CHAPTER 1

Introduction to Data Mining

We introduce in this chapter the main concepts of data mining. This scientific field, together with Cloud computing, discussed in Chapter 2, is a basic pillar on which the contents of this book are built. Section 1.1 explores the main notions and principles of data mining introducing readers to this scientific field and giving them the needed information on sequential data mining techniques and algorithms that will be used in other sections and chapters of this book. Section 1.2 outlines the most important parallel and distributed data mining strategies and techniques.

1.1 DATA MINING CONCEPTS

Computers have been created to help humans in executing complex and long operations automatically. One of the main effects of the invention of computers is the very huge amount of digital data that nowadays is stored in the memory of computers. Those data volumes can be used to know and understand facts, behaviors, and natural phenomena and take decisions on the basis of them. Researchers investigated methods for instructing computers to learn from data. In particular, machine learning is a scientific discipline that deals with the design and implementation of models, procedures, and algorithms that can learn from data. Such techniques are able to build a predictive model based on data input to be used for making predictions or taking decisions. More recently, data mining has been defined as an area of computer science where machine learning techniques are used to discover previously *unknown* properties in large data sets. More formally, data mining is the analysis of data sets to find interesting, novel, and useful patters, relationships, models, and trends. Data mining tasks include methods at the intersection of artificial intelligence, machine learning, statistics, mathematics, and database systems. The overall practical goal of a data mining task is to extract information from a data set and transform it into an understandable structure for further use. Data mining is considered also the central step

of the knowledge discovery in databases (KDD) process that aims at discovering useful patterns and models for making sense of data. The additional steps in the KDD process are data preparation, data selection, data cleaning, incorporation of appropriate prior knowledge, and interpretation of the results of mining. They are essential to ensure, together with the data mining step, that useful knowledge is derived from the data that have to be analyzed.

Many data mining algorithms have been designed and implemented in several research areas such as statistics, machine learning, mathematics, artificial intelligence, pattern recognition, and databases, each of which uses specialized techniques from the respective application field. The most common types of data mining tasks include:

- *Classification*: the goal is to classify a data set in one or more predefined classes. This is done by models that implement a mapping from a vector of values to a categorical variable. In this way using classification we can predict the membership of a data instance to a given class from a set of predefined classes. For instance, a set of outlet store clients can be grouped in three classes: high spending, average spending, and low spending clients or a set of patients can be classified according to a set of diseases. Classification techniques are used in many application domains such financial services, bioinformatics, document classification, multimedia data, text processing, and social network analysis.
- *Regression*: it is a predictive technique that associates a data set to a quantitative variable and predicts the value of that variable. There are many applications of regression, such as assessing the likelihood that a patient can get sick from the results of diagnostic tests, predicting the margin of victory of a sport team based on results and technical data of previous matches. Regression is often used in economics, environmental studies, market trends, meteorology, and epidemiology.
- *Clustering*: this data mining task is targeted to identify a finite set of categories or groupings (clusters) to describe the data. Clustering techniques are used when no class to be predicted is available *a priori* and data instances are to be divided in groups of similar instances. The groups can be mutually exclusive and exhaustive, or consist of a more extensive representation, such as in the case of hierarchical

categories. Examples of clustering applications concern the finding of homogeneous subsets of clients in a database of commercial sales or groups of objects with similar shapes and colors. Among the application domains where clustering is used are gene analysis, network intrusion, medical imaging, crime analysis, climatology, and text mining. Unlike classification in which classes are predefined, in clustering the classes must be derived from data, looking for clusters based on metrics of similarity between data without the assistance of users.

- *Summarization*: this data mining task provides a compact description of a subset of data. Summarization methods for unstructured data usually involve text classification that groups together documents sharing similar characteristics. An example of summarization of quantitative data is the tabulation of the mean and standard deviation of each data field. More complex functions involve summary rules and the discovery of functional relationships between variables. Summarization techniques are often used in the interactive analysis of data and the automatic generation of reports.
- *Dependency modeling*: this task consists in finding a model that describes significant dependencies between variables. Here the goal is to discover how some data values depend on other data values. Dependency models are at two levels: the structural level of the model specifies which variables are locally dependent on each other, while the quantitative level specifies the power of dependencies using a numeric scale. Dependency modeling approaches are used in retail, business process management, software development, and assembly line optimization.
- *Association rule discovery*: this task aims at finding sets of items that occur together in records of a data set and the relationships among those items in order to derive multiple correlations that meet the specified thresholds. It is intended to identify strong rules discovered in large data sets using different measures of interestingness. In several application domains it is useful to know how often two or more items of interest co-occur. For instance, association rules can describe which products are commonly sold with other products in one or more stores. In this case association analysis is sometimes called *market basket analysis*. Other significant domains where association discovery is used are web page access, communication

network usage, credit card services, medical diagnosis, and bioinformatics.

- *Outlier detection:* an outlier, is one observation or an item that appears to be inconsistent with or deviate significantly from other items of a data set in which it occurs. Although an outlier can be considered as an error or noise, outliers carry important information. Outlier detection methods have been proposed for several application domains, such as weather prediction, credit card fraud detection, clinical trials, network intrusion detection, system performance analysis, and other data analysis problems.
- *Episode discovery* or *prediction*: this task looks for relationships between temporal sequences of events. Data instances are viewed as sequences of events, where each event has an associated time of occurrence. One key problem in analyzing event sequences is to find frequent episodes, that is, collections of events occurring frequently together. For example, finding that if a given event E_1 happens, then, within a time interval t, there will be also an event E_2 with probability p. Examples of domains where to use episode discovery are alarm analysis in telecommunications, web log analysis, client action sequences, fault analysis in manufacturing plants, occurrences of weather events.

1.1.1 Classification

A classification task has the goal to predict the class for a given unlabeled item. The class must be selected among a finite set of predefined classes. Classification algorithms are among the most used techniques in data mining tasks because in many application domains, data associated to class label are available. In all these cases, a classification algorithm can build a classifier that is a model $\boldsymbol{M}$ that calculates the class label $\boldsymbol{c}$ for a given input item $\boldsymbol{x}$, that is, $\boldsymbol{c} = \boldsymbol{M}(\boldsymbol{x})$, where $\boldsymbol{c} \in \{\boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_n}\}$ and each $\boldsymbol{c_i}$ is a class label. To build the model the algorithm requires a set of available items together with their correct class label. This set of classified items is called the *training set*. After generating the model $\boldsymbol{M}$, the classifier can automatically predict the class for any new item that will be given as input to it. Several classification models have been introduced such as decision trees, statistical classifiers, k-nearest neighbors, support vector machines, neural networks, and others. In this section we introduce decision trees and k-nearest neighbors.

1.1.1.1 Decision Trees

Decision trees are the most developed methods for partitioning sets of items into classes. A decision tree classifies data items into a finite number of predefined classes. The tree nodes are labeled with the names of attributes, the arcs are labeled with the possible values of the attribute, and the leaves are labeled with the different classes. Decision trees were introduced in the Quinlan's (1986) ID3 system, one of the earliest data mining algorithms. An item is classified by following a path along the tree formed by the arcs corresponding to the values of its attributes.

A descendant of ID3 used often today for building decision trees is C4.5 (Quinlan, 1993). Given a set $\boldsymbol{C}$ of items, C4.5 first grows a decision tree using the divide-and-conquer algorithm as follows:

- If all the items in $\boldsymbol{C}$ belong to the same class or $\boldsymbol{C}$ is small, the tree is a leaf labeled with the most frequent class in $\boldsymbol{C}$.
- Otherwise, choose a test based on a single attribute with two or more outcomes. Make this test the root of the tree with one branch for each outcome of the test, partition $\boldsymbol{C}$ into a collection of subsets $\boldsymbol{C}_1, \boldsymbol{C}_2, ..., \boldsymbol{C}_n$ according to the outcome for each item, and apply the same procedure recursively to each subset.

Several tests could be used in this last step. C4.5 uses two heuristic criteria to rank possible tests: information gain, which minimizes the total entropy of the subsets $\{\boldsymbol{C}_i\}$, and the default gain ratio that divides information gain by the information provided by the test outcomes.

Attributes can be either numeric or nominal and this determines the format of the test outcomes. For a numeric attribute $\boldsymbol{A}$ they are $\{\boldsymbol{A} \leq \boldsymbol{t}, \boldsymbol{A} > \boldsymbol{t}\}$ where the threshold $\boldsymbol{t}$ is found by sorting $\boldsymbol{C}$ on the values of $\boldsymbol{A}$ and choosing the split between successive values that maximizes the criterion above. An attribute $\boldsymbol{A}$ with discrete values has by default one outcome for each value, but an option allows the values to be grouped into two or more subsets with one outcome for each subset.

The initial tree is then pruned to avoid overfitting. The pruning algorithm is based on a pessimistic estimate of the error rate associated with a set of $\boldsymbol{N}$ cases, $\boldsymbol{Z}$ of which do not belong to the most frequent class. Instead of $\boldsymbol{Z}/\boldsymbol{N}$, C4.5 determines the upper limit of the binomial

probability when Z events have been observed in N trials, using a user-specified confidence whose default value is 0.25.

Pruning is performed starting from the leaves towards the root. The estimated error at a leaf with N items and Z errors is N times the pessimistic error rate as above. For a subtree, C4.5 adds the estimated errors of the branches and compares it to the estimated error if, the subtree is replaced by a leaf. If the latter is no higher than the former, the subtree is pruned. Similarly, C4.5 checks the estimated error if the subtree is replaced by one of its branches and when this appears beneficial the tree is modified accordingly. The pruning process is completed in one pass through the tree.

A disadvantage of decision trees is that they grow too much in real-world data mining applications therefore they become difficult to understand. Research efforts have been carried out to find simpler representations for decision trees.

In classification techniques based on decision-tree generation such as C4.5 have introduced methods to generate production rules from decision trees. The experts because of their high modularity easily interpret production rules**.** Each rule can be understood without reference to other rules. The accuracy of classification of a decision tree can be improved by transforming the tree into a set of production rules that contain a smaller number of attribute-value conditions. In fact, some conditions may be redundant and Quinlan (1987) showed that their removal reduces the rate of misclassification on the examples that are not part of the training set.

A simple example of a set of classification rules that allow to classify driver license in some US states are as follows:

if Age $>$ *14 and Age* $<$ *18 and family = suffering then*
class = Hardship
if Age $\geq$ *16 and Age* $<$ *18 and family = regular then*
class = Provisional
default class = Unrestricted

This classification rule set states that *hardship* licenses for minors are restricted to people between 14 and 18 years old who need to drive to/from

home and school due to serious hardships. Provisional licenses are issued to new drivers between 14 and 18 years old. The remaining (most Americans) drivers have unrestricted driver licenses.

As Wu et al. (2008) reported in their survey, the main disadvantage of classification rules like the ones produced by C4.5, is the amount of computing time and memory size that they require. In an experiment, data samples ranging from 10,000 to 100,000 items were drawn from a large data set. For decision trees, including from 10,000 to 100,000 items CPU time on a PC increased from 1.4 s to 61 s. The time required for rulesets, however, increased from 32 s to 9715 s (more than 300 times).

1.1.1.2 Classification with kNN

An effective and composite algorithm for data classification is the so-called *k-nearest neighbor* (kNN) classification (Fix and Hodges, 1951), which finds a group of $\boldsymbol{k}$ objects in the training set that are closest to the test object, and bases the assignment of a label on the predominance of a particular class in this neighborhood. This allows, for example to find people who have similar financial status to give them similar credit ratings, or to predict the variety of a plant based on the similarity of its characteristics with predefined set of plant samples.

There are three main elements of this technique:

- a set of labeled items, for example a set of records;
- a distance metric or similarity metric to calculate the distance among objects; and
- the value of $\boldsymbol{k}$, that is the number of nearest neighbors.

To classify an unlabeled data item, the distance of this item to the labeled items is calculated, its *k*-nearest neighbors are identified, then the class labels of these nearest neighbors are used to select the class label of the item.

In particular, the nearest-neighbor classification algorithm operates as follows. Given a training set $\boldsymbol{T}$ and a test item $\boldsymbol{i} = (\boldsymbol{d}', \boldsymbol{f}')$, where $\boldsymbol{d}'$ is the data attributes of the test item and $\boldsymbol{f}'$ is the item class, the algorithm computes the similarity between $\boldsymbol{i}$ and all the training items $(\boldsymbol{d}, \boldsymbol{f}) \in \boldsymbol{T}$ to determine its nearest-neighbor list, $\boldsymbol{T}_i$. Notice that $\boldsymbol{d}$ represents the data attributes of the training item and $\boldsymbol{f}$ is the item class. Once the

nearest-neighbor list is obtained, the test object is classified based on the majority class of its nearest neighbors:

$$f' = \arg\max_c \sum_{(d_j, f_j) \in T_i} F(c = f_i)$$

where c is a class label, f_j is the class label for the jth nearest neighbors, and **F** (·) is an indicator function that returns 1 if its argument is true and 0 otherwise. The kNN technique has been used in many application fields such as: text and document retrieval, bioinformatics, image processing, marketing data analysis, video analysis, and multimedia databases.

The following very simple example shows how the kNN algorithms classifies the rows of the matrix M composed of three rows and two columns. Having $M = [0.90\ 0.86;0.10\ 0.31;0.28\ 0.40]$ and using the following training set $TR = [0\ 0;0.5\ 0.4;0.8\ 1.0]$, where row 1 defines class 1, row 2 defines class 2, and row 3 defines class 3, the classes of the rows of M are obtained as follows: row 1 of M is closest to row 3 of TR, so class(1) = 3. Row 2 of M is closest to row 1 of TR so class(2) = 1, and row 3 of M is closest to row 2 of TR, so class(3) = 2.

1.1.2 Clustering

Clustering is an unsupervised learning technique that separates data items into a number of groups or clusters such that items in the same cluster are more similar to each other and items in different clusters tend to be dissimilar, according to some measure of similarity or proximity. Differently from supervised learning, where training examples are associated with a class label that expresses the membership of every example to a class, clustering assumes no information about the distribution of the items, and it has the task to both discover the classes present in the data set and to assign items among such classes in the best way.

Clustering algorithms can roughly be classified into the three following types (Everitt, 1977): *hierarchical*, *partitional* and *density* or *mode-seeking*.

Hierarchical methods generate a nested sequence of clusters by a hierarchical decomposition of a set of N objects represented by a *dendrogram.* To build the dendrogram, there are two techniques, which work in opposite directions: the *agglomerative* and the *divisive*. Agglomerative

methods start with the N items and at each step two clusters are merged until only one is left. Divisive methods, on the contrary, start with one single cluster containing all the items and iteratively a cluster is split until N clusters containing a single item are obtained. Well-known hierarchical methods are BIRCH, CURE, and CHAMELEON. *Partitional methods* produce a partition of a set of items into K clusters by optimizing a given criterion function. One of the most known criterion is the *squared error criterion*. The *K-means* clustering (Kaufman and Rousseeeuw, 1990) is a well-known and effective method for many practical applications that employ the squared error criterion. *Density search techniques* consider items as points in a metric space and suggest that clusters should be those parts of the space characterized by a high density of data. High-density regions are called *modes*. In Everitt (1977) several density-seeking methods are reported.

Statistical density search-based approaches consider dense regions of the probability density of the data items as different groups. This approach uses statistical concepts with the aim to represent the probability density function through a mixture model. A mixture model consists of several components such that each component generates a cluster. The problem in the mixture model framework is to find the characteristics of the data distribution in each cluster. The *Expectation-Maximization* (*EM*) algorithm of Dempster et al. (1977) and Titterington et al. (1985) is a well-known technique for estimating mixture model parameters that iteratively refines initial mixture model parameter estimates. Several implementations of the *EM* algorithm have been provided and the benefits of this approach with respect to other methods have been analyzed. The method is proved to be scalable on large databases and outperforms the traditional scaling approaches based on sampling.

1.1.2.1 Bayesian Classification

The Bayesian approach to unsupervised learning provides a probabilistic method to inductive inference. In Bayesian classification class membership is expressed probabilistically that is an item is not assigned to a unique class, instead it has a probability of belonging to each of the possible classes. The classes provide probabilities for all attribute values of each item. Class membership probabilities are then determined

by combining all these probabilities. Class membership probabilities of each item must sum to 1, thus there are not precise boundaries for classes: every item must be a member of some class, even though we do not know which one. When every item has a probability of no more than 0.5 in any class, the classification is not well defined because it means that classes are abundantly overlapped. On the contrary, when the probability of each instance is about 0.99 in its most probable class, the classes are well separated.

Let $\boldsymbol{D} = \{\boldsymbol{X}_1,\ldots\boldsymbol{X}_m\}$ denotes the observed data objects, where instances or items $\boldsymbol{X}_i$ are represented as ordered vectors of attribute values $\boldsymbol{X}_i = \{\boldsymbol{X}_{i1},\ldots, \boldsymbol{X}_{ik}\}$. Unsupervised classification aims at determining the *best* class description (hypothesis) $\boldsymbol{h}$ from some space $\boldsymbol{H}$ that predicts data $\boldsymbol{D}$. The term *best* can be interpreted as the *most probable* hypothesis given the observed data $\boldsymbol{D}$ and some prior knowledge on the hypotheses of $\boldsymbol{H}$ in the absence of $\boldsymbol{D}$, that is the prior probabilities of the various hypotheses in $\boldsymbol{H}$ when no data have been observed. Bayes theorem provides a way to compute the probabilities of the best hypothesis, given the prior probabilities, the probabilities of observing the data given the various hypotheses and the observed data.

Let $\boldsymbol{P(h)}$ denote the *prior probability* that the hypothesis $\boldsymbol{h}$ holds before the data have been observed. Analogously, let $\boldsymbol{P(D)}$ denote the prior probability that the data will be observed, that is the probability of $\boldsymbol{D}$ with no knowledge of which hypothesis holds. $\boldsymbol{P(D|h)}$ denotes the probability of observing $\boldsymbol{D}$ in some world where the hypothesis $\boldsymbol{h}$ is valid. In unsupervised classification the main problem is to find the probability $\boldsymbol{P(h|D)}$, that is the probability that the hypothesis $\boldsymbol{h}$ is valid, given the observed data $\boldsymbol{D}$. $\boldsymbol{P(h|D)}$ is called the *posterior probability* of $\boldsymbol{h}$ and expresses the degree of belief in $\boldsymbol{h}$ after the data have been seen. Thus the set of data items biases the posterior probability, while the prior probability is independent of $\boldsymbol{D}$. Bayes theorem provides a method to compute the posterior probability:

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)}$$

which is equivalent to

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{\sum_h P(D \mid h)P(h)}$$

because of the theorem of total probability which asserts that if events $\boldsymbol{h}_1$, …, $\boldsymbol{h}_n$ are mutually exclusive with $\sum_{i=1}^{n} P(h_i) = 1$ then $P(D) = \sum_{i=1}^{n} P(D \mid h_i) P(h_i)$.

When the set of possible $\boldsymbol{h}$ is continuous then the prior becomes a differential and the sums over $\boldsymbol{h}$ are integrals, thus this computation becomes difficult to realize. The undertaken approach consists in using approximations in the following way. Instead of considering all the possible states of the world and, consequently all the possible hypotheses $\boldsymbol{h}$ specifying that the world is in some particular state, in *EM* only a small space of models is considered and the assumption $\boldsymbol{S}$ that one of such models describes the world is believed true. A model consists of two sets of parameters: *a set of discrete parameters* $\boldsymbol{T}$ which describes the functional form of the model, such as number of classes and whether attributes are correlated, and *a set of continuous parameters* $\boldsymbol{V}$ that specifies values for the variables appearing in $\boldsymbol{T}$, needed to complete the general form of the model. Given a set of data $\boldsymbol{D}$, *EM* searches for the most probable pair $\boldsymbol{V}, \boldsymbol{T}$ which classifies $\boldsymbol{D}$, $\boldsymbol{P}(\boldsymbol{D}\ \boldsymbol{V}\ \boldsymbol{T} \mid \boldsymbol{S}) = \boldsymbol{P}(\boldsymbol{D} \mid \boldsymbol{V}\ \boldsymbol{T}\ \boldsymbol{S})\ \boldsymbol{P}(\boldsymbol{V}\ \boldsymbol{T} \mid \boldsymbol{S})$. This is done in two steps:

1. For a given $\boldsymbol{T}$, *EM* seeks the maximum posterior (*MAP*) parameter values $\boldsymbol{V}$.
2. Regardless of any $\boldsymbol{V}$, *EM* searches for the most probable $\boldsymbol{T}$, from a set of possible $\boldsymbol{T}_s$ with different attribute dependencies and class structure.

There are two levels of search: *parameter level search* and *model level search*. Fixed the number *J* of classes and their class model, the space of allowed parameter values is searched for finding the most probable *V*. This space is real valued and contains many local maxima, thus finding the global maximum is not an easy task. The algorithm randomly generates a starting point and converges on the maximal values nearest to the starting point. Model level search chooses the best model structure. It is assumed that all instances in *D* are independent, thus

$$\boldsymbol{P}(\boldsymbol{D} \mid \boldsymbol{VTS}) = \prod_i \boldsymbol{P}(\boldsymbol{X}_i \mid \boldsymbol{VTS})$$

and it is necessary to find the probability $\boldsymbol{P}(\boldsymbol{X}_i \mid \boldsymbol{VTS})$ of each instance $\boldsymbol{X}_i$.

The joint probability $P(DVT|S)$ has, generally, many local maxima. To find them, a variant of the *EM* algorithm of Dempster et al. (1977) and Titterington et al. (1985) implemented in *AutoClass*, is based on the fact that at a maximum the class parameter V_j can be estimated from *weighted sufficient statistics*, that summarizes all the information relevant to a model. The weights w_{ij} give the probability that an instance X_i is a member of class j. These weights must satisfy $\Sigma_j w_{ij} = 1$. This approximate information can be used to re-estimate the parameters V and this new set of parameters permits re-estimation of the class probabilities. Repeating these two steps brings to a local optimum. In order to find as many local maxima as possible, *AutoClass* generates pseudorandom points in the parameter space, converges to a local optimum, records the results and repeats the same steps.

1.1.2.2 The K-Means Algorithm

K-means is one of the most used clustering algorithms based on a *partitional strategy. K*-means is an algorithm that minimizes the squared error of values from their respective cluster means. In this way *K*-means implements hard clustering, where each item is assigned to only one cluster (Kaufman and Rousseeeuw, 1990). On the contrary, *EM* is a soft clustering approach because it returns the probability that an item belongs to each cluster. Thus *EM* can be seen as a generalization of *K*-means obtained by modeling the data as a mixture of normal distributions, and finding the cluster parameters (the mean and covariance matrix) by maximizing the likelihood of data.

The *K*-means algorithm is an iterative clustering algorithm to partition a given dataset into a user specified number of clusters, k. The algorithm has been proposed by several researchers such as Lloyd (1957, 1982), Friedman and Rubin (1967), and McQueen (1967).

The algorithm operates on a set of d-dimensional vectors, $D = \{x_i \mid i = 1,\ldots, N\}$, where x_i is a real vector, so it belongs to $\Re^d$ and denotes the ith data item. The algorithm starts by picking k points in $\Re^d$ as the initial representatives or "centroids" of the k clusters. Several techniques for selecting these initial seeds have been proposed. They include sampling at random from the dataset, setting them as the solution of clustering a small subset of the data or perturbing the global mean of the data k times. Then the algorithm iterates between two steps till it will converge:

*S*1 – *Assignment Step*: Each data item is assigned to its *nearest* centroid (whose mean yields the least within-cluster sum of squares),

with ties broken arbitrarily. This results in a partitioning of the data set in $\boldsymbol{k}$ groups or clusters.
*S*2 – *Relocation Step*: The new means to be the centroids of the observations in the new clusters are calculated. Each cluster representative is relocated to the center (mean) of all data items assigned to it. If the data items have an associated probability measure (weights), then the relocation is to the expectations (weighted mean) of the data partitions.

The *K*-means algorithm converges when the assignments (and hence the $\boldsymbol{c}_j$ values) no longer change. Notice that each iteration needs $\boldsymbol{N} \times \boldsymbol{k}$ comparisons, which determines the time complexity of one iteration. The number of iterations required for convergence varies and can depend on $\boldsymbol{N}$, but as a first cut, this algorithm can be considered linear in the dataset size.

One issue to solve is how to quantify the nearest concept in the assignment step. The default measure of closeness is the Euclidean distance, in which case one can readily show that the nonnegative cost function below will decrease whenever there is a change in the assignment or the relocation steps, and hence convergence is guaranteed in a finite number of iterations.

$$\sum_{i=1}^{N}\left(\underset{j}{\operatorname{argmin}}\left\|x_i - c_j\right\|_2^2\right)$$

As can be deduced from the assignment step, the *K*-means algorithm is sensitive to initialization, and it suffers from other limitations. For example, the algorithm works well when the data fits the cluster model, that is the clusters are spherical because the data points in a cluster are centered around that cluster, and the spread/variance of the clusters is similar, that is each data point belongs to the closest cluster. Despite those limitation, the *K*-means is the most used clustering algorithm.

1.1.3 Association Rules

As we mentioned before, in many application domains it is useful to discover how often two or more items co-occur. This holds, for example, when we wish to know what goods customers buy together or which

pages of a web site users access in the same session. Mining frequent patters is the basic task in all these cases. Once the frequent groups of items are found in a data set, it is also possible to discover the association rules that hold for the frequent itemsets. Association rule discovery has been proposed by Agrawal et al. (1993) as a method for discovering interesting association among variables in large data sets.

The association rule mining task can be defined as follows: Let $\boldsymbol{I} = \{\boldsymbol{i}_1, \boldsymbol{i}_2, \ldots, \boldsymbol{i}_n\}$ be a set of $\boldsymbol{n}$ binary attributes called *items*. Let $\mathbf{D} = \{\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_m\}$ be a set of transactions called the *data set*. Each transaction in $\boldsymbol{D}$ has a unique transaction identifier and contains a subset of the items in $\boldsymbol{I}$ called *itemset*. A *rule* is defined as an implication of the form

$$\boldsymbol{X} \Rightarrow \boldsymbol{Y} \qquad \text{where } \boldsymbol{X}, \boldsymbol{Y} \subseteq \boldsymbol{I} \text{ and } \boldsymbol{X} \cap \boldsymbol{Y} = \boldsymbol{\emptyset}.$$

The *itemsets* $\boldsymbol{X}$ and $\boldsymbol{Y}$ are called *antecedent* and *consequent* of the rule, respectively. The antecedent and consequent are sets of items that are disjoint. A simple example rule for a set of products sold in a supermarket in the same basket of a client could be *(mustard, Vienna sausages)* $\Rightarrow$ *(buns)* meaning that if customers buy mustard and hot dog sausages, they also buy buns.

The degree of uncertainty of an association rule is expressed by two values. The first one is called the *support* for the rule. The *support* is the number of transactions that include all items in the antecedent and consequent parts of the rule. The *support* can be expressed as relative value, that is a percentage of the total number of records in the data set. For example, if an itemset {*mustard, Vienna sausages, buns*} occurs in 25% of all transactions (1 out of 4 transactions), it has a *support* of 1/4 = 0.25. The second value is known as the *confidence* of the rule. The *confidence* is the ratio of the number of transactions that include all items in the consequent as well as the antecedent (the *support*) to the number of transactions that include all items in the antecedent: *confidence*($\boldsymbol{X} \Rightarrow \boldsymbol{Y}$) = *support*($\boldsymbol{X} \cup \boldsymbol{Y}$) / *support*($\boldsymbol{X}$). For example, if a supermarket database has 100,000 transactions, out of which 5,000 include both items $\boldsymbol{X}$ and $\boldsymbol{Y}$ and 1,000 of these include item $\boldsymbol{Z}$, the association rule "If $\boldsymbol{X}$ and $\boldsymbol{Y}$ are purchased, then $\boldsymbol{Z}$ is purchased in the same basket" has a support of 1,000 transactions (alternatively 2% = 1,000/100,000) and a *confidence* of 20% (=1,000/5,000).

Association rules are usually searched to satisfy user-specified *minimum support* and *minimum confidence* at the same time. Association rule generation is usually split up into two separate steps:

1. Minimum support is applied to find all frequent itemsets in a data set.
2. These frequent itemsets and the minimum confidence constraint are used to compose the rules.

Finding all frequent itemsets in a data set is a complex procedure since it involves analyzing all possible itemsets. All the possible itemsets is the power set over $\boldsymbol{I}$ and has size 2^{n-1} (excluding the empty set which is not a valid itemset). Although the size of the power set grows exponentially in the number of items $\boldsymbol{n}$ in $\boldsymbol{I}$, efficient search is possible using the *downward-closure property* of support (also called *antimonotonicity*) which guarantees that for a frequent itemset, all its subsets are also frequent, therefore for an infrequent itemset, all its supersets must also be infrequent. Efficient algorithms such as Apriori (Agrawal and Srikant, 1994) and Eclat (Zaki, 2000) can find all frequent itemsets.

In particular, Apriori is one of the most used algorithms for finding frequent itemsets using candidate generation. It is characterized as a level-wise search algorithm using *antimonotonicity* of itemsets. Let the set of frequent itemsets of size $\boldsymbol{k}$ be $\boldsymbol{F_k}$ and their candidates be $\boldsymbol{C_k}$. Apriori first scans the database and searches for frequent itemsets of size 1 by counting the occurrence of each item and collecting those that satisfy the minimum support requirement. It then iterates on the following three steps and extracts all the frequent itemsets.

1. Generate $\boldsymbol{C_{k+1}}$, candidates of frequent itemsets of size $\boldsymbol{k+1}$, from the frequent itemsets of size $\boldsymbol{k}$.
2. Scan the database and calculate the support of each candidate of frequent itemsets.
3. Add those itemsets that satisfy the minimum support requirement to $\boldsymbol{F_{k+1}}$.

In general, the Apriori algorithm gets good performance by reducing the size of candidate sets; however, when must be analyzed very many frequent itemsets, large itemsets, and/or very low minimum support is used, Apriori suffers from the cost of generating a huge number of candidate sets and for scanning all the transactions repeatedly to check a

large set of candidate itemsets. For example, it is necessary to generate 2^{80} candidate itemsets to obtain frequent itemsets of size 80. For this reason, more efficient sequential and parallel solutions for frequent itemset mining have been designed to handle large and very frequent itemsets.

1.2 PARALLEL AND DISTRIBUTED DATA MINING

This section introduces the main strategies that are used for the implementation of parallel and distributed data mining techniques and describes a few parallel data mining algorithms. Sequential data mining algorithms designed to analyze very large and/or distributed data sets on conventional computers often need to run for a very long time to produce the data mining models. The main approach to reduce response time is to use parallel and distributed computing models and systems. High performance computers, such as multiclusters, clouds, and many-core systems, if equipped with distributed and parallel data mining tools and algorithms, can provide an effective approach to analyze big data sets and obtain results in reasonable time. This approach is based on the exploitation of inherent parallelism of most data mining algorithms. Here we introduce the main issues and discuss some solutions.

Three main strategies can be identified for the exploitation of parallelism in data mining algorithms:

- *Independent parallelism*: it can be exploited when processes are executed in parallel in a fully independent way. Generally, each process accesses the whole data set and does not communicate or synchronize with other processes during the data mining operations.
- *Single Program Multiple Data* (*SPMD*) *parallelism*: a set of processes execute in parallel the same algorithm on different partitions of a data set; processes cooperate to exchange partial results during execution.
- *Task/control parallelism*: it is the more general form of parallelism since each process can execute different operations on (a different partition of) the data set, and processes communicate as the parallel algorithm requires.

It must be noted that the three models are not necessarily alternative for the implementation of parallel data mining algorithms. They

can be combined to implement hybrid parallel data mining algorithms (Talia, 2002).

1.2.1 Parallel Classification

As described in Section 1.1.1.1, decision trees are an effective and popular technique for classification. They are tree-shaped structures that represent a way to classify items. Paths in those trees, from the root to a leaf, correspond to rules for classifying a data set whereas the tree leaves represent the classes and the tree nodes represent attribute values.

Independent parallelism can be exploited in decision tree construction assigning to a process the goal to construct a decision tree according to some parameters. If several processes are executed in parallel on different computing elements, a set of decision tree classifiers can be obtained at the same time. One or more of such trees can be selected as classifiers for the data.

Using the *Task parallelism approach* one process is associated to each subtree of the decision tree that is built to represent a classification model. The search occurs in parallel in each subtree, thus the degree of parallelism P is equal to the number of active processes at a given time. This approach can be implemented using the farm parallelism pattern in which one master process controls the computation and a set of W workers that are assigned to the subtrees. The result is a single decision tree built in a shorter time with respect to the sequential tree building.

According to *SPMD parallelism*, a set of processes executes the same code to classify the items of a subset of the data set. The P processes search in parallel in the whole tree using a partition D/P of the data set D. The global result is obtained by exchanging partial results among the processes. The data set partitioning may be operated in two different ways: (i) by partitioning the D tuples of the data set assigning D/P tuples per processor or (ii) by partitioning the n attributes of each tuple and assigning D tuples of n/P attributes per processor.

Kufrin (1997) proposed a parallel implementation of the C4.5 algorithm that uses the independent parallelism approach. Some other examples of parallel algorithms for building decision trees are Top-Down Induction of Decision Trees (Pearson, 1994) and SPRINT (Shafer et al., 1996).

1.2.2 Parallel Clustering

Parallelism in clustering algorithms can be exploited both in the data clustering strategy and in the computation of the similarity or distance among the data items by computing on each processor the distance/similarity of a different partition of items. In the parallel implementation of clustering algorithms the three main parallel strategies described before can be used.

In *Independent parallelism* each processor uses the whole data set $\boldsymbol{D}$ and implements a different clustering task based on a different number of clusters $\boldsymbol{k_p}$. To get the load among the processors balanced, until the clustering task is complete a new clustering task is assigned to a processor that completed its assigned grouping.

According to *Task parallelism* each processor executes a different task that executes the clustering algorithm and cooperates with the other processors exchanging partial results. For example, in partitioning methods processors can work on disjoint regions of the search space using the whole data set. In hierarchical methods a processor can be responsible of composing one or more clusters. It finds the nearest neighbor cluster by computing the distance among its cluster and the others. Then all the local shortest distances are exchanged to find the global shortest distance between two clusters that must be merged. The new cluster will be assigned to one of the two processors that handled the merged clusters.

Finally, in *SPMD parallelism* each processor run the same algorithm on a different partition $\boldsymbol{D/P}$ of the data set to compute partial clustering results. Local results obtained on the assigned partitions are then shared among all the processors to compute global values on every processor. Global values are used in all processors to start the next clustering step until a convergence is reached or a given number of clustering steps are performed. The SPMD strategy can also be used to implement clustering algorithms where each processor generates a local approximation of a model, which at each iteration can be sent to the other processors that in turn use it to improve their clustering model.

A parallel implementation of a clustering algorithm is, for example, P-CLUSTER. Other parallel algorithms are discussed in Bruynooghe (1989) and Foti et al. (2000). In particular, in Pizzuti and Talia (2003) an SPMD implementation of the AutoClass algorithm, named P-AutoClass

is described. That paper shows interesting performance results on distributed memory MIMD machines. Finally, Olson (1995) presents a set of hierarchical clustering algorithms and an analysis of their time complexity on different parallel architectures.

1.2.3 Parallelism in Association Rules

As discussed, association rules algorithms are used for the automatic discovery of complex associations in a data set. Given a set of transactions $\boldsymbol{D}$, as described in Section 1.1.3, the problem of mining association rules is to generate all association rules that have support (how often a combination occurred overall) and confidence (how often the association rule holds true in the data set) greater than the user-specified minimum support and minimum confidence respectively.

Independent parallelism can be used to run in parallel association rule algorithms that avoid data dependencies among the different processes. This can be done by partitioning and replicating data and candidate frequent itemsets so that processes can run autonomously. For example the Candidate Distribution method proposed for implementing the Apriori algorithm (Agrawal and Shafer, 1996) partitions candidate itemsets but selectively replicates instead of partition-and-exchanging the database transactions, so that each process can proceed independently.

In *SPMD parallelism* the data set $\boldsymbol{D}$ is partitioned among the $\boldsymbol{P}$ processors but candidate itemsets $\boldsymbol{I}$ are replicated on each processor. Each process $\boldsymbol{p}$ counts in parallel the partial support $\boldsymbol{S_p}$ of the global itemsets on its local partition of the data set of size $\mathbf{D/P}$. At the end of this phase the global support $\boldsymbol{S}$ is obtained by collecting all local supports $\boldsymbol{S_p}$. The replication of the candidate itemsets minimizes communication, but it does not use memory efficiently. Due to low communication overhead, scalability is good.

According to *Task parallelism*, both the data set $\boldsymbol{D}$ and the candidate itemsets $\boldsymbol{I}$ are partitioned on each processor. Each process $\boldsymbol{p}$ counts the global support $\boldsymbol{S_i}$ of its candidate itemset $\boldsymbol{I_p}$ on the entire data set $\boldsymbol{D}$. After scanning its local data set partition $\boldsymbol{D/P}$, a process must scan all remote partitions for each iteration. The partitioning of the data set and the candidate itemsets minimizes the use of memory but requires high communication overhead in distributed memory architectures. Due to

communication overhead this approach is less scalable than the previous one.

Also in this case, a combination of the three different parallelism approaches can be implemented. As mentioned, the Apriori algorithm is the most known algorithm for association rules discovery. Several parallel implementations have been proposed for this algorithm. In (Agrawal and Shafer, 1996) three different parallel algorithms called Count Distribution, Data Distribution, and Candidate Distribution are presented. The first one is based on SPMD parallelism, the second one is based on task parallelism whereas the third one, as stated above, uses the independent parallelism approach. In Han et al. (2000) two additional parallel approaches to Apriori called Intelligent Data Distribution (IDD) and Hybrid Distribution (HD) are presented. A complete review of parallel algorithms for association rules can be found in Zaki (1999).

1.2.4 Distributed Data Mining

Distributed data mining (DDM) techniques use distributed computing systems to store data sets and run data mining algorithms exploiting their inherent parallelism by distributing data on different computers and running the mining code locally on that computers. This approach is suitable for applications that typically deal with very large amount of data that cannot be analyzed in a single computer or on a single site using traditional machines in acceptable times or that need to mine data sources located in remote sites, such as Web servers or departmental data owned by large enterprises, or data streams coming from sensor networks or satellites.

Centralized architectures are inappropriate for most of the distributed and ubiquitous data mining applications that involve the processing of Big Data. In fact, the long response time, the lack of proper use of distributed resource, and the fundamental features of centralized data mining algorithms do not work well in parallel and distributed environments. As we discussed for parallel data analysis techniques, scalable solutions for distributed applications calls for distributed processing of data, controlled by the available resources and expert data miners.

Most distributed data analysis algorithms are designed upon the potential concurrency they can apply over the given distributed data.

Typically, the same code runs on each distributed data site concurrently, producing one local model per site. Subsequently, all local models are aggregated to produce the final model. This schema is common to several distributed data mining algorithms. Among them, ensemble learning, meta-learning, and collective data mining are the most important. Moreover, we must mention that distribution and parallelism can also be used together to implement very large distributed data analysis applications where also the local models that compose the global model obtained in a distributed way, are computed in parallel according to the techniques discussed in the previous section.

1.2.4.1 Meta-Learning

The meta-learning techniques aim at implementing a global model from a set of distributed data sets. Meta-learning can be defined as learning from learned knowledge (Prodromidis et al., 2000). In a data classification scenario, this is achieved by learning from the predictions of a set of base data classifiers on a common validation set. Here we list the steps needed to build a global classifier from a set of distributed training sets according to a meta-learning approach:

1. The initial training sets are given in input to ***n*** learning algorithms that run in parallel to build ***n*** classification models (base classifiers).
2. A meta-level training set is built by combining the predictions of the base classifiers on a common validation set.
3. A global classifier is trained from the meta-level training set by a meta-learning algorithm.

Stacking is a way of combining multiple models in meta-learning. Stacking is not used to combine models of the same type. It is applied to models built by using different learning algorithms. It does not use a voting approach, but tries to learn which classifiers are the reliable ones, using another learning algorithm (the meta-learner) to discover how best to combine the output of the base learners.

1.2.4.2 Collective Data Mining

Collective data mining, instead of combining a set of complete models generated at each site on partitioned or replicated data sets, builds the global model through the combination of partial models computed in the different sites. The global model is directly composed by summing

an appropriate set of basis functions. The global function $f(x)$ that represents the global model can be expressed as:

$$f(x) = \sum_k w_k \psi_k(x)$$

where $\psi_k(x)$ is the kth basis function and w_k is the corresponding coefficient that must be learned locally on each site based on the stored data set. This result is founded on the fact that any function can be expressed in a distributed fashion using a set of appropriate basis functions that may contain nonlinear terms. If the basis functions are orthonormal, the local analysis generates results that can be correctly used as components of the global model. If in the summation function is present a nonlinear term, the global model is not fully decomposable among local sites and crossterms involving features from different sites must be considered. Kargupta et al. (2000), described the following main steps of the collective data mining approach:

1. Select an appropriate orthonormal representation for the type of data model to be generated.
2. Generate at each site approximate orthonormal basis coefficients.
3. If the global function includes nonlinear terms, move a sample of data sets from each site to a central site and generate there the approximate basis coefficients corresponding to such nonlinear terms.
4. Combine the local models to generate the global model and transform it into the user described model representation.

1.2.4.3 Ensemble Learning

Ensemble learning aims at improving classification accuracy by aggregating predictions of multiple classifiers (Tan et al., 2006). An ensemble learning method builds a set of base classifiers from training data and performs classification by voting (in the case of classification) or by averaging (in the case of regression) on the predictions made by each classifier. The final result is the ensemble classifier, which very often have higher classification quality than any single classifier that has been used to compose it.

These are the main steps that compose an ensemble learning strategy for data classification:

1. Using a partitioning tool the input data set is split into a training set and a test set.

2. The training set is given in input to ***n*** classification algorithms that run concurrently on different processing nodes to build ***n*** independent classification models.
3. Then, a voter tool ***V*** accesses the ***n*** models and performs an ensemble classification by assigning to each instance of the test set the class predicted by the majority of the ***n*** models generated at the previous step.

The identification of optimal ways to combine the base classifiers is a crucial point here. The most adopted approaches are two schemes called *bagging* and *boosting*. *Bagging* (called voting for classification and averaging for regression) combines the predicted classifications (prediction) from multiple models, or from the same type of model for different learning data. *Bagging* is also used to address the inherent instability of results when applying complex models to relatively small data sets. *Boosting* also combines the decisions of different models, like *bagging*, by amalgamating the various outputs into a single prediction, but it derives the individual models in different ways. In *bagging*, the models receive equal weight, whereas in *boosting*, weighting is used to give more influence to the more successful ones.

1.3 SUMMARY

Advances and pervasiveness of computers has been the main driving force of the very huge amounts of digital data that are collected and stored in digital repositories today. Those data volumes can be exploited to extract useful information and producing helpful knowledge for science, industry, public services, and in general for humankind. Data scientists define and use algorithms and methods for instructing computers to learn from data. From this scenario was born the data mining field as a discipline that today provides several different techniques and algorithms for the automatic analysis of large data sets. In this chapter, we introduced the main data mining tasks and presented the most important and most used data mining techniques. Each one of them can be used in several application domains to identify patterns, trends, categories, and find particular events.

Since for the analysis of several complex and large data sources sequential data mining algorithms often need to run for a very long time

to produce patterns and models, high performance system and concurrent algorithms are used. In fact, a main approach to reduce response time is based on the use of parallel and distributed algorithms and computing systems. High performance computers, such as multiclusters, clouds, and many-core systems, together with distributed and parallel data mining tools and algorithms, can provide effective solutions to analyze big data sets and obtain results in reasonable time. In general, this approach is based on the exploitation of inherent parallelism of most data mining algorithms. For this reason, in the second part of this chapter we introduce the main issues and discussed some scalable data mining solutions. All the concepts introduced in this chapter, together with those discussed in Chapter 2 pave the way for studying the rest of the book where cloud-based data mining systems, tools, and applications are discussed.

REFERENCES

Agrawal, R., Shafer, J.C., 1996. Parallel mining of association rules. IEEE Trans. Knowledge Data Eng. 8, 962–969.

Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules. In: Proceedings of the 20th VLDB Conference, pp. 487–499.

Agrawal, R., Imielinski, T., Swami, A.N., 1993. Database mining: a performance perspective. IEEE Trans. Knowledge Data Eng. 5 (6), 914–925.

Bruynooghe, M., 1989. Parallel implementation of fast clustering algorithms. In: Proceedings of the International Symposium on High Performance Computing, Elsevier Science, pp. 65–78.

Dempster, A.P., Laird, N.M., Rubin, D.B., 1977. Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc. B 39 (1), 1–38.

Everitt, B., 1977. Cluster Analysis. Heinemann Educational Books Ltd, London.

Fix, E., Hodges Jr, J.L., 1951. Discriminatory Analysis, non-Parametric Discrimination, USAF School of Aviation Medicine, Randolph Field, Tex., Project 21-49-004, Rept. 4, Contract AF41(128)-31, February 1951.

Foti, D., Lipari, D., Pizzuti, C., Talia, D., 2000. Scalable parallel clustering for data mining on multicomputers. In: Proceedings of the Third International Workshop on High Performance Data Mining, LNCS 1800, Springer-Verlag, pp. 390–398.

Friedman, H.P., Rubin, J., 1967. On some invariant criteria for grouping data. J. Am. Stat. Assoc. 62, 1159–1178.

Han, E.H., Karypis, G., Kumar, V., 2000. Scalable parallel data mining for association rules. IEEE Trans. Knowledge Data Eng. 12 (2), 337–352.

Kargupta, H., Park, B., Hershberger, D., Johnson, E., 2000. A new perspective toward distributed data mining. Advances in Distributed and Parallel Knowledge Discovery. AAAI/MIT Press, Menlo Park, CA.

Kaufman, L., Rousseeeuw, P.J., 1990. Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons, Hoboken, NJ.

Kufrin, R., 1997. Generating C4.5 production rules in parallel. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence, AAAI Press.

Lloyd, S. P., 1982. Least squares quantization in PCM. Unpublished Bell Lab. Tech. Note, portions presented at the Institute of Mathematical Statistics Meeting Atlantic City, NJ, September 1957. Also, IEEE Trans. Information Theory (Special Issue on Quantization), IT-28, pp. 129–137, March 1982.

McQueen, J., 1967. Some methods for classification and analysis of mutivariate observations. In: Proceedings Fifth Berkeley Symposium on Mathematics, Statistics and Probability, vol.1, pp. 281–296.

Olson, C.F., 1995. Parallel algorithms for hierarchical clustering. Parallel Comput. 21, 1313–1325.

Pearson, R.A., 1994. A coarse-grained parallel induction heuristic. In: Kitano, H., Kumar, V., Suttner, C.B. (Eds.), Parallel Processing for Artificial Intelligence 2. Elsevier Science, Amsterdam.

Pizzuti, C., Talia, D., 2003. P-AutoClass: scalable parallel clustering for mining large data sets. IEEE Trans. Knowledge Data Eng. 15 (3), 629–641.

Prodromidis, A.L., Chan, P.K., Stolfo, S.J., 2000. Meta-learning in distributed data mining systems: issues and approaches. Advances in Distributed and Parallel Knowledge Discovery. AAAI/MIT Press, Menlo Park, CA.

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

Quinlan, J. R., 1987. Generating production rules from decision trees. In: Proceedings International Conference on Artificial Intelligence, Milano, pp. 304–307.

Quinlan, J.R., 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, San Mateo, CA.

Shafer, J., Agrawal, R., Mehta, M., 1996. SPRINT: a scalable parallel classifier for data mining. Proceedings of the Twenty-Second International Conference on Very Large Databases.

Talia, D., 2002. Parallelism in knowledge discovery techniques. In: Proceedings Conference PARA 2002, LNCS 2367, Springer-Verlag, pp. 127–136.

Tan, P.N., Steinbach, M., Kumar, V., 2006. Introduction to Data Mining. Addison-Wesley, Boston, MA.

Titterington, D.M., Smith, A.F.M., Makov, U.E., 1985. Statistical Analysis of Finite Mixture Distribution. John Wiley & Sons, New York.

Wu, X., et al., 2008. Top 10 algorithms in data mining. Knowledge Inform. Syst. 14, 1–37.

Zaki, M.J., 1999. Parallel and distributed association mining: a survey. IEEE Concurrency 7 (4), 14–25.

Zaki, M.J., 2000. Scalable algorithms for association mining. IEEE Trans. Knowledge Data Eng. 12 (3), 372–390.