Machine learning: A review of classification and combining techniques

Article *in* Artificial Intelligence Review · November 2006

DOI: 10.1007/s10462-007-9052-3 · Source: DBLP

Sotiris Kotsiantis
University of Patras
285 PUBLICATIONS **20,246** CITATIONS
SEE PROFILE

I. D. Zaharakis
Technological Educational Institute of Western Greece
36 PUBLICATIONS **3,635** CITATIONS
SEE PROFILE

P. E. Pintelas
University of Patras
194 PUBLICATIONS **10,325** CITATIONS
SEE PROFILE

Artif Intell Rev
DOI 10.1007/s10462-007-9052-3

Machine learning: a review of classification and combining techniques

S. B. Kotsiantis · I. D. Zaharakis · P. E. Pintelas

Abstract 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 (Logic-based techniques, Perceptron-based techniques) and Statistics (Bayesian Networks, Instance-based techniques). 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 classification algorithms and the recent attempt for improving classification accuracy—ensembles of classifiers.

Keywords Classifiers · Data mining techniques · Intelligent data analysis · Learning algorithms

1 Introduction

There are several applications for Machine Learning (ML), the most significant of which is predictive data mining. Every instance in any dataset used by machine learning algorithms is represented using the same set of features. The features may be continuous, categorical

S. B. Kotsiantis (✉) · P. E. Pintelas
Department of Computer Science and Technology, University of Peloponnese, Peloponnese, Greece
e-mail: sotos@math.upatras.gr

S. B. Kotsiantis · P. E. Pintelas
Educational Software Development Laboratory, Department of Mathematics, University of Patras, P. O. Box 1399, Patras, Greece

P. E. Pintelas
e-mail: pintelas@math.upatras.gr

I. D. Zaharakis
Computer Technology Institute, Patras, Greece
e-mail: john@math.upatras.gr

Journal: 10462 MS: AIRE418 CMS: 10462_2007_9052_article ☐ TYPESET ✓ DISK ☐ LE ✓ CP Disp.:2007/10/26 Pages: 32 Layout: Small

or binary. If instances are given with known labels (the corresponding correct outputs) then the learning is called *supervised*, in contrast to *unsupervised learning,* where instances are unlabeled (Jain et al. 1999).

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. We have limited our references to recent refereed journals, published books and conferences. A brief review of what ML includes can be found in (Dutton and Conroy 1996). De Mantaras and Armengol (1998) also presented a historical survey of logic and instance based learning classifiers.

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 strengths 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.

Our next section covers wide-ranging issues of supervised machine learning such as data pre-processing and feature selection. Logic-based learning techniques are described in Sect. 3, whereas perceptron-based techniques are analyzed in Sect. 4. Statistical techniques for ML are covered in Sect. 5. Section 6 deals with the newest supervised ML technique—Support Vector Machines (SVMs). In Sect. 7, a representative algorithm for each learning technique is compared to a number of datasets in order for researchers to have baseline accuracy for new algorithms in these well-known datasets. Section 8 presents the recent attempt for improving classification accuracy—ensembles of classifiers. Finally, the last section concludes this work.

2 General issues of supervised learning algorithms

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).

2.1 Data preparation and data pre-processing

What can be wrong with data? There is a hierarchy of problems that are often encountered in data preparation and pre-processing:

- Impossible values have been inputted.
- Unlikely values have been inputted.
- No values have been inputted (missing values).
- Irrelevant input features are present in the data at hand.

Impossible values should be checked for by the data handling software, ideally at the point of input so that they can be re-entered. These errors are generally straightforward, such as coming across negative prices when positive ones are expected. If correct values cannot be entered, the problem is converted into missing value category, by simply removing the data.

Table 1 Examples for the use of variable-by-variable data cleansing

Problems	Metadata	Examples/Heuristics
Illegal values	Cardinality	e.g., cardinality (gender) > 2 indicates problem
	Max, min	Max, min should not be outside of permissible range
	Variance, deviation	Variance, deviation of statistical values should not be higher than threshold
Misspellings	Feature values	Sorting on values often brings misspelled values next to correct values

Variable-by-variable data cleansing is a filter approach for unlikely values (those values that are suspicious due to their relationship to a specific probability distribution, that is to say, a normal distribution with a mean of 5, a standard deviation of 3, but a suspicious value of 10). Table 1 shows examples of how such metadata can help in detecting a number of possible data quality problems.

Hodge and Austin (2004) have recently introduced a survey of contemporary techniques for outlier (noise) detection. These researchers have identified the techniques' advantages and disadvantages. While the focus above is on analytical methods, the use of visualization can also often be a powerful tool. Visualization is particularly good at picking out bad values that occur in a regular pattern. However, care is needed in distinguishing between natural variability and the presence of bad values, since data is often more dispersed that we think.

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 Metoda 2001). It reduces data and enables a data-mining algorithm to function and work effectively with very large datasets. There are a variety of procedures for sampling instances from a large dataset. The most well-known are (Reinartz 2002):

- *Random sampling,* which selects a subset of instances randomly.
- *Stratified sampling,* which is applicable when the class values are not uniformly distributed in the training sets. Instances of the minority class(es) are selected with greater frequency in order to even out the distribution. Other techniques for handling imbalanced datasets can be found in (Japkowicz and Stephen 2002).

Incomplete data is an unavoidable problem in dealing with most real world data sources. Generally, there are some important factors to be taken into account when processing unknown feature values. One of the most important ones is the source of "unknown-ness": (i) a value is missing because it was forgotten or lost; (ii) a certain feature is not applicable for a given instance (e.g., it does not exist for a given instance); (iii) for a given observation, the designer of a training set does not care about the value of a certain feature (so-called "don't-care values)."

Depending on the circumstances, researchers have a number of methods to choose from to handle missing data (Batista and Monard 2003):

- *Method of ignoring instances with unknown feature values:* This method is simplest: ignore any instances, which have at least one unknown feature value.
- *Most common feature value:* The value of the feature that occurs most often is selected to be the value for all the unknown values of the feature.

- *Most common feature value in class:* This time the value of the feature, which occurs most commonly within the same class is selected to be the value for all the unknown values of the feature.
- *Mean substitution:* Researchers substitute a feature's mean value (computed from available cases) to fill in missing data values on the remaining cases. A more sophisticated solution than using the "general" feature mean is to use the feature mean for all samples belonging to the same class to fill in the missing value.
- *Regression or classification methods:* A regression or classification model based on the complete case data for a given feature is developed. This model treats the feature as the outcome and uses the other features as predictors.
- *Hot deck inputting:* The most similar case to the case with a missing value is identified, and then a similar case's Y value for the missing case's Y value is substituted.
- *Method of treating missing feature values as special values:* "Unknown" itself is treated as a new value for the features that contain missing values.

Feature subset selection is the process of identifying and removing as many irrelevant and redundant features as possible (Yu and Liu 2004). This reduces the dimensionality of the data and enables data mining algorithms to operate faster and more effectively. Generally, features are characterized as:

- Relevant: These are features have an influence on the output. Their role cannot be assumed by the rest.
- Irrelevant: Irrelevant features are defined as those features not having any influence on the output. Their values could be generated at random and not influence the output.
- Redundant: A redundancy exists whenever a feature can take the role of another (perhaps the simplest way to incur model redundancy).

Feature selection algorithms in general have two components: a selection algorithm that generates proposed subsets of features and attempts to find an optimal subset and an evaluation algorithm that determines how "good" a proposed feature subset is. However, without a suitable *stopping criterion,* the feature selection process may run repeatedly through the space of subsets, taking up valuable computational time. Stopping criteria might be: (i) whether addition (or deletion) of any feature does not produce a better subset; and (ii) whether an optimal subset according to some evaluation function is obtained.

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 and 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.

2.2 Algorithm selection

The choice of which specific learning algorithm we should use is a critical step. 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, 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. A common method for comparing supervised ML algorithms is to perform statistical comparisons of the accuracies of trained classifiers on specific datasets (Bouckaert 2003). Several heuristic versions of the t-test have been developed to handle this issue (Dietterich 1998; Nadeau and Bengio 2003).

In the next sections, we will focus on the most important supervised machine learning techniques, starting with logic-based techniques.

3 Logic based algorithms

In this section we will concentrate on two groups of logic (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. 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.

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 but 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 1 presents a general pseudo-code for building decision trees.

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. 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 and 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. A comparative study of well-known pruning methods presented in (Elomaa 1999). However, Elomaa (1999) concluded that there is no single best pruning method.

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

```
DT(Instances,Target_feature,Features)
If all instances at the current node belong to the same category
  then create a leaf node of the corresponding class
 else
  {
   Find the features A that maximizes the goodness measure
   Make A the decision feature for the current node
    for each possible value v of A
     {
      add a new branch below node testing for A = v
      Instances_v := subset of Instances with A = v
       if Instances_v is empty
        then add a leaf with label the most common value of
           Target_feature in Instances;
       else
       {
        below the new branch add subtree
        DT(Instances_v,Target_feature,Features - {A})
       }
     }
  }
```

Fig. 1 Pseudo-code for building a decision tree

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 and Rousu 1999).

Decision trees are usually univariate since they use splits based on a single feature at each internal node. 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.

The most well-know algorithm in the literature for building decision trees is the C4.5 (Quinlan 1993). One of the latest studies that compare decision trees and other learning algorithms has been done by (Tjen-Sien et al. 2000). The study shows that C4.5 has a very good combination of error rate and speed. 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. 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

```
function LearnRuleSet(Target, Attrs, Examples, Threshold):
    LearnedRules := ∅
    Rule := LearnOneRule(Target, Attrs, Examples)
    while performance(Rule,Examples) > Threshold, do
        LearnedRules := LearnedRules ∪ {Rule}
        Examples := Examples \ {examples classified correctly by Rule}
        Rule := LearnOneRule(Target, Attrs, Examples)
    sort LearnedRules according to performance
    return LearnedRules
```

Fig. 2 Pseudocode for rule learners

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.

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. 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 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 Fig. 2, 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).

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. An and Cercone (2000) surveyed a number of statistical and empirical rule quality measures. 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. 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. 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).

Genetic algorithms (GAs) have also been used for learning sets of rules. Fidelis et al. (2000) used a genetic algorithm to learn binary concepts represented by a disjunctive set of propositional rules and it was found to be comparable in generalization accuracy to other learning algorithms. Assuming two binary features X_1, X_2 and the binary target value c, rule representation to chromosomes is:

$$\begin{array}{ccccccccccc} \textbf{IF} & X_1\text{=True} & \wedge & X_2\text{=False} & \textbf{THEN} & c\text{=True}, & \textbf{IF} & X_1\text{= False} & \wedge & X_2\text{=True} & \textbf{THEN} \; c\text{=False} \\ & 10 & & 01 & & 1 & & 01 & & 10 & 0 \end{array}$$

Note that there is a fixed length bit-string representation for each rule. The task of the genetic algorithm is to find good chromosomes. The goodness of a chromosome is represented in the GA by a function which is called the *fitness function* (Reeves and Rowe 2003). For the classification task, the fitness function typically scores the classification accuracy of the rule over a set of provided training instances. At the heart of the algorithm are operations which take the population of the present generation and produce the population of the next generation in such a way that the overall fitness of the population is increased. These operations repeat until some stopping criterion is met, such as a given number of chromosomes being processed or a chromosome of given quality produced. Three operations take the population of generation t and produce the new population of generation $t + 1$: selection, crossover, and mutation (Reeves and Rowe 2003).

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).

3.2.1 Inductive logic programming

The main limitation of propositional learners is their limited capacity to take into account available background knowledge. Learners that induce hypotheses in the form of logic programs (Horn clauses) are called inductive logic programming systems. De Mantaras and Armengol (1998) presented a survey of inductive logic programming.

The previously discussed logic-based methods can learn hypotheses which can be expressed in terms of propositional logic. *Inductive Logic Programming* (ILP) classifiers use framework of first order predicate logic. As with any learning system, this one can be quite complex and intractably difficult unless it is constrained by biases of some sort. One might restrict the program to Horn clauses, not allow recursion, and so on (De Raedt 1996). First, most algorithms favor those clauses which cover as many instances as possible, then, they turn to those clauses having a smaller number of features and finally they accept those clauses which have the least total number of values in the internal disjunctions.

ILP systems typically adopt the covering approach of rule induction systems. In a main loop, they construct a clause explaining some of the positive examples, add this clause to the hypothesis, remove the positive examples explained and repeat this until all positive examples are explained (the hypothesis is complete). In an inner loop, individual clauses are constructed by (heuristically) searching the space of possible clauses which was structured by a specialization or generalization operator. Typically, a search starts with a very general rule (clause with no conditions in the body), then proceeds to add literals (conditions) to this clause until it only covers (explains) positive examples (the clause is consistent). This search can be bound from below by using so-called "bottom clauses," constructed by least general generalization or inverse resolution/entailment.

There are three major patterns in which a logic program might be generalized: (i) replace some terms in a program clause by variables, (ii) remove literals from the body of a clause, (iii) add a clause to the program. Correspondingly, there are three patterns in which a logic program might be specialized: (i) replace some variables in a program clause by terms (a substitution), (ii) add literals to the body of a clause and (iii) remove a clause from the program. FOIL (Quinlan 1995) is an algorithm designed to learn a set of first order rules to predict a target predicate to be true. It differs from classifiers such as C4.5 in that it learns relations among features that are described with variables. PROGOL (Muggleton 1995) is another well-known first order rules learner.

A significant part of ILP research now goes under the heading of Relational Data Mining and is concerned with finding decision trees from multi-table relational databases (Dzeroski and Lavrac 2001). These ILP algorithms can be understood as a kind decision tree induction where each node of the decision tree is itself a sub-decision tree, and each sub-decision tree consists of nodes that make binary splits on several features using the background relations available. However, since ILP methods search a much larger space of possible rules in a more expressive language, they are computationally more demanding. Blockeel and De Raedt (1998) studied scaling up first-order logical decision trees with the TILDE

First order predicate logic rules:

- If Father (x,y) and Female(y) then Daughter(x,y)

Propositional logic rule with the same meaning:

- If (Father1=Bob) and (Name2 = Bob) and (Femal1 = True) then (Daughter1,2) = True

Fig. 3 First Order Rule versus Zero Order Rule

algorithm. A dataset is presented to TILDE in the form of a set of interpretations. Each interpretation consists of a number of Prolog facts, surrounded by a begin and end line. The background knowledge is simply a Prolog program.

ILP differs from most other forms of machine learning both by its use of an expressive representation language and its ability to make use of logically encoded background knowledge. This has allowed successful applications of ILP in areas such as molecular biology and natural language processing, which both have rich sources of background knowledge and both benefit from the use of conceptually expressive languages (Muggleton 1999).

When comparing ILP (first order predicate logic rules) and feature-value learning techniques (propositional logic rules), there is a trade-off between expressive power and efficiency (Dantsin et al. 2001). ILP techniques are typically more expressive (see Fig. 3) but also less efficient. If the available data has a standard tabular form, with rows being individual records (training instances) and columns being properties (features) used to describe the data, a data analyst has no reason to become interested in ILP if there is no expert knowledge (background knowledge). ILP can be used for data mining in multi-relational data mining tasks with data stored in relational databases and tasks with abundant expert knowledge of a relational or structural nature. Typically, each predicate will correspond to one relation in the relational database. Each fact in an interpretation is a tuple (instance) in the database, and an interpretation corresponds to a part of the database (a set of tuples). Background knowledge can be expressed by means of views as well as extensional tables.

4 Perceptron-based techniques

Other well-known algorithms are based on the notion of perceptron. Perceptron can be briefly described as:

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 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 and Warmuth 1994) is based on the perceptron idea. 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 and Warmuth (1998).

Freund and 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 and 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 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. Most perceptron-like algorithms cannot deal with numerical features, thus numerical features should be discretized before induction. 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.

4.1 Neural networks

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. Artificial Neural Networks have been created to try to solve this problem. Zhang (2000) provided an overview of existing work in Artificial Neural Networks (ANNs).

A multi-layer neural network consists of large number of units (neurons) joined together in a pattern of connections (Fig. 4). 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 (Fig. 4) allow signals to travel one way only, from input to output.

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 and Yoneyama 2001). Kon and 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 and Schizas 2002). However, the most well-known and widely used learning algorithm to estimate the values of

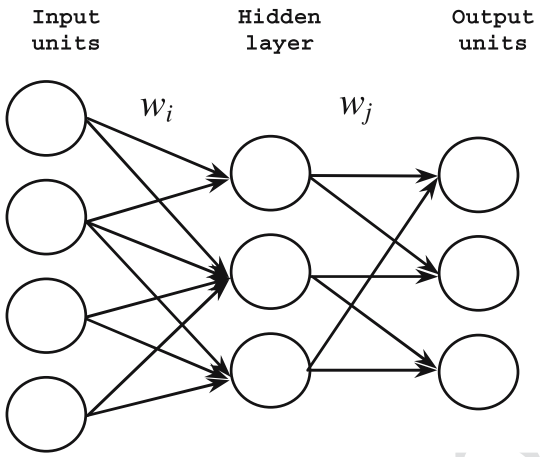

Fig. 4 Feed-forward ANN

the weights is the Back Propagation (BP) algorithm. The general rule for updating weights is: $\Delta W_j = \eta \delta_j O_i$ 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

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.

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 and Chow 2001). 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 and 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).

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.

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 Loew (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. 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).

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. Under this category of classification algorithms, one can find Bayesian networks and instance-based methods. A comprehensive book on Bayesian networks is Jensen's (1996).

5.1 Bayesian networks

A Bayesian Network (BN) is a graphical model for probability relationships among a set of variables (features). 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 $l \quad k$ of possible arcs in S encodes conditional independencies. Moreover, a feature (node) is conditionally independent from its non-descendants given its parents.

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).

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.

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. In Fig. 5 there is a pseudo-code for training BNs.

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

```
1. Let the class variant act as parents
2. Let all the features be independent
3. Find the dependence for each pair of features
4. Add the edges of dependence relationship in the DAG
5. Define the direction of the edges
6. Find all the conditional probability in the DAG
```

Fig. 5 Pseudo-code for training BN

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 and Koller (2003)s 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 $X_1, X_2, \ldots X_n$, returns the label c, which maximizes the posterior probability p$(c|X_1, X_2, \ldots X_n)$.

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: (a) declaring that a node is a root node, i.e., it has no parents, (b) declaring that a node is a leaf node, i.e., it has no children, (c) declaring that a node is a direct cause or direct effect of another node, (d) declaring that a node is not directly connected to another node, (e) declaring that two nodes are independent, given a condition-set and (f) providing partial nodes ordering, that is, declare that a node appears earlier than another node in the 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.

5.1.1 Naive Bayes classifiers

Naive Bayesian networks (NB) are very simple Bayesian networks which are composed of DAGs 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. Thus, the independence model (Naive Bayes) is based on estimating:

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

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). 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, by adding one to all numerators and adding the number of added ones to the denominator.

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 and 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 CR(Friedman et al. 1997). In this case, the network has the limitation that each feature can be related to only one other feature. An alternative generalisation, called "selective Bayesian classifiers", has been explored by (Ratanamahatana and Gunopulos 2003). They built independence Bayes model forms but included a feature selection stage, so that their final model did not need to include all of the features. The idea of this algorithm was to improve the classification performance by removing the irrelevant features or by removing one of the two totally correlated features.

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 and Webb 2003), although a researcher can use the normal distribution to calculate probabilities (Bouckaert 2004).

5.2 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.

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. 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. 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 kNN 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 kNN 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. Brighton and Mellish (2002) found that their ICF algorithm and RT3 algorithm (Wilson and 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 and 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.

6 Support vector machines

Support Vector Machines (SVMs) are the newest supervised machine learning technique. An excellent survey of SVMs can be found in (Burges 1998), and a more recent book is by (Cristianini and Shawe-Taylor 2000). 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.

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. Other data points are ignored. 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.

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 *sof margin* that accepts some misclassifications of the training instances (Veropoulos et al. 1999).

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 *feature space*, as opposed to the *input space* occupied by the training instances.

With an appropriately chosen feature space of sufficient dimensionality, any consistent training set can be made separable. A linear separation in 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 $\Phi : R^d \to H$. 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 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 feature space.

Training the SVM is done by solving Nth 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.

7 Experiment results

Supervised machine learning techniques are applicable in numerous domains (Saitta and Neri 1998). For the purpose of the present study, we used 45 well-known datasets mainly from the UCI repository (Blake and Merz 1998). The aim of this comparison is not to compare a representative algorithm for each machine learning technique in a number of datasets but to give a baseline accuracy for new algorithms based on each technique. These datasets come from: pattern recognition (anneal, iris, mushroom, vote, zoo), image recognition (ionosphere, segment, sonar, vehicle), medical diagnosis (breast-cancer, breast-w, colic, diabetes, haberman, heart-c, heart-h, heart-statlog, hepatitis, hypothyroid, lymphotherapy, primary-tumor, sick) commodity trading (autos, credit-a, credit-g, labor), music composition (waveform), computer games (badges, kr-vs-kp, monk1, monk2, monk3) and various control applications (balance). In Table 2, there is a brief description of these datasets.

The C4.5 algorithm (Quinlan 1993) was the representative of decision trees in our study. The most well-known learning algorithm used in estimating the values of the weights of a neural network — the Back Propagation (BP) algorithm (Mitchell 1997) — represented perceptron-based algorithms. The Sequential Minimal Optimization (or SMO) algorithm was the representative of the Support Vector Machines (Platt 1999). In our study, we also used the 3-NN algorithm, which combines robustness to noise and less time for classification than using a larger *k* for kNN (Wettschereck et al. 1997). Finally, the most-often used Naïve Bayes algorithm (Domingos and Pazzani 1997) represented BNs in our study.

We used the free available source code for these algorithms by (Witten and Frank 2005) for our experiments.

Apart from Witten and Frank (2005), who provide in their book a wide range of ML algorithms for free use, MLC++ is another free library of C++ classes for supervised machine learning (http://www.sgi.com/tech/mlc/).

The SMO algorithm is a 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 and Singer 2002).

In order to calculate the classifiers' accuracy, the whole training set was divided into ten mutually exclusive and equal-sized subsets and for each subset the classifier was trained on the union of all of the other subsets.

Then, cross validation was run 10 times for each algorithm and the mean value of the 10-cross validations was calculated. The results for each algorithm are summed in Table 3.

Table 2 Description of the datasets

	Instances	Categorical features	Numerical features	Missing values	Classes
Anneal	898	32	6	yes	6
Audiology	226	69	0	yes	24
Autos	205	10	15	yes	7
Badges	294	4	7	no	2
Balance	625	0	4	no	3
Breast-cancer	286	9	0	yes	2
Breast-w	699	0	9	yes	2
Colic	368	15	7	yes	2
Credit-a	690	9	6	yes	2
Credit-g	1,000	13	7	no	2
Diabetes	768	0	8	no	2
Glass	214	0	9	no	5
Haberman	306	0	3	no	2
Heart-c	303	7	6	yes	5
Heart-h	294	7	6	yes	5
Heart-statlog	270	0	13	no	2
Hepatitis	155	13	6	yes	2
Hypothyroid	3,772	22	7	yes	4
Ionosphere	351	34	0	no	2
Iris	150	0	4	no	3
kr-vs-kp	3,196	35	0	no	2
Labor	57	8	8	yes	2
Letter	20,000	0	16	no	26
Lymphotherapy	148	15	3	no	4
Mesocyc/radar	1,131	0	13	no	2
Monk1	124	6	0	no	2
Monk2	169	6	0	no	2
Monk3	122	6	0	no	2
Mushroom	8,124	22	0	yes	2
Nist-digits	1,000	0	784	no	10
Nursery	12,960	8	0	no	5
Primary-tumor	339	17	0	yes	21
Relation	2,201	3	0	no	2
Segment	2,310	0	19	no	7
Sick	3,772	22	7	yes	2
Sonar	208	0	60	no	2
Soybean	683	35	0	yes	19
Spambase	4,601	0	57	no	2
Vehicle	846	0	18	no	4
Vote	435	16	0	yes	2
Vowel	990	3	10	no	11
Waveform	5,000	0	40	no	3
Wine	178	0	13	no	3
Zoo	101	16	1	no	7

Below, we present our conclusions about the use of each technique. Discussions of all the pros and cons of each individual algorithm 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. In contrast, logic-based systems (e.g., decision

Table 3 The accuracy of each algorithm in each dataset

	NB	C4.5	3NN	RIPPER	BP	SMO
Anneal	86.59(±3.31)	98.57(±1.04)	97.29(±1.73)	98.13(±1.49)	94.97(±2.05)	96.89(±1.74)
Audiology	72.64(±6.10)	77.26(±7.47)	67.97(±7.73)	72.23(±7.87)	82.68(±8.16)	80.77(±8.40)
Autos	57.41(±10.77)	81.77(±8.78)	67.23(±11.07)	74.05(±9.68)	48.84(±7.42)	56.55(±10.14)
Badges	99.66(±0.17)	100(±0.0)	100(±0.0)	100(±0.0)	100(±0.0)	100(±0.0)
Balance	90.53(±1.67)	77.82(±3.42)	86.74(±2.72)	80.91(±3.31)	85.67(±2.55)	87.62(±2.64)
Breast-cancer	72.70(±7.74)	74.28(±6.05)	73.13(±5.54)	71.65(±6.65)	72.95(±6.61)	69.92(±6.41)
Breast-w	96.07(±2.18)	95.01(±2.73)	96.61(±1.97)	95.72(±2.30)	96.35(±2.07)	96.81(±2.13)
Colic	78.70(±6.20)	85.16(±5.91)	80.95(±5.53)	84.97(±5.71)	83.07(±5.81)	82.69(±6.12)
Credit-a	77.86(±4.18)	85.57(±3.96)	84.96(±4.44)	85.33(±4.06)	85.94(±3.69)	84.91(±3.97)
Credit-g	75.16(±3.48)	71.25(±3.17)	72.21(±3.25)	71.86(±3.88)	74.86(±3.89)	75.11(±3.94)
Diabetes	75.75(±5.32)	74.49(±5.27)	73.86(±4.55)	75.22(±4.86)	77.04(±4.85)	77.07(±4.14)
Glass	49.45(±9.50)	67.63(±9.31)	70.02(±7.94)	66.78(±8.57)	67.32(±8.64)	57.36(±8.77)
Haberman	75.06(±5.42)	71.05(±5.20)	69.77(±5.72)	72.72(±5.90)	74.2(±6.27)	73.4(±1.06)
Heart-c	83.34(±7.20)	76.94(±6.59)	81.82(±6.55)	79.05(±6.49)	82.98(±6.30)	84.03(±6.60)
Heart-h	83.95(±6.27)	80.22(±7.95)	82.33(±6.23)	79.26(±7.20)	84.16(±6.12)	83.26(±6.46)
Heart-statlog	83.59(±5.98)	78.15(±7.42)	79.11(±6.77)	78.70(±6.62)	83.30(±6.20)	83.81(±5.59)
Hepatitis	83.81(±9.70)	79.22(±9.57)	80.85(±9.20)	77.43(±9.14)	84.29(±8.85)	85.03(±7.53)
Hypothyroid	95.30(±0.73)	99.54(±0.36)	93.21(±0.81)	99.39(±0.39)	93.44(±0.45)	93.49(±0.40)
Ionosphere	82.17(±6.14)	89.74(±4.38)	86.02(±4.31)	89.30(±4.63)	87.07(±5.52)	87.93(±4.69)
Iris	95.53(±5.02)	94.73(±5.30)	95.20(±5.11)	93.93(±6.57)	84.80(±7.10)	84.87(±7.63)
kr-vs-kp	87.79(±1.91)	99.44(±0.37)	96.56(±1.00)	99.13(±0.44)	98.92(±0.60)	95.78(±1.20)
Labor	93.57(±10.27)	78.60(±16.58)	92.83(±9.86)	83.37(±15.27)	88.93(±11.86)	94.00(±10.37)
Letter	64.12(±0.94)	87.95(±0.61)	94.54(±0.23)	87.10(±0.45)	81.93(±0.62)	81.57(±0.19)
Lymphotherapy	83.13(±8.89)	75.84(±11.05)	81.74(±8.45)	77.36(±11.11)	82.26(±8.05)	85.94(±8.75)
Mesocyc/radar	65.77(±3.52)	74.80(±3.22)	73.62(±2.98)	76.19(±2.63)	75.23(±3.14)	74.52(±0.86)
Monk1	73.38(±12.49)	80.61(±11.34)	78.97(±11.89)	83.87(±16.34)	97.69(±5.99)	79.58(±11.99)
Monk2	56.83(±8.71)	57.75(±11.18)	54.74(±9.20)	56.21(±8.89)	100.00(±0.00)	58.70(±5.80)
Monk3	93.45(±6.57)	92.95(±6.68)	86.72(±9.99)	84.8(±9.27)	88.32(±8.49)	93.45(±6.57)
Mushroom	95.76(±0.73)	100.00(±0.00)	100.00(±0.00)	100.00(±0.03)	99.97(±0.11)	100.00(±0.00)
Nist-digits	76.18(±3.81)	71.46(±4.40)	84.79(±0.82)	70.00(±1.86)	88.32(±1.37)	88.11(±1.75)
Nursery	90.30(±0.72)	97.18(±0.46)	97.36(±0.24)	98.67(±0.35)	99.43(±1.05)	93.02(±0.34)
Primary-tumor	49.71(±6.46)	41.39(±6.94)	44.98(±6.43)	38.59(±6.13)	24.90(±1.65)	29.20(±5.81)

Table 3 continued

	NB	C4.5	3NN	RIPPER	BP	SMO
Relation	77.85(±2.40)	78.55(±2.10)	78.9(±1.80)	78.01(±2.04)	78.44(±2.04)	77.6(±2.39)
Segment	80.16(±2.10)	96.79(±1.29)	96.12(±1.19)	95.53(±1.16)	91.35(±1.90)	89.72(±1.72)
Sick	92.75(±1.36)	98.72(±0.55)	96.21(±0.82)	98.30(±0.70)	94.87(±0.76)	93.88(±0.10)
Sonar	67.71(±8.66)	73.61(±9.34)	83.76(±8.51)	75.45(±10.01)	78.67(±9.21)	77.88(±8.45)
Soybean	92.94(±2.92)	91.78(±3.19)	91.2(±3.18)	91.85(±2.77)	93.35(±2.68)	93.10(±2.76)
Spambase	79.56(±1.56)	92.68(±1.08)	89.80(±1.35)	92.67(±1.21)	91.22(±2.52)	90.48(±1.22)
Vehicle	44.68(±4.59)	72.28(±4.32)	70.21(±3.93)	68.32(±4.59)	81.11(±3.84)	74.08(±3.82)
Vote	90.02(±3.91)	96.57(±2.56)	93.08(±3.70)	95.70(±2.86)	96.32(±2.87)	96.23(±2.77)
Vowel	62.9(±4.38)	80.2(±4.36)	96.99(±2.13)	71.17(±5.14)	92.73(±3.14)	70.61(±3.86)
Waveform	80.01(±1.45)	75.25(±1.90)	77.67(±1.79)	79.14(±1.72)	86.56(±1.56)	86.94(±1.48)
Wine	97.46(±3.70)	93.2(±5.90)	95.85(±4.19)	93.14(±6.94)	98.02(±3.26)	98.76(±2.73)
Zoo	94.97(±5.86)	92.61(±7.33)	92.61(±7.33)	87.03(±6.43)	60.43(±3.06)	93.16(±5.92)

trees, rule learners) 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.

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

A number of recent studies have shown that the decomposition of a classifier's error into bias and variance terms can provide considerable insight into the prediction performance of the classifier (Bauer and Kohavi 1999). Bias measures the contribution to error of the central tendency of the classifier when trained on different data. Variance is a measure of the contribution to error of deviations from the central tendency. Bias and variance are evaluated with respect to a distribution of training sets, such as a distribution containing all possible training sets of a specified size for a specified domain.

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.

Although training time varies according to the nature of the application task and dataset, specialists generally agree on a partial ordering of the major classes of learning algorithms. For instance, 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.

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: all missing values must have been eliminated, before the process analysis begins either by deleting the records or features with missing values, or by inputting missing values.

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 and 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.

Furthermore, the number of model or runtime parameters to be tuned by the user is an indicator of an algorithm's ease of use. It can help in prior model selection based on the user's priorities and preferences: for a non specialist in data mining, an algorithm with few user-tuned parameters will certainly be more appealing, while a more advanced user might find a large parameter set an opportunity to control the data mining process more closely. As expected, neural networks and SVMs have more parameters than the remaining techniques.

Logic-based algorithms are all considered very easy to interpret, whereas neural networks and SVMs have notoriously poor interpretability. kNN 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 kNN; while the resulting classifier is not quite interpretable, the method itself is very 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. Moreover, decision trees and rules are credited with high transparency.

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 according to our experiments. SVM and ANN have also a similar operational profile. No single learning algorithm can uniformly outperform other algorithms over all datasets. 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. The following section provides a brief survey of methods for constructing ensembles.

8 Combining classifiers

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.

8.1 Different subsets of training data with a single learning method

Bagging is a method for building ensembles that uses different subsets of training data with a single learning method (Breiman 1996). Given a training set of size t, bagging draws *t* random instances from the dataset with replacement (i.e. using a uniform distribution). These *t* instances are learned, and this process is repeated several times. Since the draw is with replacement, usually the instances drawn will contain some duplicates and some omissions, as compared to the original training set. Each cycle through the process results in one classifier. After the construction of several classifiers, taking a vote of the predictions of each classifier produces the final prediction.

Breiman (1996) made the important observation that instability (responsiveness to changes in the training data) is a prerequisite for bagging to be effective. A committee of classifiers that all agree in all circumstances will give identical performance to any of its members in isolation. A variance reduction process will have no effect if there is no variance. If there is too little data, the gains achieved via a bagged ensemble cannot compensate for the decrease in accuracy of individual models, each of which now considers an even smaller training set. On the other end, if the dataset is extremely large and computation time is not an issue, even a single flexible classifier can be quite adequate.

Another method that uses different subsets of training data with a single learning method is the boosting approach (Freund and Schapire 1996). Boosting is similar in overall structure to bagging, except that it keeps track of the performance of the learning algorithm and concentrates on instances that have not been correctly learned. Instead of choosing the *t* training instances randomly using a uniform distribution, it chooses the training instances in such a manner as to favor the instances that have not been accurately learned. After several cycles, the prediction is performed by taking a weighted vote of the predictions of each classifier, with the weights being proportional to each classifier's accuracy on its training set.

AdaBoost is a practical version of the boosting approach (Freund and Schapire 1996). Adaboost requires less instability than bagging, because Adaboost can make much larger changes in the training set. A number of studies that compare AdaBoost and bagging suggest that AdaBoost and bagging have quite different operational profiles (Bauer and Kohavi 1999; Quinlan 1996). In general, it appears that bagging is more consistent, increasing the error of the base learner less frequently than does AdaBoost. However, AdaBoost appears to have greater average effect, leading to substantially larger error reductions than bagging on average.

Generally, bagging tends to decrease variance without unduly affecting bias (Breiman 1996; Schapire et al. 1998; Bauer and Kohavi 1999). On the contrary, in empirical studies AdaBoost appears to reduce both bias and variance (Breiman 1996; Schapire et al. 1998; Bauer and Kohavi 1999). Thus, AdaBoost is more effective at reducing bias than bagging, but bagging is more effective than AdaBoost at reducing variance.

The decision on limiting the number of sub-classifiers is important for practical applications. To be competitive, it is important that the algorithms run in reasonable time. Quinlan (1996) used only 10 replications, while Bauer and Kohavi (1999) used 25 replications, Breiman (1996) used 50 and Freund and Schapire (1996) used 100. For both bagging and boosting, much of the reduction in error appears to have occurred after ten to fifteen classifiers. However, Adaboost continues to measurably improve test-set error until around 25 classifiers for decision trees (Opitz and Maclin 1999).

As mentioned in Bauer and Kohavi (1999), the main problem with boosting seems to be robustness to noise. This is expected because noisy instances tend to be misclassified, and the weight will increase for these instances. They presented several cases where the performance of boosted algorithms degraded compared to the original algorithms. On the contrary, they pointed out that bagging improves the accuracy in *all* datasets used in the experimental evaluation.

MultiBoosting (Webb 2000) is another method of the same category. It can be conceptualized wagging committees formed by AdaBoost. Wagging is a variant of bagging: bagging uses resampling to get the datasets for training and producing a weak hypothesis, whereas wagging uses reweighting for each training instance, pursuing the effect of bagging in a different way. Webb (2000)in a number of experiments, showed that MultiBoost achieved greater mean error reductions than any of AdaBoost or bagging decision trees in both committee sizes that were investigated (10 and 100).

Another meta-learner, DECORATE (Diverse Ensemble Creation by Oppositional Relabeling of Artificial Training Examples), was presented by (Melville and Mooney 2003). This method uses a learner (one that provides high accuracy on the training data) to build a diverse committee. This is accomplished by adding different randomly-constructed examples to the training set when building new committee members. These artificially constructed examples are given category labels that disagree with the current decision of the committee, thereby directly increasing diversity when a new classifier is trained on the augmented data and added to the committee.

8.2 Different training parameters with a single training method

There are also methods for creating ensembles, which produce classifiers that disagree on their predictions. Generally, these methods focus on altering the training process in the hope that the resulting classifiers will produce different predictions. For example, neural network techniques that have been employed include methods for training with different topologies, different initial weights and different parameters (Maclin and Shavlik 1995).

Another effective approach for generation of a set of base classifiers is ensemble feature selection. Ensemble feature selection is finding a set of feature subsets for generation of the base classifiers for an ensemble with one learning algorithm. Ho (1998) has shown that simple random selection of feature subsets may be an effective technique for ensemble feature selection. This technique is called the random subspace method (RSM). In the RSM, one randomly selects $N* < N$ features from the N-dimensional dataset. By this, one obtains the $N*$-dimensional random subspace of the original N-dimensional feature space. This is repeated S times so as to get S feature subsets for constructing the base classifiers. Then, one constructs classifiers in the random subspaces and aggregates them in the final integration procedure. An experiment with a systematic partition of the feature space, using nine different combination schemes, was performed by (Kuncheva and Whitaker 2001), showing that there are no "best" combinations for all situations and that there is no assurance that in all cases that a classifier team will outperform the single best individual.

8.3 Different learning methods

Voting denotes the simplest method of combining predictions from multiple classifiers (Roli et al. 2001). In its simplest form, called plurality or majority voting, each classification model contributes a single vote (Hall et al. 2000). The collective prediction is decided by the majority of the votes, i.e., the class with the most votes is the final prediction. In weighted voting, on the other hand, the classifiers have varying degrees of influence on the collective prediction that is relative to their predictive accuracy. Each classifier is associated with a specific weight determined by its performance (e.g., accuracy, cost model) on a validation set. The final prediction is decided by summing up all weighted votes and by choosing the class with the highest aggregate. Kotsiantis and Pintelas (2004) combined the advantages of classifier fusion and dynamic selection. The algorithms that are initially used to build the ensemble are tested on a small subset of the training set and, if they have statistically worse accuracy than the most accurate algorithm, do not participate in the final voting.

Except for voting, stacking (Ting and Witten 1999) aims to improve efficiency and scalability by executing a number of learning processes and combining the collective results. The main difference between voting and stacking is that the latter combines base classifiers in a non-linear fashion. The combining task, called a meta-learner, integrates the independently computed base classifiers into a higher level classifier, a meta-classifier, by relearning the meta-level training set. This meta-level training set is created by using the base classifiers' predictions on the validation set as attribute values and the true class as the target. Ting and Witten (1999) have shown that successful stacked generalization requires the use of output class distributions rather than class predictions. In their experiments, only the MLR algorithm (a linear discriminant) was suitable for use as a level-1 classifier.

Cascade Generalization (Gama and Brazdil 2000) is another algorithm that belongs to the family of stacking algorithms. Cascade Generalization uses the set of classifiers sequentially, at each step performing an extension of the original data by the insertion of new attributes. The new attributes are derived from the probability class distribution given by a base classifier. This constructive step extends the representational language for the high level classifiers, reducing their bias.

Todorovski and Dzeroski (2003) introduced meta-decision trees (MDTs). Instead of giving a prediction, MDT leaves specify which classifier should be used to obtain a prediction. Each leaf of the MDT represents a part of the dataset, which is a relative area of expertise of the base-level classifier in that leaf. MDTs can use the diversity of the base-level classifiers better than voting, thus outperforming voting schemes in terms of accuracy, especially in domains with a high diversity of errors made by base-level classifiers.

Another attempt to improve classification accuracy is the use of hybrid techniques. Lazkano and Sierra (2003) presented a hybrid classifier that combines Bayesian Network algorithm with the Nearest Neighbor distance based algorithm. The Bayesian Network structure is obtained from the data and the Nearest Neighbor algorithm is used in combination with the Bayesian Network in the deduction phase. LiMin et al. (2004) presented Flexible NBTree: a decision tree learning algorithm in which nodes contain univariate splits as do regular decision trees, but the leaf nodes contain General Naive Bayes, which is a variant of the standard Naive Bayesian classifier. Zhou and Chen (2002) generated a binary hybrid decision tree according to the binary information gain ratio criterion. If attributes cannot further distinguish training examples falling into a leaf node whose diversity is beyond the diversity threshold, then the node is marked as a dummy node and a feed-forward neural network named FANNC is then trained in the instance space defined by the used attributes. Zheng and Webb (2000) proposed the application of lazy learning techniques to Bayesian induction and presented the resulting lazy Bayesian rule learning algorithm, called LBR. This algorithm can be justified by a variant of the Bayes model, which supports a weaker conditional attribute independence assumption than is required by naive Bayes. For each test example, it builds a most appropriate rule with a local naive Bayesian classifier as its consequent. Zhipeng et al. (2002) proposed a similar lazy learning algorithm: Selective Neighborhood based Naive Bayes (SNNB). SNNB computes different distance neighborhoods of the input new object, lazily learns multiple Naive Bayes classifiers, and uses the classifier with the highest estimated accuracy to make the final decision. Domeniconi and Gunopulos (2001) combined local learning with SVMs. In this approach an SVM is used to determine the weights of the local neighborhood instances.

9 Conclusions

This paper describes the best-know supervised techniques in relative detail. 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 (Brazdil et al. 2003; Kalousis and Gama 2004).

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 classification accuracy without decreasing comprehensibility is the wrapper feature selection procedure (Guyon and 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.

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. 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).

References

Acid S, de Campos LM (2003) Searching for Bayesian network structures in the space of restricted acyclic partially directed graphs. J Artif Intell Res 18:445–490

Aha D (1997) Lazy learning. Kluwer Academic Publishers, Dordrecht

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

An A, Cercone N (2000) Rule quality measures improve the accuracy of rule induction: an experimental approach. Lecture notes in computer science, vol. 1932, pp 119–129

Auer P, Warmuth M (1998) Tracking the best disjunction. Machine Learning 32:127–150

Baik S, Bala J (2004) A decision tree algorithm for distributed data mining: towards network intrusion detection. Lecture notes in computer science, vol. 3046, pp 206–212

Batista G, Monard MC (2003) An analysis of four missing data treatment methods for supervised learning. Appl Artif Intell 17:519–533

Basak J, Kothari R (2004) A classification paradigm for distributed vertically partitioned data. Neural Comput 16(7):1525–1544

Blake CL, Merz CJ (1998) UCI repository of machine learning databases. University of California, Irvine. (http://www.ics.uci.edu/~mlearn/MLRepository.html)

Blockeel H, De Raedt L (1998) Top-down induction of first order logical decision trees. Artif Intell 101(1–2):285–297

Blum A (1997) Empirical support for winnow and weighted-majority algorithms: results on a calendar scheduling domain. Mach Learn 26(1):5–23

Bonarini A (2000) An introduction to learning fuzzy classifier systems. Lect Notes Comput Sci 1813:83–92

Bouckaert R (2003) Choosing between two learning algorithms based on calibrated tests. In: Proceedings of 20th International Conference on Machine Learning. Morgan Kaufmann, pp 51–58

Bouckaert R (2004) Naive Bayes classifiers that perform well with continuous variables. Lect Notes Comput Sci 3339:1089–1094

Brazdil P, Soares C, Da Costa J (2003) Ranking learning algorithms: using IBL and meta-learning on accuracy and time results. Mach Learn 50:251–277

Breiman L (1996) Bagging predictors. Mach Learn 24:123–140

Breslow LA, Aha DW (1997) Simplifying decision trees: a survey. Knowl Eng Rev 12:1–40

Brighton H, Mellish C (2002) Advances in instance selection for instance-based learning algorithms. Data Min Knowl Disc 6:153–172

Burges C (1998) A tutorial on support vector machines for pattern recognition. Data Min Knowl Disc 2(2):1–47

Camargo LS, Yoneyama T (2001) Specification of training sets and the number of hidden neurons for multilayer perceptrons. Neural Comput 13:2673–2680

Castellano G, Fanelli A, Pelillo M (1997) An iterative pruning algorithm for feedforward neural networks. IEEE Trans Neural Netw 8:519–531

Cheng J, Greiner R (2001) Learning Bayesian belief network classifiers: algorithms and system. In: Stroulia E, Matwin S (eds) AI 2001, LNAI 2056, pp 141–151

Cheng J, Greiner R, Kelly J, Bell D, Liu W (2002) Learning Bayesian networks from data: an information-theory based approach. Artif Intell 137:43–90

Chickering DM (2002) Optimal structure identification with greedy search. J Mach Learn Res 3:507–554

Cohen W (1995) Fast effective rule induction. In: Proceedings of ICML-95, pp 115–123

Cowell RG (2001) Conditions under which conditional independence and scoring methods lead to identical selection of Bayesian network models. In: Proceedings of 17th International Conference on Uncertainty in Artificial Intelligence

Crammer K, Singer Y (2002) On the learnability and design of output codes for multiclass problems. Mach Learn 47:201–233

Cristianini N, Shawe-Taylor J (2000) An introduction to support vector machines and other Kernel-based learning methods. Cambridge University Press, Cambridge

Dantsin E, Eiter T, Gottlob G, Voronkov A (2001) Complexity and expressive power of logic programming. ACM Comput Surveys 33:374–425

De Raedt L (1996) Advances in inductive logic programming. IOS Press

De Mantaras RL, Armengol E (1998) Machine learning from examples: inductive and Lazy methods. Data Knowl Eng 25:99–123

Dietterich TG (1998) Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput 10(7):1895–1924

Dietterich TG (2000) An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. Mach Learn 40:139–157

Domeniconi C, Gunopulos D (2001) Adaptive nearest neighbor classification using support vector machines. Adv Neural Inf Process Syst 14:665–672

Domingos P, Pazzani M (1997) On the optimality of the simple Bayesian classifier under zero-one loss. Mach Learn 29:103–130

Dutton D, Conroy G (1996) A review of machine learning. Knowl Eng Rev 12:341–367

Dzeroski S, Lavrac N (2001) Relational data mining. Springer, Berlin

Elomaa T, Rousu J (1999) General and efficient multisplitting of numerical attributes. Mach Learn 36:201–244

Elomaa T (1999) The biases of decision tree pruning strategies. Lect Notes Comput Sci 1642:63–74, Springer

Fidelis MV, Lopes HS, Freitas AA (2000) Discovering comprehensible classification rules using a genetic algorithm. In: Proceedings of CEC-2000, conference on evolutionary computation La Jolla, USA, v. 1, pp 805–811

Flach PA, Lavrac N (2000) The role of feature construction in inductive rule learning. In: De Raedt L, Kramer S (eds) Proceedings of the ICML2000 workshop on attribute-value learning and relational learning: bridging the Gap. Stanford University

Frank E, Witten I (1998) Generating accurate rule sets without global optimization. In: Shavlik J (ed) Machine learning: proceedings of the fifteenth international conference. Morgan Kaufmann Publishers, San Francisco

Freund Y, Schapire R (1997) A decision-theoretic generalization of on-line learning and an application to boosting. JCSS 55(1):119–139

Freund Y, Schapire R (1999) Large margin classification using the perceptron algorithm. Mach Learn 37:277–296

Friedman N, Geiger D, Goldszmidt M (1997) Bayesian network classifiers. Mach Learn 29:131–163

Friedman N, Koller D (2003) Being Bayesian about network structure: a Bayesian approach to structure discovery in Bayesian networks. Mach Learn 50(1):95–125

Furnkranz J (1997) Pruning algorithms for rule learning. Mach Learn 27:139–171

Furnkranz J (1999) Separate-and-conquer rule learning. Artif Intell Rev 13:3–54

Furnkranz J (2001) Round robin rule learning. In: Proceedings of the 18th international conference on machine learning (ICML-01), pp 146–153

Gama J, Brazdil P (1999) Linear tree. Intelligent Data Anal 3:1–22

Gama J, Brazdil P (2000) Cascade generalization. Mach Learn 41:315–343

Gehrke J, Ramakrishnan R, Ganti V (2000) RainForest—a framework for fast decision tree construction of large datasets. Data Min Knowl Disc 4(2–3):127–162

Genton M (2001) Classes of Kernels for machine learning: a statistics perspective. J Mach Learn Res 2:299–312

Guo G, Wang H, Bell D, Bi Y, Greer K (2003) KNN model-based approach in classification. Lect Notes Comput Sci 2888:986–996

Guyon I, Elissee A (2003) An introduction to variable and feature selection. J Mach Learn Res 3:1157–1182

Heckerman D, Meek C, Cooper G (1999) A Bayesian approach to causal discovery. In: Glymour C, Cooper G (eds) Computation, causation, and discovery, MIT Press, pp 141–165

Ho TK (1998) The random subspace method for constructing decision forests. IEEE Trans Pattern Anal Mach Intell 20:832–844

Hodge V, Austin J (2004) A survey of outlier detection methodologies. Artif Intell Rev 22(2):85–126

Japkowicz N, Stephen S (2002) The class imbalance problem: a systematic study. Intell Data Anal 6(5):429–450

Jain AK, Murty MN, Flynn P (1999) Data clustering: a review. ACM Comput Surveys 31(3):264–323

Jensen F (1996) An introduction to Bayesian networks. Springer

Jordan MI (1998) Learning in graphical models. MIT Press,

Kalousis A, Gama G (2004) On data and algorithms: understanding inductive performance. Mach Learn 54:275–312

Keerthi S, Gilbert E (2002) Convergence of a generalized SMO algorithm for SVM classifier design. Mach Learn 46:351–360

Kivinen J (2002) Online learning of linear classifiers. In: Advanced Lectures on Machine Learning: Machine Learning Summer School 2002, Australia, February 11–22, pp 235–257, ISSN: 0302–9743

Klusch M, Lodi S, Moro G (2003) Agent-based distributed data mining: the KDEC scheme. In: Intelligent information agents: the agentlink perspective, LNAI 2586, Springer, pp 104–122

Kon M, Plaskota L (2000) Information complexity of neural networks. Neural Netw 13:365–375

Kotsiantis S, Pintelas P (2004) Selective voting. In: Proceedings of the 4th International Conference on Intelligent Systems Design and Applications (ISDA 2004), August 26–28, Budapest, pp 397–402

Kuncheva L, Whitaker C (2001) Feature subsets for classifier combination: an enumerative experiment. Lect Notes Comput Sci 2096:228–237

Lazkano E, Sierra B (2003) BAYES-NEAREST: a new hybrid classifier combining Bayesian network and distance based algorithms. Lect Notes Comput Sci 2902:171–183

LiMin W, SenMiao Y, Ling L, HaiJun L (2004) Improving the performance of decision tree: a hybrid approach. Lect Notes Comput Sci 3288:327–335

Lindgren T (2004) Methods for rule conflict resolution. Lect Notes Comput Sci 3201:262–273

Littlestone N, Warmuth M (1994) The weighted majority algorithm. Informa Comput 108(2):212–261

Liu H, Metoda H (2001) Instance selection and constructive data mining. Kluwer,

Maclin R, Shavlik J (1995) Combining the prediction of multiple classifiers: using competitive learning to initialize ANNs. In: Proceedings of the 14th International joint conference on AI, pp 524–530

Madden M (2003) The performance of Bayesian network classifiers constructed using different techniques. In: Proceedings of European conference on machine learning, workshop on probabilistic graphical models for classification, pp 59–70

Markovitch S, Rosenstein D (2002) Feature generation using general construction functions. Mach Learn 49:59–98

McSherry D (1999) Strategic induction of decision trees. Knowl Based Syst 12(5–6):269–275

Melville P, Mooney R (2003) Constructing diverse classifier ensembles using artificial training examples. In: Proceedings of the IJCAI-2003, Acapulco, Mexico, pp 505–510

Mitchell T (1997) Machine learning. McGraw Hill

Muggleton S (1995) Inverse entailment and Progol. New Generat Comput Special issue on Inductive Logic Programming 13(3–4):245–286

Muggleton S (1999) Inductive logic programming: issues, results and the challenge of learning language in logic. Artif Intell 114:283–296

Murthy SK (1998) Automatic construction of decision trees from data: a multi-disciplinary survey. Data Min Knowl Disc 2:345–389

Nadeau C, Bengio Y (2003) Inference for the generalization error. Mach Learn 52:239–281

Neocleous C, Schizas C (2002) Artificial neural network learning: a comparative review, LNAI 2308, Springer-Verlag, pp 300–313

Okamoto S, Yugami N (2003) Effects of domain characteristics on instance-based learning algorithms. Theoret Comput Sci 298:207–233

Parekh R, Yang J, Honavar V (2000) Constructive neural network learning algorithms for pattern classification. IEEE Trans Neural Netw 11(2):436–451

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

Quinlan JR (1993) C4.5: Programs for machine learning. Morgan Kaufmann,

Quinlan JR (1995) Induction of logic programs: FOIL and related systems. New Generat Comput 13:287–312

Ratanamahatana C, Gunopulos D (2003) Feature selection for the naive Bayesian classifier using decision trees. Appl Artif Intell 17(5–6):475–487

Reinartz T (2002) A unifying view on instance selection, data mining and knowledge discovery, vol. 6. Kluwer Academic Publishers, pp 191–210

Reeves CR, Rowe JE (2003) Genetic algorithms—principles and perspectives: a guide to GA theory. Kluwer Academic

Roli F, Giacinto G, Vernazza G (2001) Methods for designing multiple classifier systems. Lect Notes Comput Sci 2096:78–87

Robert J, Howlett LCJ (2001) Radial basis function networks 2: new advances in design

Roy A (2000) On connectionism, rule extraction, and brain-like learning. IEEE Trans Fuzzy Syst 8(2):222–227

Saad D (1998) Online learning in neural networks. Cambridge University Press,

Sanchez J, Barandela R, Ferri F (2002) On filtering the training prototypes in nearest neighbour classification. Lect Notes Comput Sci 2504:239–248

Scholkopf C, Burges JC, Smola AJ (1999) Advances in Kernel Methods. MIT Press

Setiono R, Loew WK (2000) FERNN: an algorithm for fast extraction of rules from neural networks. Appl Intell 12:15–25

Siddique MNH, Tokhi MO (2001) Training neural networks: backpropagation vs. genetic algorithms. IEEE Int Joint Conf Neural Netw 4:2673–2678

Srivastava A, Han E, Kumar V, Singh V (1999) Parallel formulations of decision tree classification algorithms. Data Min Knowl Disc 3:237–261

Ting K, Witten I (1999) Issues in stacked generalization. Artif Intell Res 10:271–289

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. Mach Learn 40:203–228

Todorovski L, Dzeroski S (2003) Combining classifiers with meta decision trees. Mach Learn 50:223–249

Utgoff P, Berkman N, Clouse J (1997) Decision tree induction based on efficient tree restructuring. Mach Learn 29(1):5–44

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)

Villada R, Drissi Y (2002) A perspective view and survey of meta-learning. Artif Intell Rev 18:77–95

Vivarelli F, Williams C (2001) Comparing Bayesian neural network algorithms for classifying segmented outdoor images. Neural Netw 14:427–437

Wall R, Cunningham P, Walsh P, Byrne S (2003) Explaining the output of ensembles in medical decision support on a case by case basis. Artif Intell Med 28(2):191–206

Wettschereck D, Aha DW, Mohri T (1997) A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. Artif Intell Rev 10:1–37

Wilson DR, Martinez T (2000) Reduction techniques for instance-based learning algorithms. Mach Learn 38:257–286

Witten I, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann,

Yam J, Chow W (2001) Feedforward networks training speed enhancement by optimal initialization of the synaptic coefficients. IEEE Trans Neural Netw 12:430–434

Yang Y, Webb G (2003) On why discretization works for Naive-Bayes classifiers. Lect Notes Comput Sci 2903:440–452

Yen GG, Lu H (2000) Hierarchical genetic algorithm based neural network design. In: IEEE symposium on combinations of evolutionary computation and neural networks, pp 168–175

Yu L, Liu H (2004) Efficient feature selection via analysis of relevance and redundancy. JMLR 5:1205–1224

Zhang G (2000) Neural networks for classification: a survey. IEEE Trans Syst Man Cy C 30(4):451–462

Zhang S, Zhang C, Yang Q (2002) Data preparation for data mining. Appl Artif Intell 17:375–381

Zheng Z (1998) Constructing conjunctions using systematic search on decision trees. Knowl Based Syst J 10:421–430

Zheng Z (2000) Constructing X-of-N attributes for decision tree learning. Mach Learn 40:35–75

Zheng Z, Webb G (2000) Lazy learning of Bayesian rules. Mach Learn 41(1):53–84

Xie Z, Hsu W, Liu Z, Lee ML (2002) SNNB: a selective neighborhood based naive Bayes for lazy learning. Lect Notes Comput Sci 2336:104–115

Zhou Z, Chen Z (2002) Hybrid decision tree. Knowl Based Syst 15(8):515–528

Zhou Z (2004) Rule extraction: using neural networks or for neural networks?. J Comput Sci Technol 19(2):249–253