Chapter 2

Notions and Framework for Supervised Learning

This chapter presents the notions of supervised automatic learning on which our thesis is based upon. We then lay out the framework for supervised learning and discusses some existed supervised learning techniques. Next, we discuss the criteria for evaluating a model and the important bias/variance trade-offs and model complexities in supervised learning. We discuss in particular the role of variance in classification error of a model and present existing solutions based on aggregation techniques to cater for high variances.

2.1 Supervised Learning

Supervised learning is an effective learning technique that uses machine learning to automatically discover correlations between readily-available features or attributes and the quantity of interest or target attributes (to be predicted). The goal of supervised learning is to predict the value of a target measure based on a number of input measures consisting of a number of sample observations. The target measure could be numerical, discrete or categorical. As mentioned above, this learning of the input features is performed on a set of sample observation, where each tuple from the sample consists of an input vector X_i and a corresponding class label Y_i. The input vector contains measurable features of the system under consideration. If the target is categorical, the learning becomes a classification problem. Otherwise,

if the target feature is numerical or continuous then regression based learning is performed.

Thus, in order to perform training on a classifier, we have to find a mapping from a number of input features to target feature and this is done while trying to minimize the overall error rate. This is called as misclassification rate in case of classification. The goal of the learning algorithm is to accurately predict unseen observations or samples. The main criterion used to assess learning algorithms is their prediction accuracy, i.e. the way the model they produce generalizes to unseen data. Another important criterion, especially in the context of KDD, is the interpretability of this model. Computational efficiency and scalability are also of great concern, especially when it comes to apply learning algorithms on very large dataset.

2.1.1 Framework for Supervised Learning

Let $X(.)$ be an attribute value on the real set $\Re$. For each example ω of a learning set Ω drawn independently from a population, $X(\omega)$ is the value taken by the attribute $X(.)$ at ω. For example, if $X(.)$ is the attribute corresponding to the height of a set of people in inches, then the meaning of $X(\omega) = 110$ is that ω is 110 inches in height.

In the supervised learning problem, usually have a specific attribute $C(.)$, which is called the endogenous variable or class and it associates a belonging class to each element. This is precisely the attribute we want to predict. Unlike the explanatory attributes $X(.)$ which can be symbolic or numeric, the class $C(.)$ is usually symbolic and it takes its values in the finite set $c_1, ..., c_m$ called the class space. If an example belongs to a class c, we have $C(\omega) = c$ and reciprocally. We also suppose that $C(\omega)$ is known for all ω of the learning sample set Ω. Thus, we try to build a model, denoted by Φ, such that ideally we have:

$$C(.) = \Phi(X_1(.), ..., X_p(.))).$$

2.2 Some Supervised Learning Techniques

In the following subsections we introduce some popular supervised learning techniques that we often use for comparison and other purposes in our thesis.

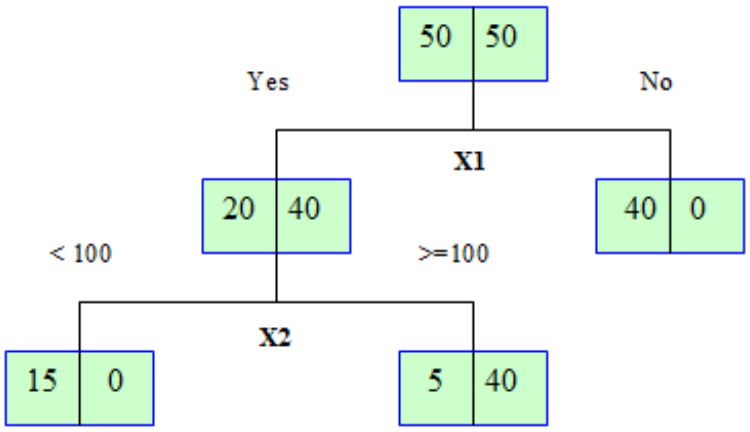

Figure 2.1: A decision tree generated from a two class problem.

2.2.1 Decision Tree Induction

Decision tree learners are widely used in solving classification problems with the classifiers represented as trees. They recursively divide attributes at each internal node in the tree based on some criteria. In a tree, leaf nodes represent classification decisions. Pruning methods are used to prevent over-fitting of training data. Although decision tree learners are not always the most competitive learners in terms of accuracy, they are computationally efficient and interpretability. These classifiers suffer in accuracy not because of bias but high variance. Thus, because of this high adaptivity to the learning sample, a small perturbation of the learning sample may result in very different decision trees.

There are several existing approaches to improve accuracy of decision trees by reducing their variance such as pruning which removes some of its test nodes so as to find the best compromise between bias and variance. However, a more efficient way to improve the accuracy of decision trees is to aggregate the predictions given by several trees. Various techniques have been proposed in the literature to generate different decision trees from the same learning sample for aggregation and they often give very impressive improvement of accuracy [36, 97]. Nevertheless, they also destroy some of the main advantages of decision

trees, namely computational efficiency and interpretability.

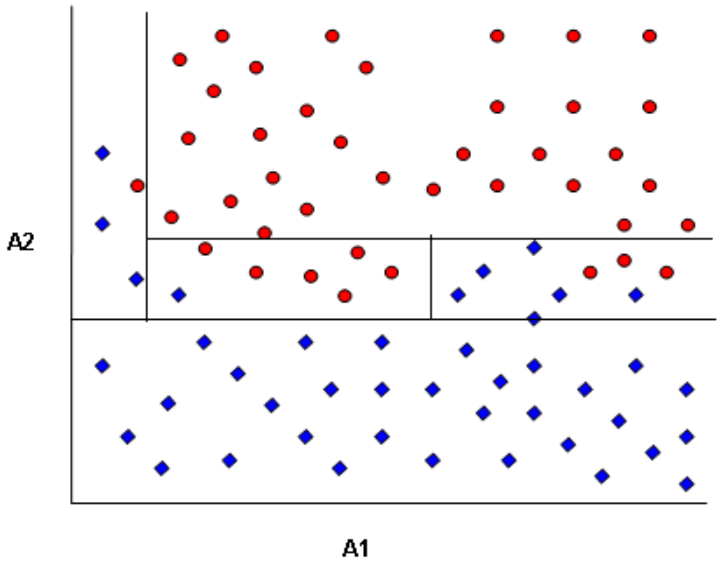

Figure 2.2: An example classification boundary generated by this decision tree.

As illustrated in Figure 2.1, the learning sample Ω_{LS} is iteratively split at each step by means of one of the predictive attributes $X_1, ..., X_p$. The goal is to separate the target feature as much as possible by using some criteria such as entropy. The leaves of the trees obtained at each step of the growing process define a partition that becomes finer with each recursion. The root of the tree corresponds to the partition $Node_1 = \Omega_{LS}$. The tree given in Figure 2.1 partitions Ω_{LS} in two subsets corresponding to the nodes $Node_2$ and $Node_3$. In leaf $Node_4$ for example, we have the set of cases that take values $X_1 = yes$ and $X_2 < 100$. Thus, we can say that the rule for firing the leaf $Node_4$ is that 'if attribute $X_1 = yes$ and if $X_2 < 100$'. Thus at any step, the partition at a node $Node_{curr}$ is derived from the previous one by seeking the best leaf-attribute tuple such that the splitting of the previous node $Node_{prev}$ according to the values of X_j maximizes the gain of information on the target variable between the current node $Node_{curr}$ and the previous node $Node_{prev}$. The gain of

information is usually measured as the reduction in uncertainty for the target variable or as the increase in the strength of association between the partition and the target variable. The growing process stops when the criterion can no longer be improved or when some stopping criterion is reached.

In Figure 2.2, another 2 class based example shows a visualization of how the search space is recursively condensed and the target features (circle and triangle) are separated at each step or node. Thus, at each node a perpendicular cut segments the classes into two regions and continue until a significant improvement can be made in the separation of the class values.

2.2.2 Naive-Bayes Classifier

Naive-Bayes classifiers are simple, effective, efficient, robust, and support incremental training [140]. Thus, because of these multiple characteristics, they have been used in numerous classification tasks. These classifiers have an attribute independence assumption, thus, naive-Bayes classifiers need only to estimate probabilities about individual attributes and the class instead of attribute combinations. In the case of qualitative attributes, their probabilities can be estimated from the corresponding frequencies. However, for quantitative attributes, the these prior probabilities are estimated by using a probability density estimation or discretization. Probability density estimation requires an assumption about the form of the probability distribution from which the quantitative attribute values are drawn. Discretization transforms a continuous attribute into discrete or qualitative attributes where each discretization point corresponds to an interval of values of the continuous or quantitative attribute.

In naive-Bayes learning, a set of instances with their classes, the training data, is provided and a test instance is presented. The learner predicts its class according to the prior probability model provided by the training data. Thus, naive-Bayes classifiers estimate the probability of a class c given an instance x by:

$$p(C=c|X=x) \propto p(C=c)\prod_{i=1}^{k} H(X_i=x_i|C=c), \text{ where}$$
$$H(X_i=x_i|C=c) = \begin{Bmatrix} p(X_i=x_i|C=c), & if\ X_i\ is\ qualitative \\ f(X_i=x_i|C=c), & if\ X_i\ is\ quantitative \end{Bmatrix}$$

2.2.3 K-Nearest Neighbors

The k-nearest neighbors algorithm (k-NN) [12, 43] is a method for classifying instances by comparing them to the closest training instances in the feature space. Considering a learning sample Ω_{LS} and a fixed value of k, the KNN method considers its k nearest neighbors in the learning set and predicts at each point an aggregation of those outputs. This is achieved according to some distance measures such as the euclidean, hamming distance etc. However, a problem with the voting method is that the classes with more frequent examples tend to dominate the prediction. In order to cater for this problem, weights are assigned to the classification taking into account the distances to eack of the k nearest neighbors.

When we are dealing with continuous attributes, euclidean distance is mostly used with a weight W such that:

$$d = \sum_{i=1}^{n} W \times (x_i - \bar{x}_i)^2$$

The optimal value of k is an important problem, which we elaborate in chapter 10. There are various techniques that cater to this problem e.g. cross validation but the optimal value of k will depend on the learning sample size N and on the dimensionality of the input features.

2.3 Criteria for Evaluation of the Learning Techniques

Three main criteria are used to compare learning algorithms:

1. Accuracy: Rate of correct predictions and reduction in the error.
2. Computational efficiency and Complexity: The time needed to learn a model but also the time to apply this model to new cases.
3. Interpretability: if it gives comprehensible models or not.
4. Robustness: The behavior of learning algorithm on unseen data over training data.

In this thesis, new improvements or algorithms will be discussed having in mind these criteria plus additional features discussed later. However, accuracy and complexity have

a trade-off between them. As with our techniques presented later an increase in accuracy could easily increase the complexity and vice versa. Thus, depending on the nature of the problem we can decide which of the two factors are more desirable to the cause than the other.

2.4 Bias and Variance in Supervised Learning

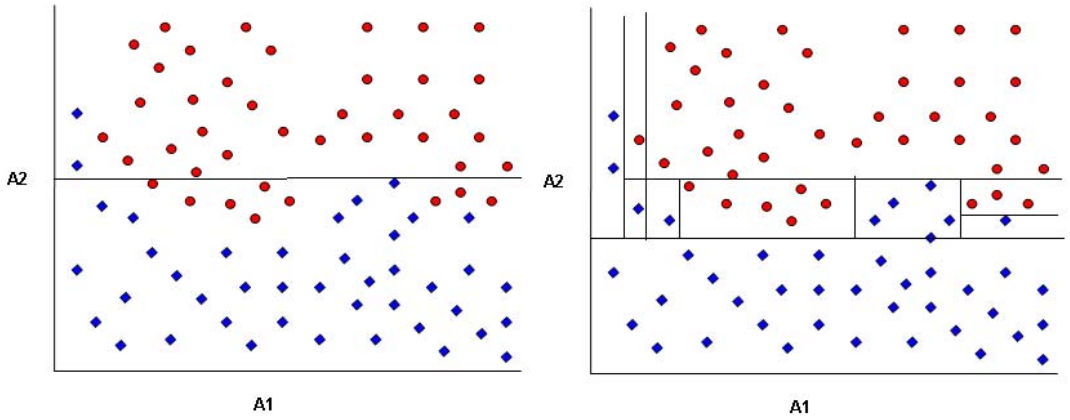

Figure 2.3: On the left a too simple model with high bias and on the right a too complex model with high variance.

In order to explain the concept and complexity of the bias/variance trade-off in supervised learning we take the help of figure 2.3 that shows an example of a two class problem and our aim is to separate or classify these two classes. In this example we use classic decision tree classifier to separate or classify the target features. The left part of figure 2.3 shows one such model. With a simple linear model consisting of one split, it is impossible to separate perfectly the two classes. Thus, this model is not satisfactory to solve this task as the separability of classes is not satisfactory and is far from the optimal model of figure 2.2. This phenomenon responsible for the error in this case is called the bias.

Now, the right part of figure 2.3 shows a more complex decision tree model. This time,

the classification boundary perfectly separates elements of the different classes. Like the linear model however, this model is still not appropriate. This time, actually, the model is too complex and hence it overfits the data. If we draw another random learning sample for the same problem, then it is very likely that the model found by the same learning algorithm will be very different from this one. In this case, we say that the learning algorithm suffers from variance.

Thus, both bias and variance are sources of error and hence they should be minimized. Thus, there is an obvious trade-off between these two effects which must be taken into account. To achieve this objective prediction aggregation methods or ensemble learning techniques are used to reduce the variance. This approach consists of building several models for the same sample and classify a new example by taking into account the output class which receives the majority of votes among all models. Among the most popular ensemble methods, there are bagging [76] and boosting [136]. Because of the very impressive results obtained by these methods, a lot of research has been carried out recently in this domain [36, 97].

2.4.1 Bias and Variance Decompositions

The bias variance decomposition is a method to measure the response and sensitivity of a learning algorithm to the training data. For example learning methods that build prediction models describing perfectly the training or sample data, i.e. a very small training error, have a high variance and they are quite sensitive to the changes of the training set. Meaning that maybe for another random sample the model could be very different from the first one. This is already illustrated in figures 2.2 and 2.3. In figure 2.2 a decision tree classifier builds a model trying to control the bias/variance trade-off and finding a near optimal solution. However, in figure 2.3 two models are shown either displaying high bias and low variance (left) or low bias and high variance (right).

Consider a bias/variance decomposition for classification. Let (X, t) where $X = X_1, X_2, ..., X_n$, be an example where t is the target value and X is the vector of the predictive attributes. Considering a series of models constructed from B random samples $\Omega LS_1, \Omega LS_1, ..., \Omega LS_B$, and then are applied to the test set Ω_{TS}. The error of a learning algorithm for a specific ex-

ample (X, t), is decomposed in two dimensions namely the bias, and the variance, according to the following formula :

$$Error(X) = Bias(X) + kVar(X).$$

Where, $X \in \sum_{i=1}^{B} X(\Omega_{LS_i})$ and k is a parameter that depends on the specific example (explained below).

2.4.1.1 Bias

The bias measures how far are the predictions that a learning algorithm does for an example (X, t) from the the optimal prediction, c^*. As optimal prediction we consider the prediction that the optimal classification algorithm does which in the case of $0/1$ loss function is the Bayes classifier. For classification and $0/1$ loss function, L, the bias is given by : $Bias(x) = L(c^*, c_\mu)$. Here, c_μ is called the most probable prediction or the central tendency for the instance (X, t). The bias can only take two values, 0 or 1 i.e unbiased or biased. To compute c_μ for an instance (X, t) of the test set, we need to get all the predictions for that instance from different models, and then find the prediction that appears most often, this will be the c_μ.

2.4.1.2 Variance

In the case of classification variance measures how the predictions of a learning algorithm for a specific instance, derived from the B different training sets, fluctuate around the most often prediction c_μ associated with that example. The variance for an instance (X, t) is given by the following formula:

$$Var(X) = \forall X \in \sum_{i=1}^{B} X(\Omega_{LS_i}) c \neq c_\mu$$
$$\text{or } Var(X) = P_B(c \neq c\mu)$$

Thus, we calculate the variance as the percentage of times, that the prediction of the learning algorithm is different from the prediction c_μ.

In the same way as above the error $Error(X)$ of a learning algorithm for an instance over a series of B models constructed from different training datasets can be written as :

$$Error(X) = \forall X \in \sum_{i=1}^{B} X(\Omega_{LS_i}) t \neq c$$
$$\text{or } Error(X) = P_B(t \neq c)$$

Which is the percentage of times that the prediction c from the B different models differs from the target value t.

Now, we take the initial decomposition and replace error and variance with the associated probabilities we have:

$$Error(x) = Bias(x) + kVar(x)$$
$$P_B(t \neq c) = Bias(x) + kP_B(c \neq c_\mu)$$

The parameter k depends on whether the examined instance x is unbiased or not.

- If x is unbiased, i.e. B(x) = 0 then we set k = 1 and the error decomposition becomes:

  $$Error(x) = Bias(x) + kVar(x)$$
  $$P_B(t \neq c) = P_B(c \neq c_\mu)$$

  What actually is stated here is that the error in predicting the target value of x comes only from the variance of the learning algorithm due to the use of different learning sets.

- If x is biased, i.e. B(x) = 1 then we set $k = -P_B(c = t|c \neq c_\mu)$. This value of c corresponds to the probability of getting a correct prediction c for x given that this prediction is different from the most often prediction c_μ, which since the example is biased is false. In this case the error decomposition becomes :

  $$Error(x) = Bias(x) + kVar(x)$$
  $$P_B(t \neq c) = 1 - P_B(c = t|c \neq c_\mu)P_B(c \neq c_\mu)$$
  $$P_B(t \neq c) = 1 - P_B(c = t, c \neq c_\mu)$$

  In this case the error is 1 minus the percentage of times that the prediction is correct and different from the most often prediction c_μ, (these are the cases that we get a correct prediction for x).

2.5 Variance Reduction Techniques

As suggested by various previous works, the main problem of these learning methods such as decision trees is not bias because we can always build complex algorithms and models to cater to this problem. But as far as variance is concerned the more complex the model becomes the higher is the probability that variance will increase with it because we use random learning samples to build these complex models. The increased interest in variance reduction has been there since a decade and various variance reduction techniques have been presented [97, 21, 36]. The reason for that is the fact that automatic learning techniques have been applied in many complex domains due to a drastic increase in computer power. But, these improvements in representational power of the learning algorithms will be useless if they are not combined with some variance reduction techniques. In [97], variance reduction techniques have been classified into two main families:

- Techniques that find the best bias/variance trade-off for one particular learning algorithm.
- Techniques that change the bias/variance configuration of a learning algorithm.

2.5.1 Aggregation and Resampling Techniques

The techniques described in this section, try to maintain the average bias of the learning method but in doing so try to decrease the variance to a minimum. Thus, they generate by using different original methods a set models or predictions and then aggregate these models or predictions using a voting or an averaging scheme. By doing so they aim to produce a more accurate and stable prediction. These aggregation techniques differ in the way they aggregate the predications from the base algorithm. Without being exhaustive we present two popular aggregation techniques.

2.5.1.1 Bagging

Bootstrap aggregating (bagging) [76, 36] is an ensemble learning method that uses perturb and combine scheme to improve the robustness and classification accuracy. It also reduces

variance and helps to avoid overfitting. Given a learning sample Ω_{LS} of size n, bagging generates b new training sets n', by resampling examples from Ω_{LS} by bootstrap. A description of bootstrapping techniques is given in the next section. A model m_{Ω_i} is build from each of these resampled training sets $\Omega_1, ..., \Omega_b$ whose output is the class predicted most often by the classifiers or we can also say that we take the majority vote classifier.

$$m^*(X(\omega)) = arg\ max \sum_{i=1}^{b} I(m_{\Omega_i}(X(\omega), c))$$

or the average model of all the predictors (used typically in regression).

$$m^*(X(\omega)) = \frac{1}{b} \sum_{i=1}^{b} m_{\Omega_i}(X(\omega), c)$$

The idea of bagging is to find based on an estimation of the average model which has a zero variance and the same bias as the original algorithm.

2.5.1.2 Boosting

The boosting algorithm by Schapire [102] was designed as a method for boosting the performance of a weak learning algorithm. Like bagging, the AdaBoost.M1 algorithm [136], the most popular boosting algorithm, sequentially generates weighted learning samples and models are built. In each weighted learning sample, each object has a weight allocated based on the previously built model in the sequence. The incorrect classified objects are weighted by a factor inversely proportional to the error on the learning set. A final model is formed by using a weighted majority vote, where the role of weights is to give higher influence to the more accurate models in the sequence.

While the perturbations on a given learning set introduced by bagging are random and independent, the perturbations introduced by boosting are chosen deterministically and serially, and are dependent on all of the previously generated models. Empirical results show that among the averaging methods, boosting (as well as error-correcting output coding) can equally reduce model bias along with the correction in variance. The main problem with boosting seems to be the robustness to noise [21].

2.6 Resampling by Bootstrap

Bootstrapping technique by Efron (1979) [9] is a statistical inference based approach that builds a sampling distribution for a statistic by sampling it with replacement from the data at hand. Suppose that we draw a sample $\Omega_{BS} = x_1, x_2, ..., x_b$ from a population $P = X_1, X_2, ..., X_N$; and that Ω_{BS} is a simple random sample.

Now suppose that we are interested in finding some statistic $T = t(\Omega_{BS})$ as an estimate of the corresponding population parameter $T^* = t(P)$. Normally, any kind of statistical inference makes assumptions about the structure or distribution of the population but in certain instances, the exact distribution of T may be intractable, and so we instead derive its asymptotic distribution. However, if the assumptions about the population are wrong, then the corresponding sampling distribution of the statistic may be seriously inaccurate. On the other hand, if asymptotic results are relied upon, these may not hold to the required level of accuracy in a relatively small sample.

In contrast, the nonparametric bootstrap allows us to estimate the sampling distribution of a statistic empirically without making assumptions about the form of the population. In addition it keeps from deriving the asymptotic distribution. The central idea behind the nonparametric bootstrap is as follows:

We proceed to draw a sample of size n from among the elements of Ω_{BS}, sampling with replacement. It is necessary to sample with replacement, because we would otherwise simply reproduce the original sample Ω_{BS}. In effect, we are treating the sample Ω_{BS} as an estimate of the population P; that is, each element of Ω_{BS} is selected for the bootstrap sample with probability 1/n, mimicking the original selection of the sample Ω_{BS} from the population P. We repeat this procedure a large number of times, B, selecting many bootstrap samples.

Next, we compute the statistic T for each of the bootstrap samples; that is $T_{BS} = t(\Omega_{BS})$. Then the distribution of T_{BS} around the original estimate T is analogous to the sampling distribution of the estimator T around the population parameter T^*. For example, the average of the bootstrapped statistics,

$$E(\bar{T^*}) = \sum_{i=1}^{B} \frac{T_i}{B}$$

estimates the expectation of the bootstrapped statistics; then $Bias = \bar{T} - T$ is an

estimate of the bias of T, that is, $T - T^*$. Similarly, the estimated bootstrap variance of T is:

$$Var(T^*) = \sum_{i=1}^{B} \frac{(T_i - \bar{T}^*)^2}{B-1}$$

There are two sources of error in bootstrap inference: One of which is the error induced by using a particular sample to represent the population. The other is the sampling error produced by failing to enumerate all bootstrap samples. The latter source of error can be controlled by making the number of bootstrap replications B sufficiently large.

2.7 Existing Studies

Among the existing studies of bias/variance trade-offs and variance reduction techniques in supervised learning, the work of Pierre Guerts and Louis Wehenkel [97] stands out. They provide a detailed analysis of these phenomena and provide solutions to control these trade-offs. They identify different types of variance e.g. discretization variance in classifiers like decision trees and present various aggregation techniques based on bagging and other perturb and combine methods to reduce these variances from classifiers.

In their thesis, Yang et al [140] analyze the bias/variance decompositions in discretization for naive-bayes classifiers and the work of C.Olaru and L.Wehenkel [21] discuss the problem of variance in decision tree induction and concentrate on fuzzy solutions to handle this type of classifier variance.

Other studies like done by Eric Bauer and Ron Kohavi [36] study the bias/variance decompositions in classifiers such as decision trees and variance reduction techniques like bagging, boosting plus variants and their effect on this bias/variance decompositions.

2.8 Conclusions of this chapter

Automatic learning is necessary when it is impossible to analytically model a system. It aims at building a model of a system whose task is to predict unseen situations by utilizing already seen data samples. The quality of a model is calculating by its accuracy to predict, its comprehensibility and complexity in terms of model building. In automatic learning

bias and variance both contribute to the prediction error, whatever the learning problem, algorithm and sample size. The error decomposition allows us to better understand the way an automatic learning algorithm will respond to changing conditions. It allows us to compare different methods in terms of their weaknesses. This understanding allows us to select methods in practice, to study their performances, and to guide us in order to find appropriate ways to improve automatic learning methods.

The resulting errors in classifiers such as decision tree learners are mostly due to very high variance, which significantly decreases the attractiveness of this method in spite of their other intrinsic qualities. In order to improve these methods it is necessary to find a way to reduce variance, ideally without jeopardizing other important features. Aggregation methods such as bagging, boosting and resampling (by bootstrap) variants aim at reducing variance and thus, improving prediction rates and other qualities of these classifiers.