NEW TRENDS IN E-SCIENCE: MACHINE LEARNING
AND KNOWLEDGE DISCOVERY IN DATABASES

Massimo Brescia
National Institute for Astrophysics, Astronomical Observatory of Capodimonte, Via Moiariello 16, 80131 Naples, Italy

ABSTRACT

Data mining, or Knowledge Discovery in Databases (KDD), while being the main methodology to extract the scientific information contained in Massive Data Sets (MDS), needs to tackle crucial problems since it has to orchestrate complex challenges posed by transparent access to different computing environments, scalability of algorithms, reusability of resources. To achieve a leap forward for the progress of e-science in the data avalanche era, the community needs to implement an infrastructure capable of performing data access, processing and mining in a distributed but integrated context.

The increasing complexity of modern technologies carried out a huge production of data, whose related warehouse management and the need to optimize analysis and mining procedures lead to a change in concept on modern science. Classical data exploration, based on local user own data storage and limited computing infrastructures, is no more efficient in the case of MDS, worldwide spread over inhomogeneous data centres and requiring teraflop processing power.

In this context modern experimental and observational science requires a good understanding of computer science, network infrastructures, Data Mining, etc. i.e. of all those techniques which fall into the domain of the so called e-science (recently assessed also by the Fourth Paradigm of Science). Such understanding is almost completely absent in the older generations of scientists and this reflects in the inadequacy of most academic and research programs. A paradigm shift is needed: statistical pattern recognition, object oriented programming, distributed computing, parallel programming need to become an essential part of scientific background.

A possible practical solution is to provide the research community with easy-to-understand, easy-to-use tools, based on the Web 2.0 technologies and Machine Learning methodology. Tools where almost all the complexity is hidden to the final user, but which are still flexible and able to produce efficient and reliable scientific results. All these
considerations will be described in the detail in the chapter. Moreover, examples of modern applications offering to a wide variety of e-science communities a large spectrum of computational facilities to exploit the wealth of available massive data sets and powerful machine learning and statistical algorithms will be also introduced.

1. INTRODUCTION

Let’s start from a real and fundamental assumption: we live in a contemporary world submerged by a tsunami of data. Many kinds of data, tables, images, graphs, observed, simulated, calculated by statistics or acquired by different types of monitoring systems. The recent explosion of World Wide Web and other high performance resources of Information and Communication Technology (ICT) are rapidly contributing to the proliferation of such enormous information repositories. In all human disciplines, sciences, finance, societies, medicine, military, the archiving and electronic retrieval of data are by now both a common practice and the only efficient way to perform enterprises.

Despite of this situation, there is an important question: how are we able to handle, understand and use them in an efficient and complete way?

It is now widely recognized the chronic imbalance between growth of available data and ability to manage them (Hey et al. 2009).

In most cases the acquired data are not directly interpretable and understandable. Partially because they are obscured by redundant information or sources of noise, and mostly because they need to be cross correlated and in principle we never know which is their hidden degree of correlation, also because in many cases we proceed to explore data without any prior knowledge about what we are looking for.

Moreover, for each social or scientific discipline the data are registered and archived in a inhomogeneous way, making inefficient the available tools useful to explore and correlate data coming from different sources.

Such a scenario imposes urgently the need to identify and apply uniform standards able to represent, archive, navigate and explore data in a homogeneous way, obtaining the required interoperability between the different disciplines.

This basic issue has been reflected within the recent definition (Hey et al. 2009) of the fourth paradigm of modern science, after theory, experiments and simulations. It is the E-science, which is the extraction of knowledge through the exploration of massive data archives, or Knowledge Discovery in Databases (KDD).

The fourth paradigm poses in primis the problem of the understanding of data, still before their representation or registering strategy. In scientific terms it implicitly identifies a methodology, based on the “with open mind” investigation and without any knowledge bias, of any kind of data set, in search of information useful to reveal the knowledge.

Of course this methodology imposes to make use of efficient and versatile computing tools, able to bridge the gap between human limited capacity (both in terms of processing time and 3D dimensionality) and progressive and steady growth in the quantity and complexity of the data. In other words able to replicate at a technological level the high learning, generalization and adaptation capabilities of human brain, by growing exponentially its information processing features.
These two prerogatives, investigation without knowledge bias and fast human intelligence, are not casually the milestones at the base of two rapidly growing disciplines: respectively Data Mining (DM) and Machine Learning (ML).

For those who prefer the formal definitions, DM can be easily defined as the extraction of information, implicit as well a priori unknown, from data.

But the definition of ML is not so easy to be formulated. There are in fact philosophical debates, partially divergent and hard to summarize in a single formal definition. However, in practice we can simplify its expression.

There are in particular two key concepts to be formally cleared: first, what we technically stand for learning? Second, how learning is practically connected to the machine (computer) processing rules?

Usually, in practical terms, what we can easily verify is not if a computer is able to learn, but mostly if it is able to give correct answers to specific questions. But such a level of ability is too weak to state that a computer has learned, especially if we consider that real learning is related to the generalization ability of a problem. In other words, to verify that a machine gives correct answers to direct questions, used to train it, is only the preliminary step of its complete learning. What is more interesting is the machine behavior in unpredicted situations, i.e. those never submitted to the machine during training.

Paraphrasing one of the key concepts of Darwinian theory of evolution of living species, ML is mainly interested to provide intelligence to a computer, i.e. adaptation and generalization to new or unpredicted evolving situations.

We defined above DM as the automatic or semi-automatic process of information discovery within massive data sets. After the previous considerations about ML, we are now able to provide an analogous operative definition also for it: a machine has learned if it is able to modify own behavior in an autonomous way such that if it can obtain the best performance in terms of answer to external stimuli.

This definition shifts the focus on a different aspect of ML, which is not the pure knowledge but the adaptation performance in real and practical situations. In other words we are able to verify the training improvements through the direct comparison between present and past reaction performances. Indeed more in terms of evolution measurement rather than abstract knowledge.

But under theoretical aspects such kind of learning (evolution of behavior) is of course too much simple and weak. We know that also animals, considered less intelligent than humans, can be trained to evolve their reaction to external stimuli. But this does not necessarily mean that they have increased their knowledge, i.e. that they have really learned!

Learning is also thinking (cogito ergo sum, to cite the philosopher Descartes), which implies to have and use own cognitive properties to reach the goal. Not only to answer more or less in a right way to external stimuli. The latter is basically a passive process of action-reaction, not a real result of an active process of thinking and controlled behavior.

So far, the final role of ML must be more than evolution performance. To really decide if any machine was able to learn, it is inevitably needed to verify if the machine may offer a conscious purpose and whether it is able to pursue and achieve its own abilities, acquired during training.

Besides these theoretical considerations, fortunately ML treats physical problems of real world, which are those composed or represented by tangible data and information, as result of
direct or indirect observations/simulations. In such cases we can restrict the scope and interest of the ML and DM techniques, by focusing on their capability to identify and describe ordered structures of information within massive data sets (essentially structures in the data patterns), mixed with noise, together with the ability to predict the behavior of real complex systems. Thus not only identification and prediction capabilities, but also description of the retrieved information, important for classification of unknown events.

2. LAYING THE SCIENCE GROUND OF MACHINE LEARNING

Computing has rapidly established itself as essential and important to many branches of science, to the point where computational science is a commonly used term. Indeed, the application and importance of computing is set to grow dramatically across almost all the sciences. Computing has started to change how science is done, enabling new scientific advances through enabling new kinds of experiments. These experiments are also generating new kinds of data of increasingly exponential complexity and volume. Achieving the goal of being able to use, exploit and share these data most effectively is a huge challenge.

It is necessary to merge the capabilities of a file system to store and transmit bulk data from experiments, with logical organization of files into indexed data collections, allowing efficient query and analytical operations. It is also necessary to incorporate extensive metadata describing each experiment and the produced data. Rather than flat files traditionally used in scientific data processing, the full power of relational databases is needed to allow effective interactions with the data, and an interface which can be exploited by the extensive scientific toolkits available, for purposes such as visualization and plotting.

Different disciplines require support for much more diverse types of tasks than we find in the large, very coherent and stable virtual organizations. Astronomy, for example, has far more emphasis on the collation of federated data sets held at disparate sites (Brescia et al. 2010). There is less massive computation, and large-scale modeling is generally done on departmental High Performance Computing (HPC) facilities, where some communities are formed of very small teams and relatively undeveloped computational infrastructure. In other cases, such as the life sciences, the problems are far more related to heterogeneous, dispersed data rather than computation. The harder problem for the future is heterogeneity, of platforms, data and applications, rather than simply the scale of the deployed resources. The goal should be to allow scientists to explore the data easily, with sufficient processing power for any desired algorithm to process it. Current platforms require the scientists to overcome computing barriers between them and the data (Fabbiano et al. 2010).

Our conviction is that most aspects of computing will see exponential growth in bandwidth, but sub-linear or no improvements at all in latency. Moore’s Law will continue to deliver exponential increases in memory size but the speed with which data can be transferred between memory and CPUs will remain more or less constant and marginal improvements can only be made through advances in caching technology. Certainly Moore’s law will allow the creation of parallel computing capabilities on single chips by packing multiple CPU cores onto it, but the clock speed that determines the speed of computation is constrained to remain limited by a thermal wall (Sutter 2005). We will continue to see exponential growth in disk capacity, but the factors which determine latency of data transfer will grow sub-linearly at
From an application development point of view, this will require a fundamental paradigm shift from the currently sequential or parallel programming approach in scientific applications to a mix of parallel and distributed programming that builds programs that exploit low latency in multi core CPUs. But they are explicitly designed to cope with high latency whenever the task at hand requires more computational resources than can be provided by a single machine. Computing machines can be networked into clouds or grids of clusters and perform tasks that were traditionally restricted to supercomputers at a fraction of the cost. A consequence of building grids over wide area networks and across organizational boundaries is that the currently prevailing synchronous approach to distributed programming will have to be replaced with a fundamentally more reliable asynchronous programming approach. A first step in that direction is Service-Oriented Architectures (SOA) that have emerged and support reuse of both functionality and data in cross-organizational distributed computing settings. The paradigm of service-oriented architectures (SOA) and the web-service infrastructures facilitate this roadmap (Shadbolt et al. 2006).

Traditionally, scientists have been good at sharing and reusing each other’s application and infrastructure code. In order to take advantage of distributed computing resources in a grid, scientists will increasingly also have to reuse code, interface definitions, data schemas and the distributed computing middleware required to interact in a cluster or grid. The fundamental primitive that SOA infrastructures provide is the ability to locate and invoke a service across machine and organizational boundaries, both in a synchronous and an asynchronous manner. The implementation of a service can be achieved by wrapping legacy scientific application code and resource schedulers, which allows for a viable migration path (Taylor et al. 2007). Computational scientists will be able to flexibly orchestrate these services into computational workflows. The standards available for service design and their implementation support the rapid definition and execution of scientific workflows. With the advent of abstract machines, it is now possible to mix compilation and interpretation as well as integrate code written in different languages seamlessly into an application or service. These platforms provide a solid basis for experimenting with and implementing domain-specific programming languages and we expect specialist languages for computational science to emerge that offer asynchronous and parallel programming models while retaining the ability to interface with legacy Fortran, C, C++ and Java code.

3. THE MACHINE LEARNING IN THEORY

The sheer size of the foreseen data streams renders impossible to discriminate objects using traditional software tools, and requires the development of specific computing infrastructures as well as the implementation of specific and robust (largely based on the ML paradigm) methods for data analysis and understanding. ML methods, in order to provide efficient and reliable answers, need also to exploit all the meta-information made available to any scientific/social community through the data repositories federated under specific data centres or virtual organizations (Genova et al. 2002).
The problem of inventing and deploying these new tools under powerful computational infrastructure has therefore become a worldwide challenge. On the algorithmic side, we wish to emphasize that commonly used decision algorithms depend on a fixed number of predefined input features for decision making. This is not the best option in Time Domain Analysis where some of the inputs may not be present within the time slot available to make the decision.

There is therefore a demand for a completely new class of tools that can dynamically select the input features and can consistently give reliable predictions. Most decision algorithms compute the inverse Bayesian probability to deal with missing attributes (mostly called features) in the input data.

Although such algorithms can handle missing features to some extent, there is always a possibility for asymptotic decline in accuracy. It may be argued that the possible state of the missing feature can be adequately constrained and compensated for by the available remaining inputs.

However this is not true, because a feature that can be well constrained by the remaining inputs is usually redundant and it is not required as an input at the first place. If this is not the case, its absence can't be appropriately compensated and that will result in a loss of information.

The idea therefore is to facilitate dynamic learning in which, for instance, the system learns from all available information (and not on a fixed set of samples) and identify strategies that can optimally handle situations in which most of the inputs are missing.

As already underlined, one of next main breakthroughs in many human fields is that we have reached the physical limit of observations. So far, like all scientific disciplines focusing their discoveries on collected data exploration, there is a strong need to employ e-science methodology and tools in order to gain new insights on the knowledge. But this mainly depend on the capability to recognize patterns or trends in the parameter space (i.e. physical laws), possibly by overcoming the human limit of 3D brain vision, and to use known patterns as Base of Knowledge (BoK) to infer knowledge on self-adaptive models in order to make them able to generalize feature correlations and to gain new discoveries (for example outliers identification) through the unbiased exploration of new collected data. These requirements are perfectly matching the paradigm of ML techniques based on the Artificial Intelligence postulate (Bishop 2006). Hence, in principle at all steps of an exploration workflow ML rules can be applied. Let us better know this methodology.

It exists a basic dichotomy in ML, by distinguish between supervised and unsupervised methodology, as described in the following.

The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking." that syllogism is, irrefutable reasoning processes. His syllogisms provided patterns for argument structures that always yielded correct conclusions when given correct premises. For example, "Socrates is a man; all men are mortal; therefore, Socrates is mortal." These laws of thought were logic supposed to govern the operation of the mind; their study initiated the field called logic. Logicians in the 19th century developed a precise notation for statements about all kinds of things in the world and about the relations among them. Contrast this with ordinary arithmetic notation, which provides mainly for equality and inequality statements about numbers.
By 1965, programs existed that could, in principle, process any solvable problem described in logical notation (Moore 1965). The so-called logicist tradition within artificial intelligence hopes to build on such programs to create intelligent systems and the ML theory represents their demonstration discipline. A reinforcement in this direction came out by integrating ML paradigm with statistical principles following the Darwin’s Nature evolution laws, (Duda et al. 2001).

3.1 The Learning Paradigms

In supervised ML we have a set of data points or observations for which we know the desired output, expressed in terms of categorical classes, numerical or logical variables or as generic observed description of any real problem. The desired output is in fact providing some level of supervision in that it is used by the learning model to adjust parameters or make decisions allowing it to predict correct output for new data. Finally, when the algorithm is able to correctly predict observations we define it a classifier. Some classifiers are also capable of providing results in a more probabilistic sense, i.e. a probability of a data point belonging to class. We usually refer to such model behavior as regression.

A typical workflow for supervised learning is shown in the diagram below (Figure 1).

![Figure 1. A workflow based on supervised learning models.](image-url)

The process is:

- Pre-processing of data. First we need to build input patterns that are appropriate for feeding into our supervised learning algorithm. This includes scaling and preparation of data;
- Create data sets for training and evaluation. This is done by randomly splitting the universe of data patterns. The training set is made of the data used by the
classifier to learn their internal feature correlations, whereas the evaluation set is used to validate the already trained model in order to get an error rate (or other validation measures) that can help to identify the performance and accuracy of the classifier. Typically you will use more training data than validation data;

- Training of the model. We execute the model on the training data set. The output result consists of a model that (in the successful case) has learned how to predict the outcome when new unknown data are submitted;

- Validation. After we have created the model, it is of course required a test of its performance accuracy, completeness and contamination (or its dual, the purity). It is particularly crucial to do this on data that the model has not seen yet. This is main reason why on previous steps we separated the data set into training patterns and a subset of the data not used for training. We intend to verify and measure the generalization capabilities of the model. It is very easy to learn every single combination of input vectors and their mappings to the output as observed on the training data, and we can achieve a very low error in doing that, but how does the very same rules or mappings perform on new data that may have different input to output mappings? If the classification error of the validation set is higher than the training error, then we have to go back and adjust model parameters. The reason could be that the model has essentially memorized the answers seen in the training data, failing its generalization capabilities. This is a typical behavior in case of overfitting, and there are various techniques for overcoming it;

- Use. If validation was successful the model has correctly learned the underlying real problem. So far we can proceed to use the model to classify/predict new data.

The kinds of problems that are suited for unsupervised algorithms may seem similar, but are very different to supervised learners. Instead of trying to predict a set of known classes, we are trying to identify the patterns inherent in the data that separate like observations in one way or another. In other words, the main difference is that we are not providing a target variable like we did in supervised learning.

This marks a fundamental difference in how both types of algorithms operate. On one hand, we have supervised algorithms which try to minimize the error in classifying observations, while unsupervised learning algorithms don’t have such gain, because there are no outcomes or target labels. Unsupervised algorithms try to create clusters of data that are inherently similar. In some cases we don’t necessarily know what makes them similar, but the algorithms are capable of finding relationships between data points and group them in possible significant ways. Differently from supervised algorithms, which aim at minimizing the classification error, unsupervised algorithms try to create groups or subsets of the data, in which points belonging to a cluster are as similar to each other as possible, by making the difference between the clusters as high as possible (Haykin 1998).

Another main difference is that in an unsupervised problem, the concept of training set does not apply in the same way as with supervised learners. Typically we have a data set that is used to find the relationships in the data that buckets them in different clusters. A common
A workflow based on unsupervised learning models is shown in the diagram below (Figure 2).

1. Pre-processing of data. As with supervised learners, this step includes selection of features to feed into the algorithm, by also scaling them to build a suitable training data set;
2. Execution of model training. We run the unsupervised algorithm on the scaled data set to get groups of like observations;
3. Validation. After clustering the data, we need to verify whether it cleanly separated the data in significant ways. This includes calculating a set of statistics on the resulting outcomes, as well as analysis based on domain knowledge, where you may measure how certain features behave when aggregated by the clusters.

Once we are satisfied of the resulting creation of clusters (or in general over-densities), there is no need to run the model with new data (although you can).

### 3.2 What we are looking for in the Data

In the DM scenario, the ML model choice should always be accompanied by the functionality domain. To be more precise, some ML models can be used in a same functionality domain, because it represents the functional context in which it is performed the exploration of data.

Examples of such domains are:
3.2.1 Dimensional Reduction

Traditional statistical methods break down partly because of the increase in the number of observations, but mostly because of the increase in the number of variables associated with each observation. The dimension of the data is the number of variables that are measured on each observation.

High-dimensional data sets present many mathematical challenges as well as some opportunities, and are bound to give rise to new theoretical developments. One of the problems with high-dimensional data sets is that, in many cases, not all the measured variables are “important” for understanding the underlying phenomena of interest. While certain computationally expensive novel methods can construct predictive models with high accuracy from high-dimensional data, it is still of interest in many applications to reduce the dimension of the original data prior to any data modeling (Samet 2006).

In mathematical terms, the problem we investigate can be stated as follows: given the p-dimensional random variable \( x = (x^1 \ldots x^p)^T \), and a lower dimensional representation of it, \( s = (s^1, \ldots, s^k)^T \) with \( k \leq p \), that captures the content in the original data, according to some criterion. The components of \( s \) are sometimes called the hidden components. Different fields use different names for the \( p \) multivariate vectors: the term “variable” is mostly used in statistics, while “feature” and “attribute” are alternatives commonly used in the DM and ML literature.

Generally speaking, dimensional reduction is the process of reducing the number of random variables under consideration, and can be divided into feature selection and feature extraction.

Feature selection approaches try to find a subset of the original variables (also called features or attributes) (Guyon et al. 2003). Two strategies are filter (e.g. information gain) and wrapper (e.g. search guided by the accuracy) approaches.

Feature extraction transforms the data in the high-dimensional space to a space of fewer dimensions. The data transformation may be linear, as in Principal Component Analysis (PCA), but many non-linear techniques also exist (Guyon et al. 2006).

Being based on the covariance matrix of the variables, PCA is a second-order method. In various fields, it is also known as the Singular Value Decomposition (SVD), the Karhunen-Loève transform, the Hotelling transform, and the Empirical Orthogonal Function (EOF) method. In essence, PCA seeks to reduce the dimension of the data by finding a few orthogonal linear combinations (Principal Components) of the original variables with the largest variance. In other words, PCA performs a linear mapping of the data to a lower
Another technique belonging, like PCA, to the latent variable methods family, is the model known as Principal Probabilistic Surfaces (PPS), in which first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible (Chang et al. 2001).

The PPS are able to find a better data aggregation than the PCA method. A PPS is trained to recognize the best projection functions from the N-dimensional parameter space to a spherical surface in a 3D space.

This surface is covered by a grid of latent variables (points), representing the Gaussian peak in the N-parameter space. It permits to visualize all data with a human compliant 3D diagram, independently from the number of initial parameters. It is hence possible to individuate the presence of sub-structures in the data. An interesting aspect is the estimation of each input data parameter incidence on the latent variables, that can help to understand the relationship between the parameter and the found clusters. The incidence of parameters is calculated by evaluating the probability density of input vector components in respect of each latent variable. During the training phase a reference variety is created.

In the test phase, a datum, never seen by the network, is attributed to the closest spherical variety. Obviously the concept of closest implies a calculation of a distance between a point and a node in the space. Before that the data must be projected on the space. This basically because a spherical variety consists of squared or triangular areas, each of them defined by 3 or 4 variety nodes. After this projection of the datum the approximated distance is calculated.

In the PPS system three main approximation criteria exist:

- Nearest Neighbor: it founds the minimum square distance from all variety nodes;
- Grid projections: it founds the shortest projection distance on the variety grid;
- Nearest Triangulation: it founds the projection closest to the possible triangulations.

The most frequently used is the first one, because it permits to evaluate the distances between data and all nodes on the spherical variety. The downside is that it is generally more time-consuming, but more precise than others (LeBlanc et al. 1994).

The technique described above makes clear the role of PPS as an efficient method for MDS pre-clustering or dimensional reduction.

More generally, the advantage to preliminarily apply a dimensional reduction model to data is that, in some cases, data analysis such as regression or classification can be done in the reduced space more accurately than in the original one.

### 3.2.2 Classification

Classification is a procedure in which individual items are placed into groups based on quantitative information on one or more features inherent to the items (referred to as features) and based on a training set of previously labelled items (Kotsiantis 2007).

A classifier is a system that performs a mapping from a feature space X to a set of labels Y. Basically a classifier assigns a pre-defined class label to a sample.
Formally, the problem can be stated as follows: given training data \( \{(x_1, y_1), \ldots, (x_n, y_n)\} \) (where \( x_i \) are vectors) a classifier \( h: X \rightarrow Y \) maps an object \( x \in X \) to its classification label \( y \in Y \).

Different classification problems could arise:

a) crispy classification: given an input pattern \( x \) (vector) the classifier returns its computed label \( y \) (scalar).

b) probabilistic classification: given an input pattern \( x \) (vector) the classifier returns a vector \( y \) which contains the probability of \( y_i \) to be the "right" label for \( x \). In other words in this case we seek, for each input vector, the probability of its membership to the class \( y_i \) (for each \( y_i \)).

Both cases may be applied to both "two-class" and "multi-class" classification. So the classification task involves, at least, three steps:

- training, by means of a training set (input: patterns and target vectors, or labels; output: an evaluation system of some sort);
- testing, by means of a test set (input: patterns and target vectors, requiring a valid evaluation system from point 1; output: some statistics about the test, confusion matrix, overall error, bitfail error, as well as the evaluated labels);
- evaluation, by means of an unlabelled data set (input: patterns, requiring a valid evaluation systems; output: the labels evaluated for each input pattern);

Because of the supervised nature of the classification task, the system performance can be measured by means of a test set during the testing procedure, in which unseen data are given to the system to be labelled.

The overall error somehow integrates information about the classification goodness. However, when a data set is unbalanced (when the number of samples in different classes varies greatly) the error rate of a classifier is not representative of the true performance of the classifier. A confusion matrix can be calculated to easily visualize the classification performance (Provost et al. 1998): each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class. One benefit of a confusion matrix is the simple way to see if the system is mixing two classes.

Optionally (some classification methods does not require it by its nature or simply as a user choice), one could need a validation procedure.

Validation is the process of checking if the classifier meets some criterion of generality when dealing with unseen data. It can be used to avoid over-fitting or to stop the training on the base of an "objective" criterion.

With "objective" we intend a criterion which is not based on the same data we have used for the training procedure. If the system does not meet this criterion it can be changed and then validated again, until the criterion is matched or a certain condition is reached (for example, the maximum number of epochs). There are different validation procedures. One can use an entire data set for validation purposes (thus called validation set); this data set can be prepared by the user directly or in an automatic fashion.
In some cases (e.g. when the training set is limited) one could want to apply a "cross validation" procedure, which means partitioning a sample of data into subsets such that the analysis is initially performed on a single subset, while the other subset(s) are retained for subsequent use in confirming and validating the initial analysis (Mosteller et al. 1968).

Different types of cross validation may be implemented, e.g. k-fold, leave-one-out, etc. Summarizing we can safely state that a common classification training task involves:

- the training set to compute the model;
- the validation set to choose the best parameters of this model (in case there are "additional" parameters that cannot be computed based on training);
- the test data as the final "judge", to get an estimate of the quality on new data that are used neither to train the model, nor to determine its underlying parameters or structure or complexity of this model;

The validation set may be provided by the user, extracted from the software or generated dynamically in a cross validation procedure. In the next paragraphs we underline some practical aspects connected with the validation techniques for classification models.

### 3.2.3 Regression

Regression methods bring out relations between variables, especially whose relation is imperfect (i.e. it has not one y for each given x). The term regression is historically coming from biology in genetic transmission through generations, where for example it is known that tall fathers have tall sons, but not as tall on the average as the fathers. The trend to transmit on average genetic features, but not exactly in the same quantity, was what the scientist Galton defined as regression, more exactly *regression toward the mean* (Galton 1877).

But what is regression? Strictly speaking it is very difficult to find a precise definition. It seems the existence of two meanings for regression (Hastie et al. 2005), that can be addressed as data table statistical correlation (usually column averages) and as fitting of a function.

About the first meaning, let start with a very generic example: let’s suppose to have two variables x and y, where for each small interval of x there is a distribution of corresponding y. We can always compute a summary of the y values for that interval. The summary might be for example the mean, median or even the geometric mean. Let fix the points ($x_i$, $\bar{y}_i$), where $x_i$ is the center of the ith interval and $\bar{y}_i$ the average y for that interval. Then the fixed points will fall close to a curve that could summarize them, possibly close to a straight line. Such a smooth curve approximates the regression curve called the regression of y on x. By generalizing the example, the typical application is when the user has a table (let say a series of input patterns coming from any experience or observation) with some correspondences between intervals of x (table rows) and some distributions of y (table columns), representing a generic correlation not well known (i.e. imperfect as introduced above) between them. Once we have such a table, we want for example to clarify or accent the relation between the specific values of one variable and the corresponding values of the other. If we want an average, we might compute the mean or median for each column. Then to get a regression, we might plot these averages against the midpoints of the class intervals.

Given the example in mind let’s try to extrapolate the formal definition of regression (in its first meaning).
In a mathematical sense, when for each value of x there is a distribution of y, with density \( f(y|x) \) and the mean (or median) value of y for that x given by:

\[
\bar{y}(x) = \int_{-\infty}^{\infty} y f(y|x) dy
\]

then the function defined by the set of ordered pairs \((x, \bar{y}(x))\) is called the regression of y on x. Depending on the statistical operator used, the resulting regression line or curve on the same data can present a slightly different slope.

Sometimes we do not have continuous populations with known functional forms. But the data may be very extensive (such as in the astrophysical case). In these cases it is possible to break one of the variables into small intervals and compute averages for each of them. Then, without severe assumptions about the shape of the curve, essentially get a regression curve. What the regression curve does is essentially to give a “big summary” for the averages for the distributions corresponding to the set of x’s. One can go further and compute several different regression curves corresponding to the various percentage points of the distributions and thus get a more complete picture of the input data set. Of course often it is an incomplete picture for a set of distributions! But in this first meaning of regression, when the data are more sparse, we may find that sampling variation makes impractical to get a reliable regression curve in the simple averaging way described (Menard 2001). From this assumption, it descends the second meaning of regression.

Usually it is possible to introduce a smoothing procedure, applying it either to the column summaries or to the original values of y’s (of course after an ordering of y values in terms of increasing x). In other words we assume a shape for the curve describing the data, for example linear, quadratic, logarithmic or whatever. Then we fit the curve by some statistical method, often least-squares. In practice, we do not pretend that the resulting curve has the perfect shape of the regression curve that would arise if we had unlimited data, but simply we obtain an approximation. In other words we intend the regression of data in terms of forced fitting of a functional form. The real data present intrinsic conditions that make this second meaning as the official regression use case, instead of the first, i.e. curve connecting averages of column distributions. We ordinarily choose for the curve a form with relatively few parameters and then we have to choose the method to fit it. In many manuals sometimes it might be founded a definition probably not formally perfect, but very clear: by regressing one y variable against one x variable means to find a carrier for x.

This introduces possible more complicated scenarios in which more than one carrier of data can be founded. In these cases it has the advantage that the geometry can be kept to three dimensions (with two carriers) up to n-dimensional spaces (n>3, with more than two carriers regressing input data). Clearly, both choosing the set of carriers from which a final subset is to be drawn and choosing that subset can be most disconcerting processes.

In substance we can declare a simple, important use of regression, consisting in:

To get a summary of data, i.e. to locate a representative functional operator of the data set, in a statistical sense (first meaning) or via an approximated trend curve estimation (second meaning).

And a more common use of regression:

- For evaluation of unknown features hidden into the data set;
For prediction, as when we use information from several weather or astronomical seeing stations to predict the probability of rain or the turbulence growing in the atmosphere;

For exclusion. Usually we may know that $x$ affects $y$, and one could be curious to know whether $z$ is associated with $y$ too, through a possible casual mechanism. In this case one approach would take the effects of $x$ out of $y$ and see if what remains is associated with $z$. In practice this can be done by an iterative fitting procedure by evaluating at each step the residual of previous fitting.

This is not exhaustive of the regression argument, but simple considerations to help the understanding of the regression term and the possibility to extract basic specifications for the use case characterization in the design phase.

3.2.4 Clustering

Clustering is a division of data into groups of similar objects. Representing the data by fewer clusters necessarily loses certain fine details (data compression), but achieves simplification (Jain et al. 1999).

From a ML perspective clusters correspond to hidden patterns, the search for clusters is unsupervised learning, and the resulting system could represent a data concept in the KDD (Knowledge Discovery in Databases).

From a practical perspective clustering plays an outstanding role in DM applications such as scientific data exploration, information retrieval and text mining, spatial database applications, Web analysis, Customer Relationships Management (CRM), marketing, medical diagnostics, computational biology, and many others.

For example, in CRM, marketing applications generally come with predictive clustering analytics to improve segmentation and targeting, and features for measuring the effectiveness of online, offline, and search marketing campaigns (Collica 2007). By evaluating “buy signals,” marketers can see which prospects are most likely to transact and also identify those who are bogged down in a sales process and need assistance.

Data mining on MDS adds to clustering the complications of very large data sets with very many attributes of different types (high dimensionality). This imposes unique computational requirements on relevant clustering algorithms.

What are the properties of clustering algorithms we are concerned with in DM?

These properties include:

- Type of attributes that the algorithm can handle;
- Scalability to large data sets;
- Ability to work with high dimensional data (multi-D parameter space, multi-wavelength, multi-epoch etc…);
- Ability to find clusters of irregular shape;
- Handling outliers;
- Time complexity (when there is no confusion, we use the term complexity);
- Data order dependency;
- Labeling or assignment (hard or strict vs. soft of fuzzy);
Reliance on a priori knowledge and user defined parameters;
- Interpretability of results;

We have to try to keep these issues in mind, realistically. The above list is in no way exhaustive. For example, we must deal also with implementation properties, such as ability to work in pre-defined memory buffer, ability to restart and to provide an intermediate solution and so on.

### 3.2.5 Segmentation

Segmentation is synonym of “image processing” and in the DM with ML context is strictly correlated with image clustering functional domain. More in general, in computer vision, segmentation refers to the process of partitioning a digital image into multiple segments (sets of pixels, also known as superpixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze (Bishop 2006). Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics.

The result of image segmentation is a set of segments that collectively cover the entire image, or a set of contours extracted from the image (Lindeberg 2001). Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristics.

Strictly related to image segmentation, a not exhaustive list of methods is the following.

- The region growing method takes a set of seeds as input along with the image (Pratt 2007). The seeds mark each of the objects to be segmented. The regions are iteratively grown by comparing all unallocated neighboring pixels to the regions. The difference between a pixel intensity and region mean values is used as a measure of similarity. The pixel with the smallest difference is allocated to the respective region.
- The K-means algorithm is an iterative technique that is used to partition an image into K clusters, which is evolved in a variety of versions (Selim et al. 1984).
- Edge detection is a well-developed field on its own within image processing. Region boundaries and edges are closely related, since there is often a sharp adjustment in intensity at the region boundaries. Edge detection techniques have therefore been used as the base of another segmentation technique (Park et al. 2008). The edges identified by edge detection are often disconnected. To segment an object from an image however, one needs closed region boundaries.
- Histogram-based methods are very efficient when compared to other image segmentation methods because they typically require only one pass through the pixels. In this technique, a histogram is computed from all of the pixels in the image, and the peaks and valleys in the histogram are used to locate the clusters in the image (Shapiro et al. 2001).
- Graph partitioning methods can effectively be used for image segmentation (Zahn 1971). In these methods, the image is modeled as a weighted, undirected graph. The graph (image) is then partitioned according to a criterion designed to model "good" clusters. Each partition of
the nodes (pixels) output from these algorithms are considered an object segment in the image.

Multi-scale segmentation. Image segmentations can be computed at multiple scales in scale-space and sometimes propagated from coarse to fine scales (Bijaoui et al. 1995). Segmentation criteria can be arbitrarily complex and may take into account global as well as local criteria. A common requirement is that each region must be connected in some sense.

Neural Network segmentation relies on processing small areas of an image using an artificial neural network or a set of neural networks. After such processing the decision-making mechanism marks the areas of an image accordingly to the category recognized by the neural network. A type of network designed especially for this is the Kohonen map (Haykin 1998).

### 3.2.6 Feature Detection

Feature detection is a low-level image processing operation. That is, it is usually performed as the first operation on an image, and examines every pixel to see if there is a feature present at that pixel. If this is part of a larger algorithm, then the algorithm will typically only examine the image in the region of the features (Lindeberg 1998). As a built-in pre-requisite to feature detection, the input image is usually smoothed by a Gaussian kernel in a scale-space representation and one or several feature images are computed, often expressed in terms of local derivative operations. There is no universal or exact definition of what constitutes a feature, and the exact definition often depends on the problem or the type of application. Given that, a feature is defined as an "interesting" part of an image, and features are used as a starting point for many computer vision algorithms. Since features are used as the starting point and main primitives for subsequent algorithms, the overall algorithm will often only be as good as its feature detector. Consequently, the desirable property for a feature detector is **repeatability**: whether or not the same feature will be detected in two or more different images of the same scene.

### 3.2.7 Forecasting

Forecasting is the process of making statements about events whose actual outcomes (typically) have not yet been observed. A commonplace example might be estimation of the expected value for some variable of interest at some specified future date.

The term forecasting is sometimes reserved for estimates of values at certain specific future times, while the term prediction is used for more general estimates. Both might refer to formal statistical methods employing time series.

An important, albeit often ignored aspect of forecasting, is the relationship it holds with planning. Forecasting can be described as predicting what the future will look like, whereas planning predicts what the future should look like.

There is no single right forecasting method to use. Selection of a method should be based on your objectives and your specific conditions (Armstrong 2001).

### 3.2.8 Data Mining Model Filtering

Data-based model filtering helps you create complex architectures based on different and multiple mining models that use subsets of data in a filtered mining structure. A useful way to have a right vision of data-driven model filtering is:
Model filtering operates without altering the underlying model data. This allows one set of data to be shared among multiple components, each of which may interpret the data in a different manner. Filters can be layered, enabling model data to be interpreted through several different filter objects. (Goldstein, 2001).

Filtering gives you flexibility when you design your mining structures and data sources, because you can create a single mining structure, based on a comprehensive data source view. You can then create filters to use only a part of that data for training and testing a variety of models, instead of building a different structure and related model for each subset of data.

For example, it is possible to develop specialized DM models on a selected part of the data sets. Next, you define a multiple mining structure that includes all the features coming out the previous filtered ensemble of models. Finally, you create a model that is filtered on a particular customer attribute, such as a specific wavelength band. You can then easily make a copy of that model, and change just the filter condition to generate a new model based on a different spectrum region.

Some real-life scenarios where you might benefit from this feature include the following:

- Creating separate models for discrete values such as wavelength, regions, and so forth;
- Experimenting with models by creating and then testing multiple groupings of the same data;
- Specifying complex filters on nested data contents.

Data-based model filtering greatly simplifies the task of managing mining structures and mining models, because you can easily create multiple models that are based on the same structure. You can also quickly make copies of existing mining models and then change only the filter condition. Good examples of such filtered mining models are the Gated Experts (GE) (Weigend et al. 1995).

Although a single global DM model can in principle emulate any function, including regime switching, it is often very hard to extract such an unstructured, global model from the data. In particular, trying to learn regimes with different noise levels by a single network is a mismatch since the network will extract features that do not generalize well in some regime (local overfitting) before it has learned all it potentially could in another regime (local underfitting). A final motivation for different experts in different regions is that they can individually focus on that subset of input variables relevant for their specific region. This turns out to be particularly advantageous in modeling multivariate problems where different variables are important in different regimes.

The basic idea behind gated experts is simple: rather than using a single global model, we learn several local models (the experts) from the data. Simultaneously, we learn to split the input space. The problem is that the splitting of the input space is unknown because the only information available is the next value of the series. This requires blending supervised and unsupervised learning: the supervised component learns to predict the (observed) next value, and the unsupervised component discovers the (hidden) regimes. Since the only observable is the combination of the gate and the experts, many different ways of splitting the input space and fitting local models are possible. This trade-off between flexibility in the gates and flexibility in the experts is an important degree of freedom in this model class.
Summarizing, the key elements of gated experts are: (i) nonlinear gate and experts, (ii) soft-partitioning the input space, (iii) adaptive noise levels (variances) of the experts.

GE allow the noise-level parameter associated with each individual expert to adapt separately to the data. Generally speaking, expert-specific variances are important for two reasons: to facilitate the segmentation (areas of different predictability are grouped together), and to prevent overfitting (different regimes are approximated with different accuracy). This is a new (efficient) approach to the problem of overfitting.

3.3 Relationships with Statistics

Developments in the field of statistical data analysis often parallel or follow advancements in other fields to which statistical methods are fruitfully applied. Because practitioners of the statistical analysis often address particular applied decision problems, methods developments is consequently motivated by the learning of the better decision making under uncertainties (Vapnik 1998).

Statistical models are currently used in various fields of business and science. However, the terminology differs from field to field. For example, the fitting of models to data, called calibration, history matching, and data assimilation, are all synonymous with parameter estimation.

Data is known to be crude information and not knowledge by itself. The sequence from data to knowledge is: from Data to Information, from Information to Facts, and finally, from Facts to Knowledge. Data becomes information, when it becomes relevant to your decision problem. Information becomes fact, when the data can support it. Facts are what the data reveals. However the decisive instrumental (i.e., applied) knowledge is expressed together with some statistical degree of confidence (Ripley 2002).

The Figure 3 illustrates the statistical thinking process based on data in constructing statistical models for decision making under uncertainties.

![Figure 3. The typical exploration process in statistical data analysis.](image-url)
As the exactness of a statistical model increases, the level of improvements in decision-making grows as well. That’s why we need statistical data analysis. Statistical data analysis arose from the need to place knowledge on a systematic evidence base. This required a study of the laws of probability, the development of measures of data properties and relationships, and so on.

The following not exhaustive examples point to a set of methods largely approached on many aspects of statistical DM and data analysis.

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility (Aho et al. 1983). Decision trees are commonly used in operations research, specifically indecision analysis, to help identify a strategy most likely to reach a goal. Another use of decision trees is as a descriptive means for calculating conditional probabilities (Clark et al. 1992). When the decisions or consequences are modeled by computational verb, then we call the decision tree a computational verb decision tree. In decision analysis, a “decision tree” — and the closely-related influence diagram — is used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated. A decision Tree consists of 3 types of nodes, (see example in Figure 4):

1) Decision nodes - commonly represented by squares
2) Chance nodes - represented by circles
3) End nodes - represented by triangles

---

Figure 4. Example of a canonical decision tree.
Maximum Likelihood Estimation (MLE) is a popular statistical method used for fitting a statistical model to data, and providing estimates for the model's parameters (Aldrich 1997). The method of maximum likelihood corresponds to many well-known estimation methods in statistics, where usually you are interested in the estimation of any parameter trend of a physical problem. You have a sample of some number of such parameter, but not the entire population, and record their values. Further, you are willing to assume that the parameter samples are normally distributed with some unknown mean and variance. The sample mean is then the maximum likelihood estimator of the population mean, and the sample variance is a close approximation to the maximum likelihood estimator of the population variance. For a fixed set of data and underlying probability model, the method of maximum likelihood selects values of the model parameters that maximize the likelihood function.

Maximum likelihood estimation gives a unified approach to estimation, which is well-defined in the case of the normal distribution and many other problems. However, in some complex problems, difficulties do occur, in such problems the maximum-likelihood estimators may be unsuitable or may even fail to exist. A maximum likelihood estimator coincides with the most probable Bayesian estimator given a uniform prior distribution on the parameters. Much of the theory of maximum-likelihood estimation was first developed for Bayesian statistics (Sorensen 1980).

A Bayesian network is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (Pearl 1985). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Formally, Bayesian networks are directed acyclic graphs whose nodes represent random variables in the Bayesian sense; they may be observable quantities, latent variables, unknown parameters or hypotheses. Edges represent conditional dependencies; nodes which are not connected represent variables which are conditionally independent of each other. Each node is associated with a probability function that takes as input a particular set of values for the node's parent variables and gives the probability of the variable represented by the node. For example, if the parents are \( m \) boolean variables then the probability function could be represented by a table of \( 2^m \) entries, one entry for each of the \( 2^m \) possible combinations of its parents being true or false.

Efficient algorithms exist that perform inference and learning in Bayesian networks (Neapolitan 2003). Bayesian networks that model sequences of variables (e.g. speech signals) are called dynamic bayesian networks. Generalizations of Bayesian networks that can represent and solve decision problems under uncertainty are called influence diagrams.

A Hidden Markov Model (HMM) is a statistical model in which the system being modeled is assumed to be a Markov process with unobserved state (Baum et al. 1966). A HMM can be considered as the simplest dynamic Bayesian network.

In a regular Markov Model (MM), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a hidden Markov model, the state is not directly visible, but output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states. Note that the adjective hidden refers to the state sequence through which the model passes, not to the
parameters of the model; even if the model parameters are known exactly, the model is still hidden. HMMs are especially known for their application in temporal pattern recognition.

Figure 5. Example of a Hidden Markov Model.

In the Figure 5 there is shown an example of probabilistic parameters of a HMM, where:

- $x$: states;
- $y$: possible observations;
- $a$: state transition probabilities;
- $b$: output probabilities;

Support Vector Machines (SVM) are a statistical toolset of related supervised learning methods used for classification and regression (Vapnik 1995). In simple words, given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other (Chang et al. 2001). Intuitively, an SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

A neural network or Artificial Neural Network (ANN) is an interconnected group of natural or artificial neurons that uses a mathematical model for information processing based
on a connectionist approach to computation (McCulloch et al. 1943). In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network.

In more practical terms neural networks are non-linear statistical data modeling or decision making tools. They can be used to model complex relationships between inputs and outputs or to find patterns in data. In a neural network model simple nodes, which can be called variously "neurons", "neurodes", "Processing Elements" or "units", are connected together to form a network of nodes — hence the term "neural network". While a neural network does not have to be intrinsically adaptive, its practical use comes with algorithms designed to alter the strength (weights) of the connections in the network to produce a desired signal flow (Rumelhart et al. 1986).

Before this brief and not complete overview of techniques, we introduced the discussion on ML and statistics. From a realistic point of view there is not a proper difference between ML and statistics, although, from a conceptual perspective we could point out the prominence of testing hypothesis trend in statistics, against a process of generalization and search through several hypotheses typical of ML. But there are cases where these two features are not so different, for example in prediction/regression trees (Breiman et al. 1984), because their investigation approach is practically the same.

Furthermore recent researches are going in the direction of a merging between the two disciplines, trying to make interoperable the techniques and models derived from both cases. The modern trend is to build workflows in which data are pre-processed in a statistical sense, in order to define and obtain a refinement of the initial data set, and after application of a ML model, often output statistical evaluation methods are employed, such as visualization, selection of attributes, extracting outliers, prediction performance validation, overfitting resolution etc.

So far, there is not a direct "competition" between ML and statistics, but a synergy between them to build efficient and reliable classification or regression systems to extract knowledge from data.

More concerning the generalization capability of ML as an optimization mechanism, that is the basic difference from statistics, its main aspect is to have a search through a multi-dimensional space (parameter space) of possible solution description to a problem for one that fits data. Of course the process could be considered extremely large, requiring a huge amount of time and experiments, but it is finite and there always exists the optimal solution.

4. Machine Learning in Practice

4.1 How to Represent Data

Before going into the working mechanisms of ML systems, it is interesting and useful to focus the attention on the shape and expressions of data to be given as input to ML information processing models.

In ML experiments the performance strongly depends from data used for training. Hence it is crucial the choice of data selection and their representation mode.
Generally, except for some particular cases, the set of input data is provided under the form of tables or matrices; in which any row identify an example (a complete pattern in the data parameter space), whose columns are all parameters (features) and their values the parameter attributes.

It may be frequent that the table can have empty entries (sparse matrix) or missing (lack of observed values for some features in some patterns). It may also happen that information of a single table is not homogeneous, i.e., attributes may be of different types, such as numerical mixed with categorical entries.

This level of diversity in the internal information could be also related with different format type of data sets, such as tables registered in ASCII code (ANSI et al. 1977), CSV (Comma Separated Values) (Repici 2002) or FITS (text header followed by binary code of an image) (Wells et al. 1981).

In order to reach an efficient and homogeneous representation of data sets, to be submitted to ML systems, it is mandatory to preliminarily take care of the data format, in order to make them intelligible by the processing framework. In other words to transform pattern features to assume a uniform representation before to submit them to the training process.

In this mechanism the real situations could be very different. Let think to time sequences (coming from any sensor monitoring acquisition), where data are collected in a single long sequence, not simply divisible, or to raw data (such as original images taken by astronomical observations), that could be affected by noise or aberration factors.

These events always require a pre-processing phase, to clean and opportune prepare the data sets to be used for any ML and DM experiment. Of course such preliminary step must take into account also the functional scope of the experiment itself.

More in practice, having in mind the functional taxonomy described in the previous section, there are essentially four kinds of learning related with ML for DM:

1) Learning by association;
2) Learning by classification;
3) Learning by prediction;
4) Learning by grouping (clustering);

The primer, learning by association consists of the identification of any structure hidden between data. It does not mean to identify the belonging of patterns to specific classes, but to predict values of any feature attribute, by simply recalling it, i.e., by associating it to a particular state or sample of the real problem.

It is evident that in the case of association we are dealing with very generic problems, i.e., those requiring a precision less than in the classification case. In fact, the complexity grows with the range of possible multiple values for feature attributes, potentially causing a mismatch in the association results.

In practical terms, fixed percentage thresholds are given in order to reduce the mismatch occurrence for different association rules, based on the experience on that problem and related data. The representation of data for associative learning is thus based on the labeling of features with non-numerical values or by alpha-numeric coding.
Classification learning is often named simply “supervised” learning, because the process to learn the right assignment of a label to a datum, representing its category or “class”, is usually done by examples. Learning by examples stands for a training scheme operating under supervision of any oracle, able to provide the correct, already known, outcome for each of the training sample. And this outcome is properly a class or category of the examples. Its representation depends on the available Base of Knowledge (BoK) and on its intrinsic nature, but in most cases is based on a series of numerical attributes, related to the extracted BoK, organized and submitted in an homogeneous way.

The success of classification learning is usually evaluated by trying out the acquired feature description on an independent set of data, having known output but never submitted to the model before.

Slightly different from classification scheme is the prediction learning. In this case the outcome consists of a numerical value instead of a class.

The numeric prediction is obviously related to a quantitative result, because is the predicted value much more interesting than the structure of the concept behind the numerical outcome.

Whenever there is no any class attribution, clustering learning is used to group data that show natural similar features. Of course the challenge of a clustering experiment is to find these clusters and assign input data to them. The data could be given under the form of categorical/numerical tables and the success of a clustering process could be evaluated in terms of human experience on the problem or a posteriori by means of a second step of the experiment, in which a classification learning process is applied in order to learn an intelligent mechanism on how new data samples should be clustered.

Coming back to the data representation and preparation for DM with ML methods, whatever being the functional analysis chosen, all experiments require a standardization of data in a set of homogeneous patterns composed by features with attributes. This process in principle not always is possible. For example what happens if different patterns contain different features? For instance, in the very general category of animals, one feature could be the number of legs. But this is a typical feature related to terrestrial species, not for fishes.

Well, from a practical point of view it is used a special value “irrelevant” or “null” as a categorical flag to indicate that for a particular feature in some patterns a specific value doesn’t exists.

The attributes of a feature could in principle assume either numeric or nominal values. Nominal attributes assume values restricted in a predefined range of categories. On the contrary numeric attributes simply measure numbers, real or integers (Witten et al. 2005).

But this dichotomy shows all its intrinsic limits when we refer to most real problems in which data features are to be considered in a statistical sense. In this case, in fact, we have to treat attribute values in terms of “levels of confidence”. They could be symbolic quantities, without any kind of relationship between them, in which it doesn’t exists any distance metric. But in some cases also ordinal quantities can be expressed as levels of confidence. This category differs from nominal values because makes possible to rank order the categories. Other two kinds of levels of confidence are interval and ratio quantities.

Intervals are quite intuitive and simple to be defined, having values ordered and measured in a fixed and well identified units.
Ratio quantities are those more interesting and directly related to statistics. From a mathematical point of view, ratios imply a measuring scheme which intrinsically defines a zero point. For instance, when we measure a distance between objects, the distance of an object from itself is unequivocally zero.

Despite such categories of representation, most practical DM methods, based on ML paradigm, when dealing with massive data sets make an intensive use so-called “meta-data”, another category of data representation, based on partial ordering or equivalently generalization/specialization relations. A meta-datum (from Greek meta “over, after” and Latin datum “information”) is the information that describes a set of data (Guenther et al. 2004).

A typical example of meta-data is an entry of a library catalogue, containing summarized information about the contents and archive position of a book. In this case summarized information could be the author name, the book title and a brief abstract, but also the number of available copies in the library.

Fields of a meta-data collection are composed by information describing resources and quick notes related to the referred original data, able to improve their fast visibility and access. They also provide the primary information retrieval, indexed through description fields, usually formatted as records (pattern of labels).

There are also more complex cases, in which meta-data are organized in a hierarchical structure. In such cases they are usually defined as ontologies, which represent semantic expressions univocally describing categories of data features and their meaning. For example the organization of titles for a library catalogue identifies categories of themes (indeed ontologies) in which all books are collected. The association of a group of books to a single theme, makes more easy their search.

In the context of digital information the term meta-data is related with both philosophical and computer science aspects. From the Informatics discipline derives the semantic restriction of the concept of a “datum” as an unitary information block that can be digitalized on a computer. From Philosophy derives the concept of “meta” as a prefix useful to denote an alternative kind of relationship between types of semantically analogous terms. The design of a well posed set of ontologies for a data warehouse is thus a crucial task to improve the efficiency in the information retrieval operations. Meta-data in fact, may be used also to provide an efficient functional description of data and documents in an information system, such as the author certification, validity expiration date, fast and safe location of the underlying base of knowledge contained in a document, quick semantic relationships with other documents and so on.

In the modern web based information era, we assist to a proliferation of meta-languages and standard representations of meta-data, able to improve the unambiguous worldwide exchange of information. An important example is the Extensible Markup Language (XML), a very flexible text format originally designed to meet the challenges of large-scale electronic publishing, and quickly became an increasingly important standard in the exchange of a wide variety of data on the Web (Bray et al. 2006).

At the beginning of this paragraph we introduced the double aspects of standards for data representation and data preparation. For what concerns this second argument, in all ML based experiments it is usual and needed to organize data bringing them together in a set of patterns
New Trends in E-Science: Machine Learning and Knowledge Discovery in Databases

and features. The preparation of data introduces two important techniques: warehousing and normalization.

With the modern explosion of data production, involving a huge variety of scientific and social disciplines, there is a crucial necessity to integrate and make interoperable data from different sources. Nothing of theoretical but simply a series of practical rules. The problem arises from the natural distribution of data extraction sources, spread over different departments, instruments and collaborations. Such data centers often use different data formats, conventions, collecting times, databases with not homogeneous primary keys and specific instrument stamps.

Besides these aspects, there is the need to collect and organize single massive data sets to be analyzed and verified in a multi-disciplinary environment. In these cases the data must be cleaned and assembled before to submit them to ML experiments.

The technique now practically employed to implement such requirements is known as data warehousing (Inmon 1992).

A data warehouse organize data and defines standard formats, procedures, archiving systems and provides interoperable navigation services to access data in an efficient and consistent way. Its final goal is to provide a single access point to a distributed data archive, in which data are organized in a consistent and homogeneous way and the user access is standardized, eliminating all original differences in the data representation. An efficient example is in the astrophysical community, where data archives, extracted from observations and simulations, are federated under a standard infrastructure, called Virtual Observatory (VO), managed and maintained by an international collaboration program, known as International Virtual Observatory Alliance (IVOA) (Pasian et al. 2007).

We point out that the recognition of multiple-source data federation as a crucial aspect of modern multi-disciplinary science may have an immense strategic valence in the future discovery process.

The efficient organization of distributed data is not the unique problem related with the preparation for experiments. Another important aspect is normalization.

How data are interpreted strongly depends on the ML technique used. For example, in many cases data are treated as numeric attributes and sorted by simple laws like greater-than or less-than comparisons. In most cases data are also handled as ratio scales and organized by distance metrics (Witten et al. 2005).

In all cases, the question of attribute value normalization arises. In ML there are usually two normalization schemes. One that normalizes data in the [0,1] range, by dividing all of them by the maximum value or by subtracting the minimum and dividing by the range between the maximum and minimum encountered.

An extension of this rule is also used in practice whenever a normalization range of [-1, +1] is required. This arises in many ML methods, in which an enlargement of the range produces effective benefits to the learning scheme. In analogy with the human brain, a negative signal as input can represent a sort of inhibition of the involved neuron stimuli response, while on the contrary a positive input represents a reinforcement of the neuron response.

Moreover, in statistical problems, the usual normalization consists of calculating mean and standard deviation of the attribute values, subtracting the mean from each of them and
dividing by the standard deviation. This process is especially used in all cases in which it is considered a statistical data distribution with zero mean and unitary standard deviation.

There are also special cases in which the numeric attribute is a codification of the nominal value. In these cases, the normal logic comparisons or distance metrics must be applied *cum grano salis*.

In presence of inaccurate values for data features, the normalization, or more generally, any conversion process usually preserves their conditions. But why could there be the presence of inaccurate values into an homogeneous set of data? Well, we must take in mind a simple consideration: data, coming from any source and related to a generic experiment, are in most case not expressly collected for DM purposes. So far it may be happened that, when originally gathered, some features were not considered relevant and thus left unchecked. Their normalization or standardization process does not change their original wrong condition, causing the potential presence of inaccurate values into the patterns. Inaccurate values could be sometimes related to systematic data errors, for example due to an hardware setup condition in the data collecting instrument. However this is usually trivial to be recovered, if user has knowledge about the collecting device conditions, but in any case a deep care about the presence of inaccurate values should be required whenever a DM process is approached. And having available an expert of data domain could prevent any wrong result in the experiment.

The process of standardization and federation of distributed data collections implies also the identification of a uniform and consistent set of features to represent patterns. There could be the possibility that some groups of data to be federated have some missing values (not foreseen for a particular data source, probably absent in the setup of any of different instruments used, due to measurement errors or embedded between other available features). Missing values are frequently identified by special entries in the patterns, like Not-A-Number, out-of-range, negative values in a numeric field normally accepting only positive entries and so on.

Missing data is one of the frequent sources of perturbation in the learning process, causing confusion in a classification experiment or mismatching in a regression problem.

User has to take particular care to the missing values in his data. Although they could arise by a simply unknown reason during data collecting process, sometimes there could be a good reason of their presence between the data, such as, for instance, as result of a particular decision or as specific information about an instance for a subset of patterns. In these cases it is not simply missing, but desired missing, like in the outlier discovery and mining experiments.

This fact implies that a special care should be dedicated in the analysis of possible presence and causes of missing values, together with the decision on how to submit missing data to the ML method to take into account such special cases, in order to prevent wrong behaviors in the learning process.

The preparation of data is not only related to the input data set to be submitted in DM experiments. The user must preliminarily know also how to represent output data. Of course it depends on the specific functional domain associated to the current experiment. Depending on the ML method employed, the user must be conscious of the target of his experiment, such as for instance a regression, classification, clustering or an association of output data.
There are different techniques to infer knowledge from data output. In the case of any decision to be taken, it is common to represent output in the form of a decision table, in which the rows are the input data patterns and the columns report the category for a pattern related to a particular type (numeric or nominal) of attribute for a pre-selected number of features. Although this is a very simple technique, its efficiency is strictly connected to the selection of features used to quantify the learning performance (Witten et al. 2005).

A more sophisticated approach consists in the creation of a decision structure based on the “divide et impera” concept. In this case through a nested and hierarchical series of comparisons between the attributes of a pattern (direct or through any function, such as a distance metric), a final decision level is associated to that pattern (for example the decision in a multi-class classification problem). This structure is known as decision tree, already encountered in the previous sections of this chapter.

Whenever the prediction target is not a specific class but a proper attribute, there is a huge variety of association rules able to categorize any input pattern. In these cases the output evaluation can be done in terms of accuracy of the prediction, i.e. the number of patterns correctly predicted in respect of the entire data set entries, together with the absolute performance, i.e. the number of patterns correctly predicted.

Tree-based structures can be approached for both classification and regression problems (in the second case they are known as regression trees), as alternative output data representation in the classical tabular format. The only difference is that the leaves in a decision tree are the outputs in terms of categories assigned for a pattern, while in a regression tree each leaf reports the output in terms of averaged numeric value for a pattern.

In case of an output evaluation performed by any distance metric, we are referring to problems in which the output for a pattern is evaluated in terms of its “instance-based learning”, i.e. each new pattern is compared with existing ones in terms of the selected distance metric and the new pattern is assigned to the class associated with the closest existing instance (Aha et al., 1991). The most famous related technique, although not the best performing, is the k-nearest-neighbor. The processes able to create clusters in the data parameter space, as regions or over-densities in which all included patterns have similar feature attributes, are the clustering rules, in which each pattern is evaluated in terms of its degree (categorical or probabilistic) of membership to a particular group (cluster) of instances.

In conclusion to this very brief discussion about data representation, it results evident that in all cases a DM with ML experiment requires always a deep analysis and knowledge about data. In case of massive data sets, the user can approach the investigation by extracting randomly a subset of data and look at them carefully in order to operate cleaning and to make decisions about patterns, features and attributes to be organized for future experiments. The presence of domain experts could of course simplify and reduce this time-consuming preliminary activity. But anyway take into account that a huge amount of time must be always dedicated to the pre and post data processing in a ML experiment. There are interesting articles in literature showing that at least the 60% of time of a DM application is dedicated to the data preparation and verification (CABENA et al. 1998).
4.2 How to Learn Data

In the wide variety of possible applications for ML, DM is of course one of the most important, but also the most challenging. Users encounter as much problems as massive is the data set to be investigated. To find hidden relationships between multiple features in thousands of patterns is hard, especially by considering the limited capacity of human brain to have a clear vision in a multiple than 3D parameter space.

In order to deeply discuss the learning of data we recall the paradigm of ML, by distinguish between data where features are provided with known labels (target attributes), defined as supervised learning, and data where features are unlabeled, called unsupervised learning. With such concepts in mind we can discuss in the next sections the wide-ranging issues of both kinds of ML.

4.2.1 Methodologies for Supervised Learning

Artificial neural networks are one of the best examples of ML methods, inspired by the human brain architecture and learning rules. Dealing with supervised learning, these models need training patterns formed by feature-target couples. Indeed for each given input pattern (list of features), there should be also given the corresponding list of targets (one or more). We already called such a complete data set as Base of Knowledge (BoK). With this data set, the network could be able to learn the right association between input features and location of its output in the parameter space. The network will be able, after this training process, to correctly classify any pattern, even if not presented to the network in the training phase (generalization).

One of the simplest models of supervised neural networks is the Perceptron (Rosenblatt 1957), composed by a single output neuron and N input neurons. The capabilities of this model are very limited, but it is a good starting point for more advanced and complex variants of such network.

Its output is defined as follows:

\[ y = H(\sum_{i=1}^{n} w_i x_i - \theta), \text{where } H(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases} \quad (2) \]

(H is called Heaviside activation function). Previous expression (2) can be simplified by introducing a fake input (bias term) \( x_0 = 1 \) so that \( w_0 = -q \) in order to obtain:

\[ y = H(\sum_{i=1}^{n} w_i x_i) \quad (3) \]

The network learns by modifying the weights to enforce right decisions and discourage those wrong. At each iteration a new pattern of the training set is presented and the network calculates its output. Then the weights are updated by the expression

\[ w_i^{(t+1)} = w_i^{(t)} + \eta(c - y)x_i \quad (4) \]

where \( c \) is the class label (the expected output, or target) and \( \eta \) is a gain term \((0 < \eta \leq 1)\) which regulates the learning speed.
Main limit of such model is that it is able to correctly classify the input only if classes are linearly separable (see Figure 6a). However the division between classes is much more complex. A typical example is a problem (see Figure 6b), where it is not possible to find a single split line for the two classes and the Perceptron fails. To overcome this problem it is needed to employ more complex classification structures, organized on more than one computational layer (Cybenko 1989).

In order to be able to operate non-linear classification, i.e. to separate complex regions, the solution is to extend the perceptron to the so-called Multi-Layer Perceptron (MLP), a network composed by one or more hidden layers of neuron, fully connected, between input and output layers (Figure 7).

![Figure 6. correct (a) and wrong (b) separation of classes made by a perceptron.](image)

![Figure 7. Regions recognized by a MLP with 0, 1, 2 hidden layers.](image)

The classical topology of MLP is shown in Figure 8. This kind of networks is able to recognize and classify any type of topological region, having as downside a more complex learning process. Moreover, the Heaviside function cannot be applied as activation function, because it is not differentiable. An alternative is to use the sigmoid function, shown in Figure 9. In this case the activation values of output neurons become differentiable functions of input values and hidden neuron weights.

The practice and expertise in the ML methods, such as MLP, are important factors, formed through a long exercise within scientific experiments. In particular the speed and effectiveness of the results strongly depend on these factors. Unfortunately there are no magic
ways to a priori indicate the best configuration of internal parameters, involving network topology and learning algorithm, but a series of heuristics.

\[ y = f(\sum_{i=0}^{n} w_i x_i) \]  \hspace{1cm} (5)

Furthermore, if we define an error function as the Mean Square Error (MSE) between expected and network output, we found that it is a differentiable function of output and weights.

The learning process based on such rule is the so-called BackPropagation (BP), because the computed error is back propagated in the network from output to input layer (Rumelhart et al. 1986). More in detail, let us consider a generic n-dimensional input pattern \( p \) and the corresponding target (expected output vector) \( c_p \), and let define the error function for the pattern as:

\[ E_p = \frac{1}{2} \sum_{i} (c_{pi} - y_{pi})^2 \]  \hspace{1cm} (6)

Figure 8. Classical topology of a MLP, with hidden neurons in white.

In the MLP, the output of a single neuron is a function of inputs and connection weights:

Figure 9. Comparison between Heaviside and sigmoid activation function.
New Trends in E-Science: Machine Learning and Knowledge Discovery in Databases

where $y_i$ is the i-th neuron output in the network. The learning algorithm must update the weights in order to minimize the error function by following the rule based on the descent gradient. As defined the network it results that the input and output of a generic neuron $j$ are:

$$\text{net}_{pj} = \sum_i w_{ij} y_{pi} \text{ and } y_{pj} = f(\text{net}_{pj})$$

(7)

where $f$ is the sigmoidal function. The derivative of error function, for the network weights can be written as:

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial \text{net}_{pj}} \frac{\partial \text{net}_{pj}}{\partial w_{ij}}$$

(8)

and, for the second term, can be replaced by:

$$\frac{\partial \text{net}_{pj}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k w_{kj} y_{pk} = \sum_k \frac{\partial w_{kj}}{\partial w_{ij}} y_{pk} = y_{pi}$$

(9)

Because $\frac{\partial w_{kj}}{\partial w_{ij}} = 0$ except for $k = i$ where it is equal to 1. By indicating the first term of the derivative of the error function as $-\delta_{pj}$ we obtain:

$$-\frac{\partial E_p}{\partial w_{ij}} = \delta_{pj} y_{pi}$$

(10)

In order to minimize the error function $E_p$, the weight must be updated proportionally to $\delta_{pj} y_{pi}$, by obtaining a variation rule as:

$$w_{ij} = w_{ij} + \Delta w_{ij}$$

(11)

where $\Delta w_{ij} = \eta \delta_{pj} y_{pi}$ and $\delta_{pj}$ for each neuron is obtained by using the usual chain rule:

$$\delta_{pj} = \frac{\partial E_p}{\partial \text{net}_{pj}} = \frac{\partial E_p}{\partial y_{pj}} \frac{\partial y_{pj}}{\partial \text{net}_{pj}}$$

(12)

By making the mathematical relations in practice, we can derive the complete standard algorithm for a generic MLP trained by BP rule as the following (ALG-1):

Let us consider a generic MLP with $m$ output and $n$ input nodes and with $w_{ij}^{(t)}$ the weight between i-th and j-th neuron at time ($t$).

1) Initialize all weights $w_{ij}^{(0)}$ with small random values, typically normalized in [-1, 1];
2) Present to the network a new pattern $p = (x_1, ..., x_n)$ together with the target $c_p = (c_{p1}, ..., c_{pm})$ as the value expected for network output;
3) Calculate the output of each neuron $j$ (layer by layer) as: $y_{pj} = f(\sum_i w_{ij} x_{pi})$, except for the input neurons (in which their output is the input itself);
4) Adapt the weights between neurons at all layers, proceeding in the backward direction from output to input layer, with the following rule: 
\[ w_{ij}^{(t+1)} = w_{ij}^{(t)} + \eta \delta_{pj} y_{pj} \], where \( \eta \) is the gain term (also called learning rate) typically in \([0,1]\);

5) Goto 2 for all input patterns of the training set;

As shown, the BP learning rule tries to adapt weights in order to minimize the error function \( E(w) \). For networks without hidden layers, the error function will be squared and will assume a multi-dimensional parabolic shape, with only one absolute minimum. But for a generic MLP, the error function will be much more complex, with more than one minimum (local minima) in which the error gradient is zero (Figure 10).

In these last cases it is important to distinguish between absolute and local minima. When, during the learning process, the error finds a local minimum, with the above adaption rule, the error function will not move anymore, resulting in a wrong (not absolute) minimization state.

There are basically two versions of the Descent Gradient Algorithm (hereinafter DGA): online and batch.

In the online version, referred to the above algorithm, the weights are updated after each input pattern presentation.

![Figure 10. The typical behavior of the error functions during the learning process.](image)

If we indicate with \( w \) the weight vector of the network and with \( \nabla E_p|_w \) the gradient of error function for the pattern \( p \), evaluated at weight \( w \), we obtain the following weight update rule (in vector form), also shown in Figure 11:

\[ w^{(t+1)} = w^{(t)} + \Delta w^{(t)} = w^{(t)} - \eta \nabla E_p|_w^{(t)} \]  \( (13) \)

and the training patterns can be considered as either sequential or randomly selected.
In the *batch* version, the weights are updated after each presentation of the whole training set, such that the (13) can be rewritten as:

$$w^{(t+1)} = w^{(t)} - \eta \sum_p \nabla E_p \mid_{w^{(t)}}$$  \hspace{1cm} (14)

Between the two approaches, the first is preferable if there is a high degree of redundancy in the training set information, otherwise the second is the best.

Moreover, in all cases the descent gradient is not fast to converge. Fortunately there exist several methods to overcome these limits. In particular, in the batch case it results relatively easy to make DGA as a multi-threaded process, in which the training data sets are split into equally large batches for each of the threads (Heaton 2009).

Both versions require that the learning rate is a priori defined in a static way. This is another important point, because an high value of learning rate causes a more instable convergence of the learning process (the error function jumps along the error surface without convergence assurance). For a learning rate too small, the convergence will result extremely slow.

The good compromise is to gradually reduce, at each learning step, the value of the learning rate (for example by simply following the law $\eta = 1/t$ or by applying more complex rules), obtaining a faster convergence of the algorithm (Jacobs 1988).

Another method, able to improve the behavior of DGA against local minima occurrence, is to add a so-called *momentum* term to the DGA updating rule. In this case the weight updating formula becomes:

$$w^{(t+1)} = w^{(t)} + \Delta w^{(t)} = w^{(t)} - \eta \nabla E^{(t)} + \mu \Delta w^{(t-1)}$$  \hspace{1cm} (15)

where $\mu$ is the momentum term.

To understand the effect of such term, let consider the weight values in an error region with a smoothly sloping, as in Figure 11a. If we approximate the gradient value, we can iterate a long series of weight updates and to make a sum of the results of the mathematical series, obtaining:

$$\Delta w = -\eta \nabla E (1 + \mu + \mu^2 + \cdots) = -\frac{\eta}{1-\mu} \nabla E$$  \hspace{1cm} (16)
where the momentum term has the effect to increase the effective learning rate from \( \eta \) to \( \eta / (1 - \mu) \). So far, we obtain a faster convergence of the algorithm (Phansalkar 1994). On the other hand, in a region with a large curvature, as in Figure 11b, in which the descent gradient is oscillating, sequential contributes of momentum term tend to cancel and the effective learning rate will be \( \eta \).

The downside of this technique is that the momentum term does not improve dramatically the algorithm and it also represents one more parameter that requires a proper definition. Moreover, the choice of learning rate and momentum is not driven by a deterministic rule, depending strongly on the specific problem. It could be better to introduce a procedure to calculate the optimal choice in an automatic way. This could be obtained by introducing the technique of bold driver (Vogl et al. 1988).

To understand this method, let us consider the learning without momentum term. The idea is to check that the error function is effectively slowing down, after a weight update. If the error grows up, then the minimum has been bypassed and hence the learning rate is too high. In this case the weight update is removed and the learning rate decreased.

We repeat this procedure until an effective error decreasing is obtained. When it happens the weight is effectively updated. In practice, the rule is:

\[
\eta_{new} = \begin{cases} 
\rho \eta_{old} & \text{if } \Delta E < 0 \\
\sigma \eta_{old} & \text{if } \Delta E > 0 
\end{cases}
\]

(17)

by choosing respectively \( \rho \) higher than 1 and \( \sigma \) between \([0, 1]\).

In practical terms, we can optimize the previous MLP with BP algorithm (ALG-1) by introducing the bold driver and momentum terms (ALG-2), with typical (experienced) values for \( \rho \) and \( \sigma \):

Let us consider a generic MLP with \( w^{(t)} \) the weight vector at time \( t \).

1) Initialize all weights \( w^{(0)} \) with small random values (typically normalized in \([-1, 1]\)) and set \( \mu, \eta, \rho = 1.1, \sigma = 0.5, \varepsilon \) by imposing also \( t = 0 \);

2) Present to the network all training set and calculate \( E(w^{(t)}) \) as the error function for the current weight configuration;

3) Calculate \( E(w') \), where \( w' = w^{(t)} - \eta \nabla E^{(t)} + \mu \Delta w^{(t-1)} \);

   a) if \( E(w') < E(w^{(t)}) \) then \( w^{(t+1)} = w' \) and \( \eta = \eta \rho \); then goto 4;

   b) else \( \eta = \eta \sigma \) and goto 3;

4) if \( E(w^{(t+1)}) > \varepsilon \) then \( t = t+1 \) and goto 2; else STOP;

By using the standard DGA, the direction of each updating step is calculated through the error descent gradient, while the length is determined by the learning rate. A more sophisticated approach could be to move towards the negative direction of the gradient (line search direction) not by a fixed length, but up to reach the minimum of the function along
New Trends in E-Science: Machine Learning and Knowledge Discovery in Databases

that direction. This is possible by calculating the descent gradient and analyzing it with the variation of the learning rate.

Let suppose that at step t the current weight vector is \( w(t) \) and consider a search direction \( d_0 = -\nabla E(t) \). If we select the parameter \( \lambda \) in order to minimize \( E(\lambda) = E(w(t) + \lambda d_0) \). The new weight vector can be then expressed as:

\[
    w(t+1) = w(t) + \lambda d(t)
\]

(18)

The problem of line search is in practice a single dimension minimization problem. A simple solution could be to move \( E(\lambda) \) by varying \( \lambda \) in small intervals, to evaluate the error function at each new position and to stop when the error starts to decrease. There exist many other methods to solve this problem. For example the parabolic search of a minimum calculates the parabolic curve crossing pre-defined learning rate points. The minimum \( d \) of the parabolic curve is a good approximation of the minimum of \( E(\lambda) \) and it can be reached by considering the parabolic curve crossing the fixed points with the lowest error values.

There are also the trust region based strategies to find a minimum of an error function, which main concept is to iteratively growing or contracting the region of the function by adjusting a quadratic model function which better approximates the error function. In this sense this technique is considered dual to line search, because it tries to find the best size of the region by preliminarily fixing the moving step (the opposite of the line search strategy that always chooses the step direction before to select the step size), (Celli et al. 1985).

Up to now we have supposed that the optimal search direction for the method based on the line search is given at each step by the negative gradient. That’s not always true!

If the minimization is done along the negative gradient, next search direction (the new gradient) will be orthogonal to the previous one. In fact, note that when the line search founds the minimum, then we have:

\[
    \frac{\partial E}{\partial \lambda}(w(t) + \lambda d(t)) = 0
\]

(19)

and hence,

\[
    g(t+1)^T d(t) = 0
\]

(20)

where \( g(t+1) \equiv \nabla E(t+1) \).

By selecting further directions equal to the negative gradient, there should be obtained some oscillations on the error function that slow down the convergence process. The solution could be to select further more directions such that the gradient component, parallel to the previous search direction (that is zero), remains unchanged at each step.

Let suppose to have already minimized in respect of the direction \( d_0 \) starting from the point \( w(t) \) and reaching the point \( w(t+1) \).

In the point \( w(t+1) \) the (20) is \( g(w(t+1))^T d(t) = 0 \) and by choosing \( d(t+1) \) to preserve the gradient component parallel to \( d(t) \) equal to zero, it is possible to build a sequence of directions \( d \) in such a way that each direction is conjugated to the previous on the dimension \( |w| \) of the search space (conjugate gradients method), (Golub et al. 1999).
In presence of a square error function, an algorithm with such technique has a weight update of the form:

\[ w^{(t+1)} = w^{(t)} + \alpha^{(t)} d^{(t)} \]  \hspace{1cm} (21)

with

\[ \alpha^{(t)} = -\frac{d^{(t)^T g^{(t)}}}{d^{(t)^T H d^{(t)}}} \]  \hspace{1cm} (22)

Furthermore, \( d \) can be obtained for the first time by the negative gradient and then as linear combination of the current gradient and of the previous search directions:

\[ d^{(t+1)} = -g^{(t+1)} + \beta^{(t)} d^{(t)} \]  \hspace{1cm} (23)

with

\[ \beta^{(t)} = \frac{g^{(t+1)^T H d^{(t)}}}{d^{(t)^T H d^{(t)}}} \]  \hspace{1cm} (24)

This algorithm founds the minimum of a square error function in almost \( |w| \) steps. On the contrary, the computational cost of each step is high, because in order to determine the values of \( \alpha \) and \( \beta \), we have to refer to the hessian matrix \( H \), highly expensive in terms of calculations. But fortunately, the coefficients \( \alpha \) and \( \beta \) can be obtained from analythical expressions that do not use the Hessian matrix explicitely. For example the term \( \beta \) can be calculated in one of the following ways:

1) expression of Polak-Ribiere: \( \beta^{(t)} = \frac{g^{(t+1)^T (g^{(t+1)} - g^{(t)})}}{g^{(t)^T g^{(t)}}} \)

2) expression of Hestenes-Sitfel: \( \beta^{(t)} = \frac{g^{(t+1)^T (g^{(t+1)} - g^{(t)})}}{d^{(t)^T (g^{(t+1)} - g^{(t)})}} \)

3) expression of Fletcher-Reeves: \( \beta^{(t)} = \frac{g^{(t+1)^T g^{(t+1)}}}{g^{(t)^T g^{(t)}}} \)

These expressions are equivalent if the error function is square-typed, otherwise they assume different values. Typically the Polak-Ribiere equation obtains better results, because, if the algorithm is slow and the consequent gradients are quite similar between them, this equation produces values of \( \beta \) such that the search direction tends to assume the negative gradient direction (Vetterling et al. 1992). And this corresponds to a restart of the procedure.

Concerning the parameter \( \alpha \), its value can be obtained by using the line search method directly.

The method of conjugate gradients reduces the number of steps to minimize the error up to a maximum of \(|w|\) because there could be almost \(|w|\) conjugate directions in a \(|w|\)-dimensional space. In practice however, the algorithm is slower because, during the learning process, the property "conjugate" of the search directions tend to deteriorate.

It is useful, to avoid the deterioration, to restart the algorithm after \(|w|\) steps, by resetting the search direction with the negative gradient direction. The following algorithm (ALG-3)
New Trends in E-Science: Machine Learning and Knowledge Discovery in Databases

shows the ALG-2 in which it is introduced the conjugate gradients method together with line search to determine the learning rate, while the momentum term is obtained by the parameter \( \beta \), through Polak-Ribiere equation:

Let us consider a generic MLP with \( w^{(t)} \) the weight vector at time \( t \).

1) Initialize all weights \( w^{(0)} \) with small random values (typically normalized in [-1, 1]), set constant \( \varepsilon \) and set \( t = 0 \);
2) Present to the network all training set and calculate \( E(w^{(t)}) \) as the error function for the current weight configuration;
3) If \( t=0 \) or \( t \) is a multiple of \( |w| \)
   a) \( d^{(t)} = -\nabla E^{(t)} \)
   b) else \( d^{(t)} = -\nabla E^{(t)} + \beta^{(t-1)}d^{(t-1)} \) where \( \beta^{(t-1)} \) assumes values from the equation 2 (Polak-Ribiere) with \( g^{(t)} = -\nabla E^{(t)} \);
4) calculate \( w^{(t+1)} = w^{(t)} - ad^{(t)} \) where \( a \) is obtained by the line search;
5) if \( E(w^{(t+1)}) > \varepsilon \)
   a) \( t=t+1 \) and goto 2
   b) else STOP

As discussed, in the conjugate gradients method the parameter \( a \) is obtained by the line search, essentially to avoid the calculation of the Hessian matrix. The algorithm is very sensible to this parameter, and it must paid attention to the line search, growing up the calculation time. In the following we describe a method alternative to the line search, called scaled conjugate gradients.

By referring to the equation (21), the hessian matrix \( H \) in the related equation (22) is multiplied by the vector \( d \).

But this approach is not efficient in presence of not squared error function, because in such cases the Hessian could not be positive. In this case the denominator of the above equation could be negative, producing weight values that increase the error function.

The problem could be circumvented by imposing that the matrix \( H \) is always positive (Nocedal et al. 1999). This can be done by adding to \( H \) a multiple of identity matrix \( \lambda I \), where \( I \) is the identity matrix and \( \lambda > 0 \) is a scaling coefficient. In this case we can replace the equation (22) with:

\[
\alpha^{(t)} = \frac{d^{(t)T} g^{(t)}}{d^{(t)T} H d^{(t)} + \lambda \|d^{(t)}\|^2} \tag{25}
\]

where the index \( (t) \) reflects that the optimum value of \( \lambda \) changes at each iteration.

This technique corresponds to the previously mentioned trust region because the model is effectively valid only in small regions around the current search point. The dimension of such trust region is determined by the parameter \( \lambda \). Now it is required a method to select a proper value for \( \lambda \).
With $\lambda = 0$ the weights move towards a minimum only if the error function is squared and the denominator is positive. If one of these conditions fails then $\lambda$ must be increased.

Let consider the case in which the matrix $H$ is not positive. If we indicate with $d^{(t)}$ the denominator then, being $d^{(t)} < 0$, it must be increased $\lambda$ until $d^{(t)} > 0$. If we denote with $\lambda$ the updated value of $\lambda$, then:

$$
\delta^{(t)} = \delta^{(t)} + (\lambda^{(t)} - \lambda^{(t)})\|d^{(t)}\|^2 = -d^{(t)\top}H^{(t)}d^{(t)\top}
$$

(26)

defined positive if $\lambda = 2\left(\lambda^{(t)} - \frac{\delta^{(t)}}{\|d^{(t)}\|^2}\right)$, that is positive and usable as denominator in the equation (25).

Now it must be considered a squared approximation of the error function in the region located by the current weight vector: if the real function is similar to the approximation, then the value of $\lambda$ will be reduced; otherwise it will be increased. The comparison could be done by evaluating the parameter:

$$
\Delta^{(t)} = \frac{E(w^{(t)} + \alpha^{(t)}d^{(t)}) - E_0(w^{(t)} + \alpha^{(t)}d^{(t)\top}g^{(t)})}{E(w^{(t)}) - E_0(w^{(t)} + \alpha^{(t)}d^{(t)\top}g^{(t)})}
$$

(27)

where $E_0(w)$ is the squared approximation of the error function around $w$, given by:

$$
E_0(w^{(t)} + \alpha^{(t)}d^{(t)}) = E(w^{(t)}) + \alpha^{(t)}d^{(t)\top}g^{(t)} + 1/2 \left[ \alpha^{(t)2}d^{(t)\top}H^{(t)}d^{(t)} \right]
$$

(28)

By simple mathematical manipulations, it is possible to obtain:

$$
\Delta^{(t)} = \frac{2}{\alpha^{(t)}d^{(t)\top}g^{(t)}} \left[ E(w^{(t)}) - E(w^{(t)} + \alpha^{(t)}d^{(t)}) \right]
$$

(29)

From the expression (29) it is possible to infer the modification of $\lambda$ value through the following rule:

$$
\begin{cases}
\lambda^{(t)}/2 & \text{if } \Delta^{(t)} > 0.75 \\
4\alpha^{(t)} & \text{if } \Delta^{(t)} < 0.25 \\
\lambda^{(t)} & \text{else}
\end{cases}
$$

(30)

Note that with $\Delta < 0$ the weights are not updated, because the step could results in an increasing of the error. In this case we increase the value of $\lambda$ and re-evaluate $\Delta$. If we obtain a decreasing of error function for a sufficient $\lambda$, the algorithm moves towards the negative gradient with a small step. The resulting practical algorithm of MLP with BP, based on the scaled conjugate gradients method is the following (ALG-4):

Let us consider a generic MLP with $w^{(t)}$ the weight vector at time $(t)$. 


1) Initialize all weights \( w^{(0)} \) with small random values (typically normalized in \([-1, 1])\), set constant \( \varepsilon \) and set \( t = 0 \);

2) Present to the network all training set and calculate \( E(w^{(t)}) \) as the error function for the current weight configuration;

3) If \( t=0 \) or \( t \) is a multiple of \( |w| \)
   a) then \( d^{(t)} = -\nabla E^{(t)} \)
   b) else \( d^{(t)} = -\nabla E^{(t)} + \beta^{(t-1)} d^{(t-1)} \) where \( \beta^{(t-1)} \) assumes values from the equation 2 (Polak-Ribiere) with \( g^{(t)} = -\nabla E^{(t)} \);

4) if \( t=0 \) then \( \lambda^{(t)} = 0.001 \) else \( \lambda^{(t)} \) assumes the value defined by the rule (a);

5) Calculate \( \delta^{(t)} = d^{(t)T} H^{(t)} d^{(t)} + \lambda^{(t)} \|d^{(t)}\|^2 \)
   a) If \( \delta^{(t)} \leq 0 \) then \( \delta^{(t)} = \delta^{(t)} \), where \( \delta^{(t)} \) assumes values defined by equation (26)

6) Calculate \( \Delta^{(t)} \) with expression (29) Where \( \alpha^{(t)} = d^{(t)T} \nabla E^{(t)}/\delta^{(t)} \) and \( g^{(t)} = -\nabla E^{(t)} \)

7) If \( \Delta^{(t)} > 0 \) then \( w^{(t+1)} = w^{(t)} - \alpha^{(t)} d^{(t)} \), else \( w^{(t+1)} = w^{(t)} \)

8) If \( E(w^{(t+1)}) > \varepsilon \) then \( t=t+1 \) and goto 2, else STOP

By looking at the local squared approximation of the error function, we can obtain an expression of minimum position. It is in fact known that the gradient in every point \( w \) is given by:

\[
g = \nabla E = H(w - w^*)
\]  

(31)

where \( w^* \) corresponds to the minimum of error function which satisfies:

\[
w^* = w - H^{-1} g
\]

(32)

The vector \(-H^{-1} g\) is known as Newton direction and it is the base for a variety of optimization strategies, such as the Quasi Newton Algorithm (QNA) which does not calculate the \( H \) matrix and then its inverse, but an approximation in a series of steps. A famous algorithm having proven good performance even for non-smooth optimizations is known as BFGS, by the names of its inventors (Broyden 1970, Fletcher 1970, Goldfarb 1970, Shanno 1970).

This approach generates a sequence of matrices \( G \) which are further accurate approximations of \( H^{-1} \), by using only information related to the first derivative of error function.

The problem to have only one matrix \( H \) not positive is solved starting from a unitary matrix (positive by definition) and by maintaining further approximations positive as well.
From the Newton formula (32) we note that the weight vectors on steps $t$ and $t+1$ are correlated to the correspondent gradients by the formula:

$$
    w(t+1) - w(t) = -H^{-1}(g(t+1) - g(t))
$$

(33)

known as Quasi Newton Condition. The approximation $G$ is therefore built to satisfy this condition. The formula for $G$ is:

$$
    G(t+1) = G(t) + pp^T/p^T v - (G(t)v)^TG(t)v + (v^TG(t)v)uu^T
$$

(34)

where the vectors are

$$
    p = w(t+1) - w(t), v = g(t+1) - g(t) \text{ and } u = P/p^T v - G(t)v/v^TG(t)v
$$

(35)

By using the identity matrix to initialize the procedure is equivalent to consider, step by step, the direction of the negative gradient while, at each next step, the direction $-Gg$ is for sure a descent direction. The above expression could carry the search out of the interval of validity for the squared approximation. The solution is hence to use the line search to found the minimum of function along the search direction.

By using such system, the weight updating expression (21) would be formulated as follows:

$$
    w(t+1) = w(t) + \alpha(t)G(t)g(t)
$$

(36)

where $\alpha$ is obtained by the line search. The following algorithm shows the MLP trained by QNA method (ALG-5).

Let us consider a generic MLP with $w(t)$ the weight vector at time ($t$).

1) Initialize all weights $w(0)$ with small random values (typically normalized in [-1, 1]), set constant $\varepsilon$, set $t = 0$ and $G(0) = I$;
2) Present to the network all training set and calculate $E(w(t))$ as the error function for the current weight configuration;
3) If $t=0$
   a) then $d(t) = -\nabla E(t)$
   b) else $d(t) = -G(t-1)\nabla E(t-1)$
4) Calculate $w(t+1) = w(t) - \alpha d(t)$ where $\alpha$ is obtained by line search expression (22);
5) Calculate $G(t+1)$ with equation (34);
6) If $E(w(t+1)) > \varepsilon$ then $t=t+1$ and goto 2, else STOP
One of main advantage of QNA, compared with conjugate gradients, is that the line search does not require the calculation of $\alpha$ with an high precision, because it is not a critical parameter. On the contrary, the downside is that it requires a big amount of memory to calculate the matrix $G |w| |w|$ for large $|w|$. One way to reduce the required memory is to replace at each step the matrix $G$ with a unitary matrix. With such replacement and multiplying by $g$ (the current gradient), we obtain:

$$d^{(t+1)} = -g^{(t)} + Ap + Bv$$

Note that if the line search returns exact values, then the above equation produces mutually conjugate directions. $A$ and $B$ are scalar values defined as

$$A = -\left(1 + \frac{\nu^Tv}{p^Tv}\right) \frac{p^Tg^{(t+1)}}{p^Tv} + \frac{\nu^Tv}{p^Tv}$$

and

$$B = \frac{p^Tg^{(t+1)}}{p^Tv}$$

There is also a slightly modified version of the QNA, known as L-QNA or Limited memory QNA (L-BFGS, Nocedal 1980). The algorithm of MLP with L-QNA is the following (alg-6):

Let us consider a generic MLP with $w^{(t)}$ the weight vector at time $t$.

1) Initialize all weights $w^{(0)}$ with small random values (typically normalized in $[-1, 1]$), set constant $\epsilon$, set $t = 0$;
2) Present to the network all training set and calculate $E(w^{(t)})$ as the error function for the current weight configuration;
3) If $t=0$
   a) then $d^{(t)} = -\nabla E^{(t)}$
   b) else $d^{(t)} = -\nabla E^{(t-1)} + Ap + Bv$, where $p = w^{(t+1)} - w^{(t)}$, $\nu = g^{(t+1)} - g^{(t)}$;
4) Calculate $w^{(t+1)} = w^{(t)} - \alpha d^{(t)}$ where $\alpha$ is obtained by line search equation (22);
5) Calculate $A$ and $B$ for the next iteration, as reported in (38);
6) If $E(w^{(t+1)}) > \epsilon$ then $t=t+1$ and goto 2, else STOP

Note that for approximate values of alpha the algorithm works well anyway.

In the ML based on supervised paradigm, there is nowadays a considerable interest in techniques based on margin regularization (Baxter 2000). The concept derives from the assumption that the distance (typically euclidean distance) of an example from the separating hyperplane is the margin of that example and the final goal is to find the best margin of separation (classification) for the submitted data. Significant examples include the Support Vector Machine (SVM), (Cortes et al. 1995), a powerful classification tool that has gained its popularity and interest due to its theoretical merits and successes in real applications (Burges 1998).
SVM were originally defined in order to classify two classes of objects linearly separable. For each class SVM identify the hyperplane that maximize the margin of separation (Figure 12).

In Figure 12 black dots are the first class, white dots the second class, the three lines are three possible margins, it is obvious that H3 (green one) is not suitable for this problem, H1 (blue one) separates the two class but it’s very near to some dots of the two class, H2 (red one) maximize the distance from the dots and is the best separator.

Figure 12. Parameter space separated by hyperplanes through SVM model.

In order to perform the research of the best margin hyperplane that algorithm search for the hyperplanes that bound each class and then find the separation between them (Figure 13).

Figure 13. SVM search for hyperplanes bounding each class and finding the separation.
The two classes (+1 and -1) are bounded by their members, so we have two Hyperplanes, H1 and H2 that are boundary of the two classes. Starting from them, we can define the Hyperplane of separation for the two classes, so the research of the best hyperplane of separation is solved when we found the two boundary hyperplanes, that are defined by some members of the two classes (using the example of Figure 13), two for the class +1 and three for the -1 class, called indeed support vectors.

As shown in Figure 14 a classifier that allows just a linear separation between objects it is not useful. SVMs remap the points in a more dimensional space called feature space where they are linearly separable. This is done with a feature function that is approximated by a weighted sum of kernel functions.

Figure 14. Not linearly separable points in the input space became separable in the feature space.

Hence it is clear that the choice of the kernel function becomes crucial in the implementation of SVM. The LIBSVM package (Chang et al. 2011), one of the widely used library supporting SVM, makes available the following kernels:

- Linear: \( K(x_i, x_j) = x_i^T x_j \);
- Polynomial: \( K(x_i, x_j) = (γ x_i^T x_j + r)^d, γ > 0 \);
- Radial Basis Function (RBF): \( K(x_i, x_j) = \exp(-γ \|x_i - x_j\|^2), γ > 0 \);
- Sigmoid: \( K(x_i, x_j) = \tanh(γ x_i^T x_j + r) \).

While the linear kernel is not able to separate the two classes, the others are all able to separate the classes, but the areas, attributed to the classes in the input space, are different, so it is crucial to validate the result. It is also interesting to observe the different number of support vectors in the different cases, although usually, a high value for the ratio support vector/total pattern is not good and may suggest under- or over-fitting.
Another technique related with the supervised ML is the one including methods called logic based algorithms. Main examples are decision trees or its derivation, rule-based estimators.

We have already introduced decision trees in the previous sections. By dealing with supervised learning, they try to classify patterns by sorting them on the base of their feature values.

However they have some defects. First, the construction of optimal binary decision trees is a well-known NP-complete problem (Hyafil et al. 1976), hence it requires complex heuristics to overcome this limit. Second, for their nature decision trees are univariate, i.e. they split the parameter space on a single feature at each node, revealing inefficient in case of diagonal partitioning requirements. The solution is hence to use alternative multivariate trees, usually obtained by the combination between linear discriminant method and decision trees (Brodley et al. 1995).

As known, it is always possible to derive a rule-based estimator by a decision tree, simply associating one tree path to a separated rule.

One positive aspect of a decision tree is of course its comprehensibility and ease of use. It is intuitive enough to understand that a decision tree corresponds to a hierarchy of tests done by simply making the data flowing through the tree branches and taking output at its leaves (Kotsiantis 2007).

Another important category of supervised ML models and techniques, in some way related with the Darwin’s evolution law, is known as evolutionary (or genetic) algorithms, sometimes also defined as based on genetic programming (Michalewicz et al. 1996).

These names however present some differences. What we can surely assert is that Evolutionary or Genetic models are a category of algorithms that emulate the living organism evolution law. In the Nature all species follow that law in order to adapt their life style to the outdoor environment and to survive. In the ML paradigm this kind of self-adaptive methods try to solve optimization problems.

The relationship is strong between them, because the surviving can be considered an optimization problem as well.

The slight conceptual difference between evolutionary and genetic algorithms is that the formers are problem-dependent, while the latters are very generic. This is also a concept derived from the biologic models, in which all living species are commonly driven by genetic laws, but present specific internal mechanisms to achieve their proper evolution through population generations.

At the base of all evolutionary models there are some general concepts, present in both biological and ML models:

- Individuals as set of genetic features (chromosomes composed by genes);
- Population of individuals evolving in parallel;
- Reproduction of individuals based on re-combination operators and on random mutation;
- Selection of better individuals (solutions of the optimization problem) through fitness operators;
These common features are easily assembled in a form of computational algorithms and are demonstrated very effective by their success in the biological case. There could be also proved that such genetic programming rules are able to solve optimization (either minimization or maximization) problems, statistically converging to the best solution (Mitchell 1998).

In order to be more precise, the question is: what we intend for optimization problem solvable by genetic/evolutionary models?

Well, such problem must include some generic issues:

- Its solution depends on many parameters, to be evolved in strict combination between them;
- It must be always an optimization problem (minimization or its dual, maximization). This is easy to understand by thinking at the final goal of evolution in Nature, i.e. optimization of species adaptation;
- The optimization evaluation function (fitness function in evolutionary jargon) is a complex one, i.e. frequently it has not a closed mathematical expression. For instance, sometimes it is given under the form of a simulation of a real physical system;
- The problem has in principle unknown structure and complexity;
- The problem presents aspects or possible representations that could require a parallel processing. Genetic algorithms are intrinsically parallel, at different levels, from lowest, in which the population members can be created and/or grown independently, to highest, where several independent populations can be grown in parallel or genetic operators can be applied to various population members in an independent way.

In all cases, genetic programming and evolutionary algorithms try always to mimic the evolution in Nature. With such issue in mind, it is easy to deduce that the genetic population corresponds to a set of possible optimization solutions to the given real problem.

The experience on such systems reveals that genetic and evolutionary algorithms are very generic, but if a specific algorithm could be created, it is very likely to be effective in the problem solving.

4.2.2 Methodologies for unsupervised learning

In the unsupervised case, the learning mechanism has something apparently magic. The data analysis model appears a closed system, except for the input data. It never interacts with external environment neither receives any kind of target outputs, but it learns!

Behind this apparently mysterious behavior, we can observe that the unsupervised learning consists in the internal re-organization of the input, based on the retrieved correlations hidden into the data by some quantities of unknown noise contributions. In a certain way, unsupervised learning can be interpreted as a self-adaptive mechanism to find patterns in the data beyond what can be considered pure unstructured noise (Gharamani 2004). Two important classic functional examples of such learning type are clustering and dimensional reduction.
Almost all unsupervised methods may be considered strictly connected with statistical and probabilistic issues. In this sense the final goal is to estimate a model representing a probabilistic distribution of input data, conditioned by the previous sequence of data submitted to the system. Obviously the simplest case is when the ordering of the data sequence is irrelevant, because the variables are considered independent.

Under these conditions we can make use of the classical Bayesian rule (Berger 1985) in which, given a certain model $A$, considered as an unknown probability distribution over a data set $S = \{x_1, \ldots, x_N\}$, the conditioned probability that the model fits the data is:

$$P(A|S) = \frac{P(A)P(S|A)}{P(S)}$$

(39)

And the corresponding model distribution, representing the estimation of the model output on new data can be expressed as:

$$P(x|S) = P(x|A)P(A|S)$$

(40)

An unsupervised model based on such considerations can be applied in many functional DM cases, such as classification, prediction, outlier detection and certainly data parameter space dimensional reduction.

Up to now we are making an important assumption, that is the input data are independent and distributed in an identical way. Although this is a very limiting condition, unreasonable in many real world cases, where current and incoming observed data are correlated with previous ones, it can be applied to time series analysis.

Important unsupervised ML methods to treat such cases are Markov Models (MM) and Hidden Markov Models (HMM), also introduced in section 3.3 of this chapter.

The MM have some strong limitations (Press et al. 2007). First, having fixed a temporal window of data monitoring, the unrealistic disappearance of influence of past input data on the present input patterns outside this window and, second, the direct analysis on incoming raw data instead of try to clean and reduce raw data before to model them (for example in astrophysics raw data sequences, coming from direct sky observations, are preliminarily reduced in order to avoid instrument stamp and atmospheric aberrations).

In the second approach, with HMM, we make use of latent or hidden variables, in which the observations are not directly modeled but assumed as derived by some underlying hidden variable, able to represent the dynamics of the system. HMM belongs to the family of state-space models, but in which the state is considered discrete rather than a continuous random vector.

For what concerns clustering, a not exhaustive taxonomy of techniques can be addressed as follows (Berkhin 2002).

Hierarchical clustering builds a cluster hierarchy or, in other words, a tree of clusters, also known as a dendrogram. Every cluster node contains child clusters; sibling clusters partition the points covered by their common parent. Such an approach allows exploring data on different levels of granularity (Murthagh 2002). Hierarchical clustering methods are categorized into agglomerative (bottom-up) and divisive (top-down).
An agglomerative clustering, considered as the bottom-up hierarchical type, starts with single-point clusters and recursively merges two or more most appropriate clusters (Jain et al. 1988).

On the contrary, the divisive clustering, the top-down hierarchical version, starts with one full-point cluster of all data and recursively splits the most appropriate cluster. The process continues until a stopping criterion (frequently, the requested number k of clusters) is achieved (Kaufman et al. 1990).

![Figure 15. Example of hierarchical clustering.](image)

In Figure 15 three clusters, each with three representatives, are shown before and after the merge and shrinkage.

Two closest representatives are connected by arrow.

Some advantages are:

- Embedded flexibility regarding the level of granularity;
- Ease of handling of any forms of similarity or distance. Consequently, applicability to any attribute types;

But there are also some disadvantages:

- Vagueness of termination criteria;
- The fact that most hierarchical algorithms do not revisit once constructed (intermediate) clusters with the purpose of their improvement;

In case of huge data sets the requirement of keeping such a large matrix in memory is unrealistic. To relax this limitation different devices are used to introduce into the connectivity matrix some sparsity. This can be done by omitting entries smaller than a certain
threshold, by using only a certain subset of data representatives, or by keeping with each point only a certain number of its nearest neighbors. With the (sparsified) connectivity matrix we can associate the connectivity graph \( G = (X, E) \), whose vertices \( X \) are data points, and edges \( E \) and their weights are pairs of points and the corresponding positive matrix entries. This establishes a connection between hierarchical clustering and graph partitioning (Berkhin 2002).

Well, while hierarchical algorithms build clusters gradually (as crystals are grown), partitioning algorithms learn clusters directly. In doing so, they either try to discover clusters by iteratively relocating points between subsets, or try to identify clusters as areas highly populated with data.

One approach to data partitioning is to take a conceptual point of view that identifies the cluster with a certain model whose unknown parameters have to be found. More specifically, probabilistic models assume that the data comes from a mixture of several populations whose distributions and priors we want to find. One clear advantage of probabilistic methods is the interpretability of the constructed clusters. Having concise cluster representation also allows inexpensive computation of intra-clusters measures of fit that give rise to a global objective function (i.e. log-likelihood), (Berkhin 2002).

Depending on how representatives are constructed, iterative optimization partitioning algorithms (such as k-medoids and k-means), concentrate the focus on the fitting into their clusters, trying to build clusters of proper convex shapes.

In the probabilistic approach, data is considered to be a sample independently drawn from a mixture model of several probability distributions. The main assumption is that data points are generated by, first, randomly picking a model \( j \) with probability \( \tau_j, j = 1 \ldots K \) and, second, by drawing a point \( x \) from a corresponding distribution. The area around the mean of each distribution constitutes a natural cluster. So we associate the cluster with the corresponding distribution’s parameters such as mean, variance, etc. Each data point carries not only its (observable) attributes, but also a (hidden) cluster ID (class in pattern recognition).

Probabilistic clustering has some important features (McLachlan et al. 1988):

- It can be modified to handle recodes of complex structure;
- It can be stopped and resumed with consecutive batches of data, since clusters have representation totally different from sets of points;
- At any stage of iterative process the intermediate mixture model can be used to assign cases (on-line property);
- It results in easily interpretable cluster system;

In the partitioning clustering, medoids are objects of a cluster containing data with minimal average dissimilarity to all other in the cluster. In practice, medoids could be considered quite similar to classical centroids (weighted average of cluster points), although medoids are always members of the data set. Medoids are most commonly used on data when a proper centroid cannot be defined. In k-medoids methods a cluster is represented by one of its points (Kaufman et al. 1990). When medoids are selected, clusters are defined as subsets of points close to respective medoids, and the target function is defined as the averaged distance between a point and its medoid.
The k-means algorithm (Hartigan et al. 1979) is by far the most popular clustering tool used in scientific and industrial applications. The name comes from representing each of k clusters by its centroid. Such method works well with numerical attributes, but not with categorical ones. The sum of discrepancies between a point and its centroid expressed through appropriate distance is used as the objective function. For example, the classical objective function, the sum of the squares of errors between the points and the corresponding centroids, is equal to the total intra-cluster variance:

$$E(C) = \sum_{p=1}^{K} \sum_{x_i \notin C_p} \|x_i - c_p\|^2$$  \hspace{1cm} (41)

where C in the (41) is the cluster and c_p is the centroid. The sum of the squares of errors can be rationalized as log-likelihood for normally distributed mixture model and is widely used in statistics (Berkhin 2002). Therefore, k-means algorithm can be derived from general probabilistic framework. Two versions of k-means iterative optimization are known.

The first version (Forgy 1965) consists of two-step major iterations that:

1) assign all the points to their nearest centroids;
2) compute centroids of newly assembled groups;
3) iterations continue until a stopping criterion is achieved.

The second (more used in iterative optimization) version of k-means assigns points with a more detailed analysis of effects on the objective function, if a point is moved from its assigned cluster to a potential new one. If it has a positive effect, the point is relocated and the two centroids are re-computed.

The k-means algorithm is very simple and based on the foundation of analysis of variances. However it suffers of some important issues:

- Only numerical attributes are covered;
- The process fails in case of outliers;
- The result strongly depends on the initial guess of centroids;
- Local optimum is frequently far from the global one;
- Obtained clusters can be unbalanced;
- To choose a good k to use is not easy;
- The algorithm lacks scalability, i.e. not suggested for massive data sets;
- No initialization actually guarantees global minimum.

Density-based methods try to discover dense connected components of data, which are flexible in terms of their shape (Han et al. 2001). Density-based methods are basically less sensitive to outliers and can discover clusters of irregular shapes. They usually work with low-dimensional data of numerical attributes (also related to extended objects) and are closely related to the method of nearest neighbors. A cluster, defined as a connected dense element, grows in any direction driven by point density. Therefore, density-based algorithms are capable of discovering clusters of arbitrary shapes.
Many other clustering techniques are developed, primarily in ML, that either are used traditionally outside the DM community, or do not fit in previously outlined categories. They are basically specialized for KDD (Knowledge Discovery in Databases) and KDT (Knowledge Discovery in Text). There are relationships with unsupervised learning and evolutionary methods (simulated annealing and genetic algorithms). There is however the emerging field of constraint-based clustering (Tung et al. 2001), that is influenced by requirements of real world DM applications.

Another frequently used technique in clustering is referred to the field of Artificial Neural Networks (ANN), in particular the model Self-Organized Map (SOM) (Kohonen 2007). SOM is very popular in many fields (such as vector quantization, image segmentation and clustering) and in this context its analytical description can be omitted, except for two important features: (i) SOM is based on the incremental approach, by processing one-by-one all input patterns; (ii) it allows to map centroids into 2D plane that provides for a quite simple visualization. In addition to SOM, other ANN developments, such as Adaptive Resonance Theory (ART) (Carpenter et al. 1991), or PPS have also relations with clustering.

Applications that derive their data from measurements have an associated amount of noise, which can be viewed as outliers. Alternately, outliers can be viewed as records having abnormal behavior. In general, clustering techniques do not distinguish between the two: neither noise nor abnormalities fit into clusters. Correspondingly, the preferable way to deal with outliers in partitioning the data is to keep one extra set of outliers. There are multiple ways of how descriptive learning handles outliers. If a data preprocessing phase is present, it usually takes care of outliers. For example, this is the case with grid-based methods. They simply rely on input thresholds to eliminate low-populated cells. Other algorithms revisit outliers during the decision tree rebuilds, but in general handle them separately, by producing a partition plus a set of outliers.

What is exactly an outlier? Statistics defines an outlier as a point that does not fit a probability distribution. Classic data analysis utilizes a concept of depth and defines an outlier as a point of low depth. This concept becomes computationally infeasible for a number of dimensions \( D > 3 \) (human limits). Data mining is gradually develops its own definitions.

Consider two positive parameters \( x, y \). A point can be declared an outlier if its neighborhood contains less than 1-fraction of a whole data set \( X \). Rank all the points by their distance to the K-nearest neighbor and define the fraction of points with highest ranks as outliers.

How to describe local outliers? Different subsets of data can have different densities and may be governed by different distributions. A point close to a tight cluster can be a more probable outlier than a point that is further away from a more dispersed cluster. The concept of local outlier factor that specifies a degree of outlier-ness comes to rescue. The definition is simply based on the distance to the k-nearest neighbor.

Statistics provides a coherent framework for learning from data and for reasoning under uncertainty. Many interesting statistical models used for unsupervised learning can be cast as latent variable models and graphical models. These types of models have played an important role in defining unsupervised learning systems for a variety of different kinds of data. Unsupervised learning theory and algorithms still have a long way to go to mimic some of the learning abilities of biological brains.
As the boundaries of unsupervised learning get pushed forward, we will hopefully not only benefit from better learning machines and also improve our understanding of how the brain learns.

### 4.3 How to Evaluate Results

The generalization is also a search technique perfectly matching the learning process. As easy to understand, it is very rare in a ML approach to converge on a single acceptable solution. This arises either because data examples during training are not qualitative exhaustive in terms of hidden information correlation between their feature representations or simply because they are not quantitative sufficient to cover the entire parameter description space.

The generalization technique is hence strictly connected with typical concepts of ML, like the order in which the parameter space is explored, the way that overfitting is avoided and the description language adopted to represent data features.

The overfitting is a problem particularly affecting all ML methods and its bias could be sufficient to get off the learning capability. Some pruning of data set features also in conjunction with statistical or pseudo-statistical techniques can be applied in order to minimize the risk to over-specialize the trained model.

One of simplest techniques is known as confusion matrix (Kohavi et al. 1998).

This is a simple diagnostic instrument useful to estimate the efficiency of the classification model (such as a supervised neural network). It basically consists in a matrix with the values of target vector and the output values produced from the model, respectively, on its rows and columns. In addition it allows calculating the success rate, e.g. the percentage of objects correctly classified from the model, the number of “bit fault” that the model badly classifies and the percentage of correctly classified objects for each class.

In the matrix the element corresponding to row i and column j is the absolute number or case percentage of “true” class i classified in the class j. On the main diagonal the correct classified cases are reported. The others are classification errors.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Total:</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>60</td>
<td>14</td>
<td>13</td>
<td>87  69.0%</td>
</tr>
<tr>
<td>B</td>
<td>15</td>
<td>34</td>
<td>11</td>
<td>60  56.7%</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>0</td>
<td>42</td>
<td>53  79.2%</td>
</tr>
<tr>
<td>Total</td>
<td>86</td>
<td>48</td>
<td>66</td>
<td>200 68.0%</td>
</tr>
</tbody>
</table>

Figure 16. An example of confusion matrix for a 3-class classification problem.
In the example of Figure 16 we have a 3-class classification problem results. The original training set consists of 200 patterns.

In the class A there are 87 cases: 60 correctly classified as A; 27 wrongly classified, of which 14 as B and 13 as C.

So far, for the class A the accuracy is 60 / 87 = 69.0%. For the class B the accuracy is 34 / 60 = 56.7% and for class C 42 / 53 = 79.2%.

The whole accuracy is hence: (60 + 34 + 42) / 200 = 136 / 200 = 68.0%. The errors (bad classification) are then 32%, e.g. 64 cases on 200 patterns.

The classification result depends not only on the percentages, but also on the relevance of single kinds of errors. In the example, if class C is the most important to be classified, the final result of the classification can be considered successful.

The cross validation is a statistical method useful to validate a predictive classification model (Kohavi 1995). Having a data sample this is divided into subsets, some of them used for the training phase (training set), while the others employed to compare the model prediction capability (validation set). By varying the value of K (different splitting of the data sets) it is possible to evaluate the prediction accuracy of the trained model, Figure 17.

The K-fold cross-validation divides the whole data set into K subsets, each of them is alternately excluded from the validation set.

There is also a special case, named leave-one-out cross validation, where alternately only one pattern is excluded at each validation run.

In practice all data are used for the training and test phases in an independent way. In this case we obtain K classifiers (2 ≤ K ≤ n) whose outputs can be used to obtain a mean evaluation. The downside of this method is that it could result very expensive in terms of computing time in case of massive data sets.

![Figure 17. Two examples of K-fold cross validation.](image)

In feed-forward neural networks, like the famous classical Multi Layer Perceptron, the term feed-forward is used to identify basic behavior of such neural models, in which the impulse is propagated always in the same direction, e.g. from neuron input layer towards output layer, through one or more hidden layers (the network brain), by combining weighted sum of weights associated to all neurons (except the input layer).

As easy to understand, the neurons are organized in layers, with proper own role. The input signal, simply propagated throughout the neurons of the input layer, is used to stimulate
next hidden and output neuron layers. The output of each neuron is obtained by means of an activation function, applied to the weighted sum of its inputs. Different shape of this activation function can be applied, from the simplest linear one up to sigmoid, arctan or tanh (or a customized function ad hoc for the specific application). The number of hidden layers represents the degree of the complexity achieved for the energy solution space in which the network output moves looking for the best solution. As an example, in a typical classification problem, the number of hidden layers indicates the number of hyper-planes used to split the parameter space (i.e. number of possible classes) in order to classify each input pattern.

There is also a special type of activation function, called softmax, (Sutton et al. 1998).

As known the activation function can be either linear or non-linear, depending on whether the network must learn a regression problem or should perform a classification.

Activation functions, for the hidden units, introduce the non-linearity into the network. Without non-linearity, the hidden units would not render the NN more powerful than just the perceptrons with only input and output units (the linear function of linear functions is again a linear function). In other words, it is the non-linearity (i.e., the capability to represent nonlinear functions) that makes multilayer networks so powerful.

For the hidden units, sigmoid activation functions (for binary problems), see equation (42b), or softmax (for multi class problem), are usually better to use instead of the threshold activation functions, see equation (42a).

\[
\begin{align*}
\begin{array}{ll}
0 & \text{if } a < 0 \\
1 & \text{else} \\
\end{array}
\end{align*}
\]

\[
f(x) = \begin{cases} 
0 & \text{if } a < 0 \\
1 & \text{else} \\
\frac{1}{1+e^{-x}} & \text{otherwise}
\end{cases}
\]  

(42)

Networks with threshold units are difficult to train, because the error function is stepwise constant, hence the gradient either does not exist or is zero, thus making it impossible to use back propagation (a powerful and computationally efficient algorithm for finding the derivatives of an error function with respect to the weights and biases in the network) or the more efficient gradient-based training methods.

With sigmoid units, a small change in the weights will usually produce a large change in the outputs, which makes it possible to tell whether that change in the weights is good or useless. With threshold units, a small change in the weights will often produce no change in the outputs. For the output units, activation functions suited to the distribution of the target values are:

- For binary (0/1) targets, the logistic sigmoid function is an excellent choice;
- For categorical targets using 1-of-C coding, the softmax activation function is the natural extension of the logistic function;
- For continuous-valued targets with a bounded range, the logistic and hyperbolic tangent functions can be used, where you either scale the outputs to the range of the targets or scale the targets to the range of the output activation function ("scaling" means multiplying by and adding appropriate constants);
- If the target values are positive but have no known upper bound, you can use an exponential output activation function, but you must beware of overflow;
For continuous-valued targets with no bounds, use the identity or "linear" activation function (which amounts to no activation function) unless you have a very good reason to do otherwise.

There are certain natural associations between output activation functions and various noise distributions. The output activation function is the inverse of what statisticians call the "link function", which usually provides the relationship between the linear predictor and the mean of the distribution function (such as binomial, Poisson, normal distributions), (Ruckstuhl et al. 1999).

As briefly mentioned, in a supervised ML scheme, training is done by means of a mechanism in which the model output is compared with desired (known from BoK) target output for each input pattern. Depending on the metric function used for the comparison, different error evaluation metrics can be applied, depending also on the problem complexity to be solved. The most common metric is the Mean Square Error (MSE) of the difference between model and target outputs (Lehmann et al. 1998).

Supervised neural networks that use an MSE cost function can use formal statistical methods to determine the confidence of the trained model. The MSE on a validation set can be used as an estimate for variance. This value can then be used to calculate the confidence interval of the output of the network, assuming a normal distribution. A confidence analysis made this way is statistically valid as long as the output probability distribution stays the same and the network is not modified.

By assigning a softmax activation function on the output layer of the neural network (or a softmax component in a component-based neural network) for categorical target variables, the outputs can be interpreted as posterior probabilities. This is very useful in classification as it gives a certainty measure on classifications.

In order to ensure that the outputs can be interpreted as posterior probabilities, they must be comprised between zero and one, and their sum must be equal to one. This constraint also ensures that the distribution is correctly normalized. In practice this is, for multi-class problems, achieved by using a softmax activation function in the output layer. The purpose of the softmax activation function is to enforce these constraints on the outputs. Let the network input to each output unit be $q_i$, $i = 1,...,c$, where $c$ is the number of categories. Then the softmax output $p_i$ is:

$$ p_i = \frac{e^{q_i}}{\sum_{j=1}^{c} e^{q_j}} \quad (43) $$

Statisticians usually call softmax a "multiple logistic" function, although it is also known as normalized exponential function. It reduces to the simple logistic function when there are only two categories. Suppose you choose to set $q_2 = 0$ in the (43):

$$ p_1 = \frac{e^{q_1}}{\sum_{j=1}^{c} e^{q_j}} = \frac{e^{q_1}}{e^{q_1} - e^{-q_1}} = \frac{1}{1 + e^{-q_1}} \quad (44) $$

The term softmax is used because this activation function represents a smooth version of the winner-takes-all activation model in which the unit with the largest input has output +1 while all other units have output 0.
Another important evaluation technique, exploiting the use of outputs as posterior probabilities is the \textit{cross entropy} (Rubinstein et al. 2004). Let introduce it by some considerations.

As already discussed, learning in the neural networks is based on the definition of a suitable error function, which is then minimized with respect to the weights and biases in the network.

For regression problems the basic goal is to model the conditional distribution of the output variables, conditioned on the input variables. This motivates the use of a sum-of-squares error function. But for classification problems the sum-of-squares error function is not the most appropriate choice. In the case of a 1-of-C coding scheme, the target values sum to unity for each pattern and so the network outputs will also always sum to unity. However, there is no guarantee that they will lie in the range [0,1].

In fact, the outputs of the network trained by minimizing a sum-of-squares error function approximate the posterior probabilities of class membership, conditioned on the input vector, using the maximum likelihood principle by assuming that the target data was generated from a smooth deterministic function with added Gaussian noise. For classification problems, however, the targets are binary variables and hence far from having a Gaussian distribution, so their description cannot be given by using Gaussian noise model.

Therefore a more appropriate choice of error function is needed.

Let us now consider problems involving two classes. One approach to such problems would be to use a network with two output units, one for each class. First let’s discuss an alternative approach in which we consider a network with a single output $y$. We would like the value of $y$ to represent the posterior probability $P(C_1|x)$ for class $C_1$. The posterior probability of class $C_2$ will then be given by $P(C_2|x) = 1 - y$.

This can be achieved if we consider a target coding scheme for which $t = 1$ if the input vector belongs to class $C_1$ and $t = 0$ if it belongs to class $C_2$. We can combine these into a single expression, so that the probability of observing either target value is $(t|x) = y^t(1 - y)^{1-t}$.

This equation is the binomial distribution, known as \textit{Bernoulli} distribution in special cases. With this interpretation of the output unit activations, the likelihood of observing the training data set, assuming that the data points are drawn independently from this distribution, is then given by:

\[
\prod_n (y^n)^t(1 - y^n)^{1-t^n} 
\]

By minimizing the negative logarithm of the likelihood, we get the cross-entropy error function (Baum and Wilczek, 1988), in the form:

\[
E = -\sum [t_n \ln y^n + (1 - t_n) \ln(1 - y^n)] \tag{46}
\]

Let’s consider some elementary properties of this error function. Differentiating this error function with respect to $y^n$ we obtain:

\[
\frac{\partial E}{\partial y^n} = \frac{(y^n - t^n)}{y^n(1-y^n)} \tag{47}
\]
The absolute minimum of the error function occurs when $y^n = t^n, \forall n$.

The network has one output whose value is to be interpreted as a probability, so it is appropriate to consider the logistic sigmoid activation function which has the property:

$$g'(x) = g(x)(1 - g(x))$$  \hspace{1cm} (48)

Combining equations (47) and (48) it can be seen that the derivative of the error with respect to $x$ takes a simple form:

$$\delta^n \equiv \frac{\delta E}{\delta x^n} = y^n - t^n$$  \hspace{1cm} (49)

This equation gives the error quantity which is back propagated through the network in order to compute the derivatives of the error function with respect to the network weights. The same equation form can be obtained for the sum-of-squares error function and linear output units. This shows that there is a natural paring of error function and output unit activation function.

From the previous equations the value of the cross entropy error function at its minimum is given by

$$E_{\text{min}} = -\sum t^n lnt^n + (1 - t^n)ln(1 - t^n)$$  \hspace{1cm} (50)

The equation (50) becomes zero for 1-of-C coding scheme. However, when $t^n$ is a continuous variable in the range $(0,1)$, representing the probability of the input vector $x^n$ belonging to class $C$, the error function is also the correct one to use. In this case the minimum value (50) of the error does not become 0. In this case it is appropriate by subtracting this value from the original error function to get a modified error function of the form:

$$E = -\sum \left( t^n lny^n + (1 - t^n)ln \frac{(1 - y^n)}{(1 - t^n)} \right)$$  \hspace{1cm} (51)

But before moving to cross-entropy for multiple classes let us describe more in detail its properties. Assume the network output for a particular pattern $n$, written in the form $y^n = t^n + e^n$. Then the cross-entropy error function (51) can be transformed to the form:

$$E = -\sum \left( t^n lne^n + (1 - t^n)ln \frac{(1 - e^n)}{1 - t^n} \right)$$  \hspace{1cm} (52)

So that the error function depends on the relative errors of the network outputs. Knowing that the sum of squares error function depends on the squares of the absolute errors, we can make comparisons. Minimization of the cross-entropy error function will tend to result in similar relative errors on both small and large target values. By contrast, the sum-of-squares error function tends to give similar absolute errors for each pattern, and will give large relative errors for small output values. This result suggests the better functionality of the cross-entropy error function over the sum-of-squares error function at estimating small
probabilities. Another advantage over the sum-of-squares error function, is that the cross-entropy error function gives much stronger weight to smaller errors.

A particular case is the classification problem involving mutually exclusive classes, i.e. where the number of classes is greater than two. For this problem we should seek the form which the error function should take. The network now has one output \( y^k \) for each class, and target data which has a 1-of-c coding scheme, so that we have \( t^k_n = b_{kl} \) for a pattern \( n \) from class \( C_l \). The probability of observing the set of target values \( t^k_n = b_{kl} \), given an input vector \( x^n \) is just \( P(C_l|x) = y_l \).

Therefore the conditional distribution for this pattern can be written as:

\[
P(t^n|x^n) = \prod_{k=1}^{c} (y^n_k)^{t^n_k}
\]  

As before, starting from the likelihood function, by taking the negative logarithm, we obtain an error function of the form:

\[
E = - \sum_n \sum_{k=1}^{c} t^n_k \ln y^n_k
\]  

For 1-of-c coding scheme the minimum value of the error function (54) equals 0. But the error function is still valid when \( t^n_k \) is a continuous variable in the range (0,1) representing the probability that \( x^n \) belongs to \( C_l \).

To get the proper target variable, the softmax activation function is used. Hence, for the cross-entropy error function for multiple classes, to make equation (54) efficient, the softmax activation function must be used.

By evaluating the derivatives of the softmax error function, by considering all inputs to all output units, (for pattern \( n \)) it can be obtained:

\[
\frac{\partial E^n}{\partial x_k} = y_k - t_k
\]  

which is the same result as found for the two-class cross-entropy error (with a logistic activation function). The same result is valid for the sum-of-squares error (with a linear activation function). This can be considered as an additional proof that there is a natural pairing of error function and activation function.

Clearly, for every activation function we get a proper error function, and as shown for the softmax activation function we must use the cross-entropy error function. It is obvious that by using non proper pairs of activation and error function the network would not perform as we would like to, giving results without sense.

5. **The Future is Now: Machine Learning and X-Informatics**

As always, innovative software-based tools proliferate as they become increasingly essential for doing science. An inevitable consequence will be the beneficial combination of shared and differentiating software. Thinking to an “open science” scenario, differentiating software can be seen as a preliminary step in a contribution to the sciences, eventually
Massimo Brescia

transferred to shared status. The huge requirement repeatability of scientific experiments equally pushes towards the standardization of instruments, parameter setup, estimation and data collection, collation and treatment. This also means (i) the obvious sharing of these tools by and in the science community, (ii) their re-usability by people who want to replicate or improve experiments, and (iii) their reliability, ensuring repeatability over long time periods.

This is what is started by successful startup of several scientific communities (like the mentioned VO in Astronomy), that design, build and share effectively their tools at the level of components, frameworks/architecture, applications and services with the broader community.

Acknowledging this process could of course include the competitive feature of the science, but it can also carry important benefits to the communities. A related challenge is educating computer scientists and software engineers to get such an approach off the ground (Hey et al. 2009).

Moreover, the creation of powerful and reliable computing architectures and component frameworks requires a much deeper integration and interoperability between computer science and the various science communities. It is clear that they need to work together far more closely to successfully build usable, robust, reliable and scalable tools for doing e-science. This is already happening in some areas and in some countries, but the scale of the integration required is not going to happen casually. We want to point out that science communities at large take the initiative and introduce schemes and instruments that enable far greater co-operation and community building between all these elements.

Nowadays, we started to often hear about the term "X-informatics", or the name of the research fields with the suffix “informatics”, such as bio-informatics or astro-informatics. These fields share the same strategy: they all aim at acquiring new viewpoints and models by applying informatics-based approaches to existing fields such as biology or astronomy. They also share the same methodology: the generation of huge amount of data with the help of advanced sensor and observation technologies, and the fast search and knowledge discovery from massive databases, perfectly compliant with the issues coming from the fourth paradigm of science (Hey et al. 2009).

It is recently emerging world-wide (cf. the US Science Council community and the recent meeting session on AstroInformatics at the 215th American Astronomical Society (AAS) and the Conference AstroInformatics 2010 and 2011) the need for all components (both hardware and software) of the X-informatics infrastructure to be integrated or, at least, made fully interoperable. In other words, the various infrastructure components (data, computational resources and paradigms, software environments, applications), should interact seamlessly exchanging information, and be based on a strong underlying network component. And such requirement and recommendation is transversal in respect of single X-informatics fields of research.

This emerging e-science reflects a very important and, in a certain way revolutionary, change in the engineering approach to perform analysis and exploration of data, especially in scientific disciplines, like Astrophysics, where it is crucial to develop efficient methods to explore MDS.

By analyzing the data exploration from a human perspective, we can observe that:
due to new instruments and new diagnostic tools, the information volume grows exponentially

- Most data will never be seen by humans!

- Information complexity is also increasing greatly

  - Most knowledge hidden behind data complexity is lost

- Most (all) empirical relationships known depend on 3 parameters (simple universe or rather human bias?)

  - Most data (and data constructs) cannot be comprehended by humans directly!

These considerations are at the base of the need for KDD, data understanding technologies, hyper dimensional visualization, Artificial Intelligence / Machine-assisted discovery etc.

E-science communities only recently started to face the deluge of data produced by new generation of scientific instruments and by numerical simulations (widely used to model the physical processes and compare them with measured ones). Now data is commonly organized in scientific repositories. Data providers have implemented Web access to their repositories, and http links between them. This data network, which consists of huge volumes of highly distributed, heterogeneous data, opened up many new research possibilities and greatly improved the efficiency of doing science. But it also posed new problems on the cross-correlation capabilities and mining techniques on these MDS to improve scientific results. The most important advance we expect is a dramatic need of the ease in using distributed e-Infrastructures for the e-science communities. We pursue a scenario where users sit down at their desks and, through a few mouse clicks, select and activate the most suitable scientific gateway for their specific applications or gain access to detailed documentation or tutorials. We call scientific gateway an e-Infrastructure which is able to offer remote access and navigation on distributed data repositories together with web services and applications able to explore, analyze and mine data. It doesn’t require any software installation or execution on user local PC, it permits asynchronous connection and launch of jobs and embeds to the user any computing infrastructure configuration or management.

In this way the scientific communities will expand their use of the e-Infrastructure and benefit from a fundamental tool to undertake research, develop collaborations, and increase their scientific productivity and the quality of research outputs. Only if this scenario becomes reality, the barrier currently placed between the community of users and technology will disappear, especially machine learning that requires by its nature a strong expertise and efficient computing tools.

The KDD discipline is a burgeoning new technology for mining knowledge from data, a methodology that a lot of heterogeneous communities are starting to take seriously. Strictly speaking KDD is about algorithms for inferring knowledge from data and ways of validating it. So far, the main challenge is applications. Wherever there is data, information can be
gleaned from it. Whenever there is too much data or, more generally, a representation in more than 3 dimensions (limit to infer it by human brain), the mechanism of learning will have to be automatic.

When a data set is too large for a particular algorithm to be applied, there are basically three ways to make learning feasible. The first one is trivial: instead of applying the scheme to the full data set, use just a small subset of available data for training. Obviously, in this case information is easy to be lost and the loss is negligible in terms of correlation discovery between data. The second method consists of parallelization techniques. But the problem is to be able to derive a parallelized version of the learning algorithm. Sometimes it results feasible due to the intrinsic natural essence of the learning rule (such as genetic algorithms). However, parallelization is only a partial remedy because with a fixed number of available CPUs, the algorithm's asymptotic time complexity cannot be improved. Background knowledge (the third method) can make it possible to reduce the amount of data that needs to be processed by a learning rule. In some cases most of the attributes in a huge data set might turn out to be irrelevant when background knowledge is taken into account. But in many exploration cases, especially related to DM against data analysis problems, the background knowledge simply does not exist, or could infer a sort of wrong biased knowledge in the discovery process.

So far, general requirements focus on the need to integrate massive databases in a computing grid and create proper science gateways to underpin the use of the infrastructure and to bridge heterogeneous e-Infrastructures. Moreover, scientists envisage some advanced functionalities in the use of new computing architectures (such as shared memory systems or GPU computing) and therefore the ability to gridify applications that require executing many independent tasks of parallel jobs. Behind these considerations there is the crucial requirement to harmonize all recent efforts spent in the fields of ICT and ML.

Figure 18. Layout of X-Informatics (KDD) process.

It arises indeed a new approach to the way of performing data investigation: Distributed Data Mining (i.e. mining of distributed DM) versus Distributed Data Mining (i.e. distributed mining of distributed data), (Borne 2009).

By looking at the desired evolution in the data archiving and exploration warehouse, we can fix following step-by-step improvement process.
The simplest and raw method is based on the presence of many observing/monitoring stations or experiments, each one with a specific individual data archiving interface system (Figure 19).

This architecture has main downsides from the scientist point of view. The data investigation requires a long download time for the information retrieval from all data centers and a very difficult pre-processing phase in order to make uniform all data collected by different sources and compliant with own application or algorithm. All time wasted for the single scientist to make science with data!

One of the first benefits introduced by the KDD paradigm can be the standardization of data representation and protocols for information retrieval (Figure 20).

By this way, the problem to make uniform data is overcome. But the time spent to download massive data sets remains still an open issue.
Figure 21. Second evolution type of data warehouse organization.

Figure 21 shows the second step of the evolution in data warehouse standardization. The introduction of meta-language and meta-data like XML, gained by the progress in Computer Science and ICT, makes the possibility to avoid the time- and memory-consuming download of original data. By interfacing databases with light files containing meta-information, uniformed under the simple standard constraints imposed by XML, representing all information related to the original data files, permitted the interoperability between data archives and an easy and quick way to import data. This of course should be accompanied by the introduction of specific navigation and data access services, mostly based on the web service paradigm.

It remained only one, but not secondary, problem. The interoperability between data could be improved if also the applications and algorithms (for instance based on ML paradigm) to mining data become interoperable and uniformed (Figure 22).
New Trends in E-Science: Machine Learning and Knowledge Discovery in Databases

This is the last desired step in the evolution of e-science, particularly important if we consider the parallel growth of data worldwide collected and the need to perform multi-discipline data exploration, by using thousands of processors. The advancement of ICT and the Web 2.0 technology give the possibility to remotely interact with data archives and DM applications by a simple web browser. De facto, with web applications a remote user does not require to install program clients on his desktop, having the possibility to collect, retrieve, visualize and organize data, configure and execute mining applications with a simple web browser and in an asynchronous way. The added value of such e-science advantage is also that the user does not need computing and storage power facilities availability in his office. He can transparently make his DM with ML experiments by exploiting computing networks and archives located worldwide, requiring only a light laptop (or a personal smartphone or tablet-PC). This is the most gain of e-science!

6. THE NEW GENERATION OF DATA ANALYSIS INFRASTRUCTURES

As discussed in the previous chapter, the broad development and dissemination of Web 2.0 technologies have dramatically changed the perspective of how to make DM and analysis experiments, either from the user access and engineering design points of view. The huge dimensions of data, the recently discovered relevance of multi-disciplinary and cross correlation in modern DM and the advanced complexity of ML methodologies have rapidly modified the requirements for applications and services to be made available to virtual communities of users, in industry, social networks, finance as well in all scientific and academic environments.

Such new requirements are indeed related with hardware computing infrastructures together with software tools, applications and services:

- Computing time efficiency, high storage systems, distributed multi-core processing farms and parallel programming: modern computing architectures, such as cloud, grid, HPC (High Performance Computing), GPU (Graphics Processing Unit) cannot be hosted by single user offices and require to concentrate computing facilities in data centres, accessible to worldwide user communities.
- The access to computing facilities must be as much as possible user-friendly, by embedding to the end user, potentially not technically skilled, all internal mechanisms and setup procedures;
- The remote access to data centres and analysis services must be asynchronous, in order to avoid the need for the user to maintain open the connection sockets for a potentially huge amount of time. It is in fact well known that massive data processing experiments, based on ML, are time-consuming;
- Data mining with ML methods are in principle structured as workflows (for example pipelines of data reduction and analysis in astrophysics), made by an ordered sequence of conceptual steps (data preparation, training, validation, test), one depending on each other. Any of the analysis and mining services must offer a complete set of tools to perform all these operational steps. In particular they
should be able to offer scripting tools, to make custom setup and execution of different scripts, composed by various ML experiments automatically sequenced and ordered;

- Machine learning methods require a strong experience and both scientific and technological knowledge on their internal setup and learning mechanisms. In principle the end user should have an expert of ML techniques available during all phases of the experiments. So far, in the absence of human experts, the remote applications and services must guarantee the possibility to guide users through simple and intuitive interfaces to all conceptual steps of the experiments;

- Multi-disciplinary data centres must be interoperable, i.e. all archives require an high level of data format and access standardization in order to perform join and cross correlation activities in a fast and efficient way, without constraining a pre-treatment of data to obtain their uniformity and homogeneity;

- Massive data sets are often composed by GB or TB of data. It is unthinkable to make repetitive data moving operation on the network in a fast and efficient way. The use of metadata could represent a good compromise for any user who intends to manipulate data and to submit them to ML services in a remote way. So far, the available application frameworks must make available such mechanisms;

- The human machine as well as graphical user interfaces of remote data analysis systems should guarantee the interoperability between their ML engines, by making use of standards for algorithm description, setup and engineering. This could permit an efficient sharing mechanism between different data warehouses, avoiding replications of tools and confusion in the end users;

- In many scientific communities, users are accustomed to use their own algorithms and tools, sometimes specifically created to solve limited problems. Such tools were not originally designed by following programming standards or to be executed and portable on cross platforms. Modern ML service infrastructures should make available automatic plug-in features able to give to users the possibility to integrate and execute their own scripts or algorithms on the remote computing platforms.

The above list of requirements is not exhaustive, but is sufficient to clear the important aspects related with what the modern virtual communities should expect to take full advantages of available Information and Communication Technologies (ICT) in the era of e-science.

Data mining with ML intrinsically contains so high levels of complexity and a wide degree of multi-disciplinary generalization to be a good candidate as benchmark for the new ICT solutions, being able to fully exploit its revolutionary features and products.

Currently there are a lot of applications and services, related to DM with ML, available in the world. Some of them were originally conceived for general purpose, others specialized to treat problems for a specific discipline or science/social community. In this wide jungle of tools, those briefly described in the following were selected as significant examples of applications, in some way related with the above requirements.
As general guideline, we refer to all examples based on the concept of cross-platform software, indeed able to function on more than one computer architecture or operating system. This is probably the most important requirement for an efficient and really useful DM tool, voluntarily omitted in the above list because is taken for granted in the entire discussion.

WEKA (Waikato Environment for Knowledge Analysis, http://www.cs.waikato.ac.nz/ml/weka/), for example, is a desktop application to be downloaded and installed on user machine, whose main goal is a framework, based on Java, hosting ML models and tools for general-purpose DM. It includes most standard ML models, especially for classification problems, that can be plugged in the program (these include Learning Vector Quantization, Self-organizing maps, Feed-forward ANNs among others). The user interface, quite user-friendly, allows users to preprocess, classify, associate, cluster and visualize both input and processed data. It has also hosts widgets to navigate and connect workflows between them. Being installed on the user side it exploit the local hardware resources and it could result limiting in scalability and efficiency. On the contrary it has the main advantage to be able to process results directly returned by database queries, as including SQL databases using Java database connectivity. There is also a useful documentation to drive users through the available ML models (Witten and Frank 2000).

KNIME (Konstanz Information Miner, http://www.knime.org/) is a user-friendly and intuitive open-source platform for data integration, analysis and exploration. The tool is widely used by professionals in a wide range of disciplines. It is not specifically conceived for DM, but hosts many statistical and ML models for data analysis at large. There are different cross-platform versions, from desktop to client/server applications and it is based on a user-friendly GUI in which data analysis workflows can be designed and configured by user through graphical drag & drop mechanisms. There available nodes for I/O, ML, statistics, modeling, visualization, pre and post processing, linkable as graphical pipelines. It also integrates all analysis modules of WEKA DM environment and additional plugins allowing execution of R-scripts and offering access to a vast library of statistical routines.

KNIME is also easily extensible with custom nodes and types to automatically treat highly domain-specific data, useful to extend its proper employment in communities where data types are very specific. The desktop application version performances and scalability are obviously limited by the user local computing resources.

DAMEWARE (Data Mining & Exploration Web Application REsource, http://dame.dsf.unina.it/) is a rich internet application, one of the main products made available through the DAME international Program Collaboration. It provides a web browser based front-end, able to configure DM experiments on massive data sets and to execute them on a distributed computing infrastructure (cloud/grid hybrid platform), hidden to the users. DAMEWARE offers the possibility to access different DM functionalities (supervised classification, regression and clustering) implemented with different ML methods (among them, traditional MLPs, Support Vector Machines, Genetic Algorithms). Originally specialized and scientifically validated on DM in Astrophysics, it can be used in a wide range of real problems and disciplines, by offering a completely transparent architecture, a user friendly interface, standards to ensure the long-term interoperability of data and the possibility to seamlessly access a distributed computing infrastructure. It also makes available an automatic tool to create user customizable workflows and models, plugged in the application engine. In order to effectively deal with MDS, DAME offers asynchronous access to the
infrastructure tools, thus allowing the running of activity jobs and processes outside the scope of any particular web application operation and without depending on the user connection status. The user, via a simple web browser, can access the application resources and can keep track of his jobs by recovering related information (partial/complete results) without having the need to maintain open the communication socket. Furthermore its GUI has widgets that make it interoperable with KNIME processing engine. The service is currently available as a beta release and under completion and test for some crucial aspects (experiment execution scheduling system, integration of new ML models, plug-in procedure). The main limit is the inability to setup and execute custom scripting procedures, constraining to manually setup and execute multiple ML experiments in a sequential way. The DAME Program website hosts other web-based services and resources for the astronomical virtual community, together with many documents useful especially for novices of ML techniques and DM methodology.

Orange (http://orange.biolab.si/) is an open source data visualization and analysis framework for novices and experts. It includes resources for DM and data analytics through visual programming or Python scripting, ML models, extensions for bioinformatics and text mining. In particular most standard DM models have been implemented like classification Trees, KNN, SVM, Bayesian models, regression tools and the library of models is still growing. The “Orange Canvas” user interface is well intuitive. All experiments are performed as schemas constructed using black-box type widgets that can be individually configured. The main limit is the lack of scalable DM routines, making it slow to feasibly run on massive data sets. There is good documentation available for the usage as well as the scripting procedures.

RapidMiner (http://rapid-i.com/content/view/181/196/lang.en/) is a resource available as a stand-alone application for data analysis, within the powerful enterprise server setup RapidAnalytics, and as a DM engine which can be integrated into own products. Beside RapidMiner and built on top of this data transformation and analysis solution, several other Rapid-I products are available. Most standard models have been implemented. There are plugins available to interface with Weka, R and other major DM platforms, providing the opportunity to integrate all operations from those programs. The setup of experiments results quite easy and intuitive, being available a large set of tutorials.

7. Conclusion

The development of an e-infrastructure for scientific data management is therefore essential. This poses major challenges for both ML and DM researches. We attempted to describe some of them here. A major issue is the distribution of data. Database technology has recognized for a long time that it is expensive or impossible to move large quantities of data. Instead one moves the code (software executing a program) to the data, and this is the core of distributed query optimization. However, we need to extend distributed query optimization, which works for the simple operations of relational algebra, to work for more general operations that support scientific programming and to include, for example, spatial queries, string searches, etc. Known database techniques, such as parallel processing, self-adaptive models, set-oriented data access and intelligent indexing need to be extended. They indeed have to support as much scientific data types as possible and to maximize their interoperability. This distributed infrastructure will have to support stream processing and
New Trends in E-Science: Machine Learning and Knowledge Discovery in Databases

advanced DM/ML techniques. We expect novel DM methodologies and novel ML techniques to be promising approaches to cope with growing data, especially where analytical and deterministic approaches have failed to yield a satisfying model to explain phenomena and where traditional techniques have failed to bring back the knowledge out of the data. In the long term, an active ML model is foreseen which autonomously requests data sources, like experiments, and leads to autonomous experimentation. But this is just the base technology that has to be developed, because it must be supported by a distributed computing environment in which it is easy for heterogeneous scientists to exploit the infrastructure. So far, one of the first requirements is to massively employ the semantics of data. This involves a full understanding and categorization of the abstract level of data, i.e. metadata. This abstraction of data is primarily used by tools that perform massive data integration and exploit web 2.0 technologies able, for instance, to transform or derive the data. Furthermore, the environment should make feasible standard tasks such as querying, scaling, programming, mining or compose tasks (pipelines) and it should make it possible for scientists to generate their own computing tasks, rather than being reliant on database experts.

A crucial improvement in the ability to mine data, in order to gain knowledge about natural phenomena will be indeed enabled by (i) new standards to collect and manage massive amounts of data from observations/simulations, (ii) integration of DM and ML tools directly into the data warehouses, (iii) improved forms of interaction between scientists and DM experts that support analysis, visualization and interactivity, (iv) formalization of new hybrid scientific disciplines, like X-informatics, (v) the proliferation of virtual communities, based on the worldwide interoperability on data and applications, and (vi) a deep transformation of scientific dissemination.

This new conceptual and technological paradigm have an incredible potential to accelerate a new era of science-based innovation and a consequent new trend of economic growth based on technological innovation in many crucial environments, like health, medical, energy, environmental management, computing and engineering sectors.

Finally, the relevance of the impact of what is occurring is such that we simply cannot afford to ignore or dismiss it. We need to re-act upon it. The developments occurring in this decades, together with what we expect to happen as a consequence of the emerging e-science, as the fourth leg of science, already started to introduce new and unexpected global challenges and opportunities and we are convinced that they will condition our future.

8. REFERENCES


New Trends in E-Science: Machine Learning and Knowledge Discovery in Databases


New Trends in E-Science: Machine Learning and Knowledge Discovery in Databases


