Increasing the Accuracy of Predictive Algorithms: A Review of Ensembles of Classifiers

Sotiris Kotsiantis
University of Patras, Greece & University of Peloponnese, Greece

Dimitris Kanellopoulos
University of Patras, Greece

Panayotis Pintelas
University of Patras, Greece & University of Peloponnese, Greece

INTRODUCTION

In classification learning, the learning scheme is presented with a set of classified examples from which it is expected tone can learn a way of classifying unseen examples (see Table 1).

Formally, the problem can be stated as follows: Given training data $\{(x_1, y_1)\ldots(x_n, y_n)\}$, produce a classifier h: X->Y that maps an object x ϵ X to its classification label y ϵ Y. A large number of classification techniques have been developed based on artificial intelligence (logic-based techniques, perception-based techniques) and statistics (Bayesian networks, instance-based techniques). No single learning algorithm can uniformly outperform other algorithms over all data sets.

The concept of combining classifiers is proposed as a new direction for the improvement of the performance of individual machine learning algorithms. Numerous methods have been suggested for the creation of ensembles 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.

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

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

BACKGROUND

Generally, support vector machines (SVMs; Scholkopf, Burges, & Smola, 1999) and artificial neural networks (ANNs; Mitchell, 1997) tend to perform much better when dealing with multidimensions and continuous features. In contrast, logic-based systems (e.g., decision trees [Murthy, 1998] and rule learners [Furnkranz, 1999]) tend to perform better when dealing with discrete or categorical features. For neural-network models and SVMs, a large sample size is required in order to achieve the maximum prediction accuracy whereas the naive Bayes model (Jensen, 1996) may need a relatively small data set. 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 multicolinearity 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 data set, 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 (Aha, 1997). 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 methods require 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 *k*-nearest-neighbor (k-NN) algorithm

(Aha, 1997) 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 nonlazy learners, the execution space is usually much smaller than the training space since the resulting classifier is usually a highly condensed summary of the data.

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 and 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 methods are very transparent as they are 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.

MAIN FOCUS OF THE ARTICLE

Mechanisms that are used to build ensembles of classifiers include (a) using different subsets of training data with a single learning method, (b) using different training parameters with a single training method (e.g., using different initial weights for each neural network in an ensemble), and (c) using different learning methods.

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 data set 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. Another method that uses different subsets of training data with a single learning method is the boosting approach (Freund & Schapire, 1997). 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 the training set. AdaBoost is a practical version of the boosting approach (Freund & Schapire). A number of studies that compare AdaBoost and bagging suggest that AdaBoost and bagging have quite different operational profiles (Bauer & Kohavi, 1999; Opitz & Maclin, 1999). 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 (Bauer & Kohavi). 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 (Breiman; Bauer & Kohavi). On the contrary, in empirical studies, AdaBoost appears to reduce both bias and variance (Breiman; Bauer & Kohavi). 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 subclassifiers 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 used 25 replications, Breiman used 50, and Freund and Schapire used 100. For both bagging and boosting, much of the reduction in error appears to have occurred after 10 to 15 classifiers. However, AdaBoost continues to measurably improve test-set error until around 25 classifiers for decision trees (Opitz & Maclin, 1999). As mentioned in Bauer and Kohavi, the main problem with boosting seems to be robustness to noise. On the contrary, they pointed out that bagging improves the accuracy in all data sets used in the experimental evaluation. MultiBoosting (Webb, 2000) is another method of the same category. It can be conceptualized as wagging committees formed by AdaBoost. Wagging is a variant of bagging: Bagging uses resampling to get the data sets 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, in a number of experiments, showed that MultiBoost achieved greater mean error reductions than AdaBoost or bagging decision trees in both committee sizes