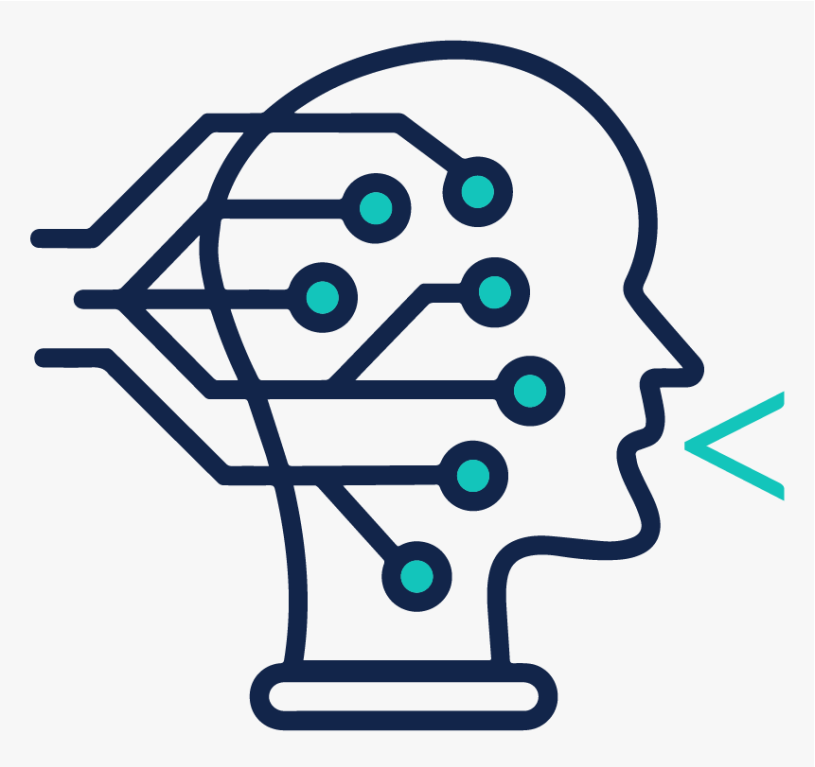
NOTES

Machine Learning

2223-1-FDS01Q002-FDS01Q002M



Vittorio Haardt
853268
mailto:vittoriohaardt@gmail.comvittoriohaardt@gmail.com
May 22, 2024

Abstract

To train the data analysis expert according to the machine learning methodology. The goal is achieved by;

- teaching how to design, develop and present machine learning projects,
- exploiting open source platforms, languages and software,
- stimulating the team working methodology.

The student will be able to design, develop, document, and present machine learning projects solving real world problems.

Contents

Contents

1 Data

In real life situations there are many different fields where the machine learning technique are largely used, like in finance, health, agriculture, e-commerce, social, chatbot and in automatic driver systems.
Understand how to treat the data is important because of the huge quantity of data available nowadays. **Machine Learning's goal is to develop a method for make the data valuables in relation a specific question that we are trying to answer.**

Typically the machine learning techniques are divide in the three following macro categories:

1. *Predictive or supervised learning:* the goal is to predict the value of a particular attribute y (target or output) based on the values of other attributes x (dependent variables or inputs). We are given a labeled set of input-output pairs (the trying set).

2. *Descriptive or un-supervised learning:* the goal is to derive patterns (correlations, trends, clusters, trajectories, associations, and anomalies) that summarize the underlying relationships in the data. They are typically exploratory tasks and frequently require postprocessing techniques to validate and explain the results.

3. *Reinforcement learning:* It is useful for learning how to act or behave when given occasional reward or punishment signals.

In this course we will focus on the firsts two types.
The *predictive or supervised learning* is divided in two sub-types:

- Classification: when there is a discrete target attribute.

- Regression: when there is a continuous target attribute.

The *descriptive or un-supervised learning* is also divided in two sub-types:

- Clustering analysis: seeks to find groups of closely related observations so that observations that belong to the same cluster are more similar to each other than observations that belong to other clusters.

- Association analysis: discovers patterns that describe strongly associated features in the data. Discovered patterns are represented as association rules or feature subsets.

In some cases the statistic correlation work very well, but unfortunately in other cases it can be even dangerous. For example if we look at the correlation between the number of homicides in the US and the number of investment funds we will see that this two values are strongly correlated.

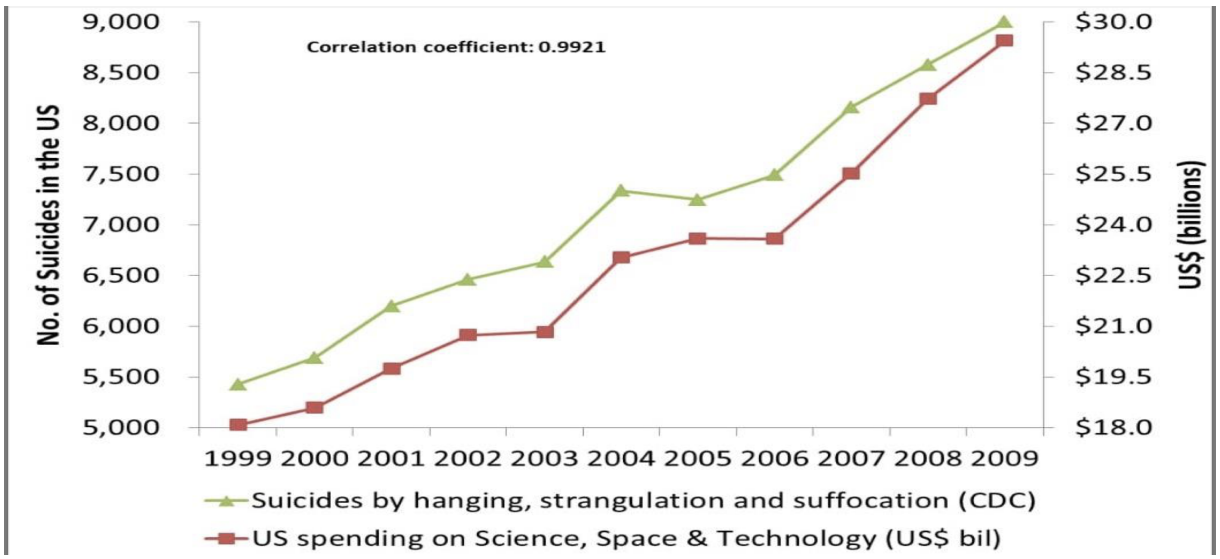


Figure 1: An example of false correlation.

This is obviously wrong and it represent the classic case of *spurious correlation.* It is famous in this topic the Simpson's paradox.
Simpson's paradox: is a phenomenon in probability and statistics in which a trend appears in several groups of data but disappears or reverses when the groups are combined.
Basically it doesn't matters how many data is available, we can not know the truth, unless we can intervene on the system.

1.1 Data types(*)

The first important step i get use to the nature of data. So we need to understand the intrinsic nature of data in order to effectively solve the problem. The data are typically organised in structures called **dataset**. The dataset are basically tables with row and columns.

Definition 1.1. We can define the **attributes** as properties or characteristics of an object that may vary, either from one object to another or from one time to another. Basically they are the dataset's columns.

Definition 1.2. We can define the **records** or **cases** or **observation** as the row of the dataset.

Each attribute is of a *type.* Knowing the type is crucial because it tells us what properties of the attribute are reflected in the values used to measure it. Knowing the type of an attribute is important because it tells us which properties of the measured values are consistent with the underlying properties of the attribute, and therefore, it allows us to avoid foolish actions, such as computing the average value of Area Code. In order to the describe attributes properties (operations) like *distinctness*, *order*, *addition* and *multiplication* are typically used, because they allow to define the four type of attributes. In particular the attributes are divided in two big groups, with some two sub-grous each:

- *Categorical (qualitative):*
  - Nominal: for example the color of eyes.

  - Ordinal: for example some judgments.

- *Numeric (quantitative):*
  - Interval: allow operations of addition and subtraction.
  - Ratio: allow all the operations.

	ATTRIBUTE TYPE	DESCRIPTION	EXAMPLES	OPERATIONS
CATEGORICAL (QUALITATIVE)	NOMINAL	The values of a nominal attribute are just different names; i.e., nominal values provide only enough information to distinguish one object from another (=, ≠).	Area Code	mode
			Churn	entropy
			State	contingency
			eye color	
			gender	
	ORDINAL	The values of an ordinal attribute provide enough information to order objects (<, >).	{bad, good, excellent}	median
			grades	percentiles
			street numbers	rank correlation
				run tests
				sign tests
NUMERIC (QUANTITATIVE)	INTERVAL	For interval attributes, the difference between values are meaningful, i.e., a unit of measurements exists (+, -).	calendar dates	mean
			temperature in Celsius or Fahrenheit	standard deviation
				Pearson's correlation
				t and F tests
	RATIO	For ratio attributes, both differences and ratios are meaningful, (*, /).	Day Mins	geometric mean
			Eve Mins	harmonic mean
			monetary quantities	percentiles
			length	variation
			electrical current	

Figure 2: Examples of different attributes for every type with relative properties.

From top to bottom the hierarchical level rises and the properties increase.

There is another method of separating attributes. It is an independent way of distinguishing between attributes by the number of values they can take:

- Discrete: that has a finite or countably infinite set of values.
  - Categorical
  - Numeric
  - Binary
- Continuous: that is one whose values are real numbers. It is typically represented as floating points variables.

1.2 Data exploration(*)

It is useful knowing how to explore data in a smart way, knowing all possible tools. We will now see some of the most important statistical tools. This tools allows a complete data exploration.

1.2.1 Definitions

Definition 1.3 (Quantile)**. Quantiles** are cut points dividing the range of a probability distribution into continuous intervals with equal probabilities, or dividing the observations in a sample in the same way. There is one fewer quantile than the number of groups created. Common quantiles have special names, such as quartiles (four groups), deciles (ten groups), and percentiles (100 groups). The groups created are termed halves, thirds, quarters, etc., though sometimes the

terms for the quantile are used for the groups created, rather than for the cut points. $q-$quantiles are values that partition a finite set of values into q subsets of (nearly) equal sizes. There are $q-1$ partitions of the $q-$quantiles, one for each integer k satisfying $0 < k < q$.

An important quantile is the one that divide the data in two half, it is called **median**.

Definition 1.4 (Mean)**.** The **mean** is defined as follows:

$$mean = \overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

The mean is not a good way for data visualisation because it doesn't tell much about the distribution. The mean is sensitive to anomalous records (*outliers*) while the median is a more robust estimate of the middle of a set of values.

In order to solve this problem we can use the trimmed mean. It is a mean with the edges values cut.

Definition 1.5 (Trimmed mean)**.** The **trimmed mean** is defined as follows:

$$mean_{trimmed} = \overline{x}_{trimmed} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad x_i = x - \max x - \min x$$

When there is a big gap between the values of the mean and the trimmed mean probably there are some outliers.

Definition 1.6 (Range)**.** The **range** is defined as follows:

$$range = \max x - \min x$$

This summary statistic is useful for measure the dispersion (*spread*) of a set of values. Unfortunately the range can be misleading if most of the values are concentrated in a narrow band of values. So for this reason variance is preferred.

Definition 1.7 (Variance)**.** The **variance** is defined as follows:

$$var = \sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

The standard deviation is preferred because, for its definition, it is on the same unit of measurement as the data.

Definition 1.8 (Standard deviation)**.** The **standard deviation** is defined as follows:

$$std = \sigma = \sqrt{\sigma^2}$$

The variance (and so also the standard deviation) depends on the mean and thus it is also sensitive to outliers. So as before there are some more robust estimates of the spread of a set of values.

Definition 1.9 (Absolute average deviation)**.** The **absolute average deviation** is defined as it follows:

$$AAD = \frac{1}{n}\sum_{i=1}^{n} |x_i - \overline{x}|$$

Definition 1.10 (Median absolute deviation)**.** The **median absolute deviation** is defined as it follows:

$$MAD = median(|x_1 - \overline{x}|, |x_2 - \overline{x}|, ..., |x_n - \overline{x}|)$$

Definition 1.11 (Interquartile range)**.** The **interquartile range** is defined as it follows:

$$IQR = q_{75\%} - q_{25\%}$$

There are some others statistic when multiple quantitative attributes are available, like covariance and linear correlation coefficient.

Definition 1.12. The **interquartile range** is defined as it follows:

$$cov(X, Y) = \frac{1}{m-1} \sum_{i=1}^{m} (x_i - \bar{x})(y_i - \bar{y})$$

When multiple Quantitative Attributes are available you usually compute the *variance-covariance matrix*. This matrix is square, symmetric and its ij element is the covariance between the ith attribute and the jth attribute.

Another measure of association between pairs of quantitative attributes which does not depend on the variance of each attribute is the linear correlation coefficient.

Definition 1.13. The **linear correlation coefficient** is defined as it follows:

$$corr(x, y) = \frac{cov(x, y)}{\sqrt{var(x)var(y)}}$$

The LINEAR CORRELATION COEFFICIENT ranges in $[-1, +1]$, the greater it is in absolute value the stronger it is the linear relationship between the two attributes.

1.2.2 Visualization

It is very useful to exploit *data visualization* (display information in a graphic or tabular format) for two main reasons.

- We can see some patterns between variables.
- We can look at the results obtained from some analyses.

We will now see some types of graphics and their characteristics.

Histogram is a plot that displays the distribution of values for attributes by dividing the possible values into bins and showing the number of records that fall into each bin. In histogram for quantitative attributes each bin is an interval of values, bins can have the same width or not.

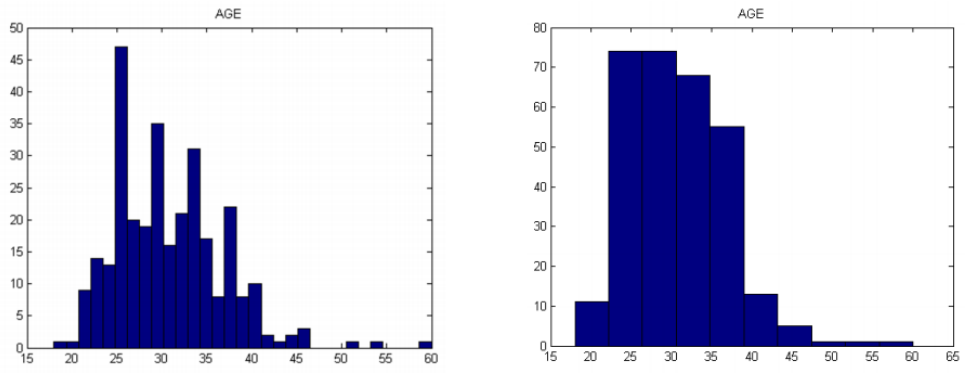


Figure 3: Differences between two histograms on the same data distribution, but with different bins.

It is also possible to create histograms for qualitative attributes. The main different is that each bin is associated with a value, when values are too much they are aggregated to possibly form meaningful bins.

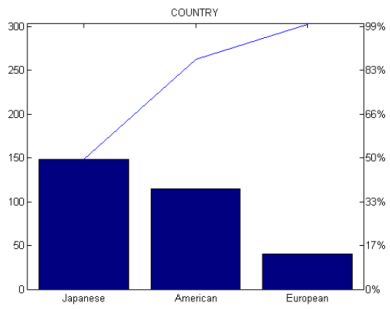


Figure 4: Histogram on qualitative attribute.

In this case the blue line is a cumulative line. It represent the level of the sum between the data so far, it obviously end at 100%.

Another useful way for data visualisation is the **Box and Whiskers** (Box plot). It can be applied only on quantitative attributes. This graph allow to get more information that the histogram. We can now see a box plot where are showed all the obtainable information.

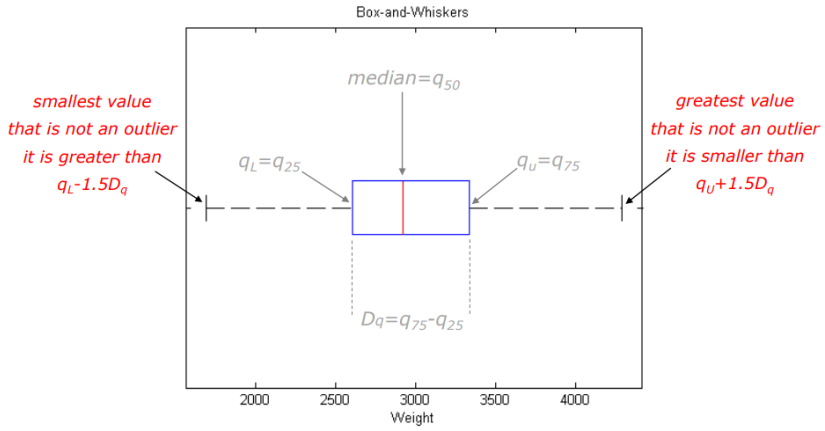


Figure 5: Box plot's obtainable information.

1.3 Missing replacement(*)

For *missing replacement* it is meant the study and the replacement of some attributes' missing records in a dataset. We will see the the basic things about this huge argument, in order to be able to do an effective data analysis.

Multiple causes are responsible for missing data or missing observation:

- The attribute's value is not always measurable/observable.
- The attribute was non considered to be relevant when first collecting the data.
- Misunderstanding concerning the guidelines to save the data.
- Bad functioning of measuring and/or of saving devices.
- Inconsistent values with respect to other attributes.

The missing data or missing observations problem must be addressed, if any, before starting any Data Mining study. There are different methods to address this problem, we will now see the most used:

- **Record removal:** it is the most drastic method, it means to brutally remove all the records that contain a missing value. It is used a few times because it comes with the risk of losing some records fundamental to the analysis.
  Basically if the probability that an attribute has a missing data is $p(NULL)$ *and it is a record with* m *attributes then the probability that it hasn't missing data in that record is* $(1 - p(NULL))^m$
- **Manual imputation:** it is to replace the missing values manually with some observations. It is extremely time consuming.
- **Global constant:** it is to replace the missing values with a constant value called *place holder* (usually 999). This method is not very effective because a constant value can not be represented in a distribution.
- **Mode/mean replacement:** it is to replace the missing values with the mode of that attribute, if the attribute are not continuous, in the other case we use the mean. It is not very effective because it strongly reduce the variance of that attribute.
- **Conditional mean replacement:** it is to replace the missing values with the mean if it is true a condition on another attribute. It is slightly more efficient that the other methods but it also ave some problems.
- **Most probable:** it is to do a regression on the dataset using other attributes and placing the missing data as the target variable. More than one Attribute can be used with the linear regression model, more complex models can be used, and it is possible to manage the case of Qualitative Attributes also, appropriate models.

It is now clear that the line between data exploitation and data modelling is very thin.

1.4 Data Preprocessing(*)

Preprocessing or data processing is a broad area and consists of a number of different strategies and techniques that are interrelated in complex ways. It is devoted to **make the data more suitable for data mining**. We will now see some of the most famous techniques of preprocessing.

1.4.1 Aggregation

Aggregation consist of combining two or more records into a single object (record). In order to manage qualitative attributes it can be omitted or summarized as the set of all values it takes for any given value of the aggregating attribute. There are many advantages reachable with data aggregation:

- *Smaller data sets:* we need less memory and processing time, thus more time consuming data mining algorithms can be used.
- *Change of scale/scope:* provides high-level view of data instead of low-level view.
- *Reduced variance:* attributes computed on aggregated records are more stable (reduced variance for quantitative attributes, reduced entropy for qualitative attributes) than those associated with the native records. A drawback consists of loosing interesting details in the data. For example aggregating sales volumes from days to weeks lost the information about which day of the week has the highest sales volume.

1.4.2 Sampling

In many case having huge quantity of data can represent a drawback for the computation, this because in order to use those quantity of data we have to use more simple algorithms. It is better to use less data but a better algorithm. A lot of attention must be payed to sampling. Indeed, the key principle for effective sampling is the following: *Using a sample will work almost as well as using the entire data set if the sample is representative.*

Definition 1.14 (Representative sample)**.** A sample is said to be **representative** when it has approximately the same property (of interest) as the original set of data, i.e., the original data set.

We must choose a sampling scheme that guarantees a *high probability* of getting a representative sample. This involves choosing the appropriate:

- Sample size
- Sample technique

Many sampling techniques exist but only the most basic ones will be presented here:

- **Simple Random Sampling:** each record of the data set has the same probability to be included in the sample. It can be done in two ways:
  - **Without replacement:** each selected record is removed form the data set.

  - **With replacement (Bootstrap samples):** each selected record is not removed form the data set.

  When samples are small compared to data set size the two sampling strategies generate samples which are not much different. When the data set consist of qualitative attributes such that the possible values they can take have highly different frequencies, this technique can fail to adequately represent those type of values which are less frequent.

- **Stratified Sampling:** the samples are created with records for emulate in the better way possible the proportion between attributes in the starting dataset. It also can be done in two ways:

  - **Equal number**: were the same amount of records is sampled form every group.
  - **Proportional**: it can be *proportionate allocation* that uses a sampling fraction in each of the strata that are proportional to that of the total population; or it can be *optimum allocation* (or disproportionate allocation) where the sampling fraction of each stratum is proportionate to both the proportion (as in proportionate allocation) and the standard deviation of the distribution of the variable. Larger samples are taken in the strata with the greatest variability to generate the least possible overall sampling variance.

  If measurements within strata have a lower standard deviation (as compared to the overall standard deviation in the population), stratification gives a smaller error in estimation. For many applications, measurements become more manageable and/or cheaper when the population is grouped into strata. When it is desirable to have estimates of the population parameters for groups within the population - stratified sampling verifies we have enough samples from the strata of interest. If the population density varies greatly within a region, stratified sampling will ensure that estimates can be made with equal accuracy in different parts of the region, and that comparisons of sub-regions can be made with equal statistical power.

Once a sampling technique has been selected, it is still necessary to choose the sample size. Having a *large sample size* increase the probability that a sample will be representative, but it also eliminate much of the advantage of sampling. Having a *small sample size* may allow to miss patterns or to detect erroneous patterns. We need to chose the right sample size in relation with the dataset, may be by doing some training.

1.4.3 Dimensionality reduction

In many cases we have to analyze data sets characterized by an high number of attributes. Reducing the number of attributes has several **advantages**:

- Many data mining algorithms work better if the dimensionality (number of attributes)is lower (irrelevant attributes are removed while noise in data is reduced).
- Interpretability of the developed model is increased, it depends on a lower number off attributes.

- Graphical representation of data is facilitated.
- Amount of time and memory required bu data mining algorithms is reduced.

Many types of data analysis become significantly *harder as the dimensionality of the data increases*. As dimensionality increases, the data become increasingly sparse in the space it occupies.

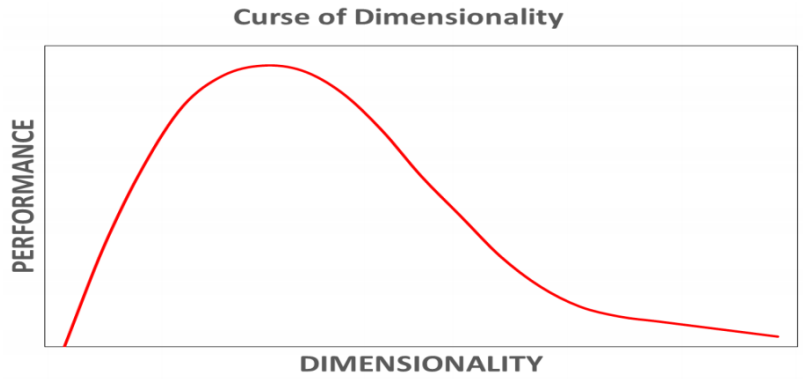


Figure 6: Performance trend of an algorithm as a function of the dimensionality of the dataset.

Some of the most common approaches for dimensinality reduction, particularly for continuous attributes, use techniques form linear algebra to project the data from a high-dimensional space into a lower-dimensional space.

The **Principal Component Analysis (PCA)** finds new attributes (principal components) that:

- Are *linear combinations* of the original attributes.
- Are *orthogonal to each other*.
- Capture the *maximum amount of variation* in the data.

We are usually asked to specify the number of principal components to retain or the percentage of variation we want to explain. In particular **Singular Value Decomposition (SVD)** is a linear algebra technique that is related to PCA and it is also commonly used for dimensionality reduction.

Definition 1.15 (Binarization)**. Binarization** is defined as a technique of transformation continuous and discrete attributes into one or more binary attributes.

In order to binarize an attribute we associate k possible values to k integer values int the interval $[0, k-1]$. If the attribute is ordinal, then the order must be maintained by the assignment. This transformation is required also if the attribute is represented by integers, in the case where such integers are not in the interval $[0, k-1]$. Than we convert the k integers to a binary number. The number of digits necessary to represent k integers is:

$$s = \lfloor \log_2 k \rfloor$$

It may be the case that only the presence of the value 1 for a binary attribute is important. It is necessary to introduce one binary attribute for each value that the categorical attribute can take on.

When we have attributes that are used in classification or association analysis, we typically use **discretization**. As is easily understandable the beast discretization depends on the algorithm being used, as well as other attributes to be considered. Typically, the discretization of an attribute is considered in isolation. The discretization can be:

- **Supervised.**
- **Unsupervised.**

The *supervised discretization* exploits additional information (class attribute) to discretize the continuous attribute. This technique places split points in such way that some measure of *purity* of the resulting intervals is maximized, the purity measure is computed exploiting the class attribute. **Entropy** is usually computed as a measure of purity of interval.

$$e_i = -\sum_{k=1}^{K} p_{ki} \log_2 p_{ki}$$

It associated with the ith interval, if it:

- contains only records of a given class $\implies$ $e_i = 0$, it means maximum purity;
- contains equally often all classes $\implies$ $e_i = \max$, it means minimum purity.

Supervised discretization based on entropy allow to find the *continuous* attributes' division points such that the total entropy is *minimized*.

Definition 1.16 (Entropy)**.** Entropy is defined as:

$$E = \sum_{i=1}^{n} w_i e_i$$

Where:

$$w_i = \frac{m_i}{m}$$

And the variables are such that:

- n: number of intervals.
- m: number of records.
- m_i: number of records in interval i.

The *unsupervised discretization* does not exploit any information except the values of the continuous attribute to be discretized. They can be divided in two category:

- **Equal width unsupervised discretization:** all the interval, in which the attribute is discretized, has the same width.
- **Equal frequency unsupervised discretization:** all the interval, in which the attribute is discretized, has the same frequency.

Sometimes categorical attributes can have to many values. If the categorical attribute is *ordinal*, then techniques similar to those for continuous attributes can be used to reduce the number of categories, but if the categorical attribute is *nominal*, then other approaches are needed. In this second case usually it is preferred to exploit domain knowledge to generate the class, but if it is not available we can only group the values if the grouping lead to a improvement of classification performance or of some goal in data management.

1.4.4 Variable transformation

Definition 1.17 (Variable transformation)**.** A **variable transformation** is defined as every transformation that is applied to all the values of a variable.

There are two types of variable transformations:

- **Simple functions**: a simple mathematical function is applied to each value individually. It is important to pay attention to the order of application and to the behaviour for negative values and near 0 values.
- **Normalization or standardization:** it transforms entire set of values to have a particular property.

  $$Z = \frac{x - \mu}{\sigma}$$

  It is important because the transformation is such that $\mu = 0$ and $\sigma = 1$. In this way the sum of different continuous attributes avoid one or few attributes taking large values to dominate the new attribute sum. The same applies to other possibilities to combine attributes.

  Because of the estimator of mean and standard deviation are strongly affected by outliers, it is possible to modify the standardization in order to see them better. In this case the mean is replaced by the median, and the standard deviation is replaced by AAD.

2 Classification

2.1 Introduction (*)

We will now focus on the validation of supervised model classification. The most important thing is **to understand what we are doing**, because it is easy to get lost while using machine learning algorithm.

We will now use a dataset for example, in order to better understand. The dataset is about churn data and we want to understand if clients will leave the service or not. To achieve this information we use a particular combination of attributes. First of all we want to know variables types.

Account Length	VMail Message	Day Mins	Churn	Intl Calls	Intl Charge	State	Area Code	Phone
128	25	265.1	n	3	2.7	KS	415	382-4657
107	26	161.6	n	3	3.7	OH	415	371-7191
137	0	243.4	n	5	3.29	NJ	415	358-1921
84	0	299.4	y	7	1.78	OH	408	375-9999
75	0	166.7	n	3	2.73	OK	415	330-6626
118	0	223.4	n	6	1.7	KS	510	391-8027
121	24	218.2	n	7	2.03	MA	510	355-9993
147	0	157	n	6	1.92	MO	415	329-9001
117	0	184.5	n	4	2.35	KS	408	335-4719
141	37	258.6	n	5	3.02	WV	415	330-8173

Figure 7: Churns' data table.

The goal is to be able to predict when a client could drop a service, and to understand the reasons of this choice. This type of tasks can be resolved through a **classification model**. A classification model is a model that use some of the data set's attributes (**explanatory attributes**) in order to predict another attribute's value (**class attribute**).

Figure 8: Classification process

We'll now see some important definition which will help us to explain better the classification models:

Definition 2.1 (Explanatory attributes)**.** The **explanatory attributes** are the input variables.

Definition 2.2 (Class attribute)**.** The **class attribute** is the output variable.

Definition 2.3 (Classification model)**.** A **classification model** is a model that solves the *classification task* or *classification problem*. It is useful for the following purposes:

- **Descriptive modeling**: can serve as explanatory tool to distinguish between objects of different classes

- **Predictive modeling**: predicts the class of an unknown record. It can be treated as a *black box* that automatically assigns a class label when presented with the attribute set of an unknown record.

Definition 2.4 (Classifier)**.** A classification technique (**classifier**) is a systematic approach to building classification models from a data set.

The building of a classifier follows the following steps:

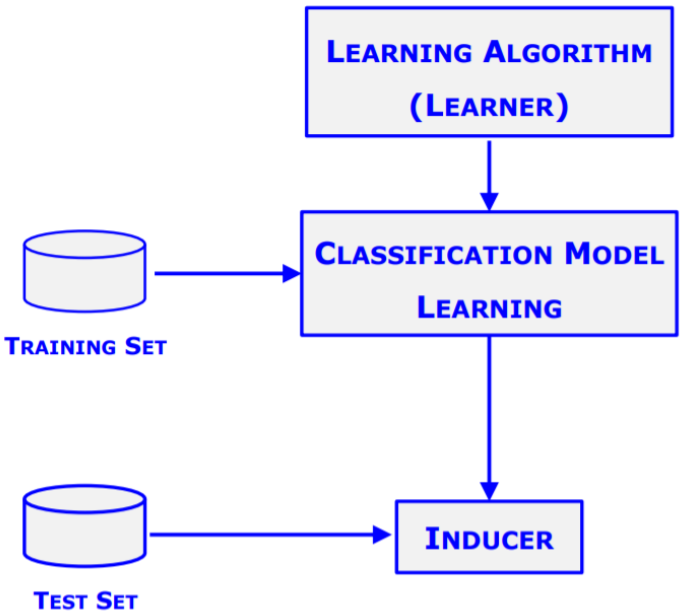


Figure 9: Classifier process.

Definition 2.5 (Training set)**.** The **training set** consist of records whose class values are known.

It is important to provide the training set, because it is used to build (learn and validate) a classification model, which is subsequently applied to the test set.

Definition 2.6 (Test set)**.** The **test set** consist of records whose class values are unknown (assumed to be such).

The classification model learning take place using the learning algorithm (**learner**) on the training set.

Definition 2.7 (Inducer)**.** The **inducer** is the output of the classification model learning, it is an instance of the classification model.

The inducer is substantially queried to predict the value of the class attribute for the test set records. The key point is the choice of the learner algorithm. Also the choice of the right explanatory attributes is crucial, because some of them could only cause rumor. Through performance analysis we will be able to assess this two choices.

Performance evaluation of a classification model is based on the counts of test records correctly and incorrectly predicted by the inducer. These counts are tabulated in a table known as **confusion matrix**.

		Inducer Prediction (IP)	
		-1	+1
Actual Class (AC)	-1	TN	FP
	+1	FN	TP

Figure 10: Example of confusion matrix.

On the inside the test set's results are classified like that:

- **TN - true negative**: number of records where $AC = -1$ which are correctly classified $IP = -1$
- **FN - false negative**: number of records where $AC = +1$ which are wrongly classified $IP = -1$
- **TP - true positive**: number of records where $AC = +1$ which are correctly classified $IP = +1$
- **FP - false positive**: number of records where $AC = -1$ which are wrongly classified $IP = +1$

The Confusion Matrix contains the information needed to evaluate the performance of the Classification Model. Summarizing this information with a single number would make it more convenient to compare the performance of different classification models, this number is a performance metric.

Definition 2.8 (Accuracy)**.** The most used performance metric is **accuracy**, which is defined as follows:

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

Basically the accuracy is the *number of correct predictions* on the *total number of predictions*. $acc \in [0, 1]$ A complementary view is provided by using as performance metric the **error**, which is defined as follows:

$$error = \frac{FN + FP}{TN + FN + FP + TP} = 1 - accuracy$$

The error is the *number of wrong predictions* on the **total number of predictions**.

It is possible to use another one coefficient that is basically an adjusted form of accuracy. It is generally thought to be a more robust measure than simple percent agreement calculation.

Definition 2.9 (Cohen's kappa)**.** Cohen's **kappa** measures concordance between observed and estimated target (accuracy) controlling for random agreement by chance (expected accuracy, E_A the expected accuracy under chance agreement).

$$Kappa = k = \frac{acc - E_A}{1 - E_A}$$

$k \in [-1, 1]$

2.2 Classification techniques

A classification technique (**classifier**) is a systematic approach to building classification models form a data set. Classification techniques can be grouped into four groups:

- **Heuristic**: neighbour inspection, widely used like: Decision Trees, Random Forest, Nearest Neighbor
- **Regression Based**: use a paramedic conditional probability, like: Logistic Regression
- **Separation**: partition the attributes' space, like: Support Vector Machine, Artificial Neural Network
- **Probabilistic**: exploit Bayes formula and compute posterior probability to classify records, like: Naïve Bayes, Tree-Augmented Naïve Bayes.

We provide a brief overview of some of this techniques. We will start with **Heuristic**

2.2.1 Decision Tree

Decision tree models are heuristic models.

A **decision tree** has a specific graphic representation where the main element are nodes. **Nodes** are subset of the dataset made on a specific condition.

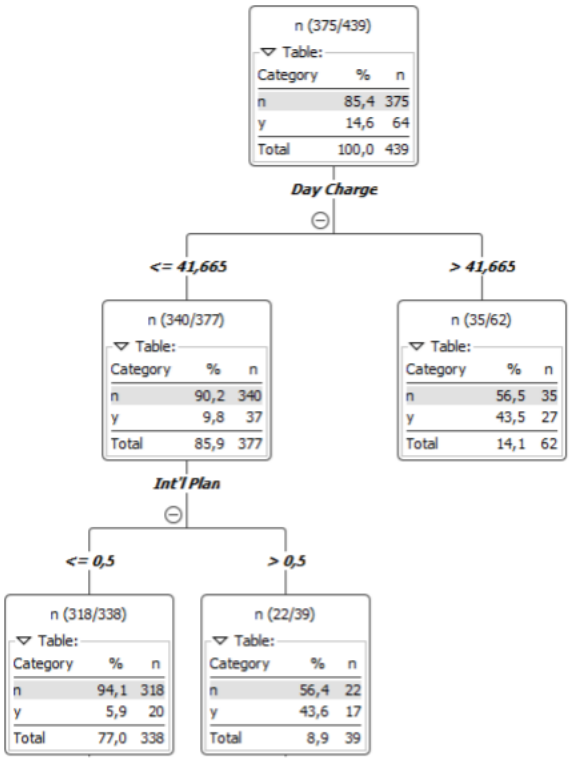


Figure 11: Decision tree model

The the main element of a decision tree are:

- **Root node**: has no incoming edges and zero or more outgoing edges

- **Internal nodes**: each of which has exactly one incoming edge and two or more outgoing edges
- **Leaf or terminal nodes**: each of which has exactly one incoming edge and no outgoing edges.

Each record is classified form **top** (root) to **bottom** (leafs).

Starting from the first node (root) in which there is an attribute's indication: in the example 'day charge' could be $\leq 41,665$ or $>$ and it separate the tree in two nodes. If the answer is true we will follow a node, if it is not true we follow the other one. Following this path lead to a leaf node, and so we can assess a variable. When we arrive to a leaf we have to count the most frequent class and give that to the variable (ex. churn = n).

So we want to understand which are the most relevant attributes that can split the nodes. Decision tree is classification model because gives the **posterior probability** for every observation and on the most frequent attribute in the leaf nodes it change to the expected probability for the target. Decision tree is an useful tool, easy to represent, it gives the attributes importance giving levels for variables that split the observations. Furthermore it gives clear classification rules and it can handle qualitative and quantitative target without any problem.

So the decision tree's characteristic are:

1. Understanding the set of split admissible on attributes for each nodes. It is important to understand why a variable split and another doesn't, why a variable split before another one. Understanding the spilt means understanding the generation criteria of two or more nodes form a starting node.
2. Which is the most used splitting criteria. We well see some metric useful to understand why a variable split before another one.
3. Deciding the **stopping rule** for choosing if a note is a leaf or not. This is what we need to *tune*.
4. Evaluation of the decision rule. Knowing a classification's risk of error.

The first 3 points are addressed in different ways by each segmentation algorithm.

The decision tree can be used for nominal attributes, ordinals as well as numerical ranges and coefficients.

Several measures can be used to select the optimal node splitting rule: *entropy*, *Gini index* and the *classification error*. However, it also depends on the type of attribute: *binary*, *nominal* and *continuous*. It is a model that always give as output *the most frequent class and is therefore useless when the dataset has particularly unbalanced classes*. They are obviously univariate tests, what we do is building hyper parallelepipeds of our dataset and drawing straight lines for the change of classification: **decision boundary**.

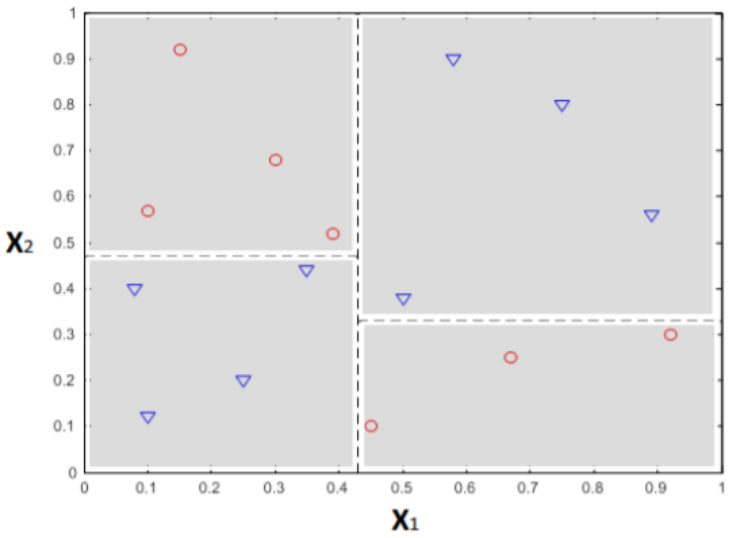


Figure 12: Decision boundary

The result of these straight lines makes the difference on the ability to highlight where elements of a certain class are present, so we have to learn how to position these hyperplanes. **The goal is to partition my space into hyperplanes maximizing accuracy**.

It is also possible to do multiple splitting and not just binaries, it can be done it on nominal values as well.

Now lets see three of most commonly used methods:

	CHAID	C5.0	CART
Split Type	Multiway	Multiway	Binary
Continuous Target	No	Yes	Yes
Continuous Inputs	No	Yes	Yes
Splitting Functions	Chi-Square	Entropy	Gini

The *Cart* approach is the most used one and it is also the most versatile one. The only limit is that a node can only split in two, while in with *Chaid* multi-split are possible. The **Cart** (Classification and Regression Tree) is used for both descriptive and classification scopes:

- **Classification Tree**: it identify groups of subjects who are *internally homogeneous and externally heterogeneous* with respect to a qualitative response variable with j levels (classification/segmentation)
- **Regression Tree**: it identify groups of subjects that are *internally homogeneous and externally heterogeneous* with respect to a quantitative response variable (prediction).

In order to make the splits is important to know the type of attributes in the dataset, because depending on the type the number of split can change.

Attribute type	Modes number	Split number
Continuous	N	$N-1$
Binary	2	1
Ordinal	m	$m-1$
Nominal	m	$2^{m-1}-1$

Now we will rapidly examine the different splitting algorithms in order to understand the pro and cons of everyone.

- **Cart** → The initial set (first node) must be divided into two nodes which are as homogeneous as possible internally and as heterogeneous as possible (with respect to the target). A split criterion must be based on a statistical index that allows selecting the best partition among all possible binary partitions of an attribute (*cutoff*) and among all possible attributes. Cart uses a selected attribute to split a node in two, it use the **maximum impurity increment**. Because of that we need to use an index to calculate how much purity or homogeneity exists in any node. This measure in Cart is the **Gini heterogeneity index**.

$$Gini(t) = 1 - \sum_j f_j^2(t)$$

  The goal is to generate two node that are more pure that the first one. Note that the index of heterogeneity is an index of variability of the target. Cart calculates all the $\Delta Gini(s,t)$ and selects the split that maximizes the impurity decrease from node t. The index is 0 when you have all the observations in a node that are distributed in a mode of the target. As it grows, the heterogeneity of the target increases. We will therefore have to evaluate how various splits can generate nodes that are maximally pure and therefore with very low index values. There are two way to measure the splitting's impurity, :

  - **Average impurity of split nodes**: it is the arithmetic mean of how impure the two leaf nodes are.
  - **Impurity Decrement/Gini index**: it is the one used by Cart, it is the difference between the impurity of the starting node and the arithmetic mean of the impurity of split nodes. The split is considered good if the $\Delta Gini(s,t)$ value is large.

  Once the first split has been created, from the nodes created the algorithm starts over, considering each split node as a new node for generating a split and restarts the search for the best split.

  If the target is continuous the procedure is pretty much the same, the only change is that the impurity is replaced with the **variance**. The variance of the target observed in the various nodes coincides with the heterogeneity index of a qualitative target. The variance whit in groups is the best option for continuous targets.

- **Chaid** → This algorithm is a mechanism that use a *frequency table*: in the row we have the target and in the columns we have the attributes that has been binarized by the split itself. In a table of this type it is possible to generate another measure for the goodness of the split: the **chi-square**. The goal is to have zeros in the frequency table, because the bigger this value is, the more discriminating the split will be. It is important to pay attention to the **p-value** associated with chi-square. The main problem is that after many splits the p-values will be very small and difficult to distinguish. To solve this problem instead of the p-value it is used a measure called **logworth**, defined as:

$$Logworth = \log_{10}(p - value)$$

The main problem with Gini and chi-square is that attributes with more classes are favored for the split than attributes with fewer classes. The chi-square is even more afflicted with this problem then Gini index. In Chaid the chi-square increases as nodes increase, i.e., the p-value will be smaller; therefore, it will select the attribute with the highest number of levels in the first divisions. Theoretically, the p-value could be adjusted, while Gini based impurity measures (CART) cannot be adjusted to account for this problem. If we notice that an attribute is used many times for the splits, may be we should use a multi-way classifier.

- **Error rate/misclassification rate** → This algorithm generates (like a Chaid) a frequency table. The frequency table it generates is the **confusion matrix**. The metric that allows us to judge this classification is the **error rate**. Maximum purity means having maximum accuracy or minimum error rate. Error rate is the weighted average of the line errors, weighted for the priors. The best split is the one with the lowest error rate.

- **C4.5 or J48** → algorithm that respect uses entropy to evaluate splitting. Entropy is a logarithmic measure of the Gini heterogeneity index.

$$h = -\sum p(x) log p(x)$$

  This algorithm is very popular because it adapts to all kinds of situations.

- **C5.0** → C5.0 set of rules is an addition of the C4.5 process that too extends ID3. This is a good categorization algorithm for large databases C4.5 faster than memory and performance. C5.0 Model by maximum weight training data records. C5.0 handles missing attributes from very valuable attribute and pest training records. In this re- search, preparation pest information is charity intended for predicting pest information.

Another important thing for the decision tree is selecting the **stopping rule** to prevent the model form *overfitting*. The stopping rules define the rules for managing the complexity of a tree in an effective point. There are basically to stopping rules:

- **Pre-pruning**: rules that established before generating the tree. Before generating the tree it is possible to decide one of this rules:
  - *Dimension of leafs* (n): the minimum of element in the leafs
  - *Minimum number of observation in a node to have a split* (k): k is a multiple of n
  - *Maximum depth of the tree.*

  The most common is choosing the dimension of leafs.

- **Post-pruning**: rules that established before generating the tree. They allow the decision tree to grow in depth by pruning the branches that do not generate further discrimination according to various criteria:
  - *Error rate*
  - *Profit and loss*
  - *Average Square Error* (ASE)

The concept is to prune of final nodes that are not significant in terms of explanatory power. It means generating a decision tree by evaluating how the error metrics change.

One effective way for generating a decision tree is to use more post-pruning. The advice is to use lapse criteria for pre-pruning to obtain a tree with many leafs. Then study the overfitting and chose the number of leaf with the minimum ASE.

After the generation of a decision tree it is possible to assess the quality of the classification and the robustness of the tree. To achieve this result we can use *ASE train vs ASE validation* or **accuracy train vs accuracy validation**, in general the first one is better. These measurements are used to evaluate a possible overfitting even if it is a very expensive point and therefore we trust the software procedure. If the percentage difference between the two measures is more than 10% there will be overfitting.

Now that we have seen how to use the decision tree at their better, we will rapidly focus on advantages and disadvantages of this type of model. The *advantages* are:

- Decision trees have easy readability of classification rule.
- Decision trees acts as model selection (selection of more discriminating variables, with criteria non-parametric selection methods)
- They are not sensitive to scale differences, nor to out-of-scale values (Not sensitive to skewed distribution or outliers) nor to asymmetric distributions.
- They are resistant to missing data: when the missing data are categorical attributes trees create a new category for missing, when they are quantitative attributes use "surrogate split" supplementary variables that replace the missing case variable with one "similar" to it.
- Trees give us a list of surrogate attributes. A variable is a surrogate for another one if it like in the same way.

So decision tree doesn't need any type of preprocessing. The main **disadvantages** of decision tree is:

- Decision trees are unstable models, it mans that results of a tree change a lot across different data partitions. Small changes to the training data can cause large changes, so results has high variance. In the example below, two completely different trees have nearly the same precision.

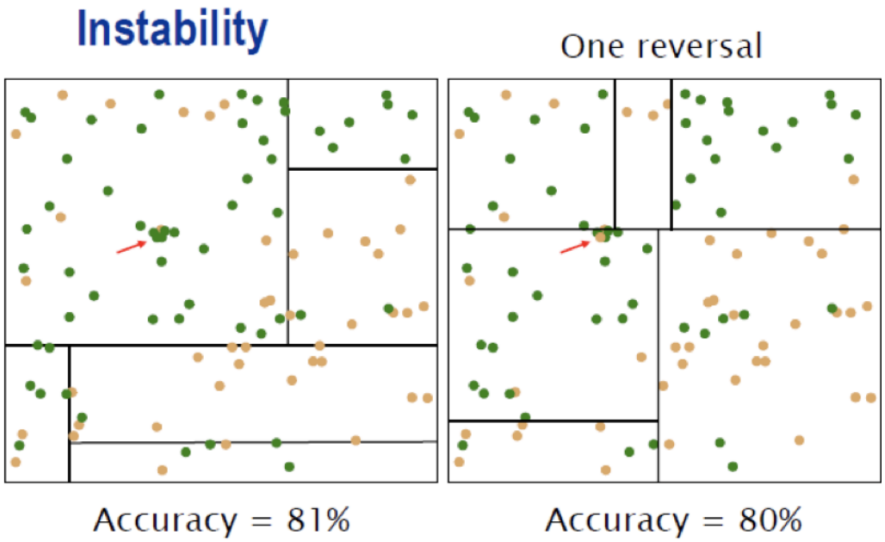


Figure 13: Example of decision tree's instability.

This problem has been resolved by incorporating in into a larger process. New methods were born that exploit the instability of decision tree to create more powerful models.

2.2.2 Ensemble methods

As said before, the decision tree has a severe problem of instability, because of that we will see some models that have generally better performances. The first approach we see is **Ensemble** that consist in combining data. It is a very simple method in which several trees are fitted and then combined together to obtain the classification. The decision boundary is then chosen on the **majority vote**, that is, an observation is classified in a particular target class if the majority of the individual classifiers have placed that observation in that particular class.

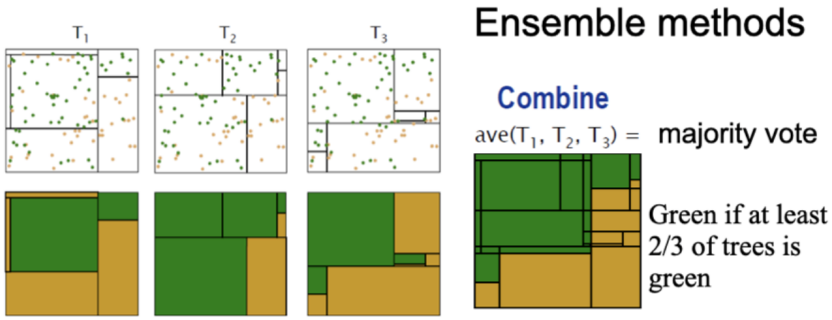


Figure 14: Ensemble methods decision boundary.

In the example T_1, T_2, T_3 could be 3 different datasets or 3 different classification algorithms of the same dataset. This class of models can be divided in some different type of algorithms:

- **Combined models**
- **Bagging**
- **Random forests**

- **Boosted trees**

This models are generally very good estimators, but the main disadvantage is that the simple structure of the model is lost. This can limit your ability to interpret the model and the relationship between the attributes, as the main focus of this models is to generate more accurate estimates.

Combined models: it consists of calculating estimate means on the prediction made by some different predictive models (fitting the many models to the same training data set). The point is to use the same dataset, but different classifiers which are then combined.

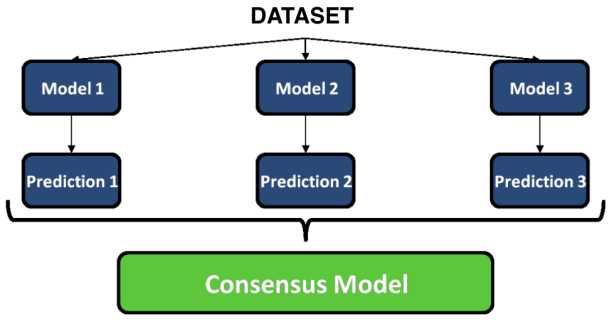


Figure 15: Representation of a combined model.

A final model is obtained which is the **consensus model**. This model works because, assuming that we have many independent classifiers that have a their own error rate, it is improbable that the combined error rate is higher than the single ones. It has been shown that the ensemble classifier works better than the single classifiers if and only if the base classifier has an error rate below 50%.

A successful strategy in combined methods is **stacking**. Stacking train a learning algorithm to combine the predictions of several other learning algorithms (instead of a simple majority vote). First, all algorithms are trained using the available data, then a *meta-classifier* algorithm is trained to make the final prediction using all the other algorithms' predictions as input attributes. In practice, the logistic regression model is commonly used as a Meta-Classifier. The explanatory attributes of this meta-classifier are the predicted probabilities derived from the classifiers and not the data used initially. Stacking typically produces better performance than any single model. It has been successfully used in both supervised learning and unsupervised learning.

Bagging: it combines predictions by doing the mean (majority vote). That predictions comes from the same model trained on different datasets, obtained by bootstrap sampling. The same classification method is used several times on different datasets. This model is commonly used with trees and it is called **bagging trees**. We create bootstrap samples from the starting dataset and we train a tree (the most depth possible) on each sample and at the end they are combined using the major vote or the posterior mean of the individual trees. It is one of the first approach created for reducing decision tree's instability.

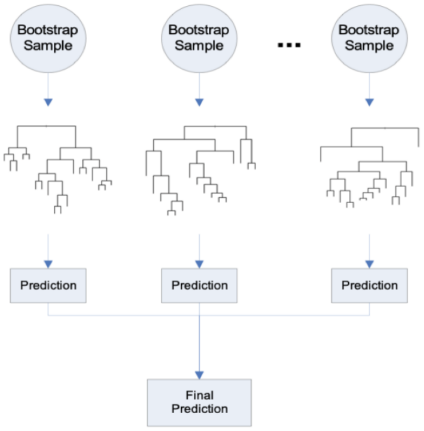


Figure 16: Bagging trees structure.

This tool can be also used to input missing data. If the target is continuous the algorithm can be done, we will have an expected target which is the average of the expected values of the target taken from the bootstrap samples. If the classifier produces probabilities, we can simply average the probabilities for each class and then predict the class with the highest average probability. Since bootstrapping involves randomly selecting subsets of observations for create a training dataset, the left ones could used as test set. On average, each bagged tree uses about 2/3 of the observations, so we have 1/3 of the observations for the test set. For this model we have hundreds of trees and it is no longer clear which variables are the most important. Thus, bagging trees improves prediction accuracy at the expense of interpretability. The big problem with bagging is that if we have a very strong explanatory attribute and other with less explanatory power, all the tree will use that attribute for the first split. This leads to having all the trees with similar aspects, thus all the posterior probability (previsions) will be highly correlated. Averaging many highly correlated quantities does not lead to a reduction of the variance. So the instability of the tree could still be present.

Random forest: it is very similar to bagging, the difference is that within each bootstrap sample the random forest *artificially forces the use of some explanatory attributes in certain trees and not in others*. For every tree, it randomly selects a sample of k of input attributes from the p starting ones, i.e., an attribute is not necessarily present in all trees, but it split only if it is been sampled in each bootstrap sample. As bagging the trees has to be the more depth as possible. The k dimension of the subset is the tuning parameter (can be decided), but the default value is the square root of the number of explanatory variables $\sqrt{p}$. Obviously if $k = p$ then the random forest is equal to a bagging.

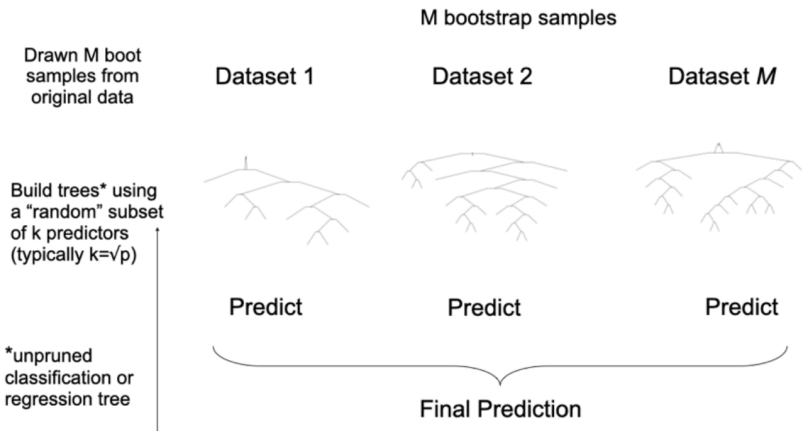


Figure 17: Bagging trees structure.

In case of a continuous output attribute, the random forest will return the expected target value for each observation through the mean of the classification of trees. On the other hand, if the output attribute is in class, the classification rule is the major vote. The random forest model can be use also to assess the attribute's importance. The random forest has the same advantages of the bagging, so it reduce the variance, it is robust to noise, and it can give the attributes' importance. The random forest can be used also in unsupervised ways. It is basically an evolution of the bagging.

Boosted trees: like bagging the boosted trees models work on different dataset, the difference is that in this case the goal is *reducing the error rate* of the base classifier (while the bagging's goal was reducing the variance). Boosted is a classification model that fit the model several times. It learn a set of basic classifiers, where each classifier pays more attention to misclassified observation in the next iteration (assigns more weights to misclassified observations). We will discuss to methods. They are two very performant methods that work one on different data partitions, and one with different targets. AdaBoost reduces misclassification because it works on data from different training sets in each iteration. Each iteration is more likely to see misclassified data, then it combines classifications from various iterations. Gradient Boosting decreases misclassification because in each iteration it works on residuals (it works on same training set), then it combines classifications from various iterations.

- **AdaBoost**: it is a real algorithm (the tuning parameter is the number of iterations). It pay attention to classifiers' predictions, but unlike begging, it assigns a coefficient of importance to each classifier/iteration. It is based on the weighted major vote.

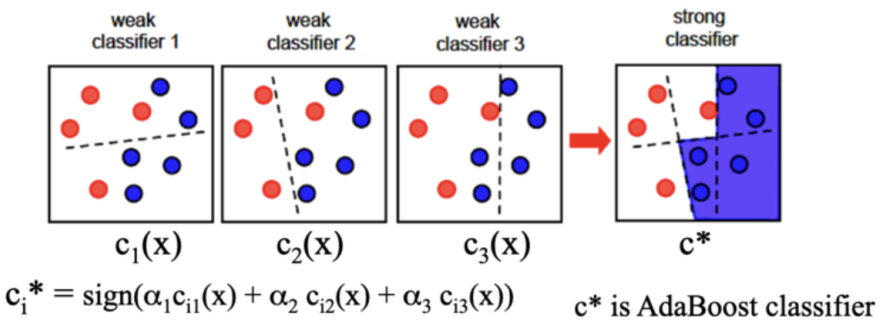


Figure 18: Example of an AdaBoost classifier.

As said before the real difference is how the datasets are built at each iteration. The data are not bootstrap samples of the initial dataset, but they are samples that minimize the error rate iteration by iteration. A classifier as more important as more it is performing. It means to calculate the error rate. The importance α_m of each classifier is:

$$\alpha_m = \frac{1}{2}\ln(\frac{1-\epsilon_m}{\epsilon_m})$$

$$\text{where: } \epsilon_m = \text{error rate of classifier } m = \frac{\sum_i w_i^m misclass_i^m}{n}$$

$$\text{and } w_i^m = \text{weight of observation } i \text{ for the classifier } m$$

As it is understandable from the formula the importance of each classifier increase if the error rate decrease and vice versa.

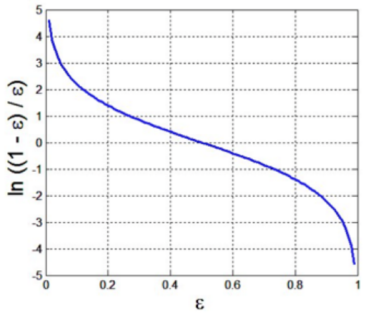


Figure 19: AdaBoost classifier importance trend.

So a classifier is important if it does few missclassifications. Now let's understand how the M datasets are generated, we can imagine that they depends on the importance of the classifier because we have weights to give to each observation. In every iteration m we generate a decision tree on the dataset and compute predictions on this tree. Before selecting a new sample $m+1$ we give a weight to the observation that was in the iteration m basing on the right or wrong classification. The weight is the following, where z is a constant:

$$w_i^{m+1} = \frac{w_i^m}{z} e^{(\pm\alpha_m)}$$

$$+a_m \text{ if } i \text{ wasn't misclassified}$$

$$-a_m \text{ if } i \text{ was misclassified}$$

This rule allows to select more easily in the iteration $m+1$ the observation that were misclassified in the iteration ,. AdaBoost iteratively selects the missclassified observation, so it minimizes the error rate. The AdaBoost increase the classification performance as it focus the decision boundary on the cases that were previously misclassified in the various iterations of the algorithm. It is also a robust classifier as it train on different sample of data, thus it minimize the variability. Its main problem is that it cannot be used with continuous target attributes,

- **Gradient Boosting**: unlike AdaBoost, Gradient Boosting can be used to both continuous and qualitative target attributes. It is an algorithm with

high performances and it will be often the best model. It use a many basic models in a row (algorithm):

$$f_{GBM}(x) = \sum_{m=1}^{m} amf_m(x)$$

The tuning parameter is the number of iterations M, which are basically the M trees that we will fit on the dataset. f_m is the classification (if continuous target) of the m-th iteration, which is the last prediction at the end of all iterations. So, fGB is a weighted sum. Now the point is to understand how α weights are obtained. To have a higher performance, in the second iteration, the algorithm study the unexplained residual as a target attribute and jointly also obtains alpha (importance of the tree). It combine the predictions of the trees in the two iteration. In the first iteration the Gradient Boosting model work on the training set, it generate the splits and it select the most important attributes. Doing so it generates the first prediction $f_1(x)$, using the first decision tree. Only in the first iteration the Gradient Boosting's prediction is the same to the tree's one, it search the best tree and set $\alpha_1 = 1$. On the second iteration the model search a new tree, which differently form the first one it change the explanatory attribute, which it become the residual of the first tree and we compute α_2 obtained trough the minimisation of:

$$min_\gamma = \sum_{i=1}^{n} (r_{i1} - Tree(\gamma)_i)^2$$

Where r_1 is the residual quantity not explained by the first tree, $f_2(x)$ is the prediction at the second iteration and α_2 is the quality of the second tree (to be estimated). So the decision boundary at iteration two is:

$$f_{GB2}(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) = \sum_{m=1,2} \alpha_m f_m(x)$$

Gradient Boosting at the second iteration has a mean between the predictions of the first two trees weighted for the the accuracy, as the decision boundary $f_{GB2}(x)$. For construction every iteration improve the decision boundary of the interaction before. This procedure is the same for m interactions. It is important to pay attention at the loss function. Given a loss function $J(w)$, the **gradient of the lost function** is defined as:

$$\text{Gradient of } J(w) = \frac{\partial J(W)}{\partial w}$$

It is the vector of partial derivatives on w. In the iteration m, we suppose that we have estimated w_m. So the gradient rule tells us to update the parameters in iteration $m + 1$, minimizing the gradient. So we try to make the gradient null, i.e., making the residual null in the next iteration, i.e., modeling the iteration m residuals as a new explanatory attributes in the iteration $m + 1$. To minimize the gradient it is necessary to reduce the residual to zero because the x are fixed. If the residual is positive, the parameters should increase but the gradient is negative. The obtained rule say that we need to **update the parameters in the opposite it direction**

of the gradient. So minimizing the residuals at each iteration is the same to iteratively minimizing the gradient or **the residual sum of square** at each iteration. If the target attributes is binary we use **log-likelihood** instead of the residual sum of square. In the Gradient Boosting the tuning parameters are:

- *Number of trees*: it is the iteration number.
- *Interaction depth*: it is the number of final nodes in a given tree in iteration m and it is fundamental as tells us how much we can make interact the attributes in search of the optimal tree.
- *Shrinkage*: it is the learning rate λ. Empirically it has been found that the use of reduced learning rates (such as $\lambda \leq 0.1$) produces big improvements the model generalization compared to increasing the gradient without shrinking ($\lambda = 1$). However, it has the problem of increasing computation time, because a lower learning rate requires more iterations.
- *Variable importance*: it is $\sqrt{[mean(\text{importance in every fitted tree})]^2}$, (most important variable was scaled to 100). It is preferable not to give too much weight the importance of variables in Gradient Boosting.

The advantages of the Gradient Boosting is that it has high precision and stability, because the calculation of the weighted average of the values predicted by combining model estimates is generally more accurate and more stable than to individual models; and the high possibility of generalization. On the contrary it has also some cons like the fact that the performance and the stability depend on sample size on the number of input attributes in the training dataset. The use of a small training dataset that is not an appropriate representation of the real population will result in inadequate ensemble model estimates.

2.2.3 Binomial Logistic Regression

The **logistic regression** is a method based on the regression. This models are useful to solve binary regression problems at different levels. It is usable also on continuous targets and with some niceties even on the nominal ones.

Assume the class attribute Y takes value in $\{0, 1\}$, then the binomial logistic regression classifier computes the *posterior probability* of the class attribute Y given the value of the input or explicative attributes $\underline{X}$. Computed as it follows:

$$P(Y = 0|\underline{X} = \underline{x}) = \frac{1}{1 + exp(\underline{x} \cdot \underline{w})}$$

$$P(Y = 1|\underline{X} = \underline{x}) = \frac{exp(\underline{w} \cdot \underline{x})}{1 + exp(\underline{w} \cdot \underline{x})}$$

where $\underline{w}$ is called *parameters vector.*

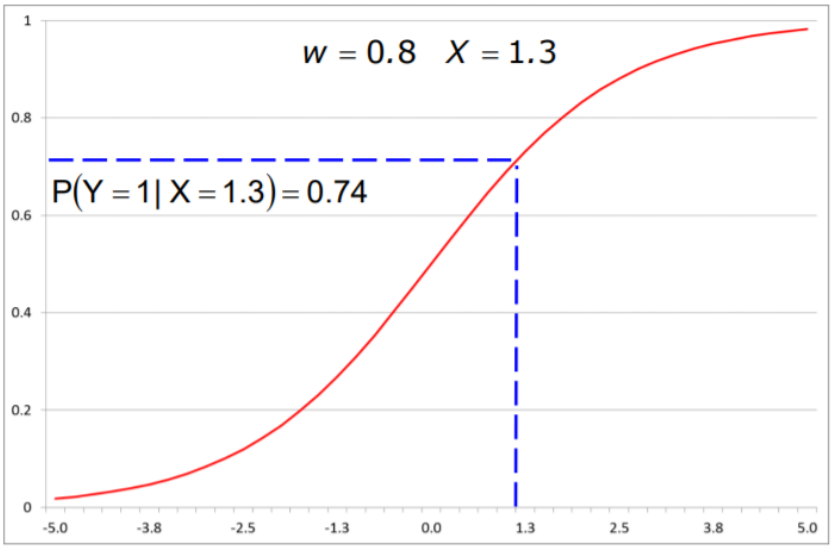


Figure 20: Example of logistic regression

In this way we can compute the probability of being in a particular class.

2.2.4 Support Vector Machines

The **Support Vector Machines** is a method of classification with separation. The goal is to separate or "learn" the target class.

Given a bidimensional space in which there are m records where two continuous attributes are measured: : $D = \{(\underline{x}_1, y_1), ..., (\underline{x}_m, y_m)\}$ where $\underline{x}_i \in R^2$ e $y_i \in \{-1, +1\}$

The idea is to drwo a line (if we have only two classes) for which we can define the belonging to a class or another.

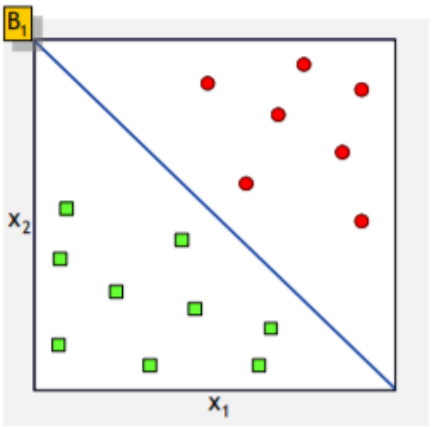


Figure 21: Example of a possible separation line

The line is described by the following equation:

$$\underline{w} \cdot \underline{x} + b = w_1 x_1 + w_2 x_2 + b = 0$$

It is fully described by the following vector $\underline{w} = [w_1, w_2]$ or b. ($\underline{w}$ rotate the line, b translate the line). If the line exist it means that the set of observation exist a *linear decision boundary.*

Problem: there can be more than one straight lines (even infinite), that can be used to carry out the decision boundary. It's not enough to find a straight line that works, but we wont the best straight line. A sort of gray area is created in which I can't tell exactly which of the two classes to categorize.

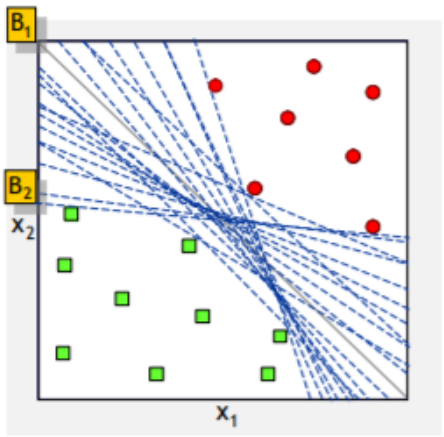


Figure 22: Different lines as solution of the same problem.

We have to select the line that **maximizes the margin**, e.i., the **optimal linear decision boundary**.

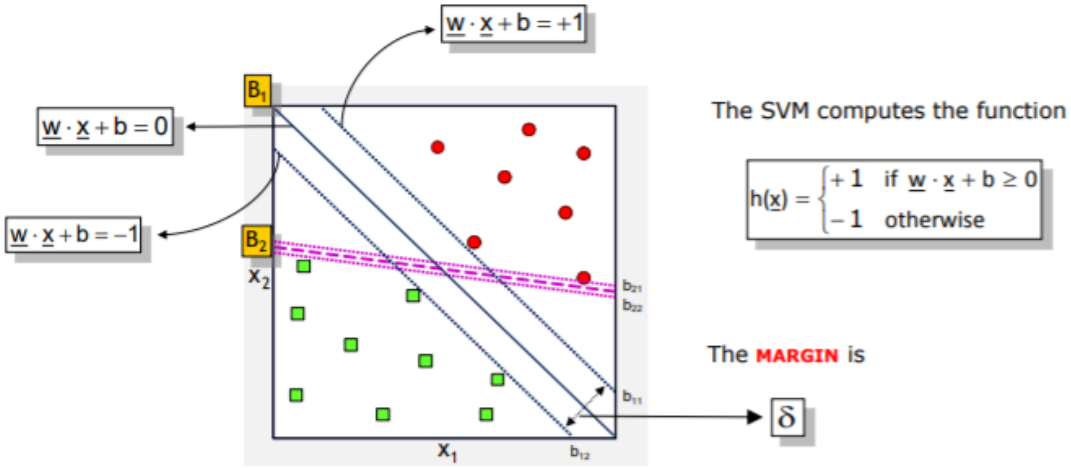


Figure 23: Boundary

In the example the B_1 line is preferable on the B_2 line, because it has a bigger margin δ.

Naturally, for n attributes we need to find the optimal *hyperplane* to divide a given set of observations. Mathematically speaking we try to maximize the margin $\delta = 1/|w|^2$, where the straight line it is: $\bar{w} \cdot \bar{x} + b = 0$. The lines on the edges of the margin are fixed on $\bar{w} \cdot \bar{x} + b = 1$ and $\bar{w} \cdot \bar{x} + b = -1$.

The argument of the line is in two dimension, but after using the $h(\underline{x})$ function that line become a plane that goes to -1 on one side and to $+1$ on the other side.

Training a **Linear Hard-margin SVM** consist of formulating and solving the following mathematical programming problem:

$$\min_{\underline{w},b} \frac{1}{2} \underline{w} \cdot \underline{w}^T$$

s.t.

$$y_i(\underline{w} \cdot \underline{x}_i + b) \geq 1 \quad \forall i = 1, ..., m$$

It is a quadratic programming problem with liner constraints which must be solved with special numerical techniques.

To find the line we have to minimize the inverse of the margin δ; to do that, we need to set constraints on each attribute. In particular, if $\underline{w}$ and $\underline{x}$ *agree in sign* (positive or negative) then it classify perfectly because the result is $>= 1$. These constraints ensure that all cases in the dataset are classified correctly. Between all cases that classify correctly, we choose the one that **maximizes the margin**.

This setting work well is when data are linearly separable.

In the cases where the decision boundary is not liner, there is not a straight line capable of correctly splitting the classes. In these cases the this formulation doesn't admit solutions, because for some data the constraints admit no solution.

The Linear Hard-margin SVM formulation is infeasible and thus we need to introduce the **Linear Soft-margin SVM**:

$$\min_{\underline{w},b,\underline{\epsilon}} \frac{1}{2}\underline{w} \cdot \underline{w}^T + \Delta \sum_{i=1}^{m} \epsilon_i$$

s.t.

$$\forall_{i=1}^{m} : y_i(\underline{w} \cdot \underline{x}_i + b) \geq 1 - \epsilon$$

$$\forall_{i=1}^{m} : \epsilon \geq 0$$

The ϵ must be non-negative (slack variables), if we use this parameter to admit an error then there is at least one line that solves the optimization problem. We basically *relaxed* the optimization problem, especially the constraints. Graphically speaking, this translates into the translation of observations of a different class from the region to which they belong towards the region of the class of that observation.

It is very important to notice that the support vectors are those observations on margin's edges.

Problem: The separation between class is such that it can be done with a **non-linear** function.

Solution: We search a transformation that generate a *feature space* where the classification problem becomes easier and a linear separation is applicable.

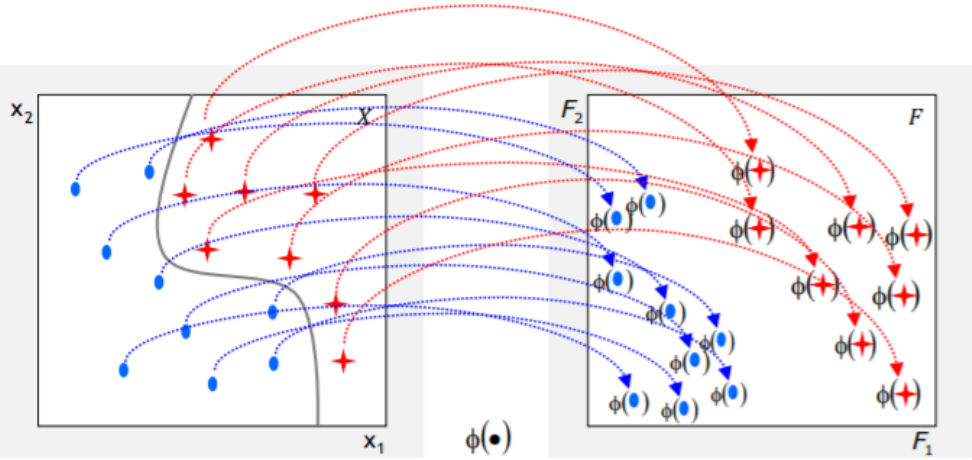


Figure 24: Application of a non-linear separation

In this way we can use the Linear Soft-margin SVM in a controlled space. But to do that we have to find a transformation $\phi(x)$ which maps X to F. The linear decision boundary in the feature space F has the following equation:

$$\underline{w} \cdot \phi(\underline{x}) + b = 0$$

Training a **Non-linear SVM** consist of formulating and solving the following mathematical programming problem:

$$\min_{\underline{w},b} \frac{1}{2}\underline{w} \cdot \underline{w}^T$$

s.t.

$$y_i(\underline{w} \cdot \phi(\underline{x}_i) + b) \geq 1 \quad \forall i = 1, ..., m$$

The main difference with Linear Soft-margin SVM learning is that instead of using the attributes X the transformed attributes $\phi(x)$ are used.

The learning algorithm uses the following function:

$$K(\underline{u}, \underline{v}) = \phi(\underline{u}) \cdot \phi(\underline{v})$$

K is a similarity function computed in the original attribute space X and it is referred to as the *kernel function*.

2.2.5 Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) is a classification technique based on separation between spaces of attributes. This models are very common nowadays for deep learning.

Definition 2.10 (Multi-Layer Perceptron)**.** A **Multi-Layer Perceptron** consists of artificial neurons which communicate unidirectionally, from the input attributes X to the class attribute Y.

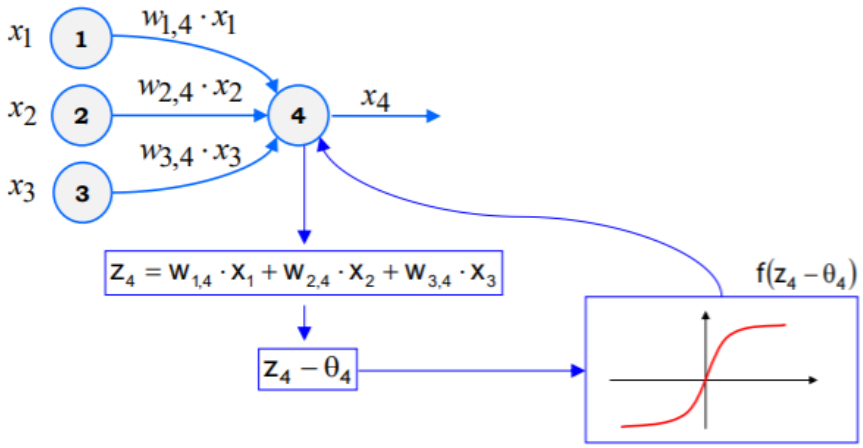


Figure 25: Example of the a MLP's structure.

In figure there are tree neuron of continuous parameters, to which is added a fourth neuron to compute a function. A weight w is associated with every neuron. A neuron compute a **linear combination** between the input attributes and its weight, the j-th neuron compute:

$$y_j = f(\sum_{i=1}^{n} w_{i,j} \cdot x_i - \theta_j)$$

For example, the fourth neuron in figure is:

$$z_4 = w_{1,4} \cdot x_1 + w_{2,4} \cdot x_2 + w_{3,4} \cdot x_3$$

To this computation is given a threshold (or bias) value θ applied to the linear combination. Subsequently, a transfer function is applied on the neuron minus the threshold, in our case $f(z_4 - \theta_4)$.

Definition 2.11 (Transfer function)**.** The **transfer function** is defined as the function applied on a neuron that gives the value brought to another communicating layer.

As the name suggests, a MLP use multiple perception. A perceptron is the most simple neural network with only one layer. It is composed by the inputs x with the wight w that combine in the output layer. As for the MLP the inputs are combined in a linear combination $H = w_0 + \sum_{j=1}^{p} w_j x_{ij}$ to which is applied a transfer function F. It is usable only if the problem is linear separable, because it produce a linear decision boundary. We introduce the MLP because it is an evolution of the perceptron with more complex decision boundaries.

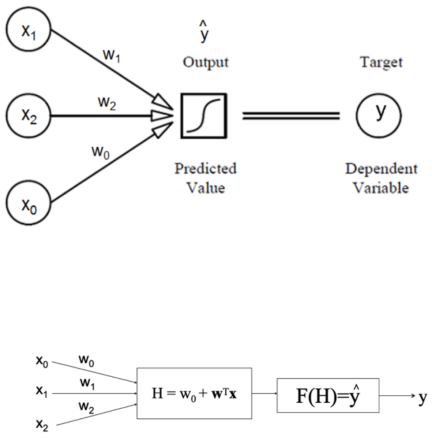


Figure 26: Structure of a perceptron.

The most used transfer function are the following:

- *Identity/linear*: multiple linear regression is a perceptron that essentially assumes that a linear activation function F is applied to the linear combination. The difference is that the network estimates the w weights with the descendent gradient.
- *Sigmoid/logistic*: the logistic regression is a simple perceptron with sigmoid F activation function. The difference is that the network estimates the weights w differently from maximum likelihood. It is useful if we want expected probability as output.
- *Sign*: the binary target is coded with $(+1, -1)$ if the dependent variable is binomial. F processes the combination and based on the sign obtains the output (the expected value). Consequently the sign will represent the decision boundary.

- *Kernel*: we may be transfer functions that activate the functions according to particular distances based on the Kernel. Which can be a Gaussian.

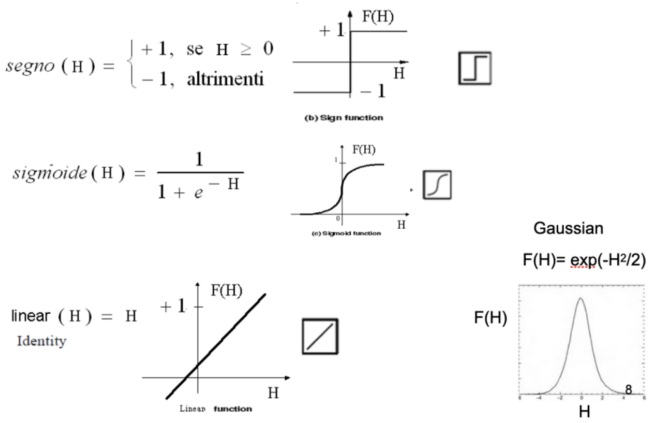


Figure 27: Perceptron's transfer functions.

Thinking of a MLP with 2 layers, the input attributes arrive with their weight in the first layer (hidden layer), they are treated as perceptrons so a transfer function F is applied on each one. Then they are used as input for the second layer (the output layer in this case) and linearly combined with new weights as $S = \sum_j w_j F(H_j)$. Then a new transfer function G is used on the linear combination S.

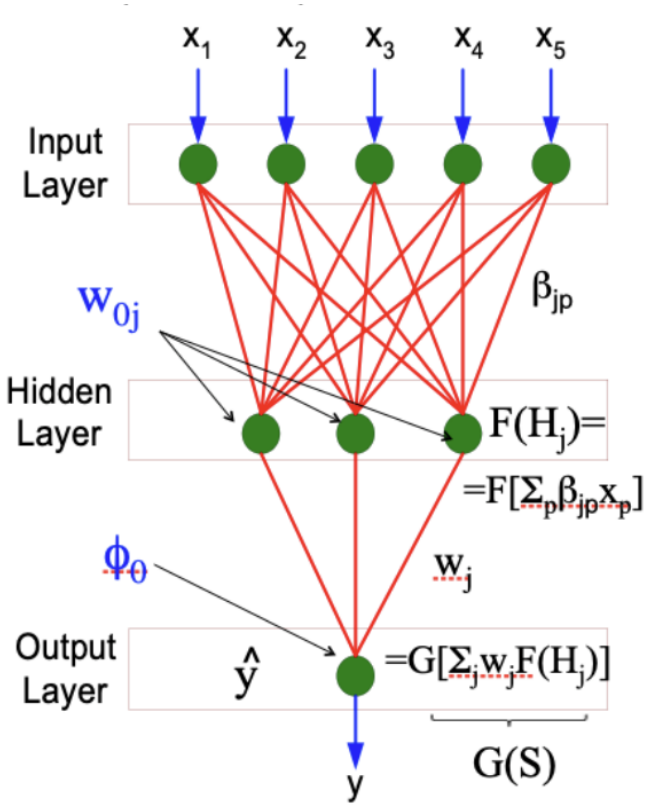


Figure 28: Perceptron's transfer functions.

In the traditional network there are typically two or tree hidden layers, but in the *deep learning* we can have a huge number of them. The transfer function typically used in the hidden layers are:

- *Hyperfolic tangent*: $F(H) = \frac{e^H - e^{-H}}{e^H + e^{-H}}$
- *Logistic*: $F(H) = \frac{1}{1+e^(-H)}$

Differently, it the used output layer the transfer functions G are:

- *Identity/linear*: $G(S) = S$. Used if the problem is a standard regression.
- *Logistic/sigmoid*: $G(S) = \frac{1}{1+e^(-S)}$. Used if the problem is a binary classification.
- *Softmax*: $G(S) = \frac{e^{S_i}}{\sum_j e^{S_j}}$. Used for multiclass problems.

Exploding gradient or vanishing gradient could cause errors at edges of the transfer function, so it is preferable to use more simple functions.

As anticipated there are tree type of neurons:

- **Input**: they are connected with the explanatory attributes and with *each* hidden node.
- **Hidden**: they receive as input the linear combination of the input neurons and the signal is propagated to the output ones.
- **Output**: they are associated with the class variable and receive the signals to be displayed.

Each input neuron is connected through directional links with all hidden neurons. The signal propagates form neurons of the input layer to neurons of the hidden layer, then the output neuron labels the records. This type of network is called **fully-connected**. Every connection has its weight and every node has its threshold θ_j and its transfer function.

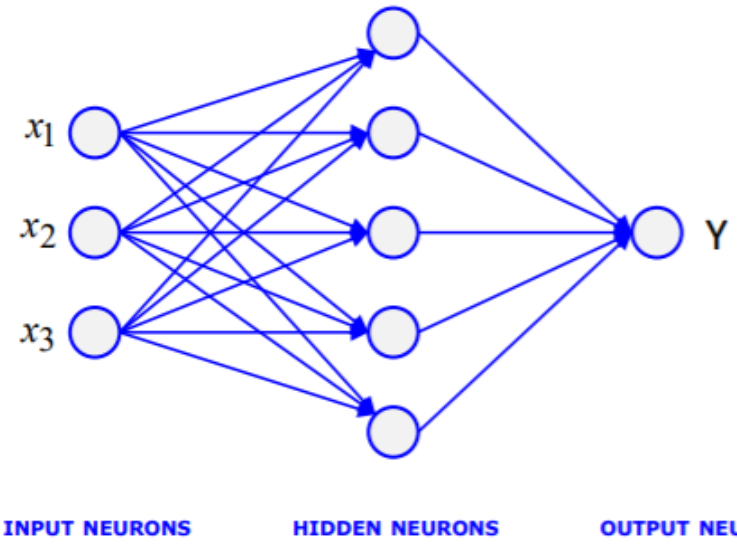


Figure 29: Simple MLP model.

MLPs can have different architectures, i.e., different number of hidden layers and different number of hidden neurons for each hidden layer. There are no constraints on communicating by skipping levels. However, it is not possible to communicate backwards, in fact these networks are called **feed-forward neural networks** (they can have up to hundreds of hidden layers.

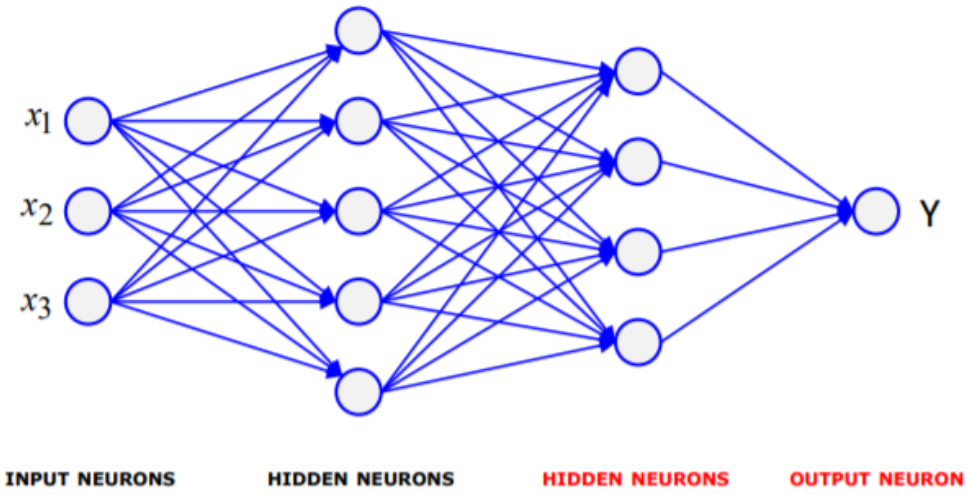


Figure 30: Example of a MLP with two hidden layer.

MLP learning is a difficult task, different optimization measures have been proposed in the specialized literature, many different learning algorithms have been designed and described, but **multiple local minima usually exist** and thus it is always extremely difficult to learn an MLP. The number of nodes, arc and hidden levels it is usually selected by trials and experience.

Now lets rapidly see how network parameters are estimated. They are not estimated with the gradient descendant but rather with **back propagation** algorithm. The mechanism of updating the estimates becomes complex because the output of the network depends on many factors that must be taken into account. This is the loss function:

$$J(w) = \frac{1}{n}\sum_i (y_i - \hat{y}_i)^2$$

Every iteration has two phases:

1. *Forward phase*: where, with random initial weights, the network is propagated from the input layers to the output ones, giving final outputs $\hat{y}_i$.

2. *Backward phase*: where the error between the observed y_i and predicted $\hat{y}_i$ is propagated/separated back to change the initial layers' weights in an adaptive way.

The parameters are updated in the gradient opposite direction, splitting the latter into the individual components that generated the outputs in the forward phase. The problem is to derive the gradient of the loss function as $J(w)$ is generated by a lot of steps in the hidden layers. We will split the gradient for each contribution to get the target.

A MLP tuning parameters are:

- *Early stop*: it can be used to prevent the MLP from overfitting. The last iteration can be selected using the residual sum of square on the training set.

- Decay: it adjusts the update rate of parameters (weights) from in sequential iterations.

- *The penalty parameter* λ: it tries to "reduce" the parameter estimates towards zero: large parameter estimates are penalized.

By using the back propagation algorithm to a perceptron, it can be viewed as a logistic regression that estimates parameters with a descendent gradient. It is important to observe if the algorithm converges (how many epochs are necessary).

MLPs are universal approximators, it means that with at least two layers and a sufficient number of neurons in the Hidden layer (at least three): can approximate the shape of the relationship between x and y with few information. The MLPs are considered *shallow networks* because they have few layers and many neurons, on the contrary of the deep networks.

2.2.6 Bayes Classifier

A **probabilistic classifier** solves the classification problem using the **Bayes theorem**.

Definition 2.12 (Bayes theorem)**.** The **Bayes theorem** describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

$$P(Y|\underline{X}) = \frac{P(\underline{X}|Y) \cdot P(Y)}{P(\underline{X})}$$

Where:

- $P(Y)$ is the *prior probability* of the class attribute.
- $P(\underline{X}|Y)$ is the *likelihood* of the explanatory attribute vector given the class attribute.
- $P(\underline{X})$ is the *probability of evidence.*
- $P(Y|\underline{X})$ is the *posterior probability* of the class attribute given the explanatory attribute vector.

Lets see an example, we can assume to have:

- Y binary class attribute $\{-1, +1\}$
- $\underline{X}$ binary explanatory attribute $\{Male, Female\}$

Lets assume to have this probabilities:

$$P(Y) = (0.3, 0.7) \quad P(\underline{X}|Y = -1) = (0.2, 0.8) \quad P(\underline{X}|Y = +1) = (0.9, 0.1)$$

We wont classify $\underline{X} = Male$ using the Bayes theorem:

$$P(Y = -1|\underline{X} = M) = \frac{P(\underline{X} = M|Y = -1) \cdot P(Y = -1)}{P(\underline{X} = M)} = \frac{0.06}{P(\underline{X} = M)}$$

$$P(Y = +1|\underline{X} = M) = \frac{P(\underline{X} = M|Y = +1) \cdot P(Y = +1)}{P(\underline{X} = M)} = \frac{0.63}{P(\underline{X} = M)}$$

In this situation Y is more likely to have $+1$ value if $\underline{X} = Male$.

Now lets assume that the number of explanatory attributes rise as **n**, and they are binary as the class attribute. So we need to know this parameters:

$$\theta_{ki} = P(\underline{X} = x_k | Y = y_i) \qquad k \in \{1, ..., 2^n\}, y_i \in \{-1, +1\}$$

When the number of explanatory attribute rise we need to binarize them, so we will 2^n parameters:

n	1	2	3	10	20	30
# parameters	2	4	8	1,024	1,048,576	1,073,741,824

The **Bayes classifier is infeasible** since $P(X|Y)$ or is unknown and usually hard to estimate, because of the high dimensionality of X (strongest motivation) and because is ignored what are useful/significant input X. In other words, the Bayes classification error is unknown. *The Bayes classifier is the best possible classifier but can be used only when the class-conditional densities and priors are known.* Because of that we use model with simplifying hypothesis like LDA or Naïve Bayes.

2.2.7 Naïve Bayes

Naïve Bayes classifier uses a simplified feasible version of Bayes classifier.

Definition 2.13 (Conditional independence)**.** Given X,Y and Z, attributes we say that X is **conditionally independent** from Y given Z, if and only if the probability of X is independent from the value of the attribute Y once the value of the attribute Z is known, formally:

$$\forall i, j, k P(X = x_i | Y = y_j, Z = z_k) = P(X = x_i | Z = z_k)$$

Conditional independence of explanatory attributes given the class attribute allows to drastically reduce the number of parameters to be computed; the **Naïve Bayes** assumes conditional independence and thus allows to compute the posterior probability of the class attribute given the explanatory attributes as follows:

$$P(X_1, ..., X_n | Y) = \prod_{i=1}^{n} P(X_i | Y)$$

$$2 \cdot (2^n - 1) \rightarrow 2 \cdot n$$

The **Naïve Bayes classifier** computes the posterior probability of the class attribute as follows:

$$P(Y = y_k | X_1, ..., X_n) = \frac{P(Y = y_k) \cdot \prod_{i=1}^{n} P(X_i | Y = y_k)}{\sum_j P(Y = y_j) \cdot \prod_{i=1}^{n} P(X_i | Y = y_j)}$$

Then the record is labeled with the class value which maximizes the posterior probability:

$$arg \max_{y_k} P(Y = y_k | X_1, ..., X_n)$$

The Naïve Bayes model belong to the family of **probabilistic graphical models**. There are many models that also belong to this family, like dynamic Bayes network, but we are not going to detail this argument.

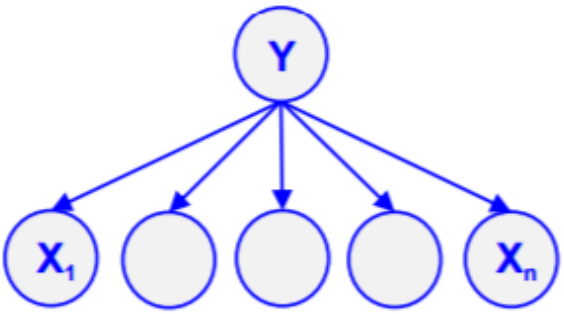


Figure 31: Naïve Bayes net

It is advisable to give a probabilistic measure with the output, that could suggest an *reliability measure*.

The Naïve Bayes classifier just presented can be used with categorical attributes (nominal and ordinal) and with numeric attributes (interval and ratio). Each numeric attribute is associate with a **class conditional probability density**:

$$P(X_i = x_i|Y = y_k) = \frac{1}{\sigma_{ik}\sqrt{2 \cdot \pi}} exp(-\frac{(x_i - \mu_{ik})^2}{2\sigma_{ik}^2})$$

whose parameters are:

$$\mu_{ik} = E[X_i|Y = y_k] \qquad \sigma_{ik}^2 = E[(X_i - \mu_{ik})^2|Y = y_k]$$

Note that the Naïve Bayes naturally combines the use of categorical attributes with numeric attributes.

The Naïve Bayes is generally a good model, but it requires the huge assumption of the conditional Independence in order to be correctly applied. To overcome the problem, a new and more flexible version has been developed.

2.2.8 Bayesian Networks

The Naïve classifier is generalized by **Bayesian networks** which make less drastic conditional independence assumption but exploit the concept of **sparsity**.

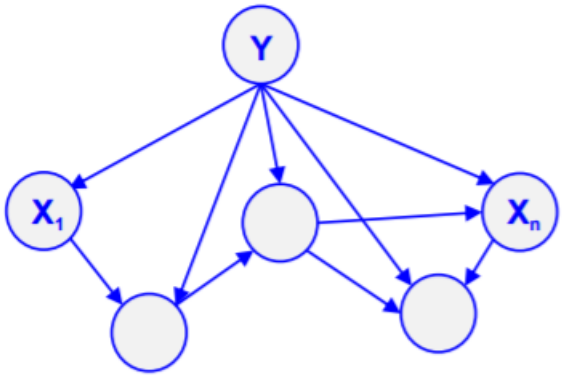


Figure 32: Example of a Bayesian network classifier.

The class attribute is still associated with the root node Y, while the explanatory attributes are associeated with nodes having not only the root node Y as their incoming arches, explanatory attributes can point to other explanatory

attributes. A constraint of Bayesian networks is that **no cycles are allowed**. It has to be done a conditional probability table for every node with respect of its parent nodes. Basically, we consider every configuration between the parent nodes and for every one we compute a different output.

This generalization keeps into proper account dependencies between the explanatory attributes that are no longer assumed to be independent given the class attribute.

This model is a particular Byes net used for classification. This model can be also use with missing values.

2.2.9 Tree-augmented Naive Bayes

Many modifications of Bayes network classifiers have been proposed in the specialized literature. Removing the dashed arc from the Bayesian Net in figure brings to Tree-Augmented Naïve Bayes.

Definition 2.14 (Tree-Augmented Naïve Bayes)**. Tree-Augmented Naïve Bayes** is a semi-Naïve Bayesian Learning method. It relaxes the Naïve Bayes attribute independence assumption by employing a tree structure, in which each attribute only depends on the class and one other attribute. A maximum weighted spanning tree that maximizes the likelihood of the training data is used to perform classification.

Each node has at most one explanatory attribute as its parent (incoming arc from), thus if we remove the root node Y a tree is obtained which gives the name to this particular type of Bayesian net classifier.

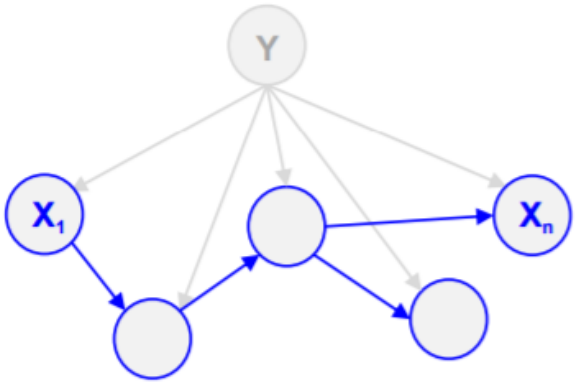


Figure 33: Tree-Augmented Naïve Bayes

It is very powerful for feature selection, basically the selection of the most important attributes. Many other modifications include AODE, HNB and many more. All oriented toward the implementation of a trade-off between expressiveness and complexity.

2.2.10 Summary

In order to summary and understand we propose the following table, which it shows the type of attributes usable for the proposed models:

	Explanatory Attribute				Class Attribute	
	Categorical			Numeric	Categorical	
	Nominal	Ordinal	Binary		Nominal	Binary
Decision Tree	X	X	X	X		X
Logistic Regression				X		X
SVM		X	X	X		X
MLP		X	X	X		X
NB	X	X	X	X	X	X

Figure 34: Comparison between models.

However, is it possible to transform nominal Explanatory Attributes and nominal Class Attributes to apply the classification techniques that can not allocate nominal Attributes. Note that classification techniques are not the best option when an ordinal is to be predicted.

2.3 Performance Evaluation (*)

A point estimate of classification model's accuracy is not enough to be confident that the developed classification model will provide a reliable prediction when queried on unseen records. Furthermore, we have to be aware of the risk of **overfitting/underfitting**.

Errors committed by a classification model are divided into two types:

- **Training error**: the number of records of the training set which are misclassified.
- **Generalization error**: expected error on previously unseen records (test set).

A good Classification Model must not only fit the Training Set well, it must also accurately classify records it has never seen before (Test Set).

Definition 2.15 (Overfitting)**. Overfitting** is the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit to additional data or predict future observations reliably. An overfitted model is a mathematical model that contains more parameters than can be justified by the data.

In practice: A good classification model must have a compromise between a low training error and a low generalization error.

A classification model fitting the training set to well can have a poorer generalization error (estimated through the test set) than a model with higher training error.

Lets take a look to the following example:

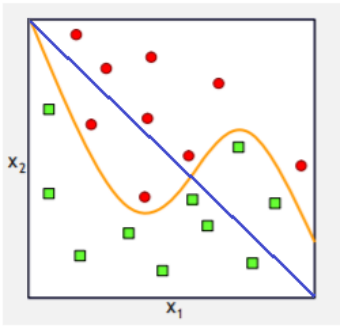


Figure 35: Example of overfitting.

- Blue line: model with some training error (no overfitting)
- Orange line: model with zero training error (overfitting)

As we can see the orange model archives a better training error then the blue one. The point is that the orange will probably fail in the classification of new data because it is too specialized on the training ones.

Lets now see how the models react to new records previously unseen data (test set):

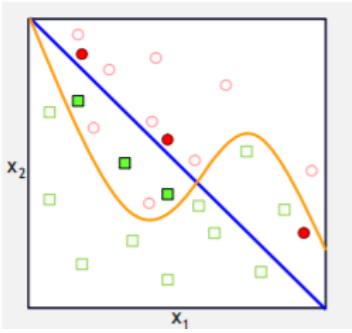


Figure 36: Example of overfitting in test set.

The blue model has a higher training error but a lower generalization error, compered to the orange model that has a lower training error but an higher generalization error. Overall the blue model is the better one because it usable for classification purposes on unseen records.

Definition 2.16 (Underfitting)**. Underfitting**, happens when a model is not complex enough to accurately capture relationships between a dataset's features and a target variable. An underfitted model results in problematic or erroneous outcomes on new data, or data that it wasn't trained on, and often performs poorly even on training data.

The model complexity is not enough for the data that we are studying. We didn't used all the possible flexibility in the model definition.

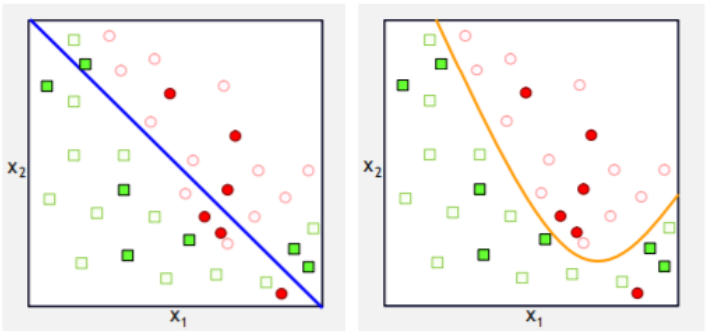


Figure 37: Example of underfitting.

Looking at the examples, the blue model wrongly classify six record, so it is a bad model. On the other hand, the orange fit well the training and also it response better to the test set (one error). It is important to not overfit while trying not to underfit.

The optimal solution is to check what happen as well in the training set as in the test set. A real mensuration of the model's error is impossible, we can only get a estimate.

2.3.1 Measures of Performance Evaluation

A classification analysis consists of developing different classification models in terms of both the type of classification model and the associated parameters value. Such a procedure aims to *select* a specific instance of a classification model, algorithm and parameters value, capable to *ensure* the *maximum possible predictive accuracy*:

Classification models are compared in terms of:

- *Accuracy*
- *Speed*
- *Robustness*
- *Scalability*
- *Interpretability*

2.3.2 Accuracy

Accuracy is an important measure because it:

- measures the capability of the classification model to give reliable predictions on new records (i.e. records not available when the model was developed),
- allows to select the instance of classification model which likely provide the best prediction performance on new records.

We use the following notation:

D_T training set, consists of t records. D_{TS} test set, consists of v records. $D = D_T \cup D_{TS}, D_T \cap D_{TS} = \emptyset, m = t + v$

A good indicator of the accuracy achieved by a classifier is represented by the percentage of test record D_{TS} which are correctly classified.

We let:

- y_i be the class value associated with the instance $\underline{x_i} \in D_{TS}$
- $f(\underline{x_i})$ the class value tor the same instance as predicted by the classification model

then we can use the following loss function:

$$L(y_i, f(\underline{x_i})) = \begin{cases} 0 & \text{if} \quad y_i = f(\underline{x_i}) \\ 1 & \text{if} \quad y_i \neq f(\underline{x_i}) \end{cases}$$

to compute the **accuracy** as follows:

$$acc(D_{TS}) = 1 - \frac{1}{v} \sum_{i=1}^{v} L(y_i, f(\underline{x_i}))$$

In some case we prefer to use the **error**:

$$err(D_{TS}) = 1 - acc(D_{TS}) = \frac{1}{v} \sum_{i=1}^{v} L(y_i, f(\underline{x_i}))$$

2.3.3 Speed

Classification algorithms differ from:

- Learning time
- Space memory

Selecting the type of classification model to learn depends on the above aspects.

A classifier which requires high learning time and/or great memory space to be learnt can be learnt after sampling the original dataset. In such a case we accept not to exploit all the available information spite of learning a type of classification model that we think will ensure good performance measures.

2.3.4 Robustness

Definition 2.17 (Robustness)**.** A classification mode/algorithm can be **robust** or not with respect to:

- *Outliers*: could significantly influence the model.
- *Missing data*: main problem (the base data are dirt).
- *Variation of training and test data*: we started form the assumption that training set and test set have the same distribution, if this assumption is not true the model is no longer robust.

2.3.5 Scalability

Definition 2.18 (Scalability)**.** A classifier is **scalable** if it is capable to learn from huge amount of data. This property is intrinsically connected to the learning speed.

For example Bayesian networks scale in a really bad way.

2.3.6 Interpretability

Definition 2.19 (Interpretability)**.** When machine learning is oriented towards **interpretability** of the problem to be solved and it is not limited to ensure good accuracy values, it is relevant the extracted rules to be simple and clearly understandable by the domain expert for the problem under study.

Interpretability is a huge problem, because humans don't think in quantitative terms, but in qualitative terms. An explanation can be as detailed and precise as possible, but if it doesn't meet the language and context in which the subject we want to reach lives, the goal of having the work interpreted is not achieved. Due to this fact some tools are being developed even for the most uninterpretable models (black box models). An example is the tool *Lime* for neural networks.

2.3.7 Holdout

Definition 2.20 (Holdout)**. Holdout** consists of limiting the data which are used to learn the classifier, i.e. some records are saved (hold back) to estimate the reliability level of the classifier for the task to be solved.

Best practice: $\frac{2}{3}$ for training set and $\frac{1}{3}$ for the test set.
This proportion is not mandatory, especially if we have many data we can increase the training set.

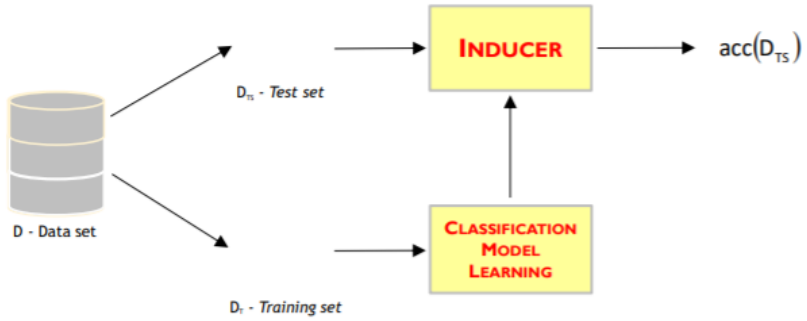


Figure 38: Holdout procedure.

The accuracy estimate depends on the choice of the test set, thus we can over(under)estimate the true value of the classification accuracy. A more robust estimate can be obtained through iterated holdout and cross validation.

Definition 2.21 (Iterated holdout)**. Iterated holdout** consists of iterating r times the holdout method, for which we learn and estimate the accuracy.

For each iteration r we extract random sample D_{T_r} consisting of t records and we obtain:

$$D_{TS_r} = D - D_{T_r}$$

The procedure is repeated r times and the accuracy of the classifier is estimated by the sample mean of the accuracy values $acc(D_{TS_r})$ computed on each test set D_{TS_r}:

$$acc = \frac{1}{R} \sum_{r=1}^{R} acc(D_{TS_r})$$

The number of iterations r can be selected exploiting specific statistical techniques.

Iterated holdout significantly improve on holdout, it estimates performance measures with a bias which is smaller than the bias associated with estimates obtained form the holdout. However, iterated holdout does not allow to control the number of times a given record is contained in the training and in the test set. This could result in a strong bias with specific reference to cases where the data set contains dominant records (i.e. outliers). In such a cases we resort to use more effective estimation schema, i.e. capable to control (reduce) the impact of outlier record(s) on the estimated value for the classifier's performance measures.

Definition 2.22 (Cross validation)**. Cross-validation** is the holdout technique which ensures that each record of the dataset D is included into the training sets the same number of times and exactly one time in the test set.

The dataset D is partitioned into k disjoint subset exhaustive (a partition of D) and with almost a constant number of records:

$$D_1, D_2, ..., D_K$$

We perform k learning-testing iterations. At the k-th iteration we have:

$$D_{T_k} = \{D_1, ..., D_{k-1}, D_{k+1}, ..., D_K\} \qquad D_{TS_k} = D_k$$

The set D_{T_k} is used for training while its complement set D_k is used for testing at the k-th iteration. The classification algorithm undergoes k training-testing phases, while the k estimates of the classifier's accuracy used to compute the arithmetic mean (which offers a more effective estimates of the classifier's accuracy):

$$acc = \frac{1}{K} \sum_{k=1}^{K} acc(D_k)$$

Different options exist to select the value of the parameter k. Typical values are $k = 3, 5, 10$. This type of cross validation is called **k-fold cross validation**, the most used one is the ten-fold cross validation.

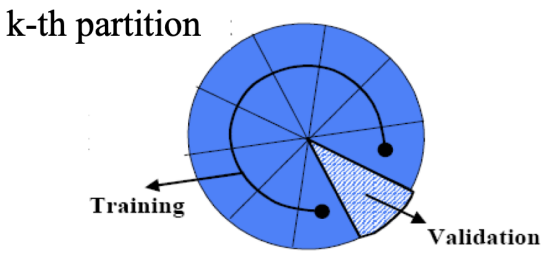


Figure 39: k-th partition in ten-fold cross validation.

Basically we train the model k time, and every partition is used as test set and the remaining $\frac{k-1}{k}$ of data are used as training set. In every step we estimate the model and we get the previsions on the test set without using that $\frac{1}{k}$ of obs

to train the model. The metrics doesn't change, but now they are computed with cross validation previsions.

A limit case applied when data are scarce in known under the name of **leave one out cross validation**(**LOOCV**) or **Jackknife method**. LOOCV is obtained by assuming that each record is a partition of the dataset, thus the value of k equals the number of recors of the dataset D.

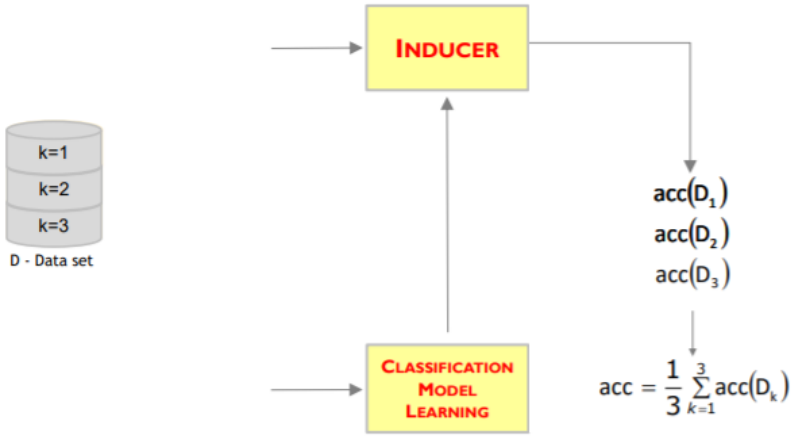


Figure 40: Cross validation process.

We usually ask that each partition of the dataset contains the *same proportion* for possible values of the class attribute. When class proportions are strongly different (unbalanced classes) we adopt specific sampling schemes, i.e. **stratified sampling**. It builds data subset trying to ensure that classes proportion are maintained form train to test data sets. We use the stratified sampling in order to have the same proportion of target between the folds.

It is advisable to trust more the **k-folds cross-validation** because we are shore that records are not repeated, contrary to the *iterated holdout*.

2.4 Comparing Classifiers (*)

It is often useful to compare the performance of different classification models to determine which one work better on a given data set. Depending on size of the data, the observed difference in accuracy between two classification models may not be statistically significant. The accuracy has different value if is computed with few records. For example:

- **Inducer A**: accuracy = 0.85 when tested on a test data set containing 30 records.
- **Inducer B**: accuracy = 0.75 when tested on a test data set containing 5000 records.

Is inducer A better than inducer B? Although inducer A has a higher accuracy than inducer B, it was tested on a smaller test set. *Accuracy alone is not enough. We need to estimate the confidence interval of the accuracy for the two inducers and test the statistical significance of the observed deviations.*

2.4.1 Confidence interval

We consider the task of predicting the value of the class attribute for a test record as a **binomial** experiment.

given a test set D_N containing N records, we have:

- X: number of records correctly predicted by a given inducer
- p: true, but unknown, accuracy of the inducer.

By modelling the prediction task as a binomial experiment, X has a binomial distribution with:

- $mean = N \cdot p$
- $\sigma^2 = N \cdot p \cdot (1-p)$

The object of out study is the **empirical accuracy** computed as:

$$acc = \frac{X}{N}$$

that has a binomial distribution with $\mu = p$ and $\sigma^2 = \frac{p \cdot (1-p)}{N}$

Although the binomial distribution can be used to estimate the confidence interval for accuracy, it is often **approximated by a normal distribution** when the number of records of the test set is sufficiently large. Based on the normal distribution, the following confidence interval for the empirical accuracy is obtained:

$$P\Big(-Z_{1-\alpha/2} < \frac{acc - p}{\sqrt{p \cdot (1 - p/N)}} < Z_{1-\alpha/2}\Big) = 1 - \alpha$$

Rearranging the above inequality leads to the following confidence interval with significance α (confidence $1-\alpha$) for p, the unknown classification model's accuracy:

$$\left[\frac{acc + \frac{Z^2_{1-\frac{a}{2}}}{2 \cdot N} - Z_{1-\frac{a}{2}} \cdot \sqrt{\frac{acc}{N} - \frac{acc^2}{N} + \frac{Z^2_{1-\frac{a}{2}}}{4 \cdot N^2}}}{\big(1 + \frac{Z_{1-\frac{a}{2}}{}^2}{N}\big)}, \frac{acc + \frac{Z^2_{1-\frac{a}{2}}}{2 \cdot N} + Z_{1-\frac{a}{2}} \cdot \sqrt{\frac{acc}{N} - \frac{acc^2}{N} + \frac{Z^2_{1-\frac{a}{2}}}{4 \cdot N^2}}}{\big(1 + \frac{Z_{1-\frac{a}{2}}{}^2}{N}\big)}\right]$$

Important: if you take several independent test sets it is obvious that this produce different confidence intervals. But in these cases, if we fix α, we can statistically establish which intervals are more significant than others. The more α decreases, the more β increases, i.e. the error case in which the null hypothesis is not rejected when instead I should reject it.

Given a classification model achieving an empirical accuracy $acc = 0.8$ on a test set consisting of 100 records, we want to know the confidence interval for its true accuracy at a 95% confidence level. This confidence level corresponds to $Z_{0.975} = q.96$ which brings to a $(0.711, 0.867)$ confidence interval for the true unknown accuracy of the considered classification model:

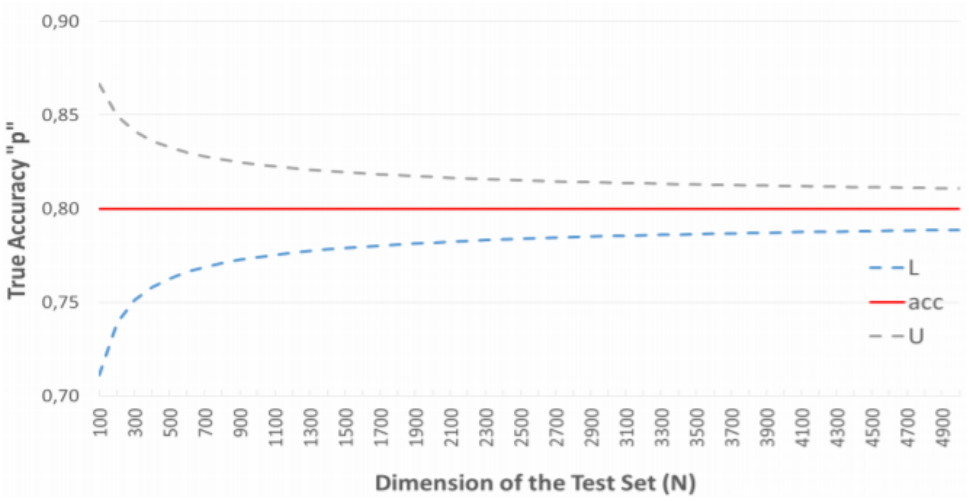


Figure 41: Trend of the confidence interval.

2.4.2 Different test set

Consider two classification models evaluated on different test sets (it is a common thing):

- M_1 evaluated on test set D_1 containing n_1 records achieves error rate e_1
- M_2 evaluated on test set D_2 containing n_2 records archives error rate e_2

The two test sets are assumed to be *independent*. Our goal is to test whether the observed difference between e_1 and e_2 is statistically significant. Assume that n_1 and n_2 are sufficiently large, the error rates e_1 and e_2 can be approximated using *normal distributions*.
If the observed difference in the error rate is dented as:

$$d = e_1 - e_2$$

it is also distributed according to a **normal** distribution with

$$\mu = d_t \qquad \sigma^2 = \sigma_d^2$$

The variance of d can be estimated as follows:

$$\sigma^2 \cong \hat{\sigma}_d^2 = \frac{e_1 \cdot (1 - e_1)}{n_1} + \frac{e_2 \cdot (1 - e_2)}{n_2}$$

The confidence interval for the true difference d_t:

$$\left(d - z_{1-\alpha/2} \cdot \hat{\sigma}_d, d + z_{1-\alpha/2} \cdot \hat{\sigma}_d\right)$$

We basically could have three situations:

1. If the confidence interval spans the value 0, we conclude that the observed difference d is not **statistically significant** at the α level (confidence $1-\alpha$).

$$0 \in \left(d - z_{1-\alpha/2} \cdot \hat{\sigma}_d, d + z_{1-\alpha/2} \cdot \hat{\sigma}_d\right)$$

2. If the upper limit of the confidence interval is negative then the classification model M_1 is better than the classification model M_2 at the $\frac{\alpha}{2}$ level (confidence $1 - \frac{\alpha}{2}$).

$$d + z_{1-\alpha/2} \cdot \hat{\sigma}_d < 0$$

3. If the lower limit of the confidence interval is positive then the classification model M_2 is better than the classification model M_1 at the $\frac{\alpha}{2}$ level (confidence $1 - \frac{\alpha}{2}$).
$$d - z_{1-\alpha/2} \cdot \hat{\sigma}_d > 0$$

It is advisable to proceed with the hypothesis test **only** if we ask if the sets are different or not. It is important to not do tests at random because otherwise we risk to have absurd results. We have to ask the question and only then proceed with the test on the couple of models. Due to the fact that there is a **problem of multiple comparisons**, we have to adapt all model to the same α level, otherwise the differences are not comparable.

2.4.3 Same test set

In the situation where is possible to use the **same test set** we can proceed with a more **powerful** test. It is possible to assess the second type error, the one when the difference between two classifiers is not statistically significant while in reality it is (β).

Consider two classification models M_1 and M_2 to be compared using the *K-folds cross validation* procedure. The dataset D is partitioned in to K disjoint subsets, exhaustive and with almost a constant dumber of records.

$$D_1, D_2, ..., D_K$$

We then apply classification techniques to construct classification models M_1 and M_2 from $k-1$ of the partitions and the test them on the remaining partition. This step is repeated k times, each time using a different partition as the test set. Let:

- M_{1k} **inducer** for model M_1 obtained at the k-th iteration with e_{1k} error rate
- M_{2k} **inducer** for model M_2 obtained at the k-th iteration with e_{2k} error rate

Each pair of models are tested on the same partition k. The difference between their error rates during the k-th iteration (fold) can be written as:

$$d_k = e_{1k} - e_{2k}$$

If K is sufficiently large, then d_k is *normally* distributed with:

$$\mu = d_t^{cv} \quad \sigma = \sigma^{cv}$$

The overall variance in the observed differences is estimated using the following formula:

$$\hat{\sigma}^2_{d^{cv}} = \frac{\sum_{k=1}^{K}(d_k - \bar{d})^2}{K \cdot (K-1)} \qquad \bar{d} = \frac{1}{K}\sum_{k=1}^{K} d_k$$

We use the **Student T** distribution to compute the confidence interval for the value of the true mean d_t^{cv}:

$$\left(\hat{d} - t^{K-1}_{1-\frac{\alpha}{2}} \cdot \hat{\sigma}_{d^{cv}}, \hat{d} + t^{K-1}_{1-\frac{\alpha}{2}} \cdot \hat{\sigma}_{d^{cv}}\right)$$

where

$$t^{K-1}_{1-\frac{\alpha}{2}}$$

It is obtained from a probability table, we get the quantile associated with confidence $1-\alpha$ and $K-1$ degrees of freedom.
Concerning the use of the confidence interval to establish statistical significance, the **same consideration** made for the case of two classifiers apply here: if the confidence interval spans the value 0, we conclude that the observed difference is *not statistically significant* at the α level (confidence $1-\alpha$).

2.5 Class Imbalance Problem (*)

Lets consider again the Churners dataset. In particular, the 14.5% of costumers are churners. A classification model which label all records of the data set ass non churners achieves an accuracy value equal to 85.5% because it is the most frequent class in the dataset.

Definition 2.23 (ZeroR rule)**. ZeroRule** is defined as that model which has always the most frequent class as output. It is basically an useless model.

The *accuracy* measure treats every class as equally important, it may not be suitable for analyzing **imbalanced datasets**, where the rare class is considered *more interesting* than the majority class. For binary classification, the rare class is often denoted as *positive class* while the majority class is denoted as *negative class*. We already introduced the concept of confusion matrix:

		INDUCER PREDICTION (IP)	
		-1	+1
ACTUAL CLASS (AC)	-1	TN	FP
	+1	FN	TP

Figure 42: Binary confusion matrix.

From the matrix we can compute the following measures:

- **TNR**, **True Negative Rate** or **Specificity**: fraction of negative records predicted correctly by the classification model

$$TNR = \frac{TN}{TN + FP}$$

- **TPR**, **True Positive Rate** or **Sensitivity**: fraction of positive records predicted correctly by the classification model

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

- **FPR**, **False Positive Rate**: Fraction of negative records predicted as a positive class by the classification model

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

- **FNR**, **False Positive Rate**: fraction of positive records predicted as a negative class by the classification model

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

Recall and **Precision** are two widely used metrics employed in applications where detection of one of the classes is considered to be more important than detection of the other class.

Definition 2.24 (Precision)**. Precision** determines the fraction of records that actually turns out to be positive in the group the classification model has declared as positive.

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

We take the classification model point of view, evaluating how much of the positive it classified as positive. The *higher* the precision is, the *lower* the number of false positive errors committed by the classification model.

Definition 2.25 (Recall)**. Recall** measures the fraction of positive records correctly predicted by the classification model.

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

In this case we take the reality point of view. *Large* recall means very *few* positive records *misclassified* as negative class. In fact, recall is equivalent to the true positive rate (TPR).

Back to the churn example, assume the following:

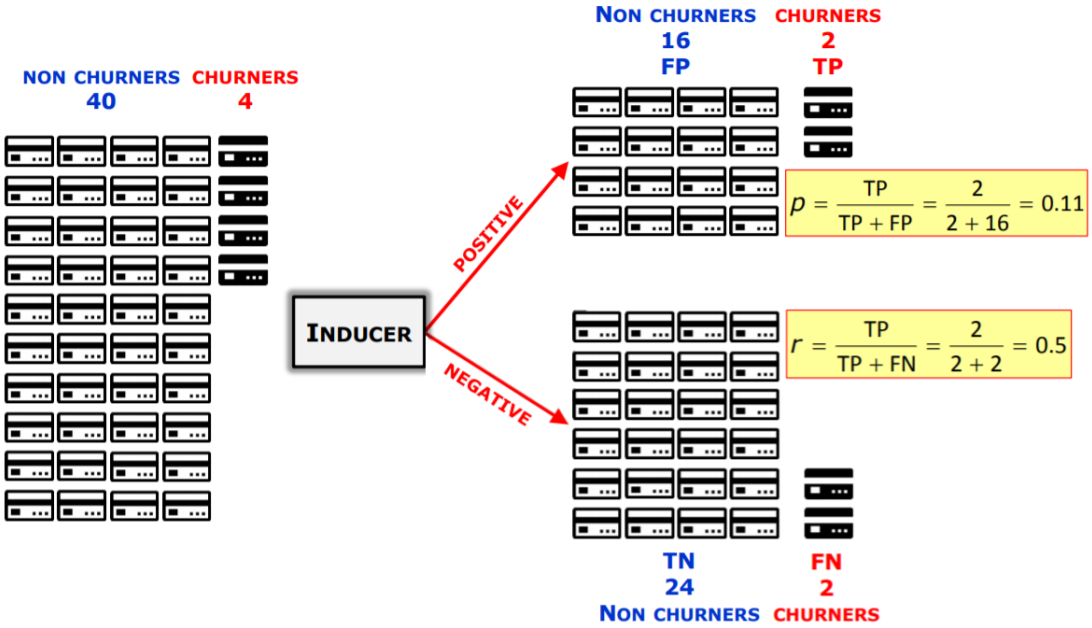


Figure 43: Example of computing precision and recall.

There is a *close bond between this two measures*, if we have an high recall probably we will have a low precision and vice versa. It also could be the case that this quantity are not feasible because a 0/0 fraction.

So it is often possible to construct baseline models that maximize one metric but not ht other. Precision and recall are usually summarized into another metric:

Definition 2.26 (F_1 measure)**.** The F_1 **measure** is the harmonic mean between recall and precision.

$$F_1 = \frac{2 \cdot r \cdot p}{r + p}$$

A high value of F_1 measure ensures that both recall and precision are reasonably high. A generalization exists under the name F_β **measure** to examine the tradeoff between recall and precision:

$$F_\beta = \frac{(\beta^2 + 1) \cdot r \cdot p}{r + \beta^2 \cdot p}$$

- F_β with $\beta = 0$ is the precision
- F_β with $\beta = \infty$ is the recall

2.6 Counting the cost (*)

2.6.1 Cost matrix

Back on the churner problem, the company wont to avoid that clients left, basically we try to predict who wants to leave and we try to prevent this prediction by applying dissuasion policies. The company can spend a certain budget to reverse this phenomenon, we must identify possible churners and implement the dissuasion policies **only** on them (if we did it for all customers it would make no sense regarding the costs).
We have to convince the company that the model developed by us is better than the one they have historically used. The most used measure to achieve this is the accuracy related to the confusion matrix. In addition we have to the associate **cost matrix** with it.

Definition 2.27 (Cost matrix)**.** The **cost matrix** decides how much cost (work/space/money/tim the company incurs to classify an observation, according to false positives and false negatives.

The cost is computed as follows:

		PREDICTED CLASS	
		-1	+1
ACTUAL CLASS	-1	TN	FP
	+1	FN	TP

		PREDICTED CLASS	
		-1	+1
ACTUAL CLASS	-1	p	q
	+1	q	p

$$N = TP + FN + FP + TN$$

$$\text{ACCURACY} = (TP+TN)/N$$

$$\begin{aligned}\text{COST} &= p \cdot (TP+TN) + q \cdot (FN+FP)\\ &= p \cdot (TP+TN) + q \cdot (N-TP-TN)\\ &= q \cdot N - (q-p) \cdot (TP+TN)\\ &= N \cdot [q - (q-p) \cdot \text{ACCURACY}]\end{aligned}$$

Figure 44: Link between confusion matrix (left) and cost matrix (right)

$$Cost = C_{--} \cdot TN + C_{-+} \cdot FP + C_{+-} \cdot FN + C_{++} \cdot TP$$

Important: if the cost matrix is symmetric the cost is the accuracy. We'll now see an example in order to understand this concept. We compare the accuracy's performance and the cost, alias the *standard Mailout procedure* (**SMP**). This procedure was already use for deter the client, with the new classification model **M** that we built:

SMP		PREDICTED CLASS	
		-1	+1
ACTUAL CLASS	-1	830	111
	+1	45	114

ACCURACY = 0.858

M		PREDICTED CLASS	
		-1	+1
ACTUAL CLASS	-1	931	10
	+1	57	102

ACCURACY = 0.939

Figure 45: Cost matrix: SMP on the left and model M on the right

Now we have to use the **cost matrix** and it turns out:

COST = 4,497

		PREDICTED CLASS	
		-1	+1
ACTUAL CLASS	-1	0	1
	+1	100	-1

COST = 5,608

Figure 46: Churner cost matrix

As we can see, although our model has better **accuracy** measure, it also has higher **cost** (in terms of money) than the model historically used by the company. Therefore, in this case, the old SMP procedure will be preferred rather than our new solution. It is important to note however that there are situations in which losing a customer is more serious than losing another one. We need to understand if we're using the true cost matrix, or if it's only representative.

2.6.2 Cumulative Gains

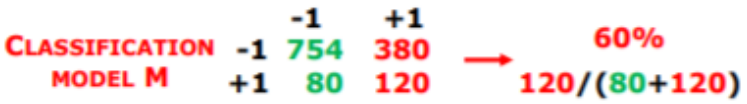
Lets suppose that we have a population of 1334 clients, where 500 are possible churners. For experience we know that the 15% of clients is a churner, so:
$1334 * 0.15 = 200$ churners on the total;
$500 * 0.15 = 75$ churner on potential churners.

If we consider as positive the churner class and negative the non churner class and we extract with casual sampling on them, we will have the following result:

		-1	+1	
RANDOM SAMPLING	-1	709	425	→ 37.5%
	+1	125	75	75/(125+75)

If applied, the developed classification model identify the 60% of the effective churner on the potential ones:

$200 * 0.6 = 120$ churners correctly classified.



So we can say that our model is way better than the *ZeroRule* (the casual sampling).

Definition 2.28 (Lift factor)**.** The **lift factor** is the proportion between the model M and the casual sampling:

$$Lift = \frac{performace(M)}{performance(zeroRule)}$$

This coefficient is used for computing the increase of models' performance. In this case: $0.6/0.375 = 1.6$

Important: our classifier is work well for identifying simple cases but could make a lot of mistakes in complex ones. As it is forced to answer it will tend to answer in less efficient way.

The *Lift factor* can now be used for understanding the **profitability** once the costs involved are known. Given a classification model which output the predicted probabilities for the positive class. We must find subset of the customers set that have *high proportion* of positive records, *greater* than the starting dataset.

Lets see how generate a lifting:

1. **Sort** the output in decreasing order of the probability**prob(y)**
2. **Extract** a first subset of a given size with the greatest possible proportion of positive records, just read the requisite number of records off the list, starting at the top and compute the*lift factor*, it will be high
3. **Consider** a bigger subset (always starting form the top) and compute the *lift factor*, that supposedly will be lower
4. **Repeat** the point 3 until all records are included

In the example we see two step of the procedure:

Rank	RowID	Prob(y)	Churn
1	134	0.95	y
2	221	0.88	y
3	18,923	0.86	n
4	90,034	0.73	n
5	874,823	0.64	n
6	67	0.47	n
7	98,324	0.32	n
8	2,553	0.15	y
9	289	0.10	n
10	2,349	0.03	n

subset consisting of 3 records

2 records out of the 3 positive ones are correctly identified

2/3 =0.66 of positive records

Lift = 0.66/0.3 = **2.22**

Rank	RowID	Prob(y)	Churn
1	134	0.95	y
2	221	0.88	y
3	18,923	0.86	n
4	90,034	0.73	n
5	874,823	0.64	n
6	67	0.47	n
7	98,324	0.32	n
8	2,553	0.15	y
9	289	0.10	n
10	2,349	0.03	n

subset consisting of 5 records

2 records out of the 3 positive ones are correctly identified

2/3 =0.66 of positive records

Lift = 0.66/0.5 = **1.33**

Figure 47: **Lift factor** computation: first subset on the left and second one on the right.

Definition 2.29 (Cumulative Gains)**.** The computed value of the lift factor form the so called **cumulative gains**.

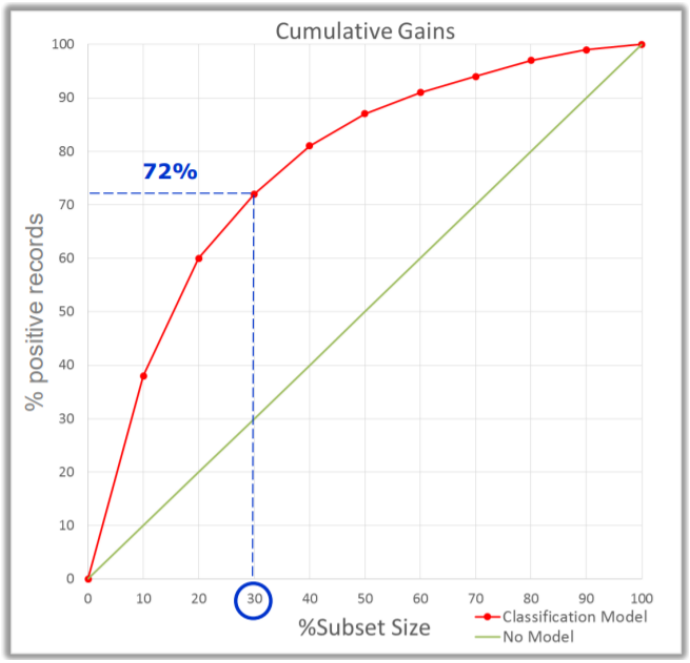


Figure 48: Cumulative gains spotting a point of high added value.

Important: The green line calculates the cumulative gain for a model that produce the outputs in a *random* way, the x axis represent *recall*, the correctly recognized positives (repeating the test at each chosen step).

2.6.3 Lift Chart

The *cumulative gains* can be mapped to the **lift chart**.

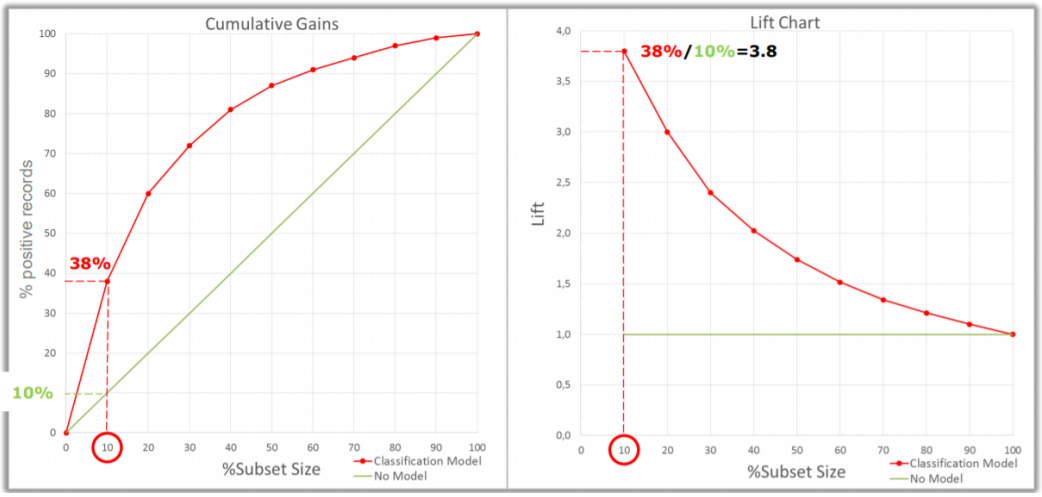


Figure 49: Cumulative gain (left) and lift chart (right)

This is useful to evaluate when the classification model lose effectiveness. It is important to spot the point where the accuracy is maximised in relation with the percentage of instances.

2.6.4 ROC curve

Lift are closely related to another graphical technique.

Definition 2.30 (ROC curve)**.** The Receiver Operating Characteristic curve or more simply the **ROC curve** is a graphical technique for evaluating classification models.

It is similar to the cumulative gains but on axes it has:

- **y axe**: the number of positive records included in a selected subset of records, expressed as percentage of the total number of positive records (**TPR**)
- **x axe**: the number of negative records included in the subset, expressed as a percentage of the total number of negative records (**FPR**)

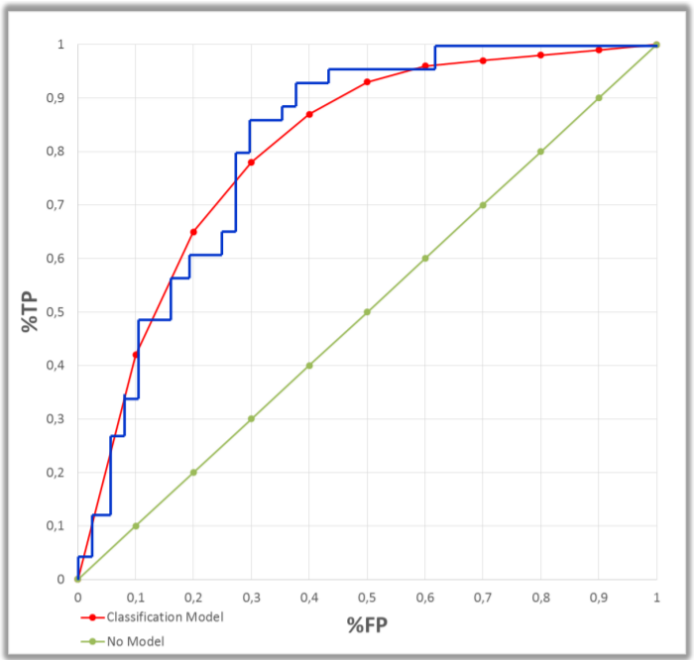


Figure 50: ROC curve

The jagged ROC curve in blue depends on the details of the particular subset of data. This dependence can be reduced by applying cross-validation. The ROC curve depicts the performance of a classifier without regard to class distribution or error costs. So it is useful if for confronting different classification models in order to understand where a classifier is more efficient or less efficient that the others. Like in the following example.

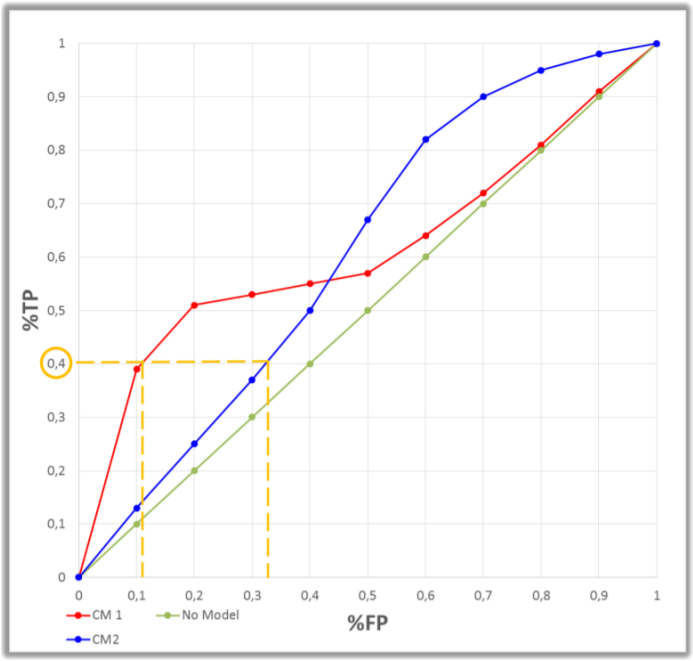


Figure 51: ROC comparison between two models.

A classification model is **preferable** to another if its ROC curve is over the other one.

Referring to the example, CM1 excels if a small, focused subset is sought; that is, if we are working towards the left-hand side of the graph. Clearly, if we

aim to cover just the 40% of the true positive records we should choose the CM1, which gives a false positive rate of around 10%, rather than CM2, which gives over 30% false positive records. CM2 excels if we are planning a large subset: if we are covering 80% of the true positive records, CM2 will give false positive rate around 60% as compared with the CM1's 80%.

2.7 Feature Selection (*)

Often is better not to use all the possible attributes to develop a classification model. This because some attribute can only bring noise and aggravate computational burden. Because of that we usually work with a subset of the features, selecting the best and more informative ones. La **feature selection** is a very important step and it is strictly linked with the problem solution. By analysing the relationship between the variables, in the pre processing step, we can help to find a better solution to the problem.

Definition 2.31 (Feature selection)**. Feature selection** is a process that wont to find if and which attribute are:

- *Redundant*: contains information already available due to one of more other attributes.
- *Irrelevant*: contains information not useful to solve the considered data mining task.

Different **approaches** are available:

- *Brute-force*: to apply all possible subset of the available attributes as the input attributes to the classification model. It is feasible only if there is a small number of attributes, basically it is not an option in real situations. There are $sum_{n=1}^{10}\binom{10}{n}$ possible combinations.
- *Embedded*: attributes selection occurs as a by-product of classification learning.
- *Filter*: attribute are selected before learning the classifier through an objective function.
- *Wrapper*: a classifier is used to fine the optimal subset of the available attributes, it is the most effective way to proceed but it require more computational resources.

Lets see the Filter and the Wrapper approach in comparison.

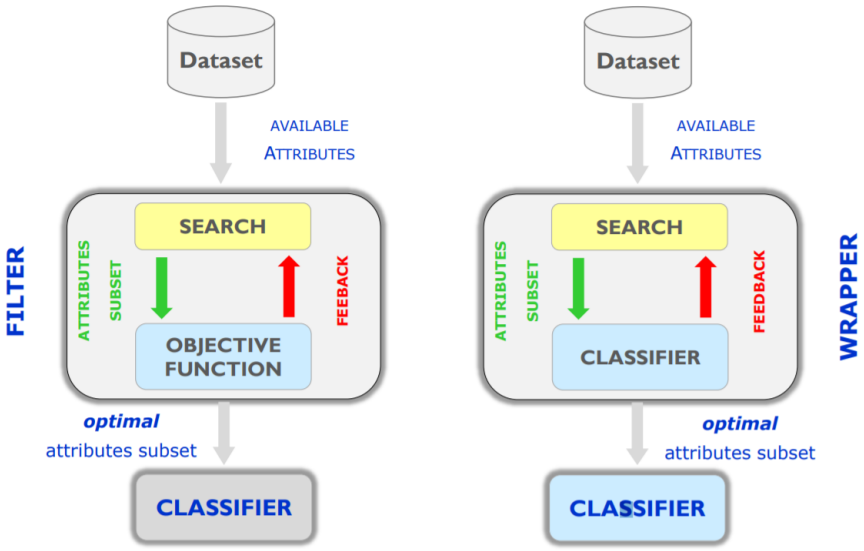


Figure 52: Filter - Wrapper

We cannot do a feature selection on all the dataset and then train only on a subset, because the outputs aren't comparable.

The main **advantages** for the feature selection are:

- Reduction of the cost of data collection
- Reduction of the inference time, i.e., time required by classifier to predict the value of the class attribute
- Increased interpretability
- Increased accuracy

The main **motivation** for using this approach are:

- To avoid the overfitting phenomena
- To develop a faster and cost-effective classifier
- To improve the understanding of the data generation process

It is important to proceed following the below flowchart:

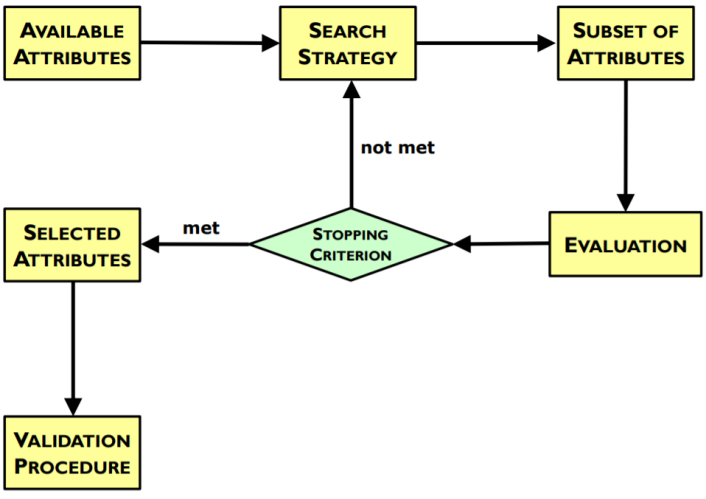


Figure 53: Flowchart for feature selection.

2.7.1 Embedded

Embedded methods are a catch-all group of techniques which perform feature selection as part of the model construction process. The exemplar of this approach is the LASSO method for constructing a linear model, which penalizes the regression coefficients with an L1 penalty, shrinking many of them to zero. Any features which have non-zero regression coefficients are 'selected' by the LASSO algorithm. These approaches tend to be between filters and wrappers in terms of computational complexity.

Embedded methods have been recently proposed that try to combine the advantages of both previous methods. A learning algorithm takes advantage of its own variable selection process and performs feature selection and classification simultaneously.

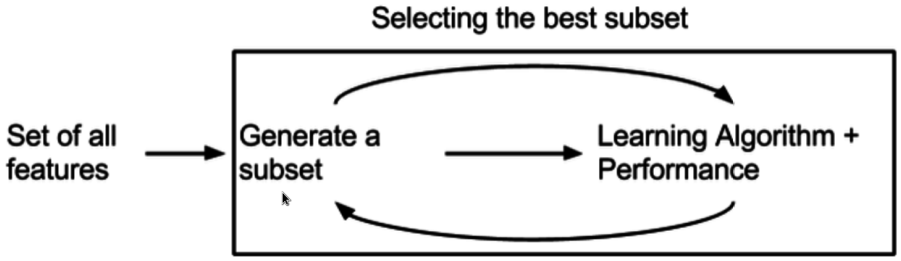


Figure 54: Embedded method.

The idea of this technique is to estimate a model minimizing the Ordinary Least Square (**OLS**) but using also a **budget** on the total of estimated coefficients.

$$OLS = min(\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2)$$

The point is to put a function of the sum of the coefficients as a budget in the loss function to estimate the parameters (the purpose is to reduce the estimate of the coefficients).

The **Ridge regression** minimize the sum of residual sum of square with a budget function that is the sum of the square of the parameters' weights.

$$Ridge = min(\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\sum_{j=1}^{p}\beta_j^2)$$

Where λ is the regularization parameter balancing the OLS fit (sum of square residuals) and the magnitude of parameters values. For every different λ value we will have a different vector of β estimated coefficients. λ has to be tuned.

$$\lambda \in [0, +\infty)$$

When $\lambda \to 0$ the budget has no effect so $\hat{\beta}^{ridge} \to \hat{\beta}^{OLS}$, when λ is to small we can overestimate the data and the model will have an high variance. On the other hand when $\lambda \to +\infty$ the budget has the maximum effect so $\hat{\beta}^{ridge} \to 0$, the model is to simplistic because it has only the intercept, it result in high bias. Basically while λ rise also the bias rise, but the variance decrease. The problem with ridge regression is that it never manages to reach 0 as parameter values, so it cannot be used as a model selector.

The problem of the Ridge regression has been resolved by **LASSO**. It change the budget with the sum of β absolute values. It has all the Ridge properties and it can be used as feature selector.

$$LASSO = min(\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j|)$$

This method is feasible also in more large situations (continuous target).

Improvements to the LASSO include Bolasso which bootstraps samples; Elastic net regularization, which combines (weighed mean) the L1 penalty of LASSO with the L2 penalty of ridge regression; and FeaLect which scores all the features based on combinatorial analysis of regression coefficients. AEFS further extends LASSO to nonlinear scenario with autoencoders.

2.7.2 Filter

Filter methods use a proxy measure instead of the error rate to score a feature subset. This measure is chosen to be fast to compute, while still capturing the usefulness of the feature set. Common measures include the mutual information, the point-wise mutual information, Pearson product-moment correlation coefficient, Relief-based algorithms, and inter/intra class distance or the scores of significance tests for each class/feature combinations. Filters are usually less computationally intensive than wrappers, but they produce a feature set which is not tuned to a specific type of predictive model. This lack of tuning means a feature set from a **filter is more general than the set from a wrapper, usually giving lower prediction performance than a wrapper**. However the feature set doesn't contain the assumptions of a prediction model, and so is more useful for exposing the relationships between the features. Many filters provide a feature ranking rather than an explicit best feature subset, and the cut off point in the ranking is chosen via cross-validation. Filter methods have also been used as a preprocessing step for wrapper methods, allowing a wrapper to be used on larger problems. One other popular approach is the Recursive Feature Elimination algorithm, commonly used with Support Vector Machines to repeatedly construct a model and remove features with low weights.

Filter type methods select variables regardless of the model. They are based only on general features like the correlation with the variable to predict. Filter methods suppress the least interesting variables. The other variables will be part of a classification or a regression model used to classify or to predict data. These methods are particularly effective in computation time and robust to overfitting.

Filter methods tend to select redundant variables when they do not consider the relationships between variables. However, more elaborate features try to minimize this problem by removing variables highly correlated to each other.

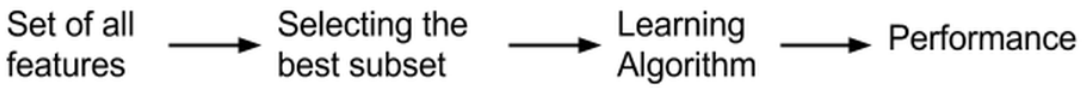


Figure 55: Filter method.

The filter approach can be uni or multi variate. In case of **uni-variate** attributes:

1. We chose an *association measure* between candidate attributes and class attribute (mutual information, gain ratio, ...)
2. We **sort** the candidate attributes according to the association measure
3. We **select** the R best candidate attributes as input attributes for classification learning

Usually with this procedure it is like that **irrelevant attributes** are correctly identified (removed), while *redundant attributes* are not correctly identified (not relevant). In case of **multi-variate** attributes:

- We jointly identify *irrelevant* and *redundant* attributes
- A good subset of attributes contains attributes which are strongly associated with the class attribute but uncorrelated among them (symmetric uncertainty measures or correlation-based measures)

Lets see the most used techniques:

Uni-Variate	Multi-Variate
Parametric	Correlation Feature Selection (CFS)
t-test	Relief
ANOVA	Blanket
Mutual Information	
Non Parametric	
Mann-Whitney	
Kruskal-Wallis	
Permutation test	

Figure 56: Filter techniques

	Advantages	Disadvantages
Uni-Variate	speed scalability independent on the classifier	ignore that attributes can be dependent ignore interactions with the classifier
Multi-Variate	model dependency between attributes independent on the classifier computational cost compares favorably to wrapper	slower than Uni-Variate techniques less scalable than Uni-Variate Techniques ignore interactions with the classifier

Figure 57: Advantages and disadvantages of using uni or multi variate techniques.

2.7.3 Wrapper

Wrapper methods use a predictive model to score feature subsets. Each new subset is used to train a model, which is tested on a hold-out set. Counting the number of mistakes made on that hold-out set (the error rate of the model) gives the score for that subset. As wrapper methods train a new model for each subset, they are very computationally intensive, but usually provide the best performing feature set for that particular type of model or typical problem.

Wrapper methods evaluate subsets of variables which allows, unlike filter approaches, to detect the possible interactions amongst variables. The two main disadvantages of these methods are:

- The increasing overfitting risk when the number of observations is insufficient
- The significant computation time when the number of variables is large

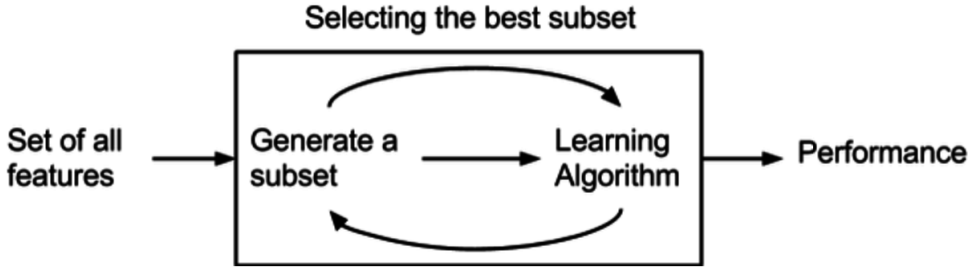


Figure 58: Filter method.

Usually **decision tree** models are the best option for this method because they are not very complex and they are easily readable. Typically the algorithm **C4.5 or J48** is solid option.

A more interesting and performing model to use is the **random forest**, which can be also used for feature selection. Basically there are tow ways to do it. The first one is to calculate the average improvement of the trees in the splits produced by each attribute. An example of such a measure in classification is the Gini available in random forest implementations. The second and more interesting way is **Boruta**, it is a "permutation accuracy importance" measure. By randomly permuting the the explanatory attribute x_j, its association with the target y is stopped. If the original variable x_j is matched to the answer ,when the permuted attribute x_j, with the remaining non-permuted (randomly selected) ones, is used to predict the output, the prediction accuracy (i.e., the number of correctly classified observations) decreases substantially. So we can see how the measure of attribute importance for x_j is the difference in accuracy between the prevision with and without the attribute x_j.

2.7.4 Feature Creation

It is often possible to create, form the original attributes, a new set of attributes that captures the relevant information in the data set much more effectively. Furthermore, the number of new attributes can be smaller than the number of original attributes, allowing us to reap the benefit of dimensionality reduction. Some methodologies for creating new attributes (features) are:

- **Feature extraction**: a much broader set of classification models can be applied, they are highly domain-specific, like image processing.
- **Mapping the data to a new space**: a totally different view of the data can reveal interesting features which increase understanding and classification performance, it is typically used in deep-learning.
- **Feature construction**: the available attributes contain information that is not suitable for some classification algorithms, one or more new features constructed out of the original attributes can be more useful than the original features.

2.7.5 Regularization

Some machine learning models allow hyper-parameters. It has been demonstrated that a good choice of neural network architecture allows to approximate any problem. However, it is not free from overfitting, on the contrary, since it is not always clear how it works (black box model), it is more difficult to evaluate when it overfit. We try to choose the parameter allocation that reduces the quadratic error of the model as much as possible, therefore the hyper-parameter λ called regularization. It is a parameter on the loss function as the magnitudes of the fitting parameters increase, there will be an increasing penalty on the cost function (similar to the on seen for Ridge and Lasso):

$$E(\mathbf{w}, \lambda) = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 + \frac{\lambda}{2}\sum_{j=1} K w_i^2$$

K: number of parameters free(weights + thresholds) from neural network
λ: regularization parameter
The first part of the formula is about fitting the function, the second part is about the flexibility we can access.

We should not use the training set to tune the λ regularization parameter. That would overfit the model. If the test set is used to optimize the lambda parameter then the dataset has burned and we have to redo the model, the goal is always to recognize data never seen before, for this reason different dataset division schemes are adopted.

2.7.6 Schema division

Optimal schema division could be the following:

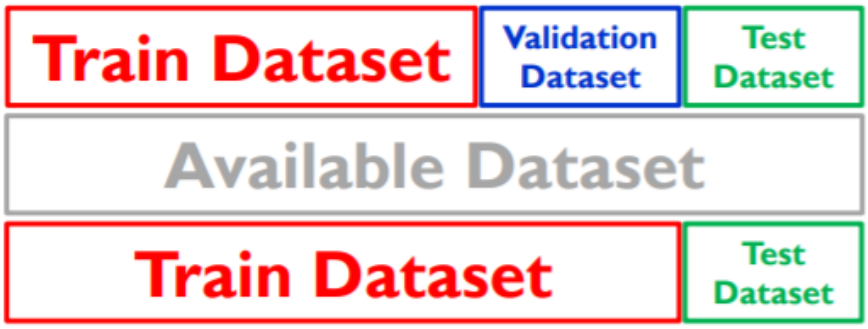


Figure 59: Different dataset division schema.

- The train dataset is used to train the model (on the w parameters)
- The λ parameter is optimized using the validation set
- The test set is used to provide an unbiased performance estimate
- Can be done via holdout, iterated holdout and cross-validation

When **Filter** approach is used for feature selection usually it is preferable a **Train-Test** schema.

- Relevance and/or redundancy are estimated using the Train set

- After selecting the attributes the Train set is used to train the classifier
- Performance estimates are obtained by applying the classifier to the Test set

When **Wrapper** approach is used for feature selection usually it is preferable a **Train-Validation-Test** schema.

- The Validation set is used to optimize performance when different attributes are used for the model (the wrappers)
- Selecting optimal subsets of attributes is the same as selecting the optimal value of the regularized parameter λ
- Once the attributes are selected, the Train set and the Validation set are merged and used to train the classifier (typically the same type of classifier used for feature selection)
- Performance estimates are obtained by applying the classifier to the Test set

2.8 Non Binary Classification

It may be the case that we are asked to solve a **non binary classification** problem, e.i., a classification problem where the class attribute is:

- **Multi-class**: a single class value is allowed
- **Multi-label**: no one, one or more class values are allowed

There could be also problem of **ranking** where the class value are in a specific order. A non binary classification problem is typically solved by applying the **One-Vs-All** (or One-Against-All) transformation.

2.8.1 One-Vs-All

The idea is to transform a multi-class problem into many binary class attributes. In this way, we can verify whether or not the current set has each of the evaluated characteristics.

We must be remembered that binarizing is not always necessary. In particular it is not necessary for Naïve Bayes, but it is for a decision tree.
In **One-vs-all** different numbers of *binary classifiers* is created according to the number of modes of the class attribute; it is also necessary to normalize the output of the classifiers. In the *Multi-class* model it is necessary to collect the values resulting from the classifiers and set a threshold above which the result is considered significant (potentially more than one class exceeds the threshold).

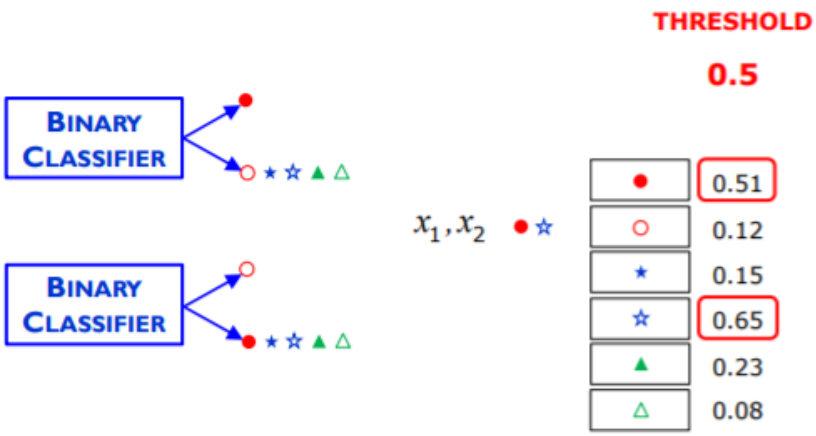


Figure 60: Example of binary classifiers.

3 Clustering

3.1 Introduction(*)

The cluster analysis is largely used to resolve many practical problem. In particular the **cluster analysis** is applied for two main purposes:

- **Understanding**: classes, or conceptually meaningful groups of objects that share common characteristics, play an important role in how people analyze and describe the world. This happens in biology (hierarchical classification of all living things), information retrieval (clustering users queries), business (segmenting customers for marketing activity).
- **Utility**: provides an abstraction from individual data objects to the cluster in which these data objects reside, More precisely, the goal is *to find the most representative cluster prototypes* (an object that is representative of the other objects in the same cluster). Specific sub-goal are; summarization (dimensionality reduction), compression 9each cluster prototype is associated in an integer) and efficiently finding nearest neighbors.

We wont now a more complete and formal definition of cluster analysis.

Definition 3.1 (Cluster analysis)**. Cluster analysis** or **clustering** is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters), based on information found in the data that describes the objects and their relationships.. Cluster analysis itself is not one specific algorithm, but the general task to be solved.

The goal is rather simple, it is that:

- The objects within a group should be similar (related) to on another and different from (unrelated) the objects in the other groups.
- The grater is the *similarity* (or homogeneity) within a group and the greater is the *difference* between groups, the better or more distinct is clustering.

In clustering there is **no longer distinction between training and test sets**, all data are used, because the analysis is for the present data, not for the future ones like in classification.

Problem: in many applications the notion of cluster is not well defined.

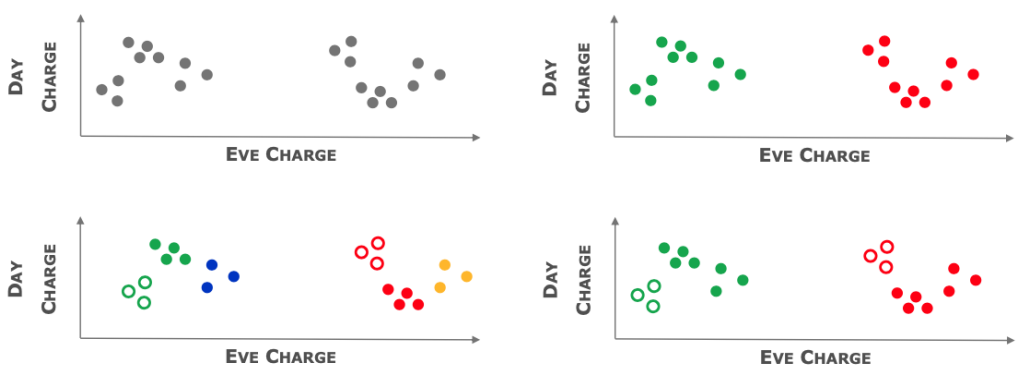


Figure 61: Example of the difficulty of deciding what constitutes a cluster.

The definition of cluster is **imprecise** and the best definition depends on the nature of data and the desired results.

3.1.1 Clustering types

Clustering is to create a set of clusters. There are several types of cluster analysis.

- Partitional vs Hierarchical
- Exclusive vs Overlapping vs Fuzzy
- Complete vs Partial.

We are going to see all the definitions of them.

Definition 3.2 (Partitional)**.** A clustering is called **partitional** if the dataset is separable in non stackable sets, such that an element belong only to one set.

Definition 3.3 (Hierarchical)**.** A clustering is called **hierarchical** if every cluster can be divided itself in another sub-cluster, in this case the clustering is a set of clusters organised like a tree.

Definition 3.4 (Exclusive)**.** A clustering is called **exclusive** if a subject is assigned to a single cluster.

Definition 3.5 (Overlapping)**.** A clustering is called **exclusive** if every subject can be assigned to more than one cluster.

Definition 3.6 (Fuzzy)**.** A clustering is called **fuzzy** if every subject can be assigned to more then one cluster at the same time with a value that consider the weight with reference to the belonging to a single cluster, the sum of weights is 1.

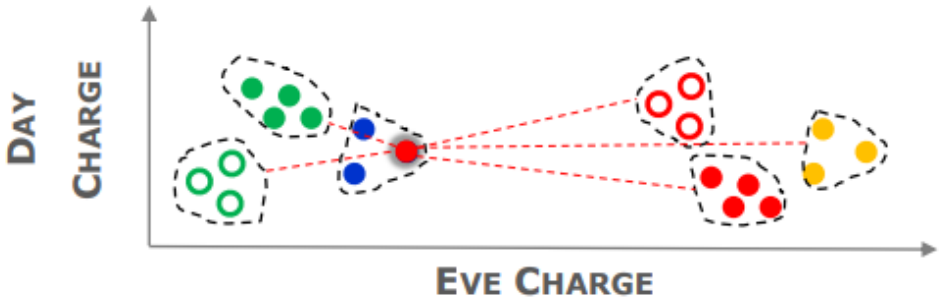


Figure 62: Clustering with fuzzy model.

Definition 3.7 (Complete)**.** A clustering is called **complete** if every subject is assigned to a cluster (there are no free subjects).

Definition 3.8 (Partial)**.** A clustering is called **partial** if exist at least one subject that doesn't belong to any cluster. This model is used because could be some outliers, considering them in a cluster could change the clusters forms in an irreversible wrong way.

3.1.2 Clusters' categories

It is believed that natural clusters really exists, in fact so it is, but is vary difficult to happen. Data as bidimensional points are used in order to understand the difference between different types of data. The category in which different clustering algorithms are divided are the following:

- **Well separated Cluster**: given a cluster, each object is closer to every object in the cluster it belongs to than to any other object in any other cluster. Such a well-formed cluster allows for very clear separations, but this type of thing happens very rarely in reality.
- **Prototype-based cluster**: given a cluster, every object in that cluster is closer to the prototype that defines the cluster than to every prototype in other clusters. The prototype is usually the *centroid* of the cluster. The *prototype* of a cluster corresponds to the subject best represented by the cluster (it can also be fake). This type of clusters tend to be globular. Typically the prototype after the creation of the clusters is not further considered in analysis.
- **Density-based cluster**: a cluster is a dense region of objects bounded by a low density region. These are used when clusters are irregular or intermittent, or when there is a lot of noise or outliers. Low density regions are used to separate the clusters.
- **Graph-based cluster**: if the data is represented by graphs, the nodes represent objects and the links connect the objects. Then each cluster is a connected component. The connection can also be weighed and chosen based on a certain threshold. These clusters are heavily used as there is a lot of research already done.

3.1.3 Cluster analysis components

The first phase of a cluster analysis is the *feature selection* which ensures that the attribute of the dataset are worthy of significance. Afterwards, there is the *feature extraction* phase, which produce features that could be better suited to discover the data structure, however this practice could generate features that are difficult to understand. We should use features that allow to distinguish the patterns of the elements that belong to the different clusters, that are immune to noise and easy to interpret.

The second step is to *determine the proximity measure and construct the merit function.* Once we have determined a proximity measure, the clustering problem translates into an optimization problem for a specific function.

It is important to remember that different clustering algorithms lead to different conclusions (sometimes completely different), this is the reason why we should not prefer any algorithm but compare the results obtained and draw conclusions from them.

3.2 Proximity(*)

Proximity is a fundamental tool to evaluate how a clustering algorithm work. This heavily influences the solution of a clustering problem. We have to decide the measure with we are going to approximate the similarity between elements in a cluster.

3.2.1 Introduction

Cluster analysis is rooted into the concepts of *similarity* and *dissimilarity*, and in many cases, once the similarities and dissimilarities have been computed the available data set is not needed anymore.

Definition 3.9 (Similarity)**.** The **similarity** between to objects (records) is a numerical measure of the degree to which the tow objects are alike. It is higher for pairs of objects that are more alike. Usually it is non-negative and are often in the interval $[0, 1]$, but it is also common for them to take values in the interval $[0, +\infty)$.

Definition 3.10 (Dissimilarity)**.** The **dissimilarity** between tow objects is a numerical measure of the degree to which the two objects are different. Dissimilarities sometimes fall in the interval $[0, 1]$, but it is also common for them to take values in the interval $[0, +\infty)$. Dissimilarities are lower for more similar pairs of objects.

For convenience, the term **proximity** will be used to refer to either similarity or dissimilarity. Transformation are often applied to convert a similarity (**s**) to a dissimilarity (**d**) or vice versa.

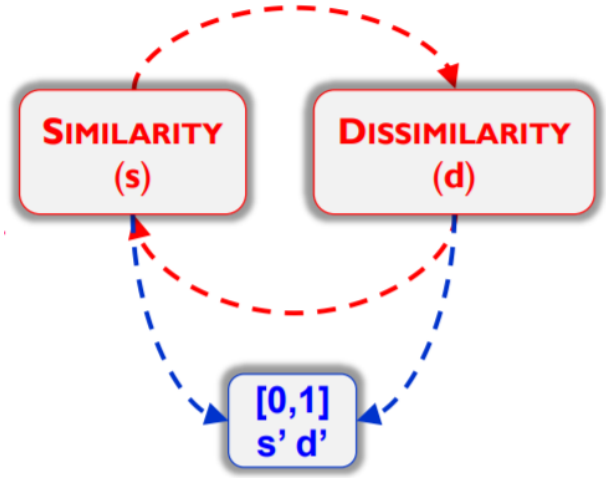


Figure 63: Relation between similarity and dissimilarity.

Frequently, proximities, especially similarities, are defined or transformed to take values on [0,1], in such a way to use a scale in which proximity indicates the fraction of similarity (dissimilarity) between two records.

In general, the new similarity s' and the new dissimilarity d' are defended as follows:

$$s' = \frac{s - \min s}{\max s - \min s} \qquad d' = \frac{d - \min d}{\max d - \min d}$$

There are many complication in mapping proximity measures to $[0, 1]$. If the proximity measure takes values on $[0, +\infty]$, then a **non-linear** transformation is needed and values will not have the same relationship to one another on the new scale. There can be used transformations like the following one:

$$d' = \frac{d}{1 + d}$$

With this transformation larger values of the original dissimilarity (d) scale are compressed in to the range of values near 1 for d', whether or not this is desirable depends on the application. Transforming similarities to dissimilarities and vice versa is also relatively straightforward although we again face the issues of preserving meaning and changing a linear scale in to a non-linear scale. We could use something like the following:

$$\text{if } s, d = [0, 1] \text{ then } s = 1 - d$$

Another approach is to define similarity as the negative of the dissimilarity (or vice versa).

$$s = -d$$

In general any monotonic decreasing function can be used to transform similarities in to dissimilarites, or vice versa.

Definition 3.11 (Proximity)**.** The **proximity** between two records is a function of the proximity between the corresponding attributes of the two records.

We first describe how to measure the proximity between records having only one simple attribute, then the case of records with multiple attributes is considered.

ATTRIBUTE TYPE		DISSIMILARITY	SIMILARITY
CATEGORICAL (QUALITATIVE)	NOMINAL	$d = \begin{cases} 0 & \text{if } x = y \\ 1 & \text{otherwise} \end{cases}$	$s = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases}$
	ORDINAL	$d = \frac{\lvert x - y \rvert}{n - 1}$	$s = 1 - d$
NUMERIC (QUANTITATIVE)	INTERVAL	$d = \lvert x - y \rvert$	$s = -d \quad s = exp(-d)$ $s = \frac{1}{1 + d}$
	RATIO		$s = 1 - \frac{d - min\_d}{max\_d - min\_d}$

Figure 64: Table for the measure of proximity of records with only one attribute, considering the type of attribute.

Consider two records consisting of the same single nominal attribute, all we can say is that they either have the same value or they don't. For all binary attributes the similarity excludes dissimilarity with 0 and 1 values.

Consider records consisting of the same single ordinal attribute, we assign an integer value to them based on the scale used and calculate the dissimilarity as the ratio between the difference of the two records and the total values. It should be clear that we are using a linear scale, this is obviously a very strong assumption but it needs to be taken into account as any chosen scale is based on assumptions. Typically we assuming equal intervals, but it can change especially with suggestion from a domain expert. In general using this transformation for ordinal data is not completely right because it clash with their non numeric nature. [Reference to partial ordered set theory]

We can see in the table that it is easy to define the proximity between two numerical attributes, it is defined as the *absolute difference between the two values*.

3.2.2 Distance measures

When the attribute are numeric, it is possible to define other distance measures as it follow:

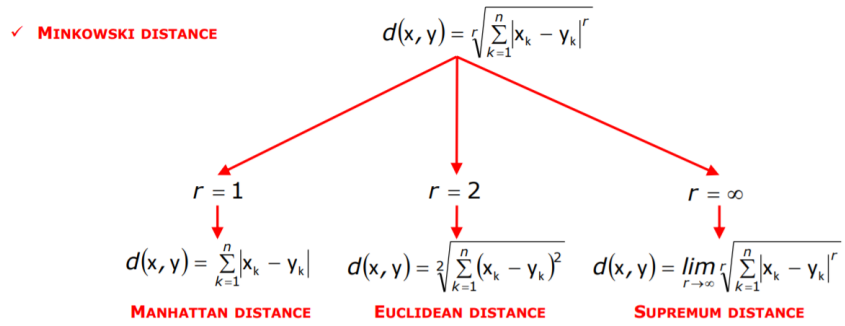


Figure 65: Distance measures starting from the Minkowski distance.

The properties that a function must have to be defined as distance are the following:

- Non-negativity: $d(x,y) \geq 0 \quad \forall x,y \quad d(x,y) = 0 \quad if \quad x = y$
- Symmetry: $d(x,y) = d(y,x) \quad \forall x,y$
- Triangle inequality: $d(x,z) \leq d(x,y) + d(y,z) \quad \forall x,y,z$

Similarity does not satisfy the triangle inequality property but symmetry and non negativity are typically satisfied.

We will now give some examples of proximity measures. In the case where records consist of only binary attributes similarity measures are referred to as **similarity coefficients**, typically take values on $[0,1]$. If the similarity coefficient is equal to 1 then the records are perfectly similar while completely different records bring to a similarity coefficient equal to 0.

There are many rationales for why one coefficient is better than another in specific problems.

Definition 3.12 (Simple matching coefficient)**.** The **simple matching coefficient** is defined with the following expression:

$$SMC(x,y) = \frac{\#maching\_attributes}{\#attributes} = \frac{f_{11} + f_{00}}{f_{11} + f_{00} + f_{01} + f_{10}}$$

This measure makes sense for *binary symmetric* attributes, i.e. for attributes that can only take on the values 0 or 1 almost in the same quantity. As we can see it is basically the *accuracy*. It has to be used when all attributes are *symmetric binary*, that means that the values 0 and 1 are equally valuable. This measure counts both presences and absences equally, counts 1 and 0 as equally relevant. If for examples there is a class more relevant (that typically is the one with less appearances) it clear that we can consider 1 and 0 as equally relevant.

In this case it is used another measure which is derived from the previous one, i.e. the assumption is that the observation of an event (identified with 1) has greater weight than the non-observation of that event (represented with 0).

Definition 3.13 (Jaccard coefficient)**.** The **Jaccard coefficient** is defined with the following expression:

$$J(x,y) = \frac{\#maching\_attributes}{\#attributes\_except00} = \frac{f_{11}}{f_{11} + f_{01} + f_{10}}$$

This is obviously a distorted measure, it is a type of measure suitable for *asymmetric* attributes. It allow to to focus on the 1s. It has to be used when all attributes are *asymmetric binary*. This are attribute corresponding to an item in a store, then the value 1 associated with that item (attribute) means that the item was purchased while a 0 means that the item was not purchased. Since the number of products not purchased by any customers far outnumbers the number of products that are purchased, a similarity measure such as $SMC(x,y)$ would say that all transactions are very similar.

Now lets see the extended version of this coefficient:

Definition 3.14 (Extended Jaccard coefficient)**.** The **Extended Jaccard Coefficient** is defined with the following expression:

$$EJ(x,y) = \frac{x \cdot y}{||x||^2 + ||y||^2 - x \cdot y}$$

This is a distorted measure to handle sparse data that is the contrary of the Jaccard Coefficient, it work on data in which there are many 0s and few 1s. It can be used for document data, and it reduces to the Jaccard Coefficient in the case of binary attributes. It is use for example in natural language processing.

3.2.3 Other proximity measures

In the following part there are some other proximity measures. There are many rationales for why one coefficient is better than another in specific problems.

Definition 3.15 (Cosine similarity)**.** The **cosine similarity** is defined with the following expression:

$$cos(x,y) = \frac{x \cdot y}{||x|| \cdot ||y||}$$

It is a similarity measure that ignores 00 matches like the Jaccard Coefficient, but that is also able to handle non-binary attributes. It is extremely useful for comparing sparse records, it is widely used in Information Retrieval where documents represented as vectors of counts must be compared. In this case $cos(x,y)$ takes values on $[0,1]$, it is equal to 1 when x and y are the same (except for magnitude) while it is equal to 0 when x and y share no terms (maximum difference).

Definition 3.16 (Correlation)**.** The **correlation** is defined with the following expression:

$$corr(x,y) = \frac{cov(x,y)}{std(x)std(y)}$$

This is the same measure as the Pearson correlation, but it is not linked with random variables. It has to be used when all attributes are *numeric*. $corr(x,y)$ takes values on $[-1,1]$, it is equal to 0 when x and y are un-correlated (not necessarily independent), it is equal to 1 in absolute value when x and y are perfectly correlated (dependent).

In the following section there are several **issues** related to proximity measures.

⇒ How to handle the case in which attributes have different scales and/or are correlated?

To solve the first problem it is advisable to proceed as follow: normalize the values, if this were not done the Euclidean distances between the two values would be totally distorted towards of the greater value.

When attributes are correlated and have different ranges of values (different variances), and the distribution is approximately Gaussian the **Mahalanobis distance** is used:

$$Mahal(x, y) = (x - y)\Sigma^{-1}(x - y)^T$$

It has to be used when all attribute are *numeric*. $\sum^{-1}$ is the inverse of the *variance-covariance matrix* of x and y. $Mahal(x, y)$ takes values on $[0, +\infty)$, it is equal to 0 when x and y are the same while the more different x and y are the greater it is.

⇒ How to calculate proximity between records x and y that are composed of different types of attributes ?

In order to solve this problem we have to assume that we have n attributes, for each attribute k, compute the similarity, $S_k(x, y)$ in such a way that it takes value in the interval $[0, 1]$ (all attributes contribute equally to the overall similarity). After that we have to define an indicator variable variable δ_k define an indicator variable δ_k for the attribute k as follows:

δ_k is:

- 0 if the kth attribute is asymmetric and both records have value 0, or at least one of the records has a missing value for the kth attribute
- 1 otherwise

Then we have to compute the overall similarity between the two records x and y using the formula:

$$similarity(x, y) = \frac{\sum_{k=1}^{n} \delta_k s_k(x, y)}{\sum_{k=1}^{n} \delta_k}$$

⇒ How to handle proximity calculation when attributes have different weights, i.e. when not all attributes contribute equally to the proximity of records?

In order to solve this point we can proceed like before, but giving weights (weights w_k sum to 1), so the formula are the following ones:

$$similarity(x, y) = \frac{\sum_{k=1}^{n} w_k \delta_k s_k(x, y)}{\sum_{k=1}^{n} \delta_k}$$

Also the **Minkowski distance** can be modified as follows:

$$d(x, y) = \sqrt[r]{\sum_{k=1}^{n} w_k |x_k - y_k|^r}$$

It is very complex to handle this type of specificity, but the following are a few general observations that may be helpful:

- Dense and continuous data: metric distances, like the euclidean one, are good representations.
- Sparse data, binary asymmetric data: similarity measures which ignore 00 match, like cosine, Jaccard or Extended Jaccard.

Sometimes one or more similarity measures are already in use in the considered application domain and thus, others will have answered the question of which proximity measure/s should be used.

3.3 Clustering Algorithms

We will discuss in the following section some clustering algorithms:

3.3.1 Prototype Based

In *Prototype-Based clustering*, a cluster is a subset of objects (records) such that any object is closer to the **prototype** that defines the cluster to which it belongs to than to the prototype of any other cluster.

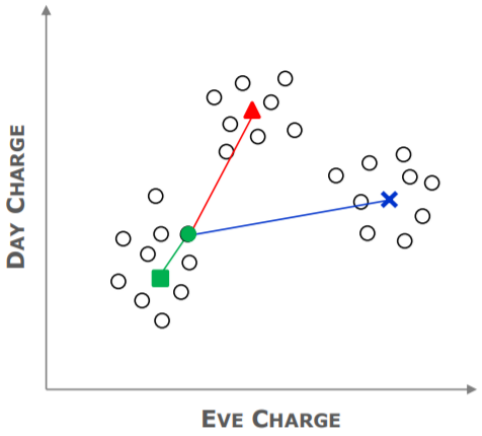


Figure 66: Example of a prototype-based clustering.

Different types of prototype-based clustering algorithms exist depending on the following characteristics:

- Objects belong to a single cluster.
- Objects are allowed to belong to more than one cluster.
- A cluster is modelled as a probability distribution.
- Clusters are constrained to have fixed relationships (neighborhood relationships).

K-Means K-means defines a prototype in terms of a *centroid*, which is usually mean of the attribute's values for a group of objects (records), and is typically applied to objects in a continuous n-dimensional space.

Below is a schematic description of algorithm steps:

1. Select K objects as centroid.

2. Form K clusters, assign each record to the closest centroid.
3. Recompute the centroid of each cluster.
4. **Repeat** points 2 and 3 **until** centroids do not change.

As can be easily seen, the number of clusters must be specified before computing the algorithm; different starting values can give very different results. The choice of the number of clusters is therefore a very important parameter.

Usually the K-mean algorithm tend to form convex clusters, so we need to understand empirically if is suitable for the dataset to have clusters with this property.

K-means is not restricted to data in Euclidean space but can be used when different proximity measures are considered:

- **Manhattan**: in this case the centroid is the median and the goal is to *minimize the sum of L_i distance of a record to its cluster centroid.*
- **Squared Euclidean**: the centroid is the mean and the goal is to *minimize the sum of the L_i distance of a record to its cluster centroid.*
- **Cosine**: the centroid is the mean and the goal is to *maximize the sum of the cosine similarity of a record to its cluster centroid.*

Below is a series of **issues** related to the use of the K-means algorithm as clustering algorithm:

- *Choosing initial centroids*: when random initialization of centroids is used, different runs of the K-mean produce different clustering. It has limitations and thus alternative techniques are used for initializing centroids (hierarchical clustering).
- *Time and space complexity*: Only records and centroids are stored thus space requirements are modest. Also time requirements are modest. It is linear in the number of records and is efficient as well as simple provided that K is significantly less than the number of records
- *Empty clusters*:a strategy is required to choose a replacement centroid.
- *Outliers*: centroids are not representative as they would whether outliers were not present. it is useful to discover outliers and remove them (for some applications we behave differently).

Below are a series of **limitations**of K-means:

- Difficulty to detect cluster with non-spherical shapes.

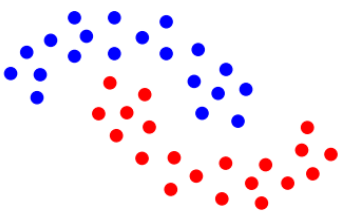


Figure 67: Non-spherical clusters.

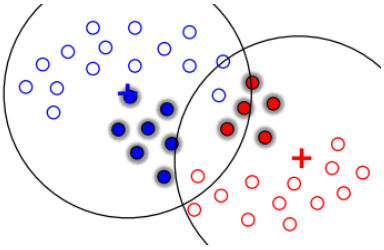


Figure 68: Wrong identification of non-spherical clusters.

- Difficulty to detect clusters with different size: this problem is linked to fixed distance.

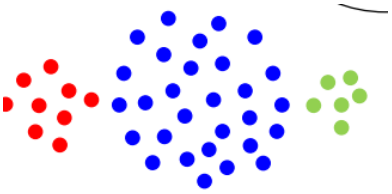


Figure 69: Non-fixed distances clusters.

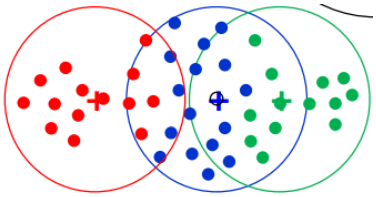


Figure 70: Wrong identification of non-fixed distances clusters.

- Difficulty to detect clusters with different density: this problems happen because cluster could have very different dimensions.

Figure 71: Different density clusters.

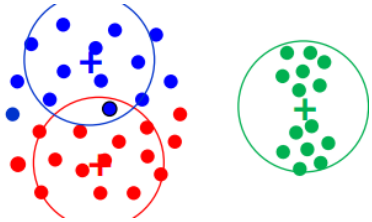


Figure 72: Wrong identification of different density clusters.

This tree problems lead to very different clusters. A way to partially solve this problems is to *accept a clustering that breaks the natural clusters into a number of sub-clusters.*

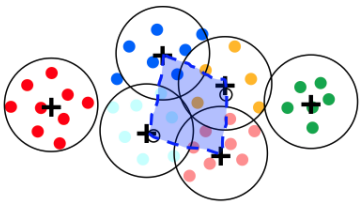


Figure 73: Breaking the cluster in sub-clusters.

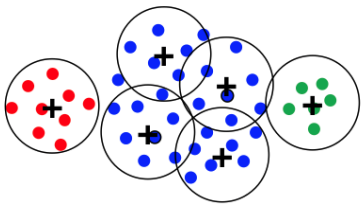


Figure 74: Recomposing the cluster.

Below is a quick list summarizing the **strengths and weaknesses** of the K-means algorithm:

- It is simple and can be used for wide variety of data types.
- It is quite efficient, even thought multiple runs are often performed.
- Some variants as *bisecting K-means* are more efficient and less susceptible to initialization problems.
- It is not suitable for all types of data. Indeed, it can not handle non-globular clusters, clusters with different sizes and of different densities.

- It has troubles clustering data which contains outliers.
- It is restricted to data for which there is a notion of a center (centroid). However, *K-Medoids* algorithm (partitioning around medoids) overcomes this restriction, it is robust with respect to outliers but extremely computationally demanding.

Fuzzy C-means If the objects (records) are distributed in well separated groups, then a crisp classification of them into disjoint clusters seems an ideal approach. However, in most cases, the objects in a data set cannot be partitioned into well separated clusters, and there will be a certain arbitrariness in assigning an object to a particular cluster. In fact, this problem is reflected in the choice of assigning the elements placed between to clusters to the respective cluster.

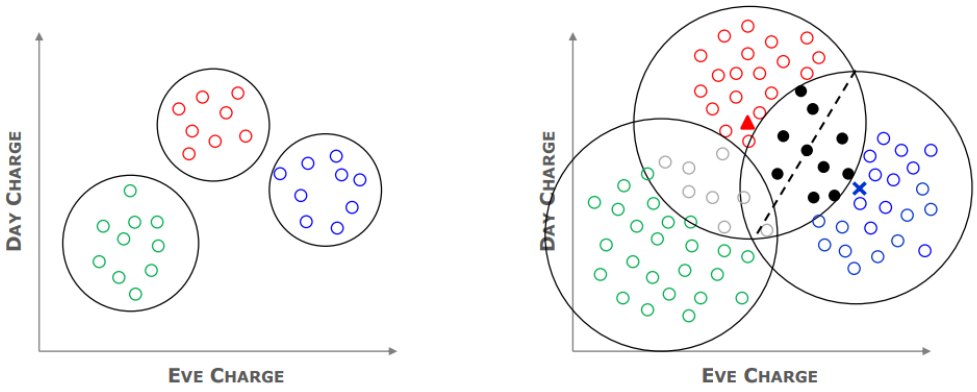


Figure 75: Difference between well separated clusters and non-well separated ones.

We consider the red circles and the blue circles clusters, the black dashed line indicates the place where objects are equally far away from the centroids of the red and blue clusters. Using the concept of minimum distance we assign all objects to the red and blue clusters except the black one which lays on the separating line. This object has the same distance from the centroid of the red and blue clusters and thus assigning it to one of the two clusters is purely arbitrary. Assign a weight to each object and each cluster that indicates the degree to which the object belongs to the cluster. w_{ij} is the weight with which the ith object belongs to the jth cluster.

Assume we have a data set consisting of m objects (records), where each object is associated with a given number of continuous attributes.

Definition 3.17 (Fuzzy pseudo-partition)**.** A **Fuzzy pseudo-partition** is defined by assigning the value of the weight to object i and cluster j:

$$w_{ij} \quad (w_{ij} \geq i = 1, ..., mj = 1, ..., K)$$

with the following set of constraints:

$$\sum_{j=1}^{K} w_{ij} = 1 \quad i = 1, 2, ..., m$$

$$0 < \sum_{i=1}^{K} w_{ij} < m \quad j = 1, 2, ..., K$$

Obviously *clusters of zero size are not allowed*, because of how clusters are defined. The **Fuzzy C-Means** *produces a clustering that provides an indication of the degree to which any object belongs to any cluster*. It has the same strengths and weaknesses as the K-means, although it is more computationally demanding.

Mixture Models Mixture models view the data as a set of observations from a mixture of different probability distributions (they can be anything but usually they are multi-variate Gaussian).

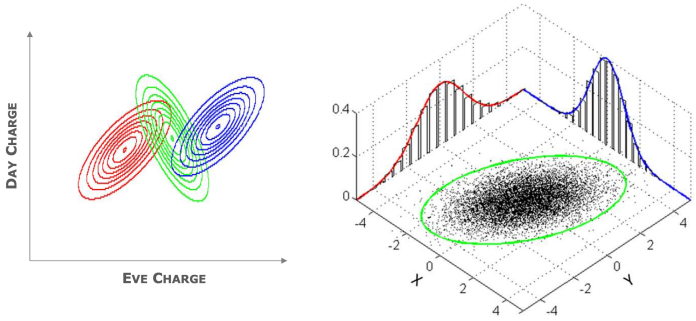


Figure 76: Mixture models' level curves.

Lets imagine that we have a three dimensional component space, the level curves *generative process* is the following:

1. Randomly select one of the 3 components.
2. Extract one sample from the selected component.
3. Repeat 1 and 2 m times to obtain the data set

The following quantities also come:

- $p(\underline{x}|\Theta) = \sum_{j=1}^{K} w_j p(\underline{x}|\theta_j)$: probability of object $\underline{x}$.
- w_j: probability to select the jth component
- $p(\bar{x}|\theta_j)$: probability to extract form $\underline{x}$ from the jth component.
- θ_j: parameters associated with the jth component.

The process consists of starting with the data and reducing it to meaningful mixtures. There is a class of algorithms used to solve this problem which is **Expectation Maximization**. These are algorithms that are different according to their application.

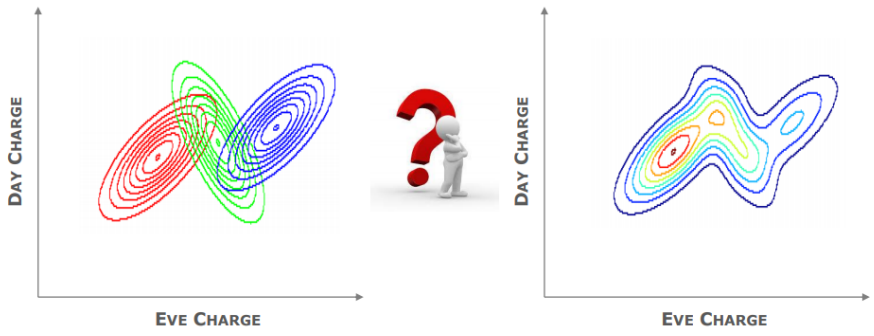


Figure 77: Desirable result.

This type of algorithms have several advantages and disadvantages. The **disadvantages** are:

- Learning with EM is slow.
- Learning with EM is not practical for models with a large number of components.
- Learning with EM does not work well when clusters contain few observations.
- Learning with EM does not work well when objects are co-linear.

However, these algorithms also have a number of **advantages** that are not indifferent to those of the k-means:

- More general than K-Means and Fuzzy C-Means, because they can use distributions of various types, can find clusters of different sizes and elliptical shapes.
- Disciplined approach of eliminating some of the complexity associated with data.

Kohonen Maps or SOMs **Self-organizing maps** (SOMs or Kohonen maps, after the name of the creator) are a non-linear dimensionality reduction tool, designed to generate two-dimensional visualizations of multi-density distributions - dimensional. The n statistical units, represented by vectors in ^{k}, are approximated by m (typically, $m < n$) k-dimensional vectors, called *codebooks*. which are associated with the cells of a two-dimensional rectangular grid and then displayed. Codebooks have a similar role to centroids in k-means (we will see later in which sense affinities between the two algorithms can be traced), i.e. to discretize or quantize the input density. Unlike the k-means, the codebooks are arranged in an "intelligent" way on the plane, so as to bring out regular patterns, which synthesize the multidimensional density. Feed forward structure with a single computational layer organized into a discrete grid, each output node is connected to all the input nodes. Connections are bidirectional, from the input node to the output nodes and from the output nodes to the input nodes. Technically, SOMs are a particular type of neural network, structured through a competitive learning process which creates a *Voronoi diagram* of the input space and maps the Voronoi cells (we will clarify later on all these terms) on the cells of the two-dimensional grid, essentially constructing a two-dimensional histogram, in the logic of non-parametric estimation procedures of a density.

Before delving into the "sense" of SOMs and interpreting their structure, let's proceed to illustrate their development algorithm.

The algorithm for generating a SOM is called **stepwise**, it consists of a sequence of learning cycles, each of which is called an *epoch*, through which the map "learns" from the data and self-organizes itself, generating the codebooks associated with the cells of the grill. To make the algorithm operational, the analyst must define some basic characteristics and some parameters of the map, like its size and topology (see below), and must choose an initial configuration of the codebooks, which allows the learning process to start. The cycles must be repeated until the codebook cells become stable, as the epochs progress.

The data matrix is X_{nk}; the SOM generation algorithm is divided into three main steps, $set - up$, *initialization* and *training*, described below.

- **Set-up**: Before activating the learning process, it is necessary to fix some fundamental characteristics of the map, in particular:
  - *Map size*: The map is rectangular and therefore the number of cells n_x and n_y must be fixed on the two sides. In general, there is no optimal criterion for choosing the number of cells and the shape of the map. We can think that as the elements of the grid increase, the "resolution" of the map increases and that therefore it is convenient to have very large maps. On the other hand, maps that are too large can also be too "empty" and fail to structure themselves in an interesting way; conversely, maps that are too small risk in condensing very different inputs in the same codebooks, thus failing to resolve the input points. The number of map elements must be in some way linked to the number of input points and to their density in the original space (which can of course be different from area to area) and, in general, must grow with as the number of observations increase. This information is not known in advance (otherwise we would not need a SOM) and therefore we need to proceed heuristically, by trial and error. As regards the shape of the map (the ratio n_x/n_y) a pragmatic way to proceed is to check whether there is a principal direction of variability of the data or not. If there is, it will be to structure an "stretched" map, if not the map will be square. In practice, we could extract the first two principal components from the data matrix and build a map so that n_x/n_y is similar to the ratio of the first two eigenvalues. Clearly, this is an intuitive procedure and not necessarily optimal, because the unknown distribution may have characteristics such as to require different map dimensions. Ultimately, map sizing remains a trial-and-error process.
  - *Shape of the cells and structure of their neighbourhood*: Map cells can be square or hexagonal in shape. The choice of the shape is not an "aesthetic" feature, but has a reflection on the structure of the neighborhoods of the cells themselves. In the square case, the cells have 4 close neighbors, in the hexagonal case they have 6 of them. As will be clarified by illustrating the functioning of the algorithm, this choice has potential effects on the construction of the map. Typically hexagonal cells are preferable because the visualization is more clear.

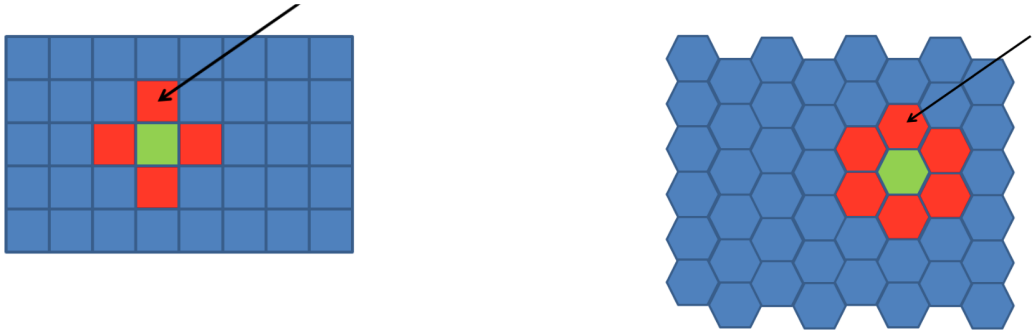


Figure 78: Type of shape of cells.

– *Map topology*: The topology of the SOM can be planar or toroidal. In the first case, the cells on the edge, or on the corners, have a different number of first neighbors, which makes their learning process different (see below), generating "edge effects" that can make the map inhomogeneous near the edges or at the corners. In the second case, the cells on the edges and corners are "stuck" to the corresponding ones on the opposite sides/corners, so as to make the topology of the map equivalent to that of a torus (mathematical shape). In this way, each cell has the same number of elements and there are no anomalies in the map, which however is visually more difficult to interpret, because it is necessary to remember that two opposite sides/angles are, in reality neighbors.

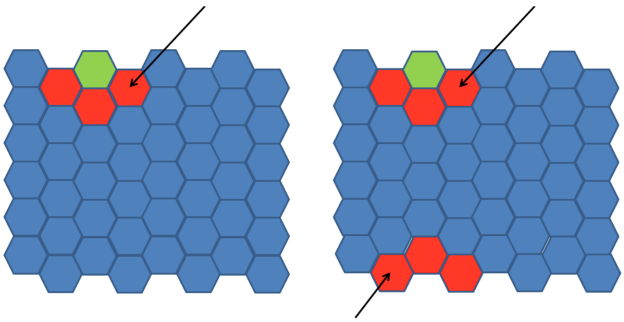


Figure 79: Plane topology on the left and toroidal topology on the right.

- **Initialization**. In order to start the learning process, the codebooks of the map must be initialized, assigning a k-dimensional vector to each cell of the grid. This assignment can be done in different ways, for example by generating the codebooks randomly, or by randomly extracting them from the dataset. The assignment of the initial codebooks may have an effect on the map that will be built by the algorithm, but in principle the learning process should "make to forget" the initial state and the essential characteristics of the map should not depend on the choice of the starting codebooks. For example, it could be that different choices of the initialization codebooks correspond to "rotated" maps, or in which the different areas are arranged in different zones of the map, but the general structure of the final map should not change.

- **Training**. Within each s epoch, the map learns from the data and organizes itself, modifying the codebooks associated with the cells, so as to make them progressively more similar to the input vectors. Learning takes place in different steps as it follows:

  1. *Competition*: The statistical units, i.e. the vectors of the input dataset, are "presented" to the map, one at a time and in random order, and are compared with the codebooks present at the beginning of the epoch. For each presented vector v and for each codebook c_i the Euclidean distance $d(v - c_i) = \|v - c_i\|^2$ is computed. We search the codebook c_{i^*} which minimizes the distance, in this context is called the *winning codebook* (if there are any were more than one, a random choice is made).

2. *Updating*: The winning codebook is made more similar to the input vector, using the following transformation:

$$c_{i^*}^{new} = c_{i^*} + h(s, i^*|i^*)[v - c_{i^*}]$$

where the coefficient $h(s, i^*)$ depends on the epoch (indexed as s) and represents the impact that the vector v has on the modification of the winning codebook. As the epochs progress, it decreases, so that the map stabilizes.

3. *Collaboration*: In addition to the winning codebook, also the other cells are also modified in a similar but in a less intense way as the codebooks belong to cells physically more distant on the map from that of the winning codebook. The modification of the generic codebook also happens as follows:

$$c_i^{new} = c_i + h(s, i|i^*)[v - c_i]$$

When all the vectors of the dataset have been presented to the map, i.e. when an epoch is concluded, the learning process restarts. The statistical units are again presented to the map, as in the previous epoch, the codebooks are modified as described continuing to incorporate the information contained in the input data. The succession of epochs continues until the map is stable, that is until the codebooks do not change anymore (or change "slightly"). The stability of the map can be measured simply by calculating the Euclidean distances of each codebook from that of the previous epoch and making an arithmetic mean of these values. Analyzing how this synthetic measure changes as s (number of epochs) increases, it is possible to control the degree of convergence of the map. Note that, in general, this average distance is not monotonic decreasing and, especially in the first learning cycles, it can have fluctuating values, which however then take on a substantially decreasing trend, from a certain point on wards.

- **The neighborhood function** $h(s, i|i^*)$. The function $h(s, i|i^*)$ is the heart of the learning process, it determines how the information contained in the input vectors is incorporated and diffused in the map through the mechanism of adaptation of the codebooks. The explicit form of the function is the following:

$$h(s, i|i^*) = \alpha(s)e^{-\frac{\|cell_i - cell_{i^*}\|^2}{2\sigma(s)}}$$

where $cell_{i^*}$ is the vector which identifies the center of the cell containing the winning codebook, $cell_i$ is the analogous vector which identifies that of the cell containing codebook i, $\alpha(\cdot)$ is a decreasing function (for example, hyperbolically, exponentially, or piecewise linear) and $\sigma(\cdot)$ is also a monotonic function. The functions $\alpha(s)$ and $\sigma(s)$ determine the way in which the intensity of the change and its diffusion radius decrease as the epochs proceed. It is necessary for these functions to decrease over time, because if this does not happen each epoch could "delete" what has been learned from the previous epoch, preventing the map from stabilizing and converging. In general, the precise form of the functions $\alpha(\cdot)$ and $\sigma(\cdot)$ is not crucial (in practice, what matters is that the function $\sigma(\cdot)$ is large enough at the beginning and of the process, for example equal to half the largest size of the grid, and gradually reduces in about 1000 steps). Indeed, it is possible

to choose a function $h(\cdot, \cdot)$ which is 1 up to a certain distance from the cell with the winning codebook and 0 beyond, making sure that this distance decreases, down to 0, as the epochs follow one another. Note that when the modification radius of the codebooks is reduced to 0, the algorithm for generating a SOM is substantially reduced to that of the k-means.

The Self Organizing Maps can be also used for dimensionality reduction they are very efficient in reducing a continuous highly dimensional space in a discrete two-dimensional space. The visual interpretation of a SOM in this case usually takes place through the so-called *component planes*. In practice, each cell is colored (with an appropriate chromatic scale) based on the values of a component (fixed for all cells) of its own codebook. In this way, a color pattern is produced which shows the distribution of the level of the variable corresponding to the component considered, as happens with the so-called thermographic maps. It could also be useful to assess the map quality by the measures of *quantization error* and *topographic error*, also it is possible to produce a *U-map* to see if the SOM has some fractures.

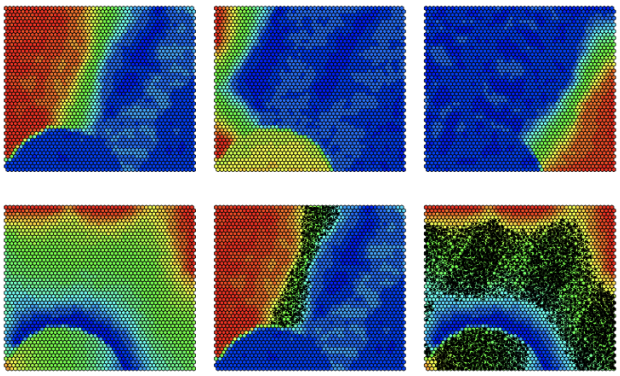


Figure 80: SOM's colored component planes.

Through the Voronoi diagram we can understand that the map is therefore a sort of "elastic surface" that adapts, without tearing, to the input density. The cells of the two-dimensional grid can then be interpreted as the basis of a histogram, the map itself, once the number of input points associated with each cell has been calculated, as a two-dimensional histogram and the fitting process as a process of "identification " of the bases themselves, under the constraint of spatial continuity of the map.

SOMs produce a valid clustering. The main strength is the *interpretability*, because clusters that are neighbors are more related to one another than clusters that are not. This facilitates the interpretation and visualization of the clustering results.

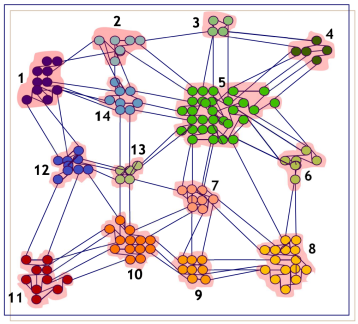


Figure 81: SOM's solution.

This algorithm has also the following limitations:

- The user must choose the setting of parameters, neighborhood function, grid type and the number of centroids.
- A SOM cluster does not often correspond to a single natural cluster (this happens also with other prototype-based clustering algorithms).
- Lacks of a specific objective function. This can make it difficult to compare different SOM clustering results.
- *Convergence is not guaranteed.*

Usually k-means is applied on top of SOM in order to merge clusters, typically SOM procedure is not the end of a clustering process because it needs a post-process in order to optimize the clusters.

3.3.2 Hierarchical Clustering

An hierarchical clustering can be:

- *Agglomerative*: starts with all objects (records) as individual clusters and, at each step, merges the closest pair of clusters. This requires defining a notion of cluster proximity.
- *Divisive*: starts with one, all-inclusive cluster and, at each step, splits a cluster until only singleton clusters of individual objects remain. In this case, we need to decide which cluster to split at each step and how to do the splitting.

Agglomerative Hierarchical Clustering This types of techniques are by gar the most common and we focus in this lecture exclusively on these methods. Hierarchical clustering is often displayed graphically using a tree-like diagram called *dendrogram*, which displays both the cluster-subcluster relationships and the order in which the clusters were merged (agglomerative) or split (divisive).

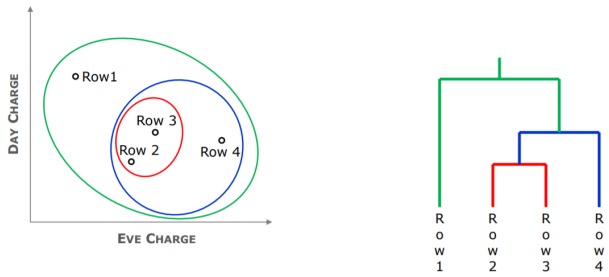


Figure 82: Dendrogram of an hierarchical cluster.

Let's now observe the algorithm steps:

1. Compute the proximity matrix, if necessary.
2. Merge the closest two clusters.
3. Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.

The steps 2 and 3 has to be repeated until only one cluster remains.

Now the problem of computing the proximity between clusters rise. There are several methods in order to do it:

- *Min or Single Linkage*: we have to use the smallest distance between all the possible pairs of elements present in the two clusters, it is the most optimistic solution.
- *Max or Complete Linkage*: we have to use the greatest distance between all possible pairs of elements present in the two clusters, is a more robust or pessimistic solution.
- *Group Average or Average Linkage*: we have to use the average distance between all possible pairs of elements present in the two clusters, is a fairly used solution as it averages the distances.
- *Ward's Method*: it is an alternative technique that assumes that a cluster is represented by its centroid, but it measures the proximity between two clusters in terms of the increase in the Sum of Squared Errors that results from merging two clusters.

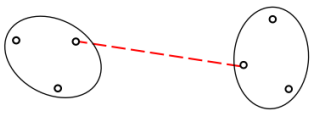


Figure 83: *Single Linkage*

Figure 84: *Complete Linkage*

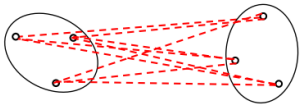


Figure 85: *Average Linkage*

The key **issues** of agglomerative hierarchical clustering are the following:

- *Lack of a global objective function*: agglomerative hierarchical clustering techniques use various criteria to decide locally, at each step which clusters should be merged. This avoids the difficulty to solve a hard combinatorial optimization problem.

- *Ability to handle different cluster sizes*: allows to treat the relative sizes of the pairs of clusters that are merged (this applies to Ward and group average). The possible approaches are weighted (clusters are treated equally) and unweighted (takes the number of objects in each cluster into account).

- *Merging decisions are final*:once a decision is made to merge two clusters, it can not be undone at a later time. This prevents a local optimization criterion to become a global optimization criterion.

Weaknesses of agglomerative hierarchical clustering are the following:

- *Computationally and space intensive*: agglomerative hierarchical clustering algorithms are expensive in terms of their computational and storage requirements.

- **Merging decisions are final**: can cause trouble for noisy, high-dimensional data such as document data

3.3.3 Density-Based Clustering

Now we will discuss the algorithm class based on density. In particular density-based clustering techniques aim to find *dense region of object that are surrounded by low-density regions*. Lets now see a specific algorithm named *DBSCAN*.

DBSCAN It is a simple and effective density-based clustering algorithm that illustrates a number of important concepts that are typical of the density-based approach.

Several methods exist do define density, we describe the *center-based approach* on which DBSCAN is based.

Definition 3.18 (Density)**.** We define **density** as the number of objects within a fixed specific radius (*Eps*) of an object.

Density of any point will depend on the specified radius.

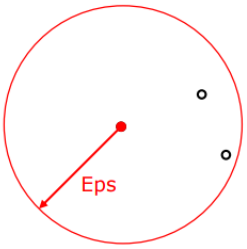


Figure 86: Visual representation of density.

Density-based clustering techniques allow to classify a point (record) in three different way:

- *Core point*: it is the interior of a density-based cluster. A point is a core point if the number of points within a given neighborhood around the point as determined by the distance function and a user-specified distance parameter, (*Eps*), exceeds a certain threshold ($MinPts$) which is also a user-specified parameter.

- *Border point* : it is not a core point, but falls within the neighborhood of a core point. A border point can fall in the neighborhood of several core points.
- *Noise point*: it is any point that is neither a core nor a border point.

As for the other techniques we will see the schematic algorithm steps:

1. Label all pints as core, border, or noise points.
2. Eliminare i noise point.
3. Put an edge between all core points that are within ϵ of each other.
4. Make each group of connected core points into a separate cluster.
5. Assign each border point to one of the cluster of its associated core points.

As said before the choice of parameter *MinPts* e *Eps* is crucial because it strongly impacts on the algorithm computation. We compare DBSCAN to K-Means assuming that there are no ties in distances for either DBSCAN and K-Means, we also assume that DBSCAN always assigns a border point which is associated with more core points to the closest core point.

- DBSCAN and K-Means assign objects to a single cluster, but K-Means assigns all objects while DBSCAN can discard noise objects.
- DBSCAN can handle clusters of different sizes and shapes and it is not strongly affected by noise or outliers. K-means has difficulties with non-globular clusters and clusters of different sizes. Both algorithms perform poorly when clusters have widely differing densities.
- K-Means can only be used for data that has a well defined centroid, such as mean or median. DBSCAN requires that its definition of density, which is based on the traditional Euclidean notion of density be meaningful for the data.
- K-Means can be applied to sparse, high-dimensional data, such as document data. DBSCAN typically performs poorly for such data because the traditional Euclidean definition of density does not work well for high-dimensional data.
- DBSCAN makes no assumption about the distribution of the data. The basic K-means is equivalent to a statistical clustering approach (Mixture Model) that assumes all clusters come from spherical Gaussian distributions with different means but the same covariance matrix.
- DBSCAN and K-Means both look for clusters using all attributes, that is, they do not look for clusters that may involve only a sub-set of the attributes.
- K-Means can find clusters that are not well separated, even if they overlap, but DBSCAN merges clusters that overlap.
- K-Means has complexity $O(m)$ while DBSCAN is $O(m^2)$ (except for low dimensional Euclidean data).

- DBSCAN produces the same set of clusters from one run to another while K-Means, which is typically used with random initialization of centroids, does not.
- DBSCAN automatically determines the number of clusters, for K-Means the number of clusters needs to be specified as a parameter.

Many additional Density-based clustering techniques exist that address the issues of efficiency, finding clusters in sub-spaces, and more accurately modelling density. They can be grouped in:

- *Grid-based clustering*: they break the data space into grid cells and then form clusters from cells that are sufficiently dense. They can be effective and efficient at least for low-dimensional data.
- *Sub-space clustering*: they look for clusters (dense regions) in subsets of all dimensions. For a data space with n attributes, potentially $2^n - 1$ sub-spaces need to be searched, and thus an efficient technique is needed to do this.
- *Kernel density function*: they use kernel density functions to model density as the sum of the influences of individual data objects.

3.3.4 Graph-based Clustering Algorithm

Hierarchical clustering algorithms take a *graph-based* view of data, in which data objects are represented as *nodes* and the proximity between two data objects is represented by the weight of the *edge* between the corresponding nodes. Additional graph-based clustering algorithms have been developed which exploit a number of properties and characteristics of graphs. The key approaches are the following:

- *Sparsify the proximity graph* to keep only the connections of an object (record) with its nearest neighbors. It is useful for handling noise and outliers, and exploits highly efficient graph partitioning algorithms which have been developed for sparse graphs.
- *Define a similarity measure* between two objects based on the number of nearest neighbors that they share. It helps to overcome problems with high dimensionality and clusters of varying density.
- *Define core objects and build clusters around them.* This requires to introduce a notion of density-based on a proximity graph or a sparsified proximity graph.
- *Use the information in the proximity graph* to provide a more sophisticated evaluation of whether two clusters should be merged. Two clusters are merged only if the resulting cluster will have characteristics similar to the original two clusters.

	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6	Row 7	Row 8	Row 9	Row 10
Row 1		0.3	0.8	0.2	0.1	0.2	0.9	0.2	0.1	0.1
Row 2			0.2	0.4	0.1	0.9	0.4	0.4	0.6	0.2
Row 3				0.1	0.2	0.1	0.8	0.1	0.1	0.1
Row 4					0.7	0.5	0.2	0.9	0.5	0.9
Row 5						0.1	0.1	0.8	0.3	0.9
Row 6							0.2	0.2	0.3	0.1
Row 7								0.1	0.1	0.1
Row 8									0.3	0.8
Row 9										0.4
Row 10										

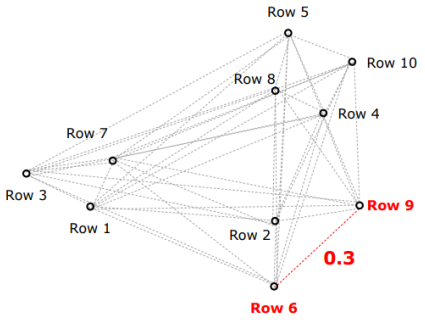


Figure 87: Correlation between proximity matrix and proximity graph.

As notable *every* object has some level of similarity to every other object (record), for most of data sets, objects are highly similar to a small number of objects and weakly similar to most other objects. An useful to set many of the low similarity values to 0 before beginning the actual clustering process. This helps *breaking all links below a threshold or keep only links to the k-nearest neighbors of each point.*

Definition 3.19 (K-Nearest Neighbour)**.** Lets define **K-Nearest Neighbour** as the number of arch values which has as neighbourhood for every matrix row.

Sparsification of the matrix has many advantages. First of all *the reduction of the data size*, eliminating most of the entries of the proximity matrix (setting them to 0) imply to not consider those arch in the algorithm. Second of all **clustering algorithm may work better**, the impact of noise and outliers is reduced due to the nearest neighbor principle, i.e., the nearest neighbors of an object tend to belong to the same class (cluster) as the object itself. Finally *graph partitioning algorithms can be used*, a considerable amount of work on heuristic algorithms for fining min-cut partitionings of sparse graphs.

Sparsification of the proximity graph should be regarded as an initial step before the use of actual clustering algorithms. In principle, *a perfect sparsification could leave the proximity matrix split into connected components corresponding to the desired clusters, but in practice, this rarely happens.* The idea is to cyclically repeat the sparsification until well-defined partitioned graphs are obtained. Many graph-based algorithms have been described in the specialized literature.

Minimum spanning tree The minimum spanning tree (MST) algorithm starts with the minimum spanning tree of the proximity graph and can be viewed as an application of sparsification for finding clusters.

Definition 3.20 (minimum spanning tree)**.** We define **minimum spanning tree** of a graph a subgraph such that:

- has no cycles (it is a tree)
- contains all the nodes of the graph
- has the minimum total edge weight of all possible spanning trees

When talking of minimum spanning tree we assume to work only with dissimilarities or distances. (this is not a limitation)

	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6	Row 7	Row 8	Row 9	Row 10
Row 1		0.7	0.2	0.8	0.9	0.8	0.1	0.8	0.9	0.9
Row 2			0.8	0.6	0.9	0.1	0.6	0.6	0.4	0.8
Row 3				0.9	0.9	0.9	0.2	0.9	0.9	0.9
Row 4					0.3	0.5	0.8	0.1	0.5	0.1
Row 5						0.9	0.9	1.0	0.7	0.1
Row 6							0.8	0.8	0.7	0.9
Row 7								0.9	0.9	0.9
Row 8									0.7	0.2
Row 9										0.6
Row 10										

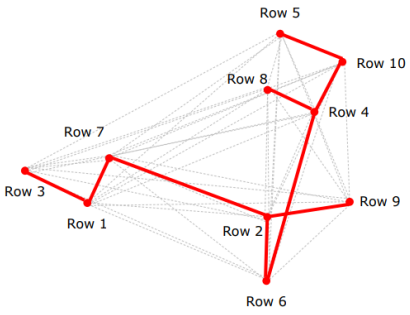


Figure 88: A minimum spanning tree with is matrix.

The **MST divisive hierarchical clustering algorithm** is the following:

1. Compute the MST for the dissimilarity graph.
2. Create a new cluster by breaking the link corresponding to the largest dissimilarity.

Repeat the second step *until singleton clusters remain.*

Opossum Opossim is an algorithm specifically designed for clustering sparse, high dimensional data, such as document or market basket data. Like MST, it performs clustering based on the sparsification of a proximity graph. However, it uses the METIS algorithm, which was designed for partitioning sparse graphs. The **OPOSSUM clustering algorithm** is the following:

1. Compute a sparsified similarity graph.
2. Partition the similarity graph into k distinct components (clusters) using METIS.

Similarity measures must be appropriate for sparse and high dimensional data, e.g., extended Jaccard measure or cosine measure. It is a fast and simple algorithm, which tends to partition the data into roughly equal-sized clusters, which depending on the goal of the clustering can be an advantage or a disadvantage.

CHAMELEON CHAMELEON is an agglomerative clustering algorithm that combines an *initial partitioning of the data*, using an efficient graph partitioning algorithm, with a novel hierarchical clustering scheme that uses the notions of *closeness and interconnectivity*, together with the local modeling of clusters. The main idea is that *two clusters should be merged only if the resulting cluster is similar to the two original clusters.*

CHAMELEON Algorithm is the following:

1. Build a k-nearest neighbor graph.
2. Partition the graph using a multilevel graph partitioning algorithm.
3. Merge the clusters that best preserve the cluster self similarity with respect to relative interconnectivity and relative closeness

We have to repeat the third step *Until no more clusters can be merged.*

It can effectively cluster spatial data, even though noise and outliers are present and the clusters are of different shapes, sizes, and density. It assumes that the groups of objects produced by the sparsification and graph partitioning process are sub-clusters, i.e., that the most of the points in a partition belong to the same true cluster. If this is not the case, then agglomerative hierarchical clustering will only compound the errors since it can never separate objects that have been wrongly put together. CHAMELEON has *problems when the partitioning process does not produce sub-clusters*, as is often the case for high-dimensional data.

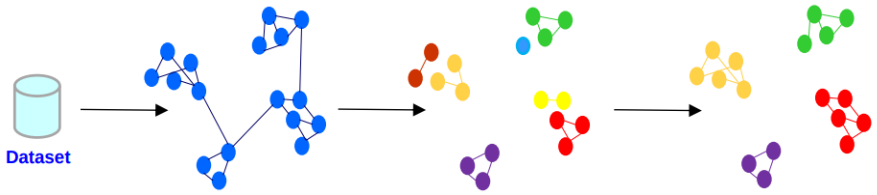


Figure 89: Chameleon algorithm schema

3.4 Clustering Evaluation(*)

When solving a clustering analysis the efficiency of a clustering algorithm you know that the following issues must be addressed:

- Similarity.
- Exclusive or overlapping or fuzzy clustering.
- Complete or partial clustering.

You make the decision to try different clustering algorithms with different parameter values, i.e. the number of clusters for K-Means and SOMs, or the type of linking (single, average, complete, ...) for agglomerative hierarchical clustering, or Eps and MinPts for DBSCAN. You know that in classification (supervised classification), many measures and methods are available to evaluate and compare classification models. Such evaluation measures (accuracy, recall, precision) and methods (cross validation, ...) are well-accepted. However, because of its very nature, CLUSTER EVALUATION is not a well-developed or commonly used part of cluster analysis. Nevertheless, cluster evaluation or cluster validation is of paramount importance Unfortunately, the concept of clustering is much more complex since the exact result to be obtained is not known. A clustering algorithm is designed to find clusters and thus it will find clusters in a data set, even if that data set has no natural cluster structure.

The most important **issues** for cluster validation are:

- Determine the clustering tendency of a set of data, i.e., distinguishing whether non-random structure actually exists in the data.
- Determine the *correct number of clusters* (whatever it means).

To address the issues of cluster validation we use evaluation measures or indices:

- *External or supervised*: measures the extent to which the clustering structure discovered by a clustering algorithm matches some external structure (assuming you have a vague idea of how the data should be organized).
- *Internal or unsupervised*:measures the goodness of a clustering structure without respect to external information. They can be:
  - *Cohesion measures* (compactness, tightness), which determine how closely related the objects in a cluster are.
  - *Separation measures* (isolation), which determine how distinct or well-separated a cluster is from other clusters.
- *Relative*: compares different clusterings or clusters. It can be supervised or unsupervised. Such measures are not a separate type of cluster evaluation measure, but are instead a specific use of such measures.

3.4.1 External or Supervised(*)

Let $P = \{P_1, ..., P_R\}$ be a partitioning, of a data set considering m objects(records), in to R categories.

Lets $C = \{C_1, ..., C_K\}$ be the partition obrained with a clustering algorithm in to K clusters. Supervised or external indices compare P to C:

Case 1. x and y belong to the same cluster of C and to the same category of P

Case 2. x and y belong to the same cluster of C and to the different category of P

Case 3. x and y belong to the different cluster of C and to the same category of P

Case 4. x and y belong to the different cluster of C and to the different categories of P

Possible couples are:

$$M = \frac{m(m-1)}{2} = a + b + c + d$$

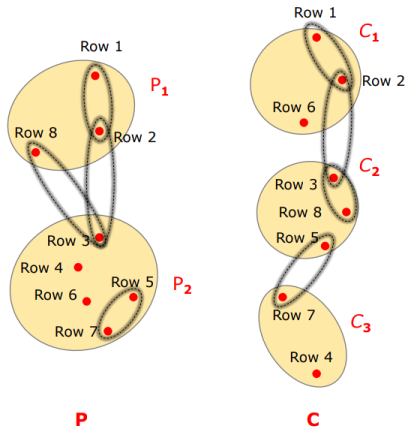


Figure 90: C and P partitions and couples formation.

Lets now define similarity measures:

Definition 3.21 (Rand)**.** We define **Rand** as the following index:

$$R = \frac{a+d}{M} \quad R \in [0,1]$$

Definition 3.22 (jaccard)**.** We define **Jaccard** as the following index:

$$J = \frac{a}{a+b+c} \quad J \in [0,1]$$

Definition 3.23 (Fowlkes and Mallows)**.** We define **Fowlkes and Mallows** as the following index:

$$FM = \sqrt{\frac{a}{a+b} \times \frac{a}{a+c}} \quad FM \in [0,1]$$

Definition 3.24 (Γ statistics)**.** We define Γ **statistics** as the following index:

$$\Gamma = \frac{M \times a - (a+b) \times (a+c)}{\sqrt{(a+b) \times (a+c) \times (M-a-b) \times (M-a-c)}} \quad \Gamma \in [-1,1]$$

The larger the values, the moresimilar are C and P.

3.4.2 Internal Measures(*)

Many internal measures or indices of cluster validity for partitional clustering schemes are based on the notions of *cohesion* or *separation*.

Diversi indici di interni di validità del partizionamento dei cluster si basano sulla *coesione* e sulla *separazione*. The idea is to obtain a high cohesion value and a low separation value. In general, we can consider *expressing the overall cluster validity for a set of K clusters as a weighted sum of the validity of individual clusters* C_i:

$$overall\_validity = \sum_{i=1}^{K} w_i \cdot validity(C_i)$$

where the validity function can be cohesion, separation or any combination of them. In cui la validità può essere la coesione, la separazione o una combinazione lineare di queste; i pesi invece variano in base alla misura di validità del clustering. The weights will vary depending on the clustering validity measure. In some cases they are set to 1 or are the cardinality of the corresponding cluster, while in other cases they reflect a more complicated property, such as the square root of the cohesion.

For graph-based clusters:

Definition 3.25 (Cohesion)**.** For graph-based clusters, the **cohesion** of a cluster can be defined as the sum of the weights of the links in the proximity graph that connect points within the cluster.

$$cohesion(C_i) = \sum_{x,y \in C_i} proximity(x,y) = \sum_{xy \in C_i} similarity(x,y)$$

Therefore, cohesion and similarity are maximized when dissimilarity/distance are minimized. When considering the attributes' space, it is useful to recall that similarity is inversely proportional to dissimilarity/distance.

Definition 3.26 (Separation)**.** For graph-based clusters, **separation** between two clusters can be measured by the sum of weights of the links form points in one cluster to points in the other cluster.

$$separation(C_i, C_j) = \sum_{x \in C_i, y \in C_j} proximity(x, y) = \sum_{x \in C_i, y \in C_j} similarity(x, y)$$

Also in this case separation and similarity are minimized when dissimilarity/distance are maximized.

For prototype-based cluster definitions are a little bit different:

Definition 3.27 (cohesion)**.** For prototype-based clusters, the **cohesion** of a cluster can be defined as the sum of the proximities with respect to the prototype (centroid or medoid) of the cluster.

$$cohesion(C_i) = \sum_{x \in C_i} proximity(x, c_i) = \sum_{x \in C_i} similarity(x, c_i)$$

Definition 3.28 (separation)**.** For prototype-based clusters, the **separation** between two clusters can be measured by the proximity of the two clusters prototypes.

$$separation(C_i, C_j) = proximity(c_i, c_j) = similarity(c_i, c_j)$$

The previous definitions of cluster cohesion and separation offer simple and well-defined measures of cluster validity that can be combined into an overall measure of cluster validity by using a weighted sum $overall\_validity = \sum_{i=1}^{K} w_i \cdot validity(C_i)$. However, we need to decide what weights w_i to use. Indeed, the weights used can vary widely, although typically they express a measure of the cluster size. Some examples are:

CLUSTER MEASURE	CLUSTER WEIGHT	TYPE
$cohesion(C_i) = \sum_{x,y \in C_i} proximity(x, y)$	$\frac{1}{m_i}$	Graph-Based cohesion
$cohesion(C_i) = \sum_{x \in C_i} proximity(x, c_i)$	1	Prototype-Based cohesion
$separation(C_i) = proximity(c_i, c)$	m_i	Prototype-Based separation

Figure 91: Weights to use based on cluster algorithms

Potentially any unsupervised measure of cluster validity can be used as an objective function for a clustering algorithm and vice versa.

So far, we focused on cohesion and separation for the overall evaluation of a group of clusters. Many of these measures of cluster validity also can be used to evaluate individual clusters and objects (records). We could *rank individual clusters according to their specific value of cluster validity*, i.e., cluster cohesion or separation. A cluster that has a high value of cohesion can be considered better than a cluster that has a lower value. This information is useful to improve the quality of the cluster analysis process. If there are clusters that are not very cohesive it may be useful to separate them into more strongly cohesive clusters. Similarly if two clusters are relatively cohesive but not too well separated it might be worth trying to merge them.

It is therefore useful to evaluate an object within the cluster according to the extent to which that object contributes to cohesion or separation. Graphically we would obtain that the objects that are more "inside" the cluster should contribute more to the overall cohesion and separation of a cluster, while the more external ones will have a lesser contribution. Precisely for this reason the Silhouette coefficient is defined which combines cohesion and separation for the i-th object within the cluster.

Definition 3.29 (Silhouette Coefficient)**.** The **Silhouette coefficient**is a cluster evaluation measure which exploits the concepts of interior and edge of a cluster to evaluate data points, clusters and the entire set of clusters. It combines cohesion and separation and for the i-th object (record) it is defined as follows:

$$s_i = \frac{b_i - a_i}{max(a_i, b_i)} \in [-1, +1]$$

Where:

- a_i is the average distance of the i-th object to all other objects in its cluster
- b_i is the minimum of the average distances of the i-th object to all the objects in each given cluster different from the cluster to which the i-th object belongs to

Negative Silhouette coefficient means that the average distance to points in its cluster (a_i) is greater than the minimum average distance to points in another cluster (b_i). We want that the coefficient is positive ($a_i < b_i$), and for a_i to be as close to 0 as possible, since the silhouette coefficient assumes its maximum value of 1 when $a_i = 0$.

We can compute the **average silhouette coefficient** of a cluster by simply taking the average of the silhouette coefficients of data points (records) belonging to the considered cluster. An overall measure of goodness of a clustering can be obtained by computing the average silhouette coefficient of all points. The silhouette coefficient is defined for partitional clustering while for hierarchical clustering a different evaluation measure is used; **Cophenetic correlation coefficient**.

It measures the degree of similarity between the PROXIMITY MATRIX P and the Cophenetic matrix Q whose elements record the proximity level where pairs of data points are grouped in the same cluster for the first time. The value of the Cophenetic correlation coefficient lies in the range of $[-1, 1]$, and the index value close to 1 indicates a significant similarity between P and Q and a good fit of hierarchy to the data. However, for average linkage, even large values of the Cophenetic correlation coefficient cannot assure sufficient similarity between the two matrices.

3.4.3 Validity Paradigm(*)

Both external and internal criteria are closely related to statistical methods and hypothesis tests.The **validaty paradigm**, for a clustering structure, is based on the following steps.

Identfy the clustering structure and validation type The validity paradigm, for a clustering structure, is based on the following null hypothesis: **There is no structure on the dataset**.

Determine a validation index Search the most appropriate validation index for the clustering structure.

Define a null hypothesis of no structure Here lets see three commonly used null hypotheses:

- *Random Position* all the locations of the m data points in some specific region of a n-dimensional space are equally likely. It is used for ratio data.
- *Random Graph* all $[m \times m]$ rank order proximity matrices are equally likely. It is used for ordinal proximities between pairs of data objects.
- *Random Label* all permutation of the labels on m data points are equally likely. It is used for all data types.

Establish the baseline distribution under the null hypothesis It can be a Monte Carlo analysis and Bootstrapping.

Calculate the Index All index seen before are now computed.

Test the hypothesis of no structure We hope to reject the null hypothesis that there is no structure in the clustering we have developed.

3.4.4 The Fundamental Problem(*)

Internal and external indices require statistical testing, which could become computationally intensive. Relative criteria eliminate such requirements and concentrate on the comparison of clustering results generated by different clustering algorithms or the same algorithm with different input parameters. *The fundamental problem of cluster validity is to determine the* **True number of clusters K**. Project the data into a 2 or 3 dimensional Euclidean space and to use visualization techniques which can provide some insights on the number of clusters. Far from being sufficient to represent the true data structures, therefore, the visual estimation of K can only be restricted to a very small scope of applications. The main indices are:

- *Calinski and Harabasz*: The value of K corresponding to the maximum is taken to be the optimal number of clusters.
- *Dunn*: The value of K corresponding to the maximum is taken to be the optimal number of clusters (a large value suggests the presence of compact and well-separated clusters).
- *Devies-Bouldin*: The value of K corresponding to the minimum is taken to be the optimal number of clusters.

For probabilistic mixture model-based clustering the main indices are:

- *Akaike Information Criterion (AIC)*: The value of K corresponding to the minimum is taken to be the optimal number of clusters.
- *Minimum Description Length (MDL)*: The value of K corresponding to the minimum is taken to be the optimal number of clusters.

- *Bayesian Information Criterion (BIC)*: The value of K corresponding to the minimum is taken to be the optimal number of clusters.

For a clustering algorithm that requires the input of the number of clusters K from users, a sequence of clustering structures is obtained by running a clustering algorithm several times (r) where K ranges from K_{min} to K_{max}. Now lets see how generate sequence of clustering structures:

- Choose a clustering algorithm and a validity index.
- **FOR** K_{min} **to** K_{max}**:**
  - **FOR** $i = 1$ **to** r**:**
    - Run the clustering algorithm with K and use parameter values different from the previous running.
    - Compute the value q of the validity index and set $q(i) = q$.
  - Choose the best value q^* of the validity index $q_1, q_2, ..., q_r$.
  - Set $Q(K) = q^*$.

The clustering structures are then evaluated based on the computed index, and the "optimal clustering solution" is determined by choosing the one with the best value of the index.

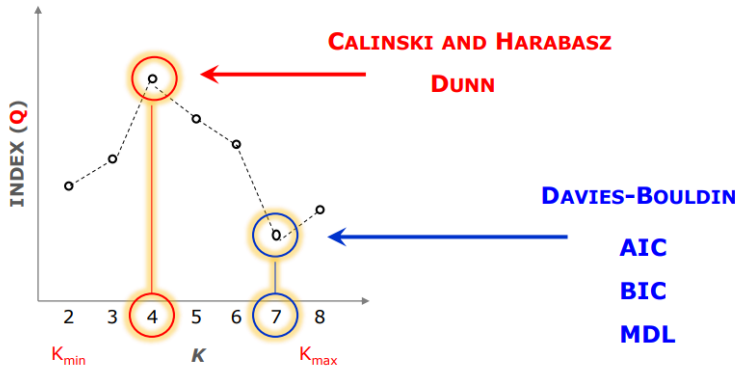


Figure 92: Choice of number of cluster based on the index.

In the case of hierarchical clustering structures, the indices are also known as stopping rules, which tell where the best level is in order to cut the dendrogram.