Introduction to Machine Learning

X

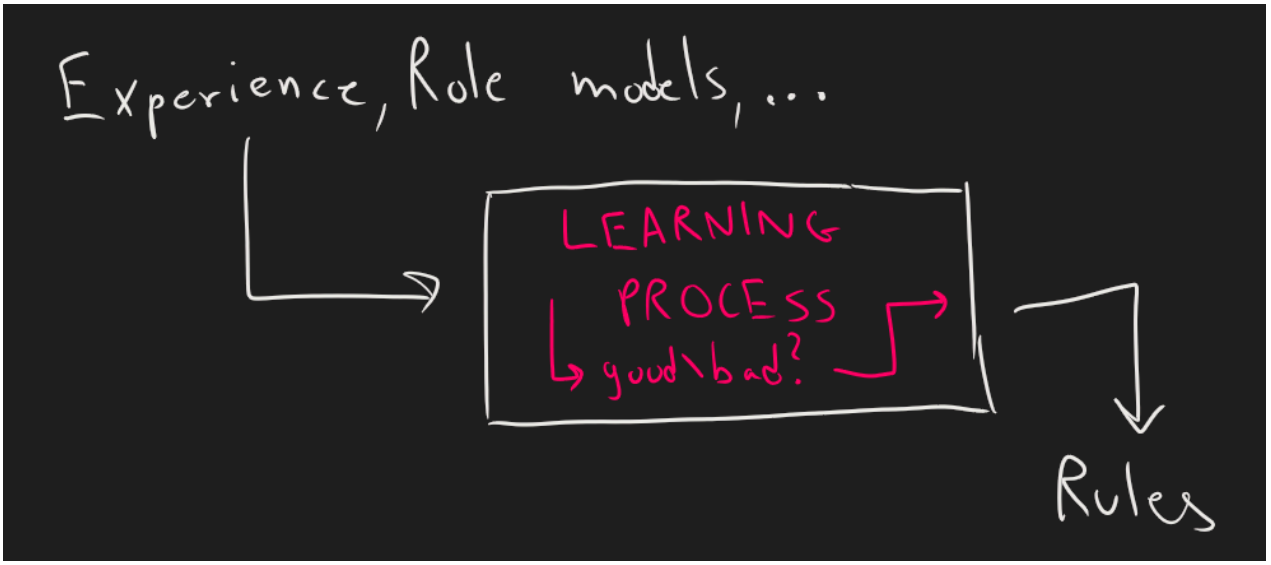
Heuristical introduction to Machine Learning. Preliminary considerations on the learning process. Definition of Machine Learning. Relation between Machine Learning and Artificial Intelligence. The Machine Learning Paradigma. Approaches to Machine Learning.

X

1. The Learning Process

Before talking about *machine learning*, we should cover the bases. What is a *learning process*? What does *"learning"* mean, exactly?

THE LEARNING PROCESS. We can think of it in this way: we have an *"input"*, that is *experience*, *role models* and so on...; then we have some sort of *process*, where we *integrate and implement information* and understand whether something is *"good"* or *"bad"*. And out of this, we infer the *rules* in our *subconscious*, which can then be used to get *new answers* for *new problems*.



Q. How do we transpose this process to machines?
A. This is the *objective of the course*; the result is what we call *machine learning*.

2. Machine Learning

DEFINITION. So we can define *Machine Learning* as an *AI technique* that teaches computers to learn from *experience*. In particular, Machine Learning is a specific subset of Artificial Intelligence:

$$\mathrm{ML} \subset \mathrm{AI}$$

REMARK. By definition, *machine learning* is something that develops by *experience*. So, we have a new paradigma: that is, instead of knowing the *context* that has to be controlled, our goal is to *know the context* itself. For example, we might want to find out relations between numbers in a dataset.

EXAMPLE. Suppose we have the following number set:

$$5 \mapsto 25$$

Our first possible approach could be to define the following rule: $D_1 : x - 20$
This works well, as $25 - 20 = 5$. Now suppose we have the new insight:

$$\begin{aligned} 5 &\mapsto 25 \\ \text{new rule : } \quad 4 &\mapsto 16 \end{aligned}$$

What now? D_1 is no longer good! Because $16 - 20 = -4$. However, if we slightly modify it to make it $D_2 : |x - 20|$, we can see now it is good!
And then we can go on as we add new end results...

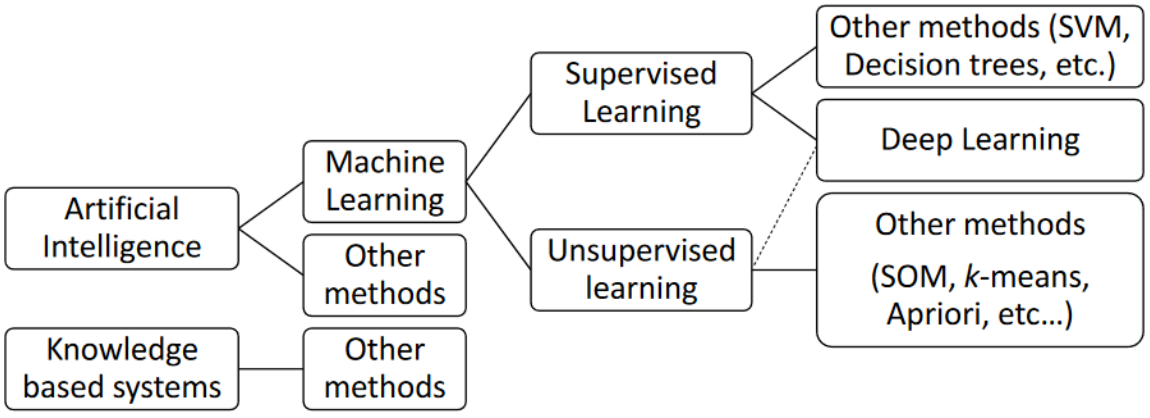
3. Other Approaches to AI

We can deduct that *machine learning* is not the only way to make *artificial intelligence*. We have the other following approaches.

1. **Neural Networks**: Our end goal here is to *replicate the human brain structure* by *"emulating"* our neural connections with the so-called *neural networks*.
2. **Genetic Programming**: We use the *Darwin's law of survival*, where instead of *species* we have *solutions*, and the *best evaluated solution* is the one that survives and gets modified (*"mutated"*)
3. **Probabilistic Approach**: Here we use statistics and probability to let machines pick options. In particular, Bayesian statistics will help us.
4. **Logical Approach**: Here we can *automate reasoning* with simple logic.

4. Approaches to Machine Learning

Now, delving deeper into *machine learning*, we have more specific approaches, as we can see in this figure:



In particular, we will cover *supervised and unsupervised* learning later.

X

Supervised and Unsupervised Learning

X

Definition of supervised and unsupervised learning. Problems of supervised learning: regression and classification. Remark: case where both regression and classification are needed. Problems of unsupervised learning: clustering and classification. Cluster analysis: definition and motivations.

X

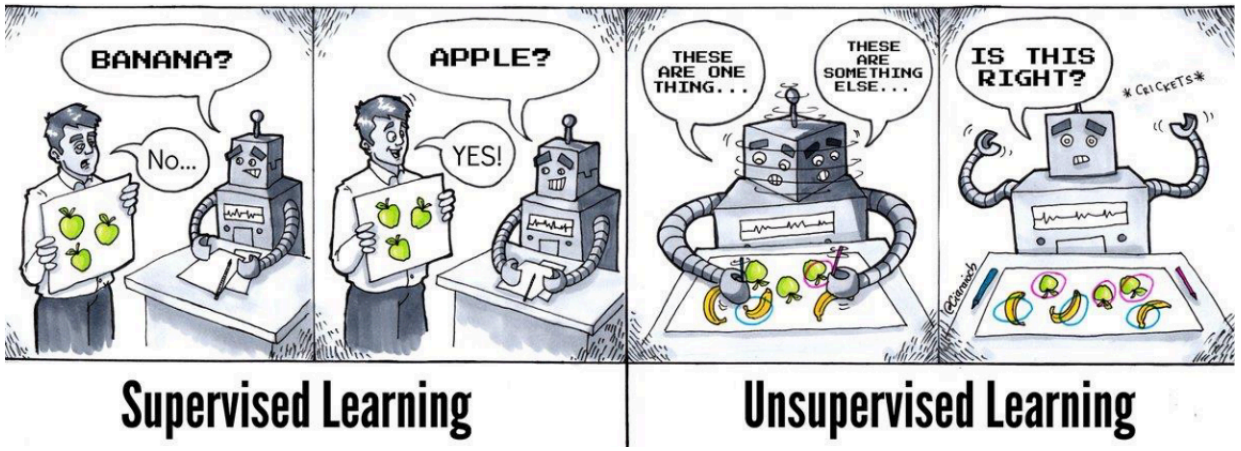
0. Voci correlate

- Introduction to Machine Learning

1. Supervised and Unsupervised Learning

DEFINITION. (*Supervised Learning*)
Supervised learning is a machine learning technique where the *machine is trained dataset with labels*; so in a way, it has the *answers* to the *context* it needs to infer from. The end object of this is to *learn relationship* between *features and target*. This is also used to *classify unseen data points*.

DEFINITION. (*Unsupervised Learning*)
Unsupervised learning is a machine learning technique where the machine is trained with *unlabeled data*; so, without *"supervision"* from above. Used in particular to *find optimal clusters*.



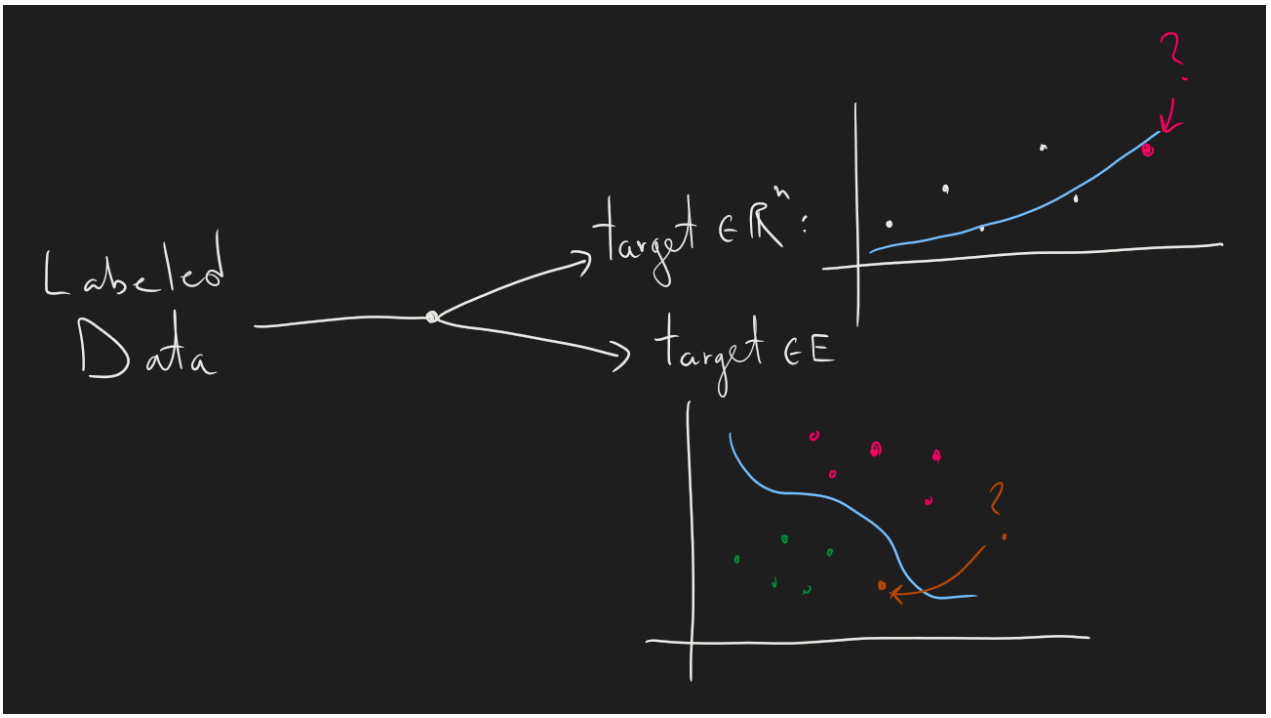
X

2. Supervised Learning

As we have seen before, *unsupervised learning* is used to train algorithms which can *classify data* or *predict outcomes*. We precisely define them as follows:

DEFINITION. (*Regression and classification problem*)
Suppose we have a *dataset* and we split it between *independent variables* and *target variables*.

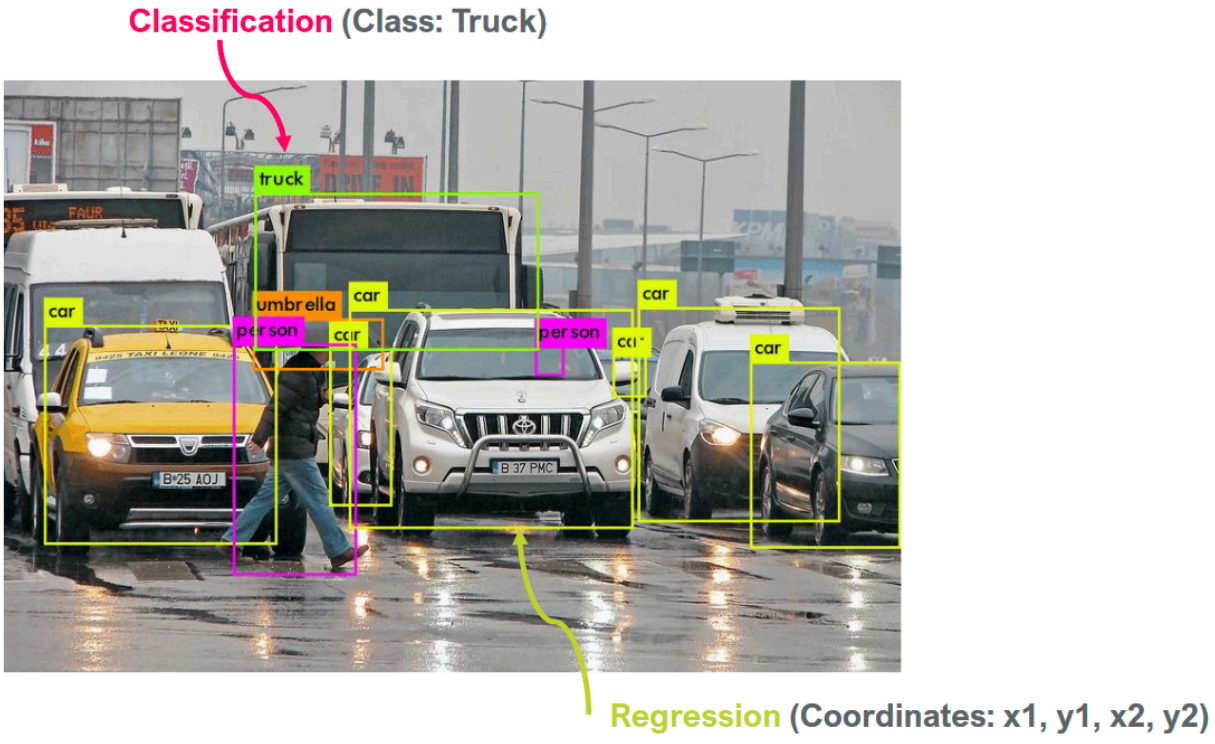
- If the *target variables* which we want to predict are (or is) numerical, then we have a *regression problem*
- If the target variables are *categorical*, then we have a *classification problem*



EXAMPLES.

- Regression problems: *predicting the price of a house* or the *stock market values*
- Classification problems: *predicting the probability of someone having diabetes* or *which animal is in a photo* or *which number is in a picture*

Q. Can we have both regress and classification in a same model?
A. Yes, there are such cases. Think of *automated driving systems*: they have first to predict *which pixels* represent some sort of relevant object (*car, person, ...*), which can be considered a *regression problem*. Then, as we have these *boxed*, we have also to predict their category: so whether they're a car, a person, an umbrella, anything.

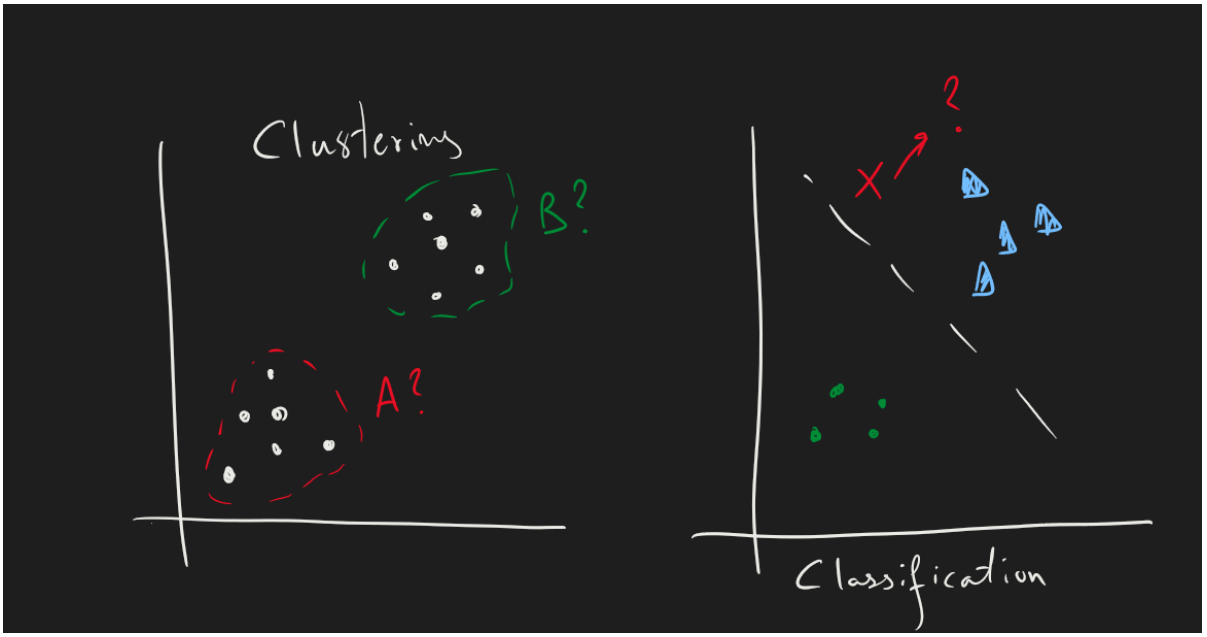


3. Unsupervised Learning

Now we're going to briefly overview *unsupervised learning*

DEFINITION. (*Clustering and classification problem*)

1. A *clustering problem* is a problem where the algorithm must be able to *distinguish the groups* in an unlabeled dataset.
2. A *classification problem* is when the algorithm should be able to predict whether a *new entry* belongs to a certain *predefined class*. Note that it is different from *supervised learning*, as data is still unlabeled here.



We're going to delve deeper into *clustering*.

CLUSTER ANALYSIS. *Clustering* is a very powerful for *cluster analysis*, which is a basic conceptual activity of human: it is also fundamental for science! However, it becomes harder as we have more dimensions. So, the possibility of reducting the complexity would be one of the most powerful tools here. Moreover, we have to optimize two things: *intra-cluster distance* (to minimize) and *inter-cluster distance* (to maximize).

X

Models in Machine Learning

X

Models in machine learning: definition. Types of models: fixed, parametric and non-parametric.

X

---

0. Voci correlate

- Introduction to Machine Learning

1. Definition of Model

DEFINITION. (*Model*)
In the context of *Machine Learning*, a *model* is the *output of machine learning algorithms* run on data.

Basically, they are some sort of equations or instructions which give a result, when data is given.

Mainly, we have three types of model:

- **Fixed model**: not driven at all; this type of model is not used in Machine Learning methods.
- **Parametric model**: model-driven
- **Non-parametric model**: data-driven

X

2. Fixed Models

DEFINITION. (*Fixed model*)
Fixed models are the *most simple form*, as they are only *closed-form equations* that define the relation between input and output. This is suitable for *simple and fully understood problems*.

EXAMPLE. As an example we have the following formula that can calculate the amount of time needed for any object to hit the ground on earth:

$$t(h) = \sqrt{\frac{2h}{9.8}}$$

3. Parametric Models

DEFINITION. (*Parametric Model*)
By generalizing *fixed models* more, we get *parametric models*: in this case we have certain *parameters* that go undefined, which are usually chosen by examining *data*. However, we still have certain *assumptions*: we have to know the *specific distribution* of the data.

ADVANTAGES.

- Easy to understand and interpret
- Faster to train
- Does not require a huge amount of data

DISADVANTAGES.

- Limited by the functional form chosen, no matter the amount of data
- Suited to simple problems (such as linear problems)
- Does not have the best fit
- Requires data to be pre-processed! (Introduction to Data Preprocessing)

EXAMPLE. Take the previous example; now we generalize it for *any planet*. In this case we have

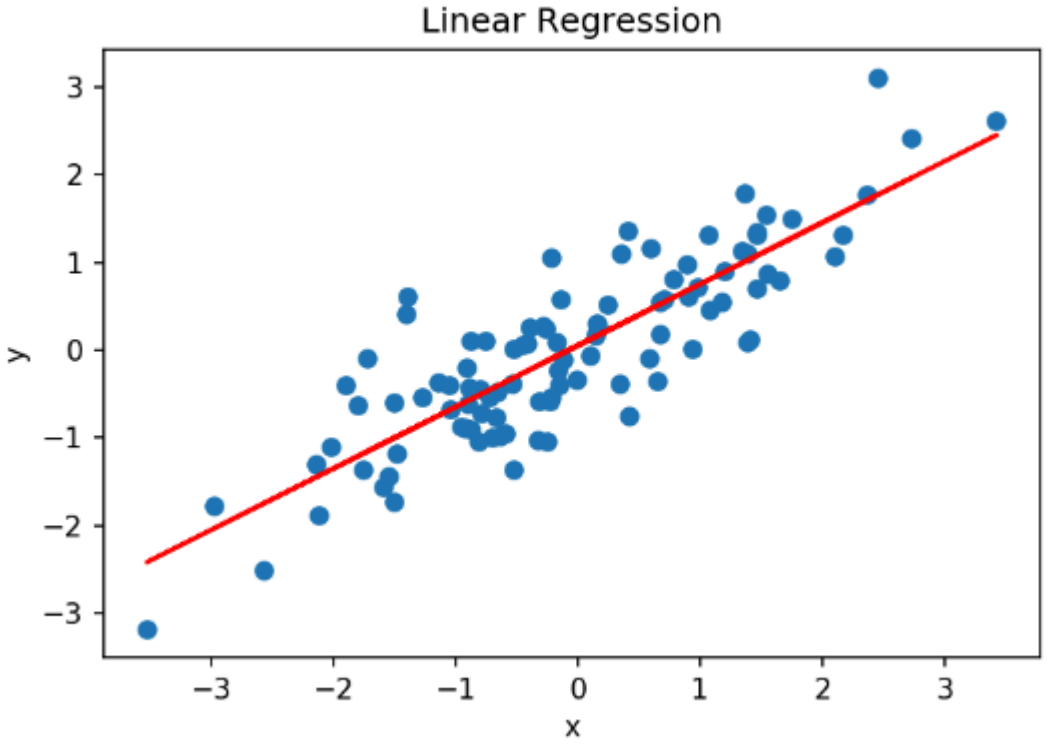
$$t_g(h) = \sqrt{\frac{2h}{g}}$$

We need to estimate the parameter g now, and it can be done with experimental data. For example, if we want to do it in Mars, we have to go to mars and actually collect data!

EXAMPLE. The most *common example* of a *parametric model* is the *linear regression*, which assumes a linear relationship between input and outputs. In particular, for a given n number of features (inputs), we have the *linear regression* defined as

$$y_i(\underline{x}) = \beta_0 + \langle \underline{\beta}, \underline{x_i} \rangle + \varepsilon_i$$

We will see that we can *"guess"* the parameters $\underline{\beta}$ by using simple calculus. (Not really simple though, especially in vector cases...)



X

4. Non-Parametric Models

DEFINITION. (*Non-parametric Models*)
If we have a really *complex problem* where the relationship between input and output is not even known, we can use *models* that *rely heavily on data*; these are *non-parametric models*, known better as *data-driven models*. These are models whose form change with different datasets.

ADVANTAGES.

- These models are very flexible and do not make any assumptions
- Has higher accuracy
- No need to preprocess data!

DISADVANTAGES.

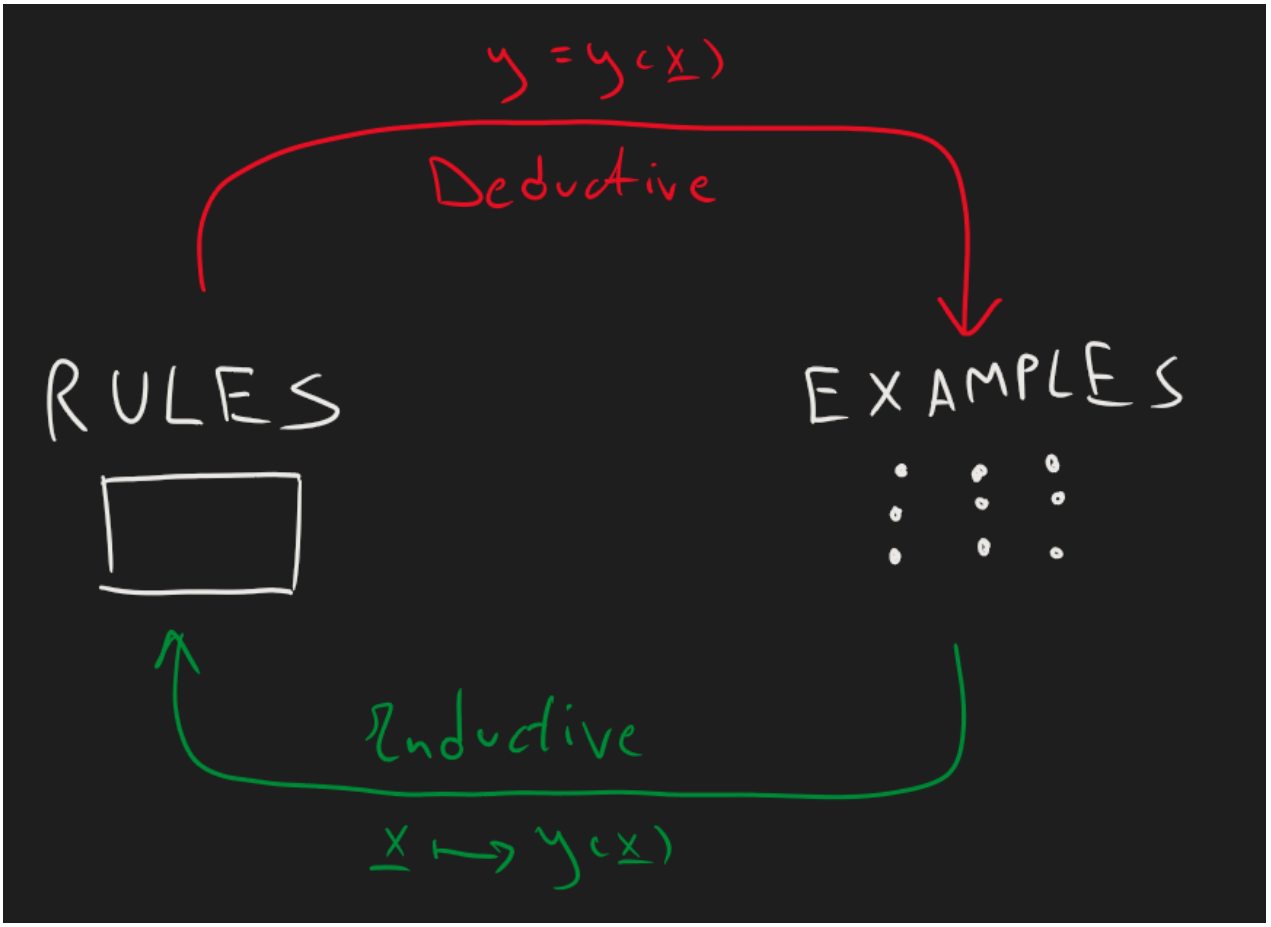
- Needs a lot of data
- Is slower to train
- High risk of overfitting; we will see the problem later
- Is hardly explainable in certain decisions

EXAMPLES. We will see some examples of *non-parametric models*, such as the *KNN* (k-nearest neighbours), or the *decision trees*.

X

5. Parametric and Non-Parametric

The difference between *parametric* and *non-parametric models* lies in the *"method" of reasoning*; with *parametric models* we have something close to *deductive reasoning*, as it gets results from the general rule; whereas *non-parametric models* get the examples first, and then infer the general rule, which is *inductive reasoning*.



For a final confrontation, we shall quote the book *"Artificial Intelligence: a Modern Approach."* (*Russel, Stuart J.*):

- **PARAMETRIC MODELS** - "A learning model that summarizes data with a *set of parameters of fixed size* (independent of the number of training examples) is called a parametric model. No matter how much data you throw at a parametric model, *it won't change its mind about how many parameters it needs*."
- **NON-PARAMETRIC MODELS** - "Nonparametric methods are good when you have *a lot of data and no prior knowledge*, and when you don't want to worry too much about choosing just the right features."
  Moreover, for *parametric* and *non-parametric* models we can define *hyperparameters* and *parameters*.
- *Parameters* are the *"result"* of their training; for example they are the coefficients $\underline{\beta}$ in a linear regression
- *Hyperparameters* are *determined* prior to modelling; for example they are the number of branches in a decision tree

X

Problems in machine learning: model selection, complexity, separability and overfitting.

X

0. Voci correlate

- Models in Machine Learning

1. Introduction

When we build a *model in machine learning*, we usually need to consider certain factors. If it not were for them, we could always use the *best algorithm*, such as *deep learning*! However, that is not the case. Let us see the reasons. In particular, we will see:

- The problem of algorithm selection
- The problem of complexity
- The problem of separability
- The problem of overfitting

X

2. Algorithm Selection

This problem starts with the following premise.

THEOREM. (*No free-lunch theorem*)
"If an algorithm performs better than random search on some class of problems then it must perform worse than random search on the remaining problems." (*David Wolpert and William G. Macready*). In other words, we always have some *trade-offs* between different models.

COROLLARY. Therefore, there is no *single machine learning algorithm* that is universally the *best-performing* for all problems! This is the reason we have to consider a good *algorithm selection* for each *problem presented*.

3. Complexity

Sometimes there are problems that are *too complex to deal with*: this means, we have a lot of *independent variables*. This makes collecting data for training *unfeasible*, as a lot of it is required. For this reason, we have to pose a series of questions.

Q. Which features do we choose and discard? Which are relevant for the task?
Q. Which is the right number of features to use? If too small, we cannot distinguish between classes. If too large, we have the so-called "curse of dimensionality".

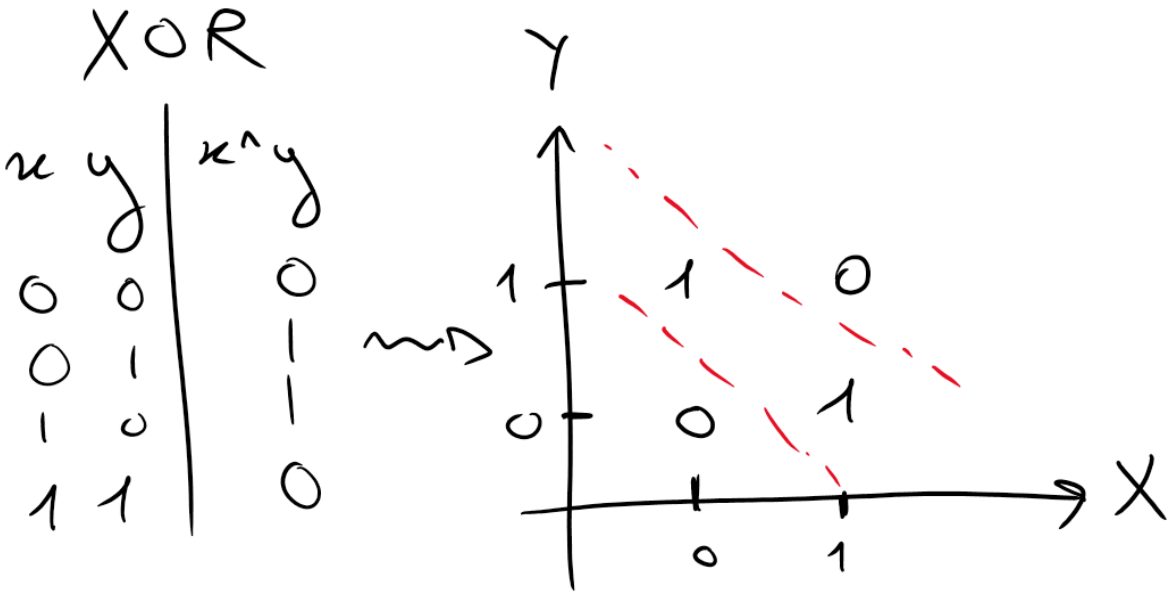
4. Separability

Is the problem *separable*? That is, if we have *data* that has clear distinction between groups.

1. If separable, then it's possible to have *no errors*.
2. If not separable, then it's impossible to have *no errors*
   A case of *non-separability* could be where we have *two cases with same independent values* but different *target values*.

Moreover, another common problem with classification problems is *linear separability*: in this case, we are asking whether our dataset can have linear decision boundaries.

- An example of a non-linear separable problem is the *XOR* conjunction, as we would need at least *two lines* to define a clear decision boundary for the target variable.

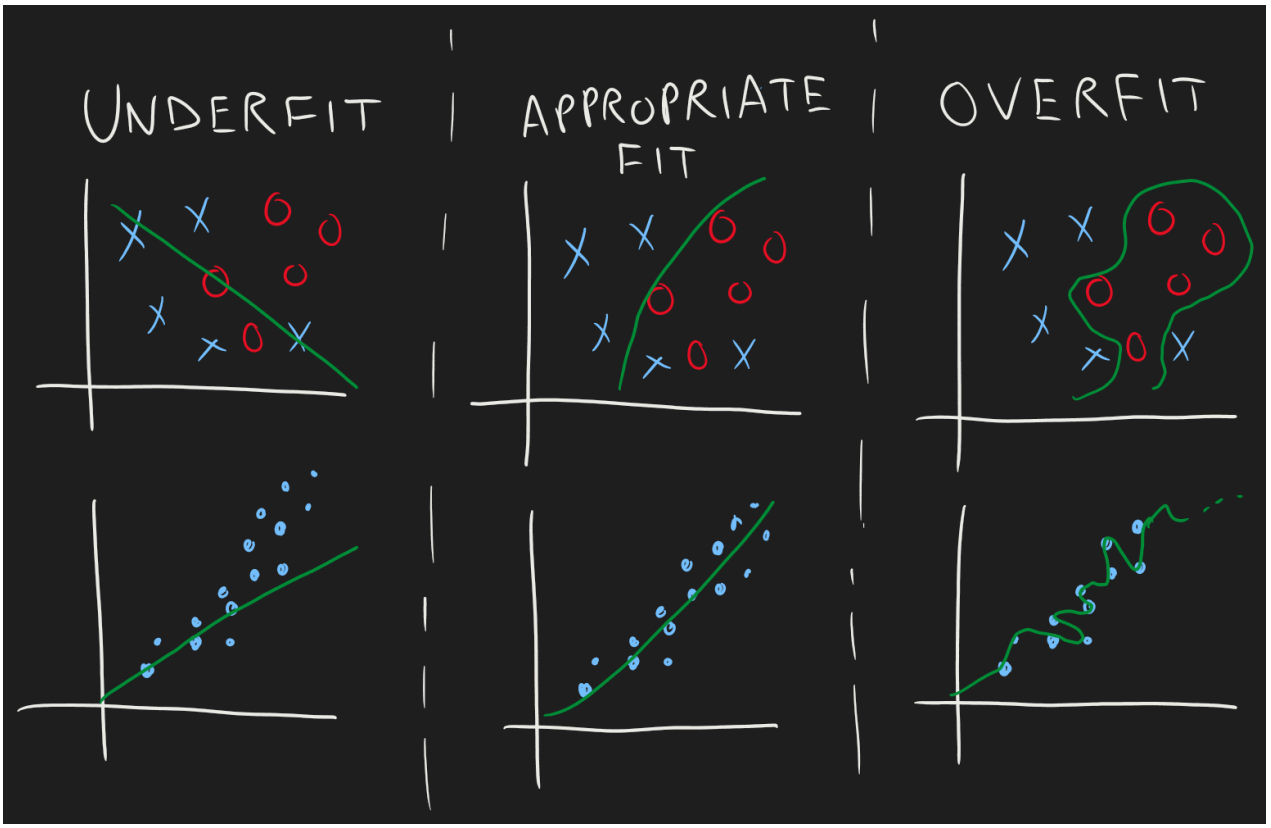


5. Overfitting

Lastly, we have to account for *models* that rely on *training data to learn*. If it learns *too much*, it might end up *"memorizing"* the training data, instead of extracting useful insights. This means, the model is *too biased* on the *training data*, which is bad news for future cases.

As an analogy, we can think of *students* memorizing information for a test, only to forget it later.

Similarly, we can have *underfitting* if the data used is too small or the model is too simple.



X

Methodology in Data Science

X

Definition of methodology in the Data Science context. Common methodologies: CRISP-DM, SEMMA and OSEMN.

X

0. Voci correlate

- Introduction to Data Preprocessing
- Introduction to Data Mining
- Models in Machine Learning

1. Definition of Methodology

DEFINITION. (*Methodology*)
A *methodology*, in the context of *Data Science*, is a *framework for recording experience*, allowing projects to be replicated. In particular it describes a series of processes which have the goal of turning *data* into *information*.

Today we have a lot of methodologies, but the commonly used are the following:

- **CRISP-DM** (1996)
- **SEMMA** (≈ 2000)

- **OSEMN** (2010)

X

2. CRISP-DM

CRISP-DM (*Cross-Industry Standard Process for Data Mining*). Designed in 1996, is considered to be the most *"complete"* methodology. In fact, this comprises *business understanding*.

Moreover, this is not a *linear process*, as in some steps we have *loops*. Let us look at the main steps.

1. **BUSINESS UNDERSTANDING.** Define objectives and requirements of the project.
2. **DATA UNDERSTANDING AND PREPARATION.** Mainly, we have to *prepare* the *data*; in particular first we have to *look* at the data to identify problems, and then *prepare* it to our needs.
3. **MODELLING.** Choose models, calibrate parameters and optimize the chosen model.
4. **EVALUATION.** Evaluate the model according to the criterions defined in the *business understanding* (1) step.
5. **DEPLOYMENT.** Deploy the model with a repeatable implementation; might not be necessary, it depends on the requirements.

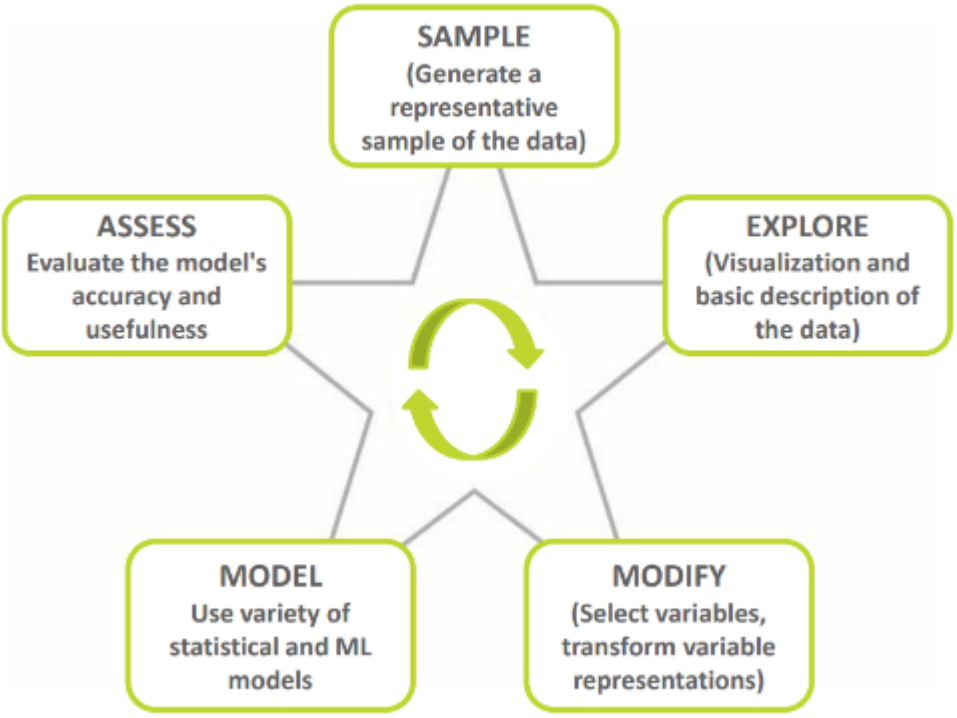
In particular, the course *"Data Preprocessing"* will only look at the first two steps.

3. SEMMA

SEMMA (*Sample, Explore, Modify, Model and Assess*). Designed around the year 2000 by the SAS company analytics. It is a more *"simple"* version of CRISP-DM, where business undersanding is no longer a requirmeent. It is a cyclical process, with the following steps:

- *Generate a sample of the data*

- *Explore and visualize data*
- *Modify the data*
- *Make a model*
- *Assess the model*



In the context of *Data Preprocessing*, we look at only the first three steps: Sample, Explore and Modify.

4. OSEMN

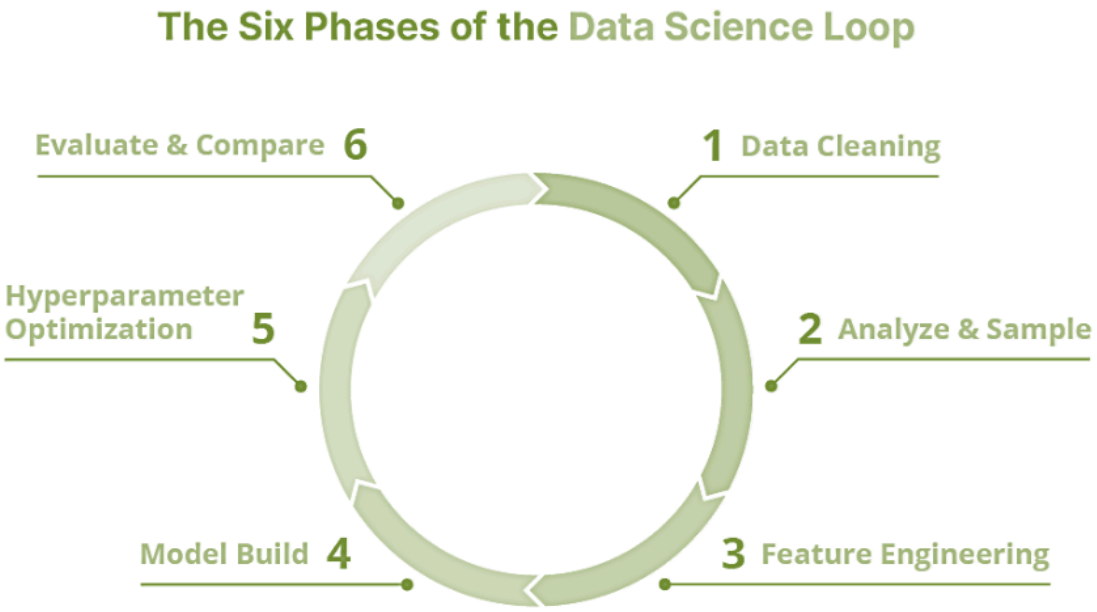
OSEMN (*Obtain, Scrub, Explore, Model, Interpret*). The most recent framework, is mostly the same as the other models. This particularly enphasizes on the *web sources*, and this framework is better documented.

5. Generalized Data Science Process

We can generalize the aformentioned frameworks with the following:

1. **DATA CLEANING.** Make sure the data is *"clean"* enough for the *algorithms*; so *no missing values*, *is correct*, has *acceptable values*.
2. **ANALYZE AND SAMPLE DATA.** Make sure that our quantity of data makes sense, explore data to see any *patterns* or how it is *distributed*.
3. **IMPROVE DATA.** If needed, apply transformations, create new features, or select some features and discard others.
4. **APPLY MODELS.** Choose and build the appropriare modeling technique

5. **IMPROVE MODEL.** Do hypeparameter tuning
6. **EVALUATE AND COMPARE.** Evaluate models by comparing their predictions with real data; pick the model based on performance and other metrics, such as *scalability, interpretability, etc...*



X

Linear Regression

X

Linear regression. Main idea, definition of simple and multiple linear regression, metrics for evaluating linear regressions. Optimization formulas for linear regression. Advantages and disadvantages of linear regression.

X

0. Voci correlate

- Supervised and Unsupervised Learning

1. Fundamental Idea of Linear Regression

#Definizione

> Definizione (linear regression).
>
> *Linear regression* is a *parametric model* (Models in Machine Learning > ^bca4c6) that fits a *straight line* (or surface if in $\mathbb{R}^{n\geq 2}$) which minimizes the *error* between predicted and actual output in a regression problem.

If in the regression problem we have only one *explanatory variable*, then the model is a *linear regression*; otherwise it is a *multiple* linear regression.

Mathematically, we have the general formula for a multiple linear regression:

$$\boxed{\hat{y} = \beta_0 + \langle \underline{\beta}, \underline{x} \rangle + \varepsilon}$$

So the main idea is the following:

- Design a line $y : mx + b$
- Use the *least-squares method* to fit the line to our data
- Calculate R^2 and p-values to evaluate our model

In particular, the function that calculates the error between predicted and actual output will be referred as *loss function* $\mathcal{L}(\hat{y}_i, y_i)$. Let us define the loss function that is commonly used for linear regressions.

#Definizione

Definizione (sum of residuals).

Let $(\hat{y}_n)_n$ be the predicted values from a model $\mathcal{M}$, and let $(y_n)_n$ be the actual values of a training dataset.
Then the *residual sum of squares* $S_R^2 = RSS$ is defined as

$$RSS = \sum_n (\hat{y}_n - y_n)^2$$

X

2. Optimization of Linear Regressions

Let us take a simple linear regression of type

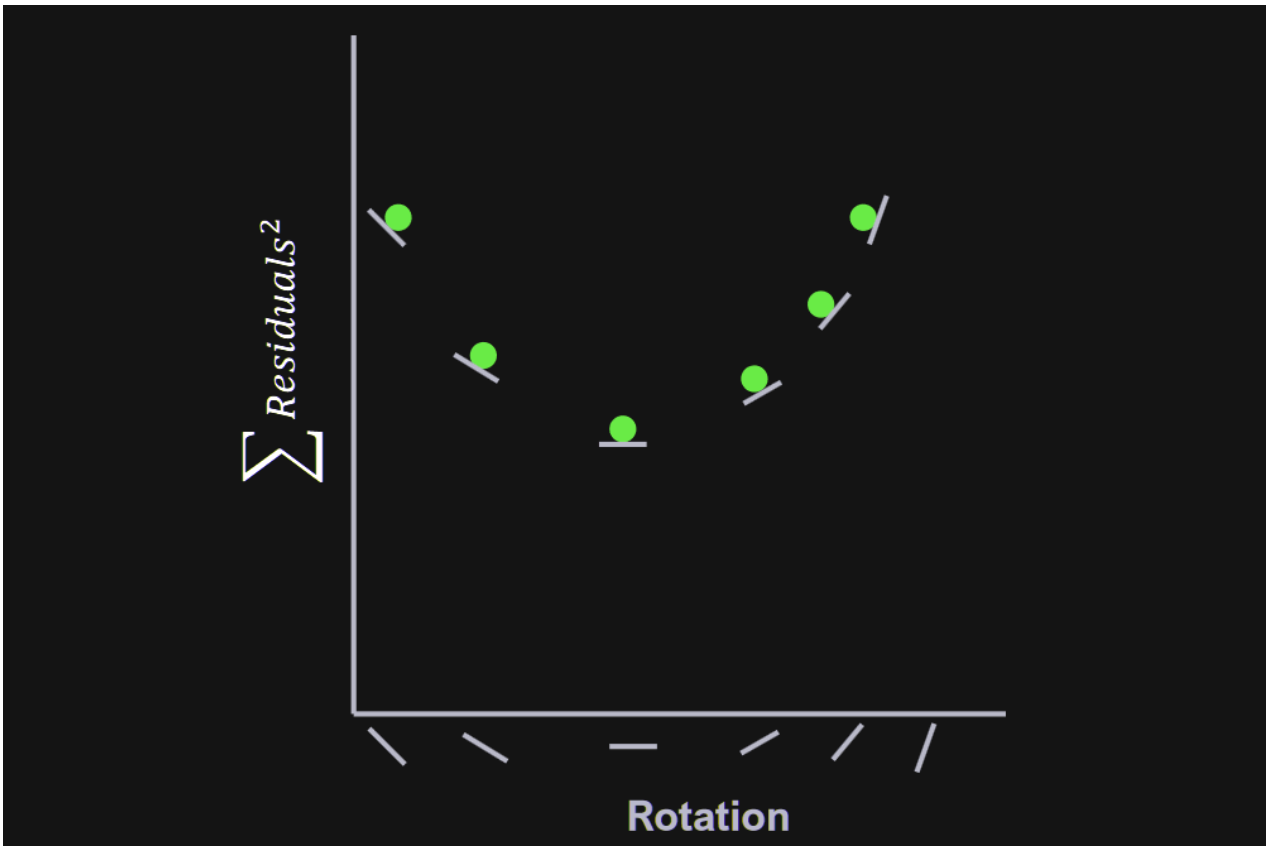
$$\hat{y} = \beta_0 + \beta_1 x$$

We notice that the can express the *loss function* in terms of β_0, β_1; infact we would have

$$RSS = \sum_n (\hat{y}_n - y_n)^2 = \sum_n (\beta_0 + \beta_1 x_n - y_n)^2$$

In particular, we have some sort of *parabola*. Therefore, we can minimize RSS by taking its gradient in respect of the terms β_0, β_1. We omit the calculations and enounce the following formulas to optimize the loss function.

In geometrical terms, by *"rotating"* our line we have different results for RSS.



#Teorema

> Teorema (formulas for optimizing the residual sum of squares).
>
> Let X be the explanatory variable and Y the target variable in a regression problem, respectively with mean $\overline{x}, \overline{y}$.
>
> Supposing our model $\mathcal{M}$ is a simple linear regression, we have the following coefficients which minimize RSS:
>
> $$\beta_1 = \frac{\sum_n [(x_n - \overline{x})(y_n - \overline{y})]}{\sum_n (x_n - \overline{x})^2}, \beta_0 = \overline{y} - \beta_1 x$$

X

3. Advantages and Disadvantages of Linear Regression

3. Advantages

Let us see the *main advantage* of this model

Interpretability. As this model is extremely intuitive to conceptualize, it is also very intuitive to interpret. In fact we have that:

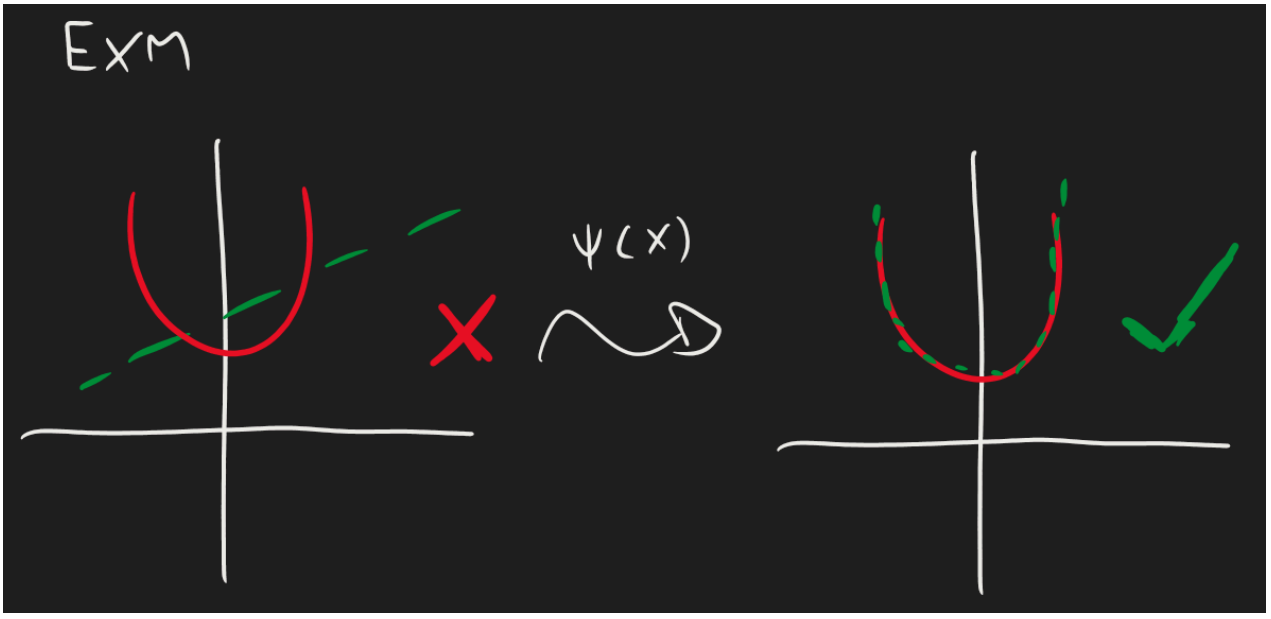
- For each $\beta_n = \pi_n(\underline{\beta})$ component of the coefficients, it can be interpreted as a *direct correlation* between an explanatory variable and the target variable.
- With statistical evaluations, we can calculate the p-values for the explanatory variables, which can be seen as a *"significance"* for the output. Smaller it is (around < 0.05), better it is. This will be reveal to be relevant for *feature selectioning*.

4. Disadvantages

However, not always a linear regression can be applied. In fact, it has *certain assumptions* to let itself work. The main disadvantage is that it requires a lot of assumptions.

1. **Linear relationship.** The first and foremost assumption is that there is a *linear relation* between the variables. If not, our model *will never* be able to properly fit the data.

A possible remedy for this problem is applying a transformation ψ to our target variables, and fit our line on $\psi(\underline{x})$.



1. **Multicollinearity.** Having highly-correlated variables might represent a problem as well, since that our model might assign a *"higher importance"* to certain variables.
   A possible remedy might be:

- If we have *perfect multicollinearity*, then we should drop one of the variables
- If we have *high correlation*, then we can drop the least-significant variable to the target.
  Alternatively, we can resort to feature engineering.

3. **I.I.D. variables.** Lastly, we are also assuming that variables are *identically independently distributed* (counterexample: time-dependent data). If not, then the *linear regression* is not a good choice for the problem.

X

Logistic Regression

X

Logistic regression. Definition of logistic regression, logistic function. Differences from linear regression.

X

---

0. Voci correlate

- Supervised and Unsupervised Learning

1. Preliminary Mathematics

#Definizione

> Definizione (logistic function or sigmoid function).
>
> Let the *logistic* (*sigmoid*) function be defined as
>
> $$\sigma(x) := \frac{1}{1 + e^{-x}}$$

#Teorema

> Teorema (properties of the sigmoid).
>
> The sigmoid is continuous on the real numbers and $\sigma(\mathbb{R}) = [0, 1]$.

#Dimostrazione

PROOF of Teorema 2 (properties of the sigmoid)
As σ is continuous (since it has no domain problems) and decreasing ($\sigma(e^{-x}) = O(n^{-1})$), by calculating the limits

$$\lim_{x \to -\infty} \sigma(x) = 0 \land \lim_{x \to +\infty} \sigma(x) = 1$$

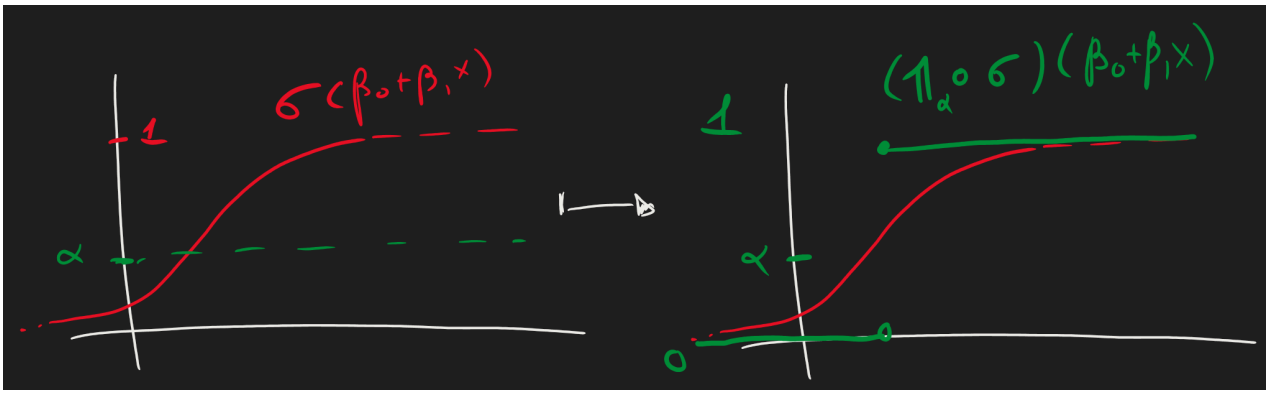
we have by the *intermediate values theorem* that σ's image is $[0, 1]$.

X

2. Logistic Regression

#Definizione

> Definizione (logistic regression).
>
> *Logistic regression* is a classification model that estimates the relationship between independent variables and a categorical target variable.
>
> It is *binary* if there are only two potential outputs; otherwise it's said to be *multiclass*.
>
> In particular, the *binary logistic regression* uses the following formula to predict the probability of an instance belonging to a class or not:
>
> $$\sigma(\beta_0 + \beta_1 x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$
>
> Then, to transform it to a class we apply the *threshold function* (with the hyperparameter α):
>
> $$\mathbb{1}_\alpha(p) = \begin{cases} 0, p < \alpha \\ 1, p \geq \alpha \end{cases}$$



#Osservazione

> Osservazione (difference from linear regression).
>
> The main difference from linear regression is that here the predicted value is a *boolean* (or a discrete number), whereas in the previous case we had a *continuous value* with could be *"out of bound"* for us.

X

3. Disadvantages of Logistic Regression

#Osservazione

> Osservazione (optimization method is harder).
>
> Unlike linear regression, we have no way of measuring *"residuals"*. Instead, we will use the *maximum likelihood estimation* to estimate the parameters β_0, β_1; the main idea is to *"shift"* the curve with the *"best likelihood of happening"*, in accordance to the observed data.
>
> Unfortunately, with this technique we have no closed-form formulas, and we need to numerically estimate the parameters.

#Osservazione

> Osservazione (interpretability is non-linear).
>
> Here interpretability becomes harder as the values $\underline{\beta}$ do not represent a linear correlation with our target variable $\hat{y}$.
>
> $$\underline{\beta} \xrightarrow{?} \hat{y}$$

X

Model Evaluation

X

Model evaluation: performance metrics. Definition of hyperparameters and parameters. Performance metrics for regression problems: MAE, MSE, RMSE. Performance metrics for classification problems: Accuracy, recall, F1-score, ROC curve.

X

0. Voci correlate

- Methodology in Data Science

1. Performance Metrics for Regression Problems

Q. How do we determine if our model *"performs well"*?
A. With performance metrics, where we measure a model's *"accuracy"*.

In regression problems we have different metrics.

#Definizione

> Definizione (MAE, MSE and RMSE).
>
> Let $(\hat{y}_n)_n$ the predicted outputs from a model and let $(y_n)_n$ be the actual outputs. Then we have the following performance metrics:
>
> - *Mean Absolute Error*
>
> $$MAE = \frac{1}{n}\sum_n \hat{y}_n - y_n$$
>
> - *Mean Square Error*
>
> $$MSE = \frac{1}{n}\sum_n (\hat{y}_n - y_n)^2$$
>
> - *Root Mean Square Error*
>
> $$RMSE = \sqrt{\frac{1}{n}\sum_n (\hat{y}_n - y_n)^2}$$
>
> - *Coefficient of determination*
>
> $$R^2 = 1 - \frac{\sum_n (\hat{y}_n - y_n)^2}{\sum_n (\bar{y}_n - y_n)}$$

#Osservazione

> Osservazione (meaning of the performance metrics).
>
> The *MAE, MSE, RMSE* evaluates the *"weighted distances"* between the predicted and actual values; note that *RMSE* makes bigger errors *"worse"*.

> Whereas the *coefficient of determination* R^2 evaluates the model's capability to explain the variability of data. We have $R^2 \in [-1, 1]$.

X

2. Performance Metrics for Classification Problems

In the case of classification problems, we can simply evaluate the *proportion* of *"right guesses"* with *"actual answers"*.

#Definizione

> Definizione (confusion matrix).
>
> Let $(\hat{y}_n)_n$ be the predicted classes by a model, let $(y_n)_n$ be the actual classes. Assume there are only two classes: *positive* and *negative*. We denote the actual positives and negatives with P, N and the predicted ones as $\hat{P}, \hat{N}$.
>
> Defining the following:
>
> - *True positives and negatives* as the intersection between actual positives and predicted positives (or actual negatives and predicted negatives): $TP = \hat{P} \cap P$ and $TN = \hat{N} \cap N$
> - *False positives and negatives* as the intersection between actual negatives and predicted positives (or actual positives and predicted negatives): $FP = \hat{P} \cap N$ and $FN = \hat{N} \cap P$
>
> We define the following matrix the object that contains all of the defined objects as the *confusion matrix of our model*:
>
> $$\text{Confusion Matrix}(\mathcal{M}) = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix}$$

#Proposizione

> Proposizione (reverting transformations).
>
> Note that from TP, TN, FP, FN we can get $P, N, \hat{P}, \hat{N}$.
>
> $$\begin{aligned} P &= TP \cup FN \\ N &= FP \cup TN \\ \hat{P} &= TP \cup FP \\ \hat{N} &= TN \cup FN \end{aligned}$$

#Dimostrazione

PROOF of Proposizione 4 (reverting transformations)
Provable by using *De Morgan's Laws*. ■

#Osservazione

> Osservazione (each element of confusion matrix might have different importances).

> In certain contexts, *false negatives* could be *"worse"* than *false positives* (or viceversa); a typical situation is disease detection. A false positive just causes a minor inconvenience, as the patient would have to do more medical inspections; however, a false negative could lead even to someone's death, as they are oblivious of their disease.
>
> This problem is especially important with *unbalanced datasets*.
>
> So, our goal will be to *evaluate* those aspects in a balanced manner.

#Definizione

Definizione (precision and recall).

Let TP, FP, TN, FN be the elements the confusion matrix of a model. Let $\Omega = P \cup N$ and $\hat{\Omega} = \hat{P} \cup \hat{N}$. Then we define the following performance metrics:

- *Global Precision: how a model is good at "guessing" in the total sense*

$$\text{Precision}(\mathcal{M}) = \frac{|TP| + |TN|}{|\hat{\Omega}|}$$
$$\overline{\text{Precision}}(\mathcal{M}) = \frac{|FP| + |FN|}{|\hat{\Omega}|}$$

- *Positive or negative precision: how good a model is guessing a certain aspect*

$$\text{Precision}_P(\mathcal{M}) = \frac{|TP|}{|\hat{P}|}$$
$$\text{Precision}_N(\mathcal{M}) = \frac{|TN|}{|\hat{N}|}$$

- *Positive or negative recall: how good a model is able to encapsulate a certain aspect*

$$\text{Recall}_P(\mathcal{M}) = \frac{|TP|}{|P|}$$
$$\text{Recall}_N(\mathcal{M}) = \frac{|TN|}{|N|}$$
$$\text{Recall}(\mathcal{M}) = \frac{|TP| + |TN|}{|\Omega|}$$

#Definizione

Definizione (F_β-score).

Let β be a factor such that *recall* is considered β times important as *precision*. We define the F_β-score as the following metric:

$$F_\beta(\mathcal{M}) = (1 + \beta^2)\frac{\text{Precision}(\mathcal{M}) \cdot \text{Recall}(\mathcal{M})}{(\beta^2 \cdot \text{Precision}(\mathcal{M})) + \text{Recall}(\mathcal{M})}$$

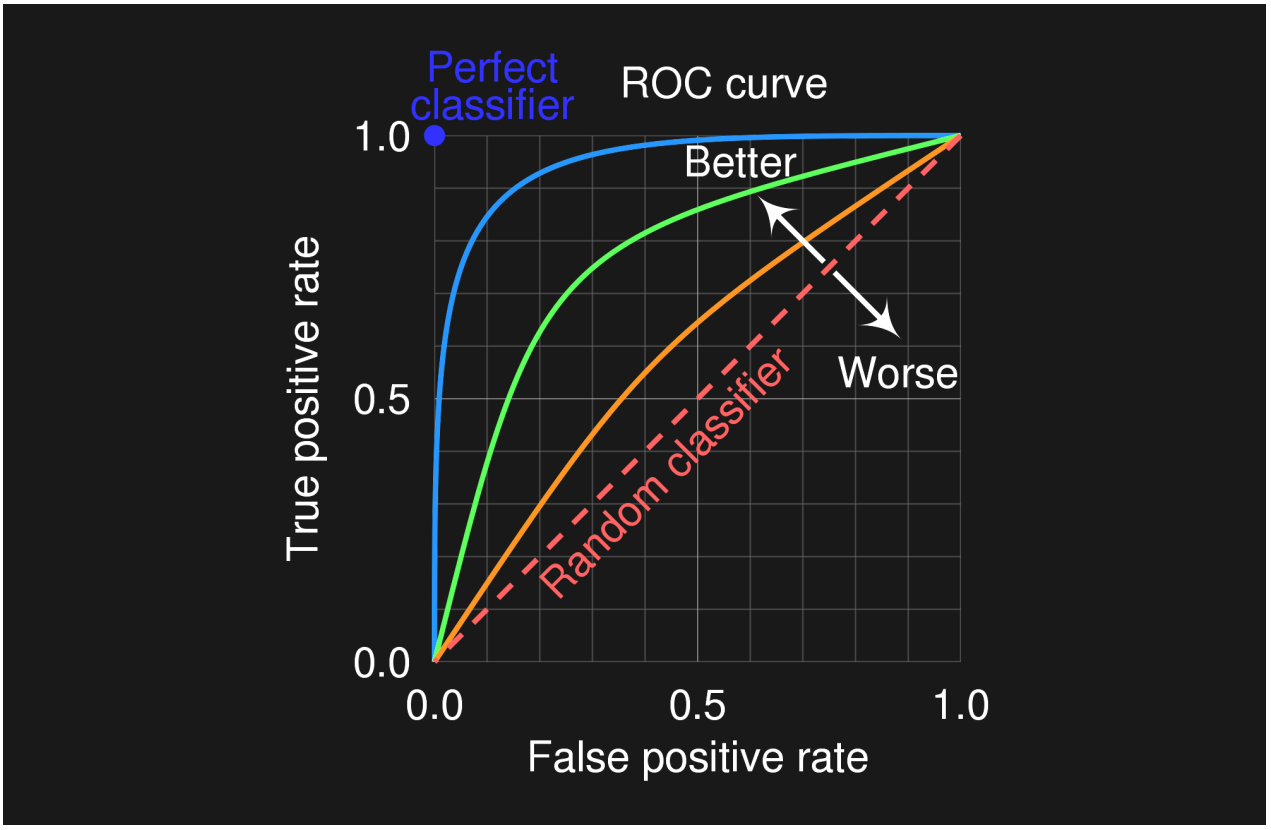
Let us see another approach, which is the *ROC curve*.

#Definizione

Definizione (ROC curve).

Let us define the *ROC curve* as the function where we *compare* the *proportion of false positives* with *positive recall*.

$$ROC : \frac{|FP|}{|N|} \mapsto \text{Recall}_P$$



To use the *ROC curve* metric, it is necessary to maximize its are under the curve:

$$\int_{[0,1]} ROC \to 1$$

X

Model Selection

X

Model Selection: possible criterions for model selection. Model selection approaches: Hold-out method, iterated hold-out method, k-fold cross validation, leave-one-out cross validation.

X

0. Voci correlate

- Model Evaluation
- Methodology in Data Science

1. Possible Criterions for Selection

Q. How do I select a model to deploy in our machine learning project?
A. It depends on the context! We have a lot of factors which can be used to evaluate a model, which goes *beyond* our traditional performance metrics; we are talking in the business sense.

Let us enumerate some of these possible factors.

1. **Performance.** This is the *"simplest factor"*, as we're using our traditional performance metrics (Model Evaluation).
2. **Interpretability.** In some cases, it might be important to understand *why* does a model take certain decisions. Sometimes, this might be even more important than model performance. This is a recurring theme today, as our models became harder to interpret (e.g. ChatGPT).
3. **Complexity.** A complex model is able to extract *interesting insights*; however it is usually tied to a harder interpretability and a higher cost.
4. **Dataset Size.** Certain models require a *large dataset*, whereas others work better with *smaller datasets* (counterexample: KNN).
5. **Dimensionality.** More dimensions means more information; however this leads us to the so-called *curse of dimensionality* (#TODO), which requires us to implement *dimensionality reduction algorithms*.
6. **Training Time and Cost.** In certain cases, having a lower *training time and cost* is more important than *performance* (or viceversa). Examples: marketing or hospitals.
7. **Inference Time.** In *time-critical systems*, it is mandatory to have an extremely low *amount of time needed to make a prediction*. Usually inversely proportional to training time:

$$t_{\text{TRAIN}} \propto \frac{1}{t_{\text{INFER}}}$$

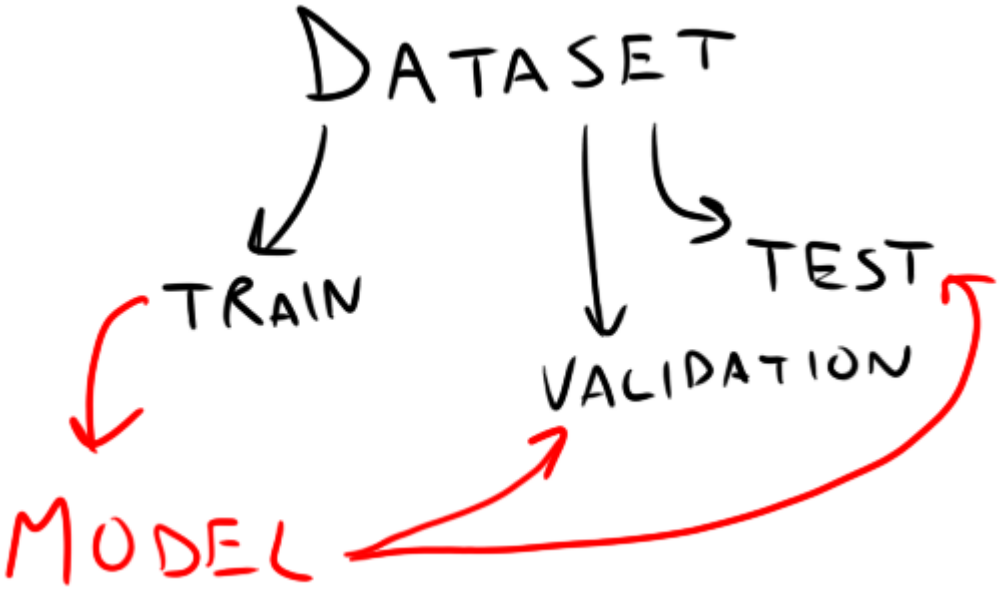
X

---

2. Model Selection Approaches

Q. I have selected a model in accordance to the previous criterions. Do I have to look out for anything else?
A. Yes, we need to avoid *overfitting* or *underfitting* (Problems in Machine Learning). To do it, we will use the following *model selection methodologies*.

The simplistic idea is to split our data into two parts, *train* and *test*. The *train* part will be used to develop our model, and the same model will be evaluated on *test* data. Our ideal point is when the *performance metric for the test and train data* are more or less the same (and has a good value).



2.1. Hold-Out Method

The main idea is to *simply* split our data into *two* (or even *three*) parts, following a certain proportion. The rule of thumb is to reserve 70% of our dataset for *training*, and the rest for *testing* (*or validation as well*).

OBSERVATION. For unbalanced datasets, we use *stratified sampling*, where the split data follows the same proportion of the whole dataset.

USE CASE. Large datasets
PROBLEM. Provides *only one single estimate*

2.2. Iterated Hold-Out Method

The problem in the previous method is that we *only have a single estimate*; if we want to be more *sure* that our estimate wasn't just a *"lucky case"*, we can just simply repeat the process.
So idea is the same as before, but we shuffle the data at each iteration.

ADVANTAGES. This also provides us enough data to calculate the *variance* of the estimates, which can give us good information on the *stability* of the model.
PROBLEM. With random shuffling, we might have overlaps between data.

2.3. k-fold Cross Validation

IDEA. Let $k \in \mathbb{N}$. Split our dataset into k equal parts; do k iterations, for in each iteration i we select the i-th part to be used for *validation*, and rest is used for *training*. This provides us multiple estimates.

COMMON PIPELINE. Usually $k = 10$.

2..4 Leave-One-Out Cross Validation

Same as k-CV but with k being the whole datasize. Very computationally expensive, but gives us the best estimate as it uses all data.

X

Feature Selection

X

Feature selection for Machine Learning: motivations. Main categories of feature selection: filters, wrappers and embedded methods.

X

---

0. Voci correlate

- Problems in Machine Learning

1. Introduction to Feature Selection

Feature selection is the process of *selecting specific features* of our *training data*, with the end of using only them for training our model.

Q. Why?
A. For various reasons, such as:

- Reducing complexity and dimensionality
- Removing irrelevant information for our predictive model

- To balance "importance" between features
- Avoid overfitting as models tend to overfit with more features

Now we will see specific (theoretical) techniques to do *feature selection*. In particular, we will have the three main methods:

- Filter: use statistical criterions to *"filter out"* variables
- Wrapper: use our models to select features, looking at performances
- Embedded methods: same as before, but looking at the model parameters (coefficients)

X

---

2. Filter Methods

IDEA. Use statistical criterions to exclude features

1. Remove *constant* or *quasi-constant* information, i.e. with variance almost 0 (or proportion between categories in the qualitative case)
2. Remove duplicate variables
3. Determine *correlation between variables* with *statistical tests*
4. Use statistical methods to test independence, such as the χ^2-score or the ANOVA.

Let us see detailed correlation tests:
PEARSON CORRELATION. Let X, Y be continuous variables. Let r_{XY} be defined as the *Pearson correlation*, where

$$r_{XY} := \frac{\sum_n [(x_n - \bar{x})(y_n - \bar{y})]}{\sqrt{(x-\bar{x})^2}\sqrt{(y-\bar{y})^2}}$$

The domain of this two-valued function is $[-1, 1]$. In particular, if:

- $-.7 \leq r_{XY} \leq .7$: The correlation is not strong enough to justify feature selection
- Otherwise: the correlation is strong enough to justify feature selection
- $r_{XY} = 0$: there is no linear correlation

SPEARMAN CORRELATION. Let X, Y be continuous or ordinal variables. Let d_n^2 be defined as the *difference between two rankings*. Then we have ρ the *Spearman correlation* defined as

$$\rho := 1 - \frac{6 \sum_n d_i^2}{n(n^2-1)}$$

The domain of this is $[-1, 1]$. If $\rho = 0$, it can be concluded that there is no correlation between X, Y at all.

X

---

3. Wrapper Methods

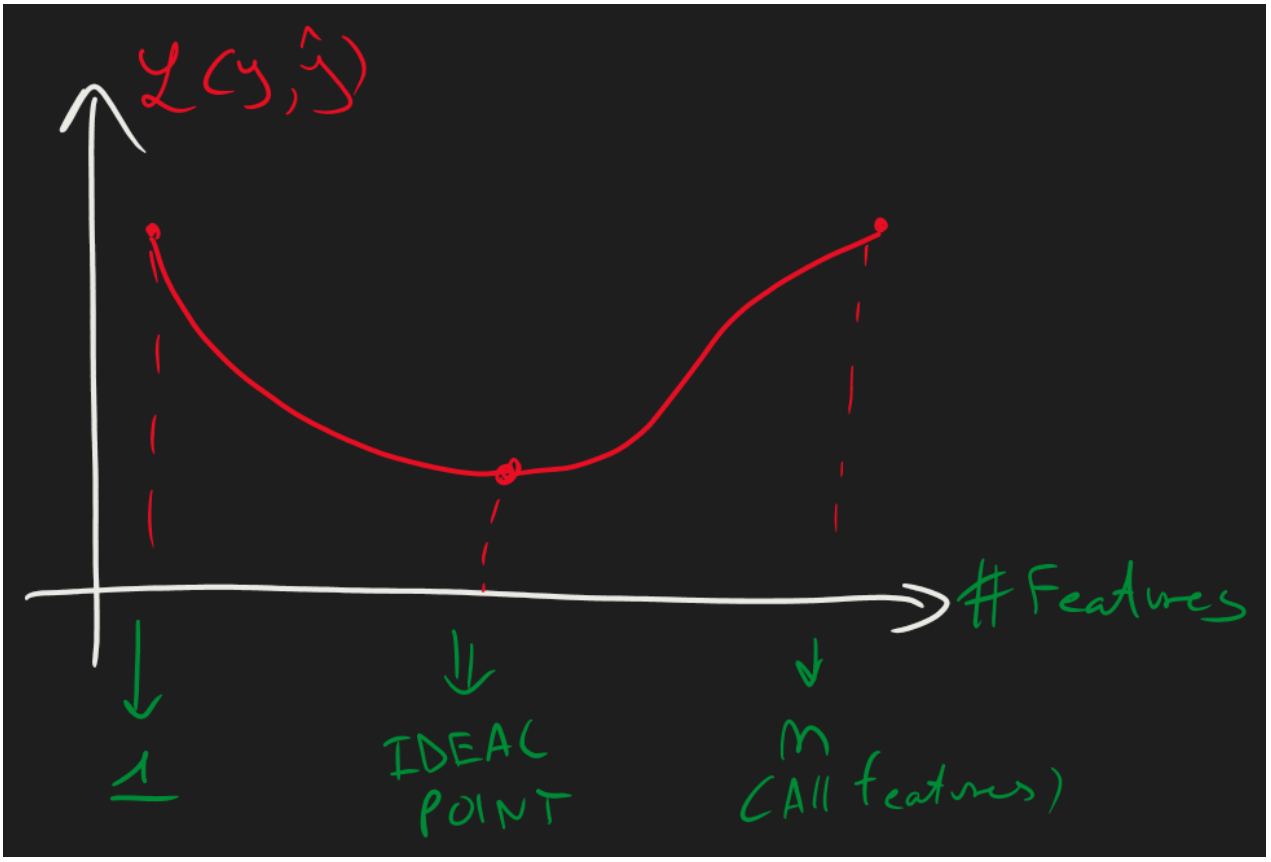
mmm wrapper 😜

OBSERVATION. (*Filter methods' limitations*) Please notice the following limitation with filter methods:

- I can use most of these methods *only* for *pairs of variables*; we might have a three-way correlation or more...

- We are only using *statistical methods*, maybe we can make good use of *predictive models*.

IDEA. Let Ω be all the explanatory variables possible. Then the idea is to grab a subset $\omega \subset \Omega$, build a model with ω and obtain its performance. Then iterate this process until you get a result (ω) that you consider satisfactory (either with minimum loss value or with best train/test score).



Let us see specific methods:

- **EXHAUSTIVE.** A *naive* method to do this is consider $\forall \omega \in 2^{\Omega}$ (i.e. every subset possible). However, as the notation suggests it, it takes $O(2^n)$ time complexity to do this, where n is the amount of time needed to train a model. It can get expensive!
- **FORWARD.** Begin by selecting *only one feature* $\omega_i \in \Omega$ and build a model on it. Then add another feature, build another model. Compare the performances: if it is better, keep the feature; otherwise discard this. Repeat this process until you have used all features.
- **BACKWARD.** Same as before, but inverse: we start from Ω and proceed to *"deconstruct"* it.
- **STEPWISE.** Same as the above of two, but we can consider both *removing* and *adding* features. Most computationally expensive in respect to forward and backward.

X

4. Embedded Methods

Q. Can we take even more advantages in using *predictive models*?
A. Yes, we can use the *embedded methods*. This might even be better, as it takes less time.

IDEA. Use the *model itself* to evaluate features. This can happen in many forms, such as:

- Add a *new component* for our loss function, which represents some sort of *"overfit penalty"*; this can take even more forms such as:
  - L^1, L^2 regularization (or known as Lasso, Ridge regression).

- Certain algorithms allow us to extract *features' importance*
  - Examples: *Decision trees* or *Random Forests*

X

Statistical Classifiers

X

Statistical classifiers: definition of statistical classifier. Foundation: Bayes' Theorem. Model: Full Bayes classifier. Terminology: posterior, likelihood, prior probability and evidence. Model: Naïve Bayes Classifier. Laplace estimation. Using Bayesian classifiers with numeric attributes. Advantages and disadvantages.

X

---

0. Voci correlate

- Probabilità Condizionale
- Supervised and Unsupervised Learning

1. Foundations of Bayesian Learning

DEFINITION. A *statistical classifier* is a *model* which is able to *perform probabilistic prediction*, i.e. predict class membership probabilities.

In particular, they will use the *Bayes' theorem* to calculate the probabilities. So, let us recall the following theorem:

> Teorema (Bayes).
>
> Let A, B events of the space (Ω, Σ, p). Assuming that A, B are dependent and denoting $p(A|B)$ and $p(B|A)$ as the *conditional probabilities*, we have the following equation:
>
> $$p(A|B) = \frac{p(A)p(B|A)}{p(B)}$$

PROOF of Teorema 1 (Bayes)
Since that A, B are not disjunct we have that

$$P(A \cap B) = p(A)p(B) \neq 0 \iff p(A)p(B|A) = p(B)p(A|B)$$

dividing either by $p(A)$ or $p(B)$ we have the theorem. ■

Q. How is this applicable in our context?
A. We can extend the theorem by considering B as a sequence of events $(B_n)_n$, so we can represent more features. So we can consider B as a *"vector"*.

X

---

2. Terminology

Let us *represent* the *Bayes'* theorem in our context, using different terminologies.

$$p(h|\mathbf{X}) = \frac{p(\mathbf{X}|h)p(h)}{p(\mathbf{X})}$$

Where:

- h is the *class membership*
  - $p(h)$ is the *prior probability*, i.e. the *"initial probability"* which does not take data into account
- $\mathbf{X}$ is the *"evidence"*, that is the *dataset*
  - $p(\mathbf{X})$ is *marginal probability*, i.e. the probability of having $\mathbf{X}$ happened
- Combining them, we have:
  - $p(h|\mathbf{X})$ the *posterior probability*, i.e. the probability that our hypothesis holds for the given evidence
  - $p(\mathbf{X}|h)$ is the *likelihood*, which is the *"inverse"* of the posterior probability; i.e. the probability that a certain evidence happens with our hypothesis

X

3. Full-Bayes Classifiers

MODEL. Let D be our training set of tuples and $\mathbf{X} \in D$. A classifier is said to be *full-Bayes* if it classifies the class membership by maximizing the *posterior probability*:

$$h_i = \arg\max_{h \in H} p(h|\mathbf{X}) = \arg\max_{h \in H} \left[\frac{p(\mathbf{X}|h)p(h)}{p(\mathbf{X})} \right]$$

OBSERVATION. As $p(\mathbf{X})$ is constant (and usually unknown!), we can just *"remove it*"; so our predicted class can also be written as

$$h_i = \arg\max_{h \in H} \left[p(\mathbf{X}|h)p(h) \right]$$

PROBLEMS. Note that as our input $\mathbf{X}$ has a higher dimension this becomes more complicated: in fact, to calculate $p(\mathbf{X}|h)$ we would have to know the following values:

$$p(\cap_n x_n|h)$$

Since that we're intersections, more features might imply a small number of datapoints selected, which makes our predictions slightly biased towards events with most occurrences. In fact, if $p(\cap_n x_n) = 0$, we cannot have any prediction at all.

X

4. Naïve-Bayes Classifier

Q. How do I resolve the previous problems?
A. We will mainly make two *"naïve"* assumptions.

1. (*Optional*) Let us assume that every prior probability is equal to each other; i.e. we have equiprobability between classes

$$p(C_1) = \ldots = p(C_n) \implies p(C_i) = \frac{1}{|C|}$$

2. Let us assume *independency* between our *attributes* (features): therefore we can have the following equation for calculating the *posterior probability*:

$$p(\mathbf{X}|C_i) = \prod_n p(x_k|C_i)$$

This simplifies our calculations, going from having to consider intersections to just considering products of singular probabilities.

MODEL. The same as *Full-Bayes classifiers*, but assuming independence between posterior probabilities.

$$h_i = \arg\max_{h \in H} \left[\prod_n p(x_n) p(h) \right] \approx \arg\max_{h \in H} \left[\prod_n p(x_n) \right]$$

Moreover, we can implement more measures to make it more compatible with our dataset:

- *Laplace estimator*: If we have events with 0 frequency (i.e. $p(x_n) = 0 \implies p(x_n|c_i) = 0 \overset{0 \cdot \prod = 0}{\implies} p(h_i|\mathbf{X}) = 0$), we can use the *Laplace estimator* to avoid unfairly biasing them:
  - $\forall n, p(x_n) = 0 \rightsquigarrow p(x_n) = \frac{1}{|X_n|}$ (in other words we add 1 to the count for every attribute class-value combination)
- *Missing values*: Missing values will not be included for our frequency count
  - They can be also be treated as a class of their own
- *Gaussian Kernel*: The probability of a numerical attribute can be calculated with the *probability density function* of the Gaussian kernel, so we estimate in the following way: $p\{X_n = x\} \approx p\{X_n \leq x\} = \mathcal{N}_{\mu,\sigma^2}(x)$
  - Or we can bin them and treat them as categories as well

ADVANTAGES.

- Easy to implement, as we only use maths here
- In most cases the results are *"more than okay"*
- Works well for *text mining*, as we convert the probabilities in a vector representation (*Bag of words*)

DISADVANTAGES.

- We're using very bold assumptions; for example, we're assuming that the variables are independent, when they could very well be dependent
  - We have a *loss of accuracy*; however, it might not matter, as the approximation is negligible enough to make a significant change in the decision
  - To resolve this problem, we can use *Bayesian Belief Networks*, where we combine both *Naive classification* and *Full classification*.

X

Instance-Based Learning

X

Instance based learning. Definition of lazy learning, difference from other algorithms. Models of instance-based learning: Rote learner, KNN classificator and regressor. Observation: choosing hyperparameter and optimizing calculations with k-D trees.

X

0. Voci correlate

- Supervised and Unsupervised Learning
- Definizione di Spazio Metrico
- Models in Machine Learning

1. Definition of Instance-Based Learning

Instance-based learning is when we teach our *models* via the *instances* themselves; by *instance* we mean a row of a dataset.
Its new characteristic is that this type of learning focuses more on *finding similarities* between *other cases*: therefore, having more variables leads to *less accurate predictions*, as a consequence of the *curse of dimensionality*.

Sometimes it is also referred as **lazy learning**, since that it's not *learning* in the sense of *getting parameters from the data*; its rather looking at the *data itself* to search for conclusions. In fact, we can also say that this kind of model is *non-parametric*.

X

2. Rote Learner

ROTE LEARNER. The simplest way to implement an *instance-based model* is to just simply check if our *unseen data* matches exactly to one of the *available data*; if there is, return its exact target as the prediction. If not, then nothing can be decided.

- This model is a *conceptual idea* rather than an actual model.

X

3. Preliminary Mathematics on Metric Spaces

Before looking at KNN, let us see some preliminary mathematics on *metric spaces*, so we can get a clear meaning of *"distance"*.

#Definizione

> Definizione (metric space).
>
> X is said to be a *metric space* if there exists an operator $d : X \times X \longrightarrow \mathbb{R}$ such that d can be defined as a *distance* (see the mathematical definition somewhere else).

Let us consider $\mathbb{R}^n$ as a metric space for our purposes; we can define the *Minkowski distance* as following:
#Definizione

> Definizione (Minkowski distance).
>
> Let $0 < p \leq +\infty$. We can define the *Minkowksi distance of order* p in $\mathbb{R}^n$ as the following function:

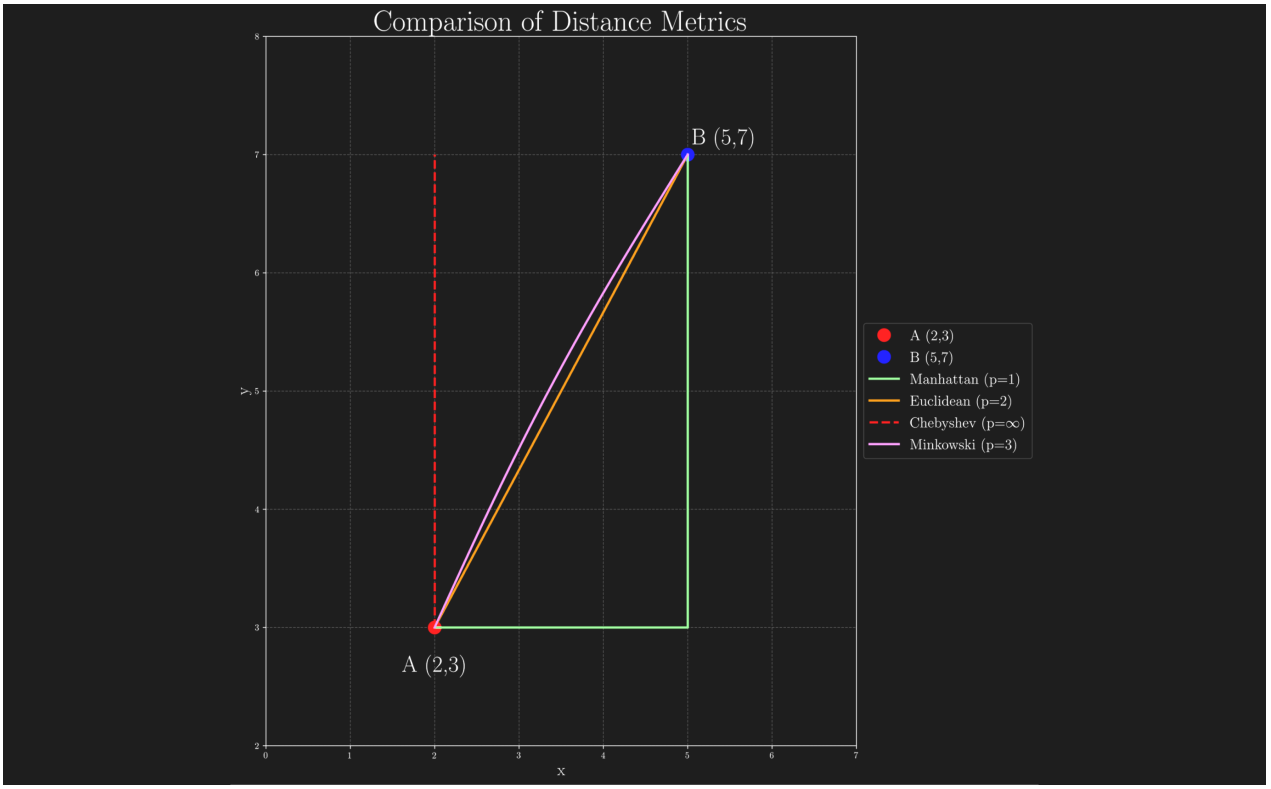
$$d_p(\mathbf{x}, \mathbf{y}) := \left(\sum_n |x_n - y_n|^p \right)^{\frac{1}{p}}$$

In particular, for $p = +\infty$ it is defined as the following:

$$d_{+\infty}(\mathbf{x}, \mathbf{y}) := \sup_n |x_n - y_n|$$

By setting p to certain values, we can define the following:

- For $p = 1$ we the *Manhattan* or *Taxicab* distance
- For $p = 2$ we have the *euclidean* distance
- For $p = +\infty$ we have the *Čebyšëv* (Чебышёв) distance



#Osservazione

> Osservazione (why is Čebyšëv defined as the $p = +\infty$ of Minkowski distance?).

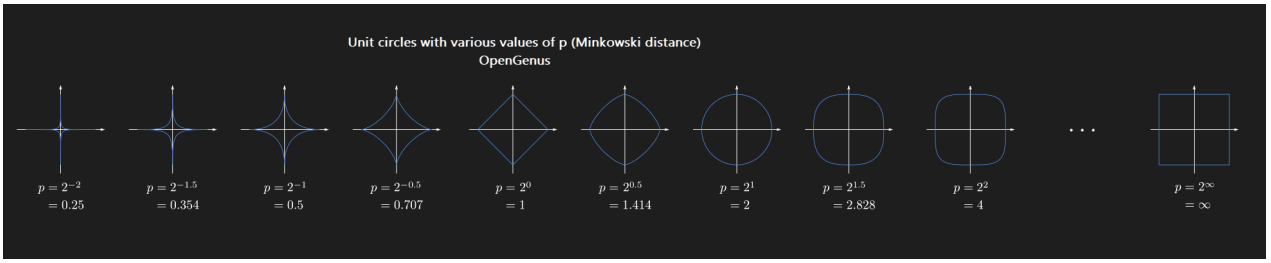
We have that for $p = +\infty$ its Minkowski distance is defined as

$$d_{+\infty}(\mathbf{x}, \mathbf{y}) := \sup_n |x_n - y_n|$$

because we have the limit

$$\lim_{p \to +\infty} \left(\sum_n |x_n - y_n|^p \right)^{\frac{1}{p}} = \sup_n |x_n - y_n|$$

We can prove it graphically with the following visualization, which gives us every *point that is distant from the origin with measure* 1 *in every distance*:



PROOF.
If you want a mathematical proof, you can do it by considering $\mathbf{y} = \underline{0}$. In fact we would have

$$\lim_{p \to +\infty} d_p(\mathbf{x}, \underline{0}) = \lim_{p \to +\infty} \left(\sum_n |x_n|^p \right)^{\frac{1}{p}}$$

Now let us define $M := \sup_n |x_n|$. Therefore we have the following inequality:

$$M^p \leq \sum_n |x_n|^p \leq nM^p$$

The lower bound is due to the fact that $\exists k : |x_k| = M$ and the upper bound is just simply due to the fact that $\sup_n |x_n| \geq |x_k|, \forall k$. Taking the p-th root, we have the following inequality:

$$M \leq \left(\sum_n |x_n|^p \right)^{\frac{1}{p}} \leq n^{\frac{1}{p}} M$$

Remember the follow identity:

$$\forall k \in \mathbb{N}^*, \lim_n \sqrt[n]{k} = 1$$

So we effectively have the limit

$$\lim_{p \to +\infty} M \leq \lim_{p \to +\infty} \left(\sum_n |x_n|^p \right)^{\frac{1}{p}} \leq \lim_{p \to +\infty} M$$

Therefore by the *squeeze theorem* we have proved the limit. ■

X

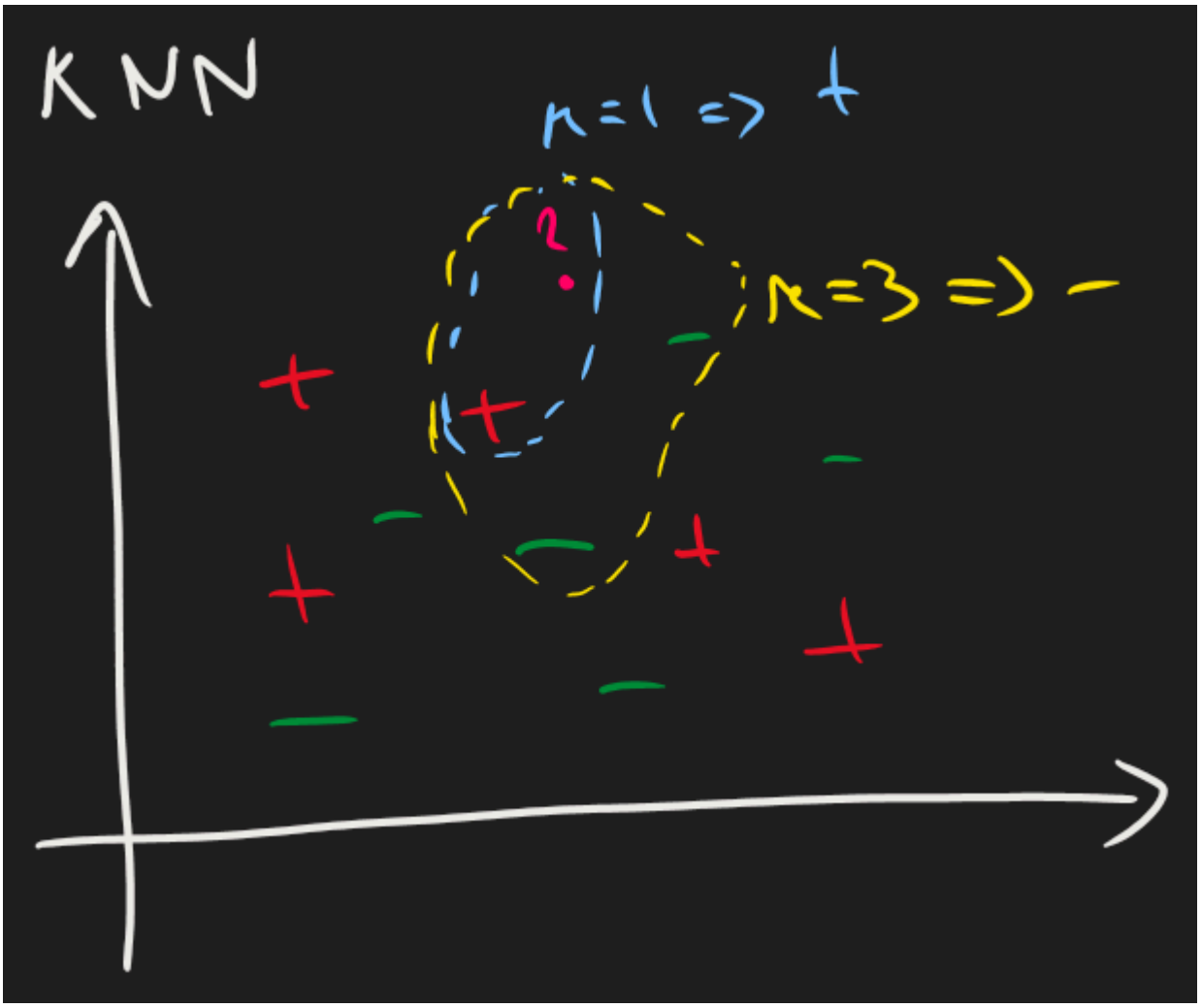
4. k-Nearest Neighbours

"Diga-me com quem andas, que te direi quem és."

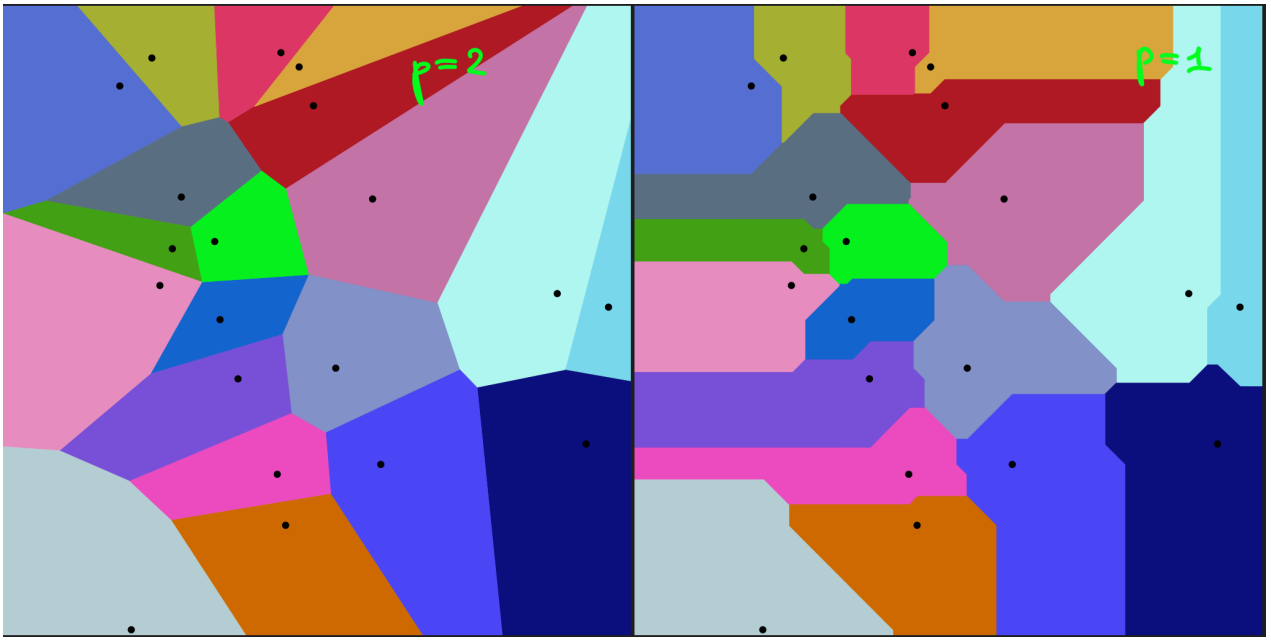
- Translation: *"Tell me who you walk with, and I'll tell you who you are."*

A **KNN** (k-nearest neighbours) is a *non-parametric model* where the predicted target is based on its k-nereast neighbours of the unseen instance. So we calculate *every distance* to each point in the dataset, and select the k nearest points.

- In the *classification case*, we can choose the class by some sort of *"majority vote"*; it can be by simple majority (> 0.5), or by weighted majority (the closest neighbours' *"votes"* are *"more important"*)
- In the *regression case*, we can choose to the number as a *"representative"* of the values for its neighbours; it can be the mean, median, ...



VORONOI DIAGRAMS. To understand better the behavior of *KNN* when we choose different parameters k , we can use *Voronoi diagrams*. In fact, if we set $k = 1$ we will have *"straight lines"* (depends on the geometry!) dividing our dataspace. These straight lines are defined by the current dataset.

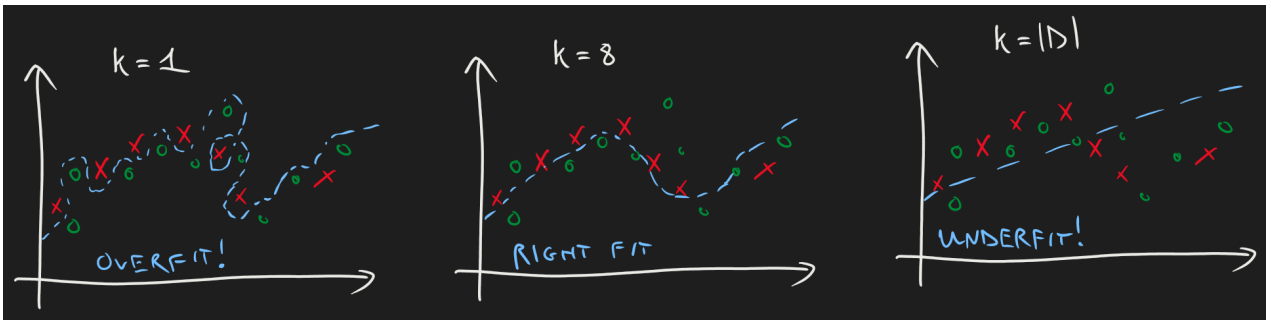


Our goal is to make these lines *"as smooth as possible"*, to avoid *overfitting*.

OBSERVATION. Does our selection in p change anything, aside from the *"geometry"*? In our terms, choosing a higher p means giving *less power to outliers* as we emphasize the *nearest neighbors* more.

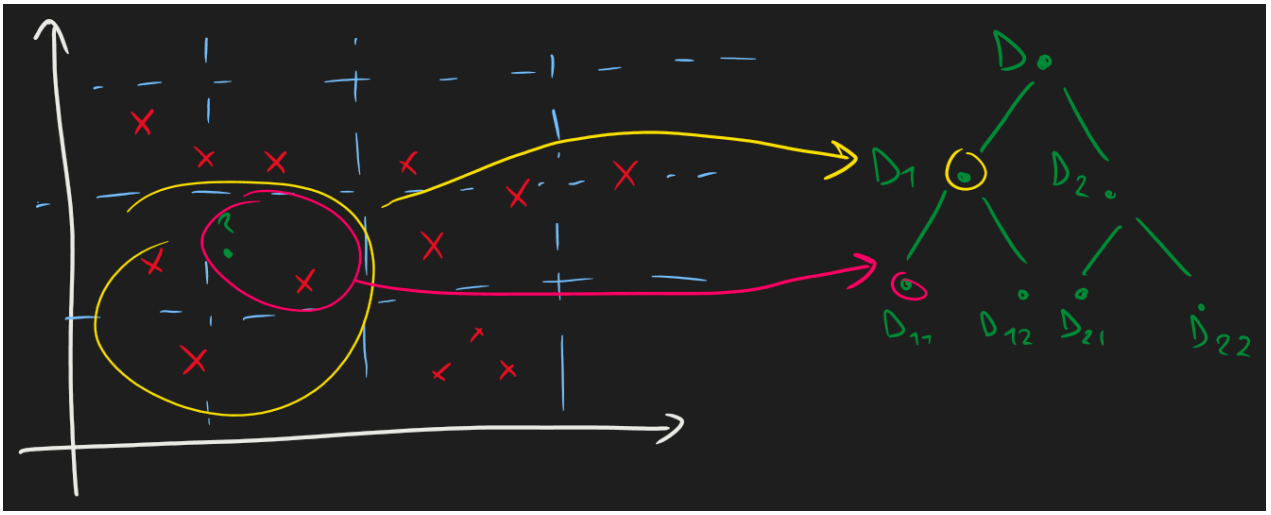
SELECTION OF NUMBER OF NEAREST NEIGHBORS. As introduced above, our selection in k can change our model quite a lot.

- For small k, our model is *sensitive to outliers* and tends to *overfit* a lot: this is because it does not have a good view on the whole dataset, and looks blindly to its nearest neighbor. In fact, the model will follow the *Voronoi diagram*.
- For bigger k, our model improves but it may *underfit* as well, since that it doesn't learn the patterns around their neighbours
  So, to select the right k, we might have to do *trial and error* processes.



OPTIMIZING CALCULATIONS. With *large datasets*, calculating distances to every point is computationally expensive: it is $O(n)$. We can use *tree-based data structures*, to *partition our dataset*, allowing the model compute the distances with only a small subset of points. In this way, we might potentially get a lower bound of $O(\log n)$.

- K-D trees are commonly used for this
- We can use *case-based reasoning*, where we select subsets by *category membership*.



X

Decision Tree Models

X

Decision trees. Definition of decision trees, main characteristic: interpretability. Main idea of decision trees: dividing dataspaces with horizontal or vertical lines. Terminology for tree data structures: root, internal, leaf nodes, branches and pure leaves. Extracting rules from trees. Advantages and disadvantages.

X

0. Voci correlate

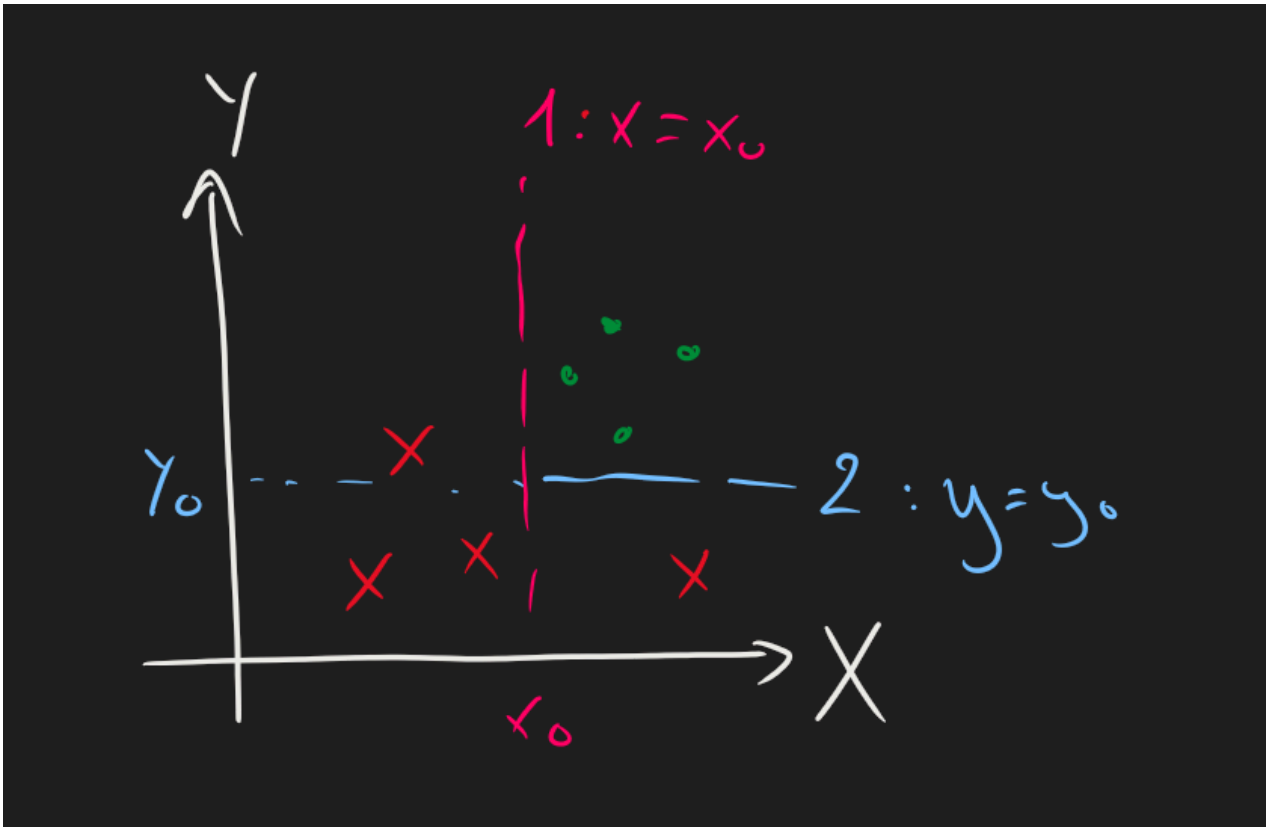
- Supervised and Unsupervised Learning
- Feature Selection

1. Introduction to Decision Trees

DECISION TREES is a *family of non-parametric supervised learning algorithms* used both for *classification* and *regression* problems. The main idea is to introduce some *"rules"* in our variables to make choices; they can be recursively repeated, achieving a certain depth. For now, we'll see decision trees for classification problems.

KEY ADVANTAGE. The key advantage of *decision trees* is *interpretability*: in some problems we might be more interested in *understanding the results* rather than *achieving high performance*. In fact, decision trees can be used as an *embedded method* for *feature selection*.

MAIN IDEA. Assume that X, Y are our *numerical explanatory variables* and T is our categorical target variable. We can represent X, Y through a *scatterplot* and split classes *independently* with lines that are parallel to our axes. If desirable, we can repeat this process until we have perfectly split the classes (even though sometimes it is undesirable).

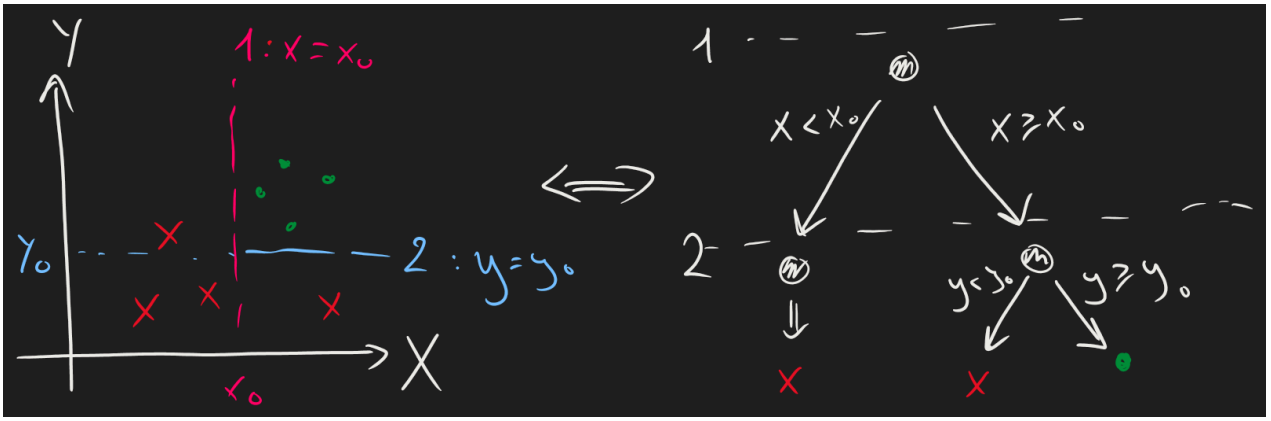


X

2. Tree Structure

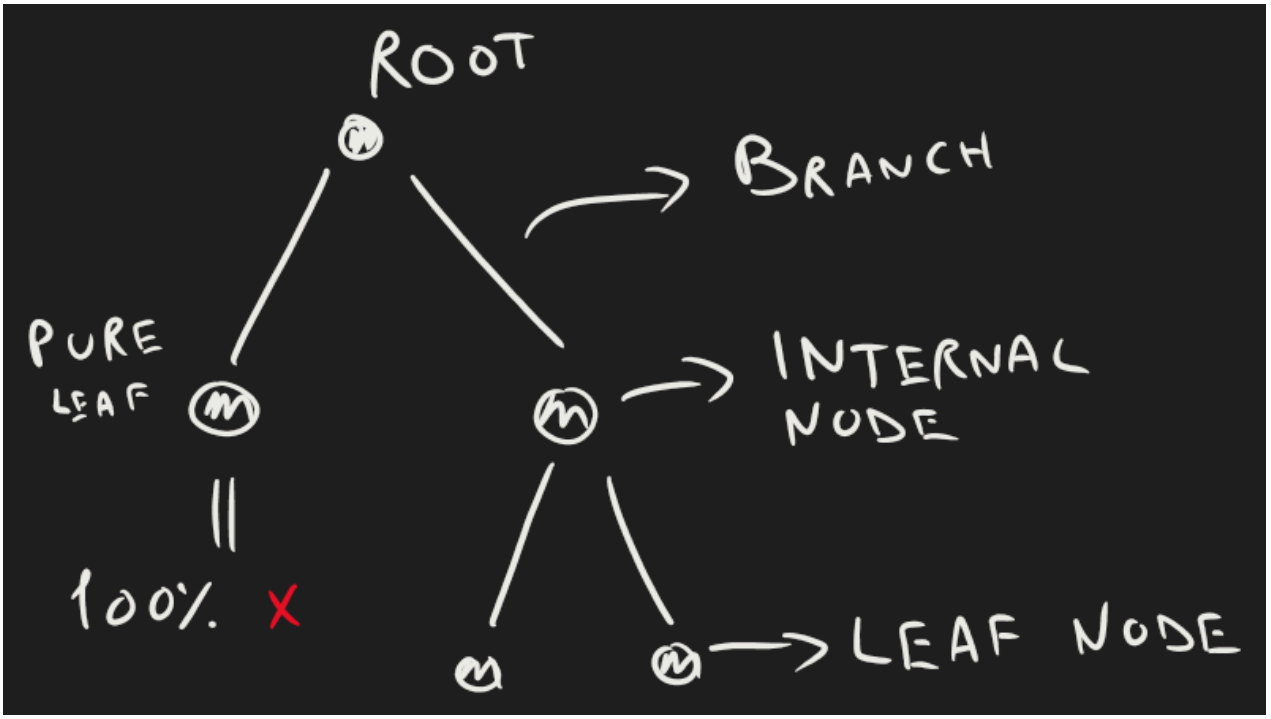
TREE DATA STRUCTURE. The reason this algorithm is called *"decision trees"* is because this process can be represented with some sort of *"if-else"* statements, which can be graphically represented with *tree data structures*. To compute a decision, we just run the tree starting from the *"top"*, and then make a decision

based on our *"final class"* (it can be for instance by majority vote or by taking the mean of the members in the group).



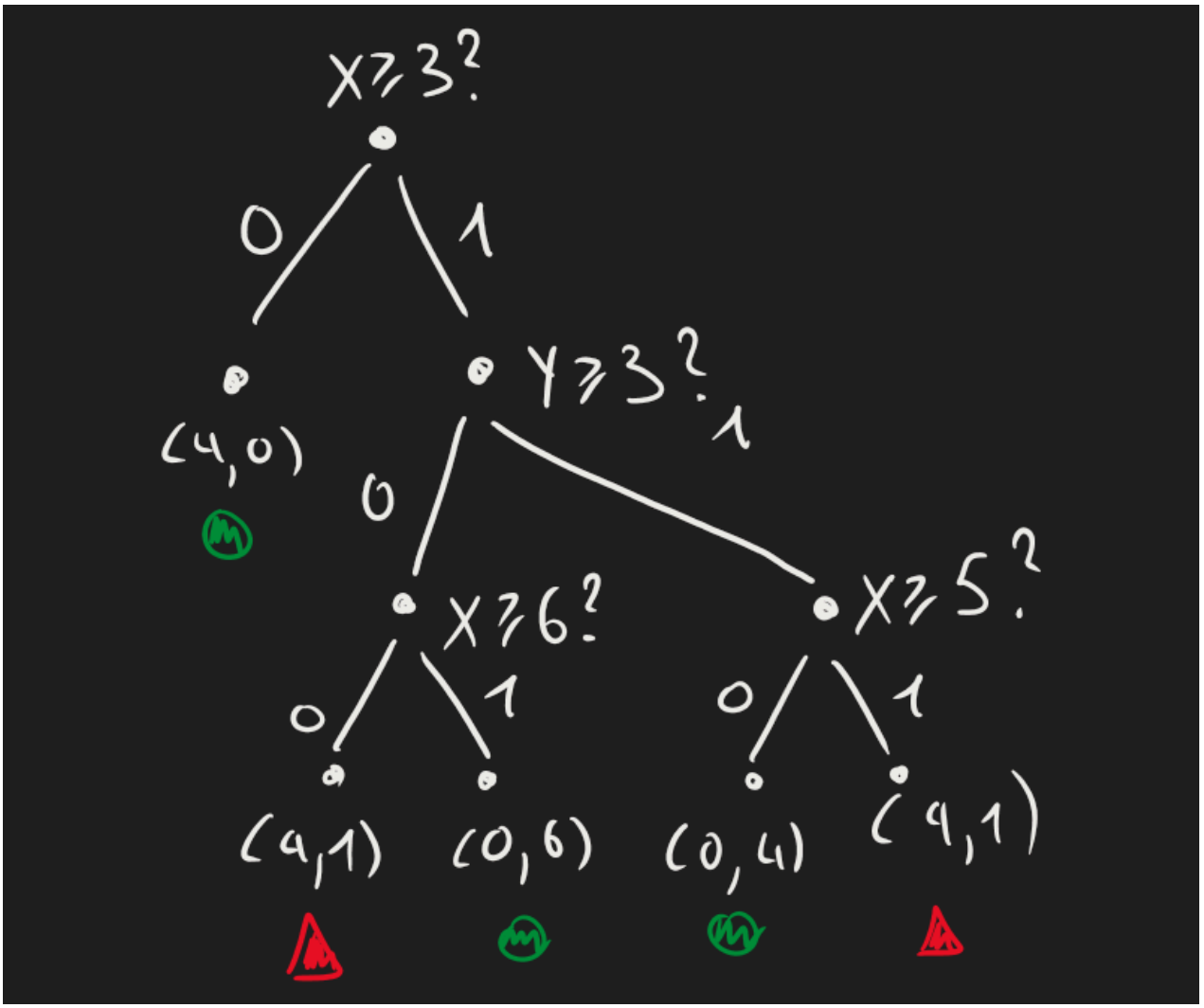
TERMINOLOGY. Let us introduce some terminology for *decision trees*.

- A *node* in a decision tree is a *"decision"* which can contain n instances of the dataset.
  - A node is said to be the *root* if it contains every observation in the dataset
- A *branch* of a *node* is the *"possible decisions"* which can be done for the node. A branch leads to other nodes.
  - If a node does not have *branches*, it is said to be a *leaf node*
    - If a leaf node contains *only one single class*, it is said to be a *"pure leaf"*.
  - An *internal node* is a node that is brought by a *branch* and not a *leaf node*



Q. Can we convert this in a logical language?
A. Yes, admitting that each *node* adds a conjunction $\wedge$ and a *new leaf* adds a disjunction $\vee$.

For instance if we have this tree:



Then a possible logical representation is

$$\begin{cases} \hat{y} = \mathrm{O} \iff \underbrace{(X < 3)}_{\mathrm{O}_1} \vee \underbrace{((X \geq 3) \wedge (Y < 3) \wedge (X < 6))}_{\mathrm{O}_2} \vee (\dots) \\ \qquad \iff \bigvee_n \mathrm{O}_n \\ \hat{y} = \Delta \iff \bigvee_n \Delta_n \end{cases}$$

X

3. Advantages and Disadvantages

ADVANTAGES.

- As said before, the main selling point of *decision trees* is its interpretability
- We have no problems with dealing *different types of data*: in certain cases, we can even not deal with missing values
- Insensitive to scale factors for numerical variables
- Can be adapted to regression problems
- Data cleaning is (mostly) unnecessary

DISADVANTAGES.

- Most of the algorithms require the problem to be of *classification*

- Small variations in our dataset can change big differences in our model, as it is *non-parametric*
- We can have recursive trees which are replicated several times
- Worse results when we're dealing with more classes
- We only have *perpendicular boundaries to the axis* in our decisions; this problem will become the basis for *Support Vector Machines* (SVM).

X

Decision Tree Induction

X

Inducing (creating) decision trees. Problems in decision tree induction. Basic algorithm for tree induction (greedy algorithm). Specific induction algorithms for classification trees: DDT (Discriminant power), ID3 AND C4.5 (Entropy), CART (Gini Index).

X

---

0. Voci correlate

- Decision Tree Models
- Definizione di Entropia

1. Problems in Decision Tree Induction

Having seen what a *decision tree* is, we also need to know *how to build one*. This is mainly because when building one we have the following *"problems"*:

- Where should we *"cut"* the variable range for the branches?
- How *many* branches should we make?
- In which layer do I stop *branching*
  These questions are fundamental to avoid *overfitting*.

X

---

1. Basic Algorithm

Let us define a *basic algorithm*, before seeing the *"official"* ones.

ALGORITHM. (*Basic decision tree induction*)
Prior to this algorithm, we discretize every continuous variable.
At the root node we have every observation, and the observations are partitioned recursively based on the selected attributes by considering every variable, one by one.
The algorithm stops iff one of the three conditions happen:

- All observations for a given node belong to the same class; so we stop branching on a *pure leaf node*
- We have no more variables or samples left for further partitioning

OBSERVATION. This is a *greedy algorithm*, as it looks at *every possibilities* and understands the logic step by step; however, it won't be able to capture the *best possibility* ever. We can compare it with *playing chess*, where a player evaluates their choices basing only on their current situation.

PSEUDOCODE.

```
ALGORITHM BASICINDUCTOR(VARIABLES, TREE):
        IF |VARIABLES| = 0:
                RETURN TREE

        IF |TREE.ROOT_NODE| = 0 OR TREE.ROOT_NODE IS PURE:
                RETURN TREE

        ELSE:
                LET SUBTREES = BRANCH TREE.ROOT_NODE BY VARIABLES[0]
                FOR i=1,...,|SUBTREES|:
                        BASICINDUCTOR(VARIABLES[1:], SUBTREE[i])
```

Time complexity can be estimated to be $O(N \times n)$, with N being the number of instances and n the number of variables, as it evaluates *every possibility* with the data.

OSS. For the next algorithms, we will use a different approach for tree induction. In particular, we will define a *numerical criterion* which represents a variable's *"importance".

X

2. Divisive Decision Trees (DDT)

IDEA. Just like the previous algorithm, it is a *greed search* and we have *no backtracking*. To select *variables* for branching, we will use the *"discriminative power"*. The algorithm idea has this following iteration:

- Calculate every discriminant power of each explanatory variable V
- Select B as variable such that the discriminant power is the best one
- Branch the current node according to B
- Continue until we run out of variables, or instances or all leaves are pure.

#Definizione

> Definizione (discriminative power of a variable).
>
> To measure the *"discrimination metric"* of an attribute we can use the following definition.
> Let n be the number of instances in a dataset, let C_i be the maximum number of examples correctly classified (in proportion) by the i-th class of our variable V.
>
> Then the *discriminative power* of V is defined as follows:
>
> $$D_P(V) := \frac{1}{n} \sum_n C_n$$
>
> Basically, it indicates *"how pure"* will our node be when branching with this variable.

#Esempio

Esempio (calculation of a discriminative power).

Suppose that our dataset is structured in a such way that the pivot table of V, T_1, T_2 is as follows:

V	T1	T2
1	10	3
2	2	2

In this case, we would have $C_1 = 10$ and $C_2 = 2$. So $D_P(V) = \frac{10+2}{17} = \frac{12}{17}$.

PSEUDOCODE. Let `N` be a node, `ASET` the attribute set and `ISET` the instance set. Then the *divisive decision tree* is defined as the following recursive algorithm (pseudocode):

```
DDT(N, ASET, ISET):
        if ISET is EMPTY:
                the terminal node N is of unknown class
        else if ASET is EMPTY or ISET.isPure:
                the terminal node N has the name of the class
        else:
                powers = {}
                for A in ASET:
                        a_power[A] = evaluate(A)
                B = powers.argmax()
                for V in B:
                        C = new_node(N)
                        C <- (B,V)
                        JSET = ISET such that B is V
                        KSET = ASET.pop(B)
                        DDT(C, KSET, JSET)
```

OBSERVATIONS. Note that this method has some disadvantages, namely:

- It tends to overfit as our goal is to reach *tree purity*.
- It might take long to induct one decision tree
- We can only use categorical variables

X

3. Iterative Dichotomizers

Iterative Dichotomizer is a family of *induction tree algorithms* which involve using *information gain* as the *selection measure*.

#Definizione

Definizione (expected information or Shannon's entropy).

Let $\mathbf{x} = (x_1, \dots, x_n)$ be a n-uple with total sum N. For each singular projection x_i we define their probability as $p_i := \frac{x_i}{N}$. Then the *Shannon's Entropy* of $\mathbf{x}$ is defined as follows:

$$\text{Entropy}(\mathbf{x}) := -\sum_{i \leq n} p_i \log_2(p_i)$$

#Osservazione

Osservazione (some observations on Shannon's Entropy).

Let us make some observations on this measure.

- If for a given x_i we have $p_i = 0$, we can define $0 \log_2(0) := 0$ (this can be justified by taking the limit $\lim_{x \to 0} x \log x$)
- We have the minus sign as we want a *positive quantity*, but $\log_2(x < 1) < -1$.

3.1. Iterative Dichotomizer 3

Let us define the following nomenclature.

- Let D be the training set of class-labeled tuples
  - Let $\text{Entropy}(D) := \text{Entropy}(\mathbf{x})$, where $\mathbf{x}$ is the tuple of the classes for the target variable
- Let $C_{i,D}$ be the set of tuples of class C_i (of a specific variable) in D
  - Let $\text{Entropy}(C_{i,D})$ be defined analogously as above.
- Let $|\bullet|$ represent cardinality of any set or tuple.

#Definizione

Definizione (entropy of a variable).

Moreover, let us define the *entropy of a variable* A as the following:

$$\text{Entropy}_A(D) := \sum_{n \in A} \frac{|C_{n,D}|}{|D|} \text{Entropy}(C_{n,D})$$

Basically this tells us *"how much information is needed to classify D with A"*.

#Definizione

Definizione (information gain of a feature).

We define the *information gain* of a feature A in respect of the training data D as follows:

$$\text{Gain}_D(A) := \text{Entropy}(D) - \text{Entropy}_A(D)$$

IDEA. Same as *DDT*, but using $\text{Gain}(\bullet)$ as the selection measure. In particular, we want to *select the maximum* the *information gain* (or the minimum of the attribute entropy).

3.2. Iterative Dichotomizer C4.5

Q. How do I handle *continuous values*?
A. Either by binning the variable, or using the variant of *ID3*, called *C4.5*.

IDEA. Instead of binning the variable, we will determine the *"best split point"* for the variable. In particular it does the following:

- Sort the values of V
- Consider each midpoint between each pair of adjacent values as a possible *split point*
- Evaluate the *entropy* $\text{Entropy}_V(D)$ considering two partitions: one *before* the midpoint, the other *after* the midpoint.
  - Select the partition as the one which *minimizes* $\text{Entropy}_V(D)$.

X

4. CART

IDEA. Same as always, but to optimize calculations we will use the *Gini index* instead of entropy as a *selection measure*.

#Definizione

Definizione (Gini Measure).

If a dataset D contains instances from n classes, then the *Gini index* is defined as the following number:

$$\text{Gini}(D) := 1 - \sum_n r_f^2(D_n)$$

where D_j is the j-th class, and $r_f(\bullet)$ is the relative frequency.

Let V be a categorical variable which will branch our dataset. Then its Gini index is similarly defined as:

$$\text{Gini}_D(V) := \sum_n \frac{|D_{n,V}|}{|D_n|}\text{Gini}(D_{n,v})$$

#Definizione

Definizione (reduction in impurity).

Similar to *information gain*, we define the *reduction impurity* as the increment (or decrement) of the Gini index to a variable:

$$\Delta\text{Gini}_D(A) := \text{Gini}(D) - \text{Gini}_D(A)$$

In this case, the value to *"maximize"* is $\Delta\text{Gini}_D(\bullet)$.

X

Regression Trees

X

Regression Trees. Preliminary introduction, definition of regression tree. Intuitive approach for inducting regression trees. Main generalized outline for regression tree induction. Metrics for inducting RTs: MSE, MAE and F-MSE.

X

---

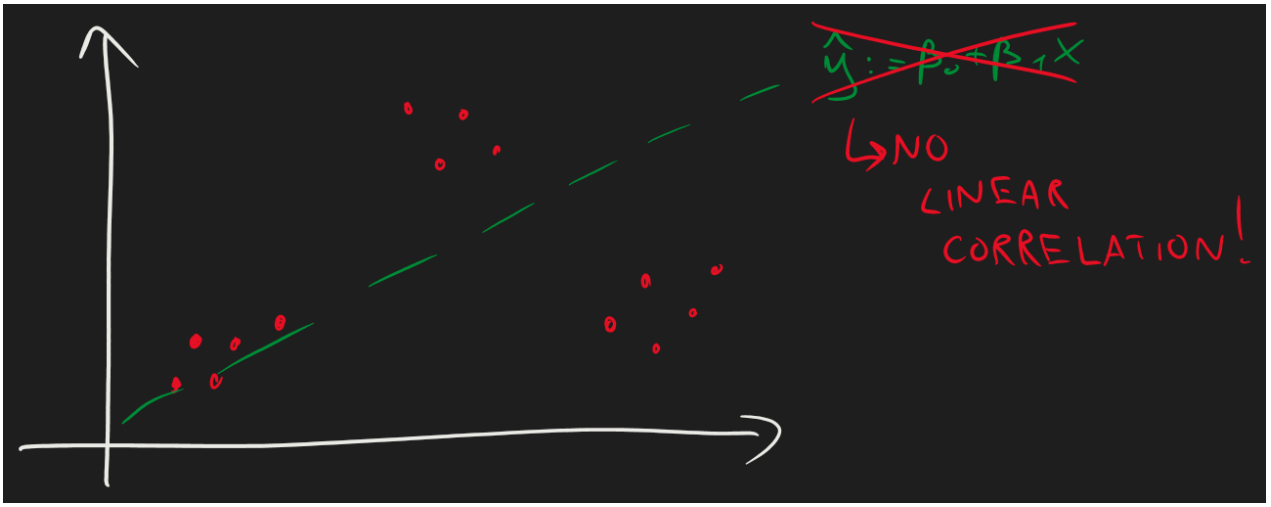
0. Voci correlate

- Decision Tree Models
- Linear Regression

1. Preliminary Introduction

Sometimes we can easily solve *regression problems* by fitting a simple *linear regression* onto the variables. However, this is under the assumption that these variables have a *linear relationship between them*.

This means there are certain problems which do not go undergo such assumptions, which may cause for *linear regression* to be not applicable (1).



So, in some datasets we should use *other methods* than the *linear regression*; one option is to use *regression trees*, which we will see in this page.

X

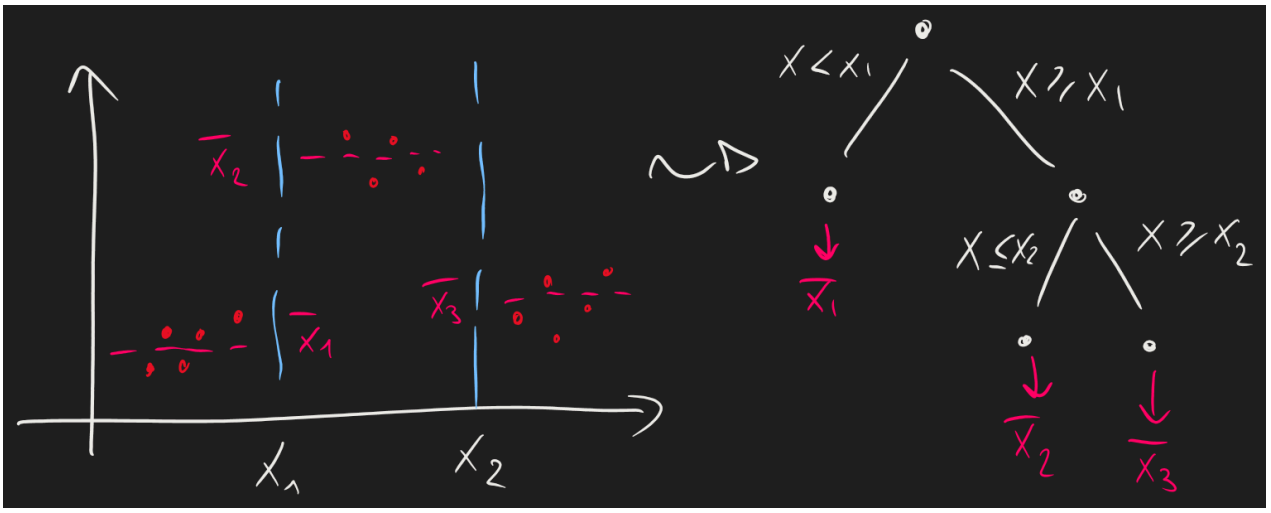
---

2. Regression Trees

2.1. Definition of Regression Tree

#Definizione

> Definizione (regression tree).
>
> *Regression Tree* is a type of decision tree (2) where *each leaf represents a numeric value* (which could be the result of an aggregation in the nodes of the leaf).



An intuitive approach for constructing a *regression tree* is to simply look at the plot and make our decision tree accordingly, just like we have done in the previous example.

2.2. Main Outline for Inducting Regression Trees

To *generalize* out intuitive approach, let us design the following outline for inducting regression trees.

ALGORITHM. (*Regression Tree Induction*)
Let $X_1, \dots, X_n$ be the explanatory variables and Y the target variable. Define $\bar{\mathbf{X}} = \bar{X}_1, \dots, \bar{X}_n$ every possible midpoint in $X_1, \dots, X_n$. Let $\mathcal{L}$ be the *cost function*, calculated in function of $\mathcal{L} = \mathcal{L}(\bar{x}_{ij}, y_i)$. We induct the decision tree with the following recursive procedure:

- Calculate every possible loss $\mathcal{L}$ for every midpoint
- Select the variable and midpoint $\bar{x}_{ij}$ such that $\mathcal{L}$ is minimum
- Branch the current node into two parts, according to the variable X_i and to the midpoint $\bar{x}_{ij}$.
- Repeat the same procedure for the generated nodes

OBSERVATION. Without putting any *stopping criterions*, this type of tree is very prone to *overfitting*. This will be the base for us to define *pre/post*-pruning techniques.

POSSIBLE LOSS FUNCTION MEASURES. Our loss function $\mathcal{L}$ can take in the following forms (and are also implemented in the Scikit-Learn library):

- *MSE*: Mean Squared Error

$$MSE(\hat{y}, y) = \frac{1}{n} \sum_n (y - \hat{y})^2$$

- *MAE*: Mean Absolute Error

$$MAE(\hat{y}, y) = \frac{1}{n} \sum_n |y - \hat{y}|$$

- *F-MSE*: Friedman-Mean Squared error; same as *MSE*, but has an iterative nature

X

Breve descrizione qui

X

0. Voci correlate

- Decision Tree Models
- Regression Trees
- Problems in Machine Learning

1. Introduction to the Problem

As introduced previously, *decision tree* is the model which is the most *prone* to overfitting. In fact:

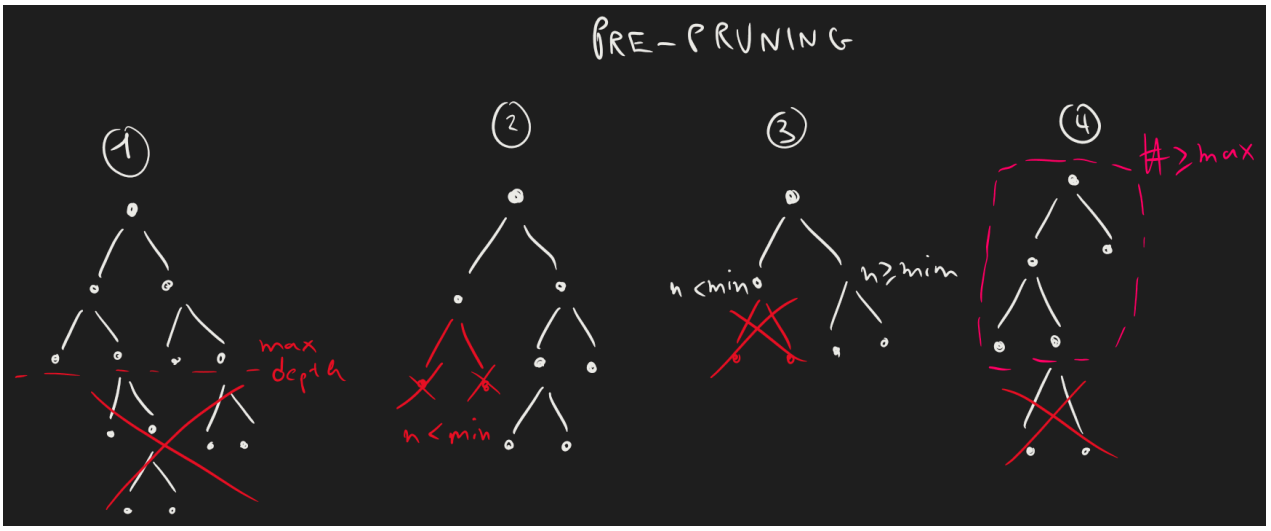
- With *decision tree classificators*, we have the goal to reach *leaf node purity*, even if it means reducing sampling size
- With *regression trees* we have to reach a *low* cost measure, which might mean reducing sampling size.
  To avoid *overfitting*, we can implement some measures. In the context of *decision trees*, these techniques are said to be *"pruning techniques"*. We have to main ways to do *pruning*.
- *Pre-pruning*: Choose a threshold for which we decide to remove or not split a node
- *Post-pruning*: Remove branches or single nodes of a *"fully grown tree"*, then evaluate with unseen data to obtain the *"best-pruned tree"*

X

2. Prepruning Techniques

We have mainly four techniques for *prepruning a tree*, all of which are implemented in the *Scikit-Learn* library.

1. **Limiting the maximum depth**: Define a certain *"maximum depth"* which can prevent a tree from growing too dep and capturing noisy data or outliers
2. **Defining minimum samples per leaf**: Define a number n of samples which every *leaf node* should have, or else they will be excluded from the tree. This helps in creating *simpler trees*.
3. **Defining minimum samples for branching**: Same as above, but the number n affects whether to branch a node or not
4. **Defining maximum number of leaf nodes**: Set a maximum number of *leaf nodes per tree*, which can prevent the tree from growing too much.

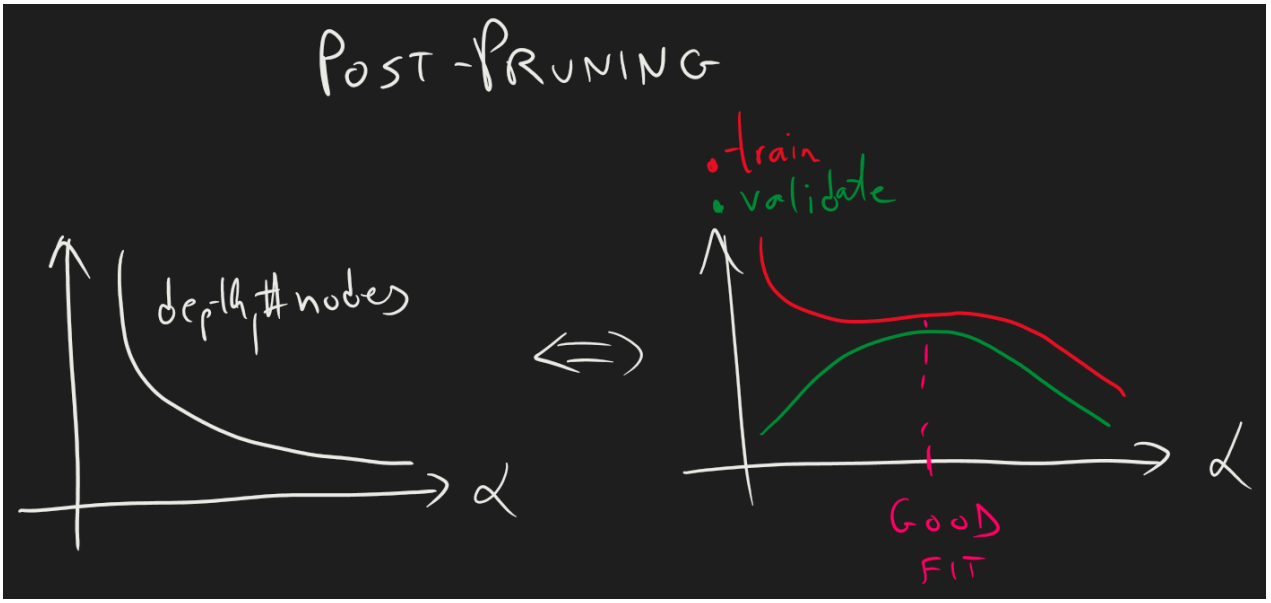


X

3. Postpruning Techniques

We mainly have one technique for *post pruning*.

- **Cost Complexity Pruning** (or known as α coefficient): Prune decision trees to avoid overfitting while optimizing the trade-off between pruning level and tree complexity. The *main idea* is to *grow a full decision tree*, remove some *nodes* and find the *best tree* after testing all options.



X

Ensemble Learning

X

Main idea of ensemble learning. Advantages and disadvantages. Motivations: statistical and computational reasons. Architecture of an ensemble learner.

X

0. Voci correlate

- Problems in Machine Learning
- Model Selection

1. Main Idea of Ensemble Learning

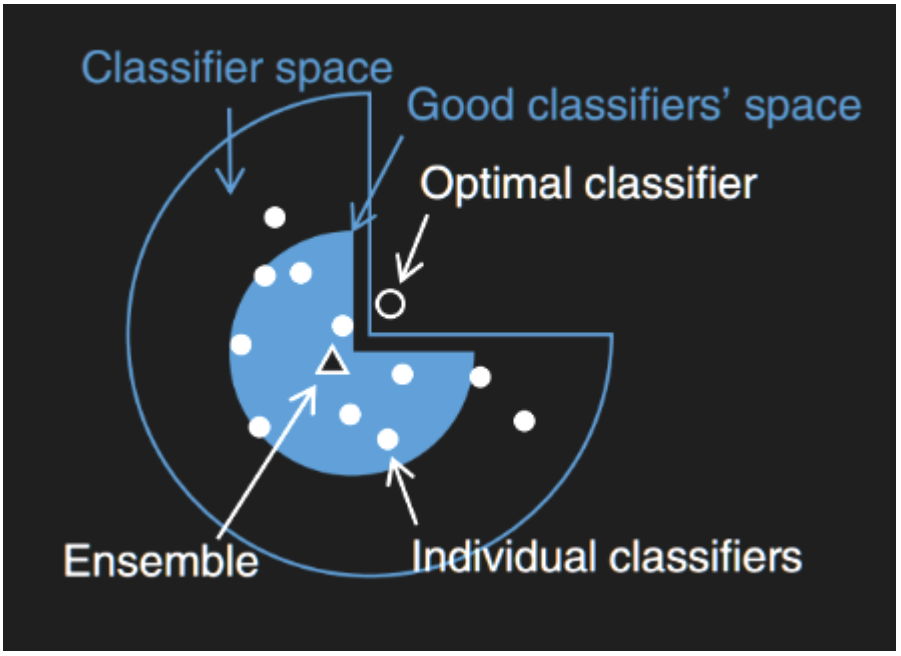
Observation. Until now, we have proceeded with the *"selective"* paradigm; that is, with a Machine Learning project we have considered only picking a final model (Model Selection), according to their best scores or aspects. However, are there any other ways to deal with the fact that we have "too many models" to use?

Idea. The main idea of *ensemble learning* is to aggregate models together, in order to obtain a final prediction. In a way, we are using the *wisdom of the crowd*.

Advantages: Predictions tend to be more accurate
Disadvantages: Complex might become too complex, leading to large costs or non-interpretability

Technical Reasons

1. *Statistical*: Combining good solutions give us a better one. We can view this in the following way: we have a space of classifiers, where we have good classifiers. Given a finite amount of data, all of their hypothesis are equally good; averaging these accurate classifiers might be a better approximation.
2. *Computational*: We can parallelize our inference process by splitting the data and feeding them to different models; or we can resample data (bootstrapping) and feed them to the models; we can also take a "divide et impera" approach, where we split big problems into a smaller problem for each model.

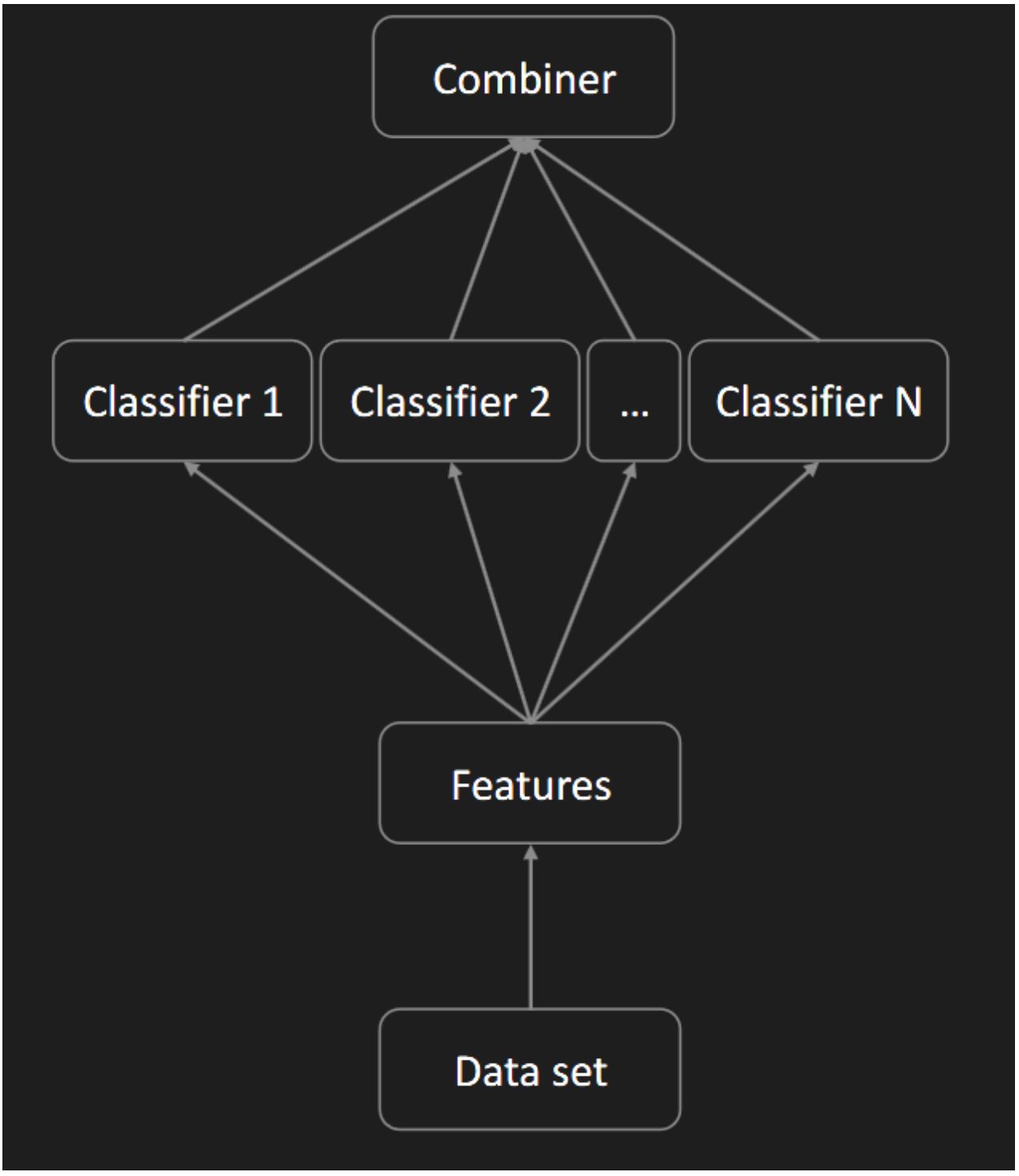


X

2. Architecture of Ensemble Learners

Q. How is an ensemble set up, to be exact?

Each ensemble learner has four levels, and they are:
Data Level: How to preprocess the data?
Feature Level: Which features do we use for each classifier?
Classifier Level: Which models to use? How many? How do we train them: sequentially or in parallel?
Combiner Level: How are the individual outputs of each model combined into a single answer?

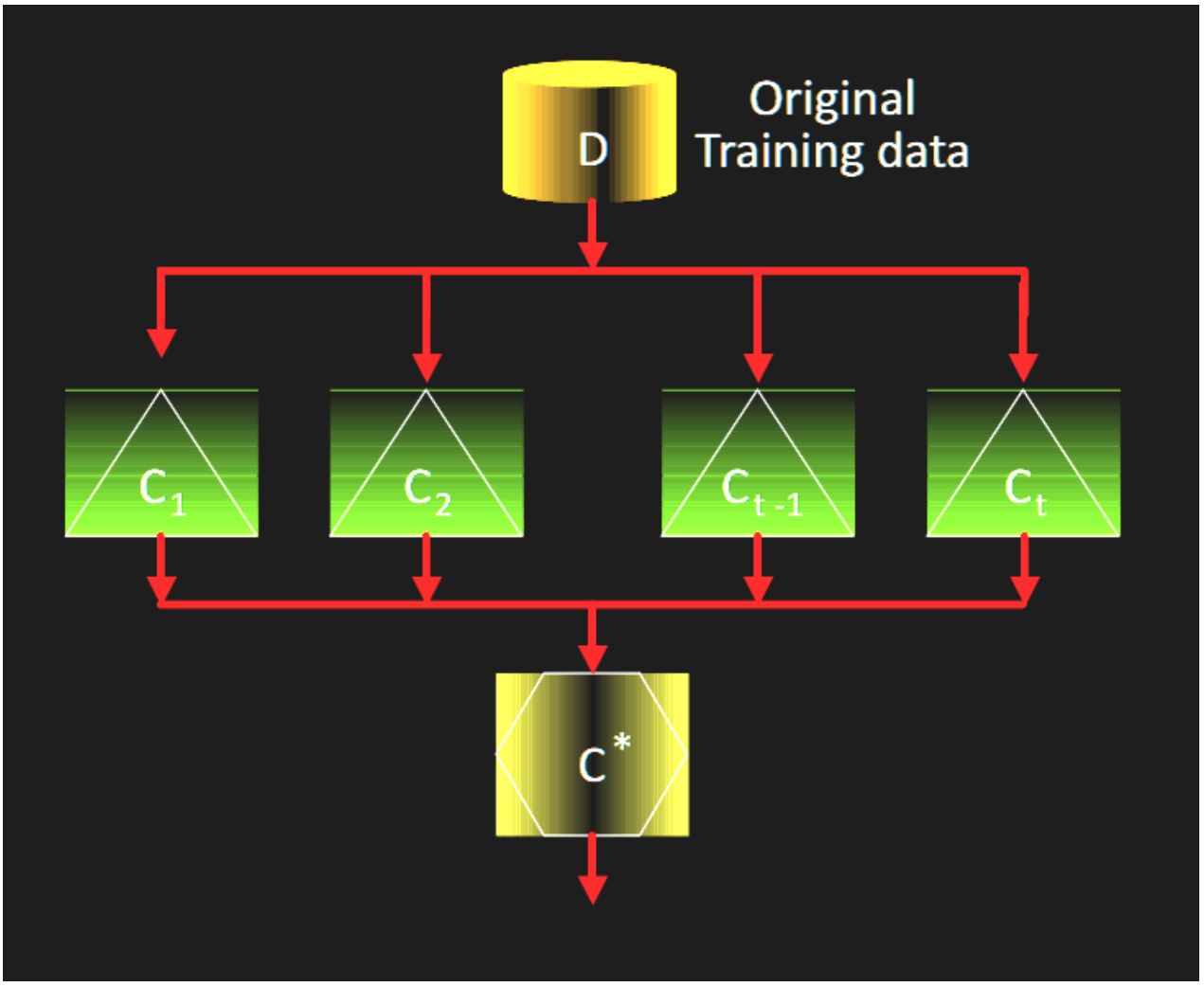


A set of answers for the questions previously put, makes an *ensemble algorithm*.

X

3. Example: Majority Vote

EXAMPLE. (*Majority Vote*)
The *simplest* ensemble learner is the *majority vote*; it considers each *model* as their own and are being fed the whole dataset, and the combiner aggregates their predictions by either taking the *mean* or *mode*.



X

Bagging Learner

X

Bagging learner. Recall of bootstrapping and majority voting. Graphical example: bagging learner with 1-KNN predictors. Observations: bagging counterpart of a weak model is not always better than the original model. Particular type: random forest.

X

0. Voci correlate

- Ensemble Learning
- Instance-Based Learning
- Decision Tree Models

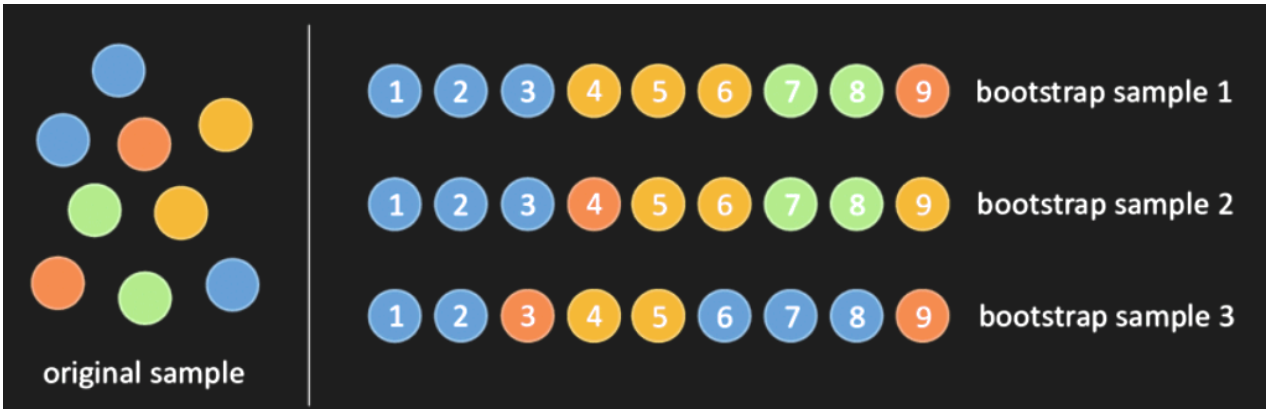
1. Definition of Bagging Learner

Bagging or **Bootstrap-aggregating** (Breiman 94) learner is a parallel *ensemble learner* model. In particular, its main idea is to combine *bootstrap* with *majority voting* (Ensemble Learning).

DEFINITION. (*Bagging according to Scikit-learn documentation*)
A *Bagging classifier* is an *ensemble meta-estimator* that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

Let us recall the following notions: *bootstrapping* and *majority voting*.

Bootstrapping is when we make *more and smaller* samples of an entire dataset (the rule of thumb is 60%), and distribute them to each model.



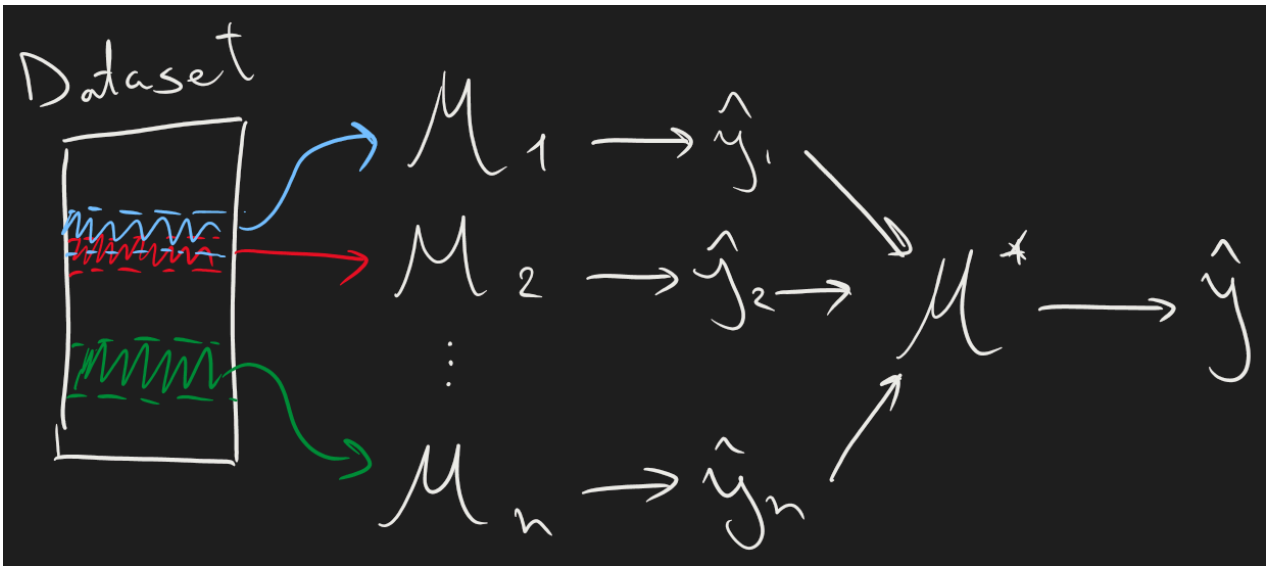
Majority Voting is when the classifier's predictions are combined together either by taking their *mean*, in the numerical case. If the target variable is categorical, we have two ways to determine the class:

- *Hard voting* if we simply take the mode of the predictions; so simply $\arg\max_C f_a(C)$
- *Soft voting* if we average the probabilities of each class, and keep class with highest probability; so we're taking $\arg\max_C p\{y \in C\}$

The main advantage of this type of method is that it mainly reduces *model variability*, giving us more consistent results; this comes from the fact that we are using *different training sets* for each model. In fact, we can think of the models having *"extreme answers"* as they have different *"perspectives"*, and by combining them we have a *good single answer*. Moreover, computational costs can be reduced as the training process can be parallelized (thus reducing a potential of time complexity from $O(\prod_i n_i)$ to $O(\max_i n_i)$).

So our bagging learner will work in the following way:

- **Training**: Given a labeled data set $(Z_n)_n$, choose the ensemble size L (usually ≈ 25) and the base classifier model. *Take* L bootstrap samples from Z and train classifiers on $(D_L)_L$, one classifier on each sample.
- **Inference**: For each new object, *classify* the new object x by all classifiers $(D_L)_L$. Taking the label assigned by classifier D_i to be a "vote" for the respective class, *assign* to x the class with the largest number of votes. Return the ensemble label of the new object.



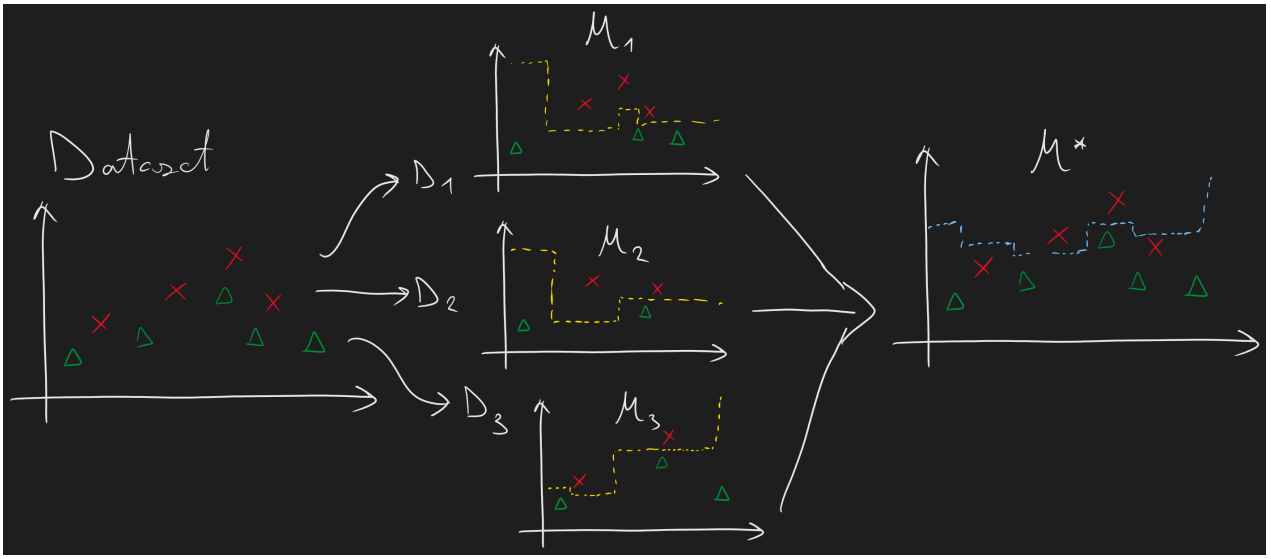
Observation. Usually bagging learners are used with *"weak learners"*, such as decision trees and/or KNN. However, let us notice that *not always* bagging learners are better than these weak learners by themselves!

X

2. Bagging with 1-NN Learners

Let us see a graphical example of bagging learners.

Example. Let L be a bagging classifier, with 1-NN as the base learner. Suppose that our bagging learner has 3 of its base learners. We can notice that each *base learner* draws a *Voronoi diagram* over the dataset, and the final output is the *"average"* between these straight lines. This same line is something that no single model could provide!



X

3. Random Forests

A **Random Forest** is a specific type of *bagging learner*. $RF \in BL$

Definition. (*Random forests according to Sklearn*)
A *random forest* is a *meta estimator* that fits *a number of decision tree classifiers* on various *sub-samples of*

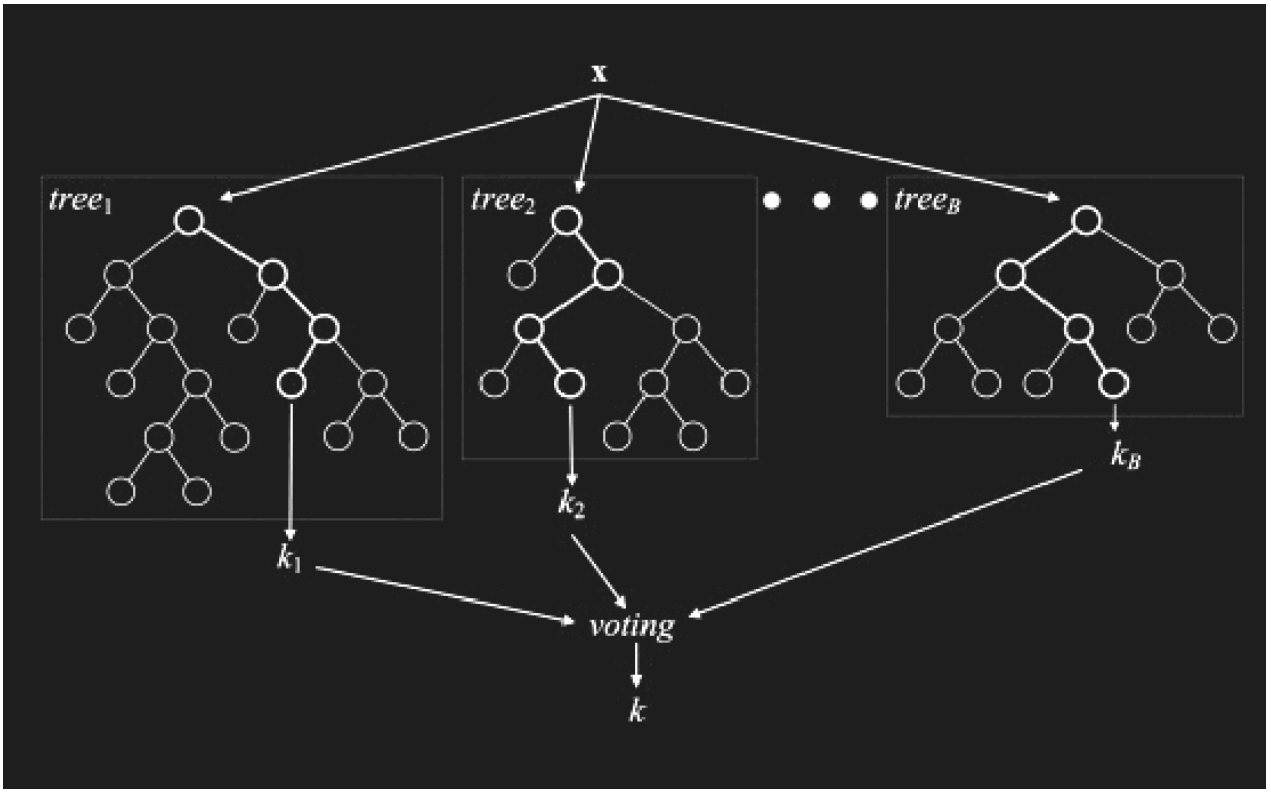
the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Idea. Reduce *overfitting* and *computational costs* by *"assigning"* different subset of features to different trees. They are assigned randomly, hence the *"random"* in its name.

Advantages. The main advantage of *random forests* is that they can run efficiently on large datasets with a large number of features.

Model. Let D be the dataset, and $(T_n)_n$ each tree of the forest tree. Then we:

1. Generate n samples of the dataset, called $(D_n)_n$; usually they have the same sizes
2. If there are a total of X attributes in the observations, then a number x is chosen such that x is very small than X. This is a *hyperparameter*, and common values include $\sqrt{X}$, $\log X$. Call each attributes subset as $(F_n)_n$
3. Build the individual trees T_i on F_i to their largest extent possible. The split attribute is choosen randomly



Observations.

- Each individual tree is a *horrible* learner; but by combining them we get good results.
- Computational costs are *reduced*, as I do not need to calculate anything to know when to stop building the tree, as everything is done randomly.

X

Boosted Learner

X

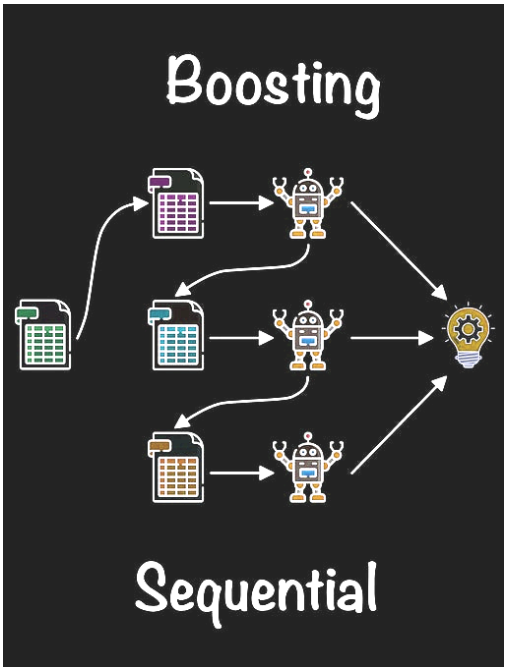
Boosting approach to ensemble learning. Main idea. Main algorithms: AdaBoost and Gradient Boosting. Idea of AdaBoost. Random Forests versus AdaBoost.

0. Voci correlate

- Ensemble Learning
- Gradient Descent

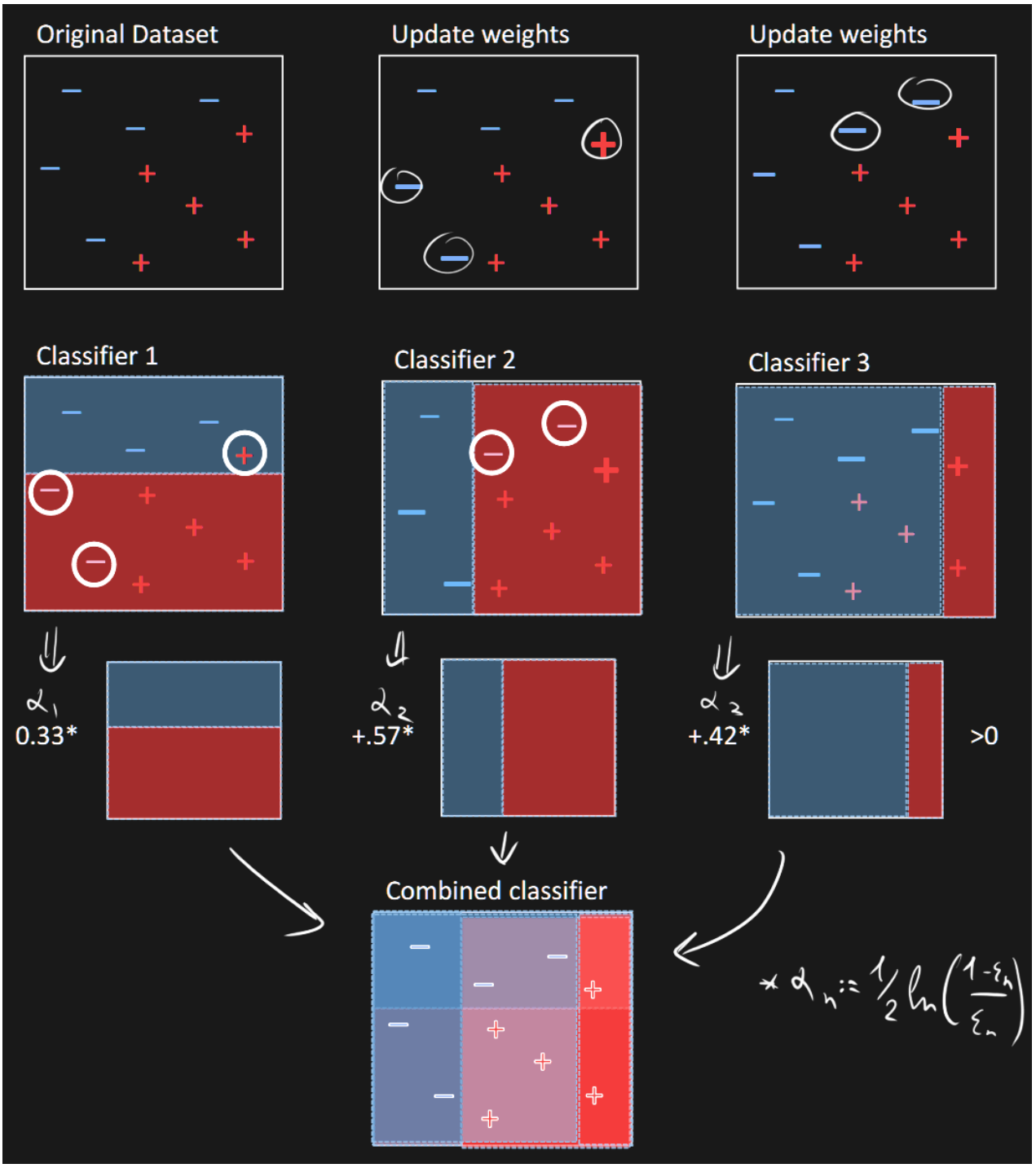
1. Idea of Boosting

IDEA. A new approach to *ensemble learning* (1) would be to train the sequence of models in a *sequential way*, rather than parallelizing their predictions. In particular, the idea is to *take a weak model* (such as KNN or decision tree) and "boost" it into a *strong learner*: the weak models make *shallow predictions*. In the end, we will make the prediction by considering every model's prediction, with weights. This improves the stability of the model and reduces bias.



Idea. A typical boosting algorithm would follow the following procedure:

- Let D_0 and M_0 be the initial dataset and the initial model.
- Fit M_0 on D_0, and get prediction $\hat{y}_0$
- Until we reach the maximum number of iterations or until we reach convergence, follow this iteration:
  - Fit M_i on D_i, get prediction $\hat{y}_i$
  - Find the mistakes in the prediction $\hat{y}_i$ and quantify them as ε_i
  - Update the dataset $D_{i-1} \to D_i$ by giving a bigger weights to the mistaken predictions
- Calculate weight α_i of each model
- Combine each model by taking the final prediction as $\sum_i \alpha_i M(y_i)$.



X

2. AdaBoost

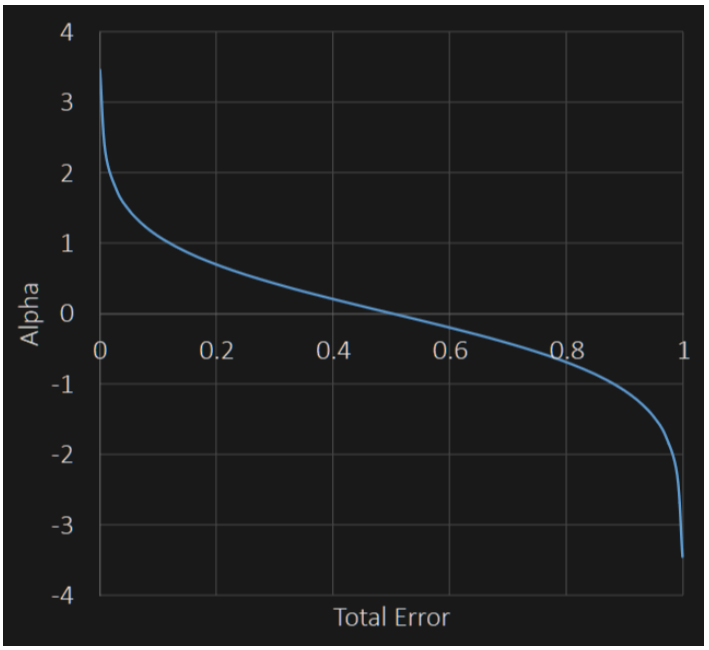
One of the main *boosted algorithms* is *AdaBoost*.

DEFINITION. (*AdaBoost according to Scikit-Learn*)
An *AdaBoost classifier* is a *meta-estimator* that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the *weights of incorrectly classified instances are adjusted* such that subsequent classifiers focus more on difficult cases.

IDEA. (*AdaBoost*)
AdaBoost is a supervised boosted ensemble model, which takes *decision trees* as the base learner. In particular, it takes *stumps*: they are decision trees with only a level of depth. In particular, this model follows the following definitions for the *model weight* and *error*:

$$\varepsilon_i := \sum_{\hat{y}_{i,n} \neq y_n} w_n$$
$$\alpha_i := \frac{1}{2} \ln \left(\frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$
$$\mathbf{w} \mapsto w_i := \begin{cases} w_i e^{\alpha_i}, y_i \neq \hat{y}_i \\ w_i e^{-\alpha_i}, y_i = \hat{y}_i \end{cases}$$

Image. (*Graphic of coefficient*)



OBSERVATION. When $\varepsilon \approx 1$, it means that the model is *completely wrong* in their predictions. Is it still bad? No, we can just inverse its answers and get *completely correct* predictions! So a *"bad error"* is when it is approaching 0.5, where it is *"weak in intensity"*.

PSEUDOCODE. The following pseudocode represents the training phase of an AdaBoost model.

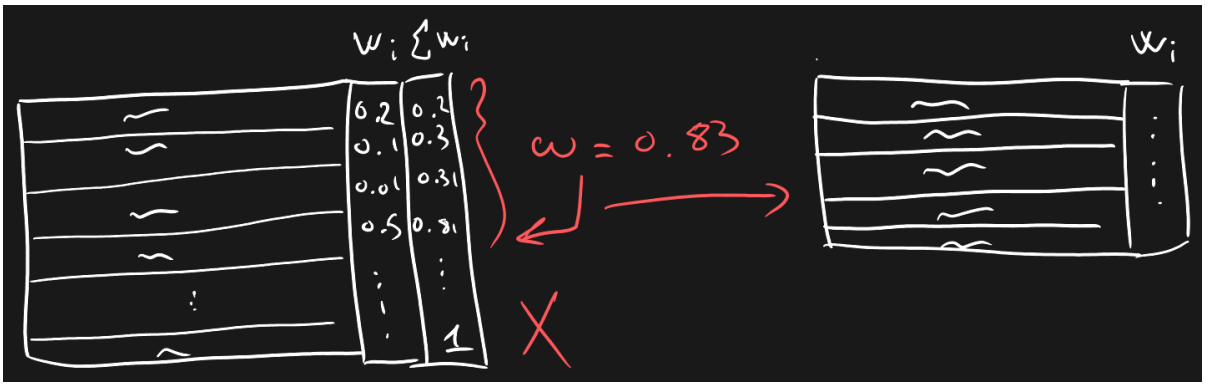
```
wts = uniform(len(X)) # Initialized uniform weights; it can also be random too
for i in 1,...,N:
        C_i <- model.train(X,y, wts)
        yhat <- C_i.predict(X) # Calculate predictions
        e <- wts*(y!=yhat) # Calculate error as weights multiplied by the number of
misclassified instances
        a_i <- 0.5*ln((1-e)/e) # Calculate model weight
        wts <- wts*exp(-a_i * y * yhat)
        wts <- wts/sum(wts)
```

OBSERVATION. When making the predictions on a model, how do we *consider* the weights on the dataset $\mathbf{w}$? We have two ways.

- During the fitting phase, use the *weighted Gini Index*, defined as

$$\text{Gini}_{\mathbf{w}} := 1 - \sum_n f_n^2(w_n) \iff f_n^2(w_n) := \frac{w_n |C_n|}{w_n |C_n| + (1 - w_n)(1 - |C_n|)}$$

- Resample the dataset by the weights; basically we calculate the cumulative sum of each row's weight, generate a random number ω and select all classes with cumulative sum lower than the randomly generated number, and then reset the weights for the next model to use. Alternatively, we can simply duplicate rows according to their weight.



OBSERVATION. Note that *AdaBoost* and *Random Forests* (1) are "similar" in the sense they use *decision tree* as the base models. Let us focus on their differences.

- *Random forests* make full trees
- The order is ignored by the *random forest*, each *tree* is considered as "equally good" to the others
- *AdaBoost* creates *stumps*, which are extremely simplified decision trees
- Order matters in *AdaBoost*, as each stump takes the previous stump's mistake; so their votes are considered differently

---

X

---

3. Gradient Boosting

(*Note: we will just see the idea*)

IDEA. This time each stump is created and evaluated in a similar fashion to the *gradient descent* (Gradient Descent); basically we will take a differentiable (1) loss function $\mathcal{L}$ and to calculate (and update) weights we will consider its gradient $\nabla\mathcal{L}$, alongside with a number called *"learning rate"* η.

For more details see this page by Google: https://developers.google.com/machine-learning/decision-forests/gradient-boosting

---

X

Stacking Learner

X

Stacking learners: main idea.

X

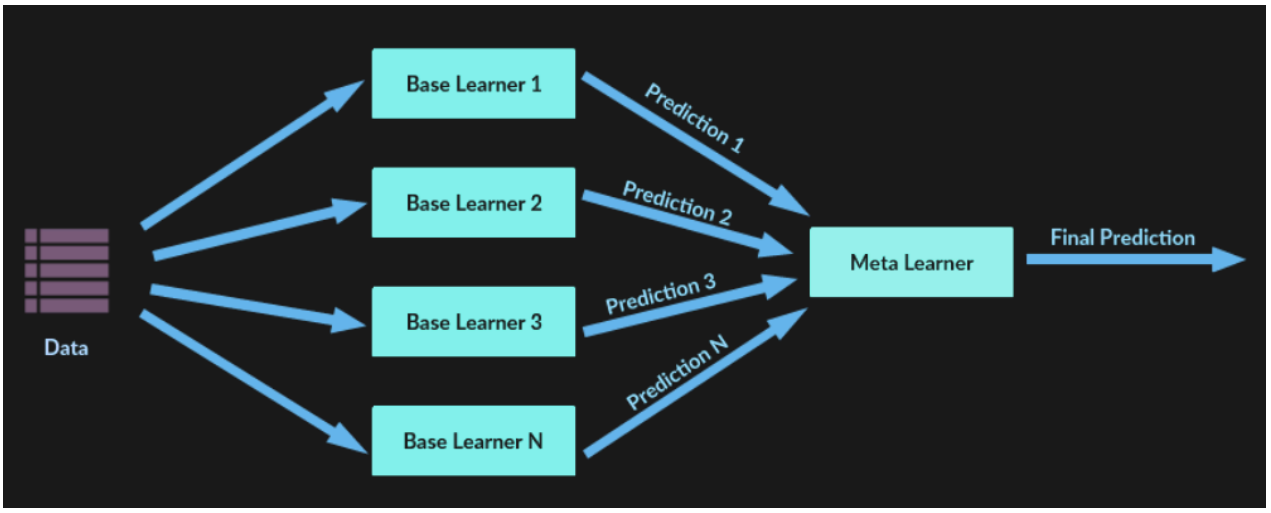
---

0. Voci correlate

- Ensemble Learning

1. Idea of Stacking Learners

IDEA. *Stacking* (or blending) is a *supervised ensemble model*, which uses *heterogeneous learners* in parallel to get different evaluations; then another learner, called *"meta learner"* will make the final decision.



DEFINITION. (*According to Scikit-Learn*)
Stack of estimators with a final classifier.
Stacked generalization consists in *stacking the output of individual estimators* and using a *different classifier to make a final prediction*. Stacking allows to use the *strength of each individual estimator* by using their output as input of a final estimator.

ADVANTAGE. This model is extremely flexible, as it can use multiple models; we can even use ensemble models! Moreover, as we use heterogeneous estimators, we can make the most of their advantages.
DISADVANTAGE. Difficulty in interpretability

Q. How do we make *meta learner* learn to make predictions?
A. We use *data splitting methods*; we will use the *"main fold"* on the first layer, and the *"validation fold"* on the last layer (i.e. meta-learner).

X

Introduction to Artificial Neural Networks

X

Introduction to Artificial Neural Networks (ANN). The Perceptron model: definition and training algorithm. Limitations of the Perceptron: non linearly separable problems. The Multi-Layer Perceptron model: definition of layer and its types. The Delta rule.

X

---

0. Voci correlate

- Neural Network
- Gradient Descent
- Supervised and Unsupervised Learning

1. Definition and Motivation of Artificial Neural Networks

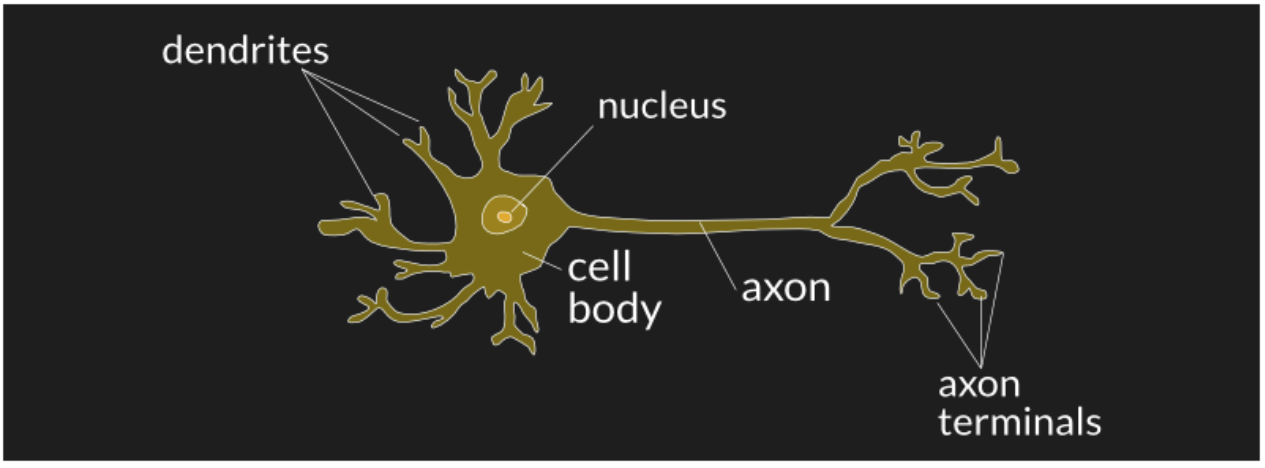
- *"...ANNs provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions*
  from examples..."
- *"ANN learning is robust to errors in the training data and has been successfully applied to problems such as*
  interpreting visual scenes, speech recognition, and learning
  robot control strategies."
  Both of them are quotes from Tom Mitchell's Machine Learning, describing both the definition and the strengths of *Artificial Neural Networks*.

Model Definition. Artificial Neural Network is a non-parametric supervised model. In particular, they are *universal approximators* and learn *"automatically"*.

Advantages: The main advantage of ANNs is that they are:

- Flexible for every problem: in fact this is applied to both clustering and regression problems, such as automated driving, OCR recognition and so on...
- Less data preprocessing is required: ANNs tend to be robust to outliers and errors
  In other words, ANN can be independent and can be used for highly complex problems.

Idea. The idea of ANNs is to replicate the human brain, focusing on the *neurons*. In fact, we can say that an ANN is composed by singular units called *"Perceptrons"*, which are the computer-equivalent of human neurons.



X

2. The Perceptron Model

Let us start with the foundation of an ANN, that is the perceptron model.

Definition. (*Perceptron Model*)
$f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ is said to be a *perceptron* if it possesses the following charasteristics:
It has a function $a : \mathbb{R} \longrightarrow \mathbb{R}$, called the *activation function*, following these conditions:

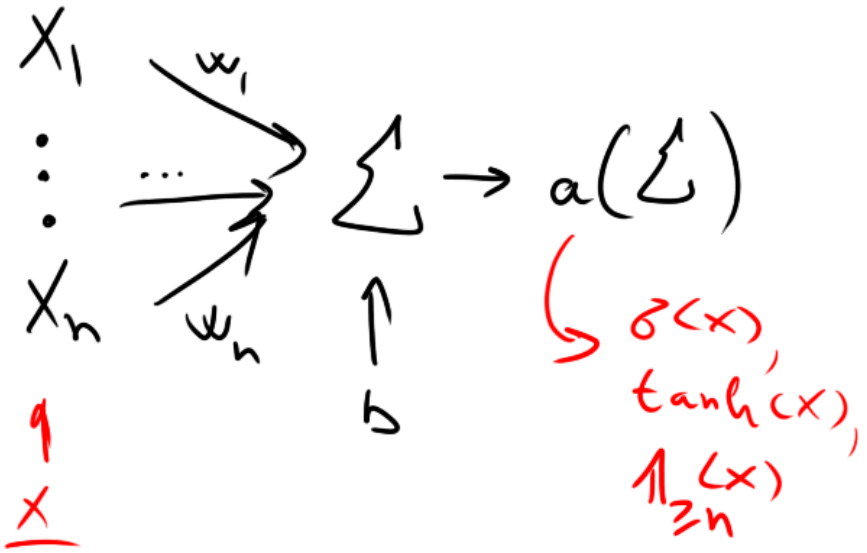
- Has a *"squashing effect"*, meaning it improves numerical calculations by adding constraints (such as minimas/maximas)
- It must be *monotonically non-decreasing*, meaning that $a'(\mathbb{R}) \subseteq \mathbb{R}^+$; this is to preserve the order of the input

It is associated to a vector $\underline{w} \in \mathbb{R}^n$ and number $b \in \mathbb{R}$: $\underline{w}$ is said to be the *weights* of f and b is the *bias*. The output of a given input $\underline{x} \in \mathbb{R}^n$ is calculated as follows:

$$f(\underline{x}) = a\left(\langle \underline{w}, \underline{x} \rangle + b\right)$$

Where $\langle \cdot, \cdot \rangle$ denotes the inner product in $\mathbb{R}^n$.

Graphically we have something that takes an *input* as a sequence of numbers, calculates their sum multiplied by the weights (and bias) - which is called the *activation input* or *logit* - and then takes the activation function of the activation input.



Learning Process. To train our *single perceptron*, we will use an iterative method. In particular, it is a *trial and error process*, as we are making small adjustments to the weights by each step.
So the algorithm is as follows:
Initialize the weights and biases randomly
For a defined numbers of iteration, calculate each $w \in \underline{w}$ as $w + \eta((t - o)x_i)$, where:

- $\eta \in [0, 1]$ is the *learning rate*
  - If it's 0, it's not learning from the model anymore; if it's 1, it completely trusts the error and makes a complete adjustment of the weight.
  - It represents the percentage of error reduced on each iteration
  - The number should converge towards $\eta \longrightarrow 0$ so we can reach some sort of convergence
- $t - o$ is the error between the actual and predicted output.
  In this way we fitted our model by obtaining our weights $\underline{w}$.

Note: this algorithm requires for the variables to be standardized!

Observation. We can observe that this model is better at *interpretation* than *extrapolation*, meaning that it has lost any sort of interpretability as it *only* minimizes our error! Meaning that whenever we use a NN for a problem, the dataset has to be as complete as possible.

X

---

3. Multi-Layer Perceptron Model

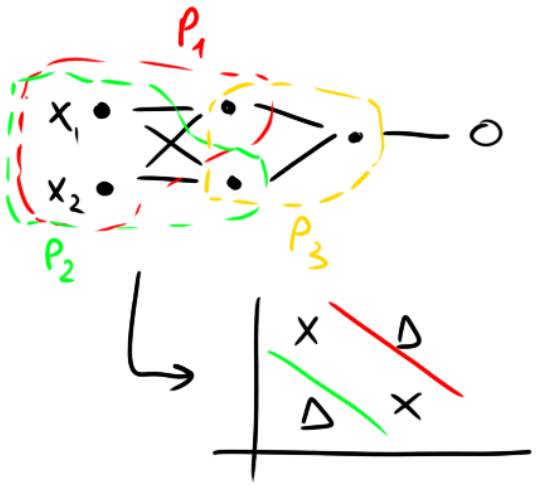
Problem: Single perceptrons are *"too simple"*, as they tend to be *linear* and only minimize the loss locally. The solution was to combine multiple *perceptrons* into an *Artificial Neural Network*!

Example. Let us define the following dataset, representing the XOR problem:

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

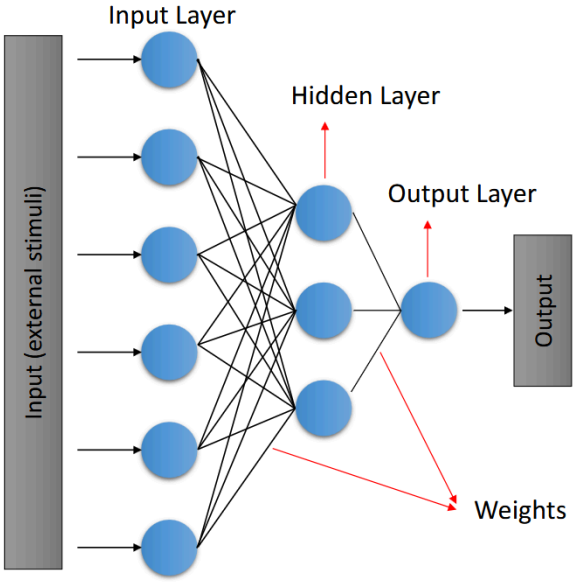
This is a *non linearly-separable* problem, as a single line is insufficient to form a *decision boundary* on the dataset. To solve this appropriately, we can use more perceptron! In particular, we combine three perceptrons in the following manner:

- The first two perceptrons are fed by the same inputs x_1, x_2 and we will call their output as $z^{(1)}$
- The second perceptron will be fed by the first two perceptrons, accepting $z^{(1)}$ as the input
  In this way, we can have two decision boundaries, as we have two layers of perceptrons!



Definition. Given a *Multi-Layer Perceptron*, we will call:

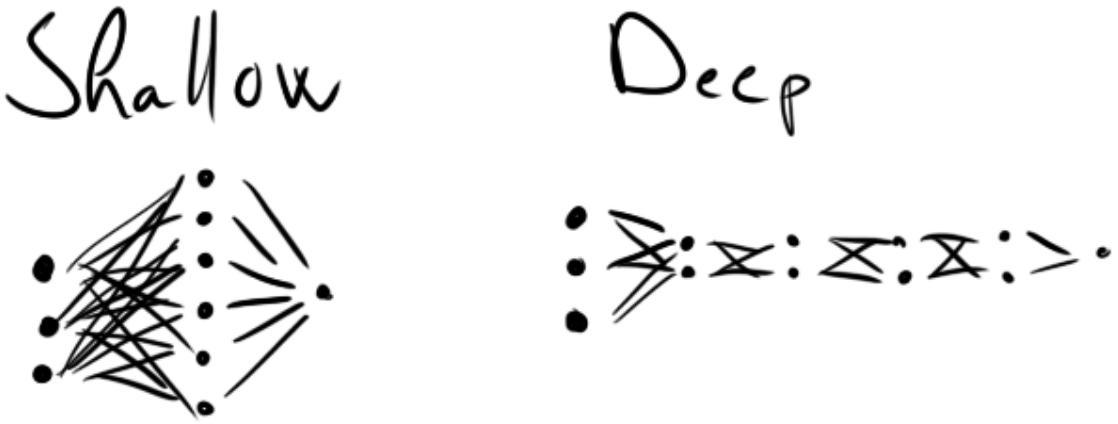
- The input layer the perceptrons receiving the input $\underline{x}$
- The output layer the perceptrons giving the output
- The hidden layer the ones in between the input and output layer



By deciding *"how"* we want to stack up our perceptrons, we can have different versions of an Artificial Neural Network

Definition. (Feed Forward MultiLayer Perception)
An ANN is said to be a *"Feed Forward MultiLayer Perception"* (FFMLP) if the nodes are connected in an input-to-output manner; meaning that the hidden layers are not able to send outputs to the external environment.
In particular, we have that a FFMLP is said to be:

- Shallow if it has more *weights* than *layers*
- Deep if it has more *layers* than *weights*



Notation. For a given FFMLP, we can use the following notations:

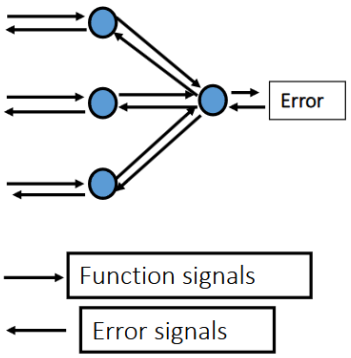
- We can call a FFMLP as a x-y-...-z MLP where x, y, ..., z are the numbers of perceptrons in each layer
- Given a layer i, we will call:
  - $\mathbf{w}^{(i)}$ the weights and $b^{(i)}$ the bias
  - $z^{(i)}$ the resulting logit of the layer
  - $a^{(i)}$ the resulting output from the logit
- **Observation**: Using the above defined notation, we have the following equations:
  $z^{(i)} = \langle \mathbf{w}^{(i)}, a^{(i-1)} \rangle + b^{(i)}$ and $a^{(i)} = a^{(i)}\left(z^{(i)}\right)$.

Q. Which is better? A shallow MLP of a deep MLP?
A. A partial answer is given by the *Universal Approximation Theorem*

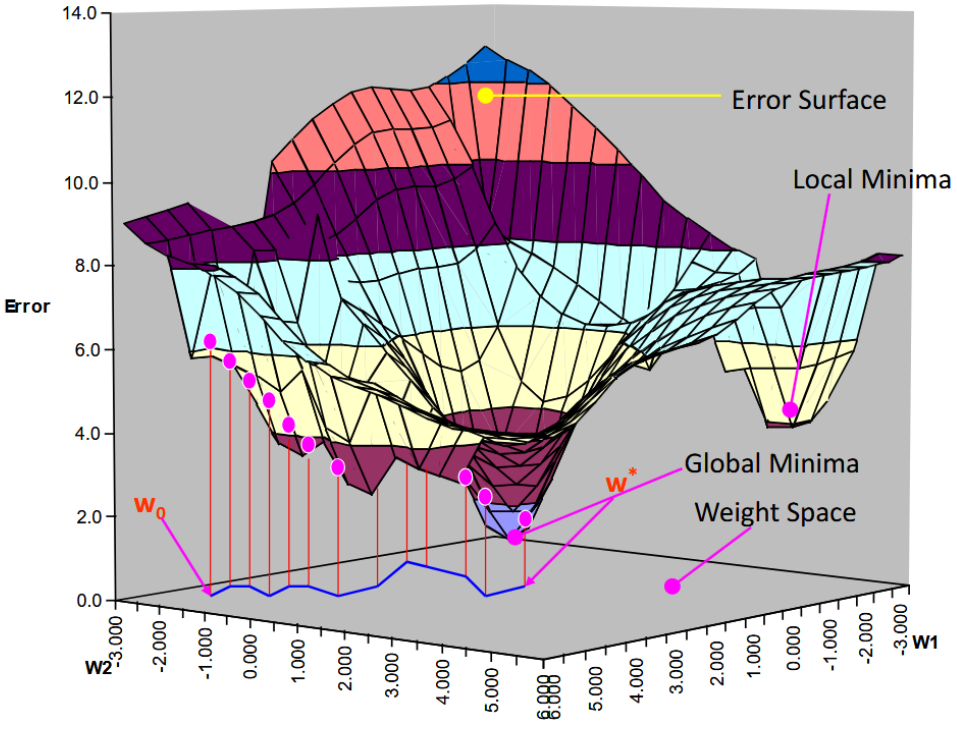
Theorem. (*Universal Approximation Theorem* or *Cybenko's Theorem*)
Given a family of neural networks, we have that for each function $f \in \mathcal{F}$ there exists a sequence of neural networks $(\phi_n)_n$ such that we have the convergence $\phi_n \xrightarrow{+\infty} f$
In other words, given enough time and neurons, any neural network will adjust well to a problem.

Observation. The UAT tells us that in terms of performance, any type of FFMLP should be able to adjust well to a problem if given enough iterations. However, deep NNs might be better in terms of performance, as there are *less weights* to update.

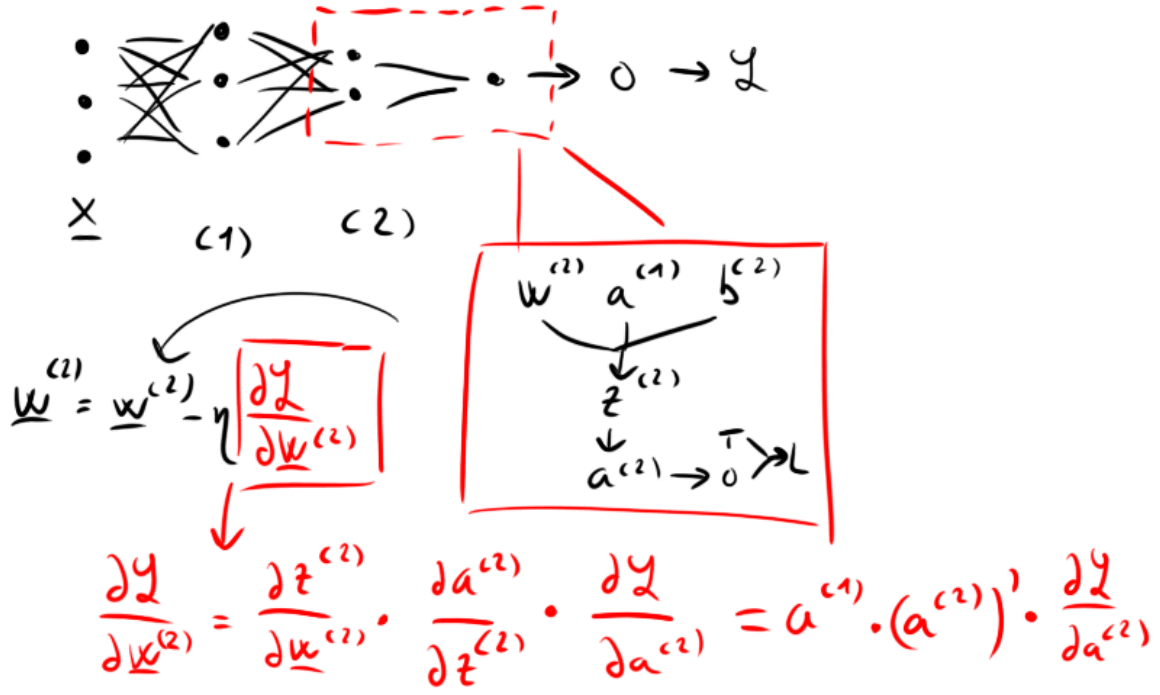
Learning Process. This time training MLPs become more complex as we need to "back-propagate" errors!



In the case of one-layered neural networks, we will have to consider our *error* as a *surface*, which should be minimized; to do it we can simply take its gradient and update the weight accordingly. This algorithm is known as *Gradient Descent*; the only requirement is that our loss function $\mathcal{L}$ has to be differentiable.



In the case of MLPs, the idea is even more complex as we need to effectively propagate the error through the layers; to do it, we will have to compute chain derivatives...



$$\frac{\partial Y}{\partial W^{(2)}} = \frac{\partial z^{(2)}}{\partial W^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial z^{(2)}} \cdot \frac{\partial Y}{\partial a^{(2)}} = a^{(1)} \cdot \left(a^{(2)}\right)' \cdot \frac{\partial Y}{\partial a^{(2)}}$$