

Chapter 1

Introduction to Pattern Classification

Pattern recognition is the scientific discipline whose purpose is to classify patterns (also known as instances, tuples and examples) into a set of categories which are also referred to as *classes* or *labels*. Commonly, the classification is based on statistical models that are induced from an exemplary set of preclassified patterns. Alternatively, the classification utilizes knowledge that is supplied by an expert in the application domain.

A pattern is usually composed of a set of measurements that characterize a certain object. For example, suppose we wish to classify flowers from the Iris genus into their subgeni (such as Iris Setosa, Iris Versicolour and Iris Virginica). The patterns in this case will consist the flowers features, such as the length and the width of the sepal and petal. The label of every instance will be one of the strings *Iris Setosa*, *Iris Versicolour* and *Iris Virginica*. Alternatively, the labels can take a value from 1,2,3, a,b,c or any other set of three distinct values.

Another common application which employs pattern recognition is Optical character recognition (OCR). These applications convert scanned documents into machine-editable text in order to simplify their storage and retrieval. Each document undergoes three steps. First, an operator scans the document. This converts the document into a bitmap image. Next, the scanned document is segmented such that each character is isolated from the others. Then, a *feature extractor* measures certain features of each character such as open areas, closed shapes, diagonal lines and line intersections. Finally, the scanned characters are associated with their corresponding alpha-numeric character. The association is obtained by applying a pattern recognition algorithm to the features of the scanned characters. In this case, the set of labels/categories/classes are the set of alpha-numeric character i.e. letters, numbers, punctuation marks, etc.

1.1 Pattern Classification

In a typical statistical pattern recognition setting, a set of patterns S , also referred to as a *training set* is given. The labels of the patterns in S are known and the goal is to construct an algorithm in order to label new patterns. A classification algorithm is also known as an *inducer* and an instance of an inducer for a specific training set is called a *classifier*.

The training set can be described in a variety of ways. Most frequently, each pattern is described by a vector of *feature* values. Each vector belongs to a single class and associated with the class label. In this case, the training set is stored in a table where each row consists of a different pattern. Let A and y denote the set of n features: $A = \{a_1, \dots, a_i, \dots, a_n\}$ and the class label, respectively.

Features, which are also referred to as attributes, typically fall into one of the following two categories:

Nominal the values are members of an unordered set. In this case, it is useful to denote its domain values by $dom(a_i) = \{v_{i,1}, v_{i,2}, \dots, v_{i,|dom(a_i)|}\}$, where $|dom(a_i)|$ is the finite cardinality of the domain.

Numeric the values are real numbers. Numeric features have infinite cardinalities.

In a similar way, $dom(y) = \{c_1, c_2, \dots, c_k\}$ constitutes the set of labels. Table 1.1 illustrates a segment of the Iris dataset. This is one of the best known datasets in the pattern recognition literature. It was first introduced by R. A. Fisher (1936). The goal in this case is to classify flowers into the Iris subgeni according to their characteristic features.

The dataset contains three classes that correspond to three types of iris flowers: $dom(y) = \{IrisSetosa, IrisVersicolor, IrisVirginica\}$. Each pattern is characterized by four numeric features (measured in centimeters): $A = \{sepal\ length, sepal\ width, petal\ length, petal\ width\}$.

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 denoted by $S(B)$ and it is composed of m tuples.

Table 1.1 The Iris Dataset Consisting of Four Numeric Features and Three Possible Classes.

Sepal Length	Sepal Width	Petal Length	Petal Width	Class (Iris Type)
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
6.0	2.7	5.1	1.6	Iris-versicolor
5.8	2.7	5.1	1.9	Iris-virginica
5.0	3.3	1.4	0.2	Iris-setosa
5.7	2.8	4.5	1.3	Iris-versicolor
5.1	3.8	1.6	0.2	Iris-setosa
⋮				

$$S(B) = (\langle x_1, y_1 \rangle, \dots, \langle x_m, y_m \rangle) \quad (1.1)$$

where $x_q \in X$ and $y_q \in \text{dom}(y)$, $q = 1, \dots, m$.

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

As mentioned above, the goal of an inducer is to generate classifiers. In general, a classifier partitions the instance space according to the labels of the patterns in the training set. The borders separating the regions are called *frontiers* and inducers construct frontiers such that new patterns will be classified into the correct region. Specifically, 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 as defined above, the objective is to induce an optimal classifier with a minimum generalization error.

The generalization error is defined as the misclassification rate over the distribution D . Formally, let I be an inducer. We denote by $I(S)$ the classifier that is generated by I for the training set S . The classification that is produced by $I(S)$ when it is applied to a pattern x is denoted by $I(S)(x)$. In case of nominal attributes, the generalization error be expressed as:

$$\varepsilon(I(S), D) = \sum_{\langle x, y \rangle \in U} D(x, y) \cdot L(y, I(S)(x)) \quad (1.2)$$

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

$$L(y, I(S)(x)) = \begin{cases} 0 & \text{if } y = I(S)(x) \\ 1 & \text{if } y \neq I(S)(x) \end{cases} \quad (1.3)$$

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

1.2 Induction Algorithms

An *induction algorithm*, or more concisely an *inducer* (also known as learner), is an algorithm that is given a training set and constructs a model that generalizes the connection between the input attributes and the target attribute. For example, an inducer may take as input specific training patterns with their corresponding class labels, and produce a *classifier*.

Let I be an inducer. We denote by $I(S)$ the classifier which is induced by applying I to the training set S . Using $I(S)$ it is possible to predict the target value of a pattern x . This prediction is denoted as $I(S)(x)$.

Given the on going fruitful research and recent advances in the field of pattern classification, it is not surprising to find several mature approaches to induction that are now available to the practitioner.

An essential component of most classifiers is model which specifies how a new pattern is classified. These models are represented differently by different inducers. For example, C4.5 [Quinlan (1993)] represents the model as a decision tree while the Naïve Bayes [Duda and Hart (1973)] inducer 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)

There two ways in which a new pattern can be classified. The classifier can either explicitly assign a certain class to the pattern (Crisp Classifier) or, alternatively, the classifier can produce a vector of the conditional probability the given pattern belongs to each class (Probabilistic Classifier). In this case it is possible to estimate the conditional probability $\hat{P}_{I(S)}(y = c_j | a_i = x_{q,i} ; i = 1, \dots, n)$ for an observation x_q . Note the addition of the “hat” to the conditional probability estimation is used for distinguishing it from the actual conditional probability. Inducers that can construct Probabilistic Classifiers are known as *Probabilistic Inducers*.

The following sections briefly review some of the common 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 on methods that are described in details in this book.

1.3 Rule Induction

Rule induction algorithms generate a set of *if-then* rules that describes the classification process. The main advantage of this approach is its high comprehensibility. Namely, the rules can be written as a collection of consecutive conditional statements in plain English which are easy to employ. Most of the Rule induction algorithms are based on the separate and conquer paradigm [Michalski (1983)]. Consequently, these algorithms: (a) are capable of finding simple axis parallel frontiers; (b) well suited for symbolic domains; and (c) can often dispose easily of irrelevant attributes. However, Rule induction algorithms can have difficulty when non-axis-parallel frontiers are required to correctly classify the data. Furthermore, they suffer from the *fragmentation problem*, i.e., the available data dwindles as induction progresses [Pagallo and Huassler (1990)]. Another pitfall that should be avoided is the small disjuncts problem or *emphoverfitting*. This problem is characterized by rules that cover a very small number of training patterns. Thus, the model fits the training data very well, however, it fails to classify new patterns, resulting in a high error rate [Holte *et al.* (1989)].

1.4 Decision Trees

A Decision tree is a classifier whose model forms a recursive partition of the instance space. The model is described as a rooted tree, i.e., a direct tree with a node called a “root” that has no incoming edges. All other nodes have exactly one incoming edge. A node with outgoing edges is referred to as an “internal” node or a “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 sub-spaces according to a certain discrete function of the input attribute values. In the simplest and most frequent case, each test considers a single attribute, such that the instance space is partitioned according to the attributes value. In the case of numeric attributes, the condition refers to a range.

Each leaf is assigned to one class corresponding to the most appropriate target value. Alternatively, the leaf may hold a probability vector (affinity

vector) whose elements indicate the probabilities of the target attribute to assume a certain value. Figure 1.1 describes an example of a decision tree that solves the Iris recognition task presented in Table 1.1.

Internal nodes are represented as circles, whereas leaves are denoted by triangles. Each internal node (not a leaf) may have two or more outgoing branches. Each node corresponds to a certain property and the branches correspond to a range of values. These value ranges must partition the set of values of the given property.

Instances are classified by traversing the tree starting from the root down to a leaf where the path is determined according to the outcome of the partitioning condition at each node. Specifically, we start at the root of a tree and consider the property that corresponds to the root. We then find to which outgoing branch the observed value of the given property corresponds. The next node in our path is the one at the end of the chosen branch. We repeat the same operations for this node and traverse the tree until we reach a leaf.

Note that decision trees can incorporate both nominal and numeric attributes. In case of numeric attributes, decision trees can be geometrically interpreted as a collection of hyperplanes, each orthogonal to one of the axes.

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 performance accuracy. Usually, large trees are obtained by over fitting the data and hence exhibit poor generalization ability (a pitfall they share with Rule Classifiers). Nevertheless, a large decision tree can generalize well to new patterns if it was induced without over fitting the data. The tree complexity is explicitly controlled by the stopping criteria used for the construction of the tree and the pruning method that is employed. Common measures for the tree complexity include the following metrics: (i) The total number of nodes; (ii) Total number of leaves; (iii) Tree Depth; and (iv) The 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. The resulting rule set can then be simplified in order to improve its comprehensibility to a human user, and to improve its accuracy [Quinlan (1987)].

Decision tree inducers are algorithms that automatically construct a

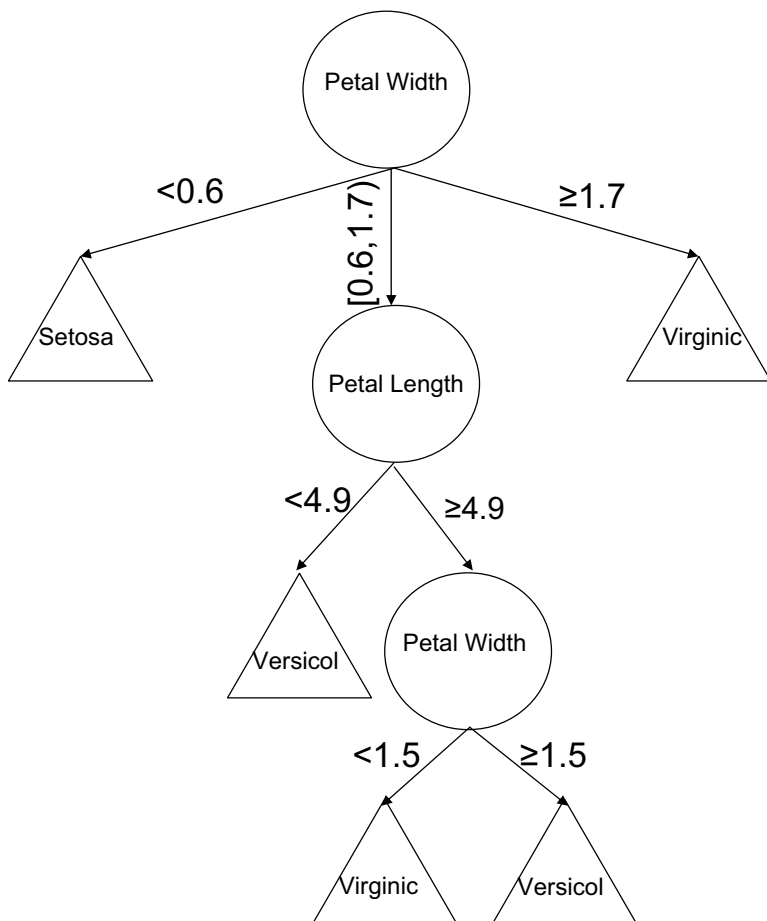


Fig. 1.1 Decision tree for solving the Iris classification task.

decision tree from a given dataset. Typically, the goal is to find the optimal decision tree by minimizing the generalization error. However, other target functions can also be defined, for instance, minimizing the number of nodes or minimizing the average depth of the tree.

Construction of an *optimal* decision tree based on a given dataset is considered to be a difficult task. [Hancock *et al.* (1996)] have shown that finding a *minimal* decision tree for a given training set is NP-hard while [Hyafil and Rivest (1976)] have proved that constructing a minimal binary tree with respect to the expected number of tests required for classifying

an unseen instance is NP-complete. Even finding the minimal equivalent decision tree for a given decision tree [Zantema and Bodlaender (2000)] or building the optimal decision tree from decision tables is known to be NP-hard [Naumov (1991)].

These results indicate that optimal decision tree algorithms are only suitable for very small datasets and a very small number of attributes. Consequently, heuristic methods are required for solving this problem. Roughly speaking, these methods can be divided into two groups: methods that employ a top-down approach and methods that follow a bottom-up methodology with clear preference in the literature to the first group.

There are various top-down decision trees inducers such as ID3 [Quinlan (1986)], C4.5 [Quinlan (1993)], CART [Breiman *et al.* (1984)]. Some inducers consist of two conceptual phases: *Growing* and *Pruning* (C4.5 and CART). Other inducers perform only the growing phase.

Figure 1.2 shows an example of typical pseudo code for a top-down inducing algorithm of a decision tree using growing and pruning. Note that these algorithms are greedy by nature and construct the decision tree in a top-down, recursive manner (also known as divide and conquer). In each iteration, the algorithm considers the partition of the training set using the outcome of discrete input attributes. The selection of the most appropriate attribute is made according to a splitting measure. After an appropriate split is selected, each node further divides the training set into smaller subsets, until a stopping criterion is met.

Common stopping rules include:

- (1) All instances in the training set belong to a single value of y .
- (2) A maximum tree depth has been reached.
- (3) The number of cases in the terminal node is less than the minimum number of cases for parent nodes.
- (4) In case a node is split: the number of cases in one or more child nodes is less than the minimum number of cases for child nodes.
- (5) The best splitting criterion is not greater than a certain threshold.

1.5 Bayesian Methods

1.5.1 Overview

Bayesian approaches employ probabilistic concept modeling, and range from the Naïve Bayes [Domingos and Pazzani (1997)] to Bayesian net-

works. The basic assumption of Bayesian reasoning is that the attributes are connected via a probability. Moreover, when the problem at hand is supervised, the objective is to find the conditional distribution of the target attribute given the input attribute.

1.5.2 *Naïve Bayes*

1.5.2.1 *The Basic Naïve Bayes Classifier*

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

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.

TreeGrowing ($S, A, y, SplitCriterion, StoppingCriterion$)

Where:

S - Training Set

A - Input Feature Set

y - Target Feature

$SplitCriterion$ - the method for evaluating a certain split

$StoppingCriterion$ - the criteria to stop the growing process

Create a new tree T with a single root node.

IF $StoppingCriterion(S)$ THEN

 Mark T as a leaf with the most
 common value of y in S as a label.

ELSE

$\forall a_i \in A$ find a that obtain the best $SplitCriterion(a_i, S)$.

 Label t by a

 FOR each outcome v_i of a :

 Set $Subtree_i = TreeGrowing(\sigma_{a=v_i} S, A, y)$.

 Connect the root node of t_T to $Subtree_i$ with
 an edge that is labeled as v_i

 END FOR

END IF

RETURN TreePruning (S, T, y)

TreePruning (S, T, y)

Where:

S - Training Set

y - Target Feature

T - The tree to be pruned

DO

 Select a node t in T such that pruning it
 maximally improve some evaluation criteria

 IF $t \neq \emptyset$ THEN $T = pruned(T, t)$

UNTIL $t = \emptyset$

RETURN T

Fig. 1.2 Top-down algorithmic framework for decision trees induction.

The class of the instance is determined according to value of the target attribute which maximizes the following calculated probability:

$$v_{MAP}(x_q) = \operatorname{argmax}_{c_j \in \operatorname{dom}(y)} \hat{P}(y = c_j) \cdot \prod_{i=1}^n \hat{P}(a_i = x_{q,i} | y = c_j) \quad (1.4)$$

where $\hat{P}(y = c_j)$ denotes the estimation of the *a-priori* probability of the target attribute to obtain 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 to obtain 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:

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

where $|\sigma_{y=c_j} S|$ denotes the number of instances in S for which $y = c_j$. Using the Bayes rule, the above equations can be rewritten as:

$$v_{MAP}(x_q) = \operatorname{argmax}_{c_j \in \operatorname{dom}(y)} \frac{\prod_{i=1}^n \hat{P}(y=c_j | a_i=x_{q,i})}{\hat{P}(y=c_j)^{n-1}} \quad (1.5)$$

Or alternatively, after applying 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, by a direct application of Bayes’ Theorem, 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). It was shown in [Domingos and Paz-zani (1997)] that the Naïve Bayes inducer 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 originally assumed, for instance for learning conjunctions and disjunctions. Moreover, numerous empirical results show 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 include easy adaptation of the model to incremental learning environments and robustness to irrelevant attributes. The main disadvantage of the Naïve Bayes inducer is that it is limited to only simplified models, which in some cases are incapable of 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 Naïve Bayesian classifier uses all the available attributes, unless a feature selection procedure is applied as a pre-processing step.

1.5.2.2 *Naïve Bayes Induction 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 are discretized before using the Naïve Bayes approach. It is suggested in [Domingos and Pazzani (1997)] to construct ten equi-length intervals for each numeric attribute (or one per observed value, whichever produces the least number of possible values). Each attribute value will be assigned an interval number. Obviously, there are many other more context-aware 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 the conditional probabilities calculation.

1.5.2.3 *Correction to the Probability Estimation*

Using the probability estimation described above as-is will typically over-estimate (similar to over-fitting in decision trees) 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 circumvent this phenomenon. The following sections describe these corrections.

1.5.2.4 Laplace Correction

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

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

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:

$$\hat{P}_{Laplace}(a_i = x_{q,i} | y = c_j) = \frac{|\sigma_{y=c_j \text{ AND } a_i=x_{q,i}} S| + k \cdot p}{|\sigma_{y=c_j} S| + k} \quad (1.6)$$

In order to use the above correction, the values of p and k should be determined. There are several possibilities to determine their values. It is possible to use $p = 1/|\text{dom}(y)|$ and $k = |\text{dom}(y)|$. In [Ali and Pazzani (1996)] it is suggested 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. Another option is to use $k = |\text{dom}(y)| / |S|$ and $p = 1/|\text{dom}(y)|$ [Kohavi *et al.* (1997)].

1.5.2.5 *No Match*

According to [Clark and Niblett (1989)] only zero probabilities should be corrected and replaced by the following value: $p_a/|S|$ where [Kohavi *et al.* (1997)] suggest to use $p_a = 0.5$. An empirical comparison of Laplace correction and No-Match correction indicates that there is no significant difference between them. However, both of them are significantly better than not performing any correction at all.

1.5.3 *Other Bayesian Methods*

A more sophisticated Bayesian-based model that can be used is 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 they provide a natural platform for combining domain knowledge (in the initial network structure) and empirical learning (of the probabilities, and possibly of a new structure). However, time complexity of inference in Bayesian networks can be high, 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.

1.6 Other Induction Methods

1.6.1 *Neural Networks*

Neural network methods construct a model using a network of interconnected units called neurons [Anderson and Rosenfeld (2000)]. The neurons are connected in an input/output manner i.e. the output of one neuron (antecedent) is the input of another (descendant). A neuron may have several antecedents and several descendants (including itself in some settings). Every unit performs a simple data processing task by generating an output from the received inputs. The task is usually obtained via a nonlinear function. The most frequently used type of unit, incorporating 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. In a multilayer feedforward neural network the neurons are organized in layers. Figure 1.3 illustrates a typical feedforward neural network. This network consists of neurons (also referred to as nodes) organized in three layers: an input layer, a hidden layer and an 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. The neurons in the hidden layer are connected to both the input and the output neurons and they are the key to the induction of the classifier. Note that the signal flow is directed from the input layer to the output layer and there are no loops.

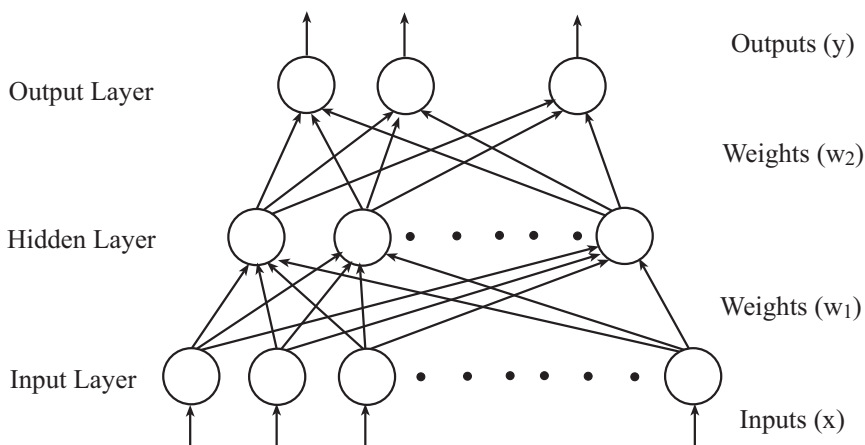


Fig. 1.3 Three-layer feedforward neural network.

In order to construct a classifier from a neural network inducer, a *training* step must be employed. The training step calculates the connection weights which optimize a given evaluation function of the training data. Various search methods can be used to train these networks, of which the most widely applied one is back propagation [Rumelhart *et al.* (1986)]. This method efficiently propagates values of the output evaluation function

backward to the input, allowing the network weights to be adapted so as to obtain a better evaluation score. Radial basis function (RBF) networks employ Gaussian nonlinearity in the neurons [Moody and Darken (1989)], and can be seen as a generalization of nearestneighbor methods with an exponential distance function [Poggio and Girosi (1990)].

Most neural networks are based on a unit called a *perceptron*. A perceptron performs the following: (a) it calculates a linear combination of its inputs; and (b) it invokes an activation function which transforms the weighted sum into a binary output. Figure 1.4 illustrates the perceptron.

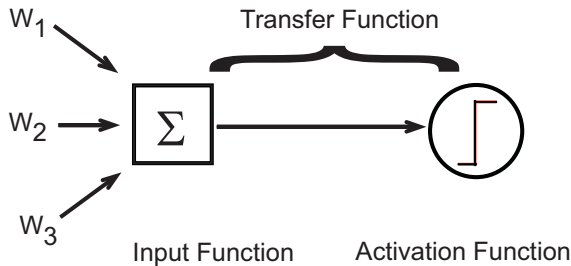


Fig. 1.4 The perceptron.

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

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

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 the information required for the classification is not concentrated in a small subset of the attributes, but rather it is spread across many of the attributes. Furthermore, neural networks can be trained incrementally i.e. they can easily be adjusted as new training examples become available.

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, and, also, long training time, both in terms of CPU time, and of manually finding parameter settings that will enable successful learning i.e. optimize the evaluation function. The rule extraction algorithm, described in [Lu *et al.* (1996)], makes an effective use of the neural network structure. The algorithm extracts the (Boolean) rules in a deterministic manner without using the connection weights. The network is then pruned by removal of redundant links and units with the exception of attributes (Feature selection) whose removal is not considered.

1.6.2 Genetic Algorithms

Genetic algorithms are a collection of search methods that can be used to train a wide variety of models. - of which the most frequently used one is probably *rule sets* [Booker *et al.* (1989)]. Genetic algorithms maintain a population of classifiers during the training, as opposed to just one in other search methods. They employ an iterative process whose goal is to find an optimal classifier by improving the evaluation performance of the classifier population. In order to achieve this, pairs of classifiers that achieve better performance are chosen. Random mutations are plied to the classifier pair and part are exchanged between them. This process has a lower chance to reach a local minima than simple greedy search employed in most learners do. However, this process may incur a high computational cost. Furthermore, there is a higher risk of producing poor classifiers that accidentally perform well on the training data.

1.6.3 Instance-based Learning

Instance-based learning algorithms [Aha *et al.* (1991)] are non-parametric general classification algorithms that classify a new unlabeled instance according to the labels of similar instances in the training set. At the core of these algorithms, there is a simple search procedure. These techniques are able to induce complex classifiers from a relatively small number of examples and are naturally suited to numeric domains. However, they can be very sensitive to irrelevant attributes and are unable to select different attributes in different regions of the instance space. Furthermore, although (or more accurately *because*) the time complexity to train these models is low, it is relatively time consuming to classify a new instance.

The most basic and simplest Instance-based method is the nearest neighbor (NN) inducer, 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 given 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, the nearest neighbor classifier requires smaller training examples to achieve the same classification performance. Moreover, new information can be incrementally incorporated at runtime a property it shares with neural networks. consequently, the nearest neighbor classifier can achieve a performance that is competitive to more modern and complex methods such as decision trees or neural networks.

1.6.4 *Support Vector Machines*

Support Vector Machines [Vapnik (1995)] map the input space into a high-dimensional feature space through a non-linear mapping that is chosen *a-priori*. An optimal separating hyperplane is then constructed in the new feature space. The method searches for a hyperplane that is optimal according to the VC-Dimension theory.