Supervised Machine Learning: A Review of Classification Techniques

S. B. Kotsiantis
Department of Computer Science and Technology
University of Peloponnese, Greece
End of Karaiskaki, 22100 , Tripolis GR.
Tel: +30 2710 372164
Fax: +30 2710 372160
E-mail: sotos@math.upatras.gr

Overview paper

Keywords: classifiers, data mining techniques, intelligent data analysis, learning algorithms

Received: July 16, 2007

Supervised machine learning is the search for algorithms that reason from externally supplied instances to produce general hypotheses, which then make predictions about future instances. In other words, the goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features. The resulting classifier is then used to assign class labels to the testing instances where the values of the predictor features are known, but the value of the class label is unknown. This paper describes various supervised machine learning classification techniques. Of course, a single article cannot be a complete review of all supervised machine learning classification algorithms (also known induction classification algorithms), yet we hope that the references cited will cover the major theoretical issues, guiding the researcher in interesting research directions and suggesting possible bias combinations that have yet to be explored.

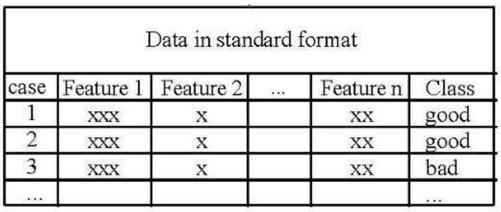
Povzetek: Podan je pregled metod strojnega učenja.

1 Introduction

There are several applications for Machine Learning (ML), the most significant of which is data mining. People are often prone to making mistakes during analyses or, possibly, when trying to establish relationships between multiple features. This makes it difficult for them to find solutions to certain problems. Machine learning can often be successfully applied to these problems, improving the efficiency of systems and the designs of machines.

Every instance in any dataset used by machine learning algorithms is represented using the same set of features. The features may be continuous, categorical or binary. If instances are given with known labels (the corresponding correct outputs) then the learning is called supervised (see Table 1), in contrast to unsupervised learning, where instances are unlabeled. By applying these unsupervised (clustering) algorithms, researchers hope to discover unknown, but useful, classes of items (Jain et al., 1999). Another kind of machine learning is reinforcement learning (Barto & Sutton, 1997). The training information provided to the learning system by the environment (external trainer) is in the form of a scalar reinforcement signal that constitutes a measure of how well the system operates. The learner is not told which actions to take, but rather must discover which actions yield the best reward, by trying each action in turn.

Numerous ML applications involve tasks that can be set up as supervised. In the present paper, we have concentrated on the techniques necessary to do this. In particular, this work is concerned with classification problems in which the output of instances admits only discrete, unordered values.



Data in standard format					
case	Feature 1	Feature 2	...	Feature n	Class
1	xxx	x		xx	good
2	xxx	x		xx	good
3	xxx	x		xx	bad
...					...

Table 1. Instances with known labels (the corresponding correct outputs)

We have limited our references to recent refereed journals, published books and conferences. In addition, we have added some references regarding the original work that started the particular line of research under discussion. A brief review of what ML includes can be found in (Dutton & Conroy, 1996). De Mantaras and Armengol (1998) also presented a historical survey of logic and instance based learning classifiers. The reader should be cautioned that a single article cannot be a comprehensive review of all classification learning algorithms. Instead, our goal has been to provide a representative sample of existing lines of research in each learning technique. In each of our listed areas, there are many other papers that more comprehensively detail relevant work.

Our next section covers wide-ranging issues of supervised machine learning such as data pre-processing and feature selection. Logical/Symbolic techniques are described in section 3, whereas perceptron-based techniques are analyzed in section 4. Statistical techniques for ML are covered in section 5. Section 6 deals with instance based learners, while Section 7 deals with the newest supervised ML technique—Support Vector Machines (SVMs). In section 8, some general directions are given about classifier selection. Finally, the last section concludes this work.

2 General issues of supervised learning algorithms

Inductive machine learning is the process of learning a set of rules from instances (examples in a training set), or more generally speaking, creating a classifier that can be used to generalize from new instances. The process of applying supervised ML to a real-world problem is described in Figure 1.

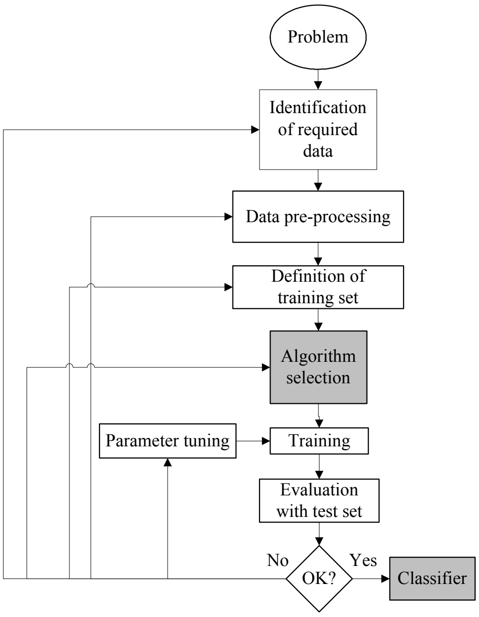


Figure 1. The process of supervised ML

The first step is collecting the dataset. If a requisite expert is available, then s/he could suggest which fields (attributes, features) are the most informative. If not, then the simplest method is that of “brute-force,” which means measuring everything available in the hope that the right (informative, relevant) features can be isolated. However, a dataset collected by the “brute-force” method is not directly suitable for induction. It contains in most cases noise and missing feature values, and therefore requires significant pre-processing (Zhang et al., 2002).

The second step is the data preparation and data pre-processiong. Depending on the circumstances, researchers have a number of methods to choose from to handle missing data (Batista & Monard, 2003). Hodge & Austin (2004) have recently introduced a survey of contemporary techniques for outlier (noise) detection. These researchers have identified the techniques' advantages and disadvantages. Instance selection is not only used to handle noise but to cope with the infeasibility of learning from very large datasets. Instance selection in these datasets is an optimization problem that attempts to maintain the mining quality while minimizing the sample size (Liu and Motoda, 2001). It reduces data and enables a data mining algorithm to function and work effectively with very large datasets. There is a variety of procedures for sampling instances from a large dataset (Reinartz, 2002).

Feature subset selection is the process of identifying and removing as many irrelevant and redundant features as possible (Yu & Liu, 2004). This reduces the dimensionality of the data and enables data mining algorithms to operate faster and more effectively. The fact that many features depend on one another often unduly influences the accuracy of supervised ML classification models. This problem can be addressed by constructing new features from the basic feature set (Markovitch & Rosenstein, 2002). This technique is called feature construction/transformation. These newly generated features may lead to the creation of more concise and accurate classifiers. In addition, the discovery of meaningful features contributes to better comprehensibility of the produced classifier, and a better understanding of the learned concept.

2.1 Algorithm selection

The choice of which specific learning algorithm we should use is a critical step. Once preliminary testing is judged to be satisfactory, the classifier (mapping from unlabeled instances to classes) is available for routine use. The classifier's evaluation is most often based on prediction accuracy (the percentage of correct prediction divided by the total number of predictions). There are at least three techniques which are used to calculate a classifier's accuracy. One technique is to split the training set by using two-thirds for training and the other third for estimating performance. In another technique, known as cross-validation, the training set is divided into mutually exclusive and equal-sized subsets and for each subset the classifier is trained on the union of all the other subsets. The average of the error rate of each subset is therefore an estimate of the error rate of the classifier. Leave-one-out validation is a special case of cross validation. All test subsets consist of a single instance. This type of validation is, of course, more expensive computationally, but useful when the most accurate estimate of a classifier's error rate is required.

If the error rate evaluation is unsatisfactory, we must return to a previous stage of the supervised ML process (as detailed in Figure 1). A variety of factors must be examined: perhaps relevant features for the problem are not being used, a larger training set is needed, the dimensionality of the problem is too high, the selected algorithm is inappropriate or parameter tuning is needed. Another problem could be that the dataset is imbalanced (Japkowicz & Stephen, 2002).

A common method for comparing supervised ML algorithms is to perform statistical comparisons of the accuracies of trained classifiers on specific datasets. If we have sufficient supply of data, we can sample a number of training sets of size N, run the two learning algorithms on each of them, and estimate the difference in accuracy for each pair of classifiers on a large test set. The average of these differences is an estimate of the expected difference in generalization error across all possible training sets of size N, and their variance is an estimate of the variance of the classifier in the total set. Our next step is to perform paired t-test to check the null hypothesis that the mean difference between the classifiers is zero. This test can produce two types of errors. Type I error is the probability that the test rejects the null hypothesis incorrectly (i.e. it finds a "significant" difference although there is none). Type II error is the probability that the null hypothesis is not rejected, when there actually is a difference. The test's Type I error will be close to the chosen significance level.

In practice, however, we often have only one dataset of size N and all estimates must be obtained from this sole dataset. Different training sets are obtained by sub-sampling, and the instances not sampled for training are used for testing. Unfortunately this violates the independence assumption necessary for proper significance testing. The consequence of this is that Type I errors exceed the significance level. This is problematic because it is important for the researcher to be able to control Type I errors and know the probability of incorrectly rejecting the null hypothesis. Several heuristic versions of the t-test have been developed to alleviate this problem (Dietterich, 1998), (Nadeau and Bengio, 2003).

Ideally, we would like the test's outcome to be independent of the particular partitioning resulting from the randomization process, because this would make it much easier to replicate experimental results published in the literature. However, in practice there is always certain sensitivity to the partitioning used. To measure replicability we need to repeat the same test several times on the same data with different random partitionings — usually ten repetitions— and count how often the outcome is the same (Bouckaert, 2003).

Supervised classification is one of the tasks most frequently carried out by so-called Intelligent Systems. Thus, a large number of techniques have been developed based on Artificial Intelligence (Logical/Symbolic techniques), Perceptron-based techniques and Statistics (Bayesian Networks, Instance-based techniques). In next sections, we will focus on the most important supervised machine learning techniques, starting with logical/symbolic algorithms.

3 Logic based algorithms

In this section we will concentrate on two groups of logical (symbolic) learning methods: decision trees and rule-based classifiers.

3.1 Decision trees

Murthy (1998) provided an overview of work in decision trees and a sample of their usefulness to newcomers as well as practitioners in the field of machine learning. Thus, in this work, apart from a brief description of decision trees, we will refer to some more recent works than those in Murthy's article as well as few very important articles that were published earlier. Decision trees are trees that classify instances by sorting them based on feature values. Each node in a decision tree represents a feature in an instance to be classified, and each branch represents a value that the node can assume. Instances are classified starting at the root node and sorted based on their feature values. Figure 2 is an example of a decision tree for the training set of Table 2.

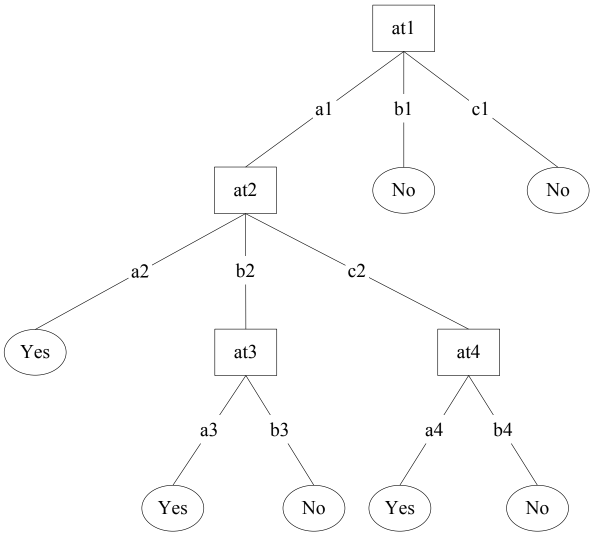


Figure 2. A decision tree

at1	at2	at3	at4	Class
a1	a2	a3	a4	Yes
a1	a2	a3	b4	Yes
a1	b2	a3	a4	Yes
a1	b2	b3	b4	No
a1	c2	a3	a4	Yes
a1	c2	a3	b4	No
b1	b2	b3	b4	No
c1	b2	b3	b4	No

Table 2. Training Set

Using the decision tree depicted in Figure 2 as an example, the instance ⟨at1 = a1, at2 = b2, at3 = a3, at4 = b4⟩ would sort to the nodes: at1, at2, and finally at3, which would classify the instance as being positive (represented by the values "Yes"). The problem of constructing optimal binary decision trees is an NP-complete problem and thus theoreticians have searched for efficient *heuristics* for constructing near-optimal decision trees.

The feature that best divides the training data would be the root node of the tree. There are numerous methods for finding the feature that best divides the training data such as information gain (Hunt et al., 1966) and gini index (Breiman et al., 1984). While myopic measures estimate each attribute independently, ReliefF algorithm (Kononenko, 1994) estimates them in the context of other attributes. However, a majority of studies have concluded that there is no single best method (Murthy, 1998). Comparison of individual methods may still be important when deciding which metric should be used in a particular dataset. The same procedure is then repeated on each partition of the divided data, creating sub-trees until the training data is divided into subsets of the same class.

Figure 3 presents a general pseudo-code for building decision trees.

```
Check for base cases
 For each attribute a
    Find the feature that best
    divides the training data such
    as information gain from
    splitting on a
Let a best be the attribute with the
highest normalized information gain
   Create a decision node node that
   splits on a_best
Recurse on the sub-lists obtained by
splitting on a best and add those
nodes as children of node
```

Figure 3. Pseudo-code for building a decision tree

A decision tree, or any learned hypothesis h, is said to overfit training data if another hypothesis h' exists that has a larger error than h when tested on the training data, but a smaller error than h when tested on the entire dataset. There are two common approaches that decision tree induction algorithms can use to avoid overfitting training data: i) Stop the training algorithm before it reaches a point at which it perfectly fits the training data, ii) Prune the induced decision tree. If the two trees employ the same kind of tests and have the same prediction accuracy, the one with fewer leaves is usually preferred. Breslow & Aha (1997) survey methods of tree simplification to improve their comprehensibility.

The most straightforward way of tackling overfitting is to pre-prune the decision tree by not allowing it to grow to its full size. Establishing a non-trivial termination criterion such as a threshold test for the feature quality metric can do that. Decision tree classifiers usually employ post-pruning techniques that evaluate the performance of decision trees, as they are pruned by using a validation set. Any node can be removed and assigned the most common class of the training instances that are sorted to it. A comparative study of well-known pruning methods is presented in (Elomaa, 1999). Elomaa (1999) concluded that there is no single best pruning method. More details, about not only postprocessing but also about preprocessing of decision tree algorithms can be fould in (Bruha, 2000).

Even though the divide-and-conquer algorithm is quick, efficiency can become important in tasks with hundreds of thousands of instances. The most time-consuming aspect is sorting the instances on a numeric feature to find the best threshold *t*. This can be expedited if possible thresholds for a numeric feature are determined just once, effectively converting the feature to discrete intervals, or if the threshold is determined from a subset of the instances. Elomaa & Rousu (1999) stated that the use of binary discretization with C4.5 needs about the half training time of using C4.5 multi-splitting. In addition, according to their experiments, multi-splitting of numerical features does not carry any advantage in prediction accuracy over binary splitting.

Decision trees are usually univariate since they use splits based on a single feature at each internal node. Most decision tree algorithms cannot perform well with problems that require diagonal partitioning. The division of the instance space is orthogonal to the axis of one variable and parallel to all other axes. Therefore, the resulting regions after partitioning are all hyper-rectangles. However, there are a few methods that construct multivariate trees. One example is Zheng's (1998), who improved the classification accuracy of the decision trees by constructing new binary features with logical operators such as conjunction, negation, and disjunction. In addition, Zheng (2000) created at-least *M*-of-*N* features. For a given instance, the value of an at-least *M*-of-*N* representation is true if at least *M* of its conditions is true of the instance, otherwise it is false. Gama and Brazdil (1999) combined a decision tree with a linear discriminant for constructing multivariate decision trees. In this model, new features are computed as linear combinations of the previous ones.

Decision trees can be significantly more complex representation for some concepts due to the replication problem. A solution is using an algorithm to implement complex features at nodes in order to avoid replication. Markovitch and Rosenstein (2002) presented the FICUS construction algorithm, which receives the standard input of supervised learning as well as a feature representation specification, and uses them to produce a set of generated features. While FICUS is similar in some aspects to other feature construction algorithms, its main strength is its generality and flexibility. FICUS was designed to perform feature generation given any feature representation specification complying with its general purpose grammar.

The most well-know algorithm in the literature for building decision trees is the C4.5 (Quinlan, 1993). C4.5 is an extension of Quinlan's earlier ID3 algorithm (Quinlan, 1979). One of the latest studies that compare decision trees and other learning algorithms has been done by (Tjen-Sien Lim et al. 2000). The study shows that C4.5 has a very good combination of error rate and speed. In 2001, Ruggieri presented an analytic evaluation of the runtime behavior of the C4.5 algorithm, which highlighted some efficiency improvements. Based on this analytic evaluation, he implemented a more efficient version of the algorithm, called EC4.5. He argued that his implementation computed the same decision trees as C4.5 with a performance gain of up to five times.

C4.5 assumes that the training data fits in memory, thus, Gehrke et al. (2000) proposed Rainforest, a framework for developing fast and scalable algorithms to construct decision trees that gracefully adapt to the amount of main memory available. It is clear that in most decision tree algorithms; a substantial effort is "wasted" in the building phase on growing portions of the tree that are subsequently pruned in the pruning phase. Rastogi & Shim (2000) proposed PUBLIC, an improved decision tree classifier that integrates the second "pruning" phase with the initial "building" phase. In PUBLIC, a node is not expanded during the building phase, if it is determined that the node will be pruned during the subsequent pruning phase.

Olcay and Onur (2007) show how to parallelize C4.5 algorithm in three ways: (i) feature based, (ii) node based (iii) data based manner. Baik and Bala (2004) presented preliminary work on an agent-based approach for the distributed learning of decision trees.

To sum up, one of the most useful characteristics of decision trees is their comprehensibility. People can easily understand why a decision tree classifies an instance as belonging to a specific class. Since a decision tree constitutes a hierarchy of tests, an unknown feature value during classification is usually dealt with by passing the example down all branches of the node where the unknown feature value was detected, and each branch outputs a class distribution. The output is a combination of the different class distributions that sum to 1. The assumption made in the decision trees is that instances belonging to different classes have different values in at least one of their features. Decision trees tend to perform better when dealing with discrete/categorical features.

3.2 Learning set of rules

Decision trees can be translated into a set of rules by creating a separate rule for each path from the root to a leaf in the tree (Quinlan, 1993). However, rules can also be directly induced from training data using a variety of rule-based algorithms. Furnkranz (1999) provided an excellent overview of existing work in rule-based methods.

Classification rules represent each class by disjunctive normal form (DNF). A k-DNF expression is of the form: $(X_1\wedge X_2\wedge\ldots\wedge X_n) \vee (X_{n+1}\wedge X_{n+2}\wedge\ldots X_{2n}) \vee \ldots\vee (X_{(k-1)n+1}\wedge X_{(k-1)n+2}\wedge\ldots\wedge X_{kn})$, where k is the number of disjunctions, n is the number of conjunctions in each disjunction, and X_n is defined over the alphabet $X_1, X_2,\ldots, X_j \cup \sim X_1, \sim X_2, \ldots,\sim X_j$. The goal is to construct the smallest rule-set that is consistent with the training data. A large number of learned rules is usually a sign that the learning algorithm is attempting to "remember" the training set, instead of discovering the assumptions that govern it. A separate-and-conquer algorithm (covering algorithms) search for a rule that explains a part of its training instances, separates these instances and recursively conquers the remaining instances by learning more rules, until no instances remain. In Figure 4, a general pseudo-code for rule learners is presented.

The difference between heuristics for rule learning and heuristics for decision trees is that the latter evaluate the average quality of a number of disjointed sets (one for each value of the feature that is tested), while rule learners only evaluate the quality of the set of instances that is covered by the candidate rule. More advanced rule learners differ from this simple pseudo-code mostly by adding additional mechanisms to prevent over-fitting of the training data, for instance by stopping the specialization process with the use of a quality measure or by generalizing overly specialized rules in a separate pruning phase (Furnkranz, 1997).

```
   On presentation of training examples
training examples:
   1. Initialise rule set to a default
(usually empty, or a rule assigning all
objects to the most common class).
   2. Initialise examples to either all
available examples or all examples not
correctly handled by rule set.
   3. Repeat
      (a) Find best, the best rule with
   respect to examples.
      (b) If such a rule can be found
         i. Add best to rule set.
         ii. Set examples to all
         examples not handled
         correctly by rule set.
      until no rule best can be found
      (for instance, because no
      examples remain).
```

Figure 4. Pseudocode for rule learners

It is therefore important for a rule induction system to generate decision rules that have high predictability or reliability. These properties are commonly measured by a function called rule quality. A rule quality measure is needed in both the rule induction and classification processes such as J-measure (Smyth and Goodman, 1990). In rule induction, a rule quality measure can be used as a criterion in the rule specification and/or generalization process. In classification, a rule quality value can be associated with each rule to resolve conflicts when multiple rules are satisfied by the example to be classified. An and Cercone (2000) surveyed a number of statistical and empirical rule quality measures. Furnkranz and Flach (2005) provided an analysis of the behavior of separate-and-conquer or covering rule learning algorithms by visualizing their evaluation metrics. When using unordered rule sets, conflicts can arise between the rules, i.e., two or more rules cover the same example but predict different classes. Lindgren (2004) has recently given a survey of methods used to solve this type of conflict.

RIPPER is a well-known rule-based algorithm (Cohen, 1995). It forms rules through a process of repeated *growing* and *pruning*. During the growing phase the rules are made more restrictive in order to fit the training data as closely as possible. During the pruning phase, the rules are made less restrictive in order to avoid overfitting, which can cause poor performance on unseen instances. RIPPER handles multiple classes by ordering them from least to most prevalent and then treating each in order as a distinct two-class problem. Other fundamental learning classifiers based on decision rules include the AQ family (Michalski and Chilausky, 1980) and CN2 (Clark and Niblett, 1989). Bonarini (2000) gave an overview of fuzzy rule-based classifiers. Fuzzy logic tries to improve classification and decision support systems by allowing the use of overlapping class definitions.

Furnkranz (2001) investigated the use of round robin binarization (or pairwise classification) as a technique for handling multi-class problems with separate and conquer rule learning algorithms. The round robin binarization transforms a c-class problem into c(c-1)/2 two-class problems <i,j>, one for each set of classes {i,j}, i= 1 ... c-1, j = i+1 ...c. The binary classifier for problem <i,j> is trained with examples of classes i and j, whereas examples of classes k ≠ i,j are ignored for this problem. A crucial point, of course, is determining how to decode the predictions of the pairwise classifiers for a final prediction. Furnkranz (2001) implemented a simple voting technique: when classifying a new example, each of the learned base classifiers determines to which of its two classes the example is more likely to belong to. The winner is assigned a point, and in the end, the algorithm predicts the class that has accumulated the most points. His experimental results show that, in comparison to conventional, ordered or unordered binarization, the round robin approach may yield significant gains in accuracy without risking a poor performance.

There are numerous other rule-based learning algorithms. Furnkranz (1999) referred to most of them. The PART algorithm infers rules by repeatedly generating partial decision trees, thus combining the two major paradigms for rule generation − creating rules from decision trees and the separate-and-conquer rule-learning technique. Once a partial tree has been build, a single rule is extracted from it and for this reason the PART algorithm avoids postprocessing (Frank and Witten, 1998).

For the task of learning binary problems, rules are more comprehensible than decision trees because typical rule-based approaches learn a set of rules for only the positive class. On the other hand, if definitions for multiple classes are to be learned, the rule-based learner must be run separately for each class separately. For each individual class a separate rule set is obtained and these sets may be inconsistent (a particular instance might be assigned multiple classes) or incomplete (no class might be assigned to a particular instance). These problems can be solved with decision lists (the rules in a rule set are supposed to be ordered, a rule is only applicable when none of the preceding rules are applicable) but with the decision tree approach, they simply do not occur. Moreover, the divide and conquer approach (used by decision trees) is usually more efficient than the separate and conquer approach (used by rule-based algorithms). Separate-and-conquer algorithms look at one class at a time, and try to produce rules that uniquely identify the class. They do this independent of all the other classes in the training set. For this reason, for small datasets, it may be better to use a divide-and-conquer algorithm that considers the entire set at once.

To sum up, the most useful characteristic of rule-based classifiers is their comprehensibility. In addition, even though some rule-based classifiers can deal with numerical features, some experts propose these features should be discretized before induction, so as to reduce training time and increase classification accuracy (An and Cercone, 1999). Classification accuracy of rule learning algorithms can be improved by combining features (such as in decision trees) using the background knowledge of the user (Flach and Lavrac, 2000) or automatic feature construction algorithms (Markovitch and Rosenstein, 2002).

4 Perceptron-based techniques

Other well-known algorithms are based on the notion of perceptron (Rosenblatt, 1962).

4.1 Single layered perceptrons

A single layered perceptron can be briefly described as follows:

If x_1 through x_n are input feature values and w_1 through w_n are connection weights/prediction vector (typically real numbers in the interval [-1, 1]), then perceptron computes the sum of weighted inputs: $\sum_i x_i w_i$ and output goes through an adjustable threshold: if the sum is above threshold, output is 1; else it is 0.

The most common way that the perceptron algorithm is used for learning from a batch of training instances is to run the algorithm repeatedly through the training set until it finds a prediction vector which is correct on all of the training set. This prediction rule is then used for predicting the labels on the test set.

WINNOW (Littlestone & Warmuth, 1994) is based on the perceptron idea and updates its weights as follows. If prediction value y'=0 and actual value y=1, then the weights are too low; so, for each feature such that x_i=1, w_i=w_i·α, where α is a number greater than 1, called the *promotion parameter*. If prediction value y'= 1 and actual value y=0, then the weights were too high; so, for each feature x_i = 1, it decreases the corresponding weight by setting w_i=w_i·β, where 0<β<1, called the *demotion parameter*. Generally, WINNOW is an example of an *exponential update algorithm*. The weights of the relevant features grow exponentially but the weights of the irrelevant features shrink exponentially. For this reason, it was experimentally proved (Blum, 1997) that WINNOW can adapt rapidly to changes in the target function (concept drift). A target function (such as user preferences) is not static in time. In order to enable, for example, a decision tree algorithm to respond to changes, it is necessary to decide which old training instances could be deleted. A number of algorithms similar to

WINNOW have been developed, such as those by Auer & Warmuth (1998).

Freund & Schapire (1999) created a newer algorithm, called *voted-perceptron*, which stores more information during training and then uses this elaborate information to generate better predictions about the test data. The information it maintains during training is the list of *all* prediction vectors that were generated after each and every mistake. For each such vector, it counts the number of iterations it "survives" until the next mistake is made; Freund & Schapire refer to this count as the "weight" of the prediction vector. To calculate a prediction the algorithm computes the binary prediction of each one of the prediction vectors and combines all these predictions by means of a weighted majority vote. The weights used are the survival times described above.

To sum up, we have discussed perceptron-like linear algorithms with emphasis on their superior time complexity when dealing with irrelevant features. This can be a considerable advantage when there are many features, but only a few relevant ones. Generally, all perceptron-like linear algorithms are *anytime online algorithms* that can produce a useful answer regardless of how long they run (Kivinen, 2002). The longer they run, the better the result they produce. Finally, perceptron-like methods are binary, and therefore in the case of multi-class problem one must reduce the problem to a set of multiple binary classification problems.

4.2 Multilayered perceptrons

Perceptrons can only classify linearly separable sets of instances. If a straight line or plane can be drawn to seperate the input instances into their correct categories, input instances are linearly separable and the perceptron will find the solution. If the instances are not linearly separable learning will never reach a point where all instances are classified properly. Multilayered Perceptrons (Artificial Neural Networks) have been created to try to solve this problem (Rumelhart et al., 1986). Zhang (2000) provided an overview of existing work in Artificial Neural Networks (ANNs). Thus, in this study, apart from a brief description of the ANNs we will mainly refer to some more recent articles. A multi-layer neural network consists of large number of units (neurons) joined together in a pattern of connections (Figure 5). Units in a net are usually segregated into three classes: input units, which receive information to be processed; output units, where the results of the processing are found; and units in between known as hidden units. Feed-forward ANNs (Figure 5) allow signals to travel one way only, from input to output.

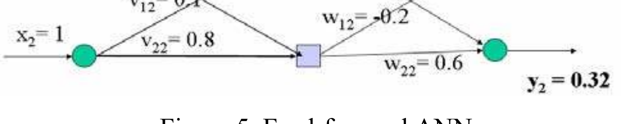


Figure 5. Feed-forward ANN

First, the network is trained on a set of paired data to determine input-output mapping. The weights of the connections between neurons are then fixed and the network is used to determine the classifications of a new set of data.

During classification the signal at the input units propagates all the way through the net to determine the activation values at all the output units. Each input unit has an activation value that represents some feature external to the net. Then, every input unit sends its activation value to each of the hidden units to which it is connected. Each of these hidden units calculates its own activation value and this signal are then passed on to output units. The activation value for each receiving unit is calculated according to a simple activation function. The function sums together the contributions of all sending units, where the contribution of a unit is defined as the weight of the connection between the sending and receiving units multiplied by the sending unit's activation value. This sum is usually then further modified, for example, by adjusting the activation sum to a value between 0 and 1 and/or by setting the activation value to zero unless a threshold level for that sum is reached.

Generally, properly determining the size of the hidden layer is a problem, because an underestimate of the number of neurons can lead to poor approximation and generalization capabilities, while excessive nodes can result in overfitting and eventually make the search for the global optimum more difficult. An excellent argument regarding this topic can be found in (Camargo & Yoneyama, 2001). Kon & Plaskota (2000) also studied the minimum amount of neurons and the number of instances necessary to program a given task into feed-forward neural networks.

ANN depends upon three fundamental aspects, input and activation functions of the unit, network architecture and the weight of each input connection. Given that the first two aspects are fixed, the behavior of the ANN is defined by the current values of the weights. The weights of the net to be trained are initially set to random values, and then instances of the training set are repeatedly exposed to the net. The values for the input of an instance are placed on the input units and the output of the net is compared with the desired output for this instance. Then, all the weights in the net are adjusted slightly in the direction that would bring the output values of the net closer to the values for the desired output. There are several algorithms with which a network can be trained (Neocleous & Schizas, 2002). However, the most well-known and widely used learning algorithm to estimate the values of the weights is the Back Propagation (BP) algorithm. Generally, BP algorithm includes the following six steps:

1. Present a training sample to the neural network.
2. Compare the network's output to the desired output from that sample. Calculate the error in each output neuron.
3. For each neuron, calculate what the output should have been, and a scaling factor, how much lower or higher the output must be adjusted to match the desired output. This is the local error.

4. Adjust the weights of each neuron to lower the local error.
5. Assign "blame" for the local error to neurons at the previous level, giving greater responsibility to neurons connected by stronger weights.
6. Repeat the steps above on the neurons at the previous level, using each one's "blame" as its error.

With more details, the general rule for updating weights is: $\Delta W_{ji} = \eta \delta_j O_i$ where:

- η is a positive number (called learning rate), which determines the step size in the gradient descent search. A large value enables back propagation to move faster to the target weight configuration but it also increases the chance of its never reaching this target.
- O_i is the output computed by neuron i
- $\delta_j = O_j(1-O_j)(T_j - O_j)$ for the output neurons, where T_j the wanted output for the neuron j and
- $\delta_j = O_j(1-O_j)\sum_k \delta_k W_{kj}$ for the internal (hidden) neurons

The back propagation algorithm will have to perform a number of weight modifications before it reaches a good weight configuration. For n training instances and W weights, each repetition/epoch in the learning process takes O(nW) time; but in the worst case, the number of epochs can be exponential to the number of inputs. For this reason, neural nets use a number of different *stopping rules* to control when training ends. The four most common stopping rules are: i) Stop after a specified number of epochs, ii) Stop when an error measure reaches a threshold, iii) Stop when the error measure has seen no improvement over a certain number of epochs, iv) Stop when the error measure on some of the data that has been sampled from the training data (hold-out set, validation set) is more than a certain amount than the error measure on the training set (overfitting).

Feed-forward neural networks are usually trained by the original back propagation algorithm or by some variant. Their greatest problem is that they are too slow for most applications. One of the approaches to speed up the training rate is to estimate optimal initial weights (Yam & Chow, 2001). Another method for training multilayered feedforward ANNs is Weight-elimination algorithm that automatically derives the appropriate topology and therefore avoids also the problems with overfitting (Weigend et al., 1991). Genetic algorithms have been used to train the weights of neural networks (Siddique and Tokhi, 2001) and to find the architecture of neural networks (Yen and Lu, 2000). There are also Bayesian methods in existence which attempt to train neural networks. Vivarelli & Williams (2001) compare two Bayesian methods for training neural networks. A number of other techniques have emerged recently which attempt to improve ANNs training algorithms by changing the architecture of the networks as training proceeds. These techniques include *pruning* useless nodes or weights (Castellano et al. 1997), and *constructive algorithms,* where extra nodes are added as required (Parekh et al. 2000).

4.3 Radial Basis Function (RBF) networks

ANN learning can be achieved, among others, through i) synaptic weight modification, ii) network structure modifications (creating or deleting neurons or synaptic connections), iii) use of suitable attractors or other suitable stable state points, iv) appropriate choice of activation functions. Since back-propagation training is a gradient descending process, it may get stuck in local minima in this weight-space. It is because of this possibility that neural network models are characterized by high variance and unsteadiness.

Radial Basis Function (RBF) networks have been also widely applied in many science and engineering fields (Robert and Howlett, 2001). An RBF network is a three-layer feedback network, in which each hidden unit implements a radial activation function and each output unit implements a weighted sum of hidden units outputs. Its training procedure is usually divided into two stages. First, the centers and widths of the hidden layer are determined by clustering algorithms. Second, the weights connecting the hidden layer with the output layer are determined by Singular Value Decomposition (SVD) or Least Mean Squared (LMS) algorithms. The problem of selecting the appropriate number of basis functions remains a critical issue for RBF networks. The number of basis functions controls the complexity and the generalization ability of RBF networks. RBF networks with too few basis functions cannot fit the training data adequately due to limited flexibility. On the other hand, those with too many basis functions yield poor generalization abilities since they are too flexible and erroneously fit the noise in the training data.

Even though multilayer neural networks and decision trees are two very different techniques for the purpose of classification, some researchers (Eklund & Hoang, 2002), (Tjen-Sien Lim et al. 2000) have performed some empirical comparative studies. Some of the general conclusions drawn in that work are:

i) neural networks are usually more able to easily provide incremental learning than decision trees (Saad, 1998), even though there are some algorithms for incremental learning of decision trees such as (Utgoff et al, 1997) and (McSherry, 1999). Incremental decision tree induction techniques result in frequent tree restructuring when the amount of training data is small, with the tree structure maturing as the data pool becomes larger.

ii) training time for a neural network is usually much longer than training time for decision trees.

iii) neural networks usually perform as well as decision trees, but seldom better.

To sum up, ANNs have been applied to many real-world problems but still, their most striking disadvantage is their lack of ability to reason about their output in a way that can be effectively communicated. For this reason many researchers have tried to address the issue of improving the comprehensibility of neural networks, where the most attractive solution is to extract symbolic rules from trained neural networks. Setiono and Leow (2000) divided the activation values of relevant hidden units into two subintervals and then found the set of relevant connections of those relevant units to construct rules. More references can be found in (Zhou, 2004), an excellent survey. However, it is also worth mentioning that Roy (2000) identified the conflict between the idea of rule extraction and traditional connectionism. In detail, the idea of rule extraction from a neural network involves certain procedures, specifically the reading of parameters from a network, which is not allowed by the traditional connectionist framework that these neural networks are based on.

5 Statistical learning algorithms

Conversely to ANNs, statistical approaches are characterized by having an explicit underlying probability model, which provides a probability that an instance belongs in each class, rather than simply a classification. Linear discriminant analysis (LDA) and the related Fisher's linear discriminant are simple methods used in statistics and machine learning to find the linear combination of features which best separate two or more classes of object (Friedman, 1989). LDA works when the measurements made on each observation are continuous quantities. When dealing with categorical variables, the equivalent technique is Discriminant Correspondence Analysis (Mika et al., 1999).

Maximum entropy is another general technique for estimating probability distributions from data. The overriding principle in maximum entropy is that when nothing is known, the distribution should be as uniform as possible, that is, have maximal entropy. Labeled training data is used to derive a set of constraints for the model that characterize the class-specific expectations for the distribution. Csiszar (1996) provides a good tutorial introduction to maximum entropy techniques.

Bayesian networks are the most well known representative of statistical learning algorithms. A comprehensive book on Bayesian networks is Jensen's (1996). Thus, in this study, apart from our brief description of Bayesian networks, we mainly refer to more recent works.

5.1.1 Naive Bayes classifiers

Naive Bayesian networks (NB) are very simple Bayesian networks which are composed of directed acyclic graphs with only one parent (representing the unobserved node) and several children (corresponding to observed nodes) with a strong assumption of independence among child nodes in the context of their parent (Good, 1950).Thus, the independence model (Naive Bayes) is based on estimating (Nilsson, 1965):

$$R=\frac{P(i\mid X)}{P(j\mid X)}=\frac{P(i)P(X\mid i)}{P(j)P(X\mid j)}=\frac{P(i)\prod P(X_r\mid i)}{P(j)\prod P(X_r\mid j)}$$

Comparing these two probabilities, the larger probability indicates that the class label value that is more likely to be the actual label (if *R*>1: predict *i*. else predict *j*). Cestnik et al (1987) first used the Naive Bayes in ML community. Since the Bayes classification algorithm uses a product operation to compute the probabilities P(*X*, *i*), it is especially prone to being unduly impacted by probabilities of 0. This can be avoided by using Laplace estimator or m-esimate, by adding one to all numerators and adding the number of added ones to the denominator (Cestnik, 1990).

The assumption of independence among child nodes is clearly almost always wrong and for this reason naive Bayes classifiers are usually less accurate that other more sophisticated learning algorithms (such ANNs). However, Domingos & Pazzani (1997) performed a large-scale comparison of the naive Bayes classifier with state-of-the-art algorithms for decision tree induction, instance-based learning, and rule induction on standard benchmark datasets, and found it to be sometimes superior to the other learning schemes, even on datasets with substantial feature dependencies.

The basic independent Bayes model has been modified in various ways in attempts to improve its performance. Attempts to overcome the independence assumption are mainly based on adding extra edges to include some of the dependencies between the features, for example (Friedman et al. 1997). In this case, the network has the limitation that each feature can be related to only one other feature. Semi-naive Bayesian classifier is another important attempt to avoid the independence assumption. (Kononenko, 1991), in which attributes are partitioned into groups and it is assumed that x_i is conditionally independent of x_j if and only if they are in different groups.

The major advantage of the naive Bayes classifier is its short computational time for training. In addition, since the model has the form of a product, it can be converted into a sum through the use of logarithms - with significant consequent computational advantages. If a feature is numerical, the usual procedure is to discretize it during data pre-processing (Yang & Webb, 2003), although a researcher can use the normal distribution to calculate probabilities (Bouckaert, 2004).

5.2 Bayesian Networks

A Bayesian Network (BN) is a graphical model for probability relationships among a set of variables (features) (see Figure 6). The Bayesian network structure *S* is a directed acyclic graph (DAG) and the nodes in *S* are in one-to-one correspondence with the features *X*. The arcs represent casual influences among the features while the *lack* of possible arcs in *S* encodes conditional independencies. Moreover, a feature (node) is conditionally independent from its non-descendants given its parents (X_1 is conditionally independent from X_2

given X_3 if P(X_1|X_2,X_3)=P(X_1|X_3) for all possible values of X_1, X_2, X_3).

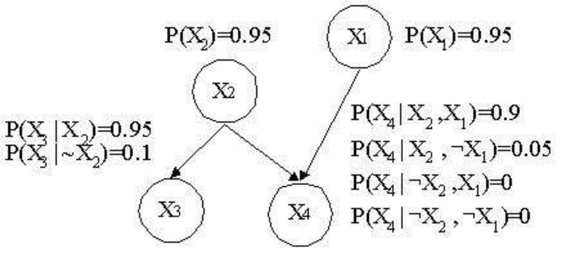


Figure 6. The structure of a Bayes Network

Typically, the task of learning a Bayesian network can be divided into two subtasks: initially, the learning of the DAG structure of the network, and then the determination of its parameters. Probabilistic parameters are encoded into a set of tables, one for each variable, in the form of local conditional distributions of a variable given its parents. Given the independences encoded into the network, the joint distribution can be reconstructed by simply multiplying these tables. Within the general framework of inducing Bayesian networks, there are two scenarios: known structure and unknown structure.

In the first scenario, the structure of the network is given (e.g. by an expert) and assumed to be correct. Once the network structure is fixed, learning the parameters in the Conditional Probability Tables (CPT) is usually solved by estimating a locally exponential number of parameters from the data provided (Jensen, 1996). Each node in the network has an associated CPT that describes the conditional probability distribution of that node given the different values of its parents.

In spite of the remarkable power of Bayesian Networks, they have an inherent limitation. This is the computational difficulty of exploring a previously unknown network. Given a problem described by *n* features, the number of possible structure hypotheses is more than exponential in *n*. If the structure is unknown, one approach is to introduce a scoring function (or a score) that evaluates the "fitness" of networks with respect to the training data, and then to search for the best network according to this score. Several researchers have shown experimentally that the selection of a single good hypothesis using greedy search often yields accurate predictions (Heckerman et al. 1999), (Chickering, 2002). In Figure 7 there is a pseudo-code for training BNs.

Within the score & search paradigm, another approach uses local search methods in the space of directed acyclic graphs, where the usual choices for defining the elementary modifications (local changes) that can be applied are arc addition, arc deletion, and arc reversal. Acid and de Campos (2003) proposed a new local search method, restricted acyclic partially directed graphs, which uses a different search space and takes account of the concept of equivalence between network structures. In this way, the number of different configurations of the search space is reduced, thus improving efficiency.

```
Initialize an empty Bayesian network
G containing n nodes (i.e., a BN with n
nodes but no edges)
Evaluate the score of G: Score(G)
G' = G
for i = 1 to n do
    for j = 1 to n do
        if i • j then
            if there is no edge between the
            nodes i and j in G• then
                Modify G' by adding an edge between
                the nodes i and j in G• such that i
                is a parent of j: (i • j)
                if the resulting G' is a DAG then
                    if (Score(G') > Score(G)) then
                        G = G'
                    end if
                end if
            end if
        end if
        G' = G
    end for
end for
```

Figure 7. Pseudo-code for training BN

A BN structure can be also found by learning the conditional independence relationships among the features of a dataset. Using a few statistical tests (such as the Chi-squared and mutual information test), one can find the conditional independence relationships among the features and use these relationships as constraints to construct a BN. These algorithms are called *CI-based* algorithms or constraint-based algorithms. Cowell (2001) has shown that for any structure search procedure based on CI tests, an equivalent procedure based on maximizing a score can be specified.

A comparison of scoring-based methods and CI-based methods is presented in (Heckerman et al., 1999). Both of these approaches have their advantages and disadvantages. Generally speaking, the dependency analysis approach is more efficient than the search & scoring approach for sparse networks (networks that are not densely connected). It can also deduce the correct structure when the probability distribution of the data satisfies certain assumptions. However, many of these algorithms require an exponential number of CI tests and many high order CI tests (CI tests with large condition-sets). Yet although the search & scoring approach may not find the best structure due to its heuristic nature, it works with a wider range of probabilistic models than the dependency analysis approach. Madden (2003) compared the performance of a number of Bayesian Network Classifiers. His experiments demonstrated that very similar classification performance can be achieved by classifiers constructed using the different approaches described above.

The most generic learning scenario is when the structure of the network is unknown and there is missing data. Friedman & Koller (2003) proposed a new approach for this task and showed how to efficiently compute a sum over the exponential number of networks that are consistent with a fixed order over networks.

Using a suitable version of any of the model types mentioned in this review, one can induce a Bayesian Network from a given training set. A classifier based on the network and on the given set of features X_1,X_2, ... X_n, returns the label *c,* which maximizes the posterior probability p($c \mid X_1, X_2, \dots X_n$).

Bayesian multi-nets allow different probabilistic dependencies for different values of the class node (Jordan, 1998). This suggests that simple BN classifiers should work better when there is a single underlying model of the dataset and multi-net classifier should work better when the underlying relationships among the features are very different for different classes (Cheng and Greiner, 2001).

The most interesting feature of BNs, compared to decision trees or neural networks, is most certainly the possibility of taking into account prior information about a given problem, in terms of structural relationships among its features. This prior expertise, or domain knowledge, about the structure of a Bayesian network can take the following forms:

1. Declaring that a node is a root node, i.e., it has no parents.
2. Declaring that a node is a leaf node, i.e., it has no children.
3. Declaring that a node is a direct cause or direct effect of another node.
4. Declaring that a node is not directly connected to another node.
5. Declaring that two nodes are independent, given a condition-set.
6. Providing partial nodes ordering, that is, declare that a node appears earlier than another node in the ordering.
7. Providing a complete node ordering.

A problem of BN classifiers is that they are not suitable for datasets with many features (Cheng et al., 2002). The reason for this is that trying to construct a very large network is simply not feasible in terms of time and space. A final problem is that before the induction, the numerical features need to be discretized in most cases.

6 Instance-based learning

Another category under the header of statistical methods is Instance-based learning. Instance-based learning algorithms are lazy-learning algorithms (Mitchell, 1997), as they delay the induction or generalization process until classification is performed. Lazy-learning algorithms require less computation time during the training phase than eager-learning algorithms (such as decision trees, neural and Bayes nets) but more computation time during the classification process. One of the most straightforward instance-based learning algorithms is the *nearest neighbour* algorithm. Aha (1997) and De Mantaras and Armengol (1998) presented a review of instance-based learning classifiers. Thus, in this study, apart from a brief description of the *nearest neighbour* algorithm, we will refer to some more recent works.

k-Nearest Neighbour (kNN) is based on the principle that the instances within a dataset will generally exist in close proximity to other instances that have similar properties (Cover and Hart, 1967). If the instances are tagged with a classification label, then the value of the label of an unclassified instance can be determined by observing the class of its nearest neighbours. The kNN locates the *k* nearest instances to the query instance and determines its class by identifying the single most frequent class label. In Figure 8, a pseudo-code example for the instance base learning methods is illustrated.

```
procedure    InstanceBaseLearner(Testing
Instances)
   for each testing instance
   {
   find the k most nearest instances of
   the training set according to a
   distance metric
   Resulting Class= most frequent class
   label of the k nearest instances
   }
```

Figure 8. Pseudo-code for instance-based learners

In general, instances can be considered as points within an *n*-dimensional instance space where each of the *n*-dimensions corresponds to one of the *n*-features that are used to describe an instance. The absolute position of the instances within this space is not as significant as the relative distance between instances. This relative distance is determined by using a distance metric. Ideally, the distance metric must minimize the distance between two similarly classified instances, while maximizing the distance between instances of different classes. Many different metrics have been presented. The most significant ones are presented in Table 3.

Minkowsky: $D(x,y)=\left(\sum_{i=1}^{m}\lvert x_i-y_i\rvert^r\right)^{1/r}$
Manhattan: $D(x,y)=\sum_{i=1}^{m}\lvert x_i-y_i\rvert$
Chebychev: $D(x,y)=\max_{i=1}^{m}\lvert x_i-y_i\rvert$
Euclidean: $D(x,y)=\left(\sum_{i=1}^{m}\lvert x_i-y_i\rvert^2\right)^{1/2}$
Camberra: $D(x,y)=\sum_{i=1}^{m}\frac{\lvert x_i-y_i\rvert}{\lvert x_i+y_i\rvert}$
Kendall's Rank Correlation: $D(x,y)=1-\frac{2}{m(m-1)}\sum_{i=j}^{m}\sum_{j=1}^{i-1}sign(x_i-x_j)sign(y_i-y_j)$

Table 3. Approaches to define the distance between instances (x and y)

For more accurate results, several algorithms use weighting schemes that alter the distance measurements and voting influence of each instance. A survey of weighting schemes is given by (Wettschereck et al., 1997).

The power of kNN has been demonstrated in a number of real domains, but there are some reservations about the usefulness of kNN, such as: i) they have large storage requirements, ii) they are sensitive to the choice of the similarity function that is used to compare instances, iii) they lack a principled way to choose *k*, except through cross-validation or similar, computationally-expensive technique (Guo et al. 2003).

The choice of *k* affects the performance of the kNN algorithm. Consider the following reasons why a k-nearest neighbour classifier might incorrectly classify a query instance:

- When noise is present in the locality of the query instance, the noisy instance(s) win the majority vote, resulting in the incorrect class being predicted. A larger *k* could solve this problem.
- When the region defining the class, or fragment of the class, is so small that instances belonging to the class that surrounds the fragment win the majority vote. A smaller *k* could solve this problem.

Wettschereck et al. (1997) investigated the behavior of the kNN in the presence of noisy instances. The experiments showed that the performance of kNN was not sensitive to the exact choice of *k* when *k* was large. They found that for small values of *k*, the kNN algorithm was more robust than the single nearest neighbour algorithm (1NN) for the majority of large datasets tested. However, the performance of the kNN was inferior to that achieved by the 1NN on small datasets (<100 instances).

Okamoto and Yugami (2003) represented the expected classification accuracy of k-NN as a function of domain characteristics including the number of training instances, the number of relevant and irrelevant attributes, the probability of each attribute, the noise rate for each type of noise, and k. They also explored the behavioral implications of the analyses by presenting the effects of domain characteristics on the expected accuracy of k-NN and on the optimal value of k for artificial domains.

The time to classify the query instance is closely related to the number of stored instances and the number of features that are used to describe each instance. Thus, in order to reduce the number of stored instances, instance-filtering algorithms have been proposed (Kubat and Cooperson, 2001). Brighton & Mellish (2002) found that their ICF algorithm and RT3 algorithm (Wilson & Martinez, 2000) achieved the highest degree of instance set reduction as well as the retention of classification accuracy: they are close to achieving unintrusive storage reduction. The degree to which these algorithms perform is quite impressive: an average of 80% of cases are removed and classification accuracy does not drop significantly. One other choice in designing a training set reduction algorithm is to modify the instances using a new representation such as prototypes (Sanchez et al., 2002).

Breiman (1996) reported that the stability of nearest neighbor classifiers distinguishes them from decision trees and some kinds of neural networks. A learning method is termed "unstable" if small changes in the training-test set split can result in large changes in the resulting classifier.

As we have already mentioned, the major disadvantage of instance-based classifiers is their large computational time for classification. A key issue in many applications is to determine which of the available input features should be used in modeling via feature selection (Yu & Liu, 2004), because it could improve the classification accuracy and scale down the required classification time. Furthermore, choosing a more suitable distance metric for the specific dataset can improve the accuracy of instance-based classifiers.

7 Support Vector Machines

Support Vector Machines (SVMs) are the newest supervised machine learning technique (Vapnik, 1995). An excellent survey of SVMs can be found in (Burges, 1998), and a more recent book is by (Cristianini & Shawe-Taylor, 2000). Thus, in this study apart from a brief description of SVMs we will refer to some more recent works and the landmark that were published before these works. SVMs revolve around the notion of a “margin”—either side of a hyperplane that separates two data classes. Maximizing the margin and thereby creating the largest possible distance between the separating hyperplane and the instances on either side of it has been proven to reduce an upper bound on the expected generalisation error.

If the training data is linearly separable, then a pair $(\mathbf{w}, b)$ exists such that

$$\mathbf{w}^T\mathbf{x}_i + b \geq 1, \text{for all } \mathbf{x}_i \in P$$

$$\mathbf{w}^T\mathbf{x}_i + b \leq -1, \text{for all } \mathbf{x}_i \in N$$

with the decision rule given by $f_{\mathbf{w},b}(\mathbf{x}) = \operatorname{sgn}(\mathbf{w}^T\mathbf{x} + b)$ where $\mathbf{w}$ is termed the weight vector and b the bias (or $-b$ is termed the threshold).

It is easy to show that, when it is possible to linearly separate two classes, an optimum separating hyperplane can be found by minimizing the squared norm of the separating hyperplane. The minimization can be set up as a convex quadratic programming (QP) problem:

$$\underset{\mathbf{w},b}{\text{Minimize}}\ \Phi(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 \qquad (1)$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, i = 1,\ldots,l.$$

In the case of linearly separable data, once the optimum separating hyperplane is found, data points that lie on its margin are known as support vector points and the solution is represented as a linear combination of only these points (see Figure 9). Other data points are ignored.

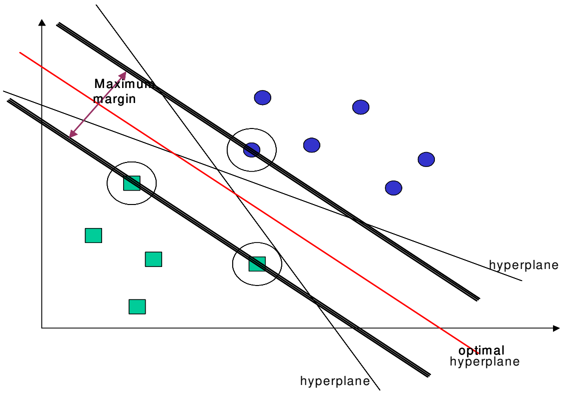


Figure 9. Maximum Margin

Therefore, the model complexity of an SVM is unaffected by the number of features encountered in the training data (the number of support vectors selected by the SVM learning algorithm is usually small). For this reason, SVMs are well suited to deal with learning tasks where the number of features is large with respect to the number of training instances.

A general pseudo-code for SVMs is illustrated in Figure 10.

```
1) Introduce positive Lagrange multipliers, one for each of the inequality constraints (1). This gives Lagrangian:
   $L_P \equiv \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N}\alpha_i y_i (x_i \cdot w - b) + \sum_{i=1}^{N}\alpha_i$
   2) Minimize $L_P$ with respect to w, b. This is a convex quadratic programming problem.
   3) In the solution, those points for which $\alpha_i > 0$ are called "support vectors"
```

Figure 10. Pseudo-code for SVMs

Even though the maximum margin allows the SVM to select among multiple candidate hyperplanes, for many datasets, the SVM may not be able to find any separating hyperplane at all because the data contains misclassified instances. The problem can be addressed by using a *soft margin* that accepts some misclassifications of the training instances (Veropoulos et al. 1999). This can be done by introducing positive slack variables $\xi_i, \quad i = 1,...,N$ in the constraints, which then become:

$$w \cdot x_i - b \geq +1 - \xi \quad for \quad y_i = +1$$

$$w \cdot x_i - b \leq -1 + \xi \quad for \quad y_i = -1$$

$$\xi \geq 0,$$

Thus, for an error to occur the corresponding ξ_i must exceed unity, so $\sum_i \xi_i$ is an upper bound on the number of training errors. In this case the Lagrangian is:

$$L_P \equiv \frac{1}{2}\|w\|^2 + C\sum_i \xi_i - \sum_i \alpha_i \{y_i(x_i \cdot w - b) - 1 + \xi_i\} - \sum_i \mu_i \xi_i$$

where the μ_i are the Lagrange multipliers introduced to enforce positivity of the ξ_i.

Nevertheless, most real-world problems involve non-separable data for which no hyperplane exists that successfully separates the positive from negative instances in the training set. One solution to the inseparability problem is to map the data onto a higher-dimensional space and define a separating hyperplane there. This higher-dimensional space is called the *transformed feature space*, as opposed to the *input space* occupied by the training instances.

With an appropriately chosen transformed feature space of sufficient dimensionality, any consistent training set can be made separable. A linear separation in transformed feature space corresponds to a non-linear separation in the original input space. Mapping the data to some other (possibly infinite dimensional) Hilbert space H as $\Phi : R^d \rightarrow H$. Then the training algorithm would only depend on the data through dot products in H, i.e. on functions of the form $\Phi(x_i) \cdot \Phi(x_j)$. If there were a "kernel function" K such that $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$, we would only need to use K in the training algorithm, and would never need to explicitly determine Φ. Thus, kernels are a special class of function that allow inner products to be calculated directly in feature space, without performing the mapping described above (Scholkopf et al. 1999). Once a hyperplane has been created, the kernel function is used to map new points into the feature space for classification.

The selection of an appropriate kernel function is important, since the kernel function defines the transformed feature space in which the training set instances will be classified. Genton (2001) described several classes of kernels, however, he did not address the question of which class is best suited to a given problem. It is common practice to estimate a range of potential settings and use cross-validation over the training set to find the best one. For this reason a limitation of SVMs is the low speed of the training. Selecting kernel settings can be regarded in a similar way to choosing the number of hidden nodes in a neural network. As long as the kernel function is legitimate, a SVM will operate correctly even if the designer does not know exactly what features of the training data are being used in the kernel-induced transformed feature space.

Some popular kernels are the following:

(1) $K(x, y) = (x \cdot y + 1)^P$,

(2) $K(x, y) = e^{-\|x - y\|^2 / 2\sigma^2}$,

(3) $K(x, y) = \tanh(\kappa x \cdot y - \delta)^P$

Training the SVM is done by solving N^{th} dimensional QP problem, where N is the number of samples in the training dataset. Solving this problem in standard QP methods involves large matrix operations, as well as time-consuming numerical computations, and is mostly very slow and impractical for large problems. Sequential Minimal Optimization (SMO) is a simple algorithm that can, relatively quickly, solve the SVM QP problem without any extra matrix storage and without using numerical QP optimization steps at all (Platt, 1999). SMO decomposes the overall QP problem into QP sub-problems. Keerthi and Gilbert (2002) suggested two modified versions of SMO that are significantly faster than the original SMO in most situations.

Finally, the training optimization problem of the SVM necessarily reaches a global minimum, and avoids ending in a local minimum, which may happen in other search algorithms such as neural networks. However, the SVM methods are binary, thus in the case of multi-class problem one must reduce the problem to a set of multiple binary classification problems. Discrete data presents another problem, although with suitable rescaling good results can be obtained.

8 Discussion

Supervised machine learning techniques are applicable in numerous domains. A number of ML application oriented papers can be found in (Saitta and Neri, 1998) and (Witten and Frank, 2005). Below, we present our conclusions about the use of each technique. Discussions of all the pros and cons of each individual algorithms and empirical comparisons of various bias options are beyond the scope of this paper; as the choice of algorithm always depends on the task at hand. However, we hope that the following remarks can help practitioners not to select a wholly inappropriate algorithm for their problem.

Generally, SVMs and neural networks tend to perform much better when dealing with multi-dimensions and continuous features. On the other hand, logic-based systems tend to perform better when dealing with discrete/categorical features. For neural network models and SVMs, a large sample size is required in order to achieve its maximum prediction accuracy whereas NB may need a relatively small dataset.

SVMs are binary algorithm, thus we made use of error-correcting output coding (ECOC), or, in short, the output coding approach, to reduce a multi-class problem to a set of multiple binary classification problems (Crammer & Singer, 2002). Output coding for multi-class problems is composed of two stages. In the training stage, we construct multiple independent binary classifiers, each of which is based on a different partition of the set of the labels into two disjointed sets. In the second stage, the classification part, the predictions of the binary classifiers are combined to extend a prediction on the original label of a test instance.

There is general agreement that k-NN is very sensitive to irrelevant features: this characteristic can be explained by the way the algorithm works. Moreover, the presence of irrelevant features can make neural network training very inefficient, even impractical.

Bias measures the contribution to error of the central tendency of the classifier when trained on different data (Bauer & Kohavi, 1999). Variance is a measure of the contribution to error of deviations from the central tendency. Learning algorithms with a high-bias profile usually generate simple, highly constrained models which are quite insensitive to data fluctuations, so that variance is low. Naive Bayes is considered to have high bias, because it assumes that the dataset under consideration can be summarized by a single probability distribution and that this model is sufficient to discriminate between classes. On the contrary, algorithms with a high-variance profile can generate arbitrarily complex models which fit data variations more readily. Examples of high-variance algorithms are decision trees, neural networks and SVMs. The obvious pitfall of high-variance model classes is overfitting.

Most decision tree algorithms cannot perform well with problems that require diagonal partitioning. The division of the instance space is orthogonal to the axis of one variable and parallel to all other axes. Therefore, the resulting regions after partitioning are all hyperrectangles. The ANNs and the SVMs perform well when multicollinearity is present and a nonlinear relationship exists between the input and output features.

Lazy learning methods require zero training time because the training instance is simply stored. Naive Bayes methods also train very quickly since they require only a single pass on the data either to count frequencies (for discrete variables) or to compute the normal probability density function (for continuous variables under normality assumptions). Univariate decision trees are also reputed to be quite fast—at any rate, several orders of magnitude faster than neural networks and SVMs.

Naive Bayes requires little storage space during both the training and classification stages: the strict minimum is the memory needed to store the prior and conditional probabilities. The basic kNN algorithm uses a great deal of storage space for the training phase, and its execution space is at least as big as its training space. On the contrary, for all non-lazy learners, execution space is usually much smaller than training space, since the resulting classifier is usually a highly condensed summary of the data. Moreover, Naive Bayes and the kNN can be easily used as incremental learners whereas rule algorithms cannot. Naive Bayes is naturally robust to missing values since these are simply ignored in computing probabilities and hence have no impact on the final decision. On the contrary, kNN and neural networks require complete records to do their work.

Moreover, kNN is generally considered intolerant of noise; its similarity measures can be easily distorted by errors in attribute values, thus leading it to misclassify a new instance on the basis of the wrong nearest neighbors. Contrary to kNN, rule learners and most decision trees are considered resistant to noise because their pruning strategies avoid overfitting the data in general and noisy data in particular.

What is more, the number of model or runtime parameters to be tuned by the user is an indicator of an algorithm's ease of use. As expected, neural networks and SVMs have more parameters than the remaining techniques. The basic kNN has usually only a single parameter (k) which is relatively easy to tune.

Logic-based algorithms are all considered very easy to interpret, whereas neural networks and SVMs have notoriously poor interpretability. k-NN is also considered to have very poor interpretability because an unstructured collection of training instances is far from readable, especially if there are many of them. While interpretability concerns the typical classifier generated by a learning algorithm, transparency refers to whether the principle of the method is easily understood. A particularly eloquent case is that of k-NN; while the resulting classifier is not quite interpretable, the method itself is quite transparent because it appeals to the intuition of human users, who spontaneously reason in a similar manner. Similarly, Naive Bayes' is very transparent, as it is easily grasped by users like physicians who find that probabilistic explanations replicate their way of diagnosing (Kononenko, 1993). Similarly, Naive Bayes' explanations in terms of the sum of information gains is very transparent, as it is easily grasped by users like physicians who find that explanations replicate their way of diagnosing (Kononenko, 1993).

Finally, decision trees and NB generally have different operational profiles, when one is very accurate the other is not and vice versa. On the contrary, decision trees and rule classifiers have a similar operational profile. SVM and ANN have also a similar operational profile. No single learning algorithm can uniformly outperform other algorithms over all datasets. Features of learning techniques are compared in Table 4 (from evidence of existing empirical and theoretical studies).

	Decision Trees	Neural Networks	Naïve Bayes	kNN	SVM	Rule-learners
Accuracy in general	**	***	*	**	****	**
Speed of learning with respect to number of attributes and the number of instances	***	*	****	****	*	**
Speed of classification	****	****	****	*	****	****
Tolerance to missing values	***	*	****	*	**	**
Tolerance to irrelevant attributes	***	*	**	**	****	**
Tolerance to redundant attributes	**	**	*	**	***	**
Tolerance to highly interdependent attributes (e.g. parity problems)	**	***	*	*	***	**
Dealing with discrete/binary/continuous attributes	****	***(not discrete)	***(not continuous)	***(not directly discrete)	**(not discrete)	***(not directly continuous)
Tolerance to noise	**	**	***	*	**	*
Dealing with danger of overfitting	**	*	***	***	**	**
Attempts for incremental learning	**	***	****	****	**	*
Explanation ability/transparency of knowledge/classifications	****	*	****	**	*	****
Model parameter handling	***	*	****	***	*	***

Table 4. Comparing learning algorithms (**** stars represent the best and * star the worst performance)

When faced with the decision "Which algorithm will be most accurate on our classification problem?", the simplest approach is to estimate the accuracy of the candidate algorithms on the problem and select the one that appears to be most accurate. The concept of combining classifiers is proposed as a new direction for the improvement of the performance of individual classifiers. The goal of classification result integration algorithms is to generate more certain, precise and accurate system results. Numerous methods have been suggested for the creation of ensemble of classifiers (Dietterich, 2000). Although or perhaps because many methods of ensemble creation have been proposed, there is as yet no clear picture of which method is best (Villada and Drissi, 2002). Thus, an active area of research in supervised learning is the study of methods for the construction of good ensembles of classifiers. Mechanisms that are used to build ensemble of classifiers include: i) using different subsets of training data with a single learning method, ii) using different training parameters with a single training method (e.g., using different initial weights for each neural network in an ensemble) and iii) using different learning methods.

9 Conclusions

This paper describes the best-known supervised techniques in relative detail. We should remark that our list of references is not a comprehensive list of papers discussing supervised methods: our aim was to produce a critical review of the key ideas, rather than a simple list of all publications which had discussed or made use of those ideas. Despite this, we hope that the references cited cover the major theoretical issues, and provide access to the main branches of the literature dealing with such methods, guiding the researcher in interesting research directions.

The key question when dealing with ML classification is not whether a learning algorithm is superior to others, but under which conditions a particular method can significantly outperform others on a given application problem. Meta-learning is moving in this direction, trying to find functions that map datasets to algorithm performance (Kalousis and Gama, 2004). To this end, meta-learning uses a set of attributes, called meta-attributes, to represent the characteristics of learning tasks, and searches for the correlations between these attributes and the performance of learning algorithms. Some characteristics of learning tasks are: the number of instances, the proportion of categorical attributes, the proportion of missing values, the entropy of classes, etc. Brazdil et al. (2003) provided an extensive list of information and statistical measures for a dataset.

After a better understanding of the strengths and limitations of each method, the possibility of integrating two or more algorithms together to solve a problem should be investigated. The objective is to utilize the strengthes of one method to complement the weaknesses of another. If we are only interested in the best possible classification accuracy, it might be difficult or impossible to find a single classifier that performs as well as a good ensemble of classifiers. Despite the obvious advantages, ensemble methods have at least three weaknesses. The first weakness is increased storage as a direct consequence of the requirement that all component classifiers, instead of a single classifier, need to be stored after training. The total storage depends on the size of each component classifier itself and the size of the ensemble (number of classifiers in the ensemble). The second weakness is increased computation because in order to classify an input query, all component classifiers (instead of a single classifier) must be processed. The last weakness is decreased comprehensibility. With involvement of multiple classifiers in decision-making, it is more difficult for non-expert users to perceive the underlying reasoning process leading to a decision. A first attempt for extracting meaningful rules from ensembles was presented in (Wall et al, 2003).

For all these reasons, the application of ensemble methods is suggested only if we are only interested in the best possible classification accuracy. Another time-consuming attempt that tried to increase the classification accuracy without decreasing comprehensibility is the wrapper feature selection procedure (Guyon & Elissee, 2003). Theoretically, having more features should result in more discriminating power. However, practical experience with machine learning algorithms has shown that this is not always the case. Wrapper methods wrap the feature selection around the induction algorithm to be used, using cross-validation to predict the benefits of adding or removing a feature from the feature subset used.

Finally, many researchers in machine learning are accustomed to dealing with flat files and algorithms that run in minutes or seconds on a desktop platform. For these researchers, 100,000 instances with two dozen features is the beginning of the range of “very large” datasets. However, the database community deals with gigabyte databases. Of course, it is unlikely that all the data in a data warehouse would be mined simultaneously. Most of the current learning algorithms are computationally expensive and require all data to be resident in main memory, which is clearly untenable for many realistic problems and databases. An orthogonal approach is to partition the data, avoiding the need to run algorithms on very large datasets. Distributed machine learning involves breaking the dataset up into subsets, learning from these subsets concurrently and combining the results (Basak and Kothari, 2004). Distributed agent systems can be used for this parallel execution of machine learning processes (Klusch et al., 2003). Non-parallel machine learning algorithms can still be applied on local data (relative to the agent) because information about other data sources is not necessary for local operations. It is the responsibility of agents to integrate the information from numerous local sources in collaboration with other agents.

References

[1] Acid, S. and de Campos. L.M. (2003). Searching for Bayesian Network Structures in the Space of Restricted Acyclic Partially Directed Graphs. *Journal of Artificial Intelligence Research* 18: 445-490.

[2] Aha, D. (1997). *Lazy Learning*. Dordrecht: Kluwer Academic Publishers.

[3] An, A., Cercone, N. (1999), Discretization of continuous attributes for learning classification rules. Third Pacific-Asia Conference on Methodologies for Knowledge Discovery & Data Mining, 509-514.

[4] An, A., Cercone, N. (2000), Rule Quality Measures Improve the Accuracy of Rule Induction: An Experimental Approach, Lecture Notes in Computer Science, Volume 1932, Pages 119-129.

[5] Auer P. & Warmuth M. (1998). Tracking the Best Disjunction. *Machine Learning* 32: 127–150.

[6] Baik, S. Bala, J. (2004), A Decision Tree Algorithm for Distributed Data Mining: Towards Network Intrusion Detection, Lecture Notes in Computer Science, Volume 3046, Pages 206 – 212.

[7] Barto, A. G. & Sutton, R. (1997). *Introduction to Reinforcement Learning.* MIT Press.

[8] Batista, G., & Monard, M.C., (2003), An Analysis of Four Missing Data Treatment Methods for Supervised Learning, Applied Artificial Intelligence, vol. 17, pp.519-533.

[9] Basak., J., Kothari, R. (2004), A Classification Paradigm for Distributed Vertically Partitioned Data. Neural Computation, 16(7):1525-1544.

[10] Blum, A. (1997), Empirical Support for Winnow and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain, Machine Learning, Volume 26, Issue 1, Pages 5-23.

[11] Bonarini, A. (2000), An Introduction to Learning Fuzzy Classifier Systems, Lecture Notes in Computer Science, Volume 1813, Pages 83-92.

[12] Bouckaert, R. (2003). Choosing between two learning algorithms based on calibrated tests. Proc 20th Int Conf on Machine Learning, pp. 51-58. Morgan Kaufmann.

[13] Bouckaert, R. (2004), Naive Bayes Classifiers That Perform Well with Continuous Variables, Lecture Notes in Computer Science, Volume 3339, Pages 1089 – 1094.

[14] Brazdil P., Soares C. and Da Costa J. (2003), Ranking Learning Algorithms: Using IBL and Meta-Learning on Accuracy and Time Results, Machine Learning, 50: 251-277.

[15] Breiman L., Friedman J.H., Olshen R.A., Stone C.J. (1984) Classification and Regression Trees, Wadsforth International Group.

[16] Breiman, L., Bagging Predictors. Machine Learning, 24 (1996) 123-140.

[17] Breslow, L. A. & Aha, D. W. (1997). Simplifying decision trees: A survey. *Knowledge Engineering Review 12:* 1–40.

[18] Brighton, H. & Mellish, C. (2002), Advances in Instance Selection for Instance-Based Learning Algorithms. *Data Mining and Knowledge Discovery* 6: 153–172.

[19] Bruha. I. (2000), From machine learning to knowledge discovery: Survey of preprocessing and postprocessing. , Intelligent Data Analysis, Vol. 4, pp. 363-374.

[20] Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery.* 2(2):1-47.

[21] Camargo, L. S. & Yoneyama, T. (2001). Specification of Training Sets and the Number of Hidden Neurons for Multilayer Perceptrons. *Neural Computation* 13: 2673–2680.

[22] Castellano, G., Fanelli, A., & Pelillo, M. (1997). An iterative pruning algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks 8:* 519–531.

[23] Cestnik, B., Kononenko, I., Bratko, I., (1987). Assistant 86: A knowledge elicitation tool for sophisticated users. In: Proceedings of the Second European Working Session on Learning. pp. 31-45.

[24] Cestnik, B. (1990), Estimating probabilities: A crucial task in machine learning. In Proceedings of the European Conference on Artificial Intelligence, pages 147-149.

[25] Cheng, J. & Greiner, R. (2001). Learning Bayesian Belief Network Classifiers: Algorithms and System, In Stroulia, E. & Matwin, S. (ed.), *AI 2001*, 141-151, LNAI 2056,

[26] Cheng, J., Greiner, R., Kelly, J., Bell, D., & Liu, W. (2002). Learning Bayesian networks from data: An information-theory based approach. *Artificial Intelligence* 137: 43–90.

[27] Chickering, D.M. (2002). Optimal Structure Identification with Greedy Search. Journal of Machine Learning Research, Vol. 3, pp 507-554.

[28] Clark, P., Niblett, T. (1989), The CN2 Induction Algorithm. Machine Learning, 3(4):261-283.

[29] Cohen, W. (1995), Fast Effective Rule Induction. In *Proceedings of ICML-95,* 115-123.

[30] Cover, T., Hart, P. (1967), Nearest neighbor pattern classification. IEEE Transactions on Information Theory, 13(1): 21–7.

[31] Cowell, R.G. (2001). Conditions Under Which Conditional Independence and Scoring Methods Lead to Identical Selection of Bayesian Network Models. Proc. 17th International Conference on Uncertainty in Artificial Intelligence.

[32] Crammer, K. & Singer, Y. (2002). On the Learnability and Design of Output Codes for Multiclass Problems. *Machine Learning* 47: 201–233.

[33] Cristianini, N. & Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods.* Cambridge University Press, Cambridge.

[34] Csiszar, I. (1996), Maxent, mathematics, and information theory. In K. Hanson and R. Silver, editors, Maximum Entropy and Bayesian Methods. Kluwer Academic Publishers.

[35] De Mantaras & Armengol E. (1998). Machine learning from examples: Inductive and Lazy methods. *Data & Knowledge Engineering* 25: 99-123.

[36] Dietterich, T. G. (1998), Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. Neural Computation, 10(7) 1895–1924.

[37] Dietterich, T. G. (2000). An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization, *Machine Learning* 40: 139–157.

[38] Domingos, P. & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning* 29: 103-130.

[39] Dutton, D. & Conroy, G. (1996), A review of machine learning, *Knowledge Engineering Review* 12: 341-367.

[40] Eklund, P., Hoang, A. (2002), A Performance Survey of Public Domain Machine Learning Algorithms Technical Report, School of Information Technology, Griffith University.

[41] Elomaa, T. & Rousu, J. (1999). General and Efficient Multisplitting of Numerical Attributes. *Machine Learning* 36, 201–244.

[42] Elomaa T. (1999). The biases of decision tree pruning strategies. Lecture Notes in Computer Science 1642. Springer, pp. 63-74.

[43] Flach, P.A. & Lavrac, N. (2000). The role of feature construction in inductive rule learning. De Raedt, L. & Kramer, S., (ed.), In *Proceedings of the ICML2000 workshop on Attribute-Value Learning and Relational Learning: Bridging the Gap*, Stanford University.

[44] Frank, E. & Witten, I. (1998). Generating Accurate Rule Sets Without Global Optimization. In Shavlik, J., (eds), *Machine Learning: Proceedings of the Fifteenth International Conference*, Morgan Kaufmann Publishers, San Francisco, CA.

[45] Freund, Y. & Schapire, R. (1999), Large Margin Classification Using the Perceptron Algorithm, *Machine Learning* 37: 277–296.

[46] Friedman, J.H. (1989), Regularized Discriminant Analysis. Journal of the American Statistical Association.

[47] Friedman, N., Geiger, D. & Goldszmidt M. (1997). Bayesian network classifiers. *Machine Learning* 29: 131-163.

[48] Friedman, N. & Koller, D. (2003). Being Bayesian About Network Structure: A Bayesian Approach to Structure Discovery in Bayesian Networks. *Machine Learning* 50(1): 95-125.

[49] Furnkranz, J. (1997). Pruning algorithms for rule learning. *Machine Learning* 27: 139-171.

[50] Furnkranz, J. (1999). Separate-and-Conquer Rule Learning. *Artificial Intelligence Review* 13: 3-54.

[51] Furnkranz, J. (2001). Round Robin Rule Learning. *In Proceedings of the 18th International Conference on Machine Learning (ICML-01),* 146-153.

[52] Furnkranz, J., Flach, P. (2005), ROC 'n' Rule Learning—Towards a Better Understanding of Covering Algorithms, Machine Learning, Volume 58 (1), pp. 39 – 77.

[53] Gama, J. & Brazdil, P. (1999). Linear Tree. *Intelligent Data Analysis* 3: 1-22

[54] Gehrke, J., Ramakrishnan, R. & Ganti, V. (2000), RainForest—A Framework for Fast Decision Tree Construction of Large Datasets, Data Mining and Knowledge Discovery, Volume 4, Issue 2 - 3, Jul 2000, Pages 127 - 162

[55] Genton, M. (2001). Classes of Kernels for Machine Learning: A Statistics Perspective. *Journal of Machine Learning Research* 2: 299-312.

[56] Good I.J. (1950), Probability and the Weighing of Evidence, London, Charles Grin.

[57] Guo, G., Wang, H., Bell, D., Bi, Y., Greer, K. (2003), KNN Model-Based Approach in Classification, Lecture Notes in Computer Science, Volume 2888, Pages 986 – 996.

[58] Guyon, I, Elissee, A. (2003), An introduction to variable and feature selection. Journal of Machine Learning Research, 3:1157 1182.

[59] Hunt E., Martin J & Stone P. (1966), Experiments in Induction, New York, Academic Press.

[60] Heckerman, D., Meek, C. & Cooper, G. (1999). A Bayesian Approach to Causal Discovery. In Glymour, C. and G. Cooper, (ed.), *Computation, Causation, and Discovery*, 141-165. MIT Press.

[61] Hodge, V., Austin, J. (2004), A Survey of Outlier Detection Methodologies, Artificial Intelligence Review, Volume 22, Issue 2, pp. 85-126.

[62] Japkowicz N. and Stephen, S. (2002), The Class Imbalance Problem: A Systematic Study Intelligent Data Analysis, Volume 6, Number 5.

[63] Jain, A.K., Murty, M. N., and Flynn, P. (1999), Data clustering: A review, ACM Computing Surveys, 31(3): 264–323.

[64] Jensen, F. (1996). *An Introduction to Bayesian Networks*. Springer.

[65] Jordan, M.I. (1998), *Learning in Graphical Models*. MIT Press, Cambridge, MA.

[66] Kalousis A., Gama, G. (2004), On Data and Algorithms: Understanding Inductive Performance, *Machine Learning 54*: 275–312.

[67] Keerthi, S. & Gilbert, E. (2002). Convergence of a Generalized SMO Algorithm for SVM Classifier Design. *Machine Learning* 46: 351–360.

[68] Kivinen, J. (2002), Online Learning of Linear Classifiers, *Advanced Lectures on Machine Learning: Machine Learning Summer School 2002*, Australia, February 11-22, ISSN: 0302-9743, pp. 235 – 257.

[69] Klusch, M., Lodi, S., Moro, G. (2003), Agent-Based Distributed Data Mining: The KDEC Scheme. In Intelligent Information Agents: The AgentLink Perspective, LNAI 2586, pages 104-122. Springer.

[70] Kon, M. & Plaskota, L. (2000), Information complexity of neural networks, *Neural Networks* 13: 365–375.

[71] Kononenko, I. (1991), "Semi-Naive Bayesian Classifier", In Proceedings of the sixth European Working Session on Learning, 206-219.

[72] Kononenko, I. (1993), Inductive and Bayesian learning in medical diagnosis. Applied Artificial Intelligence 7(4): 317-337.

[73] Kononenko, I. (1994), 'Estimating attributes: analysis and extensions of Relief'. In: L. De Raedt and F. Bergadano (eds.): Machine Learning: ECML-94. pp. 171–182, Springer Verlag.

[74] Kubat, Miroslav Cooperson Martin (2001), A reduction technique for nearest-neighbor classification: Small groups of examples. Intell. Data Anal. 5(6): 463-476.

[75] Lindgren, T. (2004), Methods for Rule Conflict Resolution, Lecture Notes in Computer Science, Volume 3201, Pages 262 – 273.

[76] Littlestone, N. & Warmuth, M. (1994). The weighted majority algorithm. *Information and Computation 108*(2): 212–261.

[77] Liu, H. and H. Motoda (2001), Instance Selection and Constructive Data Mining, Kluwer, Boston.

[78] Madden, M. (2003), The Performance of Bayesian Network Classifiers Constructed using Different Techniques, Proceedings of European Conference on Machine Learning, Workshop on Probabilistic Graphical Models for Classification, pp. 59-70.

[79] Markovitch S. & Rosenstein D. (2002), Feature Generation Using General Construction Functions, *Machine Learning* 49: 59-98.

[80] McSherry, D. (1999). Strategic induction of decision trees. Knowledge-Based Systems, 12(5-6):269-275.

[81] Michalski, R. S., Chilausky, R. L. (1980), Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing and expert system for soybean disease diagnosis. Policy Analysis and Information Systems, 4(2)..

[82] Mika, S., Rätsch, G., Weston, J., Schölkopf, B. and Müller, K.-R. (1999), Fisher discriminant analysis with kernels. In Y.-H. Hu, J. Larsen, E. Wilson, and S. Douglas, editors, Neural Networks for Signal Processing IX, pages 41-48. IEEE.

[83] Mitchell, T. (1997). *Machine Learning.* McGraw Hill.

[84] Murthy, (1998), Automatic Construction of Decision Trees from Data: A Multi-Disciplinary Survey, *Data Mining and Knowledge Discovery* 2: 345–389.

[85] Nadeau, C. and Bengio, Y. (2003), Inference for the generalization error. In Machine Learning 52:239–281.

[86] Neocleous, C. & Schizas, C., (2002), Artificial Neural Network Learning: A Comparative Review, LNAI 2308, pp. 300–313, Springer-Verlag Berlin Heidelberg.

[87] Nilsson, N.J. (1965). Learning machines. New York: McGraw-Hill.

[88] Olcay Taner Yıldız, Onur Dikmen (2007), Parallel univariate decision trees, Pattern Recognition Letters, Volume 28 , Issue 7 (May 2007), Pages: 825-832.

[89] Okamoto, S., Yugami, N. (2003), Effects of domain characteristics on instance-based learning algorithms. Theoretical Computer Science 298, 207-233.

[90] Parekh, R., and Yang, J., and Honavar, V. (2000), Constructive Neural Network Learning Algorithms for Pattern Classification. IEEE Transactions on Neural Networks. 11(2), pp. 436-451.

[91] Platt, J. (1999). Using sparseness and analytic QP to speed training of support vector machines. In Kearns, M., Solla, S. & Cohn, D. (ed.), *Advances in neural information processing systems*. MIT Press.

[92] Quinlan, J.R. (1979), "Discovering rules by induction from large collections of examples", D. Michie ed., Expert Systems in the Microelectronic age, pp. 168-201.

[93] Quinlan, J.R. (1993). C4.5: Programs for machine learning. Morgan Kaufmann, San Francisco

[94] Rastogi, R. & Shim, K. (2000). PUBLIC: A Decision Tree Classifier that Integrates Building and Pruning. *Data Mining and Knowledge Discovery* 4: 315–344.

[95] Reinartz T. (2002), A Unifying View on Instance Selection, Data Mining and Knowledge Discovery, 6, 191–210, Kluwer Academic Publishers.

[96] Robert, J., Howlett L.C.J. (2001), Radial Basis Function Networks 2: New Advances in Design.

[97] Rosenblatt, F., (1962), Principles of Neurodynamics. Spartan, New York

[98] Roy, A. (2000), On connectionism, rule extraction, and brain-like learning. IEEE Transactions on Fuzzy Systems, 8(2): 222-227.

[99] Ruggieri, S. (2001). Efficient C4.5. *IEEE Transactions on Knowledge and Data Engineering* 14 (2): 438-444.

[100] Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986), Learning internal representations by error propagation. In: Rumelhart D E, McClelland J L et al. (eds.) Parallel Distributed Processing: Explorations in the Microstructure of Cognition. MIT Press, Cambridge, MA, Vol. 1, pp. 318-362.

[101] Saad, D. (1998). *Online learning in neural networks.* London: Cambridge University Press.

[102] Sanchez, J., Barandela, R., Ferri, F. (2002), On Filtering the Training Prototypes in Nearest Neighbour Classification, Lecture Notes in Computer Science, Volume 2504, Pages 239 - 248

[103] Scholkopf, C., Burges, J. C. & Smola, A. J. (1999). *Advances in Kernel Methods*. MIT Press.

[104] Setiono R. and Loew, W. K. (2000), FERNN: An algorithm for fast extraction of rules from neural networks, Applied Intelligence 12, 15-25.

[105] Siddique, M. N. H. and Tokhi, M. O. (2001), Training Neural Networks: Backpropagation vs. Genetic Algorithms, IEEE International Joint Conference on Neural Networks, Vol. 4, pp. 2673–2678.

[106] Smyth, P, Goodman, R., M. (1990), Rule induction using information theory, In G. Piatetsky Shapiro and W. Frawley (eds), Knowledge Discovery in Databases, MIT Press.

[107] Tjen-Sien, L., Wei-Yin, L., Yu-Shan, S. (2000). A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms. *Machine Learning* 40: 203–228.

[108] Utgoff, P., Berkman, N., Clouse, J. (1997), Decision Tree Induction Based on Efficient Tree Restructuring, Machine Learning, Volume 29, Issue 1, Pages: 5 – 44.

[109] Vapnik, V. (1995), The Nature of Statistical Learning Theory}. Springer Verlag.

[110] Veropoulos, K., Campbell, C. & Cristianini, N. (1999). Controlling the Sensitivity of Support Vector Machines. *In Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI99).*

[111] Villada, R. & Drissi, Y. (2002). A Perspective View and Survey of Meta-Learning. *Artificial Intelligence Review* 18: 77–95.

[112] Vivarelli, F. & Williams, C. (2001). Comparing Bayesian neural network algorithms for classifying segmented outdoor images. *Neural Networks* 14: 427-437.

[113] Wall, R., Cunningham, P., Walsh, P., Byrne, S. (2003), Explaining the output of ensembles in medical decision support on a case by case basis, Artificial Intelligence in Medicine, Vol. 28(2) 191-206.

[114] Weigend, A. S., Rumelhart, D. E., & Huberman, B. A. (1991). Generalization by weight-elimination with application to forecasting. In: R. P. Lippmann, J. Moody, & D. S. Touretzky (eds.), Advances in Neural Information Processing Systems 3, San Mateo, CA: Morgan Kaufmann.

[115] Wettschereck, D., Aha, D. W. & Mohri, T. (1997). A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms. *Artificial Intelligence Review* 10:1–37.

[116] Wilson, D. R. & Martinez, T. (2000). Reduction Techniques for Instance-Based Learning Algorithms. *Machine Learning* 38: 257–286.

[117] Witten, I. & Frank, E. (2005), "Data Mining: Practical machine learning tools and techniques", 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

[118] Yam, J. & Chow, W. (2001). Feedforward Networks Training Speed Enhancement by Optimal Initialization of the Synaptic Coefficients. *IEEE Transactions on Neural Networks* 12: 430-434.

[119] Yang, Y., Webb, G. (2003), On Why Discretization Works for Naive-Bayes Classifiers, Lecture Notes in Computer Science, Volume 2903, Pages 440 – 452.

[120] Yen, G. G. and Lu, H. (2000), Hierarchical genetic algorithm based neural network design, In: IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks, pp. 168–175.

[121] Yu, L., Liu, H. (2004), Efficient Feature Selection via Analysis of Relevance and Redundancy, JMLR, 5(Oct):1205-1224.

[122] Zhang, G. (2000), Neural networks for classification: a survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C 30(4): 451-462.

[123] Zhang, S., Zhang, C., Yang, Q. (2002). Data Preparation for Data Mining. Applied Artificial Intelligence, Volume 17, pp. 375 - 381.

[124] Zheng, Z. (1998). Constructing conjunctions using systematic search on decision trees. *Knowledge Based Systems Journal 10:* 421–430.

[125] Zheng, Z. (2000). Constructing *X*-of-*N* Attributes for Decision Tree Learning. *Machine Learning* 40: 35–75.

[126] Zhou, Z. (2004), Rule Extraction: Using Neural Networks or For Neural Networks?, Journal of Computer Science and Technology, Volume 19, Issue 2, Pages: 249 – 253.