NOVA IMS Information Management School

MAA

Mestrado em Métodos Analíticos Avançados
Master Program in Advanced Analytics

An Overview of the main Machine Learning Models

From Theory to Algorithms

Lucas Marnoto Figueiredo

Dissertation presented as the partial requirement for obtaining a Master's degree in Data Science and Advanced Analytics

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School

Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

An Overview of the main Machine Learning Models

From theory to Algorithms

by

Lucas Marnoto Figueiredo

Dissertation presented as the partial requirement for obtaining a Master's degree in Data Science and Advanced Analytics

Advisor: Mauro Castelli

June/2020

ACKNOWLEDGEMENTS

First of all, I would like to give a special thanks to my advisor, Professor Mauro Castelli for all the support and guidance.

To Professor Rosa Meo and Professor Roberto Esposito for giving me the opportunity to develop this thesis in University of Turin with them.

To my family and girlfriend for always encouraging me to give the best of me in every single moment.

"Feeling gratitude and not expressing it is like wrapping a present and not giving it"
William Ward

ABSTRACT

In the context of solving highly complex problems, Artificial Intelligence shows an exponential growth over the past years allowing the Machine Learning to augment and sometimes to outperform the human learning. From driverless cars to automatic recommendation on Netflix, we are surrounded by AI, even if we do not notice it. Furthermore, companies have recently adopted new frameworks in their routines which are mainly composed by algorithms able to solve complex problems in a short period of time.

The growth of AI technologies has been absolutely stunning and yes, it is only possible because a sub-field of AI called Machine Learning is growing even faster. In a small scale, Machine Learning may be seen as a simple system able to find patterns on data and learn from it. However, it is precisely that learning process that in a large scale will allow machines to mimic the human behavior and perform tasks that would eventually require human intelligence. Just for us to have an idea, according to *Forbes* the global Machine Learning market was evaluated in $1.7B in 2017 and it is expected to reach almost $21B in 2024. Naturally, Machine Learning has become an attractive and profitable scientific area that demands continuous learning since there is always something new being discovered.

During the last decades, a huge number of algorithms have been proposed by the research community, which sometimes may cause some confusion of how and when to use each one of them. That is exactly what is pretended in this thesis, over the next chapters we are going to review the main Machine Learning models and their respective advantages/disadvantages.

KEYWORDS

Machine Learning; Algorithms; Prediction; Supervised Learning; Unsupervised Learning; Regression; Classification; Clustering.

METHODOLOGY

The elaboration of this master thesis was mainly based on already existing algorithms which implied a complete theorical framework. The development of this framework was a long process composed by several steps. The first of those steps was to identify the key concepts to discuss during this work, or alternatively to define which research questions should be answered with this work. The idea was not only to write the thesis about the Machine Learning models, but also describing in which circumstances each one of them should be used as well as defining their advantages and disadvantages. Since it was a highly theoretical subject it was necessary to carry out an intensive and rigorous study of literature review, which was the second step of the process. In the third step it was time to choose which algorithms would be reviewed as well as planning all the thesis structure. From the very beginning that the structure of this thesis was considered as a differentiating factor from other similar works. To achieve this, it was decided that each chapter would be dedicated to a family of algorithms (that share the same characteristics) and in each sub-chapter a specific algorithm would be explain. The fourth and last step of the process was to extract conclusion about the algorithms and to answer the main research questions of the thesis. This was made in two different moments, in the end of each chapter with a summary of the algorithm and in the end of the thesis with the results obtained.

Index

1. INTRODUCTION

Back in 2017, The famous journal *The Economist* defined data as the new oil and as the most valuable asset. The analogy is pretty clear since raw data has no value if it is not prepared and analysed, the same happens to the crude oil that needs to be refined into more useful products such as gasoline. Also, if we look to both assets in the timeline, we quickly realize that in terms of importance, data is in 21th century as the oil was in the 18th. The concept of Big Data has been around for years, but the companies never had either the infrastructure to store all the data or the technology to analyse it, something that drastically changed in the recent years. Nowadays, companies are storing huge amounts of data every minute and they have specialists such as Data Scientists to extract value from it.

Globally speaking, storing the data is a never-end exponential curve, every digital interaction leaves a footprint of data. According to a study of the *World Economic Forum*, in 2020 the digital universe will reach 44 Zettabytes (that is 44 with 21 zeros after it). People usually fall in the mistake of thinking that data is only increasing in terms of Volume, but in fact Big Data is described in the 4v's: Volume (the amount of collected data), Variety (different forms of data), Velocity (the speed of new generated data) and Veracity (the trustworthiness of the data). A fifth "v" may be added which refers to "Value", this may seem quite obvious but the data is only useful if we can convert huge amounts of data into value to the companies, so the data must have some intrinsic value.

Big data has taken the main stage, it is a *buzzword* that everybody wants to speak about. However, despite the fact big data provides a wonderful source of information, it become useless if it is not correctly understood, which lead us to automation. Handling such amount of data requires first of all infrastructures and resources such as Hadoop to run all that data. But even assuming we can process all that information, how can we extract value from it? Or in other words, how can we convert all that data into knowledge? And which algorithms should we use to accomplish that? The answer to those questions is all about Machine Learning and the ultimate goal of this thesis is to allow the readers to answer those question by themselves at the end of this work.

Machine Learning (ML) has been one of the most revolutionary breakthroughs of computer science by allowing computers to learn by themselves as they process large datasets. Furthermore, Machine Learning is widely adopted by the companies across the world as they start to be more aware of its potential. The concept of *Business Intelligence* was just the first step of a long journey where companies start to incorporate complex data analysis in their routines. The study of Machine Leaning aims to give a step further in advanced analytics where many data scientist, engineers and analysts get their opportunity to add something new on this subject specially in algorithms innovation and optimization. The improvements of the algorithms performance have been absolutely impressive allowing the ML algorithms to surpass humans in areas such as patterns detection and image recognition. All that research result in a crescent number of new algorithms, each one with their own variations. Specially for the newcomers, the diversity of models may seem confused and complicated, particularly in the time of choosing which algorithm fits better in each situation. The model selection is a crucial moment during the algorithm implementation since the wrong model will lead us to poor results since the beginning, even an experienced data scientist may have to test several models to decide which one performs best. This thesis is exactly focus on the model selection

phase, it is proposed to cover the main Machine Learning models, explaining the fundamental concepts as well as understanding the main strengths and weaknesses of each model.

However, it is not pretended to make a ranking of the best models and even less to try to choose the best model since each problem requires a specific model. This thesis will follow a universal approach in Machine Learning called *No Free Lunch Theorem.* That theorem should be the first concept to learn for someone starting in ML and states for the idea that there isn't any model better than another for all the possible circumstances or any model that fits perfectly in all problems. In the end of this introduction it is presented a quote from the famous statistician George Box that shortly explains that idea. With that said it is worth to think about the idea of generality versus specificity of the algorithm, we should consider the trade-off between an algorithm that may be very efficient solving a certain problem or a general algorithm that is reasonable in many problems but doesn't master any of those.

Every chapter follows the same structure, firstly with an introduction of the model alongside with a brief history whenever necessary; secondly a review of the algorithm with theoretical and mathematical foundations; and finally exposing the algorithm applications as well as the respective advantages and disadvantages. All of this is well summarized in the end of each chapter in a section called *Modell Overall* where the reader gets the opportunity to quickly understand the purpose of the algorithm, the main strengths and weaknesses as well as the metrics to use for the model evaluation.

"All models are wrong, but some are useful" – George Box

2. LITERATURE REVIEW

2.1 MACHINE LEARNING: THE DIFFERENCE TO AI AND DATA SCIENCE

Over the past years, terms like Data Science, Artificial Intelligence and Machine Learning have been buzzwords that are often incorrectly misunderstood. Despite the fact, each one of those disciplines has it is own purpose, they often overlap each other since most of the times the output of one discipline is the input of the other. Before digging deeper on Machine Learning, it is important to understand the *big picture* and clarify the differences between those terms as well as explain where the Machine Learning fits.

Starting by Data Science, as the name suggest it is all about data. It is a broad field of studies since it incorporates all the techniques that together transform data into knowledge. In other words, Data Science can be interpreted as a workflow that starts with the collection of huge amounts of data and ends by providing meaningful insights from the data.

> *"*Data science is an interdisciplinary field that uses scientific methods, processes, algorithms and systems to extract knowledge and insights within various forms, both structured and unstructured*."* (Shi & Iyengar, 2020)

Coming into Artificial Intelligence, it is also a wide-ranging branch of computer science but this time drive-focus to simulate intelligence in machines. To fully understand the concept of AI, we first need to consider the concepts of Intelligence and Learning. According to the specialists, intelligence is the ability to find solutions to problems, usually based on something that you have already learned. On the other hand, Learning is the process that leads to a change (usually an improvement) as a result of past experiences (Ambrose et al., 2010). With that said, if we can teach the machines how to learn by experience and to always choose the best solution for any kind of problem, we have created *Artificial Intelligence*. Humans usually learn from past experiences, in the case of machines, the learning is made through data, meaning the more data we feed into the machine/computer the more accurate will be the decisions/results.

> *"(…) F*or the present purpose the artificial intelligence problem is taken to be that of making a machine behave in ways that would be called intelligent if a human were so behaving*."* (Mccarthy et al., 1955)

Finally, Machine Learning is a sub-set of AI that aims to automate an analytical model able to improve by itself without being explicitly programmed to do that. It is an application of AI responsible to create an algorithm that is able to learn from the data and according to what was learned is also able to adjust itself in order to optimize the results. All the recent innovations of AI such as self-driving cars or translating speech have in their main structure a lot of complex math and lines of code that represent the Machine Learning. There are several Machine Learning types and plenty of algorithms, some of those algorithms are simple and others extremely complex. However, if we take

a closer look, we realise that all those algorithm share the same characteristic, they are nothing more than a function that when fed with data is able to progressively improve over time.

> *"Machine Learning is the process of teaching a computer to perform a particular task by improving its measure of performance with experience"* (Mitchelll, 1997)

More recently, a new area of studies called Deep Learning (DL) has also showed an exponential growth over the past years. Deep Learning is not only a sub-set of ML, but in some cases, it is also considered as an evolution. The main structural idea remains the same, both ML and DL, use data to learn and optimize the algorithm. Although, Deep Learning is one step closer to AI since it is not only learning by experience but also replicating the human brain to process information. This is achieved with Artificial Neural Networks which is a computational model inspired in the human brain behaviour. In terms of results, and when compared to ML, Deep Learning break all the records of accuracy and efficiency in some specific areas such as image recognition or sound recognition.

> *"Deep Learning is subfield of Machine Learning concerned with algorithms inspired by the structure and function of the brain called ANN (…) imitates the human brain in processing data and creating patterns for the use in decision making"* (Chen, 2017)

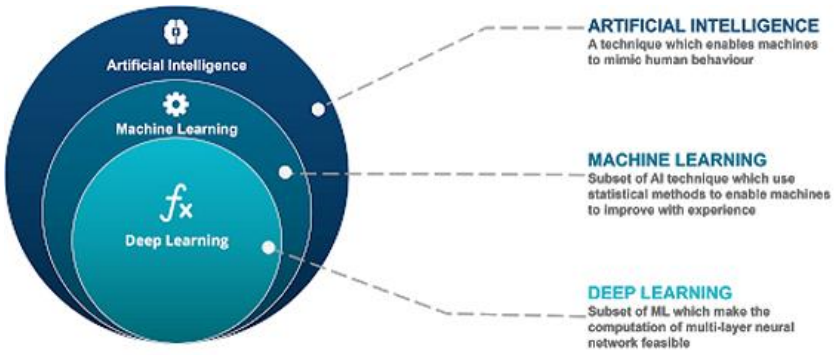


Fig 2.1. Differences between AI, ML and DL.

2.2 DATA PRE-PROCESSING AS A PRELIMINARY STEP OF MACHINE LEARNING

We are living in a data-driven world where the amount of data is growing exponentially which requires more and more computational power. If storing such an amount of data is not an easy task, analysing it is even more challenging because most of the time we are working with raw data that needs to be pre-processed before loading it on a Machine Learning model. A good pre-processing phase will for sure optimize the performance of the model, as well as save us time during the training and testing phase of the algorithm. One of the best approaches of preparing the data in order to extract knowledge from it is called KDD (Knowledge Discovery in Database). This is a technique often used on Data Mining but with some interesting concepts that should be applied before we run the data into a Machine Learning model.

2.3 KNOWLEDGE DISCOVERY IN DATABASES

The KDD process refers to the broad process of extracting knowledge from the data. It was proposed in 1996 by Fayyad who defined it as:

> *"KDD is concerned with development of methods and techniques for making sense of data (…) at the core of the process is the application of specific data-mining methods for pattern discovery and extraction."*

The ultimate objective of the KDD is to transform useless/raw data into understandable/useful data. The KDD is an interactive process of 9 steps, not necessarily linear which means it may be necessary to move back and forward to make some adjustments. In other words, most likely this process involves loops between different phases. The process is represented as a workflow in figure 2.2 and it includes the following steps:

1. This first step is a crucial moment in the process, it is a preparatory step that sets the scene of what we want to achieve and how we will do it in terms of transformations and algorithms. In this step, we define the application domain according to the goals of the end-user.
2. After identifying the KDD goals, it is time to select a target data among all the available data in order to solve our specific problem. This phase includes not only discover where the data is stored but also how to import it and if necessary, how to integrate it. The integration of the data is sub-step that may be required if we are importing data from different sources and we want to compilate all in a single location. Selecting the right data is imperative since the algorithms will learn based on the embedded data.
3. Having obtained the target data, the next step is known as pre-processing and cleaning. In this step we guarantee the reliability of the data. The range of possible techniques to use here is huge but at least the next four tasks must be followed:
    - 3.1 Remove all the noisy data that is wrong or irrelevant (including outliers).
    - 3.2 Find the right strategy to fill eventual missing values.
    - 3.3 Checking the time features and make the necessary adjustments (e.g changing the format).
    - 3.4 Creating "calculated features" based on other features.

4. More significant changes are usually needed in the data before the analysis begins. Those changes are called data transformation which incorporates techniques such as feature selection, dimension reduction and attribute transformation. Those concepts will be covered ahead on this work, but shortly, it is in this stage that we select the features and the data we need according to the goals of the first step.
5. After the data is prepared, it is time to select the data mining task to use. According to our data and goals we can choose between an Optimization, Classification, Regression or Clustering analysis.
6. The sixth step is the selection of the algorithm that is able to find patterns on the data and eventually make predictions. Given the variety of algorithms, the selection of the algorithm may not be an easy task. Throughout this thesis, the main ML algorithms will be explained allowing a better understanding of which algorithm should we use at each situation.
7. We finally reached the point of implementing the algorithm and running it. This phase may take some time, not only due to the running time of the algorithm, but also because it may be required some adjustments on the hyperparameters.
8. This step is all about evaluation. We can now evaluate and interpret the results/patterns discovered as well as comparing them to the goals we pre-define on step 1. If the goals were satisfied, we can move to the last step, otherwise it is recommended to go back on the KDD process to change whatever is necessary to achieve the initial goals.
9. The last point of the workflow is to consolidate and use all the discovered knowledge. The success of this step will determine the effectiveness of the KDD process. The process ends by incorporating the knowledge on the system and finding a solution to the initial goals.

The overall KDD process is illustrated in fig 2.2.

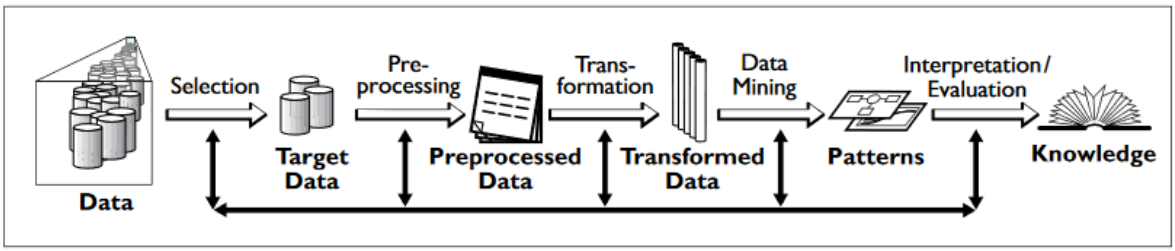


Fig 2.2. Knowledge Discovery in Databases pipeline.

2.4 DIMENSION REDUCTION

One of the most used pre-processing techniques is called Dimension Reduction and it is especially useful whenever we are working with huge datasets. Formally, the data reduction is a technique to obtain a reduced representation of the dataset, that is much smaller in volume but it closely maintains the integrity of the original data (Han et al., 2012). There are 3 different approaches of data reduction, the first one is called Numerosity Reduction which will replace the original data by a smaller representation of it. The second one is the Data Compression where the data is reduced

into a compressed format of the initial data. The last one is the Dimensionality reduction which is the most used method that requires a more detailed and careful review as follows.

The Dimensionality Reduction is a technique used in high-dimension datasets, usually working with a large number of features implies many mathematical challenges. One of those problems is when a high-dimensional dataset has high correlated features, for example a dataset with the date of birth and the age of individuals has two completely correlated variables. In that example one of the features may be dispensable and we still understand the underlying phoneme in interest. One simple way to reduce the number of dimensions is with the *feature selection* where only the relevant features are selected for the model. The problem with the feature selection is that we rarely found perfect correlation between the features which means that each one of them is adding some value to the dataset. In that case, the Principal Components Analysis (PCA) may be a good alternative. The PCA is a statistical procedure proposed by Pearson and Hotelling with the goal of transforming a number of correlated variables into a smaller number of uncorrelated variables (named as Principal Components).

The Principal Components searches for a n-dimensional orthogonal vector that is able to project the initial data into a small space. The PCA can be decomposed in 4 steps:

- Normalizing the data in order to guarantee that all the attributes fall in the same range.
- Computing a set of orthogonal vectors that are perpendicular between each other.
- Assign the Principal Components in those vectors in decreasing order by significance. The significance is measure by the variability of the feature, so the first principal component accounts for as much variability in the data as possible, and the next ones will account with as much remaining variability as possible.
- Finally, because the Principal Components are sorted in decreasing order, we can eliminate the last ones that do not give a significative contribute to the model

In short, the PCA is a dimensionality reduction that gives primacy to the variables with greater variability and it will reject the ones with lowest variability. Unlike the original variables, the principal components are not interpretable.

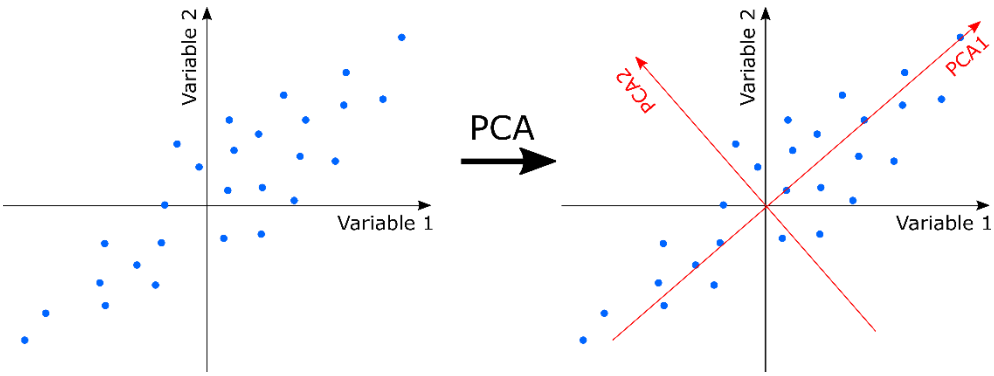


Fig 2.3. Representation of a dataset in a bidimensional space and its respective PCA application.

Another method of dimensionality reduction is the LDA that stands for Linear Discriminant Analysis. It is similar to the PCA method but instead of just trying to find the axes that maximizes the variance, the LDA will also searches for the axes that maximizes the separability of the known classes. Formally we can define LDA as:

"Given labelled data consisting of d-dimensional points $\boldsymbol{x}_i$ *along with their classes* y_i*, the goal of the linear discriminant analysis is to find the vector* $\boldsymbol{w}$ *that maximizes the separation between classes while preserving as much class discrimination as possible"* (Zaki & Meira, 2014)

Mathematically, we want to maximize the result of:

$$\max \frac{(\mu_x - \mu_o)}{S_x^2 + S_o^2}$$

Where of μ_x is the mean of class x, μ_o the mean of class o, S_x^2 the variance of class x and S_o^2 the variance of class o. Putting all together the μ_x-μ_o is the between-class scatter and the $S_x^2 + S_o^2$ represents the within-class scatter. Logically, in order to maximize that fraction we want to maximize the between-class scatter and minimize the within-class scatter.

Despite the fact the LDA and PCA are both linear methods, they have some clear differences. Firstly, the LDA assumes a normal distributed data where the features follow the principles of independence and homoscedasticity.

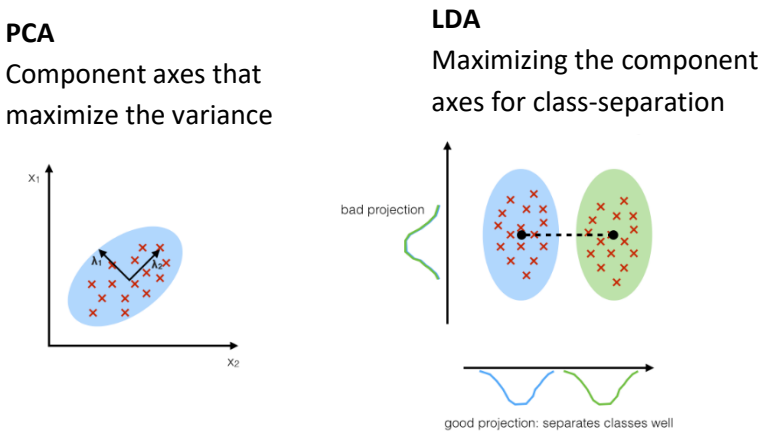


Fig 2.4. Principal Components Analysis vs Linear Discriminant Analysis.

So which method should we use for dimensionality reduction? Well, the answer really depends on the dataset. The LDA is recommended for large and complex datasets, but the PCA tend to outperform the LDA when there are fewer samples per feature.

2.5 DIFFERENT TYPES OF MACHINE LEARNING

In this section we will focus on the point 6 of the KDD workflow, by specifying the different types of Machine Learning algorithms. The main feature of any ML algorithm is to learn by experience and based on that try to make future predictions. That experience is acquired by feeding the algorithm with a training data and program the algorithm to find patterns and relationships between the inputs and outputs (which may or may not be known as we will see in this chapter)

With that said, we can distinguish two groups of Learning, the Supervised Learning and the Unsupervised Learning (Murphy, 2012). The main difference is that in the first one we already know *A Priori* what are the results of the output sample used as the training set and in that way the only goal of the algorithm is to build a function that best approximates the relationship between the inputs and the outputs. Mathematically we define supervised learning as a set of input-output pairs:

$$D = \{(x_i, y_i)\}_{i=1}^{N}$$

Where X_i represent the inputs, Y_i represent the outputs, D the training set and N is the number of training examples. X_i is a D-dimensional vector composed by a set of features and the output variable Y_i can be categorical or continuous depending if we are working on a classification or regression as it is explained below.

The Supervised Learning can be done in the context of classification when we want to map the input to output labels, or in the context of regression when it is pretended to map the inputs to a continuous output.

Algorithms of classification include models such as Decision Tree Classifier, Rule-Based Classifier, Neural Network Classifier, naive Bayesian Classifier, Neuro-Fuzzy classifier, Support Vector Machines and so on. All those model share the idea of having a function M that predicts the class label $\hat{y}$ for a given input example such as $\hat{y}=M(x)$, where $\hat{y}=\{c_1,c_2,\ldots,c_k\}$ represent the predicted class label. Regarding the regression models, there are also a huge variety of options like Linear Regression, Multiple Regression, Stepwise regression, Multivariate Adaptive Regression Splines, among others.

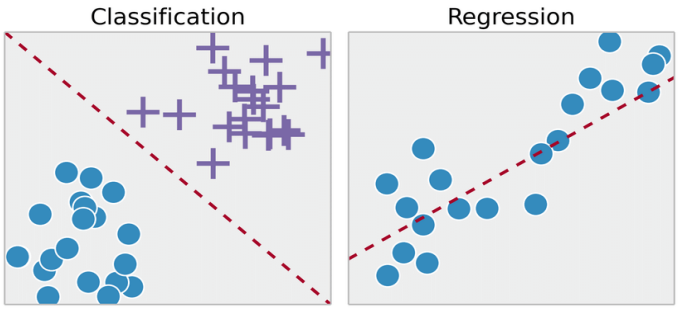


Fig 2.5. Classification vs Regression.

The second approach to apply a Machine Learning algorithm is with Unsupervised Learning, in this one we don't know the output values so the algorithm will only use the inputs and try to find

patterns on those. This type of learning is often named as *Knowledge Discovery* since we don't know *A Priori* what kind of pattern are present in the data and also we cannot measure the error between the predictions and the real results (as we do in the Supervised Learning).

$$D = \{x_i\}_{i=1}^{N}$$

The clustering analysis is the most used technique of unsupervised Learning. Unlike the classification and regression, the clustering analysis runs the data objects with the only purpose of grouping data objects. However, we can say that the clustering analysis is much more similar to classification than to regression since it is also used to categorize data points. The process of clustering will be covered on the chapter 7 but in short we define it as the process that groups data instances into subsets in such a manner that similar instances are grouped together, while different instances belong to different groups (Maimon & Rokach, 2010).

Additionally, to the supervised and unsupervised learning, there is one more type of learning which is called Reinforcement Learning. This type of algorithms is a goal-oriented software's that learn how to maximize a particular dimension. In opposition of Supervised Learning, the Reinforcement Learning does not use pairs of input/output, instead it will find a balance between exploration (what is not known yet) and exploitation (the knowledge already acquired).

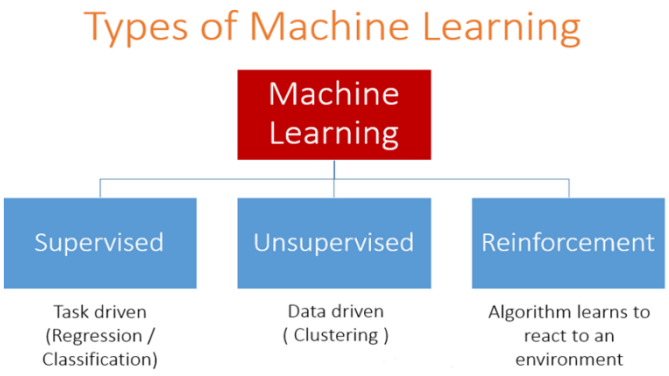


Fig 2.6. Types of Machine Learning.

2.6 THE CYCLE OF TRAINING, TESTING AND EVALUATING

As it was previously mention, an implementation of a Machine Learning is divided in several phases. In this section we will focus on the point 7 of the workflow (fig.2.2) and more precisely in the cycle of training, testing and evaluating the algorithm. After gathering and preparing the data, the data should be split into training set (seen data) and testing set (unseen data). The training set is used for the model to learn from it and based on that improve the ability to make accurate predictions. During the training phase there is only one goal, but usually difficult to achieve, at this phase, all we want is getting a model able of good generalization. This means that the model acquired the ability to make good predictions over a set of input data never seen before. However, without a correct training phase the model will fail to generalize and most probably will fall in the

problems of overfitting or underfitting. A Machine Learning model is said to be overfitted if it learned too much from the training data including irrelevant noise, in that case the model may even suggest it is able to make good predictions due to the good results in the training phase but it is completely unable to generalise with unseen data. Another risk in the training phase is the underfitting where the model did not learn enough from the seen data and consequently is also unable to have good results in unseen data. The perfect situation is somewhere in the middle of underfitting and overfitting where the model learned the essential from the seen data (without unnecessary noise) and is able to perform well with new data that was never seen before.

The unseen data usually belongs to the *test dataset*, that data is only used with the goal of providing unbiased evaluation of the trained model. One of the most famous techniques of testing is called K-Fold Cross validation where the full dataset is divided into k groups of nearly the same size, then one of the groups is used for testing and all the rest k-1 groups are used for training. This process is repeated k times, each time a different group is used for testing and in the end all the results are weighted in average. Obviously, the data split ratio depends on our dataset but as a rule of thumb it is recommended to split 70%-30% for the training and testing datasets.

Fig 2.7 K-Fold Cross Validation.

Additionally, in some cases a third dataset split is used called Validation dataset. The purpose of that dataset is a mix of the previous two, basically the validation data is unseen data that is used during the process of training. The validation data allows the model to optimize even more the accuracy and change the hyperparameters to achieve the best results.

Finally, the evaluation of the model is the moment when we realise how good our model is performing. Building a model is not a straight process of sequential steps but it is usually a cycle with several loops. In other words, it is a process of constant feedback and improvements. During the implementation of the algorithm we may have an idea how well the model is performing. However, it is in the evaluation phase where we obtain the most reliable feedback. If the results in that feedback are not good enough, we have to improve the model and restart the cycle of training, testing and evaluation until reaching the desirable performance. A proper evaluation requires the right metrics for each model, depending on the task of the algorithm we have different metrics. Let's start with the classification metrics (Hossin & Sulaiman, 2015).

Classification Accuracy: Probably the most common metric used in classification, it is the ratio between the correct predictions and all the predictions:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

The adoption of this metric is not recommended when dealing with imbalanced classes. Additionally, the accuracy gives us an overview of how the model is performing but it should be used alongside with other metrics to get a more detailed perspective.

Confusion Matrix: it is an intuitive and widely adopted metric. As the same suggest it describes a tabular visualization comparing the observed values with the predicted values (and how many were correctly classified and how many got the wrong class). In other words, this metric provides a more complete perspective comparing to only observing the accuracy. The Confusion Matrix is composed by 4 groups:

True Positives: for the number of positive correct predictions.
True Negatives: for the number of negative correct predictions.
False Positives: for the number of positive incorrect predictions.
False Negatives: for the number of negative incorrect predictions.

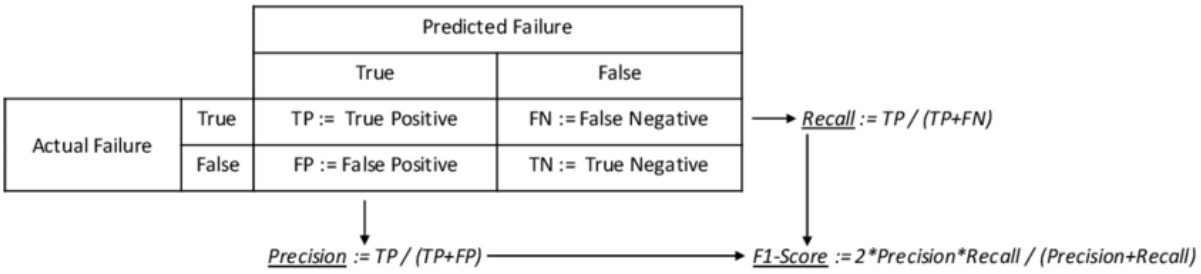


Fig 2.8. Confusion Matrix.

As showed above, based on the Confusion Matrix we can calculate 3 more interesting metrics:

Precision: Ratio between the correctly positive predictions and the total of positive predictions.
Recall or Sensitivity: Ratio between the correctly positive predictions and all the observations that are in fact positive.
F1-score: this metric uses both, Precision and Recall. It is a good alternative for the accuracy, especially when we are seeking for a metric able to make an overall evaluation of the model and at the same time penalize the result if the Precision and the Recall are not balanced.

Area Under the Curve (AUC): The AUC metric is calculated with the support of the ROC curve, so it is important to fully understand the ROC curve at the first place. The ROC curve shows graphically the True Positive rate (TPR) against the False Positive Rate (FPR). It gives us an idea of the probability of picking one random observation correctly classified as TP. The AUC uses the area under the ROC curve as a measure, it is a metric that evaluates how much the model is able to distinguish the two classes. The AUC metric assumes values between [0,1], where 0 is worst result where all the positive cases are wrongly classified, 0.5 corresponds to a random classifier and 1 is the best result possible (graphically is the case where the area under the curves fills all the plot and the ROC curve makes a 90º angle in the on the top left).

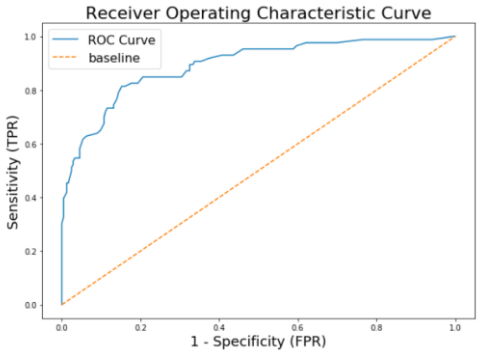


Fig 2.9. AUC-ROC Curve.

All the metrics presented so far are focus on evaluating Classification problems. For Regression algorithms it is necessary to use different types of evaluation since we are no longer working with classes as an output. The concept of regression will be explained in chapter 5 but for now and before explaining the metrics, it is important to have an idea of how the regression works and what is the main purpose of it. With that said, in any regression we use one or more independent variables x's to predict a dependent variable y. Moreover, because the huge majority of the regressions are supervised algorithms (such as the Linear Regression), the main goal is always to minimize as much as possible the difference between the predicted values and the observed values (that difference is called residual or error). It is not surprising that most of the regression metrics focus their attention in measuring the difference between the prediction and the observed value. There are however several approaches to calculate that difference (Kassambara, 2018):

Mean Squared Error: (MSE): Let us start with one of the most used metrics to evaluate the performance of a regression model. The MSE is nothing more than the simple average of the errors, meaning it is the sum of all the errors divided by the number of observations n. This metric requires some caution since the errors are squared, and it may lead to an over-estimation since it will give more importance to the bigger differences.

$$MSE = \frac{1}{n}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE): The RMSE may be a good alternative for the MSE, especially if we wish to work with a more interpretable metric. Contrary to the MSE that is working with squared units in relation to the target variable, the RMSE applies a square root over the squared errors which means this metric works in the same scale of the target variable and consequently is more interpretable

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{n}}$$

Mean Absolute Error (MAE): This metric is quite straightforward, it simple calculates the absolute errors in average. This means that contrary to MSE and RMSE, the MAE is a linear score where all the differences between the predicted values and observed values (errors) will have the same weight, which is the same of saying that MAE does not penalize the larger errors so severely (particularly in the presence of outliers). Following the MAE formula bellow, it is quite intuitive that this metric will always be lower than MSE and RMSE:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\, y_i - \hat{y}_i \,|$$

The previous 3 metrics are the most used metrics in terms of errors calculation but there are a few more that derive from these 3. Some of them will be used in the Time Series Algorithms (chapter 9) such as MAPE (Mean Absolute Percentage Error) or MASE (Mean Absolute Scaled Error).

The Coefficient of Determination (R^2) Is a key output on the regression results, it allows to measure how accurate our model is predicting the dependent variable. This coefficient shows the proportion of variance in the dependent variable that is explained by the independent variable (Rencher & Schaalje, 2008) and is given by:

$$R^2 = \frac{SRR}{SST} = 1 - \frac{SSE}{SST}$$

In a Linear Regression it is pretended to find the best fitting line that represents as much as possible the relationship between the dependent and the independent variables. The SSR denotes the Regression Sum Squares that measure the total distance between the estimated line and a hypothetic horizontal line that would represent the case of no relationship between the variables. Additionally, the Error Sum of Squares (SSE) quantifies how much the data points vary from the estimated regression line. Finally, the Total Sum of Squares (SST) is the sum of the last two measures and it calculates how much the data points vary around their mean.

$$SSR = \sum_{i=1}^{n} (\hat{y}_i - y)^2 \qquad SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \qquad SST = SSR + SSE = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

Adjusted R^2: For the simple Linear Regression, the R^2 is undoubtedly one of the best metrics to evaluate the model and specially to measure the influence of a predictor over a target variable. However, the standard R-squared do not work well in the Multiple Linear Regression since as we add more features to the model, the standard R-Squared will never decrease, even if the features are completely irrelevant to the model. In those cases, it is recommendable to use the Adjusted R^2 that also follows in the idea of the explanatory power of the predictors over the target variable, but unlike the standard R-Squared, it will penalize the result for adding irrelevant variables to predict the target. Below it is presented the mathematic expression to reach the value of Adjusted R^2 as well as an example where the ideal number of independent variables is 3.

$$Adjusted\ R^2 = 1 - (1 - R^2)\frac{n-1}{n-m-1}$$

Vars	R-Sq	R-Sq(adj)
1	72.1	71.0
2	85.9	84.8
3	87.4	85.9
4	89.1	82.3
5	89.9	80.7

Where n is the number of data points and m represents the number of independent variables. It is quite intuitive to realise that as the number of variables increase the result of the Adjusted R^2 tend to decrease.

Regarding the clustering evaluation, only a few mentions will be given in the chapter 7 which will be based on the scikit-learn documentation.

3. ITERATIVE LOCAL SEARCH ALGORITHMS

Iterative Local Search (ILS) is the first family of algorithms we are going to cover in this thesis. This type of algorithms are particularly useful to solve complex optimization problems where the path to the goal is irrelevant and where the goal state itself as the solution (Grosan & Abraham, 2011). The Local Search algorithms are metaheuristic based that start in a random initial position an interactively move to another position by exploring the neighborhood. ILS fits in the idea that sometimes *less is more*, in this case, those algorithms allow to solve highly complex optimization problems in a small amount of time and using little computational memory. However, it is not guaranteed that the final solution will be the global optimum (best solution possible) and most likely the algorithm will fall in the local optimum trap (where the algorithm returns the solution that is the best of its neighbourhood but it is not the best one among all the possible solutions).

3.1 HILL CLIMBING

The Hill Climbing algorithm is one of the most known algorithms in the family of local search. This technique starts in a sub-optimal solution and attempts to improve by incremental changes of the current solution until no further improvements are possible. When that happens the algorithm found an optimal solution (local or global) and it will return that solution as the final one (Kumar et al., 2020).

The name of the algorithm comes from the idea of starting in the base of a hill where we just have a sub-optimal solution and then we start walking up the hill improving the solution until reaching the top where no improvement can be made. The following chart shows the *State Space Diagram* that represents the set of states the algorithm can reach in comparison with our objective function.

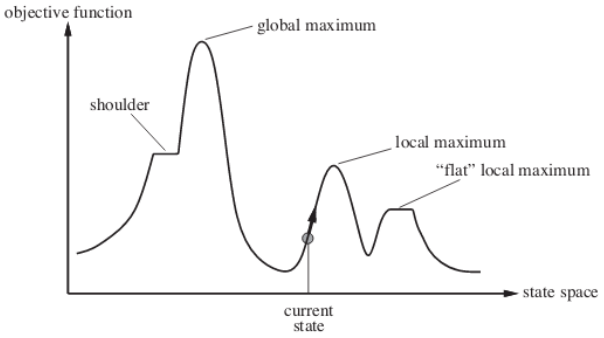


Fig 3.1. State space landscape for hill climbing optimization.

As shown in the diagram, different states can be reached during the optimization process:

1. **Local Maximum:** It's a state which is better than its neighbouring state but there is a state with a better solution. It can happen that the neighbours of a local maximum have the same value, in that situation it is called **Flat Local Maximum/Plateua.**

2. **Global Maximum:** The objective function reaches its maximum and this state becomes the best possible solution.

3. **Shoulder**: It's a plateau with an uphill edge.

4. **Current State:** Represents the current solution in the state space diagram.

The Hill Climbing Algorithm is very efficient in the memory point of view and its often used to solve computationally hard problems since it only looks for the current solution and the immediate future state. However, the fact of the optimal solution is found by making progress towards an interactive process just in the neighbouring may imply that the optimal maximum is local and not global, this means that the final solution is the best in its neighbour but there is at least one better solution in the state space.

There are, however, some ways to avoid this problem (or at least minimize it). One of them is the **Random Restart** that conducts several searches and each time it starts with a randomly generated initial state. In the end, the algorithm compares all the optimal solutions of each run and picks the best one for the final optimal solution (once again, this technique doesn't guarantee that we end with a global optimum).

Another solution is the Simulated Annealing which will be explain in the next chapter.

3.1.1 MODEL OVERALL – HILL CLIMBING

Used for: Optimization problems.

Advantages:

1) It is a helpful algorithm for solving pure optimization problems where the goal passes by finding the best state according to an objective function.
2) Large diversity of application such as job scheduling, portfolio management, vehicle routing, automatic programming and circuit design.
3) Able to perform better results than other local search algorithms while using less computational power. It is ideal for solving complex optimization problems with limited computational resources.
4) It is relatively easy to code.

Disadvantages:

1) It does not guarantee that the final solution will be a global optimum and most likely the algorithm returns a local optimum (meaning the solution is the best of its neighbourhood but it is not the best of all possible solutions).

Evaluation Metrics:

1) Convergence results and comparison to other local search algorithms.

3.2 SIMULATED ANNEALING

The Simulated Annealing (SA) (Kirkpatrick et al., 1983) is a probabilistic technique that attempts to find a global optimum of a function f(x). It is an improvement from the Hill Climbing algorithm and can be particularly useful in finding the global optimum in the presence of a large number of local optimums. This algorithm is usually applied when it is more important to find the global optimum (no matter how long it takes) than find a local optimum as soon as possible (in that situation Hill Climbing fits better).

The name of the algorithm was originally inspired by the process of annealing in metallurgy, a technique involving heating and cooling a material to alter its physical structure removing crystal defects.

The algorithm is similar to the Hill Climbing but instead of picking the best move, it picks a random move, if that move is an improvement from the past solution then it will always move for it. Otherwise the algorithm accepts to make the move anyway according to a certain probability (pr<1). That probability is calculated based on an exponential function representing how bad that move is in terms of cost of the current solution. So, a "bad" move from A to B (f(B)<f(A) is accepted with a probability equals to:

$$P(move_{A\to B}) = e^{(f(B)-f(A))/T}$$

Note that a higher T means that is more likely to accept a "bad" move. Also, as the T tends to zero, the probability also tends to zero and the SA becomes more like a Hill Climbing Algorithm. In a typical case of SA, T starts high and gradually decrease according to the programmed schedule (Marcus, 2015):

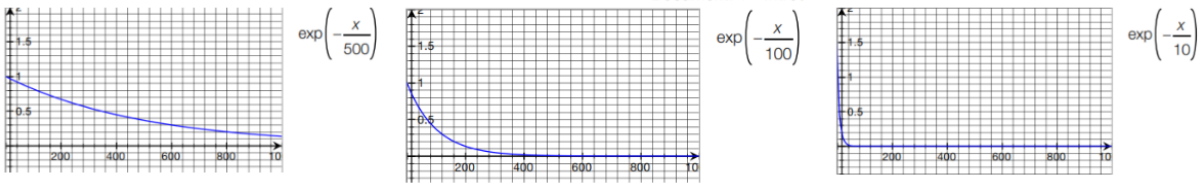


Fig 3.2. Probabilistic curves for different T.

The final solution of the SA is given by the *Theorem of Asymptotic Convergence.* This Theorem is based on the *Markov Chains* and it ensures that the algorithm will stabilize in a solution A (Delahaye et al., 2018).

Theorem: After a sufficient number of transitions with fixed T and using the probability of accepting the move $P(move_{A\to B})$, the algorithm will find a solution $i \in S$ with the probability:

$$P_T\{X = i\} = q_i(T) = \frac{1}{N(T)} e^{-\frac{f(i)}{T}}, \text{ where } N(T) = \sum_{j\in S}^{n} e^{(-\frac{f(i)}{T})}$$

And because T is tending to zero (without never reach it), we end up with the following corollary:

$$\lim_{T\to 0^+} P_T\{X = i\} = \lim_{T\to 0^+} \frac{1}{|S_{opt}|} K(S_{opt})(i)$$

Where s_{opt} is the number of global optima and k represents the characteristic function.

Based on the previous theorem, we can make some logic deductions:

If S_{qt} = 1, there is only 1 global optimum.

- The probability of stabilizing in the only global optimum is 1 and the probability of stabilizing in any other solution is 0 (for $\lim_{t\to\infty}$)

If S_{qt} = 2, there are 2 global optimums.

- The probability of stabilizing in a global optimum is 1/2 and the probability of stabilizing in any other solution is 0 (for $\lim_{t\to\infty}$)

If S_{qt} = N, there are N global optimums.

- The probability of stabilizing in a global optimum is 1/N and the probability of stabilizing in any other solution is 0 (for $\lim_{t\to\infty}$)

3.2.1 MODEL OVERALL – SIMULATED ANNEALING

Used for: Optimization problems.

Advantages:

1) Relatively easy to code and very versatile in terms of solutions for the user.
2) Better chances of finding a global optimum than the Hill Climbing. It generally always returns a good solution.
3) It is a robust algorithm and generally it can deal with arbitrary nonlinear models or noisy data.

Disadvantages:

1) Time-expensive (the quality of the solution and the time required to compute are highly correlated). There is a clear trade-off between the quality of the final solution and the running time of the algorithm.
2) Despite the better chances of finding a global optimum, the SA algorithm does not guarantee that it will necessarily happen.

Evaluation Metrics:

1) Convergence results and comparison to other local search algorithms.

4. BAYESIAN ALGORITHMS

Following the same guideline of the previous algorithms, the Bayesian algorithms fall in the idea that sometimes the simplest algorithms are the most efficient ones. In reality, Bayesian algorithms belong to the family of probabilistic algorithms and are based in the Bayes' theorem (from Thomas Bayes in 1701). It calculates the probability of a certain event happen with *À Priori* knowledge (which mathematically is expressed with a conditional probability). In this chapter we will be working with the Naïve Bayes classifier and its own variations, as said before those classifiers may be very useful in some circumstances but some caution is required before applying them since they have a strong assumption of independence between the predictors.

4.1 PROBABILISTIC NOTATION

Before going deeper in the Bayesian algorithms, it is important to remember some statistics and probabilistic concepts essential to understand the algorithms. Let us start by defining the concept of *Random Event* which is broadly known as an event that cannot be predicted before it is observed but it occurs with a certain probability. By probability, we generally refer to how likely an event may happen if we consider all the possible outcomes (Privitera, 2015). The total number of possible outcomes is called sample space and represented by Ω (e.g rolling a die has 6 possible outcomes which implies a sample space of Ω={1,2,3,4,5,6}). Consider now that we want to calculate the probability of an expected event (let us assume getting 5 in a die), logically all we have to do is dividing the number of favourable outcomes by all the possible events, P(E) = E/Ω = 1/6. Note that a probability is always a value between 0 and 1.

In some cases we are working with more than 1 event at the same time, in those circumstances we have to consider the operations between the events:

- **Intersection** (A∩B): results only in the values present in both events at the same time.
- **Union** (A∪B)**:** results in all the values of all the events.
- **Complement** A^c**:** results in all the values that are not contained in a specific event.

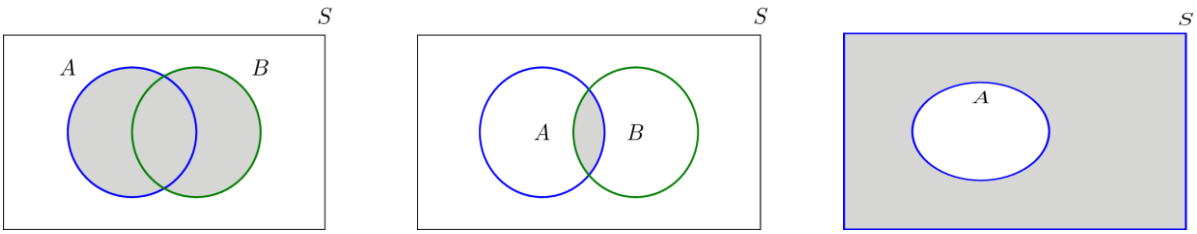


Fig 4.1. Events operation (from left to right: Union, Intersection, Complement).

Furthermore, two events are called independent if the occurrence of the one event does not affect the probability of the other one, P(A|B)=P(A). Note that P(A|B) stands for a conditional

probability of the event A occur knowing that the event B has already happened. It is a probability with *À Priori* knowledge and it can be calculated with the following expression:

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Finally, the Bayes theorem is a derivation of the previous formula and follows the same axioms of the conditional probability, it is however more versatile and can be used in a wide range of applications, especially in classification problems using Machine Learning. Bayes Theorem relies in using prior probability in order to obtain posterior probabilities. By using the Bayes theorem, we are able to discover the result of a conditional probability without using joint probability. This is achieved by assuming that the predictors are independent between each other and also by using reverse conditional probability as showed below:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

4.2 NAIVE BAYES

Naive Bayes classifier is a Machine Learning algorithm based on the Bayes' theorem that can be applied in a wide range of classification problems. Despite its simplicity, the algorithm is often surprisingly useful and sometimes able of outperform more complex algorithms, it can be programmed easily and give quick predictions. In addition, the algorithm is easily scalable and useful in high dimension problems (something that even more sophisticated models struggle to achieve). To understand how Naïve Bayes operates, let us assume we are dealing with a supervised learning problem where we want to estimate an unknown target function f : X → Y, which is the same exact thing of writing P(Y|X). Contrary to the basic formula of Bayes' Theorem, here we will have a random variable $Y=y_j$ and multiple attributes that are represented in a vector $\mathbf{X}=(x_1,x_2,\ldots,x_n)$, where n denotes the number of attributes (Berrar, 2019). By simple applying the Bayes rule to our case we have:

$$P(y_j \mid x_i) = \frac{P(x_i \mid y_j)P(y_j)}{P(x_i)}$$

Where:

- P(Y|X) is the posterior probability of the target variable Y given the attributes X.
- P(X|Y) is the probability of the attributes X given the target variable Y.
- P(X) is called Evidence that stands out for the probability of the attributes X.
- P(Y) is the prior probability of the target variable Y.

The previous equation can also be written like → $Posterior = \frac{(\text{Likelihood} \times \text{Prior})}{Evidence}$

As we know Naïve Bayes has a strong assumption that the features are independent between each other. In fact, in real world that assumption violates all the practical applications since it is pretty rare to find a problem with complete independence among the variables. The Naïve Bayes relies on that assumption and regardless of its scientific foundations, it allows us to rewrite the numerator of the previous equations like this:

$$P(x_i \mid y_j) \times P(y_j) = P(x_1 \mid y_j)P(x_2 \mid y_j)...P(x_n \mid y_j) \times P(y_j) = \prod_{i=1}^{n} P(x_i \mid y_j) \times P(y_j)$$

By replacing the numerator in the main equation, we have:

$$P(y_j \mid x_i) = \frac{\prod_{i=1}^{n} P(x_i \mid y_j) \times P(y_j)}{P(x_i)}$$

When we replace the variables by the values of the dataset, we quickly realise that the denominator does not change with the data entries. Since the denominator is always static it can be removed. Furthermore, the ultimate goal is to find the *Maximum a Posteriori* (MAP) which is directly proportional to the product of the *Likelihood* with the *Prior*.

$$\hat{y} = \arg\max_y \prod_{i=1}^{n} P(x_n \mid y_j)P(y_j)$$

The *Maximum a Posteriori* is the instrument the Naïve Bayes uses for classification, for example, assuming we have 2 possible outputs (A and B), for any new attribute x_i the algorithm calculates the probability of x_i to be classified as A or B, then it will simply pick the higher one.

Depending on the attribute type we can use 3 approaches of Naïve Bayes (Murphy, 2012). If we are dealing with continuous data it is better to use the Gaussian approach and if our data is discrete then the multinomial or Bernoulli will be the best option.

4.2.1 GAUSSIAN NAÏVE BAYES

The Gaussian Naïve Bayes is a variation from the original Naïve Bayes and it is particularly useful when are working with continuous data, in that case we assume that the data follows a certain distribution. In those cases, the gaussian distribution is the most used one and also the easiest one to implement since it only requires calculating μ_i and σ_i^2 (the mean and variance of class i). The Gaussian Distribution is also called Normal Distribution and has a bell shape as it is represented on the right

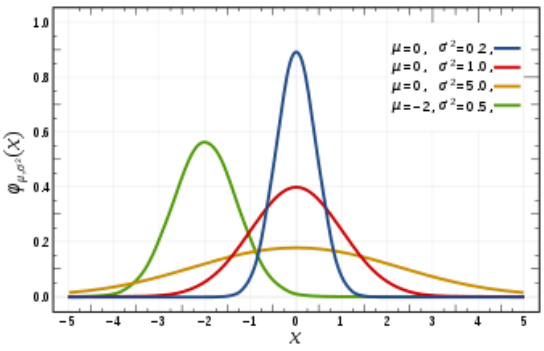


Fig 4.2. Gaussian/Normal Distribution with different μ and σ^2 .

Since we are assuming that the data now follows a Normal Distribution, some adjustments are necessary on the original Naïve Bayes Formula:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp(-\frac{(x_i - \mu_y)}{2\sigma_y^2})$$

4.2.2 MULTINOMIAL NAIVE BAYES

Multinomial Naïve Bayes is a variation from the original Naïve Bayes that fits perfectly for discrete variables. It is broadly used for document classification, particularly to detect the frequency of a word in a document. Furthermore, with a proper pre-processing, the Multinomial Naïve Bayes becomes a very competitive model when compared with more sophisticated algorithms as SVM. The algorithm starts by calculating how often a certain event happen ($x_1, x_2, \ldots, x_n$) and where each event is associated with a frequency of how many times it was observed. After counting how many times each event happens, the algorithm calculates the probability of the event to occur. The conditional probability in this case is given by:

$$p(x \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki}^{x_i}$$

4.2.3 BERNOULLI NAÏVE BAYES

The Bernoulli Naïve Bayes is a special case of the Multinomial but in this case the predictors assume the form of a Binary/Boolean variables meaning they can only assume one of two possible values (for example if a word occurs in a text or not). In other words, and comparing to the Multimodal Naïve Bayes, in this case we give primacy to occurrence of an event rather than counting its frequency. Naturally, this change implies some adjustments in the conditional probability:

$$p(x \mid C_k) = \prod_{i=1}^{n} p_{ki}^{x_i} (1 - p_{ki})^{(1 - x_i)}$$

4.3 MODEL OVERALL – NAÏVE BAYES

Used for: Classification problems.

Advantages:

1) It has strong mathematical foundations and it is relatively stable for small-scale data.
2) Fast and easy to implement.
3) Highly scalable.
4) Handles discrete and continuous data.
5) Extremely useful in text mining (particularly in sentiment analysis, spam filtering and recommendation systems).

Disadvantages:

1) Strong assumption of the independent predictors which is almost impossible to achieve in the real-world problems.
2) It treats all the attributes equally (does not allow to give more weight to the most important ones).
3) Faces the zero probability problem, meaning if something was not seen in the training phase it will assign 0 probability.

Evaluation Metrics:

1) Accuracy, Confusion Matrix, Precision, Recall, F1-Measure and AUC-ROC curve.

5. REGRESSION ALGORITHMS

Regression analysis is one of the most used statistics techniques, in its simplest form it allows to draw inferences about the relationship between variables, a dependent variable and at least one independent variable (Montgomey et al., 2012). It helps to understand how much the dependent variable changes with the variation of independent variables (with *ceteris paribus*). Moreover, the regression analysis also allows to compare the importance of the independent variables over the dependent variables as well as to make predictions. Among the Machine Learning models, the regression models are usually the first ones to investigate the relationship between variables and attempt to make some predictions due to their simplicity of adoption and accuracy. We will be covering the Linear Regression models that aims to find a linear relationship between the dependent variable and one or more independent variables (depending if it is a Simple or multiple regression). In the end of the chapter, the Logistic Regression will also be explained.

5.1 SIMPLE LINEAR REGRESSION

The Simple Linear Regression is a statistical method that allow us to study and analyse the relationship between two continuous (quantitative) variables. The first variable is known as the predictor or independent variable (x) and the other variable is usually named as the response or dependent variable (y). Basically, in this model its pretended to find the best fitting line that explains the relationship between those two variables. To achieve that the Linear Regression uses the traditional slope-intercept technique as showed below. In the learning phase the algorithm will adjust the parameters (β's) to produce the most accurate fitting line and consequently the predictions of y.

The equation for the best fitting line is given by:

$$y = \beta_0 + \beta_1 x + e$$

Where y denotes the target/dependent variable, x is the predictor/independent variable, the parameters β_0 and β_1 stand for the interception and slope of the fitting line, the constant "*e*" represent the cumulative errors between observed values and the predicted values (Rencher & Schaalje, 2008). The smaller the error, the better will be the fitting line and consequently the better will be the predictions. The ultimate goal of any linear regression will be nothing else than trying to minimize as much as possible the residual error given by the formula below:

$$e = \hat{y} - y$$

Before going deeper in how we can estimate the parameters values, we can take some conclusion about how we should interpret β_0 and β_1. It is quite intuitive that the fitting line will have a positive trend (slope) if $\beta_1 > 0$. Otherwise, if the $\beta_1 < 0$, the slope will be negative and for $\beta_1 = 0$ there is no slope. We can also extract an interpretation of the parameter β_0. For example, β_0 is the predicted response when we set x=0. Additionally, if β_0 is not included in the model, it is possible to deduce that the regression line is forced to pass over the origin.

5.1.1 LEAST SQUARES ESTIMATION

The parameters β_o and β_1 are unknown and must be estimated in order to find the line that best fits the sample data. The most common method to achieve that is called *Ordinary Least Squares*, in this method the algorithm tries to find the best fitting line for the observed data by trying to minimize the differences between the real values and predicted values. Mathematically, for a given sample of data we want to find the best combination of the parameters β_0 and β_1 that are able to minimize the Error Sum of Squares (SSE):

$$SSE(\beta_0, \beta)_1 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} (y_i - (\beta_0 + \beta_1 x_i))^2$$

The previous expression works as a baseline to build a cost function to measure how well the model is performing. Usually, the most common cost function is called MSE which is a derivation from the SSE but instead of summing all the errors, it will measure how far the predicted values are from the observed values in average. The most efficient way to minimize the value of MSE is through the gradient descendent method which is an optimization function that interactively tries to find the minimum value. The gradient descent will usually initialize with random values for the parameters and interactively change them in order to minimize the cost function (Ruder, 2017), once it is achieved the Linear Regression is fully optimized.

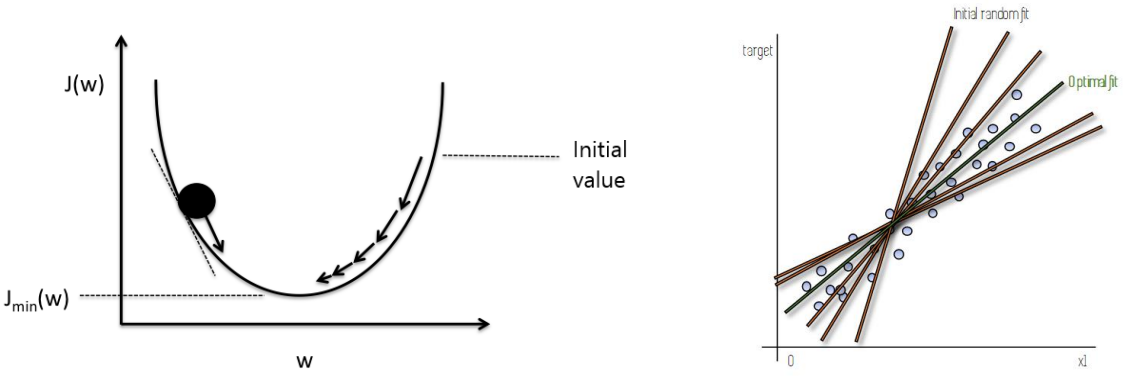


Fig 5.1. Optimization of the regression line by using the gradient descendent.

Usually this process is automatically made but it can also by manually solved by calculating the derivative of the MSE and setting it to zero. After some calculus and deductions, we reach the values of β_0 and β_1, given by:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(x_i - \bar{x})^2} = \frac{SS_{xy}}{SS_{xx}} \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} = \frac{\sum_{i=1}^{n} y_i}{n} - \beta_1 \frac{\sum_{i=1}^{n} x_i}{n}$$

5.1.2 MODEL OVERALL – SIMPLE LINEAR REGRESSION

Used for: Regression problems.

Advantages:

1) Easy implementation with satisfactory results.
2) Low demand of computational memory and relatively short running time compared to others machine learning algorithm.
3) The model is based on a simple equation that is fairly easy to understand. The results are also very interpretable.
4) High performance in linear data.

Disadvantages

1) High Sensitivity to noise data and outliers that may lead to poor results.
2) Unrealistic assumptions (explained in chapter 5.3).
3) Unable to make predictions with two or more predictors (solved in chapter 5.2).
4) It tends to fall in overfitting during the training phase. This situation can be avoided with a good pre-processing of the data, particularly with *Dimension Reduction.*

Evaluation Metrics:

1) MSE, RMSE, MAE and R-Squared.

5.2. MULTIPLE LINEAR REGRESSION

The Simple Linear Regression is definitely a useful predictive technique, but it has significative weaknesses. The most evident of all is undoubtedly the fact of it can only deal with datasets with only one independent variable. The Multiple Linear Regression may not solve all the limitations of the Simple Linear Regression, but it solves the problem mention before by allowing a regression to work with multiple independent variables. Everything that we covered so far remains valid and the Multiple Linear Regression must be seen as an extension of the Simple Linear Regression that follows the same principles. With that said, the main goal is still to find the best relationship between the dependent and independent variables, following the expression below:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k + e$$

The formula has the same structure as the one of the Simple Linear Regression, where y denotes the dependent variable, $X=\{x_1, x_2,...,x_n\}$ is a vector composed by all the independent variables, the β's are the parameters responsible to adjust the weight of each variable (with the exception of β_0 that does not have any variable associated and just reflects the interception of the y-axis). Finally, the parameter "e" is the sum of all the errors between the predicted values in relation to the observed values. The previous formula can also be written in a matrix form, for all the observations n and for all the independent variables k, we have:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1k} \\ 1 & x_{21} & x_{22} & \cdots & x_{2k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nk} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}$$

That can also be summarized in a more compact format **Y** = **Xβ** + **e**. The main goal of the model is still to estimate the regression parameter **β** which is able of adjusting the weights of each variable in order to produce the best prediction possible and consequently minimize the regression error as much as possible. The optimization method is still the Least Squares, but this time some adjustments are required since we now have several independent variables. The fundamental principle remains the same, we want to find the regression parameter that minimizes the criterion:

$$H(\boldsymbol{\beta}) = (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) = \sum_{i=1}^{n} (y_i - x_i\boldsymbol{\beta})^2$$

By calculating the derivatives in respect of β and setting it to zero, we have:

$$\frac{dH}{d\beta} = -2\mathbf{X}'(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) = 0 \;\Rightarrow\; (\mathbf{X}'\mathbf{X})\boldsymbol{\beta} = \mathbf{X}'\mathbf{Y}$$

After some simple deductions, particularly in calculating the inverse of **X'X**, we finally find the parameters that best fit the data:

$$\hat{\boldsymbol{\beta}} = \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \vdots \\ \hat{\beta}_k \end{pmatrix} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$$

5.2.1 SIGNIFICANCE TESTING

Additionally to the metrics already covered for the simple regression, when we are working with multiple independent variables it is extremely useful to test the significance of those variables in the model. In other words, before adding a new variable to the model, we may want to test if adding a new variable brings value to the model or if there is a significative correlation between that variable and the dependent variable. Before going deeper on that, it is important to fully understand how the hypothesis test works. A significance-based hypothesis test is a statistical method used for decision making. It is composed by a *null hypothesis* (H_0) that proposes there is no significative difference on

the given observation and by the *alternative hypothesis* that directly contradicts the H_0 (James et al., 2017). The formulation of any hypothesis test follows 4 main steps:

1. State the hypothesis.
2. Set the criteria decision.
3. Compute the test statistic.
4. Make a decision.

The first step is the moment to choose the hypothesis, starting by the null hypothesis that is composed by the assumption we want to test, followed by the alternative hypothesis which is the exact opposite. Then we have to set a criteria of decision, the most common approach is by using the Level of Significance (denoted by α) which is based on a probability of rejecting the null hypothesis. After this process, the test statistics is computed and based on the results a decision will be made.

Coming back to the regression analysis, the hypothesis test is particularly useful as a mechanism of testing the significance of variables in the multiple regression. In fact, there are 3 different levels of testing, depending if we want to test the significance of only one variable, of a subset of variables or an overall evaluation of all variables. In the case we are just interested in testing the significance level of only one variable, it is recommended to use the t-statistics alongside with the p-value. Let's see the following example to test the significance level for the variable x_1:

$$\text{Hypothesis test} = \begin{cases} H_0: \beta_1 = 0 \\ H_1: \beta_1 \neq 0 \end{cases} \qquad t^* = \frac{\text{sample coefficient-hypothezied value}}{\text{standard error of coefficient}} = \frac{\hat{\beta}_1 - 0}{se(\hat{\beta}_1)}$$

To decide whether or not to reject the null hypothesis, it is required to carry out the t-statistics test (showed above) and to discover the p-value associated to the result of the test. The p-value of the test is then compared to the level of significance (usually α=0.05) to check whether the predictor is statistically significant for the model. A p-value lower than α will lead us to the conclusion that there is enough evidence to reject the null hypothesis (in our example means that the variable is statistically significative). In other hand, if the p-value is greater than the significance level than there isn't enough evidence to reject the null hypothesis and the choice must follow the hypothesis that states for the no correlation between the variable (H_0).

The t-statistics becomes useless when we are trying to test the significance of a subset or even of all the variables. In those cases it is recommended to apply the F-test which is able to compare a larger model in relation to a smaller model and based on that decide if it is worth it to keep the larger model or if it is preferable to choose a model with less variables. The larger model is usually called *Full Model* and composed by n features and the smaller one is called Restricted model with m features, where m<n. Let's see an example to make it clear, assuming the regression $y=\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + e$ and the following hypothesis tests:

$$\begin{matrix}\text{Hypothesis test} \\ \text{for a subset slopes}\end{matrix} = \begin{cases} H_0: \beta_2 = \beta_3 = 0 \\ H_1: \beta_2 \neq \beta_3 \neq 0 \end{cases} \qquad \begin{matrix}\text{Hypothesis test} \\ \text{for all the slopes}\end{matrix} = \begin{cases} H_0: \beta_1 = \beta_2 = \beta_3 = 0 \\ H_1: \beta_1 \neq \beta_2 \neq \beta_3 \neq 0 \end{cases}$$

In both cases, the first thing to do is to calculate the Error Sum of Squares of the full and restricted model (SSE_f and SSE_r respectively). Once the SSE's are calculated, we also have to obtain the degrees of freedom of each model, this can simply be obtained with df=n-1 where n is the number of observations. After having the SSE's and the df's of both models we can finally calculate the F-test:

$$F^* = \frac{\frac{SSE_r - SSE_f}{df_r - df_f}}{\frac{SSE_f}{df_f}}$$

As it happens with t-statistics, the F-test will originate a p-value associated with the result of the test and it is based on the p-value that a decision is made. The rule remains the same, a p-value inferior to the Level of significance (usually 0.05) means there is enough evidence to reject the null hypothesis, otherwise the extra variables of the full model are not statistical significant to the model and the null hypothesis should be chosen.

Although the t-statistics and the F-test have different purposes, both of them may give a huge complement to the regression metrics covered in chapter 2. Those statistics tests must be used alongside with the previous metrics and never replace them. It is important to note that the R^2 and specially the R^2-adjusted should be the first metric to look in the multiple regression to clearly evaluate and interpret the performance of a model.

5.2.2 MODEL OVERALL – MULTIPLE LINEAR REGRESSION

Used for: Regression problems.

Advantages:

1) Allows more than one predictor and it is able to measure the relative importance of each predictor in the target variable.
2) The model is based on a simple equation that is fairly easy to understand. The results are also very interpretable.
3) Easy implementation with satisfactory results.
4) High performance in linear data.

Disadvantages

1) High Sensitivity to noise data and outliers that may lead to poor results.
2) Unrealistic assumptions (explained in chapter 5.3).

Evaluation Metrics:

1) MSE, RMSE, MAE, R-Squared, R-Squared Adjusted, T-statistics and F-test.

5.3 LINEAR REGRESSION ASSUMPTIONS

The Linear Regression, whether it has only one predictor or more, has strong assumption about the relationship between the predictors and the target variable (Poole & O'Farrel, 1970). In the moment of considering a linear regression as an algorithm for our data, it is recommendable to check if the data meets all the assumptions, otherwise we may end up with undesired results.

Linearity: As the name of "Linear Regression" suggest, the relationship between the predictors and the target variable must follow a linear relationship. Additionally, the presence of outliers has to be checked since the linear regression is very sensitive to outliers and it may influence the linearity of the model. This assumption can be tested by simply draw a scatter plot with all the data points between the predictors and the target variable, if the data points follow a linear pattern then this assumption is confirmed.

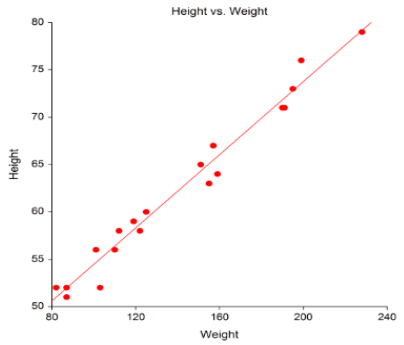


Fig 5.2. Linear Relationship between height and weight.

Little or no multicollinearity between the predictors: The concept of multicollinearity and correlation are highly related to each other. Two variables are said to be correlated if the change in one variable has an impact in the other one. If they are perfect correlated means they contain the same information and adding both variables to the model is redundant. A model with high multicollinearity suggests the presence of unwanted correlated features. By calculating a matrix of correlation, we can detect the existence of multicollinearity.

Fig 5.3. A heatmap as a correlation matrix.

Homoscedasticity: this assumption states for the idea that the errors or the noise in the regression should be similar across all the values of the Linear Regression. Basically, a perfect scenario would be a model with constant variance for the errors without any random disturbance, or at least with little fluctuations. This assumption can also be proven with the support of a scatter plot showing how the errors are distributed in space.

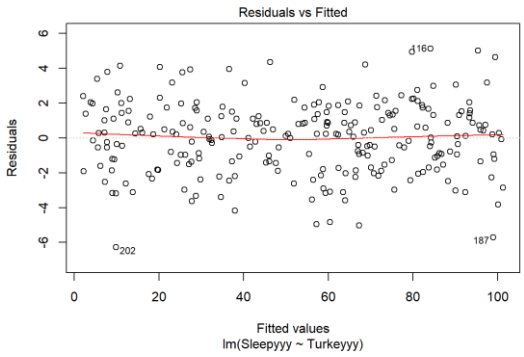


Fig 5.4. Example of the homoscedastic errors.

The errors are normally distributed: Finally, this last assumption ensure that the residuals follow a normal distribution. However, in some cases where the sample size is considerable big, this assumption may not be necessary. This assumption alongside with the previous one, result in having errors iid. N (0, σ^2) meaning the errors are independent and identically distributed following a Normal Distribution with mean 0 and variance σ^2. This assumption can be proven with the support of Q-Q plot where the residuals are displayed in quantiles and should be as symmetric as possible.

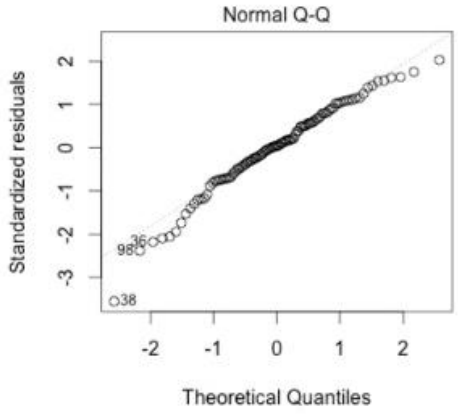


Fig 5.5. Q-Q plot with normal distributed residuals.

5.4 LOGISTIC REGRESSION

So far, we have only covered the estimation of a continuous variable using one or more independent variables. However, sometimes we still want to use this input-output methodology where we predict a variable based in others, but the output variable is this time categorical rather than continuous, which is exactly the purpose of the Logistic Regression. At the first look, the logistic regression looks more like a classification problem than a regression which may cause some confusion since the algorithm is called Logistic Regression. In fact, the Logistic Regression is a Machine Learning classification algorithm, but the classification is made by estimating a probability and that is why it is named as a regression (Burkov, 2019).

Every algorithm performs better under certain conditions and for each problem we have to decide which model suits better and consequently provides the best results. With that said, the Linear Regression is an extremely useful algorithm, but it fails in classification problems. This happens essentially because the Logistic Regression works with probabilities that fall in a range between [0,1] and the range of the Linear Regression is from [-∞, +∞] which represents something completely impossible since a probability can never be negative or superior to one. Moreover, the Linear Regression aims to finds the best fitting line for the relationship between x and y, something that makes no sense in classification as showed in fig. 5.6. Instead of a straight line, the Logistic Regression is represented with a sigmoid function with a shape of an "S" between the boundaries of 0 and 1 in y-axis.

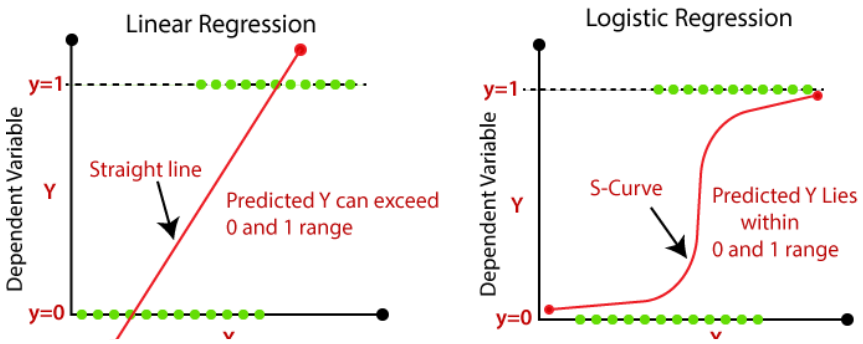


Fig 5.6. On the left, a failed attempt to predict a categorical variable with a Linear Regression. On the right, the sigmoid function used in logistic regression that fits better in the context of classification.

The sigmoid function presented on the right is mathematically expressed by:

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

One of the most interesting characteristics of the sigmoid function is the ability of converting any x-value into a probability between 0 and 1. But even after converting the x-value in a probability, how can we map that probability to a output class (let's assume for now it is a binary output). Well, the answer is quite simple, the first step is to set a decision threshold (also called classification threshold) in which above that value it indicates one class and below it indicates another class. Usually this threshold is defined at 0.5 but of course it depends on the problem, also saying 0.5 or 50% is exactly the same thing since the threshold always represents a probability. The concept of probability was already explained in the chapter 4, but it is important to briefly remember it, by probability we are referring to the likelihood of a certain event occur. To make it clear let us see the example below with a standard sigmoid function:

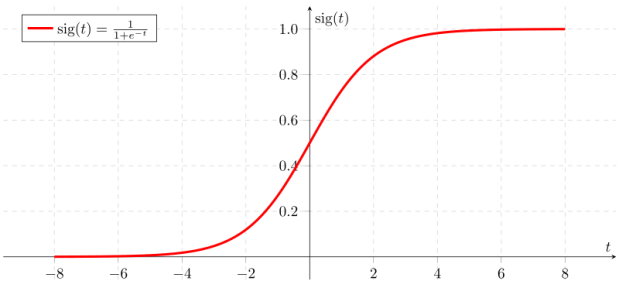


Fig 5.7. Sigmoid Function.

Supposing we have set the threshold in 0.5, where above that value the observation is classified as A, otherwise it gets the class B. It becomes quite intuitive that an observation x_i=2 will have a probability $p(x_i)$>0.5 and consequently it will be classified as A.

5.4.1 THE FUNDAMENTALS OF THE SIGMOID FUNCTION

Of course, the first sigmoid function may not fit the data perfectly and some adjustments may be necessary. Those adjustments are graphically represented by moving the s-curve but to fully understand how that process works, we first have to clear up the fundamentals of the sigmoid function and how it was initially built. The Logistic Regression follows a specific type of binomial distribution called Bernoulli which assumes two possible outputs (A and B), in which n=A represents the successful event and it is associated with a probability P and n=B is the failure event with a probability 1-P (Rodríguez, 2007).

$$f_{Bernoulli} = \begin{cases} P; & for\ n = A \\ 1-P; & for\ n = B \end{cases}$$

Another important concept is the concept of *Odds* which is often misunderstood and wrongly defined as the same of a probability. In fact, odds are a ratio between the probability of an event occur in relation to the probability of the same event do not occur, as follows:

$$odds = \frac{p}{1-p}$$

Where p stands for the probability of an event to happen. But how is the Bernoulli distribution related to the concept of odds? Well, to perform the classification in the Logistic Regression we have to find a function that is able to link the independent variables (that can assume any value from negative to positive infinite) to a categorical dependent variable that follows a Bernoulli distribution. The easiest function that fulfils this requirement is the logarithm of the odds. The logging odds, also called Logit is an unbounded function able to transform any real number into a probability

$$\ln(odds)=\text{loggit}(P)=\ln(\frac{P}{1-P})$$

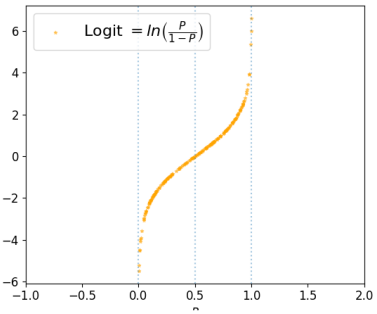


Fig 5.8 Logit Function.

If we take a close look the logit function, we quickly realise that the sigmoid function is nothing else than the inverse of the logit. By incorporating the logit function in the regression expression, we can formally describe the Logistic Regression as:

$$\log\frac{P(X)}{1-P(X)} = \beta_0 + X.\beta$$

And by calculating the inverse:

$$P(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}} = \frac{1}{1+e^{-(\beta_0+\beta_1 X)}}$$

The logistic regression can handle multiple independent variables which implies to have multiple parameters β_n. The parameter β_0 moves the S curve to the left or to the right and all the others β's are responsible for the steepness of the curve.

$$P(X) = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_n x_n)}}$$

This last equation represents the sigmoid function of the model that will be used for the learning phase.

5.4.2 LEARNING PHASE

As any other algorithm, also the Logistic Regression has to perform the learning phase to maximize the accuracy of the model. In this phase, the algorithm initializes the model with random values of the parameters. Each parameter may assume different values, and during the learning phase the algorithm will basically try all the possible combination of parameters until find the best combination possible. It is a process of incremental improvements while modifying the parameters of the model.

For the logistic Regression, the learning phase is all about trying to minimize as much as possible the difference between the estimated values and the observed values. To achieve that we will use the same method of the Linear Regression, the cost function. The idea is then to find a cost function or loss function that is able to measure in each interaction how much ŷ differs from y. There are several cost functions that could be used here, one of the most used was invented by Rubinstein in 1997 and it is called Cross-Entropy or Log-loss. The Cross entropy is a cost function specially design for classification problems where the output is binary and based on a probability, just like the Logistic Regression. Assuming we have two output classes, for y=1 we want minimize as much possible the log-loss and for y=0 we want to maximize it:

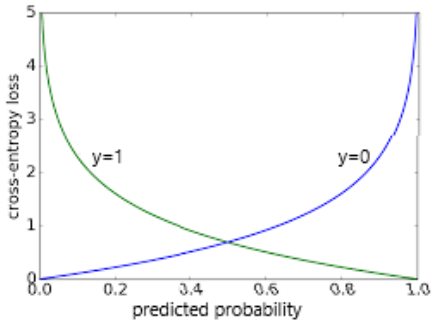


Fig 5.9. Cross Entropy.

Regardless the output class, the rule of thumb is always trying to maximize the likelihood of the model or in other words to guarantee that each observation is assign to the most probable output class. So, contrary to the Linear Regression that uses the Least Squares, the Logistic Regression is based on the Maximum Likelihood Estimation (MLE).

$$L(\beta_0, \beta) = \prod_{i=1}^{n} p(x_i)^{y_i} (1 - p(x_i))^{1-y_i}$$

The instrument to optimize the cost function is however the same, both models use the gradient descendent technique (explained in section 5.2) to find the combination of parameters that minimize the difference between the observed values and the predicted values.

5.4.3 MODEL OVERALL – LOGISTIC REGRESSION

Used for: Classification Problems.

Advantages:

1) Very efficient model able to perform with low computational resources and low time of running.
2) Easy implementation and hypertuning.
3) A difficult to beat algorithm for linear separable problems.
4) It is used many times as a benchmark to be compared to other algorithms.

Disadvantages

1) High reliance on the feature engineering. The model fails to perform well without a proper data pre-processing where we identify the most important independent variables as well as the ones to exclude (high correlated variables or irrelevant variables).
2) Unable to solve non-linear separable problems.
3) Sensitive to outliers and vulnerable of overfitting.

Evaluation Metrics:

1) Accuracy, Confusion Matrix, Precision; Recall, F-Measure and AUC-ROC curve.

6. INSTANCED-BASED ALGORITHMS

Instanced-Based Learning (IBL) is a Machine Learning method that classifies or estimates new examples by comparing them to the ones already seen and in memory (Martin, 1995). This kind of algorithms are particularly useful for a problem that needs to be *locally optimized*. For huge datasets, the Instanced-based algorithms usually fail to generalize and are computational expensive.

The Instanced-Based classifiers are also called as Lazy Learners because each time a classification occurs they have to determine the important instances and create a local model during the classification time. In other words all the effort is made during the classification phase rather than in the learning phase. It becomes intuitive to realise that in Instanced-Based model the separation between training and testing phase is not as clear as it is in other algorithms. Nevertheless, the fundamental principle of classification algorithms remains the same, and IBL assume that similar instances must have similar classifications (Aha et al., 1991).

6.1 K-NEAREST NEIGHBORS (KNN)

The most commonly method of Instanced-Based Algorithms is the nearest neighbour method. The KNN is a supervised machine learning algorithm able to solve classification and regression problems. The origin of the KNN algorithm comes from real life and how people tend to create friendships with people similar to them that share the same interests.

The application of this algorithm is quite straightforward, if it is solving a classification problem, new examples will be classified based on the class of its neighbours. The same happens for regression problems where new examples will be estimated according to the values of their neighbours. In a certain way, KNN follows in the same principle of clustering algorithm, by using distance as a measure of similarity. The KNN is usually called a *Lazy Non-Parametric Model* since it leaves all the data for the classification/regression (without any training phase) and it does not assume anything about the parameters of the model.

For the implementation of the KNN let us assume we are solving a classification problem, the implementation occurs in two different stages, first we have to define the number of k neighbours and then proceed to the classification (Cunningham & Delany, 2007). starting with the first stage, choosing the right number of neighbours (k) is a crucial moment of the algorithm with a direct impact on the results. Unfortunately, KNN does not offer a precise method to find the value of K and most of the times the strategy passes by a trial and error methodology. There are, however, some guidelines that should be followed when we are selecting the k value:

1) A smaller K will be very noisy and consequently lead to poor results.
2) A higher K will have lower variance for each neighbour, but it may end with bias results and it is computational expensive.
3) A good starting point is to define K=sqrt(n), where n represents the number of samples.
4) Another interesting strategy is by using the cross validation as a method to choose k. In this approach we select a validation dataset and we use it to evaluate the results for different values of K on the rest of the dataset. Then we simply choose the best option, which is the one that minimizes the error between the predicted values and the observed values.

5) It is recommendable to choose an odd number of k. Otherwise it may cause some confusion between 2 classes of data.

After the K value is selected, we can move forward to the second stage of the process where the classification of new examples will take place. The classification process in KNN works like an election where the k-most similar instances vote to give their class to the new observation, the winning class will be the one with the majority of votes. To better understand how this process works let us see an example where we have a new observation (in red):

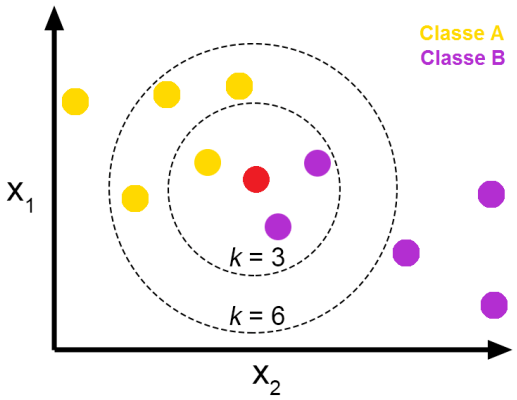


Fig 6.1. KNN Classification, the new observation (in red) will be classified as purple for k=3.

In the example above, we have 2 different classes A and B, painted respectively with yellow and purple. In the case K=3, we have 3 observation already classified inside the circle, 2 observation in purple and 1 in yellow which means the new observation will be classified as purple. For a k=6, it is all a different story, in that case we have inside the big circle 4 yellows and only 2 purples, naturally the new observation will be classified as a yellow. This example makes us realise the importance of selecting the right number of k and how the classification can change completely with little variations of K.

For Regression problems, it all remains the same but instead of classifying the new observation with the mode of the neighbourhood, it will return the mean. In both cases, classification and regression, it is possible to calculate a distance vector to give more weight to the closer observations.

6.1.1 MODEL OVERALL – KNN

Used for: Classification and Regression.

Advantages:

1) Simple Implementation.
2) Robust to noisy data.
3) Very flexible without any prior assumption of the data.
4) Able to solve Classification and Regression problems.
5) Relatively good performance for multiclass classification.
6) Parallel Computing allowed.

Disadvantages

1) Computational expensive with high demand of memory.
2) The prediction time is quite long.
3) Difficulty in choosing the right value of k.

Evaluation Metrics:

1) For Classification: Accuracy, Confusion Matrix, Precision, Recall, F-Measure and AUC-ROC curve.
2) For Regression: MSE, RMSE and MAE.

6.2 SUPPORT-VECTOR MACHINE (SVM)

Identically to the KNN, the SVM is a Machine Learning algorithm that belongs to the family of the Instanced Based Algorithms which is able of solving both classification and regression problems. It is however, mostly used in Classification which will be the main focus in this chapter. The SVM was officially introduced in 1992 by Bernhard Boser and since that time is has been one of the best supervised algorithms (and for many the best, surpassing more complex algorithms as Artificial Neural Network). It became famous due to its robustness, good generalization ability and by providing a unique global optimum solution (Awad & Khanna, 2015). Even so, the SVM is much simpler than an ANN, it solves some of the main weaknesses of the ANN, particularly the overfitting and the multilocal minimum issues. All of this will be explained in detail on the chapter 11, but even in terms of image classification, the SVM may be a good alternative to a CNN (Convolutional Neural Network).

To fully understand how the SVM works, it is important to dominate the concept of vector, hyperplane and linear separable data. Starting by the vector definition, a vector is a quantity that is characterized by having both magnitude and direction, by contrast to scalar that has no direction (Serway & Vuille, 2006) .Geometrically, a vector can be found in space with a directed line segment in a shape of arrow indicating the direction of the vector and the length represents the magnitude. Mathematically, the magnitude of a vector is usually represented with || x || and its direction by an w, both can be calculated as follows:

$$|| x ||= \sqrt{x_1^2 + x_2^2 + ... + x_n^2} \qquad w = (\frac{x_1}{|| x ||}, \frac{x_2}{|| x ||}, ..., \frac{x_n}{|| x ||})$$

This will be used ahead, for now let us focus on the concept of linear separability of the data, the SVM can only perform the classification if the data is linear separable. In fact, even if the original data is not linear separable, there are some transformation we can make in order to achieve that, for example adding one more dimension.

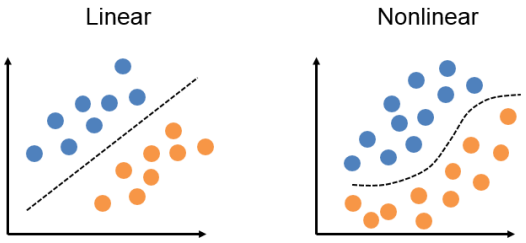


Fig 6.2. Linear and Non-Linear separable data.

The example above represents a two-dimensional data, but of course it is possible to include more dimensions on the analysis. For now, and to make it simple in terms of visualization, we are going to assume only two dimensions. The intuition of the SVM will be then to find the best hyperplane between the two classes, so the margin between the two classes is maximized (Yan, 2016). The margin is calculated by the distance between the separation line and the closest datapoints of each class:

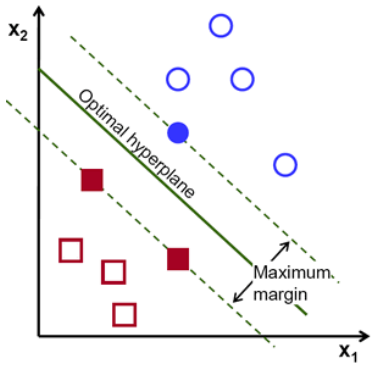


Fig 6.3. Optimal hyperplane that best separates the two classes.

But where do we use the vectors? Well the datapoints that are closer to the hyperplane are called support vectors and are the ones responsible by the position and orientation of the hyperplane. Geometrically, the model is relatively easy to understand, let's focus now on what the algorithm is really doing behind the fig.6.3. We already know that a line function is defined by y=ax+b where *a* is the slope of the line and *b* the interception with the y-axis. Moreover, assuming we have 2 independent variables we will end with $y=a_1x_1+a_2x_2+b$. Finally, if we express $x=(x_1,x_2)$ as well as $w=(a_1,a_2)$, and by setting the equation equal to zero we will get the hyperplane equation:

$$w.x+b=0$$

The classification can then occur based on a test of hypothesis, where it either return 1 or -1 depending on each observation, as presented:

$$h(x_i)\begin{cases}+1\ if\ \ w.x+b>0\\-1\ if\ \ w.x+b<0\end{cases}$$

The example showed above was relatively simple since we have only 2 linear separable variables. However, in real-world problems, most likely we will end up the non-linear situations where it is not possible to draw a hyperplane to separate the two classes. In those cases, it is recommendable to apply the Kernel Trick, where we solve a non-linear problem with a linear algorithm such as SVM. The Kernel works as a function that takes non-linear data as an input and returns linear data that can be easily separable in classes. In the SVM algorithm, this is achieved by adding a kernel function to our SVM expression (Aĭzerman et al., 1964; Boser et al., 1996)

$$w.x + b \rightarrow w.f(x) + b$$

Where f represents the kernel function. There are several types of kernel functions, such as Linear Kernel, Polynomial Kernel, Radial Basis Function Kernel (RBF), Gaussian Kernel, among others. Our job is to choose the right kernel method for our data.

In terms of regression problems, the algorithm is usually called *Support Vector Regression* (SVR) and follows the same idea of finding the best hyperplane. Contrary to the Linear Regression where we want to minimize the error, the SVR aims to set a maximum threshold for the error. That threshold is described by the limits of the hyperplane (in this case -e and e, where *e* is the error we are willing to accept.

6.2.1 MODEL OVERALL – SVM

Used for: Classification and Regression

Advantages:

1) One of the best algorithms for datasets with clear margin of separation between classes.
2) Relatively efficient in terms of memory.
3) Scales relatively well for high dimension data.
4) Good generalization.
5) The kernel trick is one of the main strengths of SVM. With the appropriate kernel it is possible to solve almost all complex problems.

Disadvantages

1) Low performance in the presence of noisy data.
2) It struggles to perform well in overlapped classes.
3) Selecting the right kernel function can be tricky.
4) Hyperparameter tuning is usually a trial and error process.

Evaluation Metrics:

1) For Classification: Accuracy, Confusion Matrix, Precision, Recall, F-Measure and AUC-ROC curve.
2) For Regression: MSE, RMSE and MAE.

7. CLUSTERING ALGORITHMS

Data clustering plays a huge role in the Machine Learning algorithms, especially in the Data Mining process. Data Clustering is one of the most useful and interesting algorithms for grouping data based on patterns. The main goal of a clustering algorithm is to divide a dataset into several groups where the data points of each group are similar between each other (Maimon & Rokach, 2010). A good clustering model is able to create high intra-class similarity and low inter-class similarity.

There are several clustering categories, in this chapter we will cover the main three implementations: Partitioning clustering, Hierarchical clustering, and Density-based clustering. All of them use distance as a measure to distinguish the elements of the dataset, the most common one is the Euclidean Distance but there are other alternatives such as Manhattan or Mahalanobis distance.

7.1 PARTITIONING CLUSTERING

In the partitioning clustering the dataset will be divided into K clusters decided *A Priori*, each cluster is composed by multiple elements that will belong to only one cluster. This type of clustering is a centroid-base which means the clusters are formed by the closeness of the data points to the centroid. This is one of the simplest clustering implementations and at the same time a very efficient one. Inside the Partitioning Clustering, we can find several variations depending on how the centroid is calculated.

7.1.1 K-MEANS

The K-means algorithm is the most common clustering approach where basically the centroids will correspond to the mean of each cluster. The implementation is pretty simple, it starts by selecting a number K of representative points as initial centroids. The centroids are selected randomly and based on those the data points will assume the class of the nearest centroid according to their distance to it. Once the clusters are formed, the centroid for each cluster is update by computing the new mean of each cluster and then the algorithm runs again to create new and more accurate clusters. This is an interactive process that will end when the algorithm converges to a local minimum, meaning the sum of all the distances between the data points and their centroid have reached the minimum value possible (Aggarwal & Reddy, 2014).

The convergence is made through an objective function that usually is the Sum of Squared Errors (SSE). The SSE will compute the sum of the distances between the data points and their centroids in each interaction, the objective is to find a solution where the SSE has the minimum value possible. Mathematically and following the Euclidean Distance procedure, given a dataset $D=\{x_1,x_2,\ldots x_N\}$ and a K number of clusters $C=\{C_1,C_2,\ldots,C_k\}$, the SSE is calculated by:

$$SSE(C) = \sum_{k=1}^{K} \sum_{x_i \in C_k} \| x_i - c_k \|^2$$

7.1.2 K-MEDIANS CLUSTERING

Contrary to the K-means, the k-medians will calculate the median of each cluster to become the centroid. All the rest remains the same and the optimization is still made by minimizing the distance between the data points and their respective centroid, but this time we are going to use the absolute distances instead of the squared distances. The K-medians allow us to get more robust results since it is less sensitive to outliers. Mathematically is express with the following formula:

$$S = \sum_{k=1}^{K} \sum_{x_i \in C_k} | x_{ij} - med_{kj} |$$

Where x_{ij} represents the attribute j in the instance x_i and the med_{kj} is the respective median of the cluster K.

7.1.3 K-MODES CLUSTERING

Until now, we have just work with numerical data that allow us to calculate distances. But what if our dataset is composed by categorical data? The solution passes by using the K-modes which was proposed by Zhexue Huang in 1998. Instead of using distances, it will use dissimilarities to assign the data points in the clusters. For dissimilarities we assume the number of mismatches between two elements, a smaller number of mismatches means that elements are very similar and then with high probability to be assign in the same cluster. The mode is expressed through a vector of elements and the optimization is made by minimize the dissimilarities within each vector. Each node represents a cluster

7.1.4 K-MEDOIDS CLUSTERING

The K-medoid is a variant of k-means clustering that is more robust to noise and outliers since the mean can be very sensitive to outliers. There are a few differences between k-means and k-medoids, first of all, while the k-means selects a hypothetic point that represents the centre of the cluster, the k-medoid will choose a real data point as the centre of the clusters. Another difference is the objective function used in both cases, while in the K-means the algorithm is trying to minimize the SSE, the k-medoid is looking for the medoids that are the most centrally located data points in the cluster and that minimize the intra-variance of the cluster. In another point of view, the way the algorithm runs is quite similar between each other, also in k-medoid the algorithm runs interactively by adjusting the representative of the cluster.

The partitioning Around Medoids (PAM) is one of the famous algorithms in the K-medoid clustering. This algorithm minimizes the objective function by swapping the non-medoids and medoids interactively in order to find convergence. However, it does not scale well for larger datasets, in that case it is better to use the CLARA (Clustering Large Applications) which is a variation of PAM that uses a sample of the dataset at a time. CLARA considers several samples and applies the PAM in all of them in order to find the medoids in each case. In the end the CLARA algorithm combines everything and it returns the optimal set of medoids. Both algorithms, PAM and CLARA, were developed by Kaufman and Rousseeuw in the book *Finding Groups of Data* in 1990.

The performance of all these implementations may be affected by two variables:

1) Having a good initialization method in order to choose the initial centroids. There are several interesting alternatives to the random initialization such as the density neighbour density suggesting that datapoint surrounded by many other data points could be a good initial cluster representative.
2) Estimating the correct number of K (number of centroids). Selecting the correct number of clusters is a fundamental step to obtain good results in any clustering algorithm. Among all the techniques, the Elbow Method is widely used specially in the k-means and its variations. The Elbow Method recalls the idea of minimizing the intra-clustering variation. That variation is then represented in a chart as a function of the number of clusters, the perfect number of cluster should be the one when adding one more cluster it won't make huge impact. Let's see an example below:

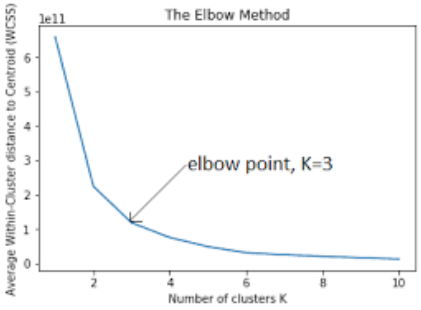


Fig 7.1. Elbow chart – In this case 3 clusters seems to be the best option.

7.1.5 MODEL OVERALL – PARTITIONING CLUSTERING

Used for: Clustering.

Advantages:

1) Easy implementation.
2) Scales better than others clustering algorithms.
3) Guarantees convergence.
4) Able to generalize to new examples.
5) Suitable for a well-separated clusters.

Disadvantages

1) The number of clusters k has to be defined *a priori* which is difficult to predict.
2) The final solution depends of the initialization.
3) Sensitive to outliers.

4) It works better with a low-dimension dataset (dimensionality reduction may be necessary).
5) Inability to find clusters of arbitrary shape.

Evaluation Metrics:

1) Extrinsic metrics: Adjusted Rand index, Fowlkes-Mallows scores, Mutual information-based scores, Homogeneity, Completeness and V-measure.
2) Intrinsic Measures: Silhouette Coefficient, Calinski-Harabasz Index and Davies-Bouldin Index.

7.2 HIERARCHICAL CLUSTERING

The hierarchical clustering was formulated with the purpose of overcoming some of the limitations of the partitional-based clustering, particularly the difficulty of setting a pre-defined parameter k for the desired number of clusters. Apart from that, the main purpose is exactly the same, the main goal is still to group datapoints in clusters based on their similarity. The approach to achieve that is however very different, the hierarchical clustering builds a tree diagram (called dendrogram) recording how the datapoints were grouped or divided.

With that said, the hierarchical clustering follows either a Agglomerative or a Divisive methodology (Aggarwal & Reddy, 2014). The agglomerative clustering is a bottom-top approach where each observation starts by belonging to its own cluster and pairs of clusters are merged together as the hierarchy goes up. In other hand, the divisive clustering is exactly the opposite, with a top-bottom approach and where all the data points start by belonging to the same cluster and then they are divided in sub-clusters as the hierarchy moves down.

7.2.1 AGGLOMERATIVE CLUSTERING

In the initialization of this algorithm, each datapoint is considered a cluster itself and in every interaction the two closest clusters are merged into a single one according to linkage criteria. All the linkage criteria's use the distance between the datapoints as a metric, but the rules of clustering's differ depending on the criteria used. Let us see the most used ones:

1. **Single-Linkage**: the distance between two clusters is defined as the shortest distance between two observations from different clusters (closest neighbour).
2. **Complete linkage:** the distance between two clusters is defined by the longest distance between two observations from different clusters (farthest neighbour).
3. **Average linkage:** as the name suggest, the distance between two clusters is defined by the average distance between all the observation of one cluster to all the observation of the other cluster.

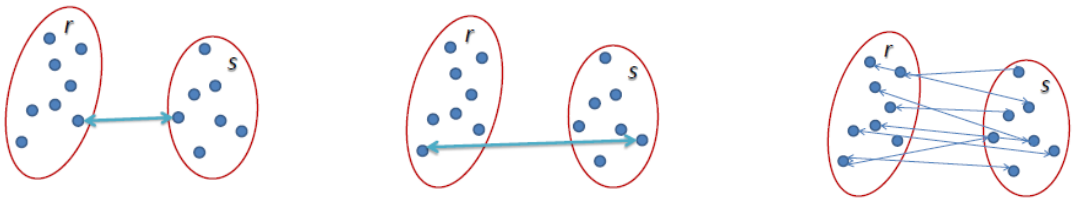


Fig 7.2. From left to right: Single linkage, complete linkage and average linkage.

In terms of the methodology to calculate the distance, there are also several ways to do it but the most used one is the Euclidean Distance (as it happens with the partitioning clustering). After selecting the linkage criteria, it is important to find a technique that allow us to easily compare distances and choose the clusters that will merge. The easiest way to find all those distances is with the support of a *Proximity Matrix*. It will naturally be a square matrix completely symmetric since the distance from A to B is the same as the one from B to A. Moreover, the main diagonal will be composed only by zeros because the distance between a cluster and itself is always zero, as it is presented below:

$$\text{Proximity Matrix} = \begin{pmatrix} 0 & d_{12}^2 & d_{13}^2 & \cdots & d_{1n}^2 \\ d_{21}^2 & 0 & d_{23}^2 & \cdots & d_{2n}^2 \\ d_{31}^2 & d_{32}^2 & 0 & \cdots & d_{3n}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d_{n1}^2 & d_{n1}^2 & d_{n1}^2 & \cdots & 0 \end{pmatrix}$$

After having the proximity matrix, the algorithm starts an interactive process of merging the two closest clusters. For example, in the first interaction, we have to find the smallest distance between clusters and merge them, after that we have to recompute the matrix and repeat the process. The historic of joins is recorded in tree diagram called Dendrogram. Just by looking to a dendrogram we can understand the full sequence of joins, starting from the bottom where all the clusters are separated as datapoints until the top where all the datapoints are converted into a single cluster.

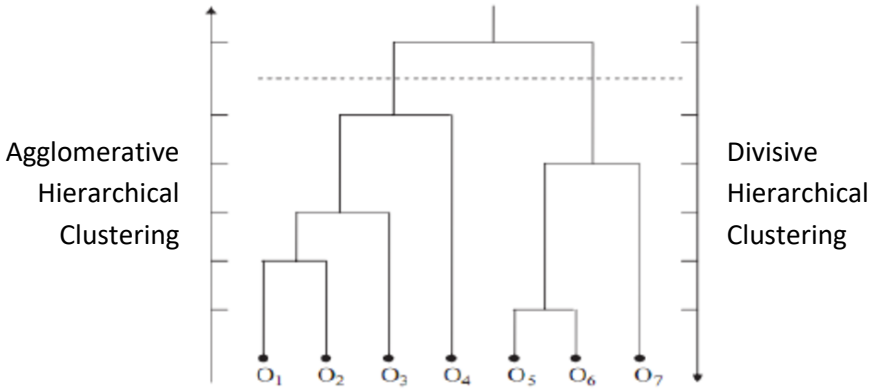


Fig 7.3. Dendrogram representation.

The previous dendrogram allow us to quickly realize that the first two clusters to merge were O_5 with O_6, followed by O_1 with O_2. The vertical lines of the dendrogram are proportional to the distance between the clusters. Furthermore, the analysis of a dendrogram also allow us to choose the right number of clusters, this is made by drawing a horizontal threshold that transverse the tallest vertical

line. The number of clusters should be the number of vertical lines that the threshold cut (in the previous example it is recommended to have only 2 clusters).

7.2.2 DIVISIVE CLUSTERING

The Divisive Clustering is exactly the reverse technique of the Agglomerative Clustering which means the clusters are made in a top-bottom methodology. With that said, instead of merging the clusters, this algorithm is now focus in splitting them until reaching a termination condition. Because of its splitting approach, the divisive clustering requires another algorithm as the k-means to perform the clustering in a sub-routine of the main algorithm. For that reason, the divisive clustering becomes more complex to adopt and it is less used in real-world situations. Apart of that, everything explain in the previous section remains valid in this algorithm.

.

7.2.3 MODEL OVEREALL – HIERARCHICAL CLUSTERING

Used for: Clustering.

Advantages:

1) No need to specify the number of clusters k in advance.
2) Easy to understand and to implement.
3) Calculates the whole hierarchy of the clustering process and allows to understand what was done with the dendrogram.

Disadvantages

1) Unable to make corrections once the split/merge is made.
2) Some parameters must be defined à priori such as the linkage criteria.
3) Misinterpretation of the dendrogram which may cause a wrong selection of the right number of clusters.
4) There are better techniques available today for finding groups of data.
5) Unsuitable for high-dimension datasets and it struggles with missing values or noisy data.

Evaluation Metrics:

1) Extrinsic metrics: Adjusted Rand index, Fowlkes-Mallows scores, Mutual information-based scores, Homogeneity, Completeness and V-measure.
2) Intrinsic Measures: Silhouette Coefficient, Calinski-Harabasz Index and Davies-Bouldin Index.

7.3 DENSITY-BASED CLUSTERING

The partitioning and the hierarchical clustering struggle to correctly find spherical-shape clusters such as an oval or a "S" shape (Han et al., 2012). Moreover, they are not robust enough to deal with outliers which usually lead to poor results of clustering. The density-based clustering is presented as a good alternative to the previous two clustering techniques allowing the algorithm to solve clustering problems with arbitrary shapes and to handle noise from outliers.

Fig 7.4. Clusters of arbitrary shape.

The main idea of any Density-based clustering is to identify dense regions in the data space, separated by sparse regions. It was initially derived from the human perception in observing a space of datapoints and intuitively group them according to density of points in certain areas.

7.3.1 DBSCAN

The Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a commonly used algorithm for clustering density. As any other algorithms of its family, the DBSCAN aims to detect regions of high density separated by regions with lower density in order to produce clusters of data. It all starts by finding core objects with dense neighbourhoods and then start to connect close neighbourhoods in order to find dense areas. To define the neighbourhood, we are going to follow the traditional methodology and set a space with radius $\epsilon > 0$ to every point of the dataset. With that said, the DBSCAN assume 2 different parameters:

1. ϵ - The radius of the neighbourhood.
2. *MinPts* - The minimum of points in ϵ-neighbourhood.

A datapoint becomes a core point or core object if it contains a neighbourhood with the minimum points around it. So given a Dataset D, with the respective parameters ϵ and MinPts, it is possible to identify all the core objects and list all the points associated to them. Furthermore, for a core point q and a random point p, we say p is *directly density-reachable* from q, if p is in the neighbourhood of q. The DBSCAN will form clusters with a similar concept called *density-reachable*. Let's say p is *density-reachable* from q if there is a chain of points that are *directly density-reachable* among them and somehow they connect p to q ($q= p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow \dots \rightarrow p_{n-1} \rightarrow p_n= p$). This chain of

connections will produce a dense region called cluster where not all the points are directly connected to each other but at least they are connected through a intermediate point.

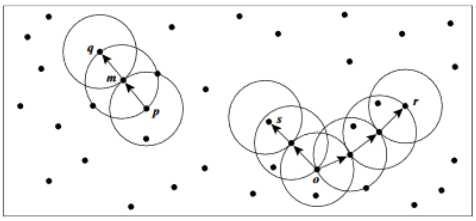


Fig 7.5. density reachable in the left (q-m) and density connected (s-r).

The only disadvantage the DBSCAN is selection of the right values of the parameters, which is usually difficult to get it right on the first attempt. To overcome this problem, Mihael Ankerst and his colleagues published a paper in 1999 with an alternative algorithm called OPTICS. This new algorithm is very similar to the DBSCAN but it solves its main weakness of being unable to correctly detect meaningful clusters in data of varying density.

.

7.3.2 MODEL OVERALL – DENSITY BASED CLUSTERING

Used for: Clustering.

Advantages:

1) It discovers clusters of arbitrary shape.
2) Robust with noisy data and outliers.
3) Widely used in real world situations such as anomaly detections, Spam filter, among others.

Disadvantages

1) Fail to identify clusters if the dataset is too sparse or if it has tricky densities.
2) The hyperparameter tuning is a trial and error process until finding the best combination.
3) Unsuitable for high-dimension datasets.

Evaluation Metrics:

1) Extrinsic metrics: Adjusted Rand index, Fowlkes-Mallows scores, Mutual information-based scores, Homogeneity, Completeness and V-measure.
2) Intrinsic Measures: Silhouette Coefficient, Calinski-Harabasz Index and Davies-Bouldin Index.

8. TREE-BASED ALGORITHMS

The tree-based algorithms are supervised models that are able of solving either a classification or regression problems. The main algorithm of this family is the Decision Tree which is one of the most powerful techniques for predictive modelling and has multiple application in a large range of scientific and social sciences.

The history of decision analysis started in 1931 with Frank P. Ramsey who proposed a methodology for decision-making based on probability and utility. In the following years, many other researchers gave their contribute on this field, one of them was the mathematician Abraham Wald in 1950 that used the theory of games to produce statistical methods for make decisions. However, the first full algorithm for classification and regression trees was developed in 1984 by Breiman with the CART algorithm. Despite their simplicity, decision trees are effective technique that provide human-readable rules (something that is rare in the current machine learning models, even the ones that are supervised are usually a black box during the training process).

8.1 DECISION TREES

A decision tree is a flowchart-like structure composed essentially by nodes, branches and leafs (Maimon & Rokach, 2010) .Each node represents a test of an attribute (e.g Yes/No), the outcome of the node is called branch and basically it expresses the decision made on that node. The first node of all is called *root node* and a decision tree may have multiple levels of nodes, the total number of levels/layers is called Depth (the root node is excluded from this calculation). Finally, in the end of a decision tree we have the leaf node that represents the class label according to all the previous classification rules.

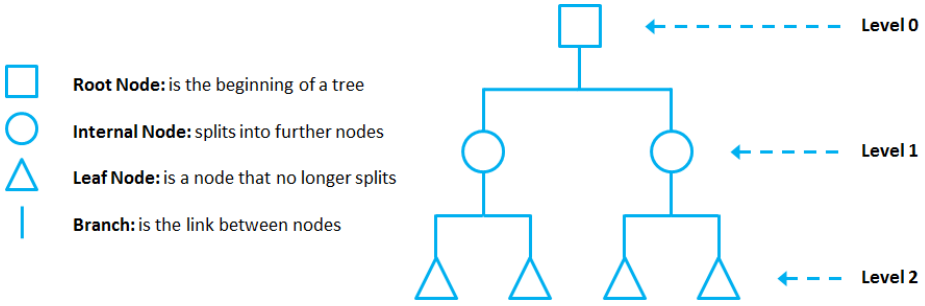


Fig 8.1. Decision Tree structure.

The data is embedded in a top-down methodology which means the root node always receives the original data and then it sends for the next descending nodes in order to apply the classification rules. Those rules are produced and organized during the training phase where the algorithm will design a decision tree that best split the data according to its similarity. The elements that show high similarity between each other should get the same classification (Homogeneity) and the ones with different characteristics may fall in different classification groups. Note that those splits may have a

binary or a multiway format depending on the number of branches a node has (e.g the figure 8.1 is an example of nodes with binary splits since there are only two possible outcomes in each node).

As we increase the tree dimension, it will not only become more difficult to read but it also increases the probability of falling in overfitting. This happens because we are adding too much detail to the tree and some of that information could just be noisy data (which will also be learned during the training phase). Sometimes less is more and this is surely one of those cases. Our goal is to have a decision tree with a good generalization that is able to split the data in the best way possible, in other words we want to reduce the complexity of the tree in order to improve its predictive power. The solution to avoid the overfitting was proposed by Breiman in 1984 and it is called **pruning**, basically it will remove all sections of the tree that provide little predictive power for the classification.

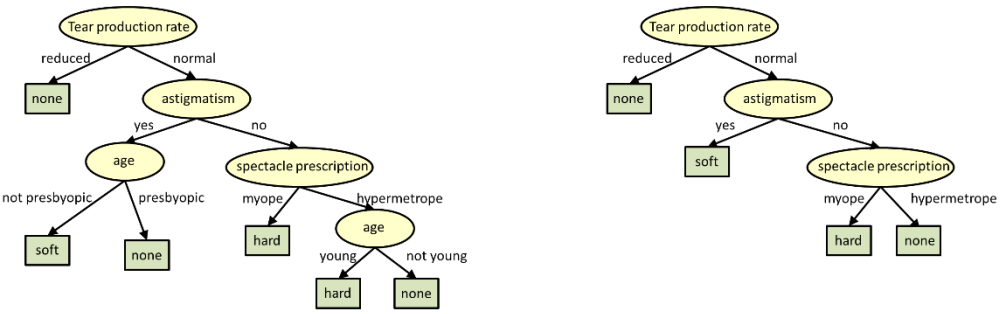


Fig 8.2. Pruning illustration.

One fundamental step in decision trees is the selection of the algorithm for splitting the nodes. The algorithms are usually based on a single criteria (univariate) but it is possible to have multiple criteria's on a single algorithm (multivariate). All of those criteria's are focus in calculating the impurity of the node which is a measure of homogeneity of the node labels. In other words, the algorithm will split the nodes which result in the most homogeneous sub-nodes. It is then easy to deduce that the homogeneity of the nodes will increase in the direction of the leaf-node.

8.1.1 ID3 ALGORITHM

The ID3 was an algorithm proposed by Ross Quinlan in 1986, it is an acronym for *Interactive Dichotomiser*. The criteria for splitting the nodes in the ID3 is based on the concepts of entropy and information gain, so first of all it is important to understand those concepts and how they are calculated:

- **Entropy**: is a measure of disorder, it calculates the homogeneity of the sample or in other words how similar are the values between each other. If all the values have the same value (full homogeneity) then the entropy will be equal to zero and if the values are equally distributed it will reach its maximum value of 1. The entropy chart has a shape of an inverted U and it can be calculated by the following formula:

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

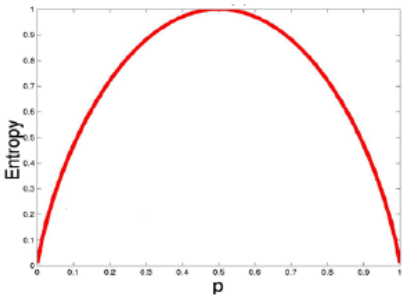


Fig 8.3. Entropy curve.

- **Information Gain:** Is the key to build a good decision tree. The main goal is to find the highest information gain in each node-split in order to reduce the disorder (entropy) as fast as possible. Mathematically it can be expressed as below:

$$IG(Y, X) = E(Y) - E(Y \mid X)$$

Where E(Y) is the entropy of the target variable and E(Y|X) represents the expected entropy after Y has been partitioned using X. The ID3 algorithm has a lot of advantages including the fact that is able to build a short tree with less computational power. However, it is unable to handle continuous or missing values which lead us to some other algorithms as an alternative.

8.1.2 C4.5 ALGORITHM

From the same author of the ID3, Quinlan proposed in 1993 a new algorithm called C4.5 that represents an improvement over the ID3. It solves the main disadvantages of the ID3, the new algorithm can handle continuous and categorical data, as well as the missing values by removing the section of tree where those values are present.

Additionally, Quinlan discovered that the splitting criteria based on information gain would give an unjustified favouritism to the attributes with many outcomes. To fix that and avoid bias problems, C4.5 uses another criterion to split the nodes called Gain Ratio. The Gain Ratio works like a normalizer for the attributes with many outcomes, it firstly calculates the intrinsic information of an attribute which is given by:

$$IntI(S, A) = -\sum_{i} \frac{|S_i|}{|S|} \log(\frac{|S_i|}{|S|})$$

Then it uses that value and divides it by the information gain of the feature:

$$GR(S, A) = \frac{Gain(S, A)}{IntI(S, A)}$$

The attribute with the largest Gain Ratio is chosen for the first splitting and the next ones come in descent order. Also the training of C4.5 differs from the ID3, in this new algorithm a subset of the data is used to build the decision tree, then the rest of the data is used to test that decision tree and if the classification is correct then the algorithm finishes, otherwise it will make the necessary adjustments in an interactively process.

Recently there was an update on this algorithm for the C5.0 version where several improvements were made over the previous version. Among those improvements, it is worth to mention that the C5.0 is now able to compute the algorithm faster and with less computational power compared to the C4.5, also it is capable of performing a more accurate classification using smaller trees.

8.1.3 CART ALGORITHM

CART stands for Classification and Regression Trees and as the same suggest it differs from the other algorithms for being able to generate regression trees. If the target variable is continuous (regression) then the splitting criteria will be based on minimizing the Least Squares Deviation (LSD), which is a metric used to measure the difference between the real values and the predicted values. Otherwise, if the target variable is categorical, the splitting criteria is either the Gini Index or the Twoing Criteria. The Gini Index measures the probability of a random instance being wrong classified, the Gini Index will assume values between 0 and 1, where 0 means all the elements belong to the same class and 1 denotes that the elements are equally distributed among all the labels.

$$Gini\ Index = 1 - \sum [p(\frac{i}{t})]^2$$

Where p(i/t) represents the fraction of records belonging to class i on the node t. However, sometimes the Gini Index does not work well when the domain of targeting attribute is relatively wide, in those cases is recommended to use the twoing criteria defined as:

$$Twoing\ criteria\ (t) = \frac{P_L P_R}{4} (\sum | p(\frac{i}{t_L}) - p(\frac{i}{t_R}) |)^2$$

After the splitting criteria has been chosen, the Cart Algorithm will produce sequentially several trees and test them with unseen data to choose the best one for the final solution. The table below summarizes the main characteristics of all the 3 algorithms explained so far (Singh & Gupta, 2014).

Characteristic(→) / Algorithm(↓)	Splitting Criteria	Attribute type	Missing values	Pruning Strategy	Outlier Detection
ID3	Information Gain	Handles only Categorical value	Do not handle missing values.	No pruning is done	Susceptible on outliers
CART	Towing Criteria	Handles both Categorical and Numeric value	Handle missing values.	Cost-Complexity pruning is used	Can handle Outliers
C4.5	Gain Ratio	Handles both Categorical and Numeric value	Handle missing values.	Error Based pruning is used	Susceptible on outliers

Fig 8.4. Comparison table of the algorithms ID3, C4.5 and CART.

8.1.4 MODEL OVERALL – DECISION TREES

Used for: Classification and Regression.

Advantages:

1) Does not require a lot of preparation in pre-processing compared to other algorithms.
2) Does not require normalization and scaling before running the model.
3) It is easy to explain and very intuitive to understand since it follows the same logic that humans use to take decisions.
4) Robust to outliers.

Disadvantages:

1) Very instable model, a small change in data cause huge changes in the structure of the tree.
2) The cost in terms of time and computational power increases exponentially as more class labels are added to the model.
3) It is not the most appropriate model for regression problems.
4) High risk of overfitting compared to other models.
5) A single tree rarely present solid results, usually it is necessary to combine trees (chapter 8.2).

Evaluation Metrics:

1) For Regression trees: MSE, RMSE and MAE.
2) For Classification trees: Accuracy, Confusion Matrix, Precision, Recall, F-Measure and AUC-ROC curve.
3) For both: Stability, Simplicity and Discriminatory power (Osei-Bryson, 2004).

8.2 RANDOM FOREST

Random Forest as the same suggest consists on multiple decision trees working together as an ensemble predictor. The algorithm was proposed by Leo Breiman in a series of several researches and technical reports between 1996 and 2004, on those reports Breiman proved that there are significant advantages by using ensembles of trees instead of using just one tree. The fundamental principle of the Random Forest states that a Large number of relatively uncorrelated models (trees) operating as a committee will outperform any individual constituent models (Yiu, 2019).

The main key of a good Random Forest is all about low correlation between its trees. Only in this way we can ensure the diversity of combinations and guarantee that the algorithm will have a high accuracy even if there are some trees with bad results on it. This diversity is provided by the randomness of the model, this is achieved in two stages:

1) With random samplings of data in the training phase
2) With random sets of features to perform the splitting nodes.

The random sampling is obtained by a technique called *bagging* which represents an abbreviation for Bootstrap Aggregation. This technique is based on the idea of creating several subsets of training data chosen randomly and with replacement. This procedure will allow to reduce the variance of the random forest since generally based-tree algorithms are well known as an algorithm with the problem of high variance.

Alternatively, we can use a different type of ensemble method called Boosting. This ensemble technique is still focus in reducing the variance and providing stability to the model, but this time the Boosting ensemble will give privilege to the input data wrongly classified. This is made by assigning weights to samples of data and give more weight to the samples that were incorrectly classified more often. In this way the Boosting ensemble raises the difficulty of the training phase and consequently it increases the model ability for classification. The only if with the boosting refers to the possibility of falling in overfitting, so some caution is suggested during its implementation.

Finally, it is important to mention which parameters are necessary to run the model. During the training phase, the Random Forest requires 4 hyperparameters

1) How many trees we want to combine;
2) The maximum depth of tree we are willing to accept;
3) The maximum number of features used in each split;
4) The Ensemble method.

Naturally, training multiple trees instead of just one brings more accurate results but also some drawbacks such as the running time and the amount of memory necessary to run the full model.

8.2.1 MODEL OVERALL – RANDOM FOREST

Used for: Classification and Regression.

Advantages:

1) High performance algorithm that competes with the top machine learning model nowadays.
2) Does not require a lot of preparation in pre-processing compared to other algorithms.
3) Does not require normalization and scaling before running the model.
4) More stable than the Decision Tree algorithm since a change in the dataset may affect one tree but it is difficult to affect all them.
5) Robust to outliers.
6) Works well with high-dimensionality datasets.

Disadvantages:

1) It will require more time and memory than the majority of the models.
2) It is less intuitive than a simple decision tree.
3) Since it is an ensemble model it is less interpretable and the path to achieve the results becomes more like a black-box.
4) Prone to overfitting.

Evaluation Metrics:

1) For Regression trees: MSE, RMSE and MAE.
2) For Classification trees: Accuracy, Confusion Matrix Precision, Recall, F-Measure and AUC-ROC curve.
3) For both: Stability, Simplicity and Discriminatory power.

9. TIME SERIES

Over the past decades, the time series modelling and forecasting have been a main topic of research in which several researchers have proposed different models in order to improve the accuracy of the Time Series algorithms. Generally, the main goal of time series modelling is to collect data from the past, find patterns and try to predict the future from it. In other words, we can define this family of algorithms as an attempt of predicting the future by understanding the past.

9.1 FUNDAMENTAL CONCEPTS OF TIME-SERIES MODELLING

Firstly, it is important to give a rigorous definition of a Time Series, according to the book *An Introductory Study on Time Series and Forecasting*, it is defined by:

> *"Time Series is a sequential set of data points, usually measured over successive times and in chronological order. Mathematically, it can be expressed by a set of vectors x(t) where t represents the time elapsed".* (Agrawal & Adhikari, 2013)

Now that we better understand what a Time Series is all about, we can move forward to a more detailed analysis, including the main elements of a Time-series:

1) The Time-Series is called *Univariate* if it is only recording one single variable along the time. Otherwise, if two or more variables are considered for the analysis then it is named as *Multivariate*.
2) The data modelling is either composed by *Continuous* observations if there are no interruptions in data collection over the time, or *Discrete* if the data collection is made from time to time and there is a finite number of observations.
3) Finally, a time series can be decomposed in 4 main components: Trend, Cyclical, Seasonal and Irregular components. The decomposition provides a useful perspective and a better understanding of a time series, it also helps to make a more reliable forecast. All four components are well explained below:

   3.1) A T*rend* exists when there is a long-term increase or decrease in the data (not necessarily linear). The trend may suffer a "changing direction" if it passes from an increase trend to a decrease trend (or the opposite).

   3.2) Seasonal patterns occurs every time the series is affected by seasonal factors. It is represented as a fluctuation that is caused by the seasonality (e.g people usually buy more sunscreens during the summer).

   3.3) Cyclical patterns happens when the data exhibit rises and falls without being possible to detect its frequency. The duration of a cycle depends on many variables, it can have the same length of a seasonal pattern but it can also be shorter or longer.

   3.4) The irregular or the random effect is last component and it cause fluctuations on the time series due to unpredictable events. The irregular and cyclical patterns are usually called Residual when combined.

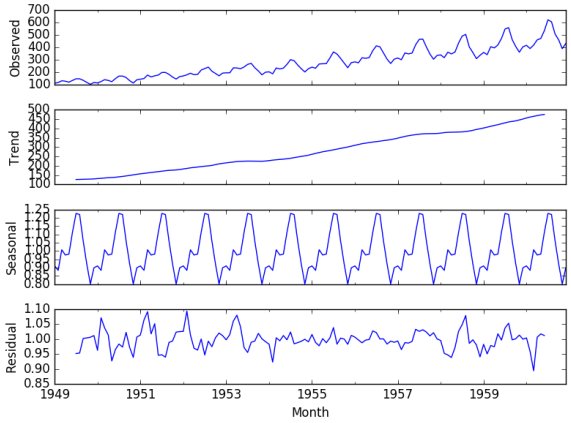


Fig 9.1. Time-Series Decomposition.

Considering the effect of the 4 components, we can express it mathematically through a multiplicative or additive format depending on our model. Usually, the multiplicative approach is used when we assume that the components are independent between each other, otherwise it is recommended to choose the additive format.

$$Y(t) = T(t) \times S(t) \times C(t) \times I(t) \qquad Y(t) = T(t) + S(t) + C(t) + I(t)$$

Where Y(t) reflects the observation and T(t), C(t), I(t) represent the Trend, Cyclical and Irregular patterns over the time t.

9.2 STATIONARITY OF A TIME SERIES

Most statistical forecasting methods assume that the time series is strictly *stationary*, which means the joint probability distribution does not change over time and consequently neither does the parameters such as mean or variance (Terrell, 2019). A stationary process is much easier to analyse and has become an assumption before taking conclusions, estimating or forecasting a time series. However, in most cases it is highly probable that the time series is non-stationary specially if it has a long historical of observations or if it is influenced by trend/seasonal pattern. In this last case it is possible to make some transformations by removing those patterns and make the time series stationary. Sometimes, in the cases where it is not possible to have a strictly stationary time series it may be enough to use *weakly stationary* process which requires shift-invariance of a certain number of statistical orders as well as the covariance. For example, a second order stationary has independent mean and variance, in that case the covariance between 2 data points, t and t-k, will only depend on k.

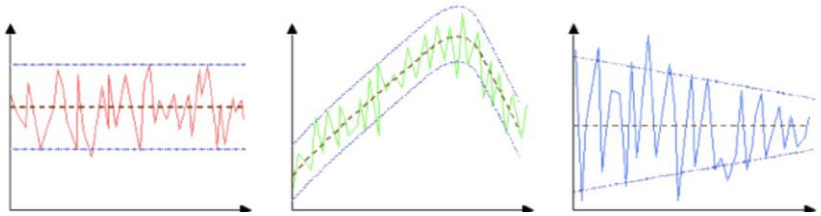


Fig 9.2 In the left, the time series has both mean and variance stationary. In the middle it has a non-stationarity mean but a stationary variance. And in the right, the time series is non-stationary in terms of mean and variance.

There are many models available to extract useful information from a time series and produce credible predictions to the future. All of them share the same fundamental principle of trying to understand the past to predict the future.

The Time Series algorithms can be separated in two main groups, the Linear models and the non-linear models. The first ones are extremely useful and very straightforward in terms of implementation. The Linear models assume a certain level of linearity and stationarity without expecting irregular oscillations. However, in most real-life cases this is not what usually happens and most of the times we will have high-volatile data reaching extreme values. A Linear model will not deal well with that kind of data and it is highly recommended to choose an algorithm among the non-linear models as we will see ahead.

9.3 LINEAR MODELS

There are two main groups of Linear Models, the Autoregressive and the Moving Average models, that can be combined into a more complex algorithm called ARIMA that stands for Autoregressive Integrated Moving Average. Generally, the ARIMA model is expressed by:

$$y_t = a_t + \sum_{m=1}^{M} b_m y_{t-m} + \sum_{n=0}^{N} c_n x_{t-n}$$

AR MA

Where a_t represents a constant of initialization, AR is the output of the Linear Regression over the past p observations, MA is the moving average calculated with a q-point range window. Additionally, the model has a d parameter that stands for the degrees of differencing. Combining those simple models with the respective parameters we get the ARIMA(p,d,q). As we saw before, the model has to be stationary and some transformations may be required.

Let us just remember that a Linear Regression consists of finding the best-fitting line based on the existing data points and in order to predict future values. The mathematical expression is the same as the one explained in the chapter 5 but in this case we want the predict the value X_t based on the past values of X_{t-j}.

$$X_t = \sum_{j=1}^{P} \beta_j X_{t-j} + e_t$$

Regarding the Moving Average (MA) segment, despite its simplicity it is known as one of the best techniques to evaluate the trend of a time series and to smooth short-term fluctuations that are not important for the analysis. This is made by calculating successive averages based on subsets of the full dataset like a rolling window.

There are several types of MA, the most used ones are the Simple Moving Average (SMA) and the Exponential Moving Average (EMA). The SMA is simply calculated by the arithmetic mean of the number of periods (n).

$$SMA = \frac{P_1 + P_2 + ... + P_n}{n}$$

The SMA is very useful in many cases but it has the disadvantage of giving the same weight for all the periods P. In reality, this is not what often happens and becomes evident in some examples such as the stock markets where the price of yesterday is more important than the price of the 10 days ago in order to predict the price of tomorrow. To bridge that gap, it was proposed a new approach called Exponential Moving Average that will give more weight to the most recent values:

$$EMA_t = [v_t \times (\frac{s}{1+p}) + EMA_y \times [1 - (\frac{s}{1+p})]$$

Where, V_t is the value of the current period, s is the smooth operator and p represents the number of periods. Both approaches can be applied on the ARIMA model.

9.3.1 ARIMA

Forecasting through an ARIMA model will only work if the time series is stationary, meaning it does not depend on time and for that reason it is not influenced by trends or seasonality. An example of a stationary time series is the White Noise series where the data is uncorrelated across time with zero mean and constant variance. However, it is pretty rare to find a time series originally stationary and most likely it will need some transformations to make it stationary. These transformations are usually made by differencing or logging it. Both transformations have the same goal of stabilizing the series, however a logarithm is more focus on stabilizing the variance and the differencing on stabilizing the mean. So in cases we have huge oscillations and unpredictable ups/downs in the chart it is required to calculate the **differenced series:**

- Random walk for the first difference: the first difference is nothing more than the change between two consecutive points:

$$y_t^{'} = y_t - y_{t-1}$$

- Sometimes the first difference is not enough and a second difference must be calculated to obtain a stationary time-series:

$$y_t^n = y_t^{'} - y_{t-1}^{'}$$
$$(=)\ y_t^n = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2})$$
$$(=)\ y_t^n = y_t - 2y_{t-1} + y_{t-2}$$

But how do we know if the series is already stationary or not? The answer was proposed by Box and Jenkins in 1976 with the calculus of the correlations between the series x_t and the lagged values (x_{t-1},x_{t-2}, and so on). Those differences are represented on a chart named ACF (Autocorrelations Function). An example of an ACF chart is illustrated in fig.9.3.

Usually, for a stationary series, the ACF will drop quickly to almost zero, while the non-stationary series will have a slowly decrease until reaching almost zero. Alternatively, we can use the PACF (Partial Autocorrelation Function) which gives a more clear view over the correlation existing between x and x_{t-k}. Basically in PACF we analyse the correlation at any lag k after removing the effect of all the previous lags. For example, in ACF the price stock of today is influenced by the price of yesterday which was influenced by the price of the day before yesterday. The same does not happen with PACF where the price effect of the day before yesterday will be deduced on the price effect of yesterday in order to avoid accounting the same effect multiple times.

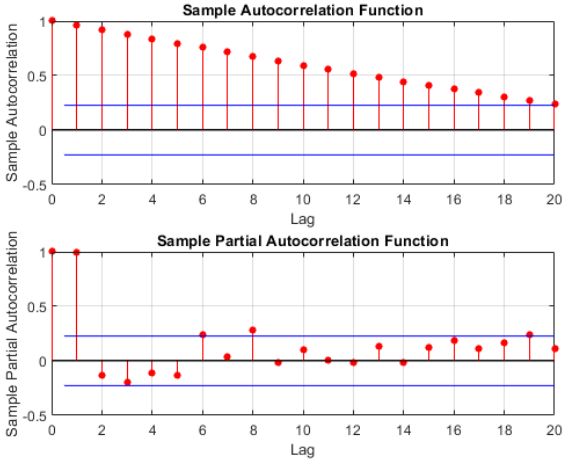


Fig 9.3. ACF and PACF plot.

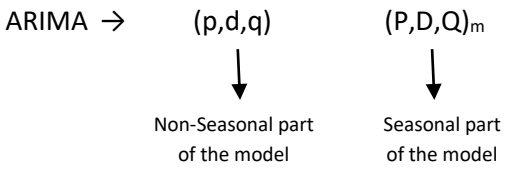
After knowing how the correlation between x_t and its lags behaves, especially the ACF and PACF plots, we can move forward to the second step of building an ARIMA model which is to fit the model with the best parameters possible.

This phase is called model tuning, in the past the selection of the hyperparameters was made manually based on a set of rules associated with the ACF and PACF. Obviously with that technique there was a huge margin of error and the model was rarely full optimized. Fortunately, nowadays we can find the best combination of the hyperparameters with one single line of code (for example in python it is enough to import the library auto_arima and execute the command model.fit()). Behind that command, the auto_arima is processing iteratively multiple combinations of the parameters and selecting the best one based on two criterions (AIC and BIC that stands for Akaike Information Criterion and Bayesian Information Criterion respectively).

So far, we have been focused only on non-seasonal ARIMA. Although, ARIMA models are also able of modelling seasonal data. Box and Jenkins have also proposed a generalized model called SARIMA (Seasonal Autoregression Integrated Moving Average) that is able to deal with seasonality. Also, in this model we may need to perform *differencing* but in this case between seasonal periods, as it is showed below:

$$y_t^{'} = y_t - y_{t-m}$$

Where m is the number of periods, for monthly time series m=12 and for quarterly time series m=4. The model is generally expressed by:



The first part of the model is exactly the same as an ordinary ARIMA model which is complemented with the Seasonal part (written in uppercases to distinguish) that also works with the same parameters but this time the backshift is made to the previous season.

9.3.2 MODEL OVEREALL – ARIMA

Used for: Regression and Forecasting.

Advantages:

1) Very intuitive model, easy to understand and implement.
2) The short-term forecast is usually very accurate and difficult to beat even with more complex models.

Disadvantages

1) Weak results in long-term forecasting.
2) Sometimes it is outperformed by simpler models such as exponential smoothing.
3) ARIMA model has a fixed structure with low flexibility.

Evaluation Metrics:

1) MAPE, MAE, MPE and MASE.

9.4 NON-LINEAR MODELS

The study of non-linear model on time series is much more recent when compared to the linear models. Despite the fact there are already some interesting models proposed, it is evident the lack of literature on this subject. As covered before, an assumption of the linear models is that the evolution of the observations has to be regular without sudden jumps and unpredictability. The true is in most of the real-world cases we will be working with non-linear data and we have to choose the correct model according to that.

First of all, we want to make sure that we are dealing with a non-linear problem. To ensure that we can make some tests such as BDS (Brock et al., 1996). Once we have guaranteed the data is following a non-linear pattern, we can move forward to the selection of the model that seems more appropriate for our data and for the purpose of the forecast. There are several of models that can be chosen. During the model selection it should be considered the fact that some of the models are focussed to deal with non-linear mean and others to work with non-linear variance.

With that said, there are two big families of non-linear models, the Threshold Autoregressive (TAR) and the Autoregressive Conditional Heteroskedasticity (ARCH). The idea of the TAR model is to approximate a general nonlinear model by using a threshold and it can be expressed by:

$$y_t = (\alpha_0 + \alpha_1 y_{t-1} + ... + \alpha_p y_{t-p})\, f(q_{t-1} < \gamma) + (\beta_0 + \beta_1 y_{t-1} + ... + \beta_p y_{t-p})\, f(q_{t-1} > \gamma) + e_t$$

Where α represent the parameters values below the threshold γ and β the parameters values above that threshold. Additionally, the $f(q_{t-1}=\gamma)$ is a known function and the autoregressive order is 1 in this model. If the discontinuity of the threshold is replaced by a smooth transition than the model is called STAR for Smooth Transition Autoregressive.

The other main model for non-linear data is called ARCH and it is particularly useful when we are working with high volatile data. The model was proposed by Engle in 1982 in a paper where he explicitly recognizes the difference between the unconditional and the conditional variance allowing the later to change over the time as a function of past errors. The ARCH(q) model is nothing more that AR(q) applied to the variance of a time series and where q is the number of lags (squared) that we want to include in the model. For example, an ARCH(q) denotes the conditional variance of y_t in relation to the q past observations:

$$\text{var}(y_t \mid y_{t-1}, ..., y_{t-q}) = \sigma_t^2 = \alpha_0 + \alpha_1 y_{t-1}^2 + ... + \alpha_q y_{t-q}^2$$

If we assume that we have mean equal to zero, the ARCH model can be written as:

$$y_t = \sigma_t E_t \text{ with } \begin{cases} \sigma_t = \sqrt{\alpha_0 + \alpha_1 y_{t-1}^2 + ... + \alpha_q y_{t-q}^2} \\ E_t \sim iid(\mu = 0, \sigma^2 = 1) \end{cases}$$

There are some related models to the original ARCH including the Generalized Autoregressive heteroskedasticity (GARCH) that also includes a Moving Average together with the Autoregressive component.

9.4.1 MODEL OVEREALL – NON-LINEAR MODELS (TIME SERIES)

Used for: Regression and Forecasting.

Advantages:

1) It overcomes the main limitations of the Linear Models.
2) More versatile models for the time series analysis.

Disadvantages

3) Lack of bibliography and models.
4) Complex implementation.
5) Low adoption by the scientific community.

Evaluation Metrics:

1) MAPE, MAE, MPE and MASE.

10. GENETIC ALGORITHMS

The use of evolutionary principles for automatic problem-solving was firstly introduced in 1950s when several specialists explore this idea in order to get better results in optimizations problems. During the decades of 50s and 60s, many researchers dedicated their time and effort in evolutionary programming strategies, among them it is worth mentioning Rechenberg (1965, 1973), Schwefel (1975, 1977), Box (1957), Friedman (1959), Bledsoe (1961), Bremermann and Reed (1962), Toombs, and Baricelli (1967). It was undoubtedly an interesting topic of research in the middle of the twenty century, something that has maintain until nowadays.

However, the name Genetic Algorithms just appeared in 1960 when John Holland and his colleagues of the university of Michigan started a research program with two main goals:

1. To abstract and explain in detail the adaptative process of natural systems;
2. To design an artificial system software able to retain the important mechanism of natural systems.

The research of John Holland in the University Michigan lasted until 1975 when he finally published his book *Adaptation in Natural and Artificial Systems* with a deep review over the two topics mentioned before. The results of his work were absolutely revolutionary in the research community allowing an all new set of possibilities in optimization problems. Based on Biological evolution, a Genetic Algorithm (GA) is a optimization algorithm that starts with a given population of chromosomes (e.g binary string) and uses natural selection to generate a group of individuals with higher chances to survive (e.g fitness evaluation). In fact, the process of generating new individuals involves several genetic operators such as Crossover, Mutation and Selection (Goldberg, 1989).

Since the publication of the Holland's book, a lot has already changed. The study of Genetic Algorithms become a main topic of research and several new approaches have been discovered allowing to get better results and also to compute more complex data.

10.1 BIOLOGICAL TERMINOLOGY

Before we move forward to a more detailed review of Genetic Algorithms, it is important to clear up the origin of some concepts related to the algorithm and how they establish a relationship of pure analogy to the biological terminology.

Chromosomes are long stretches containing many genes carrying essential genetic information (DNA) needed to build a new organism. Chromosomes in genetic algorithms are represented by strings that are usually binary, each of them is analogous to a gene.

The entire combination of genes has the name of genotype which leads to the phenotype that represents the organism formed in the end of the genetic process (biologically this could represent the eyes colour, height, hair colour). In terms of genetic algorithms, the same idea is used, in this case, the genotype represents the entire combination of strings and the phenotype the final solution. A solution is obtained by decoding the best genotype after termination. Let us just remember that there are two possible approaches of coding:

1. **Encoding:** the mapping from the phenotype to the genotype;

2. **Decoding:** the inverse mapping from the genotype to the phenotype.

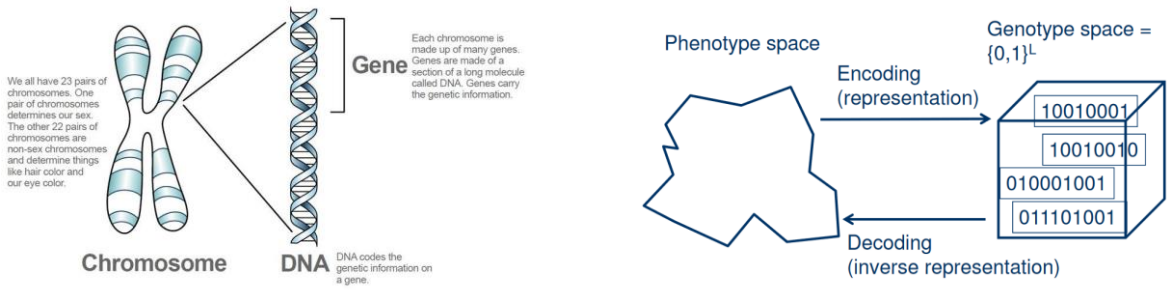


Fig 10.1. On the left the DNA structure and on the right the representation of the encoding/decoding.

In fact, from the genotype to the phenotype, a lot of operations occur and it is important to understand their biological origin. In nature, most species reproduce through diploid chromosomes meaning that the chromosomes are arrayed in pairs. During the sexual reproduction, the Crossover occurs, moment when genes (bits) from each parent are pair up in order to form a new single chromosome (string). In Genetic Algorithms, the crossover is usually followed by mutation which is the moment where single bits of DNA change (mutate) themselves, producing a new chromosome (offspring) different than the previous one. The mutation operator is particularly useful to ensure the diversity of the population. Finally, in order to choose the best individuals we need a scale of measure, in GA the measure used is called *Fitness Function* and as it happens in biology is often defined as the probability of a certain individual to survive and reproduce (viability) or as a function of the number of offspring's (fertility) (Melanie, 1999). Over the next chapters, all these 3 operators will be explain in detail, but it is important to understand from the beginning that GA are not a choice between crossover, mutation or selection but rather a balance among those operators that will produce the best result.

10.2 THE GA OPERATORS

In this section we will cover all the sequence of GA operators. Firstly, those operators work in a cycle called evolutionary process. As it is represented in figure 10.2, a GA is based on a natural evolutionary process composed by 5 phases. The algorithm starts with a random or manual initialization defined in a certain search space. Then the evolutionary process passes through Crossover operator, where the genetic information of two parents is used to make a

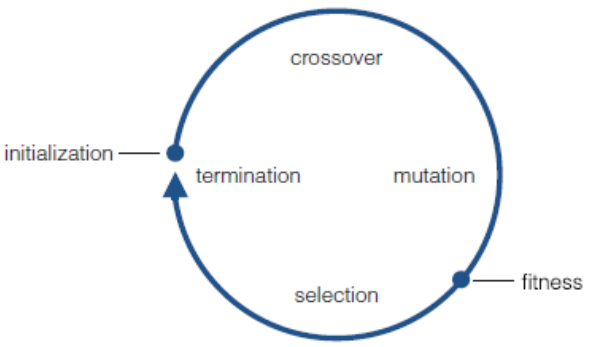


Fig 10.2. GA operators cycle.

new offspring. That offspring is then mutated in the third phase of the evolutionary process. In the next phase, Selection, the fitness of all the new offspring's (individuals) is calculated and the best ones are selected for the following generation. Finally, in the last phase, one of two situations may occur, either the termination condition (TC) was met and the process ends or it goes back to the beginning of the process and repeats all the steps until the genetic optimization is good enough according to termination condition (Kramer, 2017).

10.2.1 CROSSOVER

Crossover is one of the most prominent operators in Genetic Algorithms. Also called recombination, the crossover is the operator that allows to combine genetic information of different parents in only one offspring creating a new individual. There are several ways to apply the crossover, each one with their own advantages and disadvantages. In this work we will cover the main ones.

Starting by the ***One-Point Crossover,*** this is probably one of the simplest crossovers methods that will just pick a random point on both parents chromosomes and swapped them in order to create two new offsprings:

Parent 1: 1 0 1 0 | 1 0 0 1 0
Parent 2: 1 0 1 1 | 1 0 1 1 0

Offspring 1: 1 0 1 0 | 1 0 1 1 0
Offspring 2: 1 0 1 1 | 1 0 0 1 0

Another approach is called the ***N-Points Crossover***, where the idea is almost the same as the previous one but instead of selecting just one point to cut the parents, there are multiple cutting points chosen randomly and the new offsprings are generated by combining the parents.

Parent 1: 1 0 | 1 0 | 1 0 0 | 1 0
Parent 2: 1 1 | 0 0 | 1 0 1 | 1 0

Offspring 1: 1 0 | 0 0 | 1 0 0 | 1 0
Offspring 2: 1 1 | 1 0 | 1 0 1 | 1 0

In the example above, we have a 3-points crossover where the 3rd, 4th, 8th and 9th are the selected points for the crossover between the parents.

Finally, in contrast to the previous two crossover operators, the ***Uniform-Crossover*** does not divide the parent chromosome into segments for recombination. Each gene of the new offspring is selected by simply copy it from a parent chosen according to a binary crossover mask of the same length of the parents chromosomes. For example, if the bit in the mask is 1 then the resultant gene is copied from the parent 1, otherwise, if the bit is 0 it will copy the gene from the parent 2. The crossover mask is usually generated randomly and from this type of crossover only one offspring is created at a time.

Parent 1: 1 0 1 1 0 0 1
Parent 2: 0 0 0 1 1 0 1 0

Mask: 1 1 0 1 0 1 1 0

Offspring 1: 1 0 0 1 1 0 1 0
Offspring 2: 0 0 1 1 0 0 1 1

10.2.2 MUTATION

The next operator is called **Mutation,** this operator is one of the most important stages in GA due to its impact in the exploration of the searching space and due to the capacity of overcoming the problem of earlier convergence. There are several types of Mutation and for each problem the user has to decide which mutation approach is more appropriate. Regardless of the mutation type, it is crucial to fulfil the following 3 conditions:

1) **Reachability:** this condition ensures that each point in the solution space is reachable.
2) **Unbiasedness:** The mutation operator should not induce a drift of the search to a certain direction.
3) **Scalability:** Finally, each mutation operator must offer the degree of freedom that is able to adapt to the fitness landscape. This is often obtained by using probabilistic distributions as the Gaussian Distribution.

Among the main mutation types, we have:

Insert Mutation: The idea is to randomly pick a gene and move it to another place. All the other genes must shift in order to accommodate this move.

Unmutated offspring: 01|2|3456789
Mutated offspring: 01345|2|6789

Inversion Mutation: In this type of mutation a subset of genes is selected and then we invert the order of the subset.

Unmutated offspring: 01|234567|89
Mutated offspring: 01|765432|89

Scramble Mutation: As it happens in the previous mutation type, here we keep the idea of selecting a subset of genes but this time we randomly rearrange them instead of inverting it.

Unmutated offspring: 01|234567|89
Mutated offspring: 01|372465|89

Swap Mutation: Here we randomly select two positions of the chromosome and we interchange them:

Unmutated offspring: 012|3|45|6|789
Mutated offspring: 012|6|45|3|789

Flip Mutation: This mutation type is based on binary chromosomes meaning that the genes are either 0 or 1. To apply this mutation we choose one or more genes and flip them (from 0 to 1 or from 1 to 0).

Unmutated offspring: 001|1|010010
Mutated offspring: 001|0|010010

Uniform Mutation: The Mutation operator replaces the value of the chosen gene with an uniform random value (selected between the user specified upper and lower bounds of the gene)

Gaussian Mutation: This operator adds a random value from gaussian distribution to the chosen gene. Then if the result falls out the lower or upper bounds, the new gene is clipped.

$$X' = x + \sigma.N(0,1)$$

10.2.3 SELECTION

The selection is the last operator in the genetic algorithm cycle before it either starts a new cycle or it ends if the termination condition was met. Before we move forward to the selection operator, it is important to understand the concept of *Fitness Function* in which the selection is based on. The Fitness Function evaluates how close a given solution is to the optimum solution, in other words, it is a function that takes a given solution as input and returns as output the quality measure of that solution to the problem.

Usually a fitness function seeks for a maximum or minimum value to the problem in consideration but in some situations, we may have more complex Fitness Functions. Regardless the problem we want to optimize, every fitness function must be fast to compute and must be able to quantitatively measure how fit a specific solution is in solving the problem.

After fitness assignment has been performed to all individuals, the selection operator will choose the best individuals for the next generation. As it happens in biology, the best individuals are the ones with the best chances to survive and more fitted to the environment. This means that selection operator always works by elitism and by seeking the best individuals for the optimization problem. There are several ways to select the parents for the next generation as it is represented below:

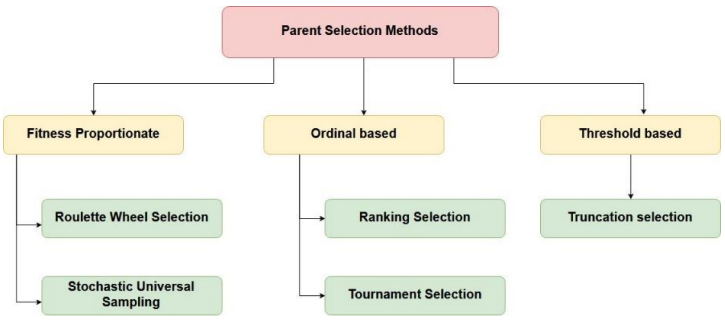


Fig 10.3. Selection Methods.

10.2.3.1. Fitness Proportionate Selection

- **Roulette Wheel**

This was the approach used by Holland's original Genetic Algorithms and it is nowadays one of the most used technique of selection. The idea is giving to every individual a chance of being selected but offering more chances to the ones with bigger fitness. There are however some limitations with this technique, especially with population with big asymmetries of fitness.

As seen before, with this technique the fittest individuals have higher chances to survive than the weaker ones. This is made by allocating each individual to a hypothetic roulette wheel where each section of the roulette is proportional to the fitness of the individual. The roulette wheel is then spun N times and the individual associated with winning section is selected as a parent of the next generation (N is defined in the parameters depending on how many parents we want in each generation).

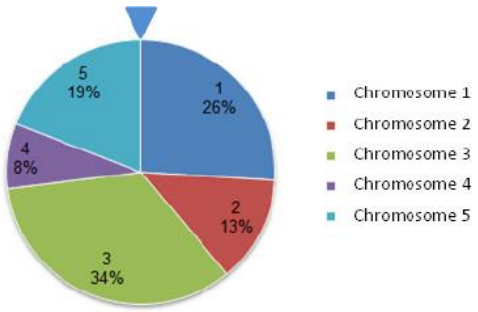


Fig 10.4. Roulette Wheel.

- ***Stochastic Universal Sampling***

This method has strong similarities with the roulette wheel, the individuals are placed in a contiguous line where each segment is proportional to the fitness of the individual. In addition, N pointers are mapped above that line according to the number of individuals we want select (e.g: if N=6 then the operator will return 6 individuals from the selection). The distance between the N pointers is given by 1/N and the first pointer is randomly chosen in a range [0, 1/N]. To better understand, assuming that in certain circumstances we want to select 6 individuals. This means that we will also have 6 pointers with a distance between each other of 1/6=0.167.

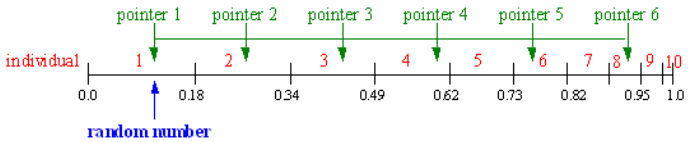


Fig 10.5. Stochastic Universal Sampling.

10.2.3.2. ORDINAL BASED SELECTION

❖ **Ranking Selection**

The Fitness Proportionate give priority to the individuals with higher fitness, which is a good thing in a certain point of view. However, in some cases where one single individual has an extremely high fitness (let's assume 80% of all roulette wheel), the selection process may suffer from early converging and inability to explore the searching space.

In order to avoid the previous situation, the ranking selection ends up as a good alternative to the fitness proportionate approaches. This method will assign a rank to each individual from the worse to the best individual (e.g the worse individual will have a ranking 1, the second worse a ranking 2, and so on until the best individual be allocated to ranking N). In this way chromosomes with a lower fitness have higher chances of being selected, avoiding the premature convergence. In another hand, this approach has the disadvantage of being more time consuming and computationally expensive.

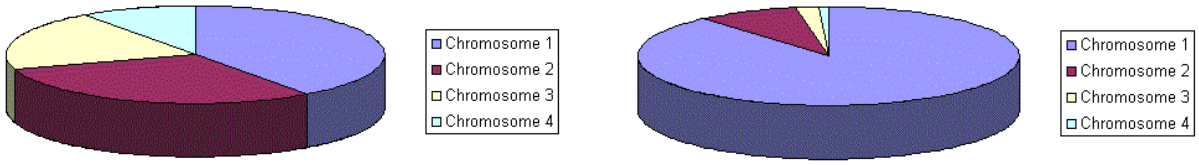


Fig. 10.6. Ranking Selection vs Roulette wheel Selection.

❖ **Tournament Selection**

If the ranking selection does not fit in our problem but we still want to use a selection method that avoids premature convergence due to extremely high fitness of some individuals, the Tournament Selection should be considered. In this approach, the selection is made by a tournament of k competitors/individuals. The winner of the tournament is selected based on the highest fitness and it is transferred to a mating pool where all the winners are allocated from all the tournaments performed so far. By changing the number of competitors of each tournament we can increase/decrease the selection pressure (e.g increasing the selection pressure is made by simply set a large number of competitors in each tournament, meaning we have higher chances to pick an individual with better fitness in each tournament).

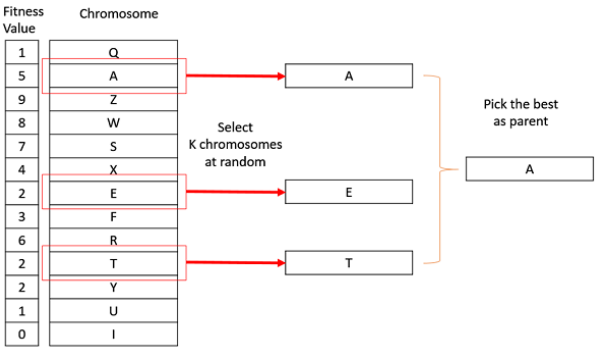


Fig 10.7. Tournament Selection with k=3.

10.2.3.3 Threshold Selection

- **Truncation Selection**

The truncation selection is the simplest technique, it is a recommended method when we are working with large population size. As the name suggest, with this approach the individuals are sorted by their fitness and a threshold is defined in the parameters. The individuals above this threshold will be selected for the mating pool and the others will be discarded.

Usually the amount of individuals selected is different than the population size, which means that the process should be repeated until the population size is reached (e.g if we always have a population size of 80 and a threshold of 25%, we will obtain 20 individuals each time. We must repeat the process 4 times to reach the 80 individuals that now represent the next generation). The threshold is often ranging between 50%-100% and, as said before, individuals below that value do not produce offsprings.

10.3 MODEL OVERALL – GENETIC ALGORITHMS

Used for: Optimization.

Advantages:

1) It searches from a population of points, instead of a single point.
2) It supports a multi-objective function.
3) Suitable for mixed discrete-continuous problems.
4) It handles well the local minimum/maximum trap.
5) Based on probabilistic rules which offers some advantages comparing to the deterministic rules.
6) It is a stochastic process and easily parallelised.

Disadvantages

1) It is computationally expensive and time consuming.
2) Designing the best objective functions according to our goals can be difficult to achieve.
3) Choosing the best combination of operators is usually a tricky part of the algorithm.

Evaluation Metrics:

1) Likelihood of optimality, Average Fitness Value and Likelihood of Evaluation Leap.

11. ARTIFICIAL NEURAL NETWORK (ANN)

The study of artificial neural network appears with the huge ambition of building a computer system that can simulate the human brain (Wang & Raj, 2017). The idea of replicate the human brain by creating an algorithm that imitates the human thinking was always one of the main goals of the researchers in this area. Despite the study of human brain has thousands of years, it was very recently when the first steps in Deep Learning took place (in this case by Warren McCulloch in 1943). However and as said before, it all started much earlier and more precisely with Aristotle (300 B.C) with the Theory of Associationism – the theory that the mind is composed by elements (like sensations or ideas) which are organized by associations.

The concept of Deep Learning was introduced for the first time in 1943, when the neurophysiologist Warren McCulloch and a mathematician Walter Pits built a computer based on the neural networks of the human brain. In order to exemplify how the human brain works, McCulloch and Pits developed an Artificial Neural Network using electrical circuits. This idea was later improved and explained by Donald Hebb in his book *Organization Behaviour* in 1949. The computational power did not follow the progress in this field of studies, the first attempt to simulate an ANN was in 1950 by IBM but it failed. Years later, Frank Rosenblatt gave a huge contribution to the ANN community with his research about the Perceptron Neural Network (which is still in use nowadays). The Perceptron is undoubtedly an extremely useful ANN, especially in binary classification, but it has evident limitations in generalizing to other scenarios as it was proven in the book "Perceptron's" by Minsky and Papert in 1969.

The credit of inventing the Back-propagation model is given to Henry J.Kelley in 1960. However, until 1985 this approach was inefficient and the first results appeared only years later. Another important input was the formulation of the *Universal Approximation Theorem*. This theorem states that for any continuous function f, there exists a feedforward neural network (with just 1 hidden layer) able to uniformly approximates f to a specific range. The first application of this theorem was made by George Cybenko in 1989 with the Sigmoid activation function.

Also, in 1989, a handwritten digit recognition using backpropagation over a Convolutional Neural Network (CNN) took place for the first time. This was the first time a modern CNN successfully occured and it was based on the idea that some brain cells respond when visualizing edges and bars. Finally, in 2006 the *unsupervised pre-training* was introduced, this method initializes an ANN based on a pre-training dataset allowing more robust results, particularly by avoiding local optima and overfitting issues.

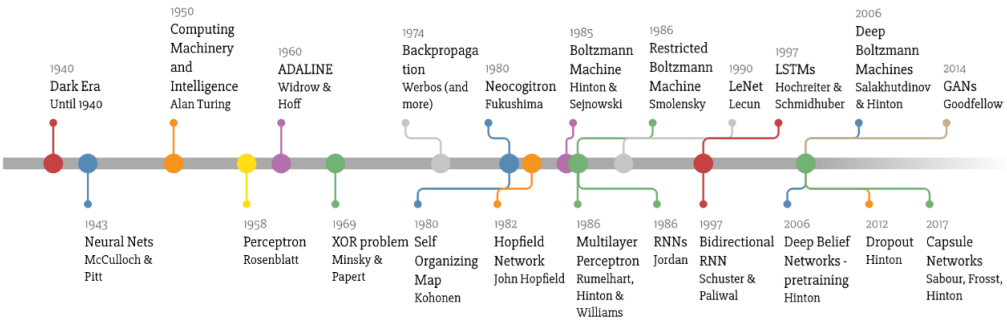


Fig 11.1. Artificial Neural Network Timeline.

11.1 MAPPING THE HUMAN BRAIN TO AN ARTIFICIAL NEURAL NETWORK

Deep Learning is irrefutably the most powerful branch of Machine Learning, specially the introduction of *Artificial Neural Networks* has revolutionized what was previously thought impossible to achieve. For the first time, with the introduction of Deep Learning, a computer is able to learn as a human being, or in other words, it is learning by example. The goal of ANN is to simulate how the human brain processes information and detects patterns. This may seem extremely difficult since biological neurons are extremely complex cells but if we focus on the essential it becomes much easier. The human nervous system is basically composed by neurons that are connected to each other by axons and dendrites, these two components are linked by a region called Synapse (Aggarwal, 2018). As it happens with biological Neural Networks, the ANN is also composed by neurons that are connected to each other by weights that play the same role of the strength of synapse.

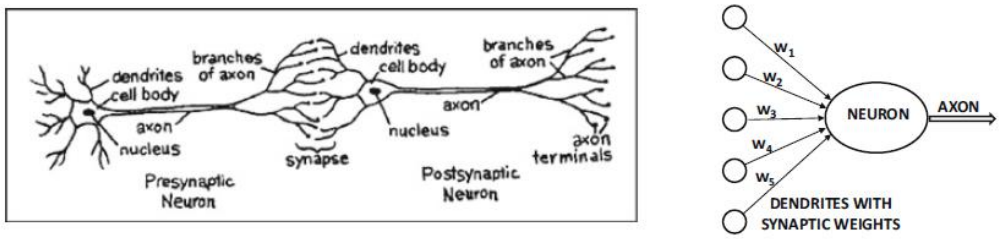


Fig 11.2 Biological Neural Network versus Artificial Neural Network.

11.2 THE BASIC ARCHITECTURE OF FEED-FORWARD NEURAL NETWORK

All the Feed Forward Neural networks share two main fundamental principles, firstly they work with input and output layers and in most of the cases also with hidden layers, secondly they are called Feed-Forward because the information always flows in that direction Let us start with the basic architecture of a simple single-layer and multi-layer Neural Network. The most famous single-layer ANN is by far the Perceptron, here the input data is directly mapped to an output layer and it is particularly useful in classification problems with only two possible output classes. A multilayer ANN uses hidden layers between the input and output layer which allow to solve more complex problems.

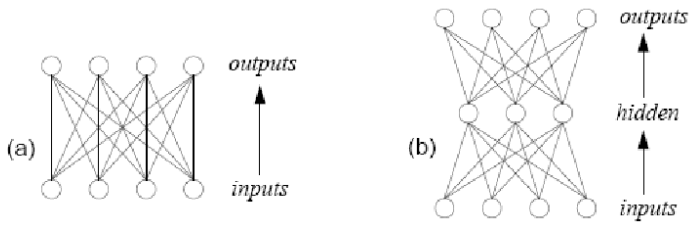


Fig 11.3 Feed-Forward Neural Network with and without hidden layer.

If we analyse the basic structure of a ANN, we will notice some common elements across any neural model (Haykin, 2009). There are 4 main elements in a neural model:

1. The first one is the *Synapse* which is represented by the weights. Each input is associated with a specific weight that is adjustable and will vary the strength of the input according to the desirable results.
2. The second one is called *adder* and it is associated with an operator that will sum all the input signals weighted by the respective synapse strength.
3. The third one, *Bias* works as an intercept of a linear equation. It is a special neuron that stores an additional value that helps the model to fit the data in the best way possible.
4. Finally, we have the *activation function* that will limit the amplitude of the neurons to a specific range.

All these elements are represented in the figure bellow:

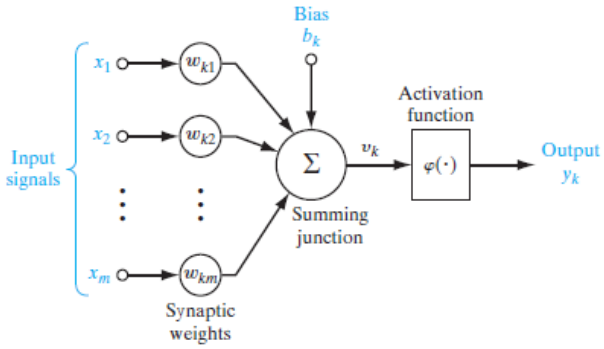


Fig 11.4. Key elements of an Artificial Neural Network.

Which can also be expressed mathematically by defining, u_k as the sum of all weighted inputs:

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

Where $x_1, x_2, \ldots, x_j$ represent all the input signals and $w_{k1}, w_{k2}, \ldots, w_{kj}$ are the weights associated to the neurons$_j$. The final output is obtained by:

$$y_k = \varphi(u_k + b_k)$$

Where b_k is the bias value and φ is the symbol of the activation function. So far we have covered the basic structure of ANN as well as the fundamental concepts associated to it. In the next chapters, it will be explained the most known ANN and their main characteristics

11.3 PERCEPTRON

Starting with the oldest ANN still in use nowadays, the Perceptron was invented by Rosenblatt in the late 50s and it is particular useful for classification problems with classes linearly separable (meaning they lie in the opposite sides of a hyperplane). The architecture of the Perceptron follows the idea described in the figure 11.4 with the exception that activation function will be in this case a hard limiter that will force the output to be either -1 or 1 (depending if the hard limiter input value is negative or positive respectively):

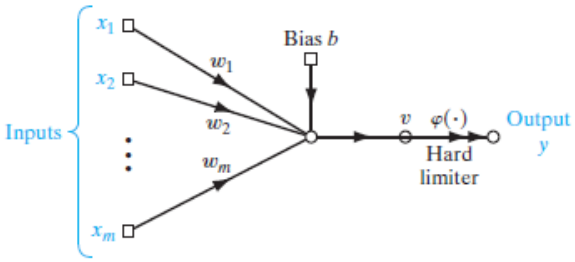


Fig 11.5. Perceptron Neural Model.

The hard limiter input value is given by v:

$$v = \sum_{i=1}^{m} w_i x_i + b$$

The idea with this hard limiter is to classify the input in two classes linear separable, C_1 and C_2. This is made by successively change the weight values until find a combination of weights that allow us to draw a decision boundary that separates the hyperplane in two regions (each one belonging to one of the classes). This interactive process of updating the weights values of Perceptron in order to find the decision boundary is known as *Perceptron Learning Phase*.

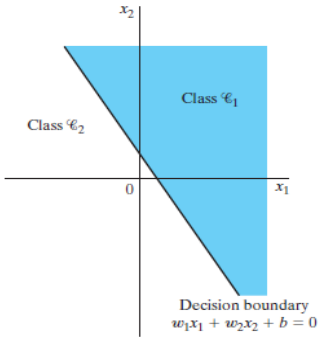


Fig 11.6. Decision Boundary representation.

The Decision Boundary is calculated by setting the hard limiter input to zero

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

This process of Learning leads us to the *Perceptron Convergence Theorem*. This theorem states that if there is a vector w_1 that gives a correct response in all training instances, then the Perceptron Learning phase will converge to a solution vector w* (not necessary unique and not necessary equal to w_1) that will also give a correct response in all training instances. If such weight vector exists, then the classes are Linear Separable.

How about the Multiclass Classification?

Until now, we have been only working with Perceptron for binary classification, where our data is partitioned in only two classes. However, with little adjustments the Perceptron can solve multiclassification problems. The idea of a Multiclass Perceptron is almost the same as the binary one with a few changes. By definition, the Perceptron can only work with a single layer which means that the multiclass perceptron must follow the architecture presented below:

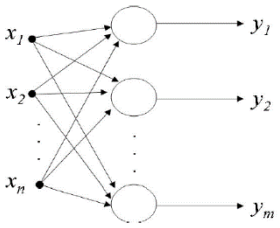


Fig 11.7. Multiclass Classification Perceptron.

In this way, the Perceptron is able to classify 2^m distinct classes. For the Learning phase, each neuron is trained independently without being influenced by others. The main goal of the learning phase is still to find a weight vector that gives correct response to all training instances and, as it happens in binary classification, this will only occur if the classes are linear separable. Meaning that the requirement of linear separability of the classes is still mandatory, otherwise the classification is not possible.

11.3.1 MODEL OVEREALL – PERCEPTRON

Used for: Classification.

Advantages:

1) It is ideal for binary classification.
2) Good generalization, it usually behaves well with unseen data.
3) Very flexible in terms of the input variables.

Disadvantages

1) For multiclass classification it is better to choose another neural network architecture.
2) The classification will only work well if the classes are linear separable.
3) Initialization-dependent, a random initialization may lead to poor results.
4) Very time consuming and computationally expensive.
5) It is a black box, meaning we rarely can understand how the results were obtained.

Evaluation Metrics:

1) Accuracy, Confusion Matrix, Precision, Recall, F-Measure and AUC-ROC curve.

11.4 XOR PROBLEM

The *XOR Problem,* also known as *exclusive or*, is a classic problem in ANN research. It uses a neural network to calculate an output based on two given inputs. Both, the input and the output argument, can either assume the value of -1 or 1 as an encoding of False and True respectively. Then the function XOR will compute a logical test that will produce true output (1) in the case that the two inputs have different values, otherwise it will return False (-1) (Rumelhart et al., 1985).

x1	x2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1

The table above contains the input pairs as well as the respective output of the XOR problem. As we can easily realise, in this case it is impossible to draw a line that linearly separates the classes which means they are not linear separable. This is one of the biggest limitations of the perceptron with only one-layer and it was a motivation to design new neural models able to solve more complex problems, particularly for non-linear separable classes. The solution for the XOR problem is to add a hidden layer to the ANN as presented below.

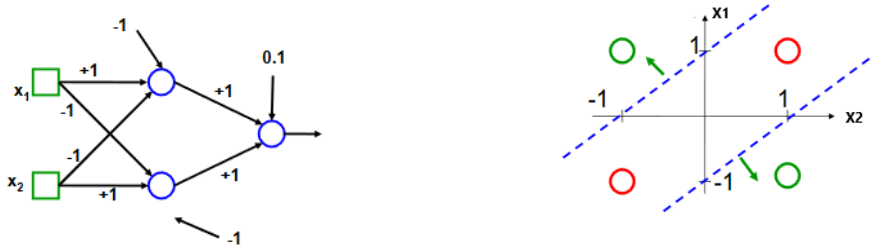


Fig 11.7. Neural network with a hidden layer able to solve the XOR Problem.

11.5 ADALINE

Despite the fact the Perceptron Convergence theorem ensures that a solution will be found if the classes are linear separable, that solution is rarely the best one available. For that reason, it is important to give a mention to the ADALINE neural network which can be seen as an improvement over the Perceptron. It was proposed by Bernard Widrow and Tedd Hoff in 1959, a little bit after Rosenblatt published his Perceptron model.

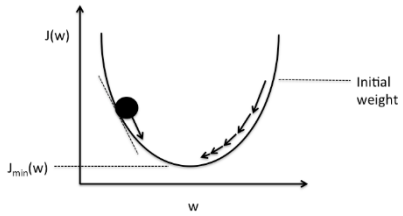
Both, Adaline and Perceptron, are single-layer neural networks and the key difference between them is how they update their weights. The ADALINE learning rule allows us to find a better set of weights and bias. Before we move forward to the learning rule it is important to remember the concepts of *Cost Function* and *Delta Rule*:

- **Cost Function**: Is one of the most used measures in supervised learning to see how well the model is able to estimate a certain event. Usually, and as it happens in ADALINE, we want to minimize the cost function.
- **Delta Rule**: looks for a vector of weights **w** that minimizes the cost function, using a method called gradient descent.

These two concepts may seem a little bit confusing, but they are actually quite simple to explain. In ADALINE, we define the cost function j to learn the weights as the sum of squared errors between the calculated outcome and true class labels (SSE). The Learning phase is then composed by calculating that difference in each interaction until finding the minimal difference that produce the best weight vector. This interactive process is called Delta Rule and can be express mathematically by:

$$J(w) = \frac{1}{2}\sum_i (y^{(i)} - \phi(z^{(i)}))^2$$

In order to find the minimal value and since the function is differentiable, we are going to calculate the gradient descent, which in this case means to calculate the partial derivative of the weights.



$$\frac{\partial J}{\partial w_j} = -\sum_i (y^{(i)} - \phi(z^{(i)}))x_j^{(i)}$$

Fig 11.8. Schematic of the gradient descendent.

The Learning phase of the Adaline become in this way an optimization problem represented graphically by a multidimensional surface, the Delta Rule will then navigate through the surface in order to find its minimum. If the surface function is unimodal, it only contains one global minimum without any local minimum. Although, some caution is required if the surface function is multimodal, in that case Adaline may get trapped into a local minimum different than the global one (Vanneschi & Castelli, 2019).

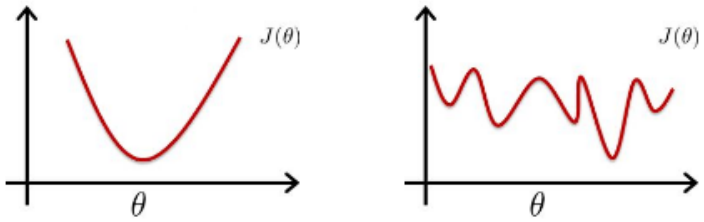


Fig 11.9. Unimodal vs multimodal surface.

11.5.1 MODEL OVEREALL – ADELINE

Used for: Classification.

Advantages:

1) It uses a Cost Function and guarantees a global optimum in the learning phase for a full optimized neural network (if and only the optimization is made in an unimodal surface).
2) It is an improvement over the Perceptron for binary classification.
3) Good generalization, it usually behaves well with unseen data.
4) Very flexible in terms of the input variables.

Disadvantages

6) The classification will only work well if the classes are linear separable.
7) Initialization-dependent, a random initialization may lead to poor results.
8) Very time consuming and computationally expensive.
9) It is a black box, meaning we rarely can understand how the results were obtained.

Evaluation Metrics:

1) Accuracy, Confusion Matrix, Precision, Recall, F-Measure and AUC-ROC curve.

11.6 MULTILAYER NEURAL NETWORK

The ANN with a single layer has sparked a huge interest in the research community. However, the paradigm changed drastically when Minsky and Papert published their book criticizing the Perceptron (and others similar ANN). They proved that a single-layer ANN has a lot of limitations, especially with the fact of being unable to solve non-linear separable problems as the XOR.

The solution for solving non-linear separable problems passes by adding more layers to the neural network which are called hidden-layers. In this way, the ANN will have neurons that are not directly linked to the output of the network. The main idea is the same as a single-layer ANN, we feed out data in the input layer but, in this case, before it reaches the output layer it will go across of at least one hidden layer. We can adjust the number of hidden layers, but usually and based on *Universal Approximation Theorem*, only 1 hidden layer is enough to approximate a function in compact subsets. The theorem was initially proved using a sigmoidal activation function in 1989 but there are more activation functions that may be more appropriate according to the problem we are trying to solve.

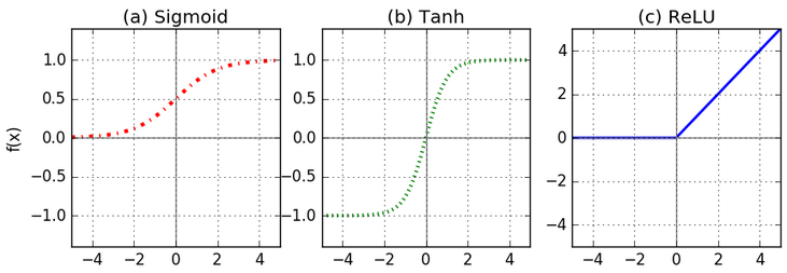
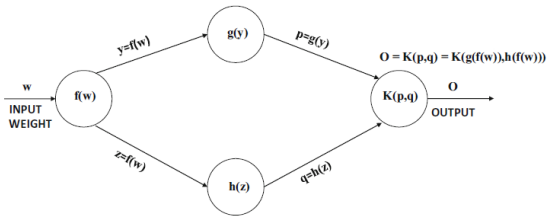


Fig 11.10. Most common activation functions.

There are several ways to implement a multilayer network but the most famous one involves the backpropagation of the errors. This is an algorithm also called Generalised Delta Rule where after each run, the network will adjust the errors in order to minimize the difference between the output and the desired values. With that in mind, the learning phase is divided in two different steps (Haykin, 2009):

1) The *forward step*, where the initial weights are set and the input signals are activated in the first layer and propagated thought the network until reach the output layer.
2) In the end of the forward phase, the total error is calculated with the difference between the network output and the desired output. That difference is called error and will be used for the *backward step.* In this phase, the ANN propagates the error signal backwards to all the layers and update their weights in order to minimize that error.

This process should be repeated several times until a termination condition is found (which can be a prefixed number of maximum interactions or a minimum value of the error). Regarding the backward step, the logic question now is how to update of the weights? The answer to that question depends on the type of the neuron. In the output neurons the adjustment of the weights is straightforward using the Delta Rule that was already explained before. But regarding the hidden neurons, we do not know the correct output so we cannot calculate the error. However, we do know that the error of a hidden neuron is defined as the sum of all the errors of all the output neurons to which is directly connected with. The solution is then to learn the gradient of the cost function associated to all the weights by using the chain rule of differential calculus as it is represented below.



$$\frac{\partial o}{\partial w} = \frac{\partial o}{\partial p}.\frac{\partial p}{\partial w} + \frac{\partial o}{\partial q}.\frac{\partial q}{\partial w} \text{ [Multivariable Chain Rule]}$$

$$= \frac{\partial o}{\partial p}.\frac{\partial p}{\partial y}.\frac{\partial y}{\partial w} + \frac{\partial o}{\partial q}.\frac{\partial q}{\partial z}.\frac{\partial z}{\partial w} \text{ [Univariate chain Rule]}$$

$$= \frac{\partial K(p,q)}{\partial p}.g'(y).f'(w) + \frac{\partial K(p,q)}{\partial q}.h'(z).f'(w)$$

Fig 11.11. Chain rule of differential calculus.

And then proceed to the respective updates of the weights of the hidden neurons. Mathematically, we define the weights updates as:

$$\Delta w_{ij} = \eta \delta_j y_i$$

Where η is a parameter from the algorithm called learning rate that will be explain later and the calculation of δ_j depends if the neuron is in a hidden layer or in the output layer. If it is in the output layer we simply apply the Delta Rule and calculate the difference between the desired value and the output from the ANN:

$$\delta_j = (d_j - y_i) y_j (1 - y_j)$$

In other hand, if it is a hidden neuron the calculation takes a little bit more time since we need more parameters. In addition to the output value y_j, we also have to discover the weight values w_{jk} (hidden neurons j connected to the output neurons k). Finally, the last parameter is the error from the output layer δ_k which is something we already know how to calculate. In the end we get the following formula.

$$\delta_j = y_j(1-y_j)\sum_{k=1}^{m}\delta_k w_{jk}$$

11.6.1 IDEAL NUMBER OF HIDDEN NEURONS

Planning the architecture of ANN is always a challenging task and at a certain point the same question always rises: how many hidden layers and how many neurons in each layer should the ANN have? After all these years, there isn't a perfect solution but there are some methods that may help in that process.

Defining the correct number of layers and neurons is a crucial moment when we are designing an ANN since it may cause overfitting or underfitting if it is not correctly prepared. As a starting point, we should have in mind these 3 rules-of-thumb (Karsoliya, 2012):

- The size of the hidden layer should be between the input layer size and the output layer size.
- The number of hidden neurons may be around 2/3 of the neurons from the input layer.
- The number of hidden neurons should not exceed twice the number of the neurons in the input layer

Of course, each problem requires a particular architecture and some adjustments may be necessary. There are 4 different methods for making those adjustments:

- The most primitive one is the **Try and Error** methodology where the developer changes repeatedly the number of hidden neurons attempting to find the best combination. This method can be divided in two possible approaches, the *forward approach,* in which it begins with a small number of hidden neurons and during the training more neurons are added. Or alternatively, the *backward approach* starts with a larger number of hidden neurons and gradually decrease them to improve the results.
- A better alternative would be the **Heuristic** approach. Since so many researchers dedicated so much effort on this subject, it would be a smart move to apply some of that knowledge and past experiences. There are already a lot of published papers related to the optima number of layers/neurons and each one with different approaches such as The Bayesian Optimization or a Genetic Algorithm over the ANN.
- Another strategy is the **Exhaustive** approach**.** As the name suggest here the idea is to try all combinations of layers and neurons. The only circumstance where this strategy may be applied is when we are dealing with a very simple and small dataset. Otherwise

it will be very complicated to evaluate the fitness evaluation due to the fact the neural network will produce different outputs depending on the initialization.

- Finally, we have the **Pruning and Constructive Algorithms** that will make a systematic search on the neural model by incrementally adding or removing neurons. One of the most knowns algorithms in this category is *Optimal Brain Damage* introduced by LeCun in 1990, which progressively removes the weights with the least impact on the ANN.

11.6.2 THE LEARNING RATE AND THE MOMENTUM

As any deep learning model, a neural network requires some caution in the moment of tuning the hyperparameters. We already know that during the training phase, an ANN uses the stochastic gradient descent to learn what is the best combination of the weights in order to obtain the minimum error. The *Learning rate* is the parameter responsible to determine how fast the ANN is learning. The Learning rate is expressed by a η and should not be too big or too small:

- If η is too small, the training is more reliable and it will produce better results but the optimization will take a lot of time because the steps in the gradient descent will be very tiny.
- If η is too big, the learning process becomes unstable because the weights changes may be so big that they never reach the minimum value of the error.

Finding the best η is a crucial moment for the success of a ANN. There are many studies about it and a large range of strategies may be applied. One of them was proposed by Sorensen in 1994 suggesting the algorithm should start with η=0.1 and then gradually decrease it in order to have a good compromise in finding the minimum error value with a reasonable amount of time.

The Momentum is another important hyperparameter, in this case to avoid the local minimum trap in a non-separable problem. This parameter is defined by α and it will be set randomly in a range between [0,1].

$$\Delta w_{ij} = \eta \ \delta \ y_i + \alpha \ \Delta w_{ij}$$

11.6.3 MODEL OVEREALL – MULTILAYER NEURAL NETWORK

Used for: Classification.

Advantages:

1) It allows backpropagation and self-optimization of the neural network for better results.
2) Ability of learning non-linear and complex relationships.
3) High performance algorithm that usually provides a good generalization with unseen data.
4) Very flexible in terms of the input variables.
5) Once the training phase is complete, the predictions are relatively easy to obtain.
6) It is possible to run the algorithm in parallel which allows a faster run of the algorithm.

Disadvantages

1) Even with the support of libraries such as Keras or Tensorflow, usually the development of this type of ANN requires some time.
2) It requires more data than usual for a satisfactory training phase.
3) Initialization-dependent, a random initialization may lead to poor results.
4) Very time consuming and computationally expensive (even with parallel computing).
5) It is a black box meaning we rarely can understand how the results were obtained.
6) It can be quite sensitive to noisy data.

Evaluation Metrics:

1) Accuracy, Confusion Matrix, Precision, Recall, F-Measure and AUC-ROC curve.

11.7 CONVOLUTIONAL NEURAL NETWORK

The Convolutional Neural Network (ConvNet/CNN) is a biologically inspired network that is used in computer vision for image classification and object detection (Aggarwal, 2018). Like others neural networks, they are composed by neurons and learnable weights and bias. ConvNets are based on the connectivity of the neurons in the human brain as well as the visual cortex and how it responds when visualizing an image.

This type of neural networks is usually used for image classification, they receive images as input and they learn the various aspects/features in order in to differentiate one from the others. Several big companies use ConvNets on their business, like Google for photo search, Facebook for automatic tagging algorithms or even Amazon for product recommendations.

The ConvNets are analogous to the multilayer ANN in the way they are composed by neurons that self-optimize through learning. Those neurons are divided in groups where each group is responsible by analysing a specific feature of the image. There are however some huge differences to the traditional ANN, starting by the types of layers.

There are 3 different types of layers in a ConvNet:

- **Convolutional Layer:** Responsible to extract the features from the original image. It will apply a scanner in the whole image (but few pixels at a time) and map the information extracted to a matrix/Kernel. There are several types of filters like edge detection, blur and sharpen. The activation function is the last component of the convolutional layer, usually the ReLu is the most used.
- **Pooling Layer:** Comes after the convolutional layer and the main task of it is to perform a down sampling of the information extracted maintaining the most essential information for the classification. A Pooling layer may alternate with the Convolutional layer several times.

- **Fully connected layers:** Have the same structure and function of a tradition ANN. At this stage, an input layer will convert the features maps into a vector, then the information from that vector goes across a certain number of hidden layer until reaching the output layer that will produce a probability to determine the class of the image.

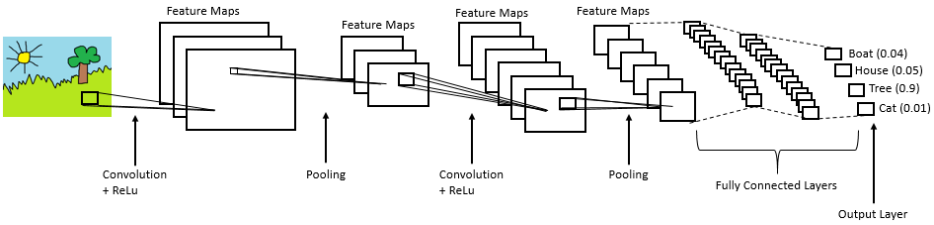


Fig. 11.12. Convolutional Neural Network Architecture.

Convolutional Layer

The main goal of a convolution is to extract features such as edges, colours or corners. This is made by convolved the original image into a kernel or a small matrix. The convolutional process is achieved with a window of pixels that will slide across the original image with the purpose of applying the filters and extracting the main features into a small matrix. The length by which the kernel slides is named *stride length*. In the example below we have a 4x4 original image that is being convolved by a 3x3 kernel with a stride length of 1.

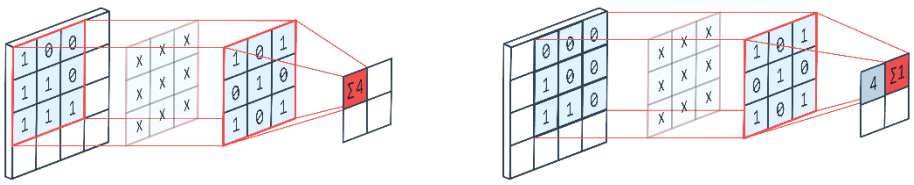


Fig 11.13. Convolution process.

It is possible to extract several features by using multiple kernels (all with the same size). The most famous kernels are the blurring, sharpening, embossing and edge detection.

Pooling Layer

A Pooling Layer comes after the convolutional layer and its main goal is to reduce the dimension of the feature matrix even more in order to decrease the computational power required and to train

the model faster. The Pooling process also prevents the situation of overfitting because it will drop the irrelevant values and maintain the most important ones. There are different types of pooling:

- Max Pooling: the maximum value of the batch is selected.
- Min Pooling: the minimum value of the batch is selected.
- Avg Pooling: The average value of the batch is selected.
- Sum Pooling: the sum of all the values from the batch is selected.

The most common ones are undoubtedly the maximum and average pooling. Just like the convolution, a pooling operator has two hyperparameters – the *stride* that sets the speed of the moving window and the *size* of the window).

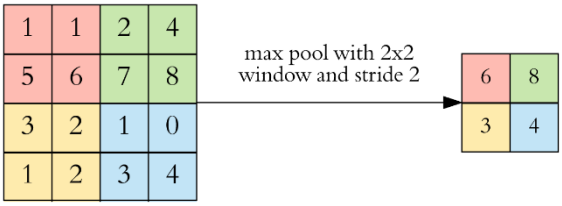


Fig 11.14 Pooling Layer representation.

Full connected layers

In the end of the Convolutional Neural Network we have a fully connected layers that will proceed for the image classification. Until now, all the work was to extract as much information as possible from the image but the classification was not done yet, that is the role of the fully connected layers. The first step here is called Flattening, the information from the image arrives at this stage in a multiple dimension state and it has to be flattened into a vector of 1D so it can be readable by the fully connected layers. Once the information is in a column-vector, the final neural network can be trained with same techniques of a normal multiple ANN, particularly using the Loss function in order to optimize the error. The last layer of the fully connected layers, and also of the CNN, is the output layer that will give a probability to determine the class of the original image.

11.7.1 MODEL OVEREALL – CONVOLUTIONAL NEURAL NETWORK

Used for: Classification.

Advantages:

1) A CNN is a specialized algorithm that rarely have competitors.
2) High performance algorithm that usually provides a good generalization with unseen data.
3) Multiple applications, from Image Classification to Speech Recognition with high demand in the real world.
4) It automatically detects the main features without human supervising.
5) It is possible to run the algorithm in parallel which allows a faster run of the algorithm.

Disadvantages

1) It is necessary to configure multiple parameters during the hyperparameter tuning.

2) A good training phase will demand a huge amount of data.
3) Very time consuming and computationally expensive (even with parallel computing).
4) It is a black box, meaning we rarely can understand how the results were obtained.
5) It requires some caution with imbalanced classes and overfitting.

Evaluation Metrics:

1) Accuracy, Confusion Matrix, Precision, Recall, F-Measure and AUC-ROC curve

---

12. RESULTS AND DISCUSSION

During this work it was presented a complete perspective of the Machine Learning models, which are now grouped by their objective task: Optimization, Regression, Classification and Clustering. Naturally, each algorithm showed its own specificity and none of them was able to fully generalize to solve any kind of problem. Despite the fact all algorithms may be useful under certain circumstances, there are significative differences among them that should be discussed.

Among the Optimization algorithms it was explained two types of algorithms: The *Iterative Local Search Algorithms* and the *Genetic Algorithms*. About the first ones, the Simulated Annealing algorithm guarantees relatively robust results with low computational power, but the Genetic Algorithms will most likely offer the best possible result. In terms of Regression algorithms, it was covered the Simple Linear Regression and the Multiple Linear Regression. They both fall in the same principle of studying the relationship between variables. The difference is that while the Simple Linear Regression only allows to predict the value of a dependent variable based on just one independent variable, the Multiple Linear Regression allows to do this analysis with multiple independent variables. If the purpose of the regression is forecasting a time series, then algorithms like ARIMA, TAR and ARCH are the most suitable ones.

Furthermore, if we are in the presence of a non-linear regression problem, algorithms such as KNN, SVM or Regression Trees should be considered. The KNN and the SVM are however more oriented to classification problems, they both belong to the family of Instanced-Based Algorithms. The performance of these two models really depends on the context, for huge datasets with low dimensional space probably the KNN fits better. The SVM would be more appropriate for a high dimensional dataset, specially if it is combined with a Kernel. Still in the same context, it is worth to mention a specific model of regression trees called CART. This model comes up as an interesting alternative for the regression problems where the relationship between variables is not linear.

For classification problems using decision trees, the ID3 and C4.5 are the most famous algorithms, both with very interesting results. Contrary to the ID3, the C4.5 handles continuous/categorical data and it solves the problem of giving unjustified favoritism to the attributes with many outcomes. From that point of view, the C4.5 is a better algorithm and can be seen as an evolution from the ID3. If the objective is to classify instances but with a probabilistic approach, the Bayesian Algorithms may be useful. These are models based on the Bayes' rule that use conditional probabilities to classify instances. There are different models of Naive Bayes such as Gaussian, Multinomial and Bernoulli but all of them have unrealistic assumptions about the independency among the predictors. Another interesting algorithm for classification is the Logistic Regression which despite its name is a classification algorithm. This algorithm is surprisingly useful for linear separable classes and can even outperform more complex algorithms.

The most complete algorithm for classification has to be the Artificial Neural Network, which does not necessary mean it is the best one for all the possible scenarios. Generally, the ANN guarantees high levels of accuracy. There are multiple ways of implementing a neural network which makes these models very versatile and therefore able of solving different problems. The Perceptron and Adaline should be considered if our problem has linear separable classes, otherwise it is better to implement a Multilayer Neural Network. In this particular case, the results may be optimized using the Back-Propagation technique. For image classification, a Convolutional Neural Network usually offers the best results and it is rarely beaten by other algorithm in terms of accuracy.

Finally, about clustering algorithms, it is important to mention some notes. The Centroid-based clustering are the most famous algorithms due to their ability to combine good results with some simplicity. They have however the limitation of being difficult to parameterize the model, particularly in discovering the k parameter related to the number of clusters. The Hierarchical and Density-based clustering should be considered as alternatives, especially the Density-based since it is the only one with techniques such as DBSCAN able to detect clusters with arbitrary shape.

13. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORK

The main goal of this thesis has always been to explain the main machine learning algorithms since explaining all the possible algorithms would be a quite impossible task to achieve. Nevertheless, in future works it would be interesting to include more algorithms in this analysis such as Association Rules, Natural Language Processing (NLP) and algorithms with Reinforcement Learning.

In terms of Limitations during the elaboration of this thesis, there isn't nothing significative to mention. The only exception was the lack of bibliography in some algorithms, particularly the Non-Linear Time Series Models which may be justified due to the low adoption of this type of algorithms. Even so, regardless the amount of existing bibliography, all models were explained in detail.

14. CONCLUSIONS

Coming back to the main research question of this thesis, it was proven not only the existence of a wide range of different algorithms, but also that none of them is better than all the others in all possible scenarios. In the introduction of this work it was proposed to prove the veracity of the *No Free Lunch Theorem* as well as to explain the main Machine Learning algorithms, allowing the reader to acquire the necessary knowledge to select the suitable model for each situation. Now, at the end of this work, both objectives were successfully achieved.

It was possible to develop a complete perspective of the Machine Learning algorithms by their objective task (Optimization, Regression, Classification and Clustering) as well as by type of learning (Supervised and Unsupervised Learning). Additionally, the full KDD process was described in detail explaining why it is so important to keep good practices during the algorithm implementation, particularly in data pre-processing, data integration, data transformation and data cleaning.

Comparatively with other similar studies, this thesis presents a well-organized structure and it is easily understandable. As mentioned in the introduction of this work, newcomers in Machine Learning will have an easier time understanding the algorithms after reading this thesis. But not only the newcomers, even for the experts this thesis may be very useful since they can remember some models and concepts in a different perspective. All the algorithms were explained with sustained theorical and mathematical foundations, alongside with detailed instructions of implementations. Moreover, in the final section of each chapter it is included a brief summary of the models with information such as Objective Task, Evaluation Metrics, Strengths and Weaknesses. This section of the *Model Overall* is undoubtedly one of the interesting features of this thesis allowing the reader to have a quick idea of each algorithm.

"Machine intelligence is the last invention that humanity will ever need to make." Nick Bostrom

15. BIBLIOGRAPHY

Aggarwal, C. C. (2018). *Neural Networks and Deep Learning* (1st ed.). Springer International Publishing.

Aggarwal, C. C., & Reddy, C. K. (2014). *Data Clustering: Algorithms and Applications* (1st ed.). CRC Press.

Agrawal, R. K., & Adhikari, R. (2013). *An Introductory Study on Time Series Modeling and Forecasting*. LAP LAMBERT Academic Publishing.

Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-Based Learning Algorithms. In *Machine Learning* (pp. 37–66). Kluwer Academic Publisher.

Aĭzerman, M. A., Braverman, Ė. M., & Rozonoėr, L. I. (1964). Theoretical foundation of potential functions method in pattern recognition. *Avtomat. i Telemekh*, *25*(6), 917–936.

Ambrose, S. A., Bridges, M. W., DiPietro, M., Lovett, M. C., & Norman, M. K. (2010). *How Learning Works* (Vol. 48, Issue 4). John Wiley & Sons.

Ankerst, M., Breunig, M. M., Kriegel, H., & Sander, J. (1999). OPTICS: Ordering Points To Identify the Clustering Structure. *ACM SIGMOD Record*.

Awad, M., & Khanna, R. (2015). *Efficient learning machines: Theories, concepts, and applications for engineers and system designers* (1st ed.).

Berrar, D. (2019). *Bayes ' Theorem and Naive Bayes Classifier*.

Boser, B. E., Guyon, I., & Vapnik, V. N. (1996). *A Training Algorithm for Optimal Margin Classifier*.

Box, G. E. P., & Jenkins, G. M. (1976). *Time series analysis, forecasting and control*. Holden-day.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees* (1st ed.). CRC Press.

Brock, W. A., Scheinkman, J., & LeBaron, B. (1996). A test for independence based on the correlation dimension. *Econometric Reviews*, *15*(3).

Burkov, A. (2019). *The Hundred-Page Machine Learning* (1st ed.). n.p.

Chen, C. (2017). *Representing Scientific Knowledge* (1st ed.). Springer International Publishing.

Cunningham, P., & Delany, S. J. (2007). *k-Nearest neighbour classifiers*.

Delahaye, D., Chaimatanan, S., & Mongeau, M. (2018). Simulated annealing : From basics to applications. In *Handbook of Metaheuristics* (3rd ed., pp. 1–35). Springer International Publishing.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From Data Mining to Knowledge Discovery inDatabases. *AI Magazine*, *17*(3), 37–63.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning* (1st ed.). Addison-Wesley Professional.

Grosan, C., & Abraham, A. (2011). Intelligent Systems: A modern approach. In *Intelligent Systems*

Reference Library (1st ed., Vol. 17). Springer-Verlag Berlin Heidelberg.

Gupta, I., & Nagpal, G. (2020). *Artificial Intelligence and Expert Systems*. Mercury Learning & Information.

Han, J., Kamber, M., & Pei, J. (2012). *Data Mining: Data Mining Concepts and Techniques* (3rd ed.). Morgan Kaufmann Publishers.

Haykin, S. (2009). *Neural Networks and Learning Machines* (3rd ed.). Pearson Education.

Hossin, M., & Sulaiman, N. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, *5*(2), 01–11.

Huang, Z. (1998). Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values. *Data Mining and Knowledge Discovery*, *2*, 283–304.

Jacobson, S. H., & Yucesan, E. (2004). Analyzing the Performance of Generalized HillClimbing Algorithms. *Journal of Heuristics*, *10*, 387–405.

James, G., Witten, D., Hastie, T., & Tibishirani, R. (2017). *An Introduction to Statistical Learning with Applications in R* (8th ed.). Springer Science.

Karsoliya, S. (2012). Approximating Number of Hidden layer neurons in Multiple Hidden Layer BPNN Architecture. *International Journal of Engineering Trends and Technology*, *3*(6), 714–717.

Kassambara, A. (2018). *Machine Learning Essentials: Practical Guide in R* (1st ed.). Statistical tools for high-throughput data analysis (STHDA).

Kaufman, L., & Rousseeuw, P. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis* (1st ed.). Wiley-Interscience.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Kramer, O. (2017). *Genetic Algorithm Essentials* (1st ed.). Springer International Publishing.

Kumar, M. S., Natarajan, V. A., & Kallam, S. (2020). *Software Engineering using Metaheuristic Algorithms* (1st ed.). Lulu Publication.

Le, C., Denker, J., & Solia, S. (1990). Optimal brain damage. *Advances in Neural Information Processing Systems*.

Maimon, O., & Rokach, L. (2010). *Data Mining and Knowledge Discovery Handbook* (2nd ed.). Springer Science.

Marcus, M. (2015). *Lecture Notes on Artificial Intelligence*. University of Pennsylvania. Retrieved from https://www.seas.upenn.edu/~cis391/Lectures/informed-search-II.pdf

Martin, B. (1995). *Instance-based Learning: Nearest Neighbour with Generalisation*. Department of Computer Science, University of Waikato.

Mccarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (1955). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence. *AI Magazine*, *27*(4), 12–14.

Melanie, M. (1999). *An Introduction to Genetic Algorithms* (5th ed.). Massachusetts Institute of Technology.

Minsky, M., & Papert, S. (1969). *Perceptrons. An Introduction to Computational Geometry*. M.I.T. Press, Cambridge, Mass.

Mitchelll, T. (1997). *Machine Learning* (1st ed.). McGraw-Hill Education.

Mittal, K., Khanduja, D., & Tewari, P. (2017). An Insight into "Decision Tree Analysis." *International Journal Peer Reviewed Journal Refereed Journal Indexed Journal UGC Approved Journal Impact Factor*, *3*(12), 111–115.

Montgomey, D. C., Peck, E. A., & Vining, G. G. (2012). *Introduction to Linear Regression Analysis* (5th ed.). John Wiley & Sons.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective* (1st ed.). MIT Press.

Osei-Bryson, K. M. (2004). Evaluation of decision trees: A multi-criteria approach. *Computers and Operations Research*, *31*(11), 1933–1945.

Poole, M. A., & O'Farrel, P. N. (1970). *The assumptions of the linear regression model*.

Privitera, G. J. (2015). *Statistics for the Behavior Sicences* (2nd ed.). SAGE Publishing.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning* (1st ed.). Morgan Kaufmann Publishers.

Rencher, A. C., & Schaalje, G. B. (2008). *Linear Models in Statistics* (2nd ed.). John Wiley & Sons.

Rodríguez, G. (2007). Logit Models for Binary Data. In *Lecture Notes on Generalized Linear Models*. Retrieved from https://data.princeton.edu/wws509/notes/

Rubinstein, R. Y., & Kroes, D. P. (2004). *The Cross-Entropy Method*. Springer-Verlag New York.

Ruder, S. (2017). *An overview of gradient descent optimization algorithms*.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). Learning representations by back-propagating errors. In *ICS report, 8506.* La Jolla, Califiornia.

Serway, R. A., & Vuille, C. (2006). *Essentials of College Physics* (1st ed.). Cengage Learning.

Shi, B., & Iyengar, S. S. (2020). *Mathematical Theories of Machine Learning - Theory and Applications* (1st ed.). Springer Nature Switzerland AG 2020.

Singh, S., & Gupta, P. (2014). Comparative study ID3, CART and C4.5 decision tree algorithm: A survey. *International Journal of Advanced Information Science and Technology (IJAIST)*, *27*(27), 97–103.

Sugihara, K. (1997). *Measures for Performance Evaluation of Genetic Algorithms*.

Terrell, C. (2019). *Predictions in Time Series Using Regression Models* (1st ed.). ED-TECH PRESS.

Vanneschi, L., & Castelli, M. (2019). Delta Rule and Backpropagation. In *Encyclopedia of Bioinformatics and Computational Biology* (1st ed., pp. 621–633). Elsevier.

Wang, H., & Raj, B. (2017). *On the Origin of Deep Learning*.

Yan, W. Q. (2016). *Introduction to Intelligent Surveillance* (1st ed.). Springer International Publishing Switzerland.

Yiu, T. (2019). *Understanding Random Forest*. Retrieved from https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Zaki, M. J., & Meira, W. (2014). *Fundamental concepts and Algorithms* (1st ed.). Cambridge University Press.