

Chapter 1

Knowledge Discovery and Data Mining: Concepts and Fundamental Aspects

1.1 Overview

The goal of this chapter is to summarize the preliminary background required for this book. The chapter provides an overview of concepts from various interrelated fields used in the subsequent chapters. It starts by defining basic arguments from data mining and supervised machine learning. Next, there is a review on some common induction algorithms and a discussion on their advantages and drawbacks. Performance evaluation techniques are then presented and finally, open challenges in the field are discussed.

1.2 Data Mining and Knowledge Discovery

Data mining is the science and technology of exploring data in order to discover previously unknown patterns. Data Mining is a part of the overall process of Knowledge Discovery in databases (KDD). The accessibility and abundance of information today makes data mining a matter of considerable importance and necessity.

Most data mining techniques are based on inductive learning (see [Mitchell (1997)]), where a model is constructed explicitly or implicitly by generalizing from a sufficient number of training examples. The underlying assumption of the inductive approach is that the trained model is applicable to future, unseen examples. Strictly speaking, any form of inference in which the conclusions are not deductively implied by the premises can be thought of as induction.

Traditionally *data collection* is considered to be one of the most important stages in data analysis. An analyst (e.g., a statistician) used

the available domain knowledge to select the variables to be collected. The number of variables selected was usually small and the collection of their values could be done manually (e.g., utilizing hand-written records or oral interviews). In the case of computer-aided analysis, the analyst had to enter the collected data into a statistical computer package or an electronic spreadsheet. Due to the high cost of data collection, people learned to make decisions based on limited information.

However, since the information-age, the accumulation of data become easier and storing it inexpensive. It has been estimated that the amount of stored information doubles every twenty months [Frawley *et al.* (1991)]. Unfortunately, as the amount of machine readable information increases, the ability to understand and make use of it does not keep pace with its growth. Data mining is a term coined to describe the process of sifting through large databases in search of interesting patterns and relationships. Practically, Data Mining provides tools by which large quantities of data can be automatically analyzed. Some of the researchers consider the term “Data Mining” as misleading and prefer the term “Knowledge Mining” as it provides a better analogy to gold mining [Klosgen and Zytkow (2002)].

The Knowledge Discovery in Databases (KDD) process was defined by many, for instance [Fayyad *et al.* (1996)] define it as “the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data”. [Friedman (1997a)] considers the KDD process as an automatic exploratory data analysis of large databases. [Hand (1998)] views it as a secondary data analysis of large databases. The term “Secondary” emphasizes the fact that the primary purpose of the database was not data analysis. Data Mining can be considered as a central step of the overall process of the Knowledge Discovery in Databases (KDD) process. Due to the centrality of data mining in the KDD process, there are some researchers and practitioners that use the term “data mining” as synonymous to the complete KDD process.

Several researchers, such as [Brachman and Anand (1994)], [Fayyad *et al.* (1996)], [Maimon and Last (2000)] and [Reinartz (2002)] have proposed different ways to divide the KDD process into phases. This book adopts a hybridization of these proposals and suggests breaking the KDD process into the following eight phases. Note that the process is iterative and moving back to previous phases may be required.

- (1) Developing an understanding of the application domain, the relevant prior knowledge and the goals of the end-user.

- (2) Selecting a data set on which discovery is to be performed.
- (3) Data Preprocessing: This stage includes operations for Dimension Reduction (such as Feature Selection and Sampling), Data Cleansing (such as Handling Missing Values, Removal of Noise or Outliers), and Data Transformation (such as Discretization of Numerical Attributes and Attribute Extraction)
- (4) Choosing the appropriate Data Mining task such as: classification, regression, clustering and summarization.
- (5) Choosing the Data Mining algorithm: This stage includes selecting the specific method to be used for searching patterns.
- (6) Employing The Data mining Algorithm.
- (7) Evaluating and interpreting the mined patterns.
- (8) Deployment: Using the knowledge directly, incorporating the knowledge into another system for further action or simply documenting the discovered knowledge.

1.3 Taxonomy of Data Mining Methods

It is useful to distinguish between two main types of data mining: verification-oriented (the system verifies the user's hypothesis) and discovery-oriented (the system finds new rules and patterns autonomously) [Fayyad *et al.* (1996)]. Figure 1.1 illustrates this taxonomy.

Discovery methods are methods that automatically identify patterns in the data. The discovery method branch consists of prediction methods versus description methods. Description-oriented Data Mining methods focus on (the part of) understanding the way the underlying data operates, where prediction-oriented methods aim to build a behavioral model that can get newly and unseen samples and is able to predict values of one or more variables related to the sample.

However, some prediction-oriented methods can also help provide understanding of the data.

Most of the discovery-oriented techniques are based on inductive learning [Mitchell (1997)], where a model is constructed explicitly or implicitly by generalizing from a sufficient number of training examples. The underlying assumption of the inductive approach is that the trained model is applicable to future unseen examples. Strictly speaking, any form of inference in which the conclusions are not deductively implied by the premises can be thought of as induction.

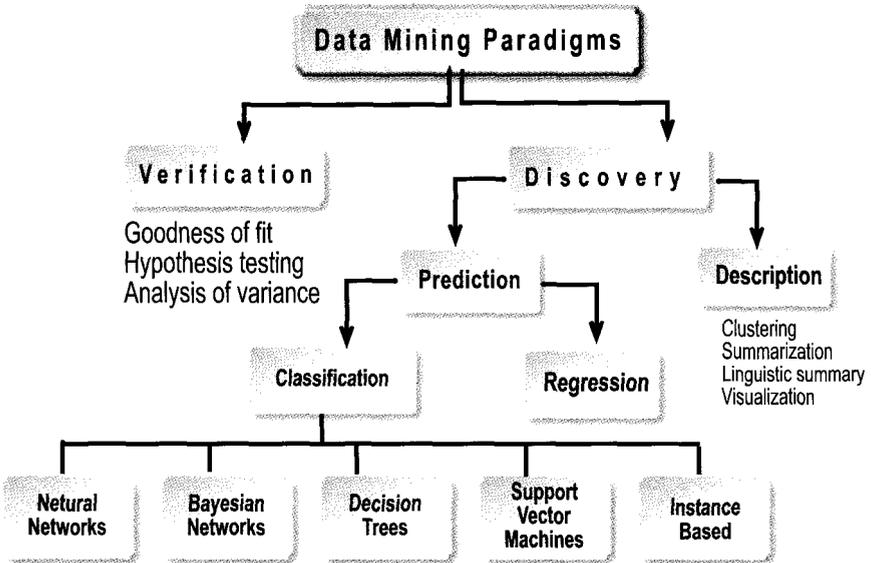


Fig. 1.1 Taxonomy of Data Mining Methods.

Verification methods, on the other hand, deal with evaluation of a hypothesis proposed by an external source (like an expert etc.). These methods include the most common methods of traditional statistics, like goodness-of-fit test, t-test of means, and analysis of variance. These methods are less associated with data mining than their discovery-oriented counterparts because most data mining problems are concerned with selecting a hypothesis (out of a set of hypotheses) rather than testing a known one. The focus of traditional statistical methods is usually on model estimation as opposed to one of the main objectives of data mining: model identification [Elder and Pregibon (1996)].

1.4 Supervised Methods

1.4.1 Overview

Another common terminology used by the machine-learning community refers to the prediction methods as supervised learning as opposed to unsupervised learning. Unsupervised learning refers to modeling the distribution of instances in a typical, high-dimensional input space.

According to [Kohavi and Provost (1998)] the term “Unsupervised learning” refers to “learning techniques that group instances without a prespecified dependent attribute”. Thus the term unsupervised learning covers only a portion of the description methods presented in Figure 1.1, for instance it does cover clustering methods but it does not cover visualization methods.

Supervised methods are methods that attempt to discover the relationship between input attributes (sometimes called independent variables) and a target attribute (sometimes referred to as a dependent variable). The relationship discovered is represented in a structure referred to as a *Model*. Usually models describe and explain phenomena, which are hidden in the dataset and can be used for predicting the value of the target attribute knowing the values of the input attributes. The supervised methods can be implemented on a variety of domains such as marketing, finance and manufacturing.

It is useful to distinguish between two main supervised models: *Classification Models (Classifiers)* and *Regression Models*. Regression models map the input space into a real-valued domain. For instance, a regressor can predict the demand for a certain product given its characteristics. On the other hand classifiers map the input space into predefined classes. For instance classifiers can be used to classify mortgage consumers to good (fully payback the mortgage on time) and bad (delayed payback). There are many alternatives to represent classifiers, for example: Support Vector Machines, decision trees, probabilistic summaries, algebraic function, etc.

This book deals mainly in classification problems. Along with regression and probability estimation, classification is one of the most studied approaches, possibly one with the greatest practical relevance. The potential benefits of progress in classification are immense since the technique has great impact on other areas, both within data mining and in its applications.

1.4.2 Training Set

In a typical supervised learning scenario, a training set is given and the goal is to form a description that can be used to predict previously unseen examples.

The training set can be described in a variety of languages. Most frequently, it is described as a *Bag Instance* of a certain *Bag Schema*. A *Bag Instance* is a collection of tuples (also known as records, rows or instances) that may contain duplicates. Each tuple is described by a vector of

attribute values. The bag schema provides the description of the attributes and their domains. For the purpose of this book, a bag schema is denoted as $B(A \cup y)$ where A denotes the set of input attributes containing n attributes: $A = \{a_1, \dots, a_i, \dots, a_n\}$ and y represents the class variable or the target attribute.

Attributes (sometimes called field, variable or feature) are typically one of two types: nominal (values are members of an unordered set), or numeric (values are real numbers). When the attribute a_i is nominal it is useful to denote by $dom(a_i) = \{v_{i,1}, v_{i,2}, \dots, v_{i,|dom(a_i)|}\}$ its domain values, where $|dom(a_i)|$ stands for its finite cardinality. In a similar way, $dom(y) = \{c_1, \dots, c_{|dom(y)|}\}$ represents the domain of the target attribute. Numeric attributes have infinite cardinalities.

The instance space (the set of all possible examples) is defined as a Cartesian product of all the input attributes domains: $X = dom(a_1) \times dom(a_2) \times \dots \times dom(a_n)$. The Universal Instance Space (or the *Labeled Instance Space*) U is defined as a Cartesian product of all input attribute domains and the target attribute domain, i.e.: $U = X \times dom(y)$.

The training set is a Bag Instance consisting of a set of m tuples. Formally the training set is denoted as $S(B) = (\langle x_1, y_1 \rangle, \dots, \langle x_m, y_m \rangle)$ where $x_q \in X$ and $y_q \in dom(y)$.

Usually, it is assumed that the training set tuples are generated randomly and independently according to some fixed and unknown joint probability distribution D over U . Note that this is a generalization of the deterministic case when a supervisor classifies a tuple using a function $y = f(x)$.

This book uses the common notation of bag algebra to present projection (π) and selection (σ) of tuples ([Grumbach and Milo (1996)]). For example given the dataset S presented in Table 1.1, the expression $\pi_{a_2, a_3} \sigma_{a_1 = "Yes"} AND_{a_4 > 6} S$ result with the dataset presented in Table 1.2.

1.4.3 Definition of the Classification Problem

This section defines the classification problem. Originally the machine learning community has introduced the problem of *concept learning*. Concepts are mental categories for objects, events, or ideas that have a common set of features. According to [Mitchell (1997)]: “each concept can be viewed as describing some subset of objects or events defined over a larger set” (e.g., the subset of a vehicle that constitutes trucks). To learn a concept is to infer its general definition from a set of examples. This definition may be either explicitly formulated or left implicit, but either way it

Table 1.1 Illustration of a Dataset S having five attributes.

a_1	a_2	a_3	a_4	y
Yes	17	4	7	0
No	81	1	9	1
Yes	17	4	9	0
No	671	5	2	0
Yes	1	123	2	0
Yes	1	5	22	1
No	6	62	1	1
No	6	58	54	0
No	16	6	3	0

Table 1.2 The Result of the Expression $\pi_{a_2, a_3} \sigma_{a_1 = \text{“Yes”} \wedge a_4 > 6} S$ Based on the Table 1.1.

a_2	a_3
17	4
17	4
1	5

assigns each possible example to the concept or not. Thus, a concept can be regarded as a function from the Instance space to the Boolean set, namely: $c : X \rightarrow \{-1, 1\}$. Alternatively one can refer a concept c as a subset of X , namely: $\{x \in X : c(x) = 1\}$. A *concept class* C is a set of concepts.

To learn a concept is to infer its general definition from a set of examples. This definition may be either explicitly formulated or left implicit, but either way it assigns each possible example to the concept or not. Thus, a concept can be formally regarded as a function from the set of all possible examples to the Boolean set $\{\text{True}, \text{False}\}$.

Other communities, such as the KDD community prefer to deal with a straightforward extension of *Concept Learning*, known as The *Classification Problem*. In this case we search for a function that maps the set of all possible examples into a predefined set of class labels which are not limited to the Boolean set. Most frequently the goal of the Classifiers Inducers is formally defined as:

Given a training set S with input attributes set $A = \{a_1, a_2, \dots, a_n\}$ and a nominal target attribute y from an unknown fixed distribution D over the labeled instance space, the goal is to induce an optimal classifier with minimum generalization error.

The Generalization error is defined as the misclassification rate over the distribution D . In case of the nominal attributes it can be expressed as:

$$\varepsilon(I(S), D) = \sum_{(x,y) \in U} D(x,y) \cdot L(y, I(S)(x))$$

where $L(y, I(S)(x))$ is the zero one loss function defined as:

$$L(y, I(S)(x)) = \begin{cases} 0 & \text{if } y = I(S)(x) \\ 1 & \text{if } y \neq I(S)(x) \end{cases} \quad (1.1)$$

In case of numeric attributes the sum operator is replaced with the integration operator.

Consider the training set in Table 1.3 containing data concerning about ten customers. Each customer is characterized by three attributes: Age, Gender and “Last Reaction” (an indication whether the customer has positively responded to the last previous direct mailing campaign). The last attribute (“Buy”) describes whether that customer was willing to purchase a product in the current campaign. The goal is to induce a classifier that most accurately classifies a potential customer to “Buyers” and “Non-Buyers” in the current campaign, given the attributes: Age, Gender, Last Reaction.

Table 1.3 An Illustration of Direct Mailing Dataset.

Age	Gender	Last Reaction	Buy
35	Male	Yes	No
26	Female	No	No
22	Male	Yes	Yes
63	Male	No	Yes
47	Female	No	No
54	Male	No	No
27	Female	Yes	Yes
38	Female	No	Yes
42	Female	Yes	Yes
19	Male	No	No

1.4.4 Induction Algorithms

An *Induction algorithm*, or more concisely an *Inducer* (also known as learner), is an entity that obtains a training set and forms a model that

generalizes the relationship between the input attributes and the target attribute. For example, an inducer may take as an input specific training tuples with the corresponding class label, and produce a *classifier*.

The notation I represents an inducer and $I(S)$ represents a model which was induced by performing I on a training set S . Using $I(S)$ it is possible to predict the target value of a tuple x_q . This prediction is denoted as $I(S)(x_q)$.

Given the long history and recent growth of the field, it is not surprising that several mature approaches to induction are now available to the practitioner.

Classifiers may be represented differently from one inducer to another. For example, C4.5 [Quinlan (1993)] represents model as a decision tree while Naïve Bayes [Duda and Hart (1973)] represents a model in the form of probabilistic summaries. Furthermore, inducers can be deterministic (as in the case of C4.5) or stochastic (as in the case of back propagation)

The classifier generated by the inducer can be used to classify an unseen tuple either by explicitly assigning it to a certain class (Crisp Classifier) or by providing a vector of probabilities representing the conditional probability of the given instance to belong to each class (Probabilistic Classifier). Inducers that can construct Probabilistic Classifiers are known as Probabilistic Inducers. In this case it is possible to estimate the conditional probability $\hat{P}_{I(S)}(y = c_j | a_i = x_{q,i} ; i = 1, \dots, n)$ of an observation x_q . Note the addition of the “hat” — $\hat{}$ — to the conditional probability estimation is used for distinguishing it from the actual conditional probability.

The following sections briefly review some of the major approaches to concept learning: Decision tree induction, Neural Networks, Genetic Algorithms, instance-based learning, statistical methods, Bayesian methods and Support Vector Machines. This review focuses more on methods that have the greatest attention in this book.

1.5 Rule Induction

Rule induction algorithms generate a set of if-then rules that jointly represent the target function. The main advantage that rule induction offers is its high comprehensibility. Most of the Rule induction algorithms are based on the separate and conquer paradigm [Michalski (1983)]. For that reason these algorithms are capable of finding simple axis parallel frontiers, are well suited to symbolic domains, and can often dispose easily of irrelevant

attributes; but they can have difficulty with nonaxisparallel frontiers, and suffer from the fragmentation problem (i.e., the available data dwindles as induction progresses [Pagallo and Huassler (1990)] and the small disjuncts problem i.e., rules covering few training examples have a high error rate [Holte *et al.* (1989)]).

1.6 Decision Trees

A Decision tree is a classifier expressed as a recursive partition of the instance space. A decision tree consists of nodes that form a *Rooted Tree*, meaning it is a Directed Tree with a node called root that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is called internal node or test nodes. All other nodes are called leaves (also known as terminal nodes or decision nodes).

In a decision tree, each internal node splits the instance space into two or more subspaces according to a certain discrete function of the input attributes values. In the simplest and most frequent case each test considers a single attribute, such that the instance space is partitioned according to the attribute's value. In the case of numeric attributes the condition refers to a range.

Each leaf is assigned to one class representing the most appropriate target value. Usually the most appropriate target value is the class with the greatest representation, because selecting this value minimizes the zero-one loss. However if a different loss function is used then a different class may be selected in order to minimize the loss function. Alternatively the leaf may hold a probability vector indicating the probability of the target value having a certain value.

Instances are classified by navigating them from the root of the tree down to a leaf, according to the outcome of the tests along the path.

Figure 1.2 describes a decision tree to the classification problem illustrated in Table 1.3 (whether or not a potential customer will respond to a direct mailing). Internal nodes are represented as circles whereas leaves are denoted as triangles. The node "Last R" stands for the attribute "Last Reaction". Note that this decision tree incorporates both nominal and numeric attributes. Given this classifier, the analyst can predict the response of a potential customer (by sorting it down the tree), and understand the behavioral characteristics of the potential customers regarding direct mailing. Each node is labeled with the attribute it tests, and its

branches are labeled with its corresponding values.

In case of numeric attributes, decision trees can be geometrically interpreted as a collection of hyperplanes, each orthogonal to one of the axes.

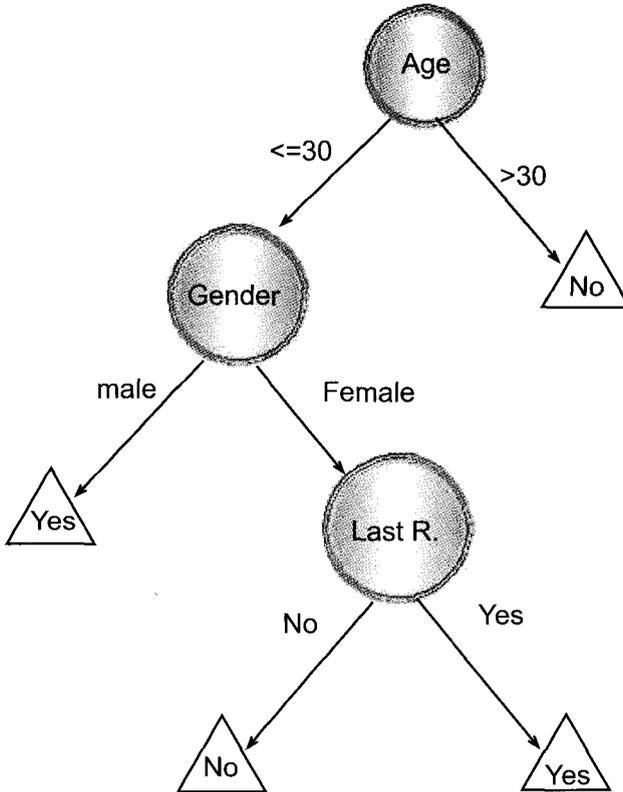


Fig. 1.2 Decision Tree Presenting Response to Direct Mailing.

Naturally, decision makers prefer a less complex decision tree, as it is considered more comprehensible. Furthermore, according to [Breiman *et al.* (1984)] the tree complexity has a crucial effect on its accuracy performance. Usually large trees are obtained by over fitting the data and hence exhibit poor generalization ability. Nevertheless a large decision tree can be accurate if it was induced without over fitting the data. The tree complexity is explicitly controlled by the stopping criteria used and the pruning method employed. Usually the tree complexity is measured by one of the following metrics: The total number of nodes, Total number of leaves, Tree

Depth and Number of attributes used.

Decision tree induction is closely related to rule induction. Each path from the root of a decision tree to one of its leaves can be transformed into a rule simply by conjoining the tests along the path to form the antecedent part, and taking the leaf's class prediction as the class value. For example, one of the paths in Figure 1.2 can be transformed into the rule: "If customer age ≤ 30 , and the gender of the customer is "Male" — then the customer will respond to the mail". The resulting rule set can then be simplified to improve its comprehensibility to a human user, and possibly its accuracy [Quinlan (1987)]. A survey of methods for constructing decision trees can be found in the following chapter.

1.7 Bayesian Methods

1.7.1 Overview

Bayesian approaches employ probabilistic concept representations, and range from the Naïve Bayes [Domingos and Pazzani (1997)] to Bayesian networks. The basic assumption of Bayesian reasoning is that the relation between attributes can be represented as a probability distribution [Maimon and Last (2000)]. Moreover if the problem examined is supervised then the objective is to find the conditional distribution of the target attribute given the input attribute.

1.7.2 Naïve Bayes

1.7.2.1 The Basic Naïve Bayes Classifier

The most straightforward Bayesian learning method is the Naïve Bayesian classifier [Duda and Hart (1973)]. This method uses a set of discriminant functions for estimating the probability of a given instance to belong to a certain class. More specifically it uses Bayes rule to compute the probability of each possible value of the target attribute given the instance, assuming the input attributes are conditionally independent given the target attribute.

Due to the fact that this method is based on the simplistic, and rather unrealistic, assumption that the causes are conditionally independent given the effect, this method is well known as Naïve Bayes.

The predicted value of the target attribute is the one which maximizes the following calculated probability:

$$v_{MAP}(x_q) = \operatorname{argmax}_{c_j \in \operatorname{dom}(y)} \hat{P}(y = c_j) \cdot \prod_{i=1}^n \hat{P}(a_i = x_{q,i} | y = c_j) \quad (1.2)$$

where $\hat{P}(y = c_j)$ denotes the estimation for the *a-priori* probability of the target attribute obtaining the value c_j . Similarly $\hat{P}(a_i = x_{q,i} | y = c_j)$ denotes the conditional probability of the input attribute a_i obtaining the value $x_{q,i}$ given that the target attribute obtains the value c_j . Note that the hat above the conditional probability distinguishes the probability estimation from the actual conditional probability.

A simple estimation for the above probabilities can be obtained using the corresponding frequencies in the training set, namely:

$$\hat{P}(y = c_j) = \frac{|\sigma_{y=c_j} S|}{|S|} \quad ; \quad \hat{P}(a_i = x_{q,i} | y = c_j) = \frac{|\sigma_{y=c_j \text{ AND } a_i=x_{q,i}} S|}{|\sigma_{y=c_j} S|}$$

Using the Bayes rule, the above equations can be rewritten as:

$$v_{MAP}(x_q) = \operatorname{argmax}_{c_j \in \operatorname{dom}(y)} \frac{\prod_{i=1}^n \hat{P}(y=c_j | a_i=x_{q,i})}{\hat{P}(y=c_j)^{n-1}} \quad (1.3)$$

Or alternatively, after using the log function as:

$$\begin{aligned} v_{MAP}(x_q) = & \operatorname{argmax}_{c_j \in \operatorname{dom}(y)} \log \left(\hat{P}(y = c_j) \right) \\ & + \sum_{i=1}^n \left(\log \left(\hat{P}(y = c_j | a_i = x_{q,i}) \right) - \log \left(\hat{P}(y = c_j) \right) \right) \end{aligned}$$

If the “naive” assumption is true, this classifier can easily be shown to be optimal (i.e. minimizing the generalization error), in the sense of minimizing the misclassification rate or zero-one loss (misclassification rate), by a direct application of Bayes’ theorem. [Domingos and Pazzani (1997)] showed that the Naïve Bayes can be optimal under zero-one loss even when the independence assumption is violated by a wide margin. This implies that the Bayesian classifier has a much greater range of applicability than previously thought, for instance for learning conjunctions and disjunctions. Moreover, a variety of empirical research shows surprisingly that this method can perform quite well compared to other methods, even in domains where clear attribute dependencies exist.

The computational complexity of Naïve Bayes is considered very low compared to other methods like decision trees, since no explicit enumeration of possible interactions of various causes is required. More specifically since the Naïve Bayesian classifier combines simple functions of univariate densities, the complexity of this procedure is $O(nm)$.

Furthermore, Naïve Bayes classifiers are also very simple and easy to understand [Kononenko (1990)]. Other advantages of Naïve Bayes are the easy adaptation of the model to incremental learning environments and resistance to irrelevant attributes. The main disadvantage of Naïve Bayes is that it is limited to simplified models only, that in some cases are far from representing the complicated nature of the problem. To understand this weakness, consider a target attribute that cannot be explained by a single attribute, for instance, the Boolean exclusive or function (XOR).

The classification using the Naïve Bayesian classifier is based on all of the available attributes, unless a feature selection procedure is applied as a pre-processing step.

1.7.2.2 *Naïve Bayes for Numeric Attributes*

Originally Naïve Bayes assumes that all input attributes are nominal. If this is not the case then there are some options to bypass this problem:

- (1) Pre-Processing: The numeric attributes should be discretized before using the Naïve Bayes. [Domingos and Pazzani (1997)] suggest discretizing each numeric attribute into ten equal-length intervals (or one per observed value, whichever was the least). Obviously there are many other more informed discretization methods that can be applied here and probably obtain better results.
- (2) Revising the Naïve Bayes: [John and Langley (1995)] suggests using kernel estimation or single variable normal distribution as part of building the conditional probabilities.

1.7.2.3 *Correction to the Probability Estimation*

Using the probability estimation described above as-is will typically overestimate the probability. This can be problematic especially when a given class and attribute value never co-occur in the training set. This case leads to a zero probability that wipes out the information in all the other probabilities terms when they are multiplied according to the original Naïve Bayes equation.

There are two known corrections for the simple probability estimation which avoid this phenomenon. The following sections describe these corrections.

1.7.2.4 Laplace Correction

According to Laplace's law of succession [Niblett (1987)], the probability of the event $y = c_i$ where y is a random variable and c_i is a possible outcome of y which has been observed m_i times out of m observations is:

$$\frac{m_i + k p_a}{m + k}$$

where p_a is an *a-priori* probability estimation of the event and k is the equivalent sample size that determines the weight of the *a-priori* estimation relative to the observed data. According to [Mitchell (1997)] k is called "equivalent sample size" because it represents an augmentation of the m actual observations by additional k virtual samples distributed according to p_a . The above ratio can be rewritten as the weighted average of the *a-priori* probability and the posteriori probability (denoted as p_p):

$$\begin{aligned} & \frac{m_i + k \cdot p_a}{m + k} \\ &= \frac{m_i}{m} \cdot \frac{m}{m + k} + p_a \cdot \frac{k}{m + k} \\ &= p_p \cdot \frac{m}{m + k} + p_a \cdot \frac{k}{m + k} = \\ &= p_p \cdot w_1 + p_a \cdot w_2 \end{aligned}$$

In the case discussed here the following correction is used:

$$\hat{P}_{Laplace}(a_i = x_{q,i} | y = c_j) = \frac{|\sigma_{y=c_j} \text{ AND } a_i = x_{q,i} S| + k \cdot p}{|\sigma_{y=c_j} S| + k} \quad (1.4)$$

In order to use the above correction, the values of p and k should be selected. It is possible to use $p = 1/|\text{dom}(y)|$ and $k = |\text{dom}(y)|$. [Ali and Pazzani (1996)] suggest to use $k = 2$ and $p = 1/2$ in any case even if $|\text{dom}(y)| > 2$ in order to emphasize the fact that the estimated event is always compared to the opposite event. [Kohavi *et al.* (1997)] suggest to use $k = |\text{dom}(y)| / |S|$ and $p = 1/|\text{dom}(y)|$.

1.7.2.5 No Match

According to [Clark and Niblett (1989)] only zero probabilities are corrected and replaced by the following value: $p_a/|S|$. [Kohavi *et al.* (1997)] suggest to use $p_a = 0.5$. They also empirically compared the Laplace correction and the No-Match correction and indicate that there is no significant difference

between them. However, both of them are significantly better than not performing any correction at all.

1.7.3 Other Bayesian Methods

A more complicated model can be represented by Bayesian belief networks [Pearl (1988)]. Usually each node in a Bayesian network represents a certain attribute. The immediate predecessors of a node represent the attributes on which the node depends. By knowing their values, it is possible to determine the conditional distribution of this node. Bayesian networks have the benefit of a clearer semantics than more ad hoc methods, and provide a natural platform for combining domain knowledge (in the initial network structure) and empirical learning (of the probabilities, and possibly of new structure). However, inference in Bayesian networks can have a high time complexity, and as tools for classification learning they are not yet as mature or well tested as other approaches. More generally, as [Buntine (1990)] notes, the Bayesian paradigm extends beyond any single representation, and forms a framework in which many learning tasks can be usefully studied.

1.8 Other Induction Methods

1.8.1 Neural Networks

Neural network methods are based on representing the concept as a network of nonlinear units [Anderson and Rosenfeld (2000)]. The most frequently used type of unit, incorporating a sigmoidal nonlinearity, can be seen as a generalization of a propositional rule, where numeric weights are assigned to antecedents, and the output is graded, rather than binary [Towell and Shavlik (1994)].

The multilayer feedforward neural network is the most widely studied neural network, because it is suitable for representing functional relationships between a set of input attributes and one or more target attributes. Multilayer feedforward neural network consists of interconnected units called neurons, which are organized in layers. Each neuron performs a simple task of information processing by converting received inputs into processed outputs. Figure 1.3 illustrates the most frequently used architecture of feedforward neural network. This network consists of neurons (nodes) organized in three layers: input layer, hidden layer, and output layer. The neurons in the input layer correspond to the input

attributes and the neurons in the output layer correspond to the target attribute. Neurons in the hidden layer are connected to both input and output neurons and are key to inducing the classifier. Note that the signal flow is one directional from the input layer to the output layer and there are no feedback connections.

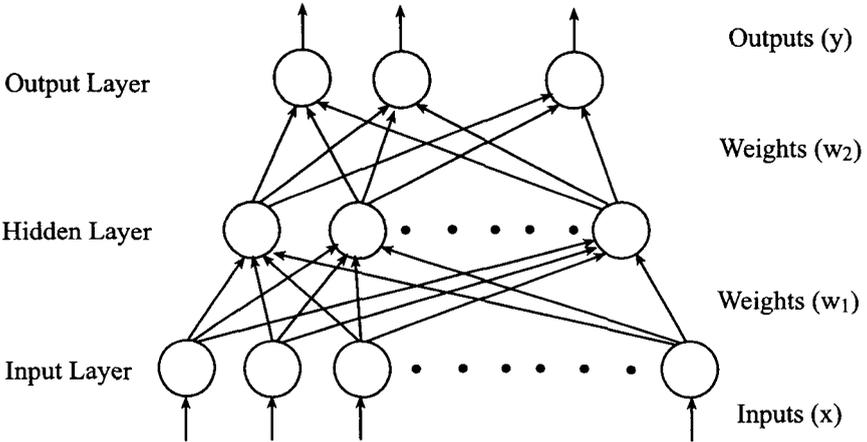


Fig. 1.3 Three-Layer Feedforward Neural Network.

Many search methods can be used to learn these networks, of which the most widely applied one is back propagation [Rumelhart *et al.* (1986)]. This method efficiently propagates values of the evaluation function backward from the output of the network, which then allows the network to be adapted so as to obtain a better evaluation score. Radial basis function (RBF) networks employ units with a Gaussian nonlinearity [Moody and Darken (1989)], and can be seen as a generalization of nearest-neighbor methods with an exponential distance function [Poggio and Girosi (1990)]. Most ANNs are based on a unit called perceptron. A perceptron calculates a linear combination of its inputs, and outputs one of two values as a result. Figure 1.4 illustrates the perceptron. The activation function turns the weighted sum of inputs into a two-value output.

Using a single perceptron, it is possible to realize any decision function that can be represented as a hyper-plane in the input attribute space, so that any instance in one side of the plane is assigned to one class, and instances on the other side of the plane are assigned to the other class. The equation for this hyperplane is:

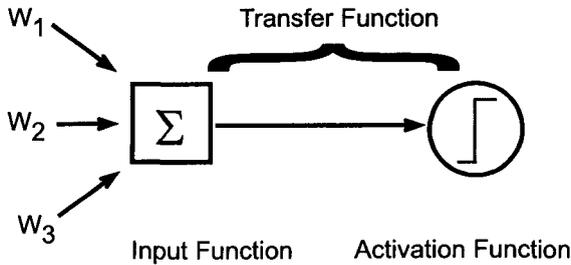


Fig. 1.4 The Perceptron.

$$\sum_{i=1}^n w_i \cdot x_i = 0$$

where each w_i is a real-valued weight, that determines the contribution of each input signal x_i to the perceptron output.

Neural networks are remarkable for their learning efficiency and tend to outperform other methods (like decision trees) when no highly relevant attributes exist, but many weakly relevant ones are present. Furthermore, ANN can easily be adjusted as new examples accumulate.

However according to [Lu *et al.* (1996)], the drawbacks of applying neural networks to data mining include: difficulty in interpreting the model, difficulty in incorporating prior knowledge about the application domain in a neural network, and, also, long learning time, both in terms of CPU time, and of manually finding parameter settings that will enable successful learning. The rule extraction algorithm, described in [Lu *et al.* (1996)], makes an effective use of the neural network structure, though the weights of the links between the nodes remain meaningless, and the rules are extracted in a deterministic (Boolean) form. The network is pruned by removing redundant links and units, but removal of entire attributes (Feature selection) is not considered.

1.8.2 Genetic Algorithms

Genetic algorithms are a search method that can be applied to learning many different representations, of which the most frequently used one is probably rule sets [Booker *et al.* (1989)]. Genetic algorithms maintain a population of classifiers during learning, as opposed to just one, and search

for a better classifier by applying random mutations to them, and exchanging parts between pairs of classifiers that obtain high evaluation scores. This endows them with a potentially greater ability to avoid local minima than is possible with the simple greedy search employed in most learners, but can lead to high computational cost, and to higher risks of finding poor classifiers that appear good on the training data by chance.

1.8.3 Instancebased Learning

Instancebased learning algorithms [Aha *et al.* (1991)] are non-parametric general classification algorithms that simply search for similar instances in the labeled database in order to classify a new unlabeled instance. These techniques are able to induce complex frontiers from relatively few examples and are naturally suited to numeric domains, but can be very sensitive to irrelevant attributes and are unable to select different attributes in different regions of the instance space. Another disadvantage of instance-based methods is that it is relatively time consuming to classify a new instance.

The most basic and simplest Instance-based method is the nearest neighbor (NN) classifier, which was first examined by [Fix and Hodges (1957)]. It can be represented by the following rule: to classify an unknown pattern, choose the class of the nearest example in the training set as measured by a distance metric. A common extension is to choose the most common class in the k nearest neighbors (kNN).

Despite its simplicity, the nearest neighbor classifier has many advantages over other methods. For instance, it can generalize from a relatively small training set. Namely, compared to other methods, such as decision trees or neural network, nearest neighbor requires smaller training examples to provide effective classification. Moreover, it can incrementally add new information at runtime, thus the nearest neighbor can provide a performance that is competitive when compared to more modern methods such as decision trees or neural networks.

1.8.4 Support Vector Machines

Support Vector Machines [Vapnik (1995)] map the input space into a high-dimensional feature space through some non-linear mapping chosen *a-priori* and then construct an optimal separating hyperplan in the new feature space. The method searches for a hyperplan that is optimal according the VC-Dimension theory. Further details and pointers to the literature on

these induction paradigms can be found in the above reference and in the following section.

1.9 Performance Evaluation

Evaluating the performance of an inducer is a fundamental aspect of machine learning. As stated above, an inducer receives a training set as input and constructs a classification model that can classify an unseen instance. Both the classifier and the inducer can be evaluated using an evaluation criteria. The evaluation is important for understanding the quality of the model (or inducer), for refining parameters in the KDD iterative process and for selecting the most acceptable model (or inducer) from a given set of models (or inducers).

There are several criteria for evaluating models and inducers. Naturally, classification models with high accuracy are considered better. However, there are other criteria that can be important as well, such as the computational complexity or the comprehensibility of the generated classifier.

1.9.1 Generalization Error

Let $I(S)$ represent a classifier generated by an inducer I on S . Recall that the generalization error of $I(S)$ is its probability to misclassify an instance selected according to the distribution D of the instance labeled space. The *Classification Accuracy* of a classifier is one minus the generalization error. The *Training Error* is defined as the percentage of examples in the training set correctly classified by the classifier, formally:

$$\hat{\epsilon}(I(S), S) = \sum_{(x,y) \in S} L(y, I(S)(x)) \quad (1.5)$$

where $L(y, I(S)(x))$ is the zero-one loss function defined in Equation 1.1.

In this book, classification accuracy is the primary evaluation criterion for experiments. A decomposition is considered beneficial if the accuracy of an inducer improves or remains the same.

Although generalization error is a natural criterion, its actual value is known only in rare cases (mainly synthetic cases). The reason for that is that the distribution D of the instance labeled space is not known.

One can take the training error as an estimation of the generalization

error. However, using the training error as-is will typically provide an optimistically biased estimate, especially if the learning algorithm *over-fits* the training data. There are two main approaches for estimating the generalization error: Theoretical and Empirical. In the context of this book we utilize both approaches.

1.9.2 Theoretical Estimation of Generalization Error

A low training error does not guarantee low generalization error. There is often a trade-off between the training error and the confidence assigned to the training error as a predictor for the generalization error, measured by the difference between the generalization and training errors. The capacity of the inducer is a determining factor for this confidence in the training error. Indefinitely speaking, the capacity of an inducer indicates the variety of classifiers it can induce. The notion of VC-Dimension presented below can be used as a measure of the inducers capacity.

Inducers with a large capacity, e.g. a large number of free parameters, relative to the size of the training set are likely to obtain a low training error, but might just be memorizing or over-fitting the patterns and hence exhibit a poor generalization ability. In this regime, the low error is likely to be a poor predictor for the higher generalization error. In the opposite regime, when the capacity is too small for the given number of examples, inducers may under-fit the data, and exhibit both poor training and generalization error. For inducers with an insufficient number of free parameters, the training error may be poor, but it is a good predictor for the generalization error. In between these capacity extremes there is an optimal capacity for which the best generalization error is obtained, given the character and amount of the available training data.

In the book “Mathematics of Generalization”, [Wolpert (1995)] discuss four theoretical frameworks for estimating the generalization error, namely: PAC, VC and Bayesian, and Statistical Physics. All these frameworks combine the training error (which can be easily calculated) with some penalty function expressing the capacity of the inducers. In this book we employ the VC framework, described in the next section.

1.9.2.1 VC-Framework

Of all the major theoretical approaches to learning from examples the Vapnik–Chervonenkis theory [Vapnik (1995)] is the most comprehensive,

applicable to regression, as well as classification tasks. It provides general necessary and sufficient conditions for the consistency of the induction procedure in terms of bounds on certain measures. Here we refer to the classical notion of consistency in statistics: both the training error and the generalization error of the induced classifier must converge to the same minimal error value as the training set size tends to infinity. Vapnik's theory also defines a capacity measure of an inducer, the VC-dimension, which is widely used.

VC-theory describes a worst case scenario: the estimates of the difference between the training and generalization errors are bounds valid for any induction algorithm and probability distribution in the labeled space. The bounds are expressed in terms of the size of the training set and the VC-dimension of the inducer.

Theorem 1.1 *The bound on the generalization error of hypothesis space H with finite VC-Dimension d is given by:*

$$|\varepsilon(h, D) - \hat{\varepsilon}(h, S)| \leq \sqrt{\frac{d \cdot (\ln \frac{2m}{d} + 1) - \ln \frac{\delta}{4}}{m}} \quad \begin{array}{l} \forall h \in H \\ \forall \delta > 0 \end{array} \quad (1.6)$$

with probability of $1 - \delta$ where $\hat{\varepsilon}(h, S)$ represents the training error of classifier h measured on training set S of cardinality m and $\varepsilon(h, D)$ represents the generalization error of the classifier h over the distribution D .

The VC dimension is a property of a set of all classifiers, denoted by H , that have been examined by the inducer. For the sake of simplicity we consider classifiers that correspond to the two-class pattern recognition case. In this case, the VC dimension is defined as the maximum number of data points that can be shattered by the set of admissible classifiers. By definition, a set S of m points is shattered by H if and only if for every dichotomy of S there is some classifier in H that is consistent with this dichotomy. In other words, the set S is shattered by H if there are classifiers that split the points into two classes in all of the 2^m possible ways. Note that, if the VC dimension of H is d , then there exists at least one set of d points that can be shattered by H , but in general it will not be true that every set of d points can be shattered by H .

A sufficient condition for consistency of an induction procedure is that the VC-dimension of the inducer is finite. The VC-dimension of a linear classifier is simply the dimension n of the input space, or the number of free parameters of the classifier. The VC-dimension of a general classifier

may however be quite different from the number of free parameters and in many cases it might be very difficult to compute it accurately. In this case it is useful to calculate a lower and upper bound for the VC-Dimension, for instance [Schmitt (2002)] have presented these VC bounds for neural networks.

1.9.2.2 PAC-Framework

The Probably Approximately Correct (PAC) learning model was introduced by [Valiant (1984)]. This framework can be used to characterize the concept class “that can be reliably learned from a reasonable number of randomly drawn training examples and a reasonable amount of computation” [Mitchell (1997)]. We use the following formal definition of PAC-learnable adapted from [Mitchell (1997)]:

Definition 1.1 Let C be a concept class defined over the input instance space X with n attributes. Let I be an inducer that considers hypothesis space H . C is said to be PAC-learnable by I using H if for all $c \in C$, distributions D over X , ε such that $0 < \varepsilon < 1/2$ and δ such that $0 < \delta < 1/2$, learner I with a probability of at least $(1 - \delta)$ will output a hypothesis $h \in H$ such that $\varepsilon(h, D) \leq \varepsilon$, in time that is polynomial in $1/\varepsilon$, $1/\delta$, n , and $size(c)$, where $size(c)$ represents the encoding length of c in C , assuming some representation for C .

The PAC learning model provides a general bound on the number of training examples sufficient for any consistent learner I examining a finite hypothesis space H with probability at least $(1 - \delta)$ to output a hypothesis $h \in H$ within error ε of the target concept $c \in C \subseteq H$. More specifically, the size of the training set should be: $m \geq \frac{1}{\varepsilon}(\ln(1/\delta) + \ln |H|)$

1.9.3 Empirical Estimation of Generalization Error

Another approach for estimating the generalization error is to split the available examples into two groups: training set and test set. First, the training set is used by the inducer to construct a suitable classifier and then we measure the misclassification rate of this classifier on the test set. This test set error usually provides a better estimation to the generalization error than the training error. The reason for that is the fact that the training error usually under-estimates the generalization error (due to the overfitting phenomena).

When data is limited, it is common practice to *resample* the data, that is, partition the data into training and test sets in different ways. An inducer is trained and tested for each partition and the accuracies averaged. By doing this, a more reliable estimate of the true generalization error of the inducer is provided.

Random subsampling and *n-fold cross-validation* are two common methods of resampling. In random subsampling, the data is randomly partitioned into disjoint training and test sets several times. Errors obtained from each partition are averaged. In *n-fold cross-validation*, the data is randomly split into n mutually exclusive subsets of approximately equal size. An inducer is trained and tested n times; each time it is tested on one of the k folds and trained using the remaining $n - 1$ folds.

The cross-validation estimate of the generalization error is the overall number of misclassifications, divided by the number of examples in the data. The random subsampling method has the advantage that it can be repeated an indefinite number of times. However, it has the disadvantage that the test sets are not independently drawn with respect to the underlying distribution of examples. Because of this, using a *t-test* for paired differences with random subsampling can lead to increased chance of Type I error that is, identifying a significant difference when one does not actually exist. Using a *t-test* on the generalization error produced on each fold has a lower chance of Type I error but may not give a stable estimate of the generalization error. It is common practice to repeat n fold cross-validation n times in order to provide a stable estimate. However, this of course renders the test sets non-independent and increases the chance of Type I error. Unfortunately, there is no satisfactory solution to this problem. Alternative tests suggested by [Dietterich (1998)] have low chance of Type I error but high chance of Type II error — that is, failing to identify a significant difference when one does actually exist.

Stratification is a process often applied during random subsampling and *n-fold crossvalidation*. Stratification ensures that the class distribution from the whole dataset is preserved in the training and test sets. Stratification has been shown to help reduce the variance of the estimated error especially for datasets with many classes. Stratified random subsampling with a paired *t-test* is used herein to evaluate accuracy.