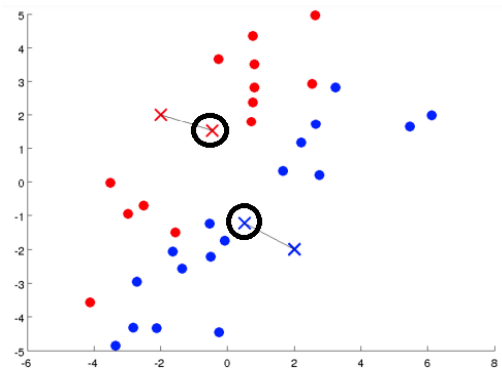
In the picture, the data are represented by the **green dots** and the first guess are the **red** and **blue crosses**

We imagine a problem with just 2 clusters so $k = 2$

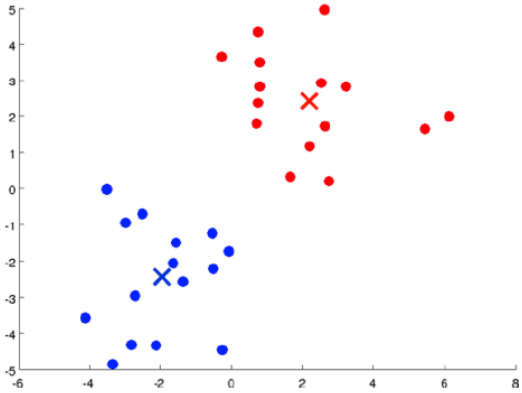
How do we determine which point for which cluster?

A generic point $x^{(i)}$ will belong to the cluster $c^{(i)}$ if the distance from the center of cluster $\mu^{(i)}$ is the closest one compared to the other centroids $\mu_1, \dots, \mu_k$.





After finding the **new centroids**, we repeat the process until convergence.



Consideration: during the process, some of the points may shift from a cluster to another one.

Random initialization (recap)

1) Set a number of clusters $k < m$
2) Randomly pick k training examples as the centroids

10.2.1 Cost function (Search for the best clusters)

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k) = \frac{1}{m}\sum_{i=1}^{m} \left|\left|x^{(i)} - \mu_{c^{(i)}}\right|\right|^2$$

Note: it is the overall variance generated by the points from their respective centroids.

Best clusters (and centroids)

Since the algorithm is dependent from the initial conditions (the first **centroids**) in order to avoid that we repeat this algorithm multiple times (Like 100) where we randomly pick new centroids and iterate etc.
Pick condition: the one with the lowest value of $J(\theta)$, this process aims on finding k centroids such that the overall variance of the points x from its centroids μ is the smallest possible.

10.2.2 Pro and cons

Its "**main usage**" is to get an idea of what is going on since it is very fast; depending on the scenario, different algorithms (which are better) are used for a detailed search of the clusters.

Pro) It is a simple algorithm and quite efficient computationally that always **terminates**.

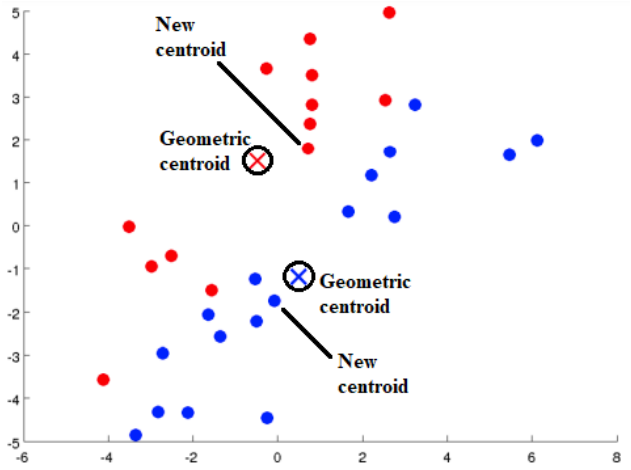
Con) The k it is not specified (**overfit** if k is too large and **underfit** if it is too small)
Con) Non-numeric values cannot be used (Such as categorical ones)
Con) The algorithm does not guarantee to find the **optimal clusters**
Con) It is sensible to **outliers** and **noise**.

10.2.3 K-medoids (Riguarda per sicurezza)

It is basically a **k-means** at the base with a difference:
The k-means will likely choose a centroid which is not one of the data of the original dataset but for the k-medoids we impose that every iteration will choose a centroid corresponding to a point in the dataset.

How do we perform that?

First) we choose *k* centroids from the data
Second) we determine which point belong to which cluster (the closest distance between a centroid and a point like the k-means)
Third) We determine the **geometrical centroids**. The **new centroids** will be the closest points to the geometrical centroids.



Pro and cons

Pro) less sensitive to outliers (than k-means)

Con) Optimal clusters not guaranteed
Con) Categorical values still not applicable
Con) Still needs *k* to be specified
Con) Still sensible to noise and outliers (we have just reduced)
Con) Fails for non-linear dataset
Con) More expensive than k-means

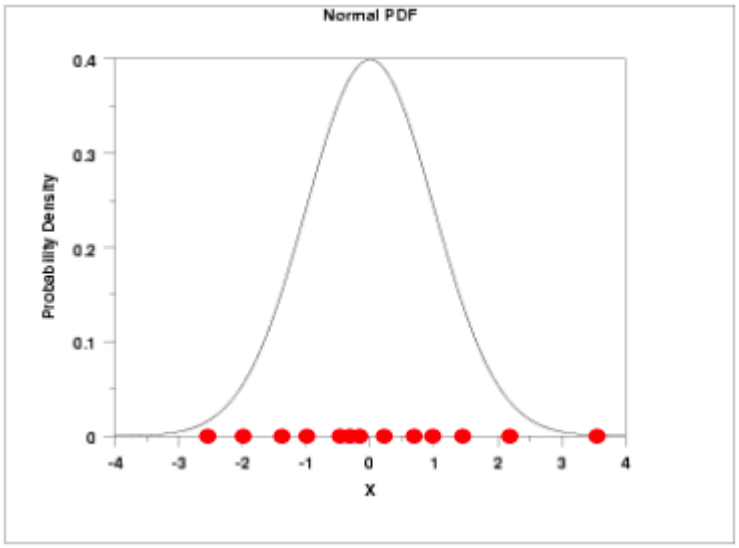
10.3 IMPORTANT

This is not the PAM implementation that the professors asks at exam, check it out on the slide.
Also, learn the Pseudo-codes since he may ask them.

10.4 Gaussian Mixture Models (Rivedi dalle slide, non ne sono certo)

First, a little recap of a **Gaussian** (or normal) **distribution**:
$x \sim N(\mu, \sigma^2)$ states that x can be represented by a normal distribution were
Mean value: μ
Variance: σ^2

$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$



We may decide to model our clusters using a series of gaussian distributions.

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} \quad \Big|\Big| \quad \sigma^2 = \frac{1}{m}\sum_{i=1}^{m} \left(x^{(i)} - \mu\right)^2$$

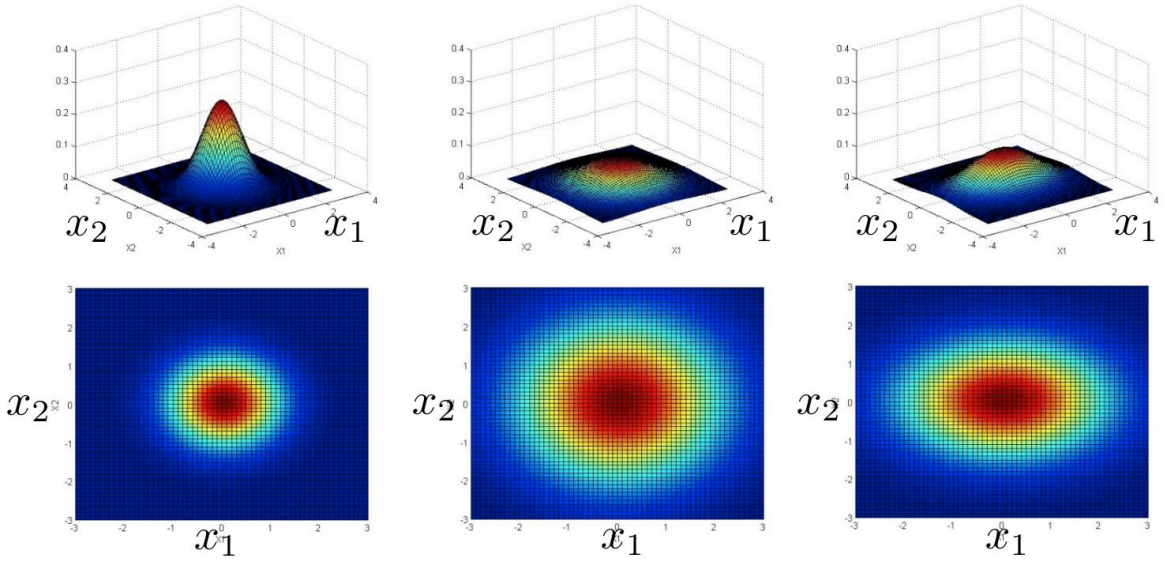
10.4.1 Multivariate gaussian mixture

$$p(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)}$$

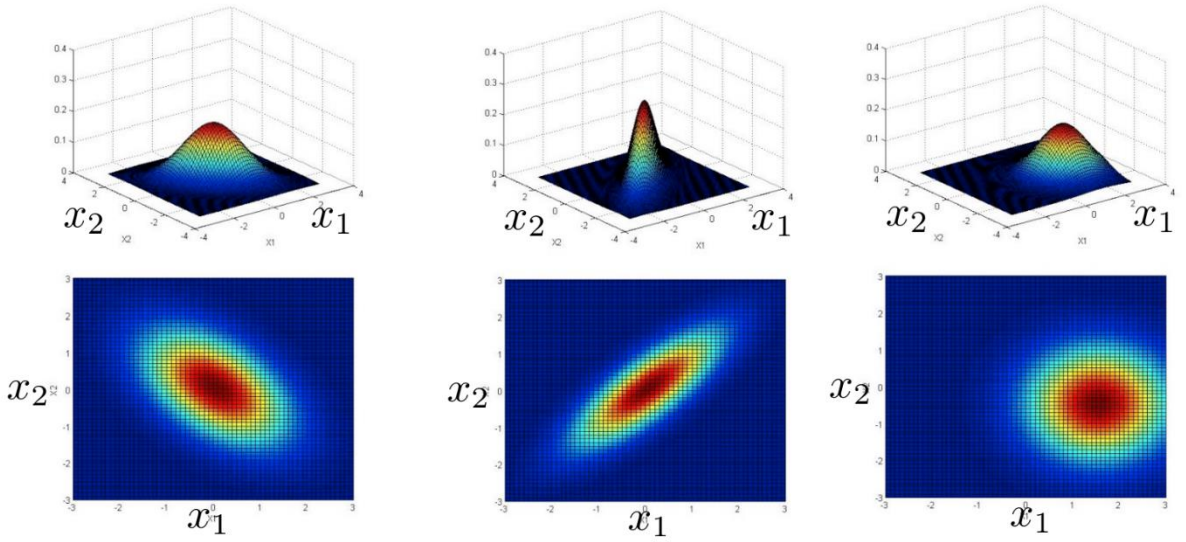
Instead of evaluating the σ, we are generating the **covariance matrix** Σ:

$$x \in R^n \,\Big|\Big|\, \mu \in R^n \,\Big|\Big|\, \Sigma \in R^{n*n}$$

$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix} \qquad \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \qquad \mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$



$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$ $\mu = \begin{bmatrix} 1.5 \\ -0.5 \end{bmatrix}$ $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



Univariate Gaussian

We can decide to **avoid** the use of matrices and to aim for a simpler and cheaper computational algorithm:

$$p(x_j; \mu_j, \sigma_j^2) = \frac{1}{\sqrt{2\pi}\sigma_j} e^{-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}}$$

And we evaluate the overall distribution:

$$p(x_1; \mu_1, \sigma_1^2) * \ldots * p(x_n; \mu_n, \sigma_n^2)$$

Pro) It allows manual combinations of values
Pro) Computationally cheaper
Pro) Scales better for large n
Pro) Works for small m
Con) Lost of information of the correlations between features

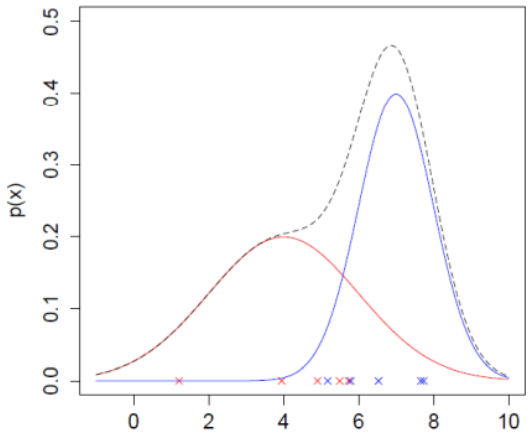
Multivariate Gaussian

$$p(x; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)}$$

Pro) Automatically captures correlations between dimensions
Con) Computationally expensive
Con) Σ not invertible if $m > n$

Main idea

The overall probability of a **variable** (gray) is assumable that is obtained by the sum of **gaussian distributions** (**red** and **blue**).
What we are computing are the **blue** and **red** (the clusters)
What we do: we can assume that data belongs to one of the n gaussian distributions in order to determine its cluster.



10.4.2 Likelihood function (Look at what theorem)

Since we assumed that the probability of a given point x in the mixture is the sum of the probability (**marginal probability**) of the same point in each k gaussian:

$$p(x) = \sum_{j=1}^{k} p(x \cap z = j)$$

Each cluster j has a contribution $p(x \cap z = j)$ on the overall probability $p(x)$ that the sample is in the mixture.

Using the **Bayes' theorem**, we can write explicitly the probability:

$$p(x) = \sum_{j=1}^{k} p(x|z = j)p(z = j)$$

Goal: we want to maximize the **likelihood** (the probability of the point (x) to belong on a specific cluster)

Notation abbreviation

1) $p_j(x) = p(x|z = j)$, probability density of the jth gaussian in the point x

2) $\pi_j = p(z = j)$ probability of the jth gaussian among all gaussian (**mixing coefficient**)

The final form of the formula can be written as:

$$p(x) = \sum_{j=1}^{k} \pi_j p_j(x)$$

More explicit formula:

$$p(x|\theta) = \sum_{j=1}^{k} \pi_j p_j(x|\theta_j)$$

Were θ_j are the parameters of the jth gaussian and θ represents the parameters of all of them.

Multivariate gaussian cluster

$$p(x; \mu_j, \Sigma_j) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_j|}} e^{\left(-\frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1}(x-\mu_j)\right)}$$

Vector of the means of the jth gaussian: μ_j

Covariance matrix of the jth gaussian: Σ_j

Number of features n.

10.4.3 Gaussian mixture parameters

Supposing that C_j represent the jth cluster and $|C_j|$ the number of points that belong to it (m).

If we knew which point belong to which cluster, we would compute the parameters as:

Univariate

$$\hat{\mu}_j = \frac{1}{|C_j|} \sum_{x^{(i)} \in C_j} x^{(i)}$$

$$\hat{\sigma}_j^2 = \frac{1}{|C_j| - 1} \sum_{x^{(i)} \in C_j} \left(x^{(i)} - \hat{\mu}_j\right)^2$$

Multivariate

$$\hat{\mu}_j = \frac{1}{|C_j|} \sum_{x^{(i)} \in C_j} x^{(i)}$$

$$\hat{\Sigma}_j^2 = \frac{1}{|C_j| - 1} \sum_{x^{(i)} \in C_j} \left(x^{(i)} - \hat{\mu}_j\right)\left(x^{(i)} - \hat{\mu}_j\right)^T$$

Unknown parameters: (μ_j, σ_j^2) or (μ_j, Σ_j)

10.4.4 Expectation-Maximization Algorithm (EM)

First) random initialization of (μ_j, Σ_j, π_j) for each k gaussian

Second) repeat the following steps until convergence (or other conditions like max steps)

Expectation step

E stand for **Expectation step**: the **responsibility** coefficient is evaluated; it gives an idea of how much the point is generated from the Gaussian that is paired with it at that moment

$$r_{ij} \coloneqq p\left(z = j | x^{(i)}, \theta^{(t-1)}\right)$$

Where $\theta^{(t-1)}$ are the parameters of the previous iteration (remember that are randomly generated during the first iteration for $\theta^{(0)}$.

Explicitly:

$$r_{ij} = \frac{\pi_j p_j\left(x^{(i)} \middle| \theta_j^{(t-1)}\right)}{\sum_{j=1}^{k} \pi_j p_j\left(x^{(i)} \middle| \theta_j^{(t-1)}\right)}$$

Maximization step

M stand for **Maximization step**: the parameters (μ, Σ and π_j) are recomputed as a function of the **responsibility coefficient** to maximize the **likelihood**.

$$\pi_j = \frac{1}{m} \sum_{i=1}^{m} r_{ij}$$

$$\mu_j = \frac{\sum_{i=1}^{m} x^{(i)} r_{ij}}{\sum_{i=1}^{m} r_{ij}}$$

$$\Sigma_j = \frac{\sum_{i=1}^{m} r_{ij} \left(x^{(i)} - \mu_j\right)\left(x^{(i)} - \mu_j\right)^T}{\sum_{i=1}^{m} r_{ij}}$$

10.4.5 PROS and CONS

Pros

1) Very flexible algorithm able to approximate multivariate gaussian to fit data by maximizing the likelihood

2) If the bias is approaching a zero means, the algorithm will not be affected

3) Very powerful against outliers

Cons

1) The most important **cons** to consider is that the algorithm will likely not converge if there are only few points with respect to the overall mixture

2) It is not needed all the flexibility offered by this model

10.5 Choosing the value of K

These algorithms (k-means, k-medoids and gaussian mixture) rely on the fact that k is known from the beginning but in reality, we haven't explained yet how to obtain hypothetically a good k.

Overfitting

A k too large will cause an **overfit** of the model since we are diving the problem into too many classes (clusters) therefore, it is too dependent on the **data**.

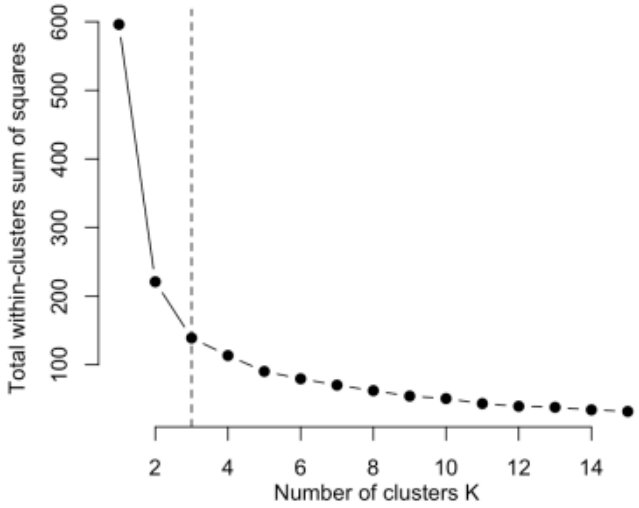
Underfitting

A k too small will cause the algorithm to **underfit** because the model will not be able to classify the data due to the absence of sufficient classes (clusters)

Likelihood (Recap)

$$\hat{L} = L(\hat{\theta}) = \max_{\theta} L(\theta; X; y; M) = p(y|X; \hat{\theta}, M)$$

10.5.1 Elbow method



The **total sum of squares** is something related to the overall **variance** of the data. Since by increasing the number of clusters k we are increasing the number of **centroids** (think about the k-means), we are reducing the distances from a given point $x^{(i)}$ to the centroids $c^{(j)}$ (the variance is reduced); we are basically increasing how well the model will fit the data, but it will come with a price such as **overfitting**.

How to avoid overfitting using the Elbow method: after increasing the number k, we will reach a point where the reduction of the **sum of the square** from $(k-1)$ to (k) start to decrease so little that it is pointless to add more clusters (we call the "limit point" the **elbow** as shown in the picture)

10.5.2 Kullback-Leibler divergence (Just an idea)

It measures how the given probability distribution diverges from the expected probability distribution (the information loss).

The main idea is that we want to evaluate how well our distribution $Q(i)$ will fit the data with a **real distribution** $P(i)$:

$$D_{KL}(P||Q) = \sum_i P(i) \frac{P(i)}{Q(i)}$$

Equivalent expression using $log(x)$ properties:

$$D_{KL}(P||Q) = -\sum_i P(i) \frac{Q(i)}{P(i)}$$

$D_{KL}(P||Q) \in [0,1]$ so, it is like a percentual of how well $Q(i)$ and $P(i)$ are similar since if they behave in the same way $D_{KL}(P||Q) = 1$ but if they are completely different $D_{KL}(P||Q) = 0$

10.5.3 Akaike Information Criterion (AIC) (Just an idea)

AIC derived as an asymptotic approximation of the Kullback-Leibler information between the model of interest and the truth:

$$AIC = 2k - 2\ln(\hat{L})$$

$\hat{L}$ **likelihood** and k **number of parameters**.

The idea is that the model with the lower the AIC, the better is the approximation.

Our likelihood $\hat{L}$ will decrease as we increase the number of parameters k but at the same time, $2k$ will increase as well so we aim on finding the right number of parameters which minimize AIC.

Note: you can relate $2k$ as a regularization term.

Note: we will take model which minimizes AIC

Correction by relating the number of samples

We can use this other formula since it considers the total number of samples injected:

$$AIC_c = AIC + \frac{2k(k+1)}{n-k-1}$$

10.5.4 Bayesian Information Criterion (BIC) (Just an idea)

BIC is the asymptotic approximation under the assumption that the model can result in an exponential distribution:

$$BIC = \ln(m)\,k - 2\ln(\hat{L})$$

m **number of samples**, k **number of features**.

Note: as for the AIC, we will take the one which minimizes the BIC

10.5.5 Deviance Information Criterion (DIC) (Just an idea)

It represents an asymptotic approximation of the deviance under the assumption that the posterior distribution is a multivariate normal

$D(\theta) = -2\log(p(y|X;\theta,M))$, the **deviance**

$\overline{D(\theta)} = E[D(\theta)]$

$\tilde{\theta} = E[\theta|y]$ the **posterior mean deviance**

$$p_D = E_{\theta|y}\left[-2\log(p(y|X;\boldsymbol{\theta},M)) + 2\log\left(p(y|X;\tilde{\boldsymbol{\theta}},M)\right)\right]$$

Note: careful at the difference

$$DIC = \overline{D(\theta)} + p_D$$

Note: we will take model which minimizes DIC

10.5.6 Comparison of Information Criterions (Just an idea)

- AIC lacks with the finite sample correction in its base form
- AIC is asymptotically equivalent to k-folds cross-validation on linear regression models with a least squares optimization
- BIC is able to detect the true model (if present in the comparison)
- BIC is only valid if n is much greater than k
- DIC tends to suggest overfitted models

10.5.7 Silhouette Coefficient

Silhouette Coefficient combines the ideas of **cohesion** and **separation**, for single samples, clusters, and clustering:

$coh =$ average distance of x_i to the samples in the same cluster

$sep =$ min of the average distance of x_i to samples in another cluster

The **silhouette coefficient** is determined by:

$$s_i = 1 - \frac{coh}{sep} \Big|\Big| \; if\ coh < sep$$

The closest to 1 the better.

10.6 Hierarchical Clustering (MANCA)

10.6.1 DBSCAN

Density-Based Spatial Clustering of Application with Noise (**DBSCAN**) it is a **density-based agglomerative** algorithm.
It computes the densities of the data within the dataspace, locates regions of high density separated from regions with low density.
Note: Density means number of points inside a radius Eps.

Core point

A point is defined as **core** if it has more than a specified number of points within Eps (MinPts)

Border Point

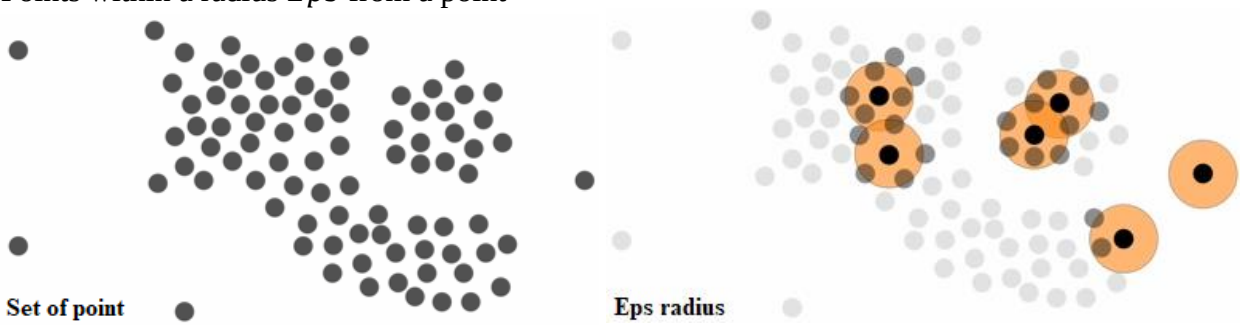
Less point than MinPts in the radius Eps but it's inside a region of a **core point**

Noise Point

Neither a **core** nor a **border** point (Outliers)

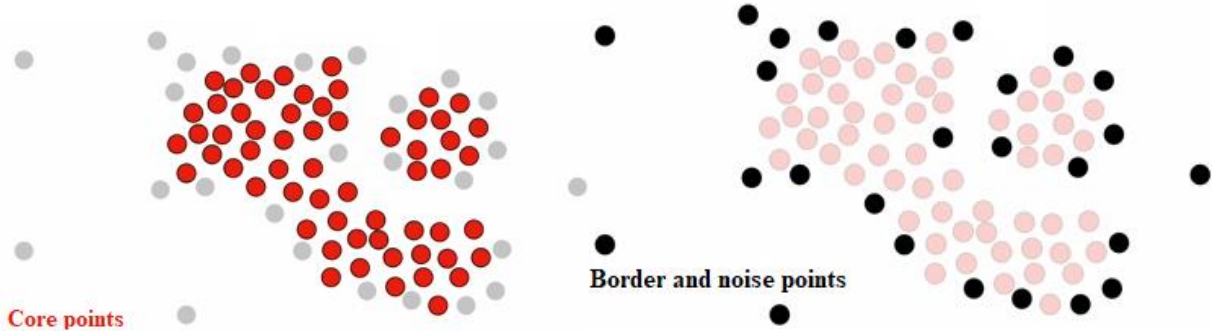
Eps-Neighborhood

Points within a radius Eps from a point

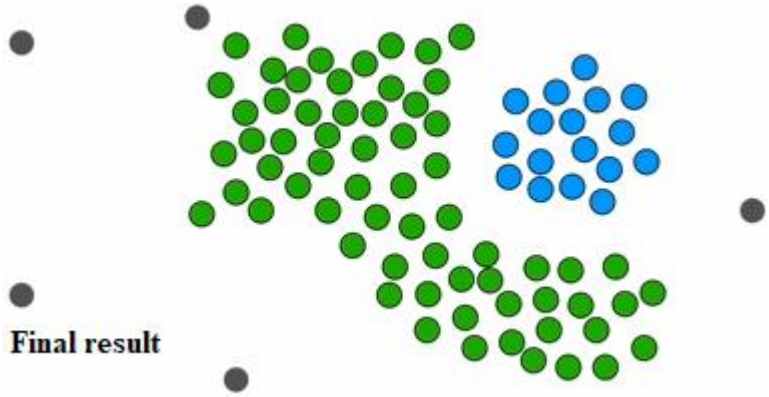


Procedure

First) a new cluster is created, and its first point is randomly pick among the **core points**
Second) every **core point** within Eps radius from a point inside a cluster are put in the same cluster. This step is repeated until there are no more **core points** to assign
Note: if the point is a **border point**, it will be assigned to the cluster, but it will not "spread" as the core points did (points inside **only** in the neighbor of a **border point** will not be classified since they are **noise points**)
Third) repeat the **first** and **second** steps until there are no more **core points** that needs to be assigned to a cluster.



Final result



10.6.2 Pros and cons

Pros

1) the number of clusters k is not required a priori
2) It can find clusters close to each other (even surrounded but not connected)
3) Notion of noise and robust against outliers
4) Requires just 2 parameters and is mostly insensitive to the order of the point in the dataset
5) If the data is well understood, MinPts and Eps can be set by a domain expert

Cons

1) Not entirely deterministic: some points can be part of different clusters depending on the order of how they were processed
2) The Euclidean distance is the most use but can be almost useless due to the "**curse of dimensionality**"
3) Since MinPts and Eps cannot be chosen appropriately for all clusters, large differences in density may result in bad clustering
4) Data and scale not well understood results in meaningless parameters (bad Eps)

10.7 Notes

The professor skips a lot of the slide, just idk what to add

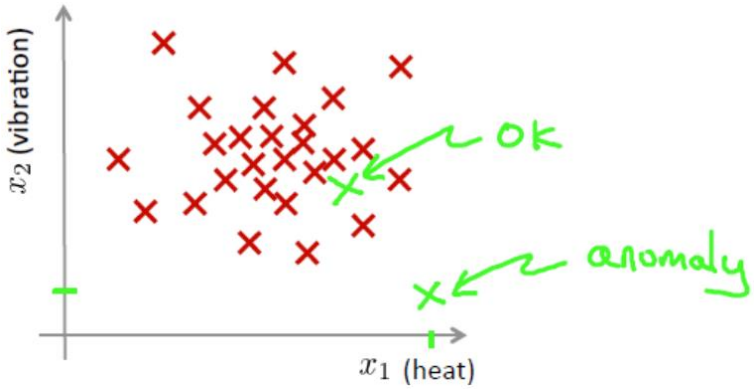
Capitolo 11 Lezione 2022-12-12

11.1.1.1.1 11b_Anomaly_Detection.pdf

11.1 Anomaly detection

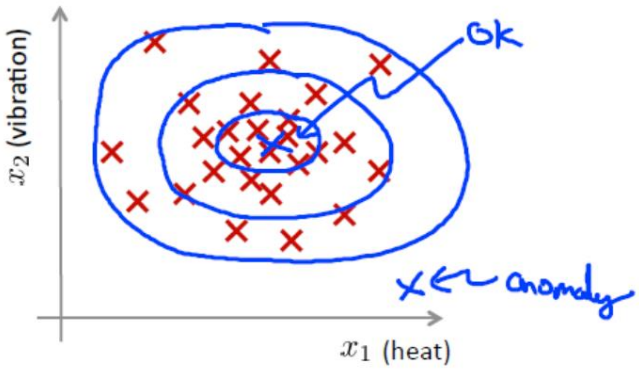
Anomaly detection is a technique used for identifying data which are much different from the others inside the same dataset. The behavior of these points is much different.

Example: suppose that we have the data of heat (x_1) and vibration intensity (x_2) generated by an **Aircraft engine**. Our dataset is composed by $Dataset: \{x^{(1)}, \dots, x^{(m)}\}$ and we would like to identify new cases $x_{test}^{(i)}$.



11.1.1 Gaussian Mixture Model (GMM) approach

Simply, we model the **non-anomalous data** with a Gaussian Distribution, or rather a mixture, in order to identify the data inside the normal behavior of the model and compute the probability of a new point x_{test} will be flag as an anomaly or not.



We model $p(x)$ such that a new point x_{test} will be flagged as an anomaly if:

$$p(x_{test}) < \varepsilon$$

Otherwise, $p(x_{test}) \geq \varepsilon$ will be flagged as an **acceptable data**.

11.1.2 Practical examples

First) supposing that a user ith performs $x^{(i)}$ activities, a **fraud detection system** may be built by analyzing unusual behavior by focusing on **anomaly behavior** ($p(x) < \varepsilon$) after the model $p(x)$ has been generated properly.

Second) supposing that a data center is analyzing each computer ith composed by $x^{(i)}$ characteristics. (Such as, $x_1 = memory\ use$, $x_2 = disk\ accesses\ /\ sec$, etc.)

We may be interested in an **anomalous behavior** of a computer based on these data.

11.1.3 Algorithm procedure

First) Choose the most indicative features (the one you think are the most)
Second) Divide the dataset into **Training**, **Validation** and **Test** sets where the **training set** is composed by non-anomalous cased
Third) Using the **EM algorithm**, fit the parameters of **GMM**
Forth) **Validation set** is used to find the best value of ε which maximizes the number of correctly classified cases (The **threshold**)
Fifth) Evaluate the error on the **test set**.

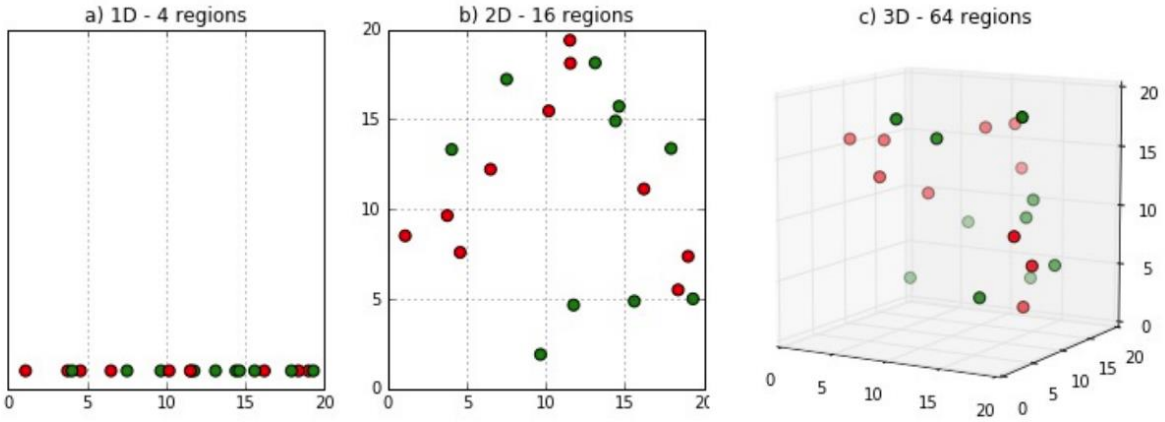
11.1.4 Anomaly Detection vs Supervised learning

Considering the case of **Anomaly detection,** the **positive number** (anomalies) is much smaller compared to the **negative number** of examples.
For an algorithm, it is difficult to learn from a little number of positive examples which could be similar to anomalies, and this makes things harder because an anomaly may be difficult to detect.
Different types of anomalies: there are different kinds of anomalies by making things even more difficult for an algorithm to learn a lot of types with a narrow number of positive examples.
On the other hand, **Supervised Learning** has a number of positive and negative examples pretty much balanced so that the algorithm is able to be trained enough in order to recognize anomalies (this makes future positive examples similar to the ones inside the training data)

11.1.4.1.1 12_Dimensionality_Reduction_PCA.pdf

11.2 The Curse of Dimensionality

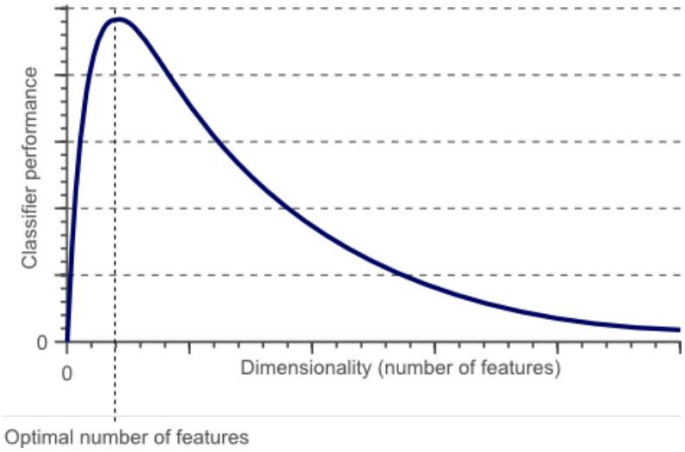
The introduction to this **problem** will be done by using an example.



As we can see, we have a problem that cannot be solved by 1D, so we pass to 2D and we realize that is not enough, so we move to a higher dimension (3D and so on) until we have enough dimensions that can generalize the problem.

Wrong guess: by increasing the number of features (therefore, the **dimensionality**) the classifier makes better and better predictions.

Short answer: No

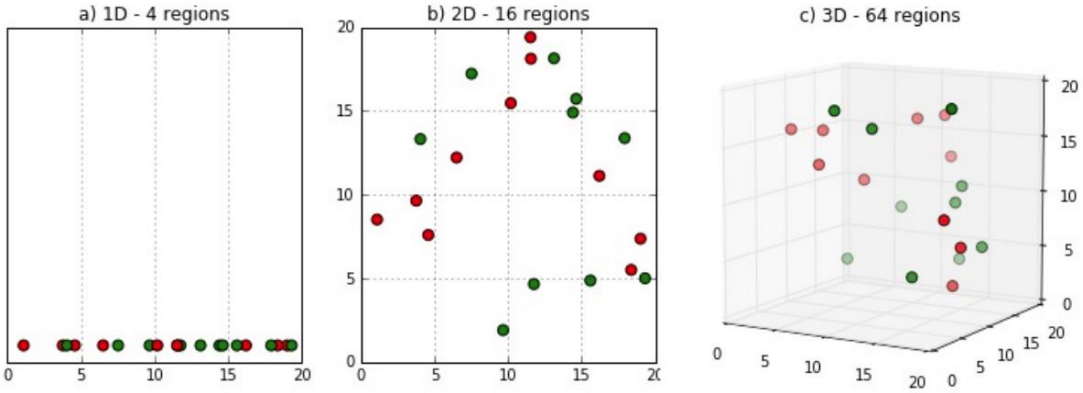


Basically, at the beginning we will have an increase of performance by increasing the number of features considered in our problem until from a certain point, the performance starts to decrease.

The **maximum performance** is not reached with the maximum number of features, so we identify the maximum by an **optimal number of features** (**Spoiler**, we are interested in that number)

11.2.1 Explanation of this phenomenon

One of the causes of this problem is related to the **density of the training samples** since it decreases by increasing the dimensionality of the problem.
Example:



1-Dimension) The graph is almost covered by points (density is approximately $30/4$)
2-Dimension) The samples starts to sparse across the graph (density approximately $30/16$)
3-Dimension) The samples are even more sparse (density approximately $30/64$)
Note: the **sample density** is evaluated by the total number of samples divided the area
From a **mathematical perspective**, there is an exponential increase in volume associated with the dimension.

11.2.2 Important concepts

Overfitting

The model become more complex by the increasing dimensionality (Training set if fixed)

Sparseness

The more features we use, the sparser the training data which makes harder to find good parameters for the estimator (decision boundaries) and increases the time needed to find them.

Enormous amount of training data required

The increase of the dimensionality needs an enormous amount of data because we need to cover the sparseness in some sense, but it is likely that we have a fixed amount of data which cannot be increased.

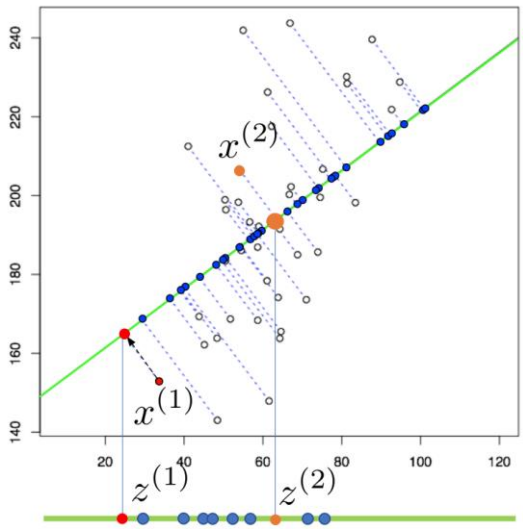
Exponential increase of the data

Another way to avoid overfitting and increasing the coverage is by increase exponentially the data which is almost never possible.

11.3 Dimensionality reduction

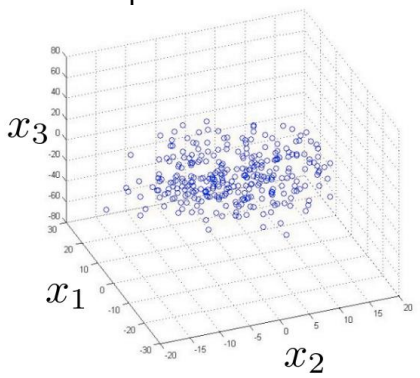
Dimensionality reduction is an important aspect since it one of the **countermeasures** adopted in order to avoid "**The Curse of Dimensionality**", since it is able to reduce the negative aspects that comes by increase the dimensionality of the problem but there are also other motivations.

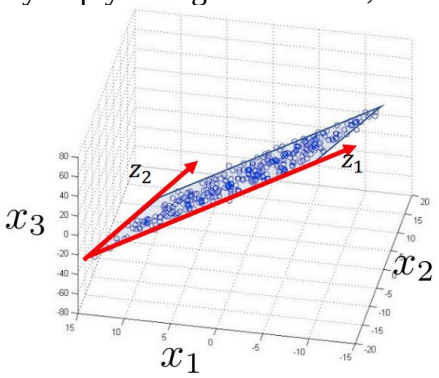
11.3.1 Motivation 1 – Data compression

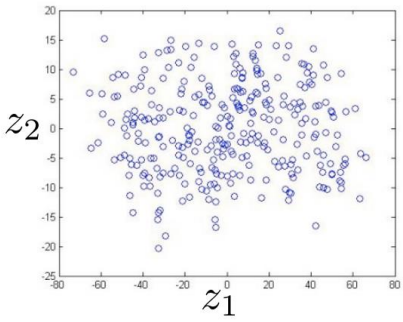


In the example, we are **projecting** 2-Dimensional data into a 1-Dimensional line by reducing its dimensionality. Since we re reducing the dimensionality, we are also reducing the **overall size** (which means a pc will need less memory to store the same information)

Note: compression doesn't necessary imply losing information, it depends.







$$x^{(i)} \in \mathbb{R}^3 \quad \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad \mathbf{z^{(i)}} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$$

Similar example from 3D to 2D.

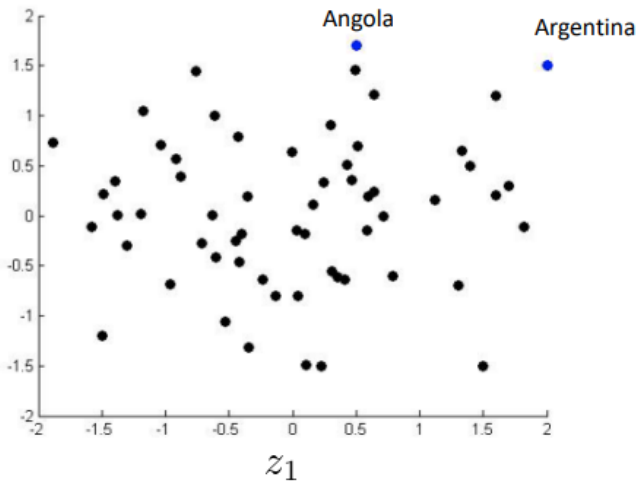
11.3.2 Motivation 2 – Data Visualization

Supposing that we would like to reduce the dimensionality of the following dataset:

	x_1	x_2	x_3	x_4	x_5	
Country	Region	Population	Area	Pop. Density	Coastline	...
Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500	48,0	0,00	...
Albania	EASTERN EUROPE	3581655	28748	124,6	1,26	...
Algeria	NORTHERN AFRICA	32930091	2381740	13,8	0,04	...
American Samoa	OCEANIA	57794	199	290,4	58,29	...
Andorra	WESTERN EUROPE	71201	468	152,1	0,00	...
Angola	SUB-SAHARAN AFRICA	12127071	1246700	9,7	0,13	...
Anguilla	LATIN AMER. & CARIB	13477	102	132,1	59,80	...
Antigua & Barbuda	LATIN AMER. & CARIB	69108	443	156,0	34,54	...
Argentina	LATIN AMER. & CARIB	39921833	2766890	14,4	0,18	...
...	...	...	...	...	...	...

After an analysis of the data, you conclude that this dataset can reduce its dimensionality (Depending on the algorithm, it may be focusing on the post important features or other methods).

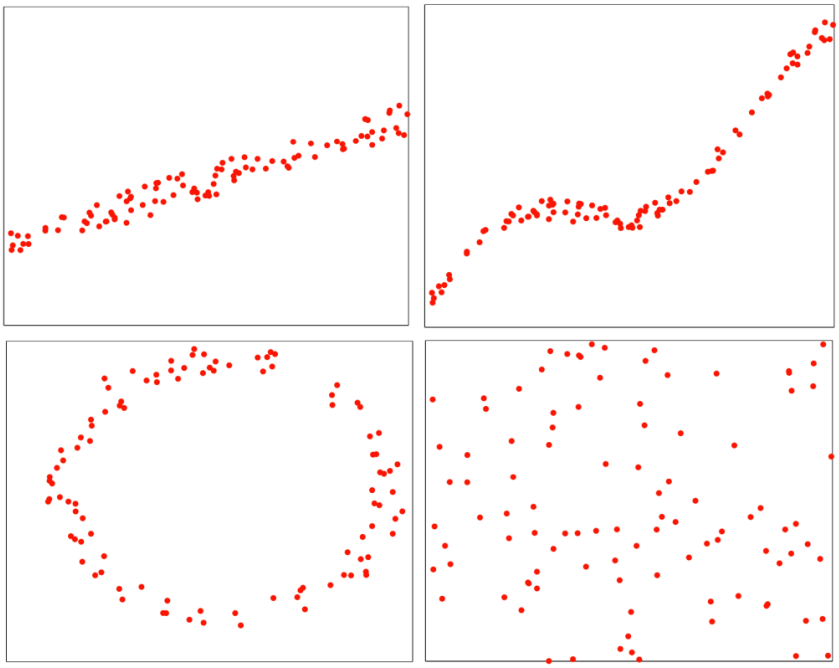
	z_1	z_2
Country	z1	z2
Afghanistan	1.6	1.2
Albania	1.7	0.3
Algeria	1.6	0.2
American Samoa	1.4	0.5
Andorra	0.5	1.7
Angola	2	1.5
Anguilla	1.4	1.5
Antigua & Barbuda	0.5	0.5
Argentina	2	1.7
...	...	...



Thanks to the Dimensionality Reduction, we are able to reduce the dimensionality to 2D, which allows us to represent on a 2D plane the points of our reduced dataset.

11.4 Principal Component Analysis (PCA)

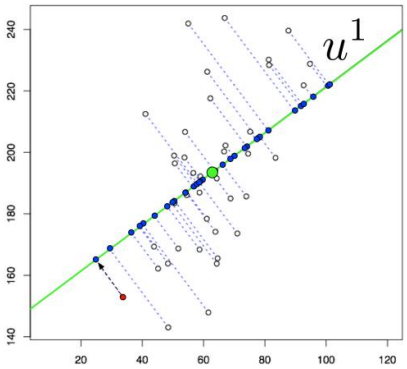
This method discovers the principal component of our data (Component which can describe the whole data with a small loss of information).

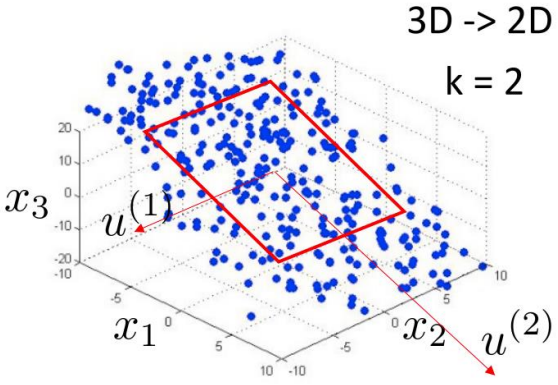


As we can see, we may conclude observe some behavior from these images (maybe except for the last one which seems random).
The PCA aims on taking the most informative information about the data so that a pretty good approximation is done (with less dimensionality)

11.4.1 Problem Intuition

We would like to reduce the dimensionality from nD to kD and minimizing in the process the projection error. In order to do that we need to find the k vectors of directions $u^{(1)}, \dots, u^{(k)}$, called **latent variables**, onto which to project the data so as to minimize the projection error.





As we can see, we would like to reduce a 3D problem from a 2D problem by choosing the best vectors to project the data.

11.4.2 PCA formulation

PCA aims to find a **linear transformation** of the variables that projects the original ones in a new coordinate system where (most of) the variation in the data can be described with fewer dimensions than the initial data.
Basically: a good representation of the original problem with less dimensions
The principal components of a collection of points in a real coordinate space are a sequence of k orthogonal vectors $u^{(1)},\ldots, u^{(k)}$ such that:
First Principal Component) explains the most variance of the original variables (it is obtained by a linear combination of it)
Second Principal Component) explains the most variance left from the **first PC** (So, still linear combinations of the original variables)
Third Principal Component) same principle, the most variance left from the **second PC**.
Note: if this list is read on the contrary, we may make a list of the most not useful PC.

Projection onto one direction

$$z_j^{(i)} = x^{(i)} u_j$$

The jth variables in the new space is computed like this.

Reconstruction

$$\tilde{x}^{(i)} = \sum_{j=1}^{k} z_j^{(i)} u_j$$

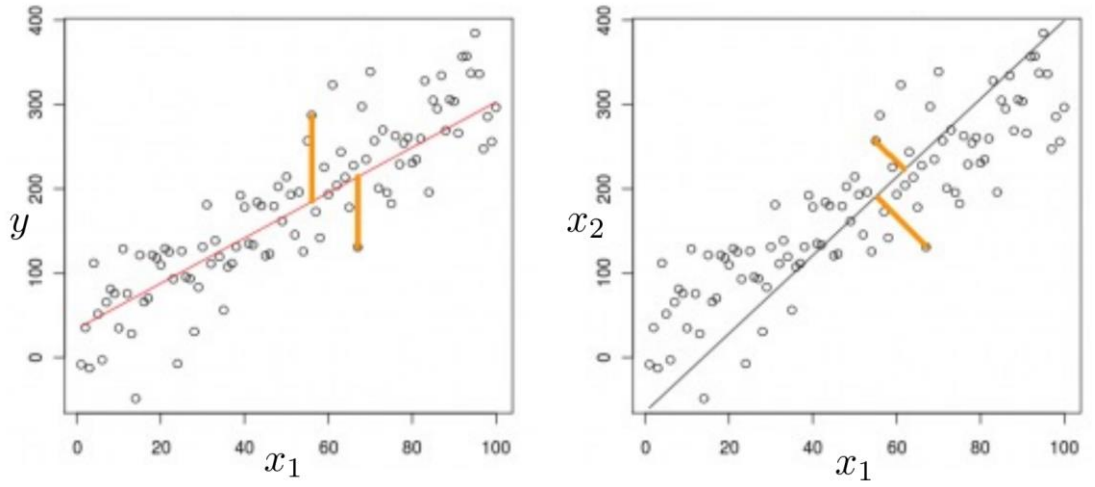
This is a reconstruction of $x^{(i)}$ in n based on k components with $n < k$.
Note: k is directly correlated to the error.

Error

$$e_k = \sum_{i=1}^{m} \left(x^{(i)} - \tilde{x}^{(i)}\right)^2 = \sum_{i=1}^{m} \left(\sum_{j=k+1}^{n} \left(u_j x^{(i)}\right)^2 \right)$$

The error that we commit by reducing from n dimensions to k dimensions is the same that we would have done by just considering the **eigenvectors** from $(k+1)$ to (n), which mean the **last representative components** since we are just considering the k most representative ones.
Basically: I'm simply considering the part that was removed due to the reconstruction.

11.4.3 PCA is not linear regression



In the first example (**Linear regression**) we are estimating the value of a variable (y) using a variable (x). In the second example (**PCA**), we are mapping 2D features into points (x_1, x_2) and we are evaluating the **error of projection** obtained by reducing the dimension from 2D to 1D.
In the **Linear regression**, the same **line** represents the error from the prediction to the real value.

11.4.4 Preprocessing

Normalization

Usually a Z-score is performed on the **training set**.

Covariance Matrix

The **covariance matrix** is computed between each pair of features:

$$\Sigma_{ij} = Cov(x_i, x_j) = E[(x_i - \mu_i)(x_j - \mu_j)] = E[(x_i)(x_j)]$$

Important: we are supposing $\mu = 0$ due to the previous Z-score normalization.

11.5 Singular Value Decomposition (SVD)

The **SVD** is a way of computing the **eigenvectors** of Σ.

$$[U, S, V] = svd(\Sigma)$$

The factorization is based on a matrix $A \in R^{m*n}$ which can be divided into:

$$A = USV^*$$

Left eigenvectors matrix *U*

$U \in R^{m*m}$ is a left unitary matrix which contains the **eigenvectors**, it is an orthogonal matrix which means that $UU^* = 1$, so $U^{-1} = U^*$

Eigenvalues matrix *S*

$S \in R^{m*n}$ is a diagonal matrix which contains the **squared roots of the eigenvalues AA^T** (The first n component are **not null,** and they are ordered in a decreasing order)

Conjugate transpose V^*

$V^* \in R^{n*n}$ is the conjugate transpose of the right unitary matrix $(n * n)$

Note: if the metrices are symmetric (covariance matrix) $U = V$ and S contains the absolute values of the eigenvalues of A.

11.5.1 SVD reduction (PCA related)

$$U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix} \in R^{n*n}$$

If we want to reduce the dimensionality of our data from n to k by taking the most informative eigenvectors, we can simply take the first k columns from U:

$$U_{reduce} = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \dots & u^{(k)} \\ | & | & & | \end{bmatrix}^T \in R^{n*k}$$

We can now transform our data (reducing them):

$$x \in R^n \rightarrow z \in R^k$$

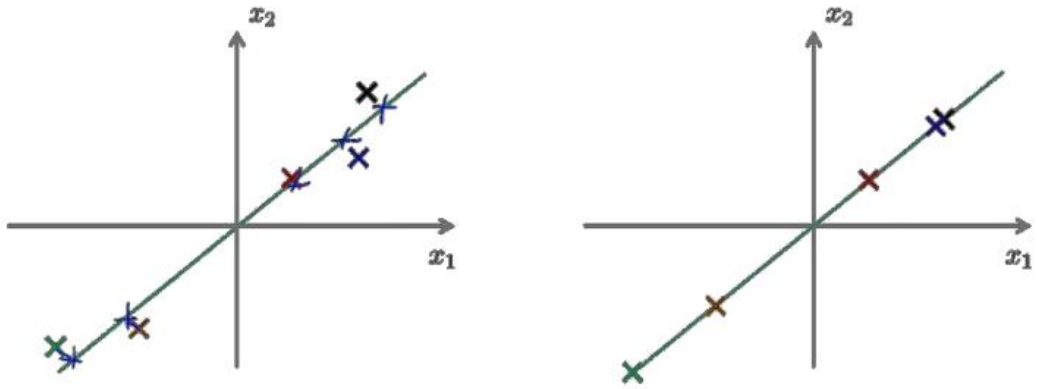
$$z^{(i)} = U_{reduce}^T x^{(i)} = \begin{bmatrix} - & (u^{(1)})^T & - \\ - & (u^{(n)})^T & - \\ & \dots & \\ - & (u^{(k)})^T & - \end{bmatrix} x^{(i)}$$

Since $U_{reduce}^T \in R^{k*n}$ and $x^{(i)} \in R^n$, their product $z^{(i)} \in R^k$.

11.6 PCA algorithm

First) Preprocessing data (Usually Z-score)
Second) Compute sigma $\Sigma_{ij} = E[(x_i)(x_j)]$ since $\mu = 0$ due to the **first step**.
Third) Compute the **SVD** $[U, S, V] = svd(\Sigma)$
Forth) Choose how many k dimension and generate $U_{reduce} = U(:, 1:k)$
Fifth) Map the data into the new space $z = U_{reduce}^T * x$

11.6.1 Reconstruction with notes



$$z = U_{reduce}^T * x$$

In this example, we have reduced the original data $x \in R^2$ to $z \in R$. Since we decided to remove all the **eigenvectors** from $(k+1)$ to (n) $(n-k$ are discarded)
Important: since we are just trying to retain the most variance, we **are not** doing a feature selection. We are changing the space of the variables; the number of features will be the same.

11.6.2 Number of Principal Components – Parameters

Average squared projection error

$$\frac{1}{m}\sum_{i=1}^{m}\left|\left|x^{(i)} - x_{approx}^{(i)}\right|\right|^2$$

Total variation

$$\frac{1}{m}\sum_{i=1}^{m}\left|\left|x^{(i)}\right|\right|^2$$

Minimum *k* condition

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left|\left|x^{(i)} - x_{approx}^{(i)}\right|\right|^2}{\frac{1}{m}\sum_{i=1}^{m}\left|\left|x^{(i)}\right|\right|^2} \leq 0.01$$

Consideration: we are imposing that the 99% of the variance is retained

Equivalently

$$\frac{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}-x_{approx}^{(i)}\right\|^2}{\frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)}\right\|^2}\leq 0.01 \Bigg| \rightarrow equivalent \rightarrow \Bigg| \frac{\sum_{i=1}^{k}S_{ii}}{\sum_{i=1}^{n}S_{ii}}\geq 0.99$$

While computing the SVD of sigma ($[U,S,V]=svd(\Sigma)$) we can use the values of S:

$$S=\begin{bmatrix} S_{11} & & \\ & \ddots & \\ & & S_{nn}\end{bmatrix}\in R^{n*n}$$

We can use these formulas:

$$1-\frac{\sum_{i=1}^{k}S_{ii}}{\sum_{i=1}^{n}S_{ii}}\leq 0.01 \ \Big|\rightarrow\Big| \frac{\sum_{i=1}^{k}S_{ii}}{\sum_{i=1}^{n}S_{ii}}\geq 0.99$$

11.6.3 Algorithm last note

After executing **PCA** for different number of $k=1,2,3,\ldots$, we will find such k which retains at least 99% of variance.
After determining that, we can evaluate U_{reduce} and changing the space $z=U_{reduce}^{T}*x$

11.6.4 Pros and cons (magari aggiungi esempi (?))

Pro) Can handle varying densities
Pro) Stable algorithm over runs and parameter choices
Pro) **Robust to outliers**.

Con) Important parameters must be set by hand
Con) Cannot distinguish non-linear structure from no structure

11.7 Kernel PCA

11.7.1 Intuition

Since non-linearity may represent a problem, it is reasonable to think that we would like to use PCA on $\Phi(x)$ instead of x, where $\Phi(x)$ is in a **new feature space** in a higher dimension R^d.
After the transformation from the x space to the $\Phi(x)$ spaces, we may use the **PCA** so that we will obtain non-linear result in the original space x (e.g., this would have resolved the circle problem since clearly it would be better to work with polar coordinates instead of cartesian one in that case)
PCA finds principal axes by working on the covariance matric:

$$Cov = \frac{1}{m}\sum_{i=1}^{m} x_i x_i^T$$

Note: Cov is positive definite and can be diagonalized with non-negative eigenvalues:

$$\lambda v = Cov\ v$$

It follows that:

$$Cov\ v = \frac{1}{m}\sum_{i=1}^{m} x_i x_i^T v = \lambda v$$

We can then compute that:

$$v = \frac{1}{\lambda m}\sum_{i=1}^{m} x_i x_i^T v = \frac{1}{\lambda m}\sum_{i=1}^{m} (x_i * v) x_i^T$$

All the solution v with $\lambda \neq 0$ (otherwise, it wouldn't be considered a **Principal Component**) can be expressed as:

$$v = \sum_{i=1}^{m} a_i x_i$$

After evaluating λ and v, the transformation is done by using this formula where x_i are the data of the dataset and a_i are coefficients used for measuring the x_i data inside the summation.

11.7.2 Kernel PCA Development (Quanto scritto non so se è corretto)

As we said, we map our data into another space: $x \in R^n$ becomes $\Phi(x) \in R^d$ so that the formulas can be written as:

$$Cov\ v = \frac{1}{m}\sum_{i=1}^{m} \Phi(x_i)\Phi(x_i)^T v = \lambda v$$

So, it follows that:

$$v = \frac{1}{\lambda m}\sum_{i=1}^{m} \Phi(x_i)\Phi(x_i)^T v = \frac{1}{\lambda m}\sum_{i=1}^{m} (\Phi(x_i) * v)\Phi(x_i)^T$$

If $\lambda \neq 0$, all the solutions v_l can be expressed as:

$$v_l = \sum_{j=1}^{m} a_{lj}\Phi(x_j)$$

Important: finding the eigenvectors is equivalent to finding the **alpha coefficients**.
Note: the coefficient for v_l is j, for $Cov\ v_l$ is i (Just, be careful)

$$Cov\ v_l = \frac{1}{m}\sum_{i=1}^{m} \Phi(x_i)\Phi(x_i)^T \left(\sum_{j=1}^{m} a_{lj}\Phi(x_j)\right) = \lambda_l \sum_{j=1}^{m} a_{lj}\Phi(x_j)$$

I move variables around:

$$Cov\ v_l = \frac{1}{m}\sum_{i=1}^{m} \Phi(x_i) \left(\sum_{j=1}^{m} a_{lj}\Phi(x_i)^T\Phi(x_j)\right) = \lambda_l \sum_{j=1}^{m} a_{lj}\Phi(x_j)$$

We then recognize the form of the **kernel function** $k(x_i, x_j) = \Phi(x_i)^T\Phi(x_j)$ where $k \in R^{m*m}$:

$$Cov\ v_l = \frac{1}{m}\sum_{i=1}^{m} \Phi(x_i) \left(\sum_{j=1}^{m} a_{lj}k(x_i, x_j)\right) = \lambda_l \sum_{j=1}^{m} a_{lj}\Phi(x_j)$$

We then now multiply both sides by $\Phi(x_k)^T$:

$$\Phi(x_k)^T \frac{1}{m}\sum_{i=1}^{m}\Phi(x_i)\left(\sum_{j=1}^{m} a_{lj}k(x_i,x_j)\right) = \Phi(x_k)^T\lambda_l\sum_{j=1}^{m} a_{lj}\Phi(x_j)$$

If we move $\Phi(x_k)^T$ inside the summations:

$$\frac{1}{m}\sum_{i=1}^{m}\Phi(x_k)^T\Phi(x_i)\left(\sum_{j=1}^{m} a_{lj}k(x_i,x_j)\right) = \lambda_l\sum_{j=1}^{m} a_{lj}\Phi(x_k)^T\Phi(x_j)$$

So, we recognize that the expressions are composed by the same kernel value ($k(x_k,x_j)$):

$$\frac{1}{m}\sum_{i=1}^{m}k(x_k,x_j)\sum_{j=1}^{m} a_{lj}k(x_i,x_j) = \lambda_l\sum_{j=1}^{m} a_{lj}k(x_k,x_j)$$

11.7.3 E da qui non ho capito più un cazzo

This whole formula can be written in matrix notation as:

$$\frac{1}{m}K^2a_l = \lambda_l K a_l$$

Since $K \in R^{d*d}$ is invertible we can simplify both sizes, then by moving m we obtain:

$$Ka_l = m\lambda_l a_l$$

If we decide to normalize K by m:

$$K\alpha = \lambda\alpha$$

And we can obtain the same formulation as before defining:

First) The kernel matrix $K \in R^{m*m}$ in which the ijth element is $K(x_i,x_j)$

Second) The vector $\alpha \in R^{m*1}$ whose jth element is α_j

Now, we can represent data in the new space by:

$$\Phi(x)^T v^k = \sum_{j=1}^{m}\alpha_i^n K(x_i,x)$$

Where n and k are the numbers of components in the old and new space.

Capitolo 12

Nota, questa parte è giusto un riassunto giusto per non fare scena muta perché non credo la chiederà mai, ma se la chiede almeno sai qualcosa.
L'ultimo pacco di slide dovrebbe contenere tutto quello che sta qui

12.1 Independent Component Analysis (ICA)

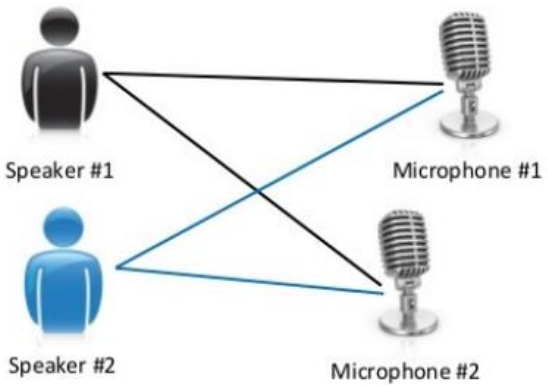
La **ICA** è un metodo che cerca di decomporre un segnale misto in modo tale da ottenere le sue sorgenti indipendenti.

Statistical Independence

Due eventi A e B sono indipendenti quando l'avvenimento o meno dell'evento A (o B) non condiziona in nessun modo la probabilità dell'altro evento di poter avvenire (*Lancio di un dado consecutivo*).

Cocktail Party Problem

Esempio molto comune, l'idea è che in una festa dove tante persone parlano contemporaneamente è difficile distinguere le diverse voci, così come in un segnale sonoro combinato.
Obbiettivo: trovare un insieme di funzioni di separazione per distinguere le varie sorgenti sonore, ovvero usando la ICA.



Soluzione: utilizzare più microfoni e cercare di separare le diverse sorgenti sonore

12.1.1 Problem and model definition

Ipotizziamo di avere un segnale x composto da due segnali $x = (x_1(t), x_2(t))$, ovvero le ampiezze all'istante t.
I segnali indipendenti vengono denominati $s = (s_1(t), s_2(t))$
Ottenendo in conclusione:

$$x_1(t) = a_{11}s_1 + a_{12}s_2$$
$$x_2(t) = a_{21}s_1 + a_{22}s_2$$

I parametri a sono dipendenti dalla distanza della sorgente s_i dal microfono ad esempio.
Definiamo la mixing matrix A:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Da cui segue che:

$$As = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} a_{11}s_1 + a_{12}s_2 \\ a_{21}s_1 + a_{22}s_2 \end{bmatrix} = x$$

In sintesi $As = x$.
Definita la **unmixing matrix** $W = A^{-1}$ possiamo affermare che:

$$s = Wx$$

Con W composta da elementi unmixing w_i^T tali che:

$$s_j^{(i)} = w_i^T x^{(i)}$$

12.1.2 Ambiguità di permutazioni

Definiamo una matrice di permutazione P (matrice $n * n$ composta da righe con un solo 1 e tutti 0):

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \left\| \right. \quad P = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \left\| \right. \quad P = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Se moltiplico P per un vettore z di dimensione n otterrò un vettore $P * z$ con le permutazioni di z.
Dato solo i segnali $x^{(i)}$, non c'è modo di distinguere nella ICA:

$$W * x$$
$$P * W * x$$

Questo perché P non altera la natura del segnale ottenendo la stessa soluzione ma permette di riordinare le sorgenti indipendenti:

$$W * x = \begin{bmatrix} 24 \\ 19 \end{bmatrix}$$
$$P * W * x = \begin{bmatrix} 19 \\ 24 \end{bmatrix}$$

12.1.3 Ambiguità di scala

Questa ambiguità si verifica quando i valori di una matrice di separazione W possono essere moltiplicati per una costante arbitraria senza che ciò influisca sulle soluzioni s trovate.

$$x^{(i)} = A * s^{(i)}$$

Il segnale $x^{(i)}$ i-esimo osservato, A matrice mixing e $s^{(i)}$ sorgente indipendente i-esima.
Siamo interessati a conoscere $s = Wx$.
Dato che la moltiplicazione per una costante di una qualsiasi riga o colonna di A non influisce sulle soluzioni s (ad esempio $A \rightarrow 2A$ per ogni $s^{(i)} \rightarrow (0.5)s^{(i)}$) otteniamo la stessa soluzione $x^{(i)} = As^{(i)}$.
Possiamo moltiplicare ogni colonna di W per una costante arbitraria senza influire sulle soluzioni s rendendo suscettibile la ICA all'ambiguità di scala.

12.1.4 Ambiguità con probabilità gaussiana

Le sorgenti non possono essere gaussiane.
Ipotizziamo che $s \sim N(0,1)$ e sappiamo che $x = A * s$.
La distribuzione di x è gaussiana con media nulla e covarianza:

$$E[xx^T] = E[A * s * s^T * A^T] = AA^T$$

Ipotizziamo di conoscere una matrice ortogonale R ($RR^T = R^T R = I$)
E definiamo $A' = AR$ e usiamo A' come matrice di mixing ottenendo $x' = A' * s$ e anche x' avrà una distribuzione gaussiana:

$$E[x'(x')^T] = E[A' * s * s^T * (A')^T] = E[AR * s * s^T * (AR)^T] = ARR^T A = AA^T$$

In conclusione, se le sorgenti fossero gaussiane non sapremmo se sono state unite con A' o A.

12.1.5 Densità e trasformazione lineare

La relazione della densità di probabilità tra s e x è pari a:

$$p_x(x) = p_s(Wx)|W|$$

Perché in maniera molto facile:

$x = As$, dove s è come se fosse proporzionato di un fattore $|A|$ rispetto a x.

$$p_x(x) = \frac{p_s(Wx)}{|A|} \rightarrow (W = A^{-1}) \rightarrow p_x(x) = p_s(Wx)|W|$$

Dove $|W|$ è il determinante di W ($\det(W)$)

12.2 Algoritmo ICA

Definiamo la probabilità congiunta come il prodotto delle distribuzioni marginali:

$$p(s) = \prod_{i=1}^{n} p_s(s_i)$$

Conoscendo la relazione che intercorre tra x e s con le rispettive probabilità.

Likelihood

$$p(x) = \prod_{i=1}^{n} p_s(w_i^T x)\,|W|$$

Noi vogliamo massimizzare questa probabilità.

Importante: possiamo moltiplicare in questo modo perché abbiamo ipotizzato l'indipendenza delle sorgenti.

Vogliamo trovare i parametri sconosciuti che massimizzino la probabilità di ottenere i dati osservati.

In realtà per risolvere questo problema di massimizzazione, noi eseguiamo la $\log(likelihood)$:

$$l(W) = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \ln\left(p_s(w_j^T x^{(i)}\right) + \ln|W| \right)$$

La sommatoria più interna deriva dal fatto che lo facciamo su tutte le componenti di W ($j = 1$ a n) mentre la seconda che lo eseguiamo su ogni segnale osservato $x^{(i)}$ ($i = 1$ a m).

12.2.1 Sigmoid-based

Noi sappiamo che $p_s(s)$ permette di calcolare $P(s)$:

$$P(s) = \int_{s_0}^{s_1} p_s(s)ds \in [0,1]$$

Ovvero l'integrale della **densità di probabilità** $p_s(s)$ ci permette di calcolare la **distribuzione cumulativa** $P(s)$.

Sapendo che anche la sigmoid $g(s) \in [0,1]$ ipotizziamo $P(s) = g(s)$ da cui segue che:

$$p_s(s) = g'(s)$$

E che quindi:

$$l(W) = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \ln\left(\boldsymbol{g'}(w_j^T x^{(i)}\right) + \ln|W| \right)$$

Procedimento

Sappiamo che il gradiente del determinante è calcolabile come:

$$\nabla_W = |W|(W^{-1})^T$$

Stochastic gradient learning rule

$$W := W + \alpha \left(\begin{bmatrix} 1 - 2g\left(w_1^T x^{(i)}\right) \\ 1 - 2g\left(w_2^T x^{(i)}\right) \\ \dots \\ 1 - 2g\left(w_n^T x^{(i)}\right) \end{bmatrix} x^{(i)} + (W^T)^{-1} \right)$$

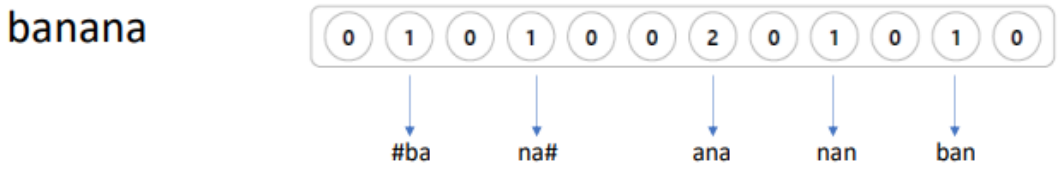
12.3 Dimensionality Reduction - Embedding

12.3.1 Vector Space Models

In un Vector Space Model, gli oggetti sono rappresentati come vettori di numeri in uno spazio vettoriale.

Esempio:

Se consideriamo le parole di un documento, possiamo rappresentare ogni parola come un vettore numerico in cui ogni dimensione corrisponde a un attributo specifico della parola, come la frequenza con cui appare nel documento o la sua relazione semantica con altre parole.

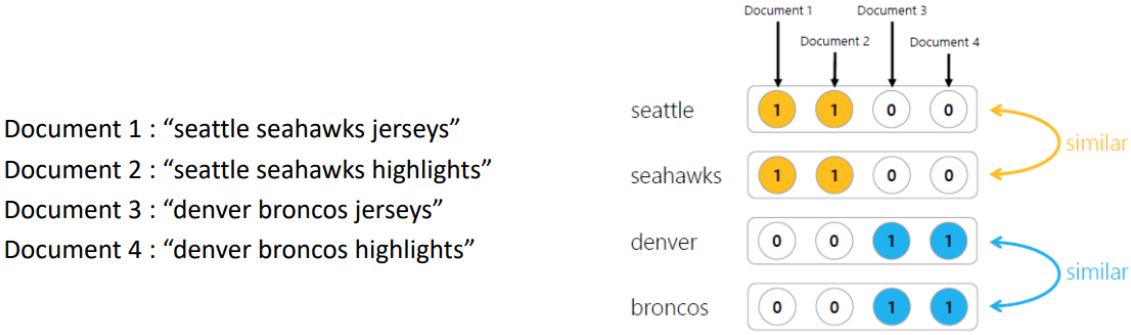


12.3.2 Relatedness

An example of a relation between two items is the **cosine similarity**:

$$similarity(A, B) = \frac{A * B}{||A||||B||}$$

However, depending on what the vector represents this notion changes.



Similar documents may contain similar words so depeding on how similar the words inside of a document are, we can detect similar documents.

This is a similar example but we are now considering the context vectors.

In the end, depending on which features or relations we are considering will determine which aspect we are relating to "similar".

12.3.3 Embedding problem

Il problema palese è l'aumento spoporzionato delle features che stiamo considerando generando vettori ad alte dimensioni con il difetto di essere sparsi (Dove l'elemento nullo è presente molto spesso, tipo lo 0 negli esempi precedenti).
Per **embedding** s'intende il vettore che descrive le caratteristiche di un determinato item in un determinato spazio vettoriale.

12.4 Word2Vec

È una tecnica molto utilizzata per risolvere il problema del generico embedding in un contesto di parole come oggetti.
Tra i **vantaggi** più noti abbiamo:
1) Utilizzo di un vettore a bassa dimensionalità per parola
2) La similarità di una parola **equivale** alla similarità tra vettori
3) Più veloce e può incorporare nuove frasi/documenti o aggiungere una parola nel vocabolario

Idea (Context based)

Cerca di predirre il significato di una parola dal **contesto** in cui è presente e che quindi gli embedding vettoriali sono in grado di catturare la somiglianza semantica (parole simili = embedding simili = distanza vettoriale piccola).
Esempio

«Rome is the **capital** of the **country** Italy»
«London is the **capital** of the **country** UK»

"**Rome**" e "**London**" possono essere considerate parole simili perché condividono lo stesso contesto di "**capital**", stessa cosa si può dire per "**Italy**" e "**UK**" poiché condividono il contesto **country**.

12.4.1 Metodologie

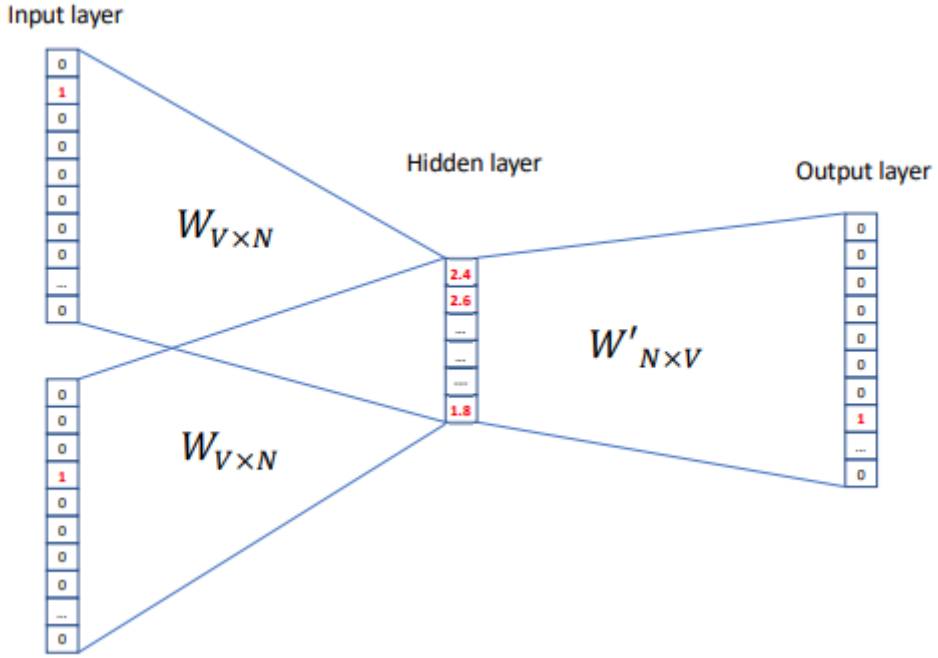
Continuous Bag-of-Words (CBOW)

Cerca di prevedere la **parola target** a partire dai contesti,
quindi usa finestre di parole per predirre le parole di mezzo.

Skip-gram (SG)

Cerca di prevedere i **contesti** a partire dalla parola target,
quindi usa singole parole per cercare di predirre quello che gli sta intorno.

12.4.2 Continuous Bag-of-Words



L'algoritmo funziona che in ingresso abbiamo n parole (Nell'esempio $n = 2$).
First) La matrice $W \in R^{V*N}$ è una delle matrici di embedding avente i pesi che permetterà di trasformare la generica parola $x_i \in R^V$ in uno spazio più contenuto ($W^T * x_i = v_i \in R^N$).
Second) Il vettore centrale contiene la media dei vettori v_i, quindi:

$$v_{target} = \frac{1}{m}\sum_{i=1}^{m} v_i$$

Nel caso dell'esempio:

$$v_{target} = \frac{1}{2}(v_1 + v_2)$$

Third) Il vettore v_{target} verrà ritrasformato utilizzando la seconda matrice di embedding $W' \in R^{V*N}$ che contiene i presi per ritrasformare lo spazio del vettore v_{target} nella dimensione originale:

$$W' * v = z \in R^V$$

Forth) Utilizziamo la funzione $softmax(x)$ per trasformare il vettore z in un vettore normalizzato con elementi compresi tra $[0, 1]$ e che la somma di tutti gli elementi sia uguale a 1, quindi come una probabilità che una determinata parola in output in un determinato contesto.

$$y_{target} = softmax(z_{target})$$

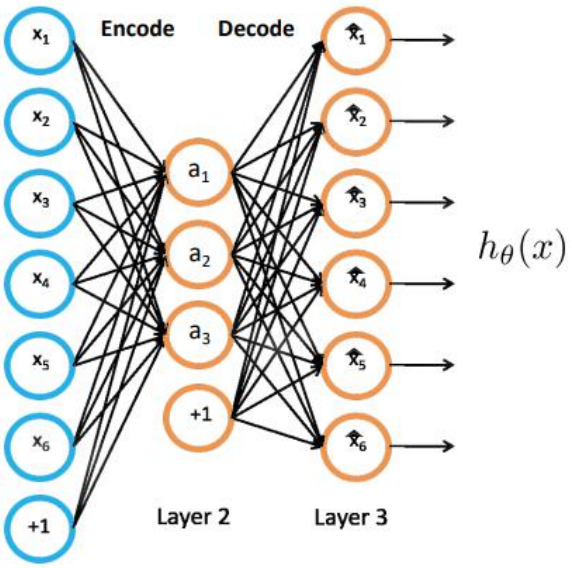
Fifth) Prendiamo il valore che più si avvicina ad y tra le parole presenti nel vocabolario.

Come cicla per trovare i migliori pesi? Giusto una idea veloce

Massimizzando la probabilità di predizione del contesto corretto per una determinata parola di input.

12.5 Autoencoders

Gli autoencoder sono un tipo di rete neurale artificiale utilizzata per la compressione dei dati. L'obiettivo principale degli autoencoder è di ridurre la dimensione di un insieme di dati di input mantenendo allo stesso tempo le informazioni più importanti.



Encoder

Accetta i dati di input e li trasforma in una rappresentazione compressa di dimensione inferiore.

Decoder

Riceve la rappresentazione compressa e cerca di ricostruire l'input originale.

12.5.1 Training

Durante l'addestramento degli autoencoder, l'obiettivo è minimizzare l'errore di ricostruzione tra l'input originale e la sua ricostruzione.

$$\min_{\theta} \left|\left|h_\theta(x) - x\right|\right|^2$$

Ciò significa che l'encoder deve trovare una rappresentazione compressa che mantenga tutte le informazioni importanti dell'input originale, mentre il decoder deve essere in grado di utilizzare questa rappresentazione compressa per ricostruire l'input originale con la massima precisione possibile.

12.5.2 Modello

L'obbiettivo è di mappare l'input $x \in [0,1]^d$in uno spazio di dimensionalità inferiore (d') l'input x attraverso una rapprsentazione complessa y:

$$y = s(Wx + b)$$

$W \in R^{d'*d}$, un termine di bias b e s una funzione di attivazione **non-lineare** (come la sigmoid).
Il **decoder** mappa y in una ricostruzione z dell'input originale x:

$$z = s(W'y + b')$$

z è visto come una predizion e di x.

12.5.3 Training a sparse Autoencored

$$\min_{\theta} \left|\left|h_\theta(x) - x\right|\right|^2 + \lambda \sum_i |a_i|$$

In questo caso si vuole addestrare un autoencoder che produca una rappresentazione complessa più "sparsa", ovvero che per ogni campione di input sono attivi solo pochi neuroni.
Il termine di penalizzazione è caratterizzato da:
First) λ che determina la forza di penalizzazione.
Second) $\sum_i |a_i|$ la norma-1 degli output dei neuroni del layer nascosto incoraggiando la rete ad avere pochi neuroni attivi per ogni campione di input.