Emerging Artificial Intelligence Applications in Computer Engineering
I. Maglogiannis et al. (Eds.)
IOS Press, 2007
© 2007 The authors and IOS Press. All rights reserved.

Part I: General Purpose Applications of AI

Manolis Wallace
Department of Computer Science
University of Indianapolis Athens
wallace@uindy.gr

This part of the book is devoted to the more "conventional" areas of computational intelligence, i.e. to machine learning and to knowledge representation.

Machine learning, the focus of Section 1, refers to the development of automated systems that are able to process large amounts of data in order to extract meaningful and potentially useful information (data mining) as well as to the exploitation of such information in practical problems (decision support). The combination of both components, i.e. information extraction and application, is called data classification; in this context, the aim is to develop tools that, having studied a large labeled base of available data, are able to automatically label not previously seen data. Chapter 1 starts by presenting the challenges that are associated with the task of classification and continues to provide an extensive review of the field, comprising information on logic based, perceptron based, statistical learning and support vector approaches and also including a comparison of the different techniques and a discussion on the combination of multiple techniques.

One problem that is typically associated with machine learning is humans' inability to monitor and verify the machine's operation. One of the parameters that make the task impossible for humans to tackle is the fact that the large amount and digital format of the data considered by the automated systems. Dimension reduction and data visualization are two closely related research fields that focus on alleviating this difficulty. Chapter 2 focuses on the utilization of neural networks in this direction. Starting with a thorough theoretical foundation, that includes presentation of both standalone and combined approaches, we continue to present applications of dimension reduction and data visualization in a series of real life problems.

Once useful information has been acquired, either through machine learning or even directly from a human expert, a decision support system can be developed that assists humans in making decisions, often in uncertain environments. One problem associated with the penetration and practical application of such systems is humans' justified reluctance to base important and some times critical decisions on the recommendation provided by a system the operation of which is not clear or fully understood. In order to overcome this, argumentation based recommender systems presented in Chapter 3 have been developed that attempt to present and justify the reasoning behind each recommendation they offer. In addition to a detailed presentation of related theory, the chapter also goes on to provide information on practical technologies and applications that are based on argumentation.

Section 2 focuses on knowledge representation in the context of artificial intelligence and its applications. Chapter 4 introduces the topic by discussing knowledge modeling, i.e. the specification of formal structures for the representation of knowledge. Based on profiling, the extensibility mechanism of UML, a UML extension is developed for knowledge modeling. As with most chapters in this book, the theoretical discussion is followed by the practical application of the developed approach in a real life problem.

The remaining of the section focuses on practical applications of ontological representations, which constitute the current trend in knowledge representation. Thus, chapter 5, using DAML+OIL, presents a web resource semantic annotation model. Based on it, a semantic navigation methodology is developed and applied.

Chapter 6 discusses the utilization of ontologies in the management of pervasive systems. Based on the Resource Description Framework, a methodology is proposed to facilitate interoperability between diverse devices and sensors, that enabling futher development of pervasive systems.

Chapter 7 takes us to a more conventional application of ontologies, that of annotation of resources and items for web applications. Specifically, we present a state-of-the-art e-commerce system that, combining ontologies with advanced visualization techniques, is able to allow customers to fully customize products according to their needs.

Chapter 8 introduces the emerging concept of the information market, where marketed goods are organizational knowledge and experience. In this context, we see how ontologies can be used in order to achieve the critical balance between i) making the existence and content overview of offered information items publicly known and searchable and ii) protecting the rights associated with these information items.

Emerging Artificial Intelligence Applications in Computer Engineering
I. Maglogiannis et al. (Eds.)
IOS Press, 2007
© 2007 The authors and IOS Press. All rights reserved.

Supervised Machine Learning: A Review of Classification Techniques

S.B. KOTSIANTIS
Department of Computer Science & Technology, University of Peloponnese, Greece
sotos@math.upatras.gr

Abstract. The goal of supervised learning is to build a concise model of the distribution of class labels in terms of predictor features. The resulting classifier is then used to assign class labels to the testing instances where the values of the predictor features are known, but the value of the class label is unknown. This paper describes various supervised machine learning classification techniques. Of course, a single chapter cannot be a complete review of all supervised machine learning classification algorithms (also known induction classification algorithms), yet we hope that the references cited will cover the major theoretical issues, guiding the researcher in interesting research directions and suggesting possible bias combinations that have yet to be explored.

Keywords. Classifiers, data mining, intelligent data analysis, learning algorithms

Introduction

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

Every instance in any dataset used by machine learning algorithms is represented using the same set of features. The features may be continuous, categorical or binary. If instances are given with known labels (the corresponding correct outputs) then the learning is called *supervised*, in contrast to *unsupervised learning,* where instances are unlabeled. By applying these unsupervised (clustering) algorithms, researchers hope to discover unknown, but useful, classes of items [1]. Another kind of machine learning is *reinforcement learning* [2]. The training information provided to the learning system by the environment (external trainer) is in the form of a scalar reinforcement signal that constitutes a measure of how well the system operates. The learner is not told which actions to take, but rather must discover which actions yield the best reward, by trying each action in turn.

Numerous ML applications involve tasks that can be set up as supervised. In the present paper, we have concentrated on the techniques necessary to do this. In particular, this work is concerned with classification problems in which the output of instances admits only discrete, unordered values. We have limited our references to recent refereed journals, published books and conferences. In addition, we have added some references regarding the original work that started the particular line of research under discussion. A brief review of what ML includes can be found in [3]. A historical survey of logic and instance based learning classifiers is also presented in [4]. The reader should be cautioned that a single chapter cannot be a comprehensive review of all classification learning algorithms. Instead, our goal has been to provide a representative sample of existing lines of research in each learning technique. In each of our listed areas, there are many other papers that more comprehensively detail relevant work.

Our next section covers wide-ranging issues of supervised machine learning such as data pre-processing and feature selection. Logic-based learning techniques are described in Section 2, whereas perceptron-based techniques are analyzed in Section 3. Statistical techniques for ML are covered in Section 4. Section 5 deals with the newest supervised ML technique—Support Vector Machines (SVMs). In Section 6, learning techniques are compared. Section 7 presents a recent attempt for improving classification accuracy—ensembles of classifiers. Section 8 presents some special classification problems such as learning from imbalanced datasets and learning from multimedia files. Finally, the last section concludes this work.

1. General Issues of Supervised Learning Algorithms

Inductive machine learning is the process of learning a set of rules from instances (examples in a training set), or more generally speaking, creating a classifier that can be used to generalize from new instances. The 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 [5].

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

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

Impossible values (noise) should be checked for by the data handling software, ideally at the point of input so that they can be re-entered. If correct values cannot be entered, the problem is converted into missing value category, by simply removing the data. Hodge & Austin [6] have recently introduced a survey of contemporary techniques for outlier (noise) detection.

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 [7].

Feature subset selection is the process of identifying and removing as many irrelevant and redundant features as possible [8]. This reduces the dimensionality of the data and enables data mining algorithms to operate faster and more effectively. The fact that many features depend on one another often unduly influences the accuracy of supervised ML classification models. This problem can be addressed by constructing new features from the basic feature set [9].

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

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

2. Logic Based Algorithms

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

2.1. Decision Trees

Murthy [10] provided an overview of work in decision trees and a sample of their usefulness to newcomers as well as practitioners in the field of machine learning. Thus, in this work, apart from a brief description of decision trees, we will refer to some more recent works than those in Murthy's article as well as few very important articles that were published earlier. Decision trees are trees that classify instances by sorting them based on feature values. Each node in a decision tree represents a feature in an instance to be classified, and each branch represents a value that the node can assume. Instances are classified starting at the root node and sorted based on their feature values. Figure 1 is an example of a decision tree.

Using the decision tree depicted in Fig. 1 as an example, the instance (at1 = a1, at2 = b2, at3 = a3, at4 = b4) would sort to the nodes: at1, at2, and finally at3, which would classify the instance as being positive (represented by the values "Yes"). The problem of constructing optimal binary decision trees is an NP-complete problem and

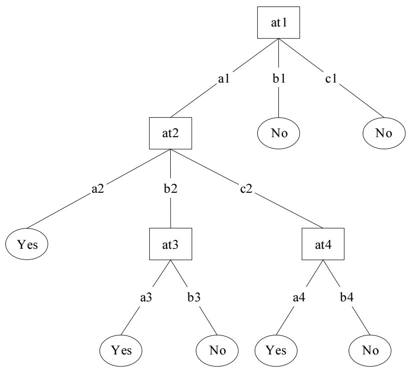

Figure1. A decision tree.

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 [10]. 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.

A decision tree, or any learned hypothesis h, is said to overfit training data if another hypothesis h′ exists that has a larger error than h when tested on the training data, but a smaller error than h when tested on the entire dataset. There are two common approaches that decision tree induction algorithms can use to avoid overfitting training data: i) Stop the training algorithm before it reaches a point at which it perfectly fits the training data, ii) Prune the induced decision tree. If the two trees employ the same kind of tests and have the same prediction accuracy, the one with fewer leaves is usually preferred. Most algorithms use a pruning method. A comparative study of well-known pruning methods presented in [11].

The most well-know algorithm in the literature for building decision trees is the C4.5 [12]. One of the latest studies that compare decision trees and other learning algorithms has been done by [13]. 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. [14] proposed Rainforest, a framework for developing fast and scalable algorithms to construct decision trees that gracefully adapt to the amount of main memory available.

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 [15], 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 [16] 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 [17] 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.

Baik and Bala [18] presented preliminary work on an agent-based approach for the distributed learning of decision trees. To sum up, one of the most useful characteristics of decision trees is their comprehensibility. People can easily understand why a decision tree classifies an instance as belonging to a specific class.

2.2. 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 [12]. However, rules can also be directly induced from training data using a variety of rule-based algorithms. Furnkranz [19] provided an excellent overview of existing work in rule-based methods. Classification rules represent each class by disjunctive normal form (DNF). A k-DNF expression is of the form: $(X_1 \wedge X_2 \wedge \ldots \wedge X_n) \vee (X_{n+1} \wedge X_{n+2} \wedge \ldots X_{2n}) \vee \ldots \vee (X_{(k\text{-}1)n+1} \wedge X_{(k\text{-}1)n+2} \wedge \ldots \wedge X_{kn})$, where k is the number of disjunctions, n is the number of conjunctions in each disjunction, and X_n is defined over the alphabet $X_1, X_2, \ldots, X_j \cup \sim X_1, \sim X_2, \ldots, \sim X_j$. The goal is to construct the smallest rule-set that is consistent with the training data. A large number of learned rules is usually a sign that the learning algorithm is attempting to "remember" the training set, instead of discovering the assumptions that govern it.

A separate-and-conquer algorithm 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. 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.

It is therefore important for a rule induction system to generate decision rules that have high predictability or reliability. These properties are commonly measured by a function called rule quality. A rule quality measure is needed in both the rule induction and classification processes. In rule induction, a rule quality measure can be used as a criterion in the rule specification and/or generalization process. In classification, a rule quality value can be associated with each rule to resolve conflicts when multiple rules are satisfied by the example to be classified. An and Cercone [20] 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 [21] has recently given a survey of methods used to solve this type of conflict.

To sum up, the most useful characteristic of rule-based classifiers is their comprehensibility. 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.

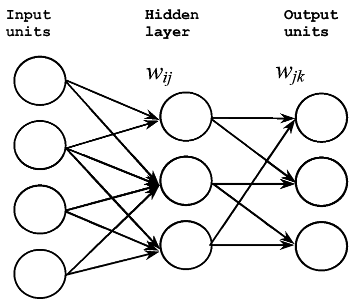

Figure 2. Feed-forward ANN.

3. 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.

3.1. Neural Networks

Perceptrons can only classify linearly separable sets of instances. If a straight line or plane can be drawn to separate 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 [22] provided an overview of existing work in Artificial Neural Networks (ANNs). Thus, in this study, apart from a brief description of the ANNs we will mainly refer to some more recent articles.

A multi-layer neural network consists of large number of units (neurons) joined together in a pattern of connections (Fig. 2). 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. 2) allow signals to travel one way only, from input to output. First, the network is trained on a set of paired data to determine input-output mapping. The weights of the connections between neurons are then fixed and the network is used to determine the classifications of a new set of data.

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 [23]. Kon & Plaskota [24] 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 [25]. However, the most well-known and widely used learning algorithm to estimate the values of the weights is the Back Propagation (BP) algorithm.

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 [26]. Genetic algorithms have been used to train the weights of neural networks [27] and to find the architecture of neural networks [28].

Even though multilayer neural networks and decision trees are two very different techniques for the purpose of classification, some researchers have performed some empirical comparative studies [29,13]. Some of the general conclusions drawn in that work are:

i) neural networks are usually more able to easily provide incremental learning than decision trees.
ii) training time for a neural network is usually much longer than training time for decision trees.
iii) neural networks usually perform as well as decision trees, but seldom better.

To sum up, ANNs have been applied to many real-world problems but still, their most striking disadvantage is their lack of ability to reason about their output in a way that can be effectively communicated. For this reason many researchers have tried to address the issue of improving the comprehensibility of neural networks, where the most attractive solution is to extract symbolic rules from trained neural networks [30].

4. 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 [31]. Thus, in this study, apart from our brief description of Bayesian networks, we mainly refer to more recent works.

4.1. Bayesian Networks

A Bayesian Network (BN) is a graphical model for probability relationships among a set of variables (features) (see Fig. 3). The Bayesian network structure *S* is a directed

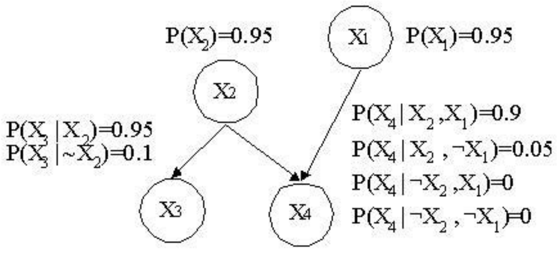

Figure 3. The structure of a Bayes Network.

acyclic graph (DAG) and the nodes in S are in one-to-one correspondence with the features X. The arcs represent casual influences among the features while the *lack* of possible arcs in S encodes conditional independencies. Moreover, a feature (node) is conditionally independent from its non-descendants given its parents (X_1 is conditionally independent from X_2 given X_3 if $P(X_1|X_2,X_3) = P(X_1|X_3)$ for all possible values of X_1, X_2, X_3).

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 [31]. 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.

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 [32,33]. 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 [34] 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 [35] has shown that for any structure search procedure based on CI tests, an equivalent procedure based on maximizing a score can be specified. 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 $p(c \mid X_1, X_2, \ldots X_n)$.

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. A problem of BN classifiers is that they are not suitable for datasets with many features [36]. The reason for this is that trying to construct a very large network is simply not feasible in terms of time and space. 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. The major advantage of the naive Bayes classifier is its short computational time for training. If a feature is numerical, the usual procedure for all Bayesian algorithms is to discretize it during data pre-processing [37], although a researcher can use the normal distribution to calculate probabilities [38].

4.2. Instance-Based Learning

Another category under the header of statistical methods is Instance-based learning. Instance-based learning algorithms are lazy-learning algorithms [39], 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 [40] and De Mantaras and Armengol [4] presented a review of instance-based learning classifiers. Thus, in this study, apart from a brief description of the nearest neighbour algorithm, we will refer to some more recent works.

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

In general, instances can be considered as points within an n-dimensional instance space where each of the n-dimensions corresponds to one of the n-features that are used to describe an instance. The absolute position of the instances within this space is not as significant as the relative distance between instances. This relative distance is determined by using a distance metric. Ideally, the distance metric must minimize the distance between two similarly classified instances, while maximizing the distance between instances of different classes. Many different metrics have been presented [41]. 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 [42].

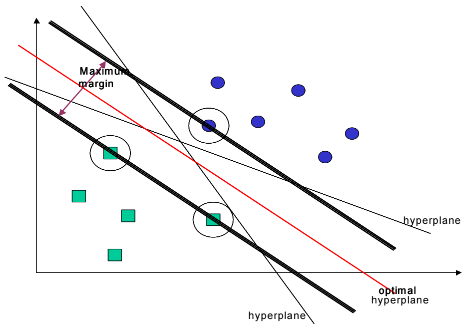

Figure 4. Maximum Margin.

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. Okamoto and Yugami [43] represented the expected classification accuracy of k-NN as a function of domain characteristics including the number of training instances, the number of relevant and irrelevant attributes, the probability of each attribute, the noise rate for each type of noise, and k. They also explored the behavioral implications of the analyses by presenting the effects of domain characteristics on the expected accuracy of k-NN and on the optimal value of k for artificial domains.

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 [8], because it could improve the classification accuracy and scale down the required classification time. Another issue is to determine which of the available instances should be used in modeling via instance selection [44].

5. Support Vector Machines

Support Vector Machines (SVMs) are the newest supervised machine learning technique. An excellent survey of SVMs can be found in [45], and a more recent book is by [46]. Thus, in this study apart from a brief description of SVMs we will refer to some more recent works and the landmark that were published before these works. SVMs revolve around the notion of a "margin"—either side of a hyperplane that separates two data classes. Maximizing the margin and thereby creating the largest possible distance between the separating hyperplane and the instances on either side of it has been proven to reduce an upper bound on the expected generalisation error.

In the case of linearly separable data, once the optimum separating hyperplane is found, data points that lie on its margin are known as support vector points and the solution is represented as a linear combination of only these points (see Fig. 4). 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 soft margin that accepts some misclassifications of the training instances [47].

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 $\Phi : R^d \rightarrow H$. Then the training algorithm would only depend on the data through dot products in H, i.e. on functions of the form $\Phi(x_i) \cdot \Phi(x_j)$. If there were a "kernel function" K such that $K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$, we would only need to use K in the training algorithm, and would never need to explicitly determine Φ. Thus, kernels are a special class of functions that allow inner products to be calculated directly in feature space, without performing the mapping described above [48]. 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 [49] 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 N^{th} dimensional QP problem, where N is the number of samples in the training dataset. Solving this problem in standard QP methods involves large matrix operations, as well as time-consuming numerical computations, and is mostly very slow and impractical for large problems. Sequential Minimal Optimization (SMO) is a simple algorithm that can, relatively quickly, solve the SVM QP problem without any extra matrix storage and without using numerical QP optimization steps at all [50]. SMO decomposes the overall QP problem into QP subproblems. Keerthi and Gilbert [51] 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 [52]. Discrete data presents another problem; although with suitable rescaling good results can be obtained.

6. Comparing Learning Techniques

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 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 Naive Bayes may need a relatively small dataset. 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.

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.

There is general agreement that k-NN 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.

Logic-based algorithms are all considered very easy to interpret, whereas neural networks and SVMs have notoriously poor interpretability. k-NN is also considered to have very poor interpretability because an unstructured collection of training instances is far from readable, especially if there are many of them.

While interpretability concerns the typical classifier generated by a learning algorithm, transparency refers to whether the principle of the method is easily understood. A particularly eloquent case is that of k-NN; while the resulting classifier is not quite interpretable, the method itself is 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.

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.

7. Combining Classifiers

Numerous methods have been suggested for the creation of ensemble of classifiers [53]. Although or perhaps because many methods of ensemble creation have been proposed, there is as yet no clear picture of which method is best [54]. 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.

7.1. 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 [55]. 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 [55] 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 [56]. 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 [56]. 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 [57,58]. 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.

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 [57]. 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. Generally, bagging tends to decrease variance without unduly affecting bias [55,57]. On the contrary, in empirical studies AdaBoost appears to reduce both bias and variance [55,57]. 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 [58] used only 10 replications, while Bauer & Kohavi [57] used 25 replications, Breiman [55] used 50 and Freund and Schapire [56] 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 [59].

As mentioned in Bauer and Kohavi [57], 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 [60] 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 [60], 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 [61]. 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.

7.2. 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 [62].

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 [63] 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 [64], 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.

7.3. Different Learning Methods

Voting denotes the simplest method of combining predictions from multiple classifiers [65]. In its simplest form, called plurality or majority voting, each classification model contributes a single vote. 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 [66] 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 [67] 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 [67] 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 [68] 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 & Dzeroski [69] 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 [70] 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. [71] 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 [72] 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 [73] 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. [74] 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 [75] combined local learning with SVMs. In this approach an SVM is used to determine the weights of the local neighborhood instances.

8. Special Learning Problems

In this section, we present two special classification problems: a) learning from imbalanced datasets and b) learning from multimedia files.

8.1. Handling Imbalanced Datasets

Learning classifiers from imbalanced or skewed datasets is an important topic, arising very often in practice in classification problems. In such problems, almost all the instances are labelled as one class, while far fewer instances are labelled as the other class, usually the more important class. It is obvious that traditional classifiers seeking an accurate performance over a full range of instances are not suitable to deal with imbalanced learning tasks, since they tend to classify all the data into the majority class, which is usually the less important class. The relationship between training set size and improper classification performance for imbalanced data sets seems to be that on small imbalanced data sets the minority class is poorly represented by an excessively reduced number of examples that might not be sufficient for learning, especially when a large degree of class overlapping exists and the class is further divided into sub-clusters.

The problem of imbalance has got more and more emphasis in recent years. Imbalanced data sets exists in many real-world domains, such as spotting unreliable telecommunication customers, detection of oil spills in satellite radar images, detection of fraudulent telephone calls, information retrieval and filtering tasks, and so on. A number of solutions to the class-imbalance problem are proposed both at the data and algorithmic levels. At the data level [76], these solutions include many different forms of re-sampling such as random over-sampling of minority class with replacement, random under-sampling of majority class, directed over-sampling (in which no new examples are created, but the choice of samples to replace is informed rather than random), directed under-sampling (where, again, the choice of examples to eliminate is informed), over-sampling with informed generation of new samples, and combinations of the above techniques. At the algorithmic level [77], solutions include adjusting the costs of the various classes so as to counter the class imbalance, adjusting the probabilistic estimate at the tree leaf (when working with decision trees), adjusting the decision threshold, and recognition-based (i.e., learning from one class) rather than discrimination-based (two class) learning. Mixture-of-experts approaches [78] (combining methods) have been also used to handle class-imbalance problems. These methods combine the results of many classifiers; each usually induced after over-sampling or under-sampling the data with different over/under-sampling rates. Gary Weiss [79] presents an extensive overview of the field of learning from imbalanced data.

8.2. Multimedia Mining

Generally, multimedia database systems store and manage a large collection of multimedia objects, such as image, video, audio and hypertext data. Thus, in multimedia documents, knowledge discovery deals with non-structured information. For this reason, we need tools for discovering relationships between objects or segments within multimedia document components, such as classifying images based on their content, extracting patterns in sound, categorizing speech and music, and recognizing and tracking objects in video streams. These multimedia files undergo various transformations and features extraction to generate the important features from the multimedia files. With the generated features, mining can be carried out using the well kwon machine learning techniques to discover significant patterns.

Text categorization is a conventional classification problem applied to the textual domain. It solves the problem of assigning text content to predefined categories. In the learning stage, the labeled training data are first pre-processed to remove unwanted details and to "normalize" the data [80]. For example, in text documents punctuation symbols and non-alphanumeric characters are usually discarded, because they do not help in classification. Moreover, all characters are usually converted to lower case to simplify matters. The next step is to compute the features that are useful to distinguish one class from another. For a text document, this usually means identifying the key-words that summarize the contents of the document.

Image categorization classifies images into semantic databases that are manually pre-categorized. In the same semantic databases, images may have large variations with dissimilar visual descriptions (e.g. images of persons, images of industries etc.). In addition, images from different semantic databases might share a common background (some flowers and sunset have similar colors). In [81] the authors distinguish three types of feature vectors for image description: 1) pixel level features, 2) region level features, and 3) tile level features.

Audio data play an important role in multimedia applications. Music information has two main branches: symbolic and audio information. Attack, duration, volume, velocity and instrument type of every single note are available information. Therefore, it is possible to easily access statistical measures such as tempo and mean key for each music item [82].

In video mining, there are three types of videos: a) the produced (e.g. movies, news videos, and dramas), b) the raw (e.g. traffic videos, surveillance videos etc), and c) the medical video (e.g. ultra sound videos including echocardiogram). The first stage for mining raw video data is grouping input frames to a set of basic units, which are relevant to the structure of the video. In produced videos, the most widely used basic unit is a shot, which is defined as a collection of frames recorded from a single camera operation. Shot detection methods can be classified into many categories: pixel based, statistics based, transform based, feature based and histogram based [83]. Color or grayscale histograms (such as in image mining) can also be used [84]. To segment video, color histograms, as well as motion and texture features can be used [85].

Compared with data mining, multimedia mining reaches much higher complexity resulting from: a) the huge volume of data, b) the variability and heterogeneity of the multimedia data (e.g. diversity of sensors, time or conditions of acquisition etc) and c) the multimedia content's meaning is subjective.

9. Conclusions

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

The key question when dealing with ML classification is not whether a learning algorithm is superior to others, but under which conditions a particular method can significantly outperform others on a given application problem. Meta-learning is moving in this direction, trying to find functions that map datasets to algorithm performance [86]. To this end, meta-learning uses a set of attributes, called meta-attributes, to represent the characteristics of learning tasks, and searches for the correlations between these attributes and the performance of learning algorithms. Some characteristics of learning tasks are: the number of instances, the proportion of categorical attributes, the proportion of missing values, the entropy of classes, etc. Brazdil et al. [87] provided an extensive list of information and statistical measures for a dataset.

After a better understanding of the strengths and limitations of each method, the possibility of integrating two or more algorithms together to solve a problem should be investigated. The objective is to utilize the 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. 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.

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 [88].

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 [89]. Theoretically, having more features should result in more discriminating power. However, practical experience with machine learning algorithms has shown that this is not always the case. Wrapper methods wrap the feature selection around the induction algorithm to be used, using cross-validation to predict the benefits of adding or removing a feature from the feature subset used.

Finally, many researchers in machine learning are accustomed to dealing with flat files and algorithms that run in minutes or seconds on a desktop platform. For these researchers, 100,000 instances with two dozen features is the beginning of the range of "very large" datasets. However, the database community deals with gigabyte databases. Of course, it is unlikely that all the data in a data warehouse would be mined simultaneously. Most of the current learning algorithms are computationally expensive and require all data to be resident in main memory, which is clearly untenable for many realistic problems and databases. An orthogonal approach is to partition the data, avoiding the need to run algorithms on very large datasets. Distributed machine learning involves breaking the dataset up into subsets, learning from these subsets concurrently and combining the results [90]. Distributed agent systems can be used for this parallel execution of machine learning processes [91]. Non-parallel machine learning algorithms can still be applied on local data (relative to the agent) because information about other data sources is not necessary for local operations. It is the responsibility of agents to integrate the information from numerous local sources in collaboration with other agents.

References

[1] S. Kotsiantis, P. Pintelas, Recent Advances in Clustering: A Brief Survey, WSEAS Transactions on Information Science and Applications, Vol. 1, No 1 (73–81), 2004.

[2] Gosavi, A., Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning, Series: Operations Research/Computer Science Interfaces Series, Vol. 25, 2003, Springer.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[31] Jensen, F. (1996). An Introduction to Bayesian Networks. Springer.
[32] Madden, M. (2003), The Performance of Bayesian Network Classifiers Constructed using Different Techniques, Proceedings of European Conference on Machine Learning, Workshop on Probabilistic Graphical Models for Classification, pp. 59–70.
[33] Chickering, D.M. (2002). Optimal Structure Identification with Greedy Search. Journal of Machine Learning Research, Vol. 3, pp 507–554.
[34] Acid, S. and de Campos. L.M. (2003). Searching for Bayesian Network Structures in the Space of Restricted Acyclic Partially Directed Graphs. Journal of Artificial Intelligence Research 18: 445–490.
[35] Cowell, R.G. (2001). Conditions Under Which Conditional Independence and Scoring Methods Lead to Identical Selection of Bayesian Network Models. Proc. 17th International Conference on Uncertainty in Artificial Intelligence.
[36] Cheng, J., Greiner, R., Kelly, J., Bell, D., & Liu, W. (2002). Learning Bayesian networks from data: An information-theory based approach. Artificial Intelligence 137: 43–90.
[37] Yang, Y., Webb, G. (2003), On Why Discretization Works for Naive-Bayes Classifiers, Lecture Notes in Computer Science, Volume 2903, Pages 440–452.
[38] Bouckaert, R. (2004), Naive Bayes Classifiers That Perform Well with Continuous Variables, Lecture Notes in Computer Science, Volume 3339, Pages 1089–1094.
[39] Mitchell, T. (1997). Machine Learning. McGraw Hill.
[40] Aha, D. (1997). Lazy Learning. Dordrecht: Kluwer Academic Publishers.
[41] Witten, I. & Frank, E. (2000). Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann, San Mateo, 2000.
[42] Wettschereck, D., Aha, D.W. & Mohri, T. (1997). A Review and Empirical Evaluation of Feature Weighting Methods for a Class of Lazy Learning Algorithms. Artificial Intelligence Review 10:1–37.
[43] Okamoto, S., Yugami, N. (2003), Effects of domain characteristics on instance-based learning algorithms. Theoretical Computer Science 298, 207–233.
[44] Sanchez, J., Barandela, R., Ferri, F. (2002), On Filtering the Training Prototypes in Nearest Neighbour Classification, Lecture Notes in Computer Science, Volume 2504, Pages 239–248.
[45] Burges, C. (1998). A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery. 2(2):1–47.
[46] Cristianini, N. & Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. Cambridge University Press, Cambridge.
[47] 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).
[48] Scholkopf, C., Burges, J.C. & Smola, A.J. (1999). Advances in Kernel Methods. MIT Press.
[49] Genton, M. (2001). Classes of Kernels for Machine Learning: A Statistics Perspective. Journal of Machine Learning Research 2: 299–312.
[50] Platt, J. (1999). Using sparseness and analytic QP to speed training of support vector machines. In Kearns, M., Solla, S. & Cohn, D. (ed.), Advances in neural information processing systems. MIT Press.
[51] Keerthi, S. & Gilbert, E. (2002). Convergence of a Generalized SMO Algorithm for SVM Classifier Design. Machine Learning 46: 351–360.
[52] Crammer, K. & Singer, Y. (2002). On the Learnability and Design of Output Codes for Multiclass Problems. Machine Learning 47: 201–233.
[53] Dietterich, T.G. (2000). An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization, Machine Learning 40: 139–157.
[54] Villada, R. & Drissi, Y. (2002). A Perspective View and Survey of Meta-Learning. Artificial Intelligence Review 18: 77–95.
[55] Breiman, L., Bagging Predictors. Machine Learning, 24 (1996) 123–140.
[56] Freund, Y. & Schapire, R. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. JCSS 55(1): 119–139.
[57] Bauer, E. & Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. Machine Learning, Vol. 36, pp. 105–139.
[58] Quinlan, J.R. (1996), Bagging, boosting, and C4.5. In Proceedings of the Thirteenth National Conference on Artificial Intelligence, pp. 725–730, AAAI/MIT Press.
[59] Opitz D. & Maclin R. (1999), Popular Ensemble Methods: An Empirical Study, Artificial Intelligence Research, 11: 169–198, Morgan Kaufmann.
[60] Webb G.I. (2000), MultiBoosting: A Technique for Combining Boosting and Wagging, Machine Learning, 40, 159–196, Kluwer Academic Publishers.
[61] Melville, P., Mooney, R. (2003), Constructing Diverse Classifier Ensembles using Artificial Training Examples, Proc. of the IJCAI-2003, pp. 505–510, Acapulco, Mexico.
[62] Maclin & Shavlik (1995), Combining the Prediction of multiple classifiers: Using competitive learning to initialize ANNs. In the Proc of the 14th International Joint Conference on AI, 524–530.

[63] Ho, T.K. (1998): The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence 20: 832–844.

[64] Kuncheva, L., Whitaker, C. (2001). Feature Subsets for Classifier Combination: An Enumerative Experiment. volume 2096 of Lecture Notes in Computer Science, pages 228–237. Springer-Verlag.

[65] Roli, F., Giacinto, G., Vernazza, G. (2001). Methods for Designing Multiple Classifier Systems. Volume 2096 of Lecture Notes in Computer Science, pages 78–87. Springer-Verlag.

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

[67] Ting, K. & Witten, I. (1999). Issues in Stacked Generalization. Artificial Intelligence Research 10: 271–289.

[68] Gama, J. & Brazdil, P. (2000). Cascade Generalization. Machine Learning 41: 315–343.

[69] Todorovski, L., Dzeroski, S. (2003). Combining Classifiers with Meta Decision Trees. Machine Learning 50: 223–249.

[70] Lazkano, E., Sierra, B. (2003), BAYES-NEAREST: A New Hybrid Classifier Combining Bayesian Network and Distance Based Algorithms, Lecture Notes in Computer Science, Volume 2902, Pages 171–183.

[71] LiMin Wang, SenMiao Yuan, Ling Li, HaiJun Li (2004), Improving the Performance of Decision Tree: A Hybrid Approach, Lecture Notes in Computer Science, Volume 3288, Pages 327–335.

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

[73] Zheng, Z., Webb, G. (2000), Lazy Learning of Bayesian Rules, Machine Learning, Volume 41, Issue 1, Pages 53–84.

[74] Zhipeng Xie, Wynne Hsu, Zongtian Liu, Mong Li Lee (2002), SNNB: A Selective Neighborhood Based Naive Bayes for Lazy Learning, Lecture Notes in Computer Science, Volume 2336, Pages 104–115.

[75] Domeniconi, C., Gunopulos, D. (2001), Adaptive Nearest Neighbor Classification using Support Vector Machines, Advances in Neural Information Processing Systems 14, 665–672.

[76] Chawla, N.V., Hall, L.O., Bowyer, K.W., and Kegelmeyer W.P. (2002), SMOTE: Synthetic Minority Oversampling TEchnique. Journal of Artificial Intelligence Research, 16:321–357.

[77] Provost, F., & Fawcett, T. (2001). Robust classification for imprecise environments. Machine Learning, 42, 203–231.

[78] Joshi, M.V., Kumar, V., and Agarwal, R.C., (2001). Evaluating boosting algorithms to classify rare cases: comparison and improvements. In First IEEE International Conference on Data Mining, pages 257–264, November.

[79] Weiss, G. (2004), Mining with rarity: A unifying framework.SIGKDD Explorations, 6(1):7–19, 2004.

[80] Sebastiani F. (2002), Machine learning in automated text categorization. ACM Computing Surveys, Vol. 34, pp. 1–47.

[81] Zhang Ji, Wynne Hsu, Mong Li Lee (2001). Image Mining: Issues, Frameworks and Techniques, in Proc. of the Second International Workshop on Multimedia Data Mining (MDM/KDD'2001), San Francisco, CA, USA, pp. 13–20.

[82] Liu Z., J. Huang, Y. Wang, and T. Chen (1998), Audio Feature Extraction and Analysis for Scene Segmentation and Classification, Journal of VLSI Signal Processing, Vol. 20, pp. 61–79.

[83] Boresczky J.S. and L.A. Rowe (1996), A comparison of video shot boundary detection techniques, Storage & Retrieval for Image and Video Databases IV, Proc. SPIE 2670, pp. 170–179.

[84] Ardizzone E. and M. Cascia (1997). Automatic video database indexing and retrieval. Multimedia Tools and Applications, Vol. 4, pp. 29–56.

[85] Yu H. and W. Wolf (1997). A visual search system for video and image databases. In Proc. IEEE Int'l Conf. On Multimedia Computing and Systems, Ottawa, Canada, pp. 517–524.

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

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

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

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

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

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