UNIVERSITY OF MINNESOTA

This is to certify that I have examined this copy of master's thesis by

Karthik Ramakrishnan

and have have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Dr. Rich Maclin

Name of Faculty Adviser

Signature of Faculty Advisor

Date

GRADUATE SCHOOL

Abstract

Inductive learning is a form of supervised learning in which a system tries to build a model of a concept (e.g. what makes a container a cup) from descriptions of things that are/are not examples of that concept. Numerous methods have been defined to induce concepts such as decision trees learning, artificial neural networks, etc. One especially powerful method is to use an ensemble of classifiers (i.e., a collection of classifiers each trained on a subset of the original training set). Bagging and Boosting are the two of the most popular methods for building ensembles. Boosting is often chosen as an ensemble building technique because it can reduce both the bias and variance of the error. But building a Boosting ensemble is a time consuming process as the process cannot be executed in parallel and it tends to overfit the training examples as the number of classifiers in the ensemble increases. In this work, I propose a new ensemble building algorithm that works similar to Boosting. This new method builds a piece-wise set of classifiers. Each classifier focusses on learning from a subset of the problem space and is trained to perform well within that region of the problem space. By combining all the classifiers, we are covering the entire problem space and thereby building a global classifier. Experiments with the new algorithm comparing it to Bagging and Boosting show a reduction in test set error rates in some data sets and comparable performance on the remaining data sets. The new method is also faster as compared to Boosting as it can be parallelized.

Acknowledgements

I would like to thank my advisor Dr. Rich Maclin for all his help and guidance. I would also like to express my gratitude to my committee members Dr. Carolyn Crouch and Dr. Robert McFarland for their patience and co-operation.

Contents

List of Figures

List of Tables

1 Introduction

In the current age of Internet and data warehouses, the fundamental paradigms of classical data analysis are ripe for change. We are today awash with data, primarily collected by government and business applications [Weiss and Indurkhya, 1997]. Automation produces an ever-increasing flood of data. Today the need to develop an effective tool to gather knowledge from this data is more critical than ever before. Organizations today have huge data "lakes" that range from data marts, data warehouses or unused data dumps. Some of these might be more useful than the rest. These data sets were collected because of the underlying assumption that collected data has value and might improve future decision making processes; but manual analysis of such huge data bodies is not feasible [Weiss and Indurkhya, 1997].

The branch of computer science which deals with analyzing huge data banks and discovering implicit patterns in the data is called machine learning (ML). The field of ML draws on ideas from various disciplines, including artificial intelligence, probability and statistics, computational complexity, information theory, psychology and neurobiology, control theory, and philosophy. The field of ML is concerned with the question of how to construct computer programs that automatically improve with experience. Machine learning algorithms must automatically acquire and integrate knowledge. This capacity to learn from experience, analytical observation, as well as other means, results in a system that can continuously self-improve and thereby offer increased efficiency and effectiveness.

One perspective offered by ML involves searching a very large space of possible hypotheses to determine which best fits the observed data and any prior knowledge held by the learner. This hypothesis space consists of all possible evaluation functions. The learner's task is to search through the vast space to locate the hypothesis that is most consistent with the available training examples. There are various ML algorithms that search a hypothesis space defined by some underlying representation (e.g., linear functions, logical descriptions, decision trees, artificial neural networks). The underlying representation dictates which learning algorithm is to be used to search through the hypothesis space. For example, if we want to approximate discrete-valued target functions we would generally prefer decision tree learning.

Some of the most commonly used ML algorithms to analyze and discover implicit patterns in the data are:

- **Artificial Neural Networks (ANNs)** Artificial Neural Networks [Chauvin and Rumelhart, 1995] provide a general, practical method for learning real-valued, discrete-valued, and vector-valued functions from examples. ANN

learning is robust to errors in the training data and has been successfully applied to problems such as interpreting visual scenes, speech recognition, and learning robot control strategies [Mitchell and Thrun, 1993]. ANNs are among the most effective learning methods currently known.

- **Decision Trees** [Quinlan, 1990, Breiman et al., 1984] are one of the most widely used and practical methods for inductive inference. Decision trees involve approximating discrete-valued target functions in which the learned function is represented by a decision tree. Learned trees can also be re-represented by a set of if-then rules to improve human readability. There are a variety of decision tree learning algorithms such as CART [Breiman et al., 1984], and C4.5 [Quinlan, 1993]. These methods search a completely expressive hypothesis space and thus avoid the difficulties of restricted hypothesis spaces.

- **Instance-Based Learning** methods [Aha et al., 1991] are sometimes referred to as lazy learning methods because they delay processing until a new instance must be classified. Instance-based learning methods simply store the training examples. Each time a new query instance is encountered, its relationship to the previously stored examples is examined in order to assign a target function value for the new instance. Examples of instance-based learning methods are nearest neighbors [Cover and Hart, 1967] and locally weighted regression methods [Atkeson et al., 1997] that assume instances can be represented as points in a Euclidean space. A key advantage of this kind of delayed, or lazy learning is that instead of estimating the target function once for the entire instance space, these methods can estimate it locally and differently for each new instance to be classified.

Machine learning algorithms have been successfully applied in a number of domains including:

- **Learning to drive an autonomous vehicle:** Machine learning methods have been used to train computer-controlled vehicles [Pomerleau, 1989] to steer correctly when driving on a variety of road types.

- **Learning to classify new astronomical structures:** Machine learning methods have been applied to a variety of large databases to learn general regularities implicit in the data. For example, decision tree learning algorithms have been used by NASA [Fayyad et al., 1995] to learn how to classify celestial objects.

- **Learning to play world-class backgammon:** The most successful computer programs for playing games such as backgammon [Tesauro, 1995] are based on ML algorithms.

1.1 Inductive Learning

Inductive learning is a kind of learning in which given a set of examples, the learner tries to estimate an evaluation function. Most inductive learning approaches use a supervised approach in which the training examples are marked with their respective classifications. More formally, a training example is a pair $<x,f(x)>$ where x is the input and $f(x)$ is the output of the function applied to x. The task of pure inductive inference is, given a set of examples for f, to find a hypothesis h that approximates f. The hypothesis h cannot guarantee results on unseen data examples. The fundamental assumption of inductive learning is that the best hypothesis regarding unseen instances is the hypothesis that "best fits" the training data is the best hypothesis for the unseen instances.

1.2 Motivation

Machine learning algorithms provide us with techniques that help us analyze huge volumes of data and in order to induce implicit relationships among the data values. Many methods have been proposed to induce models, including decision trees [Quinlan, 1993], Bayesian techniques [Jensen, 1996], neural networks with back propagation [Chauvin and Rumelhart, 1995]. These methods have proven to be computationally intensive when the volume of data is large. This issue has generated lot of interests in the ML community. Today there is a need to scale up ML algorithms to analyze very large data sets in order to discover implicit patterns and outliers within the data set. A fast and accurate tool for data analysis will prove to be extremely beneficial. We need the tool to be fast as the amount of data to be analyzed is very large and at the same time we cannot overlook accuracy as the analyzes carried out on the data will be used in future classifications. Researchers [Bauer and Kohavi, 1999, Efron and Tibshirani, 1993, Maclin and Opitz, 1997] have investigated the technique of combining multiple component classifiers to produce a single classifier. The resulting classifier, (hereafter referred to as an ensemble) is generally more accurate than any of the individual classifiers making up the ensemble. Bagging [Breiman, 1996a] and Boosting [Freund and Schapire, 1996] are currently the two most popular methods for building ensembles. Building an ensemble classifier is a time consuming process and this does not agree with our requirement of

building a fast classifier. What we need is a a new method to build an ensemble that is faster than the existing ensemble building methods and also does not compromise the accuracy of the learned system.

1.3 Statement of Thesis

Of the two methods for ensemble building, Bagging and Boosting, Boosting is generally preferred because it often produces the lowest error rates. But Boosting suffers from two drawbacks:

- The time taken to build the classifier is high as it cannot be built in parallel
- Overfitting occurs as the number of classifiers increases

The first drawback affects the speed of building the ensemble and the second affects the accuracy. *This thesis aims at studying a new method of building an ensemble that performs similar to Boosting, but which can be executed in parallel. Its accuracy will be compared with the two traditional approaches of building ensembles (i.e., Bagging and Boosting).* This new method will by sampling data points that are close to each other in the problem space, build regional classifiers. These regional classifiers will then be combined to produce a global classifier to cover the entire problem space. We use decision trees to build our ensembles. This thesis will answer the following questions:

- *Question 1:* How does the new ensemble building method compare to Bagging and Boosting in terms of test set error rates?
- *Question 2:* What is the optimal size of the ensemble (i.e., the number of classifiers making up the ensemble)?
- *Question 3:* Is the new method faster than the Boosting?

This thesis will help us determine whether this new method to build ensembles is as accurate if not more as compared to Boosting.

1.4 Outline of Thesis

In the following chapters this thesis will cover the necessary background about decision trees and ensemble learning, the new ensemble learning method that we developed, the results of our experiments followed by concluding remarks and the questions this thesis has answered. Below is a brief survey of the chapters.

- Chapter 2 covers topics related to decision tree learning such as decision tree representation, the C4.5 decision tree building algorithm, ensemble learning, and the existing techniques for building ensembles.
- Chapter 3 introduces the concepts required for the new ensemble building method. It also examines the new algorithm and discusses how it differs from the existing ensemble building techniques.
- Chapter 4 examines the results of the new ensemble building technique and does a comparative study with the two existing ensemble building methods.
- Chapter 5 presents the conclusions for the thesis based on the results, answers the questions posed in Chapter 1 and proposes directions for future research.

2 Background

Decision tree learning [Quinlan, 1993, Breiman et al., 1984] is one of the most widely used and practical methods for inductive inference [Quinlan, 1990]. It is a method for approximating discrete-valued functions that is robust to noisy data and capable of learning disjunctive expressions. This chapter focuses on decision tree learning and the various mechanisms for building an ensemble of classifiers.

2.1 Decision Tree Learning

Decision tree learning is a method for approximating discrete-valued target functions. There are various decision tree building algorithms such as ID3 [Quinlan, 1990], CART [Breiman et al., 1984], and C4.5 [Quinlan, 1993]. The inductive bias of these methods is a preference for smaller trees over large trees; that is, its search through the hypothesis space produces a tree only as large as needed in order to classify the available training examples.

A decision tree is a model that is both predictive and descriptive. It is called a decision tree because the resulting model is presented in the form of a tree structure. The visual presentation makes the decision tree easy to understand and assimilate. As a result, decision trees are a very popular data mining technique [Agarwal and Shim, 1996, Provost and Kolluri, 1999]. Decision trees graphically display the relationships found in data. The tree can also be translated into rules [Quinlan, 1990] such as

If Income = High and Years on job > 5 Then Credit risk = Good

to improve human readability.

2.2 Decision Tree Representation

Decision trees classify instances by sorting them down the tree [Mitchell, 1997, Quinlan, 1993] from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.

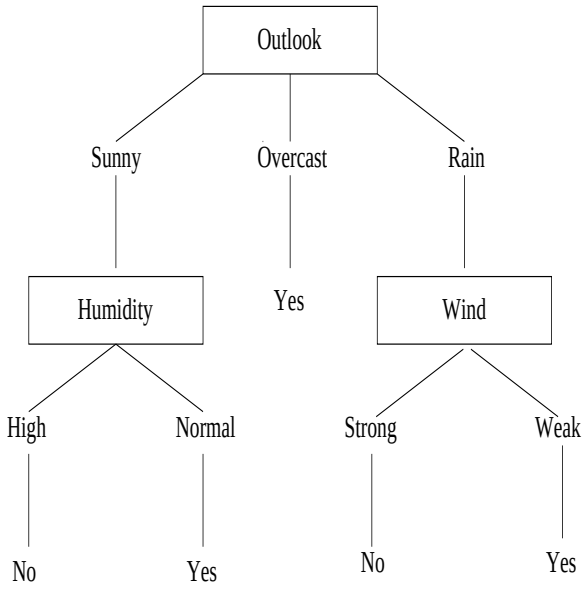


Figure 1: A decision Tree for the concept *PlayTennis*. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf (in this case, *Yes* or *No*). This tree classifies Saturday mornings according to whether or not they are suitable for playing tennis.

Figure 1 shows a typical learned decision tree. This decision tree classifies Saturday mornings according to whether they are suitable for playing tennis. For example, the instance

$$< Outlook = Sunny, Temp = Hot, Humidity = High, Wind = Strong >$$

would be sorted down to the leftmost branch of this decision tree and would therefore be classified as a negative instance (i.e., the tree predicts that *PlayTennis* $= No$).

In general, decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances. Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions. For example, the decision tree shown in Figure 1 corresponds to the expression

If (Outlook = Sunny $\wedge$ *Humidity = Normal)*
$\vee$ *(Outlook = Overcast)*
$\vee$ *(Outlook = Rain* $\wedge$ *Wind = Weak)*
Then PlayTennis = Yes

2.3 Appropriate Problems For Decision Tree Learning

Although a variety of decision tree learning methods have been developed with somewhat differing capabilities and requirements, decision tree learning is generally suited to problems with the following characteristics:

- Instances are represented by attribute-value pairs. Instances are described by a fixed set of attributes (e.g., Temperature) and their values (e.g., Hot). The easiest situation for decision tree learning is when each attribute takes on a small number of disjoint possible values (e.g., Hot, Mild, Cold). However, extensions to the basic algorithm allow handling real-valued attributes as well (e.g., representing *Temp* numerically).

- The target function has discrete output values. The decision tree in Figure 1 assigns a Boolean classification (e.g., true or false) to each example. Decision tree methods easily extend to learning functions with more than two possible output values.

- Disjunctive descriptions may be required. As noted above, decision trees naturally represent disjunctive expressions.

- The training data may contain errors. Decision tree learning methods are robust to errors, both in classification of the training examples and in the attribute values that describe these examples.

- The training data may contain missing attribute values. Decision tree methods can be used even when some training examples have unknown values.

Many practical problems have been found to fit these characteristics. Decision tree learning has therefore been applied to problems such as learning to classify medical patients by their disease [Kononenko et al., 1984] , equipment malfunctions by their cause, and loan applications by their likelihood of defaulting on payments [Stolfo et al., 1997]. Such problems, in which the task is to classify examples into one of a discrete set of possible categories, are often referred to as classification problems.

2.4 The Basic Decision Tree Learning Algorithm

Most algorithms that have been developed for learning decision trees are variations on a core algorithm that employs a top-down, greedy search through the space of possible decision trees. This approach is exemplified by the CART and ID3 algorithms and ID3's successor C4.5. C4.5 is the approach that we will be following in this thesis. The following subsections describe CART, ID3 and its successor C4.5.

2.4.1 CART

CART [Breiman et al., 1984] is an acronym for Classification and Regression Trees. CART uses strictly binary, or two-way, splits that divide each parent node into exactly two child nodes by posing questions with yes/no answers at each decision node. CART searches for questions that split nodes into relatively homogeneous child nodes, such as a group consisting largely of responders, or high credit risks, or people who bought sport-utility vehicles.

As the tree evolves, the nodes become increasingly more homogeneous, identifying important segments. CART's binary decision trees detect more structure before too little data is left for learning. Other decision-tree approaches use multi-way splits that fragment the data rapidly, making it difficult to detect rules that require broad ranges of data to discover. In the search for patterns in databases it is essential to avoid the trap of overfitting, or finding patterns that apply only to the training data. The testing and selection of the optimal tree are an integral part of the CART algorithm. Testing in other decision-tree techniques is conducted after-the-fact and tree selection is left up to the user. In addition, CART accommodates many different types of real-world modeling problems by providing a unique combination of automated solutions:

- Surrogate splitters intelligently handle missing values,
- Adjustable misclassification penalties help avoid the most costly errors,
- Multiple-tree, committee-of-expert methods increase the precision of results, and
- Alternative splitting criteria make progress when other criteria fail

2.4.2 C4.5

ID3 and C4.5 [Quinlan, 1993] form the primary focus of our discussion. As mentioned above, C4.5 is the decision tree algorithm that we use to build our trees in this work.

2.4.3 Which attribute is the best classifier?

The central choice in the C4.5 algorithm is selecting which attribute to test at each node in the tree. We would like to select the attribute that is most useful for classifying examples. Information gain is a measure which helps us to determine the best attribute. C4.5 uses the information gain measure to select among the candidate attributes at each step while growing the tree. In order to define information gain precisely, we begin by defining a measure commonly used in information theory, called *entropy*, that characterizes the impurity of an arbitrary collection of examples. Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this Boolean classification is

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus} \tag{1}$$

where p+ is the proportion of positive examples in S and p- is the proportion of negative examples in S. In all calculations involving entropy we define $0log_20$ to be 0. To illustrate, suppose S is a collection of 14 examples of some Boolean concept, including 9 positive and 5 negative examples. Then the entropy of S relative to this Boolean classification is

$$\begin{aligned} Entropy([9+, 5-]) &= (-9/14)\log_2(9/14) - (5/14)\log_2(5/14) \\ &= 0.940 \end{aligned} \tag{2}$$

Note that the entropy is 0 if all members of S belong to the same class. For example, if all members are positive, then p- is 0, and

$$Entropy(S) \equiv -1.\log_2(1) - 0.\log_2(0) \equiv 0 \tag{3}$$

Note the entropy is 1 when the collection contains an equal number of positive and negative examples. If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1. Figure 2 shows the form of the entropy function relative to a Boolean classification as p+ varies between 0 and 1.

We have thus far discussed entropy in the special case where the target classification is Boolean. More generally, if the target attribute can take on c different values, then the entropy of S relative to this c-wise classification is defined as

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i \tag{4}$$

Table 1: An algorithm for C4.5 decision tree learning

C4.5(*Examples, Target_attribute, Attributes*)
Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = −
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
- Otherwise
    * $A \leftarrow$ the attribute from *Attributes* that *best** classifies *Examples*
    * The decision attribute for *Root* $\leftarrow A$
    * For each possible value v_i of A,
        * Add a new tree branch below *Root* corresponding to the test $A = v_i$
        * Let $Examples_{vi}$ be the subset of *Examples* that have value v_i for A
        * if $Examples_{vi}$ is empty
            · Then below this new branch add a new leaf node with label = most common value of *Target_attribute* in *Examples*
            · Else below this new branch add the subtree C4.5($Examples_{vi}$, *Target_attribute, Attributes* - $\{A\}$)
- Return *Root*

**The best attribute is the one with highest information gain, as defined in Equation 5 .*

where p_i is the proportion of S belonging to class i. Note the logarithm is still

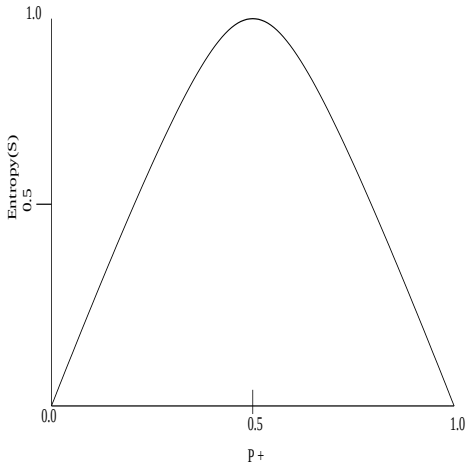


Figure 2: The entropy function relative to a Boolean classification, as the proportion of positive examples varies between 0 and 1. Note that entropy is highest when proportion of positive examples is 0.5

base 2 because entropy is a measure of the expected encoding length measured in bits. Note also that if the target attribute can take on c possible values, the entropy can be as large as $log_2 c$.

2.4.4 Information Gain Measures The Expected Reduction in Entropy

Given entropy as a measure of the impurity in a collection of training examples, we can now define a measure of the effectiveness of an attribute in classifying the training data. The measure we will use, called *information gain*, is simply the expected reduction in entropy caused by partitioning the examples according to this attribute. The information gain, Gain(S,A) of an attribute A, relative to a collection of examples S, is defined as

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{5}$$

where Values(A) is the set of all possible values for attribute A, and S_v is the subset of S for which attribute A has value v (i.e., $S_v = \ s \in S | A(s) = v$). Note the first

term in Equation 5 is just the entropy of the original collection S, and the second term is the expected value of the entropy after S is partitioned using attribute A. The expected entropy described by this second term is simply the sum of the entropies of each subset S_v, weighted by the fraction of examples $|S_v|/|S|$ that belong to S_v. Gain(S,A) is therefore the expected reduction in entropy caused by knowing the value of attribute A. Gain(S,A) is the information provided about the target function value, given the value of some other attribute A. The value of Gain(S,A) is the number of bits saved when encoding the target value of an arbitrary member of S, by knowing the value of attribute A. For example, suppose S is a collection of training-example days described by attributes including wind, which can have the values Weak or Strong. Assume S is a collection containing 14 examples, [9+,5-]. Of these 14 examples, suppose 6 of the positive and 2 of the negative examples have $Wind = weak$, and the remainder have $Wind = Strong$. The information gain due to sorting the original 14 examples by the attribute Wind may then be calculated as

$$
\begin{aligned}
Values(Wind) &= Weak, Strong \\
S &= [9+, 5-] \\
S_{Wind=Weak} &\leftarrow [6+, 2-] \\
S_{Wind=Strong} &\leftarrow [3+, 3-] \\
Gain(S, Wind) &= Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v) \\
&= Entropy(S) - (8/14) Entropy(S_{Wind=Weak}) \\
&\quad -(6/14) Entropy(S_{Wind=Strong}) \\
&= 0.940 - (8/14)0.811 - (6/14)1.00 \\
&= 0.048
\end{aligned}
$$

Information gain is used by C4.5 to select the best attribute at each step in growing the tree. The use of information gain to evaluate the relevance of attributes is summarized in Figure 3. In this Figure the information gain of two different attributes, Humidity and Wind, is computed in order to determine which is the better attribute for classifying the training examples.

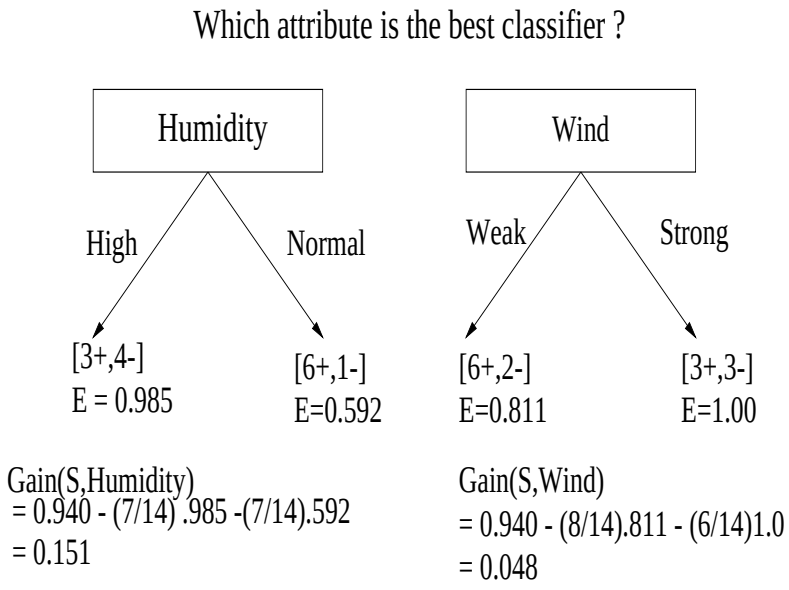


Figure 3: Humidity provides greater information gain than Wind, relative to the target classification. Here, E stands for entropy and S for the original collection of examples. Given an initial collection S of 9 positive and 5 negative examples, $[9+, 5-]$, sorting these by their *Humidity* produces collections of $[3+, 4-]$ ($Humidity = High$) and $[6+, 1-]$ ($Humidity = Normal$). The information gained by this partitioning is 0.151, compared to a gain of only 0.048 for the attribute $Wind$.

2.4.5 An Illustrative Example

To illustrate the operation of C4.5, consider the learning task represented by the training examples [Mitchell, 1997] of Table 2. Here the target attribute *PlayTennis*, which can have values yes or no for different Saturday mornings, is to be predicted based on other attributes of the morning in question. Consider the first step, in which the topmost node of the decision tree is created. Which attribute should be tested first in the tree? C4.5 determines the information gain for each candidate attribute (i.e., Outlook, Temperature, Humidity, and Wind), then selects the one with highest information gain. The computation of information gain for two of these attributes is shown in Figure 3. The information gain values for all four attributes are

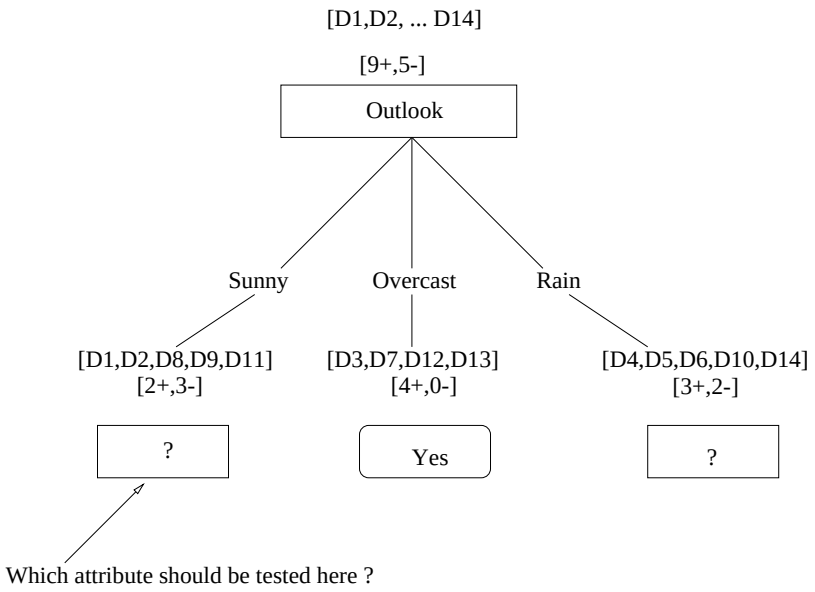
According to the information gain measure, the *Outlook* attribute provides the best prediction of the target attribute, *PlayTennis*, over the training examples. Therefore, *Outlook* is selected as the decision attribute for the root node, and branches are

Gain(S, Outlook) = 0.246
Gain(S, Humidity) = 0.151
Gain(S, Wind) = 0.048
Gain(S, Temperature) = 0.029

Table 2: The training examples for the target concept *PlayTennis*.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

created below the root for each of its possible values (i.e., *Sunny, Overcast and Rain*). The resulting partial decision tree is shown in Figure 4, along with the training examples sorted to each new descendent node. Note that every example for which $Outlook = Overcast$ is also a positive example of *PlayTennis*. Therefore, this node of the tree becomes a leaf node with the classification *PlayTennis* = Yes. In contrast, the descendents corresponding to $Outlook = Sunny$ and $Outlook = Rain$ still have nonzero entropy, and the decision tree will be further elaborated below these nodes. The process of selecting a new attribute and partitioning the training examples is now repeated for each nonterminal descendent node, this time using only the training examples associated with that node. Attributes that have been incorporated higher in the tree are excluded, so that any given attribute can appear at most once along



$S_{sunny} = D1, D2, D8, D9, D11$
$Gain(S_{sunny}, Humidity) = 0.970 - (3/5)0.0 - (2/5)0.0 = .970$
$Gain(S_{sunny}, Temperature) = 0.970 - (2/5)0.0 - (2/5)1.0 = .570$
$Gain(S_{sunny}, Wind) = 0.970 - (2/5)1.0 - (3/5).918 = .019$

Figure 4: The partially learned decision tree resulting from the first step of *C4.5*. The training examples are sorted to the corresponding descendant nodes. The *Overcast* descendent has only positive examples and therefor becomes a leaf node with classification Yes. The other two nodes will be further expanded, by selecting the attribute with highest information gain relative to the new subsets of examples.

any path through the tree. This process continues for each new leaf node until either of the following two conditions are met: (1) every attribute has already been included along this path through the tree or (2) the training examples associated with this leaf node all have the same target attribute value (i.e., their entropy is zero). Figure 4 illustrates the computations of information gain for the next step in growing the decision tree. The final decision tree learned by C4.5 from the 14 training examples of Table 2 is shown in Figure 1.

2.4.6 Hypothesis Space Search In Decision Tree Learning

C4.5 searches a hypotheses space for one that fits the training examples. The hypothesis space searched by C4.5 is the set of possible decision trees. C4.5 performs a simple-to-complex, hill-climbing search through this hypothesis space, beginning with the empty tree, then considering progressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data. The evaluation function that guides this hill-climbing search is the information gain measure. By viewing C4.5 in terms of its search space and search strategy, we can get some insight into its capabilities and limitations.

C4.5's hypothesis space of all decision trees is a complete space of finite discrete-valued functions, relative to the available attributes. Because every finite discrete-valued function can be represented by some decision tree, C4.5 avoids one of the major risks of methods that search incomplete hypothesis spaces. C4.5 maintains only a single current hypothesis as it searches through the space of decision trees. By determining only a single hypothesis, C4.5 loses the capabilities that follow from explicitly representing all consistent hypotheses. C4.5 in its pure form performs no backtracking in its search. Once it selects an attribute to test at a particular level in the tree, it never backtracks to reconsider this choice and hence it is susceptible to the usual risks of hill-climbing search without backtracking: converging to a local optima. The disadvantage here is that we will not get to the best hypotheses. C4.5 uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis. An advantage of using this method is that the resulting search is much less sensitive to errors in individual training examples.

2.4.7 Inductive Bias in Decision Tree Learning

Given a collection of training examples, there are typically many decision trees consistent with these examples. The inductive bias of C4.5 therefore consists of describing the basis by which it chooses one of these consistent hypotheses over the others. Which of these decision tree does C4.5 choose? It chooses the first acceptable tree it encounters in its simple-to-complex, hill-climbing search through the space of possible trees. The C4.5 search strategy (a) selects in favor of shorter trees over longer ones and (b) selects trees that place the attributes with highest information gain closest to the root. We can approximately characterize its bias as a preference for short decision trees with high information gain over complex trees.

2.4.8 Issues in Decision Tree Learning

Practical issues in learning decision trees include determining how deeply to grow the decision tree, handling continuous-valued attributes, choosing an appropriate attribute selection measure, handling training data with missing attribute values, handling attributes with differing costs, and improving computational efficiency.

1. **Avoiding Overfitting the Data**

   The algorithm described in Table 1 grows each branch of the tree just deeply enough to perfectly classify the training examples. While this is sometimes a reasonable strategy, in fact it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. In either cases, this simple algorithm can produce trees that overfit the training examples. *Definition:* Given a hypothesis space H, a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.

   Figure 5 illustrates the effect of overfitting in a typical application of decision

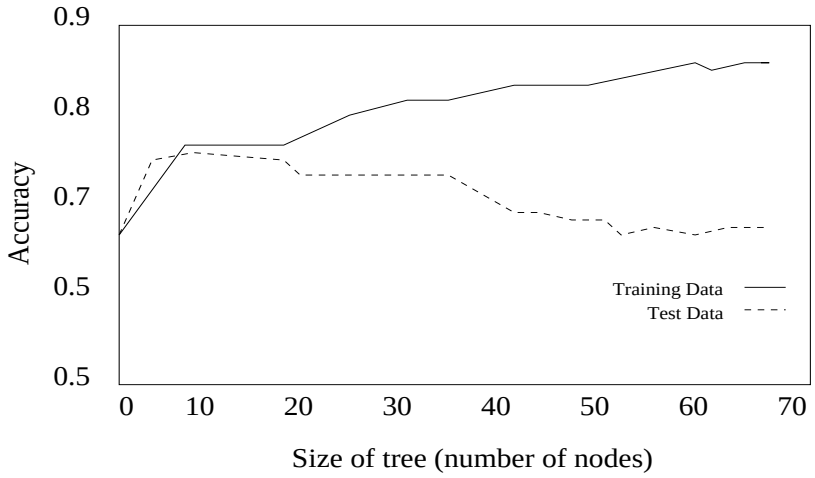


Figure 5: Overfitting in decision tree learning. As C4.5 adds new nodes to grow the decision tree, the accuracy of the tree measured over the training examples increases monotonically. However, when measured over a set of test examples independent of the training examples, accuracy first increases, then decreases.

learning. In this case, the C4.5 algorithm is applied to the task of learning which medical patients have a form of diabetes. The horizontal axis of this plot indicates the total number of nodes in the decision tree, as the tree is being constructed. The vertical axis indicates the accuracy of predictions made by the tree. The solid line shows accuracy measured over an independent set of test examples (not included in the training set). Predictably, the accuracy of the tree over the training examples increases monotonically [Mitchell, 1997] as the tree grows. However, the accuracy measured over the independent test examples first increases, then decreases. As can be seen, once the tree size exceeds approximately 25 nodes, further elaboration of the tree decreases its accuracy over the test examples despite increasing its accuracy on the training examples. Random noise in the training example can lead to overfitting. In fact, overfitting is possible even when the training data are noise-free, especially when small numbers of examples are associated with leaf nodes. In this case, it is quite possible for coincidental regularities to occur in which some attribute happens to partition the examples very well, despite being unrelated to the actual target function. Whenever such coincidental regularities exist, there is a risk of overfitting. Overfitting is a significant practical difficulty for decision trees learning. Overfitting was found to decrease the accuracy of learned decision tree by 10-25% on most problems.

2. **Incorporating Continuous-Valued Attributes**
   The initial definition of C4.5 is restricted to attributes that take on a discrete set of values. First, the target attribute whose value is predicted by the learned tree must be discrete valued. Second, the attributes tested in the decision nodes of the tree must also be discrete-valued. This second restriction can be easily be removed so that continuous-valued decision attributes can be incorporated into the learned tree. This can be accomplished by dynamically defining new discrete-valued attributes that partition the continuous-valued attribute into a discrete set of intervals. In particular, for an attribute A that is continuous-valued, the algorithm can dynamically create a new Boolean attribute A_c that is true if $A < c$ and false otherwise. The only question is how to select the best value for the threshold c. As an example, suppose we wish to include the continuous-valued attribute Temperature in describing the training example days in the learning task of Table 3. Suppose further the training examples associated with a particular node in a decision tree have the following values for Temperature and the target attribute *PlayTennis*.

   We need to a select a threshold, c, that produces the greatest information gain.

Table 3: Continuous Valued Attributes. Temperature is the continuous-valued attribute. For each temperature value, the corresponding classification for *PlayTennis* is shown.

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

By sorting the examples according to the continuous-valued attribute A, then identifying adjacent examples that differ in their target classification, we can generate a set of candidate thresholds midway between the corresponding values of A. In the current example, there are two candidate thresholds, corresponding to the values of Temperature at which the value of *PlayTennis* changes: (48 + 60)/2, and (80 + 90)/2. The information gain can be computed for each of the candidate attributes. This dynamically created Boolean attribute can then compete with the other discrete-valued candidate attributes available for growing the decision tree.

3. **Alternative Measures for Selecting Attributes**
There is a natural bias in the information gain measure that favors attributes with many values over those with few values. Such attributes are bound to separate the training examples into very small subsets. Because of this, it will have a very high information gain relative to the training examples, despite being a poor predictor of the target function over unseen instances. This difficulty can be avoided by selecting decision attributes based in some measure other than information gain. One alternative measure that has been used successfully is the *gain ratio* [Quinlan, 1990]. The gain ratio measure penalizes attributes such as date by incorporating a term, called *split information*, that is sensitive to how broadly and uniformly the attribute splits the data:

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|} \tag{6}$$

where S_1 through S_c are the subsets of examples resulting from partitioning S by the c-valued attribute A. Note that SplitInformation is actually the entropy of S with respect to the values of Attribute A. The GainRatio measure is defined in terms of the earlier Gain measure, as well as this SplitInformation, as follows

$$GainRatio(S, A) = Gain(S, A)/SplitInformation(S, A) \quad (7)$$

The SplitInformation term discourages the selection of valued attributes with many uniformly distributed values.

4. **Handling Training Examples with Missing Attribute Values**
   In certain cases, the available data may have missing values for some attributes. For example, in medical domain in which we wish to predict patient outcome based on various laboratory tests, it may be that the lab test Blood-Test-Result is available only for a subset of patients. In such cases, it is common to estimate the missing attribute value based on other examples for which this attribute has a known value.

   Consider the situation in which $Gain(S, A)$ is to be calculated at node n in the decision tree to evaluate whether the attribute A is the best attribute to test at this decision node. Suppose that $< x, c(x) >$ is one of the training examples in S and that the value $A(x)$ is unknown. One approach, that is used in C4.5 is to assign a probability to each of the possible values of A rather than simply assigning the most common value to $A(x)$. These probabilities can be estimated again based on the observed frequencies of the various values for A among the examples at node n. For example, given a Boolean attribute A, if node n contains six known examples with $A = 1$ and four with $A = 0$, then we would say the probability that $A(x) = 1$ is 0.6 and the probability that $A(x) = 0$ is 0.4. A fractional 0.6 of instance x is now distributed down the branch for $A = 1$, and a fractional 0.4 of x down the other tree branch. These fractional examples are used for the purpose of computing Information gain and can be further subdivided at subsequent branches of the tree if a second missing attribute value must be tested. This same fractioning of examples can also be applied after learning to classify new instances whose attribute values are unknown. In this case, the classification of the new instance is simply the most probable classification, computed by summing weights of the instance fragments classified in different ways at the leaf nodes of the tree.

5. **Handling Attributes with Differing Costs**
   In some learning tasks the instances may have associated costs. For example, in learning to classify medical diseases we might describe patients in terms of attributes such as temperature, BiopsyResult, Pulse, BloodTestResults, etc.

These attributes vary significantly in their costs, both in terms of monetary cost and cost to patient comfort. In such tasks, we would prefer decision trees that use low-cost attributes where possible, relying on high-cost attributes only when needed to produce reliable classifications. C4.5 can be modified to take into account attribute costs by introducing a cost term into the attribute selection measure. For example, we might divide the gain by the cost of the attribute, so that the lower cost attributes would be preferred. While such cost-sensitive measures do not guarantee finding an optimal cost-sensitive decision tree, they do bias the search in favor of low-cost attributes. One such approach defined by Tan and Schlimmer(1990) is given by the formula

$$Gain2(S, A)/Cost(A) \tag{8}$$

where $Cost(A)$ denotes the cost of Attribute A.

2.5 Ensemble Learning

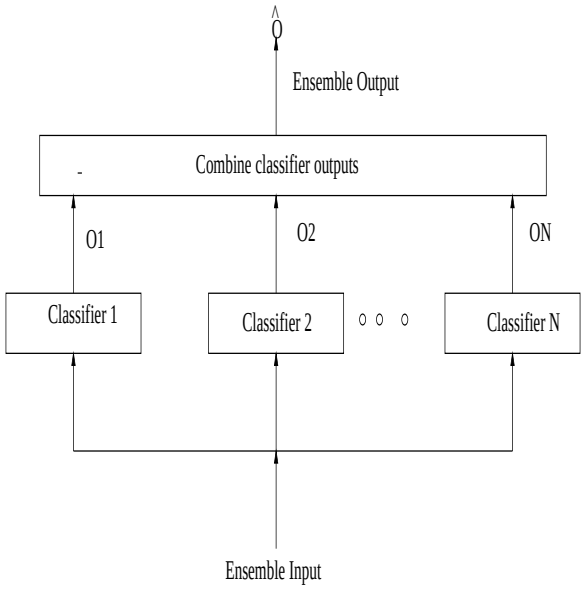


Figure 6: A ensemble of classifiers. The output from each of the classifier is combined to get the ensemble output.

An ensemble consists of a set of individually trained classifiers whose predictions are combined when classifying novel instances. Previous research has shown that

Table 4: Hypothetical runs of Bagging and Boosting. Assume there are eight training examples. Assume example 1 is an "outlier" and is hard for the component learning algorithm to classify correctly. With Bagging, each training set is an independent sample of the data; thus, some examples are missing and others occur multiple times. The Boosting training sets are also samples of the original data set, but the "hard" example (example-1) occurs more in later training sets since Boosting concentrates on correctly predicting it.

A sample of a single classifier on an imaginary set of data.
(Original) Training Set
Training-set-1: 1,2,3,4,5,6,7,8

A sample of Bagging on the same data.
(Resampled) Training Set
Training-set-1: 2, 7, 8, 3, 7, 6, 3, 1
Training-set-2: 7, 8, 5, 6, 4, 2, 7, 1
Training-set-3: 3, 6, 2, 7, 5, 6, 2, 2
Training-set-4: 4, 5, 1, 4, 6, 4, 3, 8

A sample of Boosting on the same data.
(Resampled) Training Set
Training-set-1: 2, 7, 8, 3, 7, 6, 3, 1
Training-set-2: 1, 4, 5, 4, 1, 5, 6, 4
Training-set-3: 7, 1, 5, 8, 1, 8, 1, 4
Training-set-4: 1, 1, 6, 1, 1, 3, 1, 5

an ensemble is often more accurate than any of the single classifiers in the ensemble [Breiman, 1996a, Freund and Schapire, 1996] are two relatively new but popular methods of producing ensembles. These methods rely on "resampling" techniques to obtain different training sets for each of the classifiers. Research has demonstrated that a good ensemble is one where the individual classifiers in the ensemble are both accurate and make their errors on different parts of the input space. This section describes each of these techniques in greater detail.

2.6 Bagging Classifiers

Bagging is a "bootstrap" ensemble [Breiman, 1996a, Efron and Tibshirani, 1993] method that creates individuals for its ensemble by training each classifier on a ran-

dom redistribution of the training set. Each classifier's training set is generated by randomly drawing, with replacement, N examples, where N is the size of the original training set. Many of the original examples may be repeated in the resulting training set while others may be left out. Each individual classifier in the ensemble is generated with a different random sampling of the training set. Table 4 gives a sample of how Bagging works on a imaginary set of data. Since Bagging resamples the training set with replacement, some instances are represented multiple times while others are left out. So Bagging's training-set-1 might contain examples 3 and 7 twice, but does not contain either example 4 or 5. As a result, the classifier trained on training-set-1 might obtain a higher test-set error than the classifier using all of the data. In fact all four of Bagging's component classifiers could result in higher test-set error; however when combined, these four classifiers can (and often do) produce test-set error lower than that of the single classifier (the diversity among these classifiers generally compensates for the increase in error rate of any individual classifier). A predicted class corresponding to a new input is obtained by a plurality vote among the four classifiers. Consequently, each new case must be classified by all the the classifiers, (decision trees in our case) and a running tally kept of the results. Bagging is effective on "unstable" learning algorithms where small changes in the training set result in large changes in predictions. Decision trees are unstable learning algorithms.

2.7 Boosting Classifiers

Boosting [Freund and Schapire, 1996] encompasses a family of methods. The focus of these methods is to produce a series of classifiers. The training set used for each member of the series is chosen based on the performance of the earlier classifier(s) in the series. In Boosting, examples that are incorrectly predicted by previous classifiers in the series are chosen more often than examples that were correctly predicted. Thus Boosting attempts to produce new classifiers that are better able to predict examples for which the current ensemble's performance is poor. There are two popular forms of Boosting: Ada-Boosting [Freund and Schapire, 1996] and Arcing [Breiman, 1996b]. Like Bagging, Arcing chooses a training set of size N for classifier $K+1$ by probabilistically selecting (with replacement) examples from the original N training examples. Unlike Bagging, however, the probability of selecting an example is not equal across the training set. This probability depends on how often that example was misclassified by the previous K classifiers. Ada-Boosting can use the approach of (a) selecting a set of examples based on the probabilities of the examples, or (b) use all the examples and weight each example (i.e., examples with higher weights have more effect on the error).

Both Arcing and Ada-Boosting initially set the probability of picking each example to be $1/N$. These methods then recalculate these probabilities after each trained classifier is added to the ensemble. For Ada-Boosting, let E_k be the sum of the probabilities of the misclassified instances for the currently trained classifier C_k. The probabilities for the next trial are generated by multiplying the probabilities of $C'_k s$ incorrectly classified instances by the factor $B_k = (1\text{-}E_k)/E_k$ and then renormalizing all probabilities so that their sum equals 1. Ada-Boosting combines the classifiers $C_1, ..., C_k$ using weighted voting - c_k has weight $log(B_k)$. These weights allow Ada-Boosting to discount the predictions of classifiers that are not very accurate on the overall problem.

In this work, I have used a revision described by Brieman (1996b) in which we reset all the weights to be equal and restart if either E_k is not less than 0.5 or E_k becomes 0. Table 4 shows a hypothetical run of Boosting. Note that the first training set would be the same as Bagging; however, later training sets accentuate examples that were misclassified by the earlier member of the ensembles. In Table 4, example-1 is a "example" that previous classifiers tend to misclassify. With the second training set, example 1 occurs multiple times, as do examples 4 and 5 since they were left out of the first training set and, in this case, misclassified by the first learner. For the first training set, example 1 becomes the predominant example chosen; thus, the overall test-set error for this classifier might become very high. Despite this, however, Boosting will probably obtain a lower error rate when it combines the output of these four classifiers since it focuses on correctly predicting previously misclassified examples and weights the predictions of the different classifiers based on their accuracy for the training set. But Boosting can also overfit in the presence of noise.

2.8 The Bias Plus Variance Decomposition

Classification error can be regarded as a combination of bias and variance. We can determine the effectiveness of Bagging and Boosting by taking into account their classification errors. In this decomposition we can view the expected error of a learning algorithm on a particular target function and training set as having three components:

- A bias term measuring how close the average classifier produced by the learning algorithm will be to the target function;
- A variance term measuring how much each of the learning algorithm's guesses will vary with respect to each other (how often they disagree); and
- A term measuring the minimum classification error associated with the Bayes

optimal classifier for the target function (this term is sometimes referred to as the intrinsic target noise).

Using this framework it has been suggested [Breiman, 1996b] that both Bagging and Boosting reduce error by reducing the variance term. Freund and Schapire (1996) argue that Boosting also attempts to reduce the error in the bias term since it focuses on misclassified examples. Such a focus may cause the learner to produce an ensemble function that differs significantly from the single learning algorithm, In fact, Boosting may construct a function that is not even producible by its component learning algorithm (e.g., changing linear predictions into a classifier that contains non-linear predictions). It is this capacity that makes Boosting an appropriate algorithm for combining the predictions of "weak" learning algorithms (i.e., algorithms that have a simple learning bias).

The bias-variance decomposition cannot be applied to real world data sets as we need to know the actual function being learned. This is unavailable for real-world problems. To deal with this problem, Kohavi and Wolpert [Kohavi and Wolpert, 1996] suggest holding out some of the data. The problem with this technique is that the training set size is greatly reduced in order to get a good estimates of the bias and variance terms.

3 My Ensemble Method

In this thesis we are concerned with building a fast and an accurate inductive learner. One popular method to build an accurate inductive learner is by building an ensemble of classifiers. This section describes a new algorithm of building an ensemble of classifiers. The technique described in this section deals with building an ensemble of classifiers by partitioning the data.

3.1 A New Ensemble Building Method

My new method focuses on building a global classifier by building an ensemble of piece-wise classifiers (i.e., each member of the ensemble focuses on prediction in a portion of the input space).

Each classifier in this model is trained to perform well within a sub-region of the problem space, and by combining all the classifiers, we are covering the entire problem space and building a global classifier. My method for building an ensemble assumes all instances correspond to points in the n-dimensional space $\Re^n$. The dimension of the problem space is equal to the number of attributes in the data set. We are interested in identifying all the data points that are close to each other in the problem space. We randomly select one data point from the training set. This data point will be referred to as the central data point. The data points that are close to this central data point are known as its neighbors. The neighbors of an instance are defined in terms of the standard Euclidean distance. The proximity of a data point to other data points in the problem space is inversely proportional to the distance from those data points. We are interested in identifying all the data points that are close to the central data point.

More precisely, let an arbitrary instance x be described by the feature vector:

$$< a_1(x), a_2(x), \cdots a_n(x) >$$

where $a_r(x)$ denotes the value of the r^{th} attribute of instance x. Then the distance between two instances x_i and and x_j is defined to be $d(x_i, x_j)$, where

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2} \tag{9}$$

By using the Euclidean distance as our metric, we are attempting to cluster all the data points which are close to each other in the vector space. The data points

within each cluster then serves as an input training set on which individual classifiers (Decision Trees in our case) are trained.

This method differs in the following ways from the earlier two approaches of Bagging [Breiman, 1996a] and Boosting [Freund and Schapire, 1996] of building ensembles, this method differs in the following ways:

- Each classifier's training set contains data items that are close to each other within the problem space. The closeness is defined by the distance metric. (i.e., Euclidean distance in our case). In the case of Bagging the training set contains randomly selected data samples from the data set. And Boosting attempts to select data samples that were incorrectly classified by the previous classifiers by increasing the probability of the incorrectly classified samples.

- In our method, there is an attempt to build a set of regional classifiers, each attempting to solve a sub-problem within the entire problem space. Neither Bagging nor Boosting attempt to build regional classifiers. Bagging attempts to reduce error by reducing the variance and Boosting reduces error by reducing the variance and bias.

3.2 An Example

Consider a problem in which we are building a system which is capable of predicting whether an individual is credit worthy or not based on his salary. Figure 7 shows the training data points for this example in the two dimensional problem space. On the X-axis we have the loan amount and on the Y-axis we have that individual's salary. The points within the circle are within one cluster. The function that has to be learned by the system in this case is fairly simple; if the ratio of the salary to the loan amount is less than some constant, then the person is credit worthy and can be safely given the loan amount, otherwise the person is not credit-worthy. Our method of building an ensemble attempts to identify the data points that are close to each other in the problem space and then trains different classifiers on each of these clusters. The resulting system will be a set of N-classifiers each trained on a local set of data points. When these N-classifiers are combined we get a global system which is capable of predicting the credit worthiness of any given individual with a particular salary and requested loan amount.

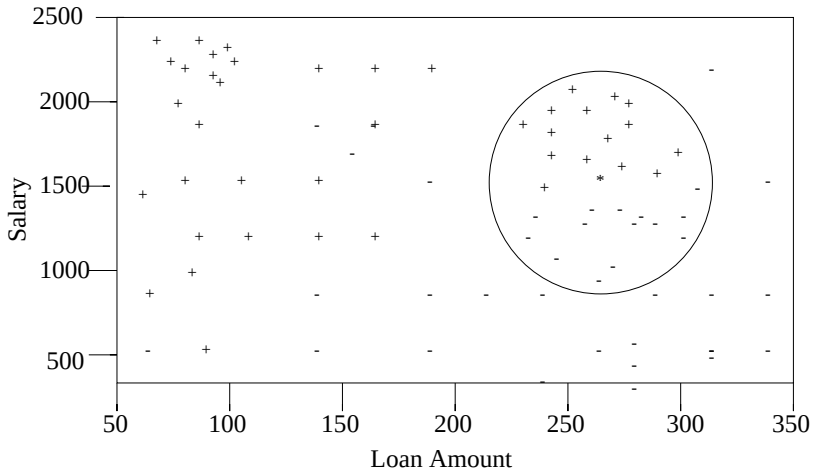


Figure 7: Loan Example: Clusters of closely located data samples. On the X-axis, we have the loan amount and on the Y-axis we have that individual's salary. '+' represents the good data points (i.e, individuals who are credit worthy).'-' represents the bad data points (i.e., individuals who are not credit worthy). The points within the circle represents a cluster of closely located data points. '*' represents the central data point of the cluster. The '*' could either be a good or a bad data point. The central data point is randomly selected. All the training examples within the cluster are used to build a regional classifier.

3.3 Computing Distance

This section describes how we compute the distance between two data points. We need to take into account the following considerations, when computing the distance between two two data points:

- Distance between two continuous-valued attributes
- Distance between two continuous-valued attributes and when one of the attribute value is unknown
- Distance between two continuous-valued attributes and when both the attribute values are unknown
- Distance between two discrete-valued attributes
- Distance between two discrete-valued attributes and when one of the attribute value is unknown

- Distance between two discrete-valued attributes and when both the attribute values are unknown

Let an arbitrary instance x be described by the feature vector with 6-attributes,

$$< a_1(x), a_2(x), a_3(x), a_4(x), a_5(x), a_6(x) >$$

For this example we will assume that the attributes a_1, a_2, a_3 have continuous values and a_4, a_5, a_6 have discrete values.

We now randomly select one data point, which will be a central data point. Let that central data point be characterized by the following attribute values

$$A \equiv <?, 5, 3, a, ?, b >$$

Attributes which have ? as values imply that the value is unknown.We need to compute the distance of all other data points from the central data point. In this sample run we will compute the distance of the central data point with respect to another data point described by the vector

$$B \equiv <?, ?, 10, ?, ?, a >$$

Since we are using the Euclidean distance metric to compute the distance between two data points, we will be using the formula

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2} \tag{10}$$

In our case since the number of attributes is 6, the value of $n = 6$.

For continuous valued attributes we need to compute the maximum and minimum values present within the data set. For our sample run, we will assume that the maximum and minimum values for the three continuous valued attributes are as as shown in Table 5.

Table 6 displays the distribution of discrete-valued attributes. In our example, we have assumed that the data set contains 10 data points. All the three attributes a_4, a_5, a_6 have 2 possible values. Column-3 for attribute a_4 has 3 values which implies that it has 5 data points with the first value, 3 data points with the second value and 2 data points with unknown values.

When the attribute values are known, irrespective of whether they are continuous-valued or discrete-valued we obtain the distance between them by subtracting the two values. For discrete-valued attributes the distance between two known discrete values is either a 1 if they have different values or a 0 if they are have the same value.

When the attribute values are unknown then we use the following rules:

Table 5: The minimum, maximum and range for continuous-valued attributes. The first column shows the attribute name, the second column shows the minimum value for the attribute, the third column shows the maximum value and the range is displayed in the fourth column.

Attribute	Minimum	Maximum	Range
a_1	0	10	10
a_2	10	100	90
a_3	0	10	10

Table 6: The total values and distribution for discrete-valued attributes. The first column shows the attribute name, the second column shows the number of values the attribute can take, the third column shows the distribution of attribute values.

Attribute	Possible values	distribution
a_4	2	[5,3,2]
a_5	2	[5,5,0]
a_6	2	[5,5.0]

- For continuous valued attributes if either of the attribute values is unknown, then the distance between the two attribute values is taken as half of the range
- For discrete valued attributes if only one of the values is unknown, then we compute the distance between the two attribute value pairs by getting the distribution of the attribute values over the entire training set

$$Distance \equiv \sqrt{(a_1(x) - a_1(y)^2 + (a_2(x) - a_2(y))^2 + \cdots + (a_6(x) - a_6(y))^2}$$

$$\equiv \sqrt{(?-?)^2 + (5-?)^2 + (3-10)^2 + (1-?)^2 + (?-?)^2 + (b-a)^2}$$

$$\equiv \sqrt{(5)^2 + (55)^2 + (7/10)^2 + ((5*1)/10 + (3*0)/10)^2 + 1 - ((3/4)^2 + (1/4)^2) + (1)^2}$$

Table 7: New method for building ensembles

Training Algorithm:

- For each Classifier
  - Select a random central data point
  - For all data points
    - compute the distance of the data point from the central data point
  - Stochastically select points for the classifier using distance as the probability measure.
  - Build the Classifier

Classification Algorithm:

- Given a query instance x_q to be classified.
  - Pass the data to all the classifiers.
  - Take a vote of all these classifiers to classify the query instance.

3.4 Algorithm

Table 7 shows the algorithm for building an ensemble of classifiers with my new method. From now on this method will be referred to as the *distance-weighted method* for building ensembles since we are using distance as a metric for sampling data points to build the ensemble.

3.5 Notes

The distance-weighted method for sampling data points for building an ensemble of classifiers is a new method. Its effectiveness will be compared with the two well established techniques for building ensembles. Note that by taking the average of all the attribute values for a data instance when computing the distance, we are reducing the impact of noise on the classifier that is being built. If we have prior knowledge about the attributes, then we can assign weights to each attribute and compute the weighted average. For example, if we have a two dimensional problem space defined

by attributes a and b and if attribute x is more important than attribute y, then we assign more weight to attribute a than attribute b when we compute the distance. In this case we can assign a weight of 2 to attribute a and a weight of 1 to attribute b and use this weighted average for computing the distance between two data points. Generalizing the above equation to a n-dimension problem space we get the following equation

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (W_r * ((a_r(x_i) - a_r(x_j))^2)} \tag{11}$$

If we do not have any prior knowledge, we can assign random weights to each attribute and then take out the weighted averages. The task of selecting a central data point and computing the distance from all other training instances can be carried out in parallel. The number of classifiers in the ensemble will control the speed at which the ensemble is built. But since each classifier and the data set on which it is built is not dependent on other factors, the entire process of sampling data and building an ensemble can be executed in parallel. The classification of an instance x_q should ideally be most similar to the classification of other instances that are nearby in the Euclidean distance. In this aspect the ensemble classifier that we are building mimics instance based learning methods such as the Nearest Neighbor Algorithm [Cover and Hart, 1967].

4 Results

To evaluate the performance of our new method, a number of experiments were carried out. The results were then compared to results using standard Bagging and Boosting.

4.1 Data Sets

The data sets [Murphy and Aha, 1994] for evaluating the performance were drawn from the UCI data set repository. The error rates for each of the three methods of building ensembles is reported along with the standard error rates. These data sets have the following underlying properties: they varied in characteristics and were deemed useful by previous researchers. Table 8 gives the characteristics of the data sets. The data sets that were selected vary in the number of examples, the number of output classes, the number of attribute features, and the type of features in the data set (i.e., continuous, discrete, mix).

Table 8: Summary of the data sets used in this paper. Shown are the number of examples in the data set, the number of output classes, and the number of continuous and discrete input features.

			Features	
Data set	Cases	Class	Continuous	Discrete
breast-cancer-w	699	2	9	-
credit-a	690	2	6	9
credit-g	1000	2	7	13
glass	214	6	9	-
heart-cleveland	303	2	8	5
hypo	3772	5	7	22
ionosphere	351	2	34	-
iris	159	3	4	-
kr-vs-kp	3196	2	-	36
labor	57	2	8	8
sick	3772	2	7	22

4.2 Methodology

Results were accumulated on the various data sets using the 10-fold cross validation approach and are averaged over five runs. For each 10-fold cross validation, the data set is first partitioned into 10 equal-sized sets, then each set is in turn used as the test set while the classifier trains on the other nine sets. For each fold an ensemble of 30 classifiers is built. The results are accumulated for each of the three ensemble building methods.

4.3 Result Error Rates

Table 9 shows test-set error rates for the data sets described in Table 8. Four decision tree methods are being compared in Table 9. The four methods are: a standard decision tree method (without ensembles), a Bagging ensemble, a Boosting ensemble and the method introduced here called the distance-weighted method. The above results were accumulated with 30 classifiers for each ensemble type. One obvious

Table 9: Summary of the results. Shown are the error rates on the test set for single decision tree classifier, a Bagging ensemble of decision trees, a Boosting ensemble of decision trees, and distance-weighted ensemble of decision trees. '*' in the Ensemble column indicates the method producing the lowest error rate for that data set.

		Ensemble Methods		
Data set	Single DT	Bagging	Boosting	Distance-Weighted
breast-cancer-w	5.3	4.2	3.8	3.5*
credit-a	22.3	18.3*	18.6	18.4
credit-g	35.29	27.3	26.1*	26.3
glass	35.05	34.3	33.7	33.2*
cleveland-heart	31.3	21.8*	24.4	24.4
hypo	2.5	0.4*	0.6	0.9
ionosphere	13.8	9.1	8.5*	9.7
iris	5.8	5.1*	5.4	5.3
kr-vs-kp	2.5	0.6	0.6	0.5*
labor	22.64	13.2*	14.7	15.1
sick	1.7	1.4*	1.4*	1.6

conclusion drawn from these results is that each ensemble method appears to reduce the error rate for almost all of the data sets., and in many cases the reduction is large. One thing that is very clear from the results is that the new method to build ensembles is better than the single decision tree classifier. Out of the eleven data sets the new method performs better than Bagging on four data sets and it performs better than Boosting on five data sets. From the results we can conclude that the performance of the new ensemble method is comparable to Bagging and Boosting. Although the results are varied, this new technique produces comparable results with the other two approaches. This new method is also potentially faster than Boosting as we can execute the process of building ensembles in parallel and we do not appear to give away on accuracy.

4.4 Results By the Number of Classifiers in the Ensemble

The Figures 8-18 plot the error rates on the test set for the three ensemble building methods on various data sets and gives a comparison of the three methods. These results display the number of classifiers on the X-axis and the error rate on the y-axis.

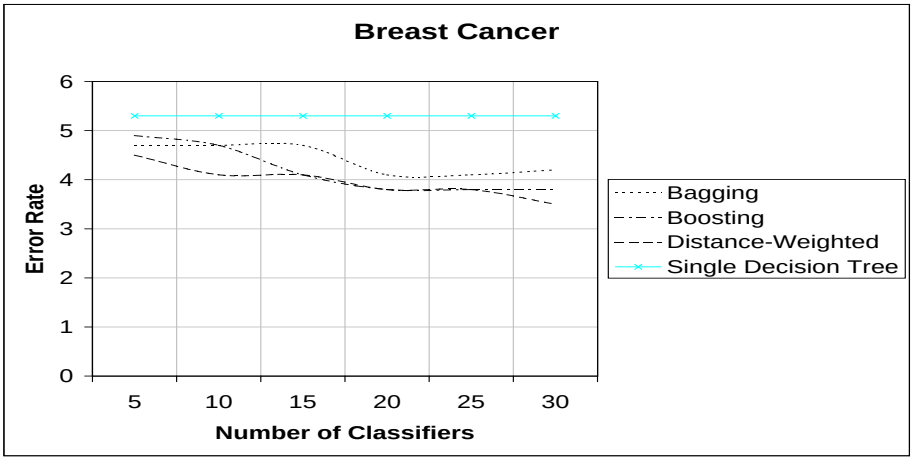


Figure 8: Bagging, Boosting, Distance-Weighted test set error rates for the breast-cancer data set as the number of classifiers in the ensemble increases.The test set error rate for a single decision tree classifier is shown as a straight line across the x-axis for comparison purposes.

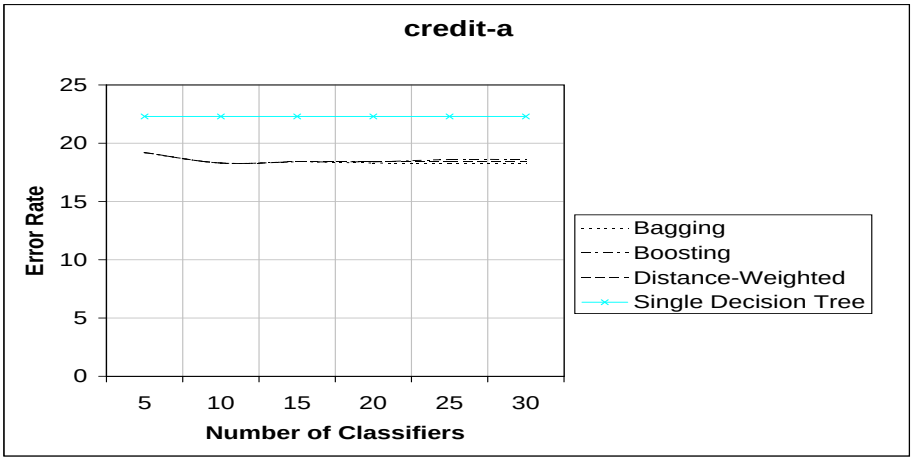


Figure 9: Bagging, Boosting, and Distance-Weighted test set error rates for the credit - a data set as the number of classifiers in the ensemble increases. The test set error rate for a single decision tree classifier is shown as a straight line across the x-axis for comparison purposes.

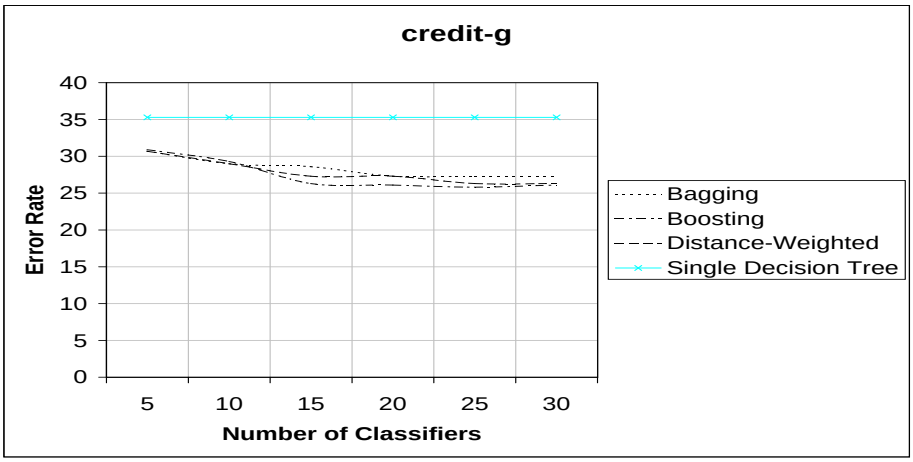


Figure 10: Bagging, Boosting, and Distance-Weighted test set error rates for the credit - g data set as the number of classifiers in the ensemble increases. The test set error rate for a single decision tree classifier is shown as a straight line across the x-axis for comparison purposes.

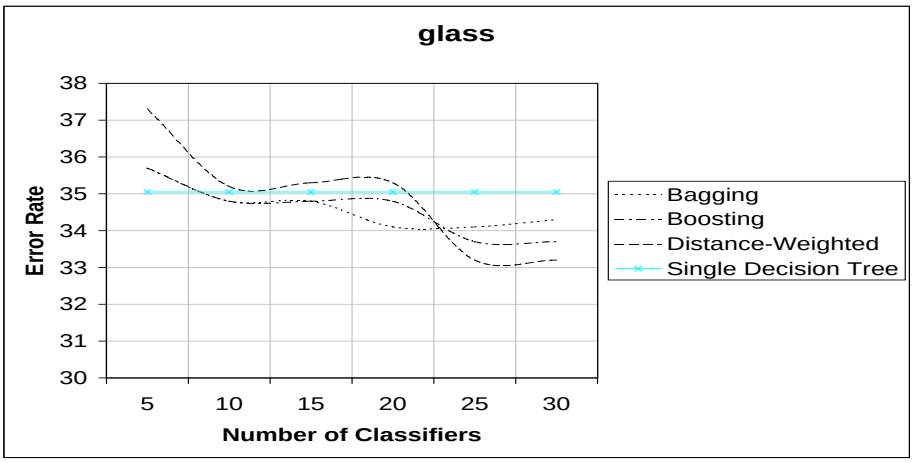


Figure 11: Bagging, Boosting, and Distance-Weighted test set error rates for the glass data set as the number of classifiers in the ensemble increases. The test set error rate for a single decision tree classifier is shown as a straight line across the x-axis for comparison purposes.

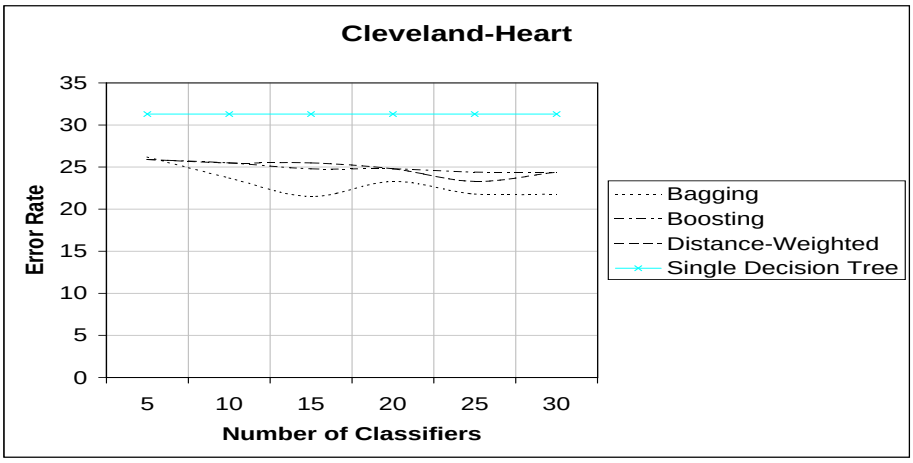


Figure 12: Bagging, Boosting, and Distance-Weighted test set error rates for the cleveland-heart data set as the number of classifiers in the ensemble increases. The test set error rate for a single decision tree classifier is shown as a straight line across the x-axis for comparison purposes.

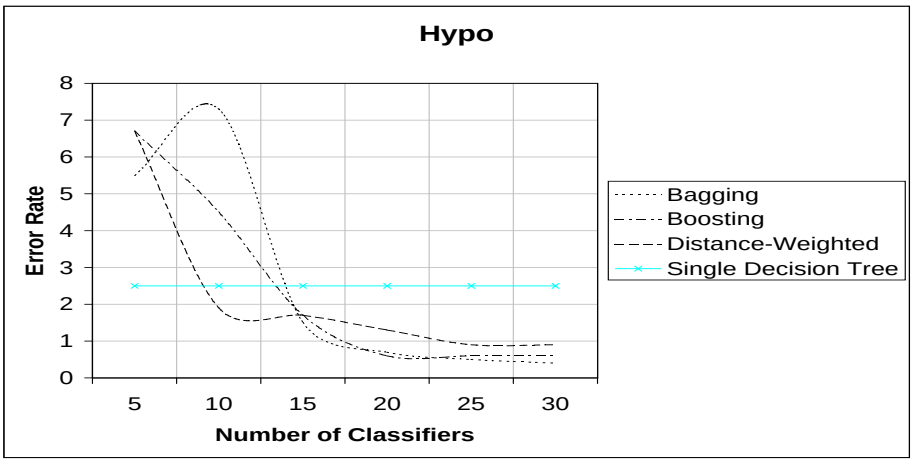


Figure 13: Bagging, Boosting, and Distance-Weighted test set error rates for the hypo data set as the number of classifiers in the ensemble increases. The test set error rate for a single decision tree classifier is shown as a straight line across the x-axis for comparison purposes.

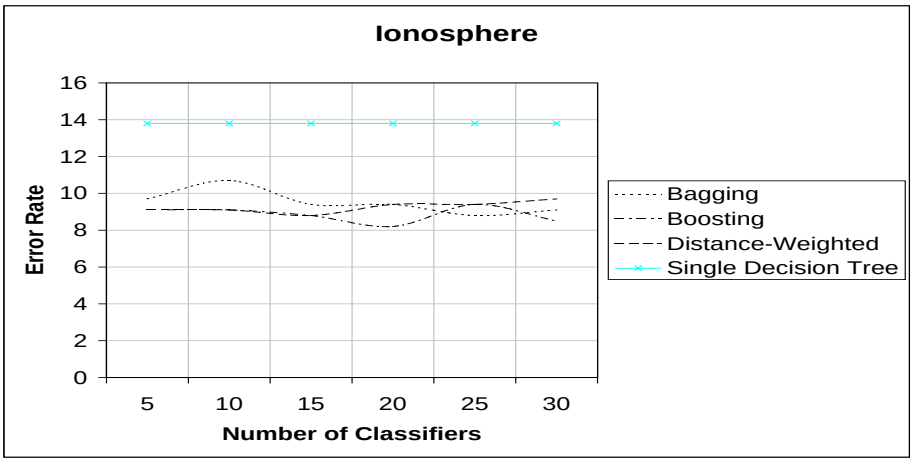


Figure 14: Bagging, Boosting, and Distance-Weighted test set error rates for the ionosphere data set as the number of classifiers in the ensemble increases. The test set error rate for a single decision tree classifier is shown as a straight line across the x-axis for comparison purposes.

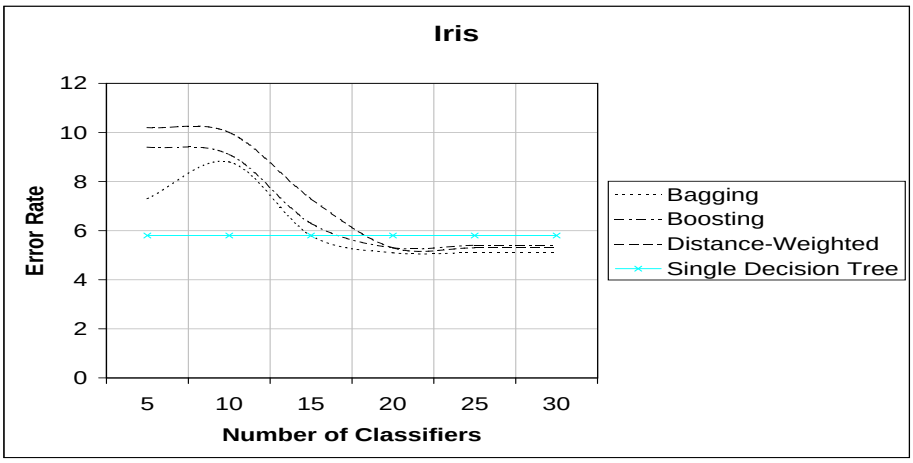


Figure 15: Bagging, Boosting, and Distance-Weighted test set error rates for the iris data set as the number of classifiers in the ensemble increases. The test set error rate for a single decision tree classifier is shown as a straight line across the x-axis for comparison purposes.

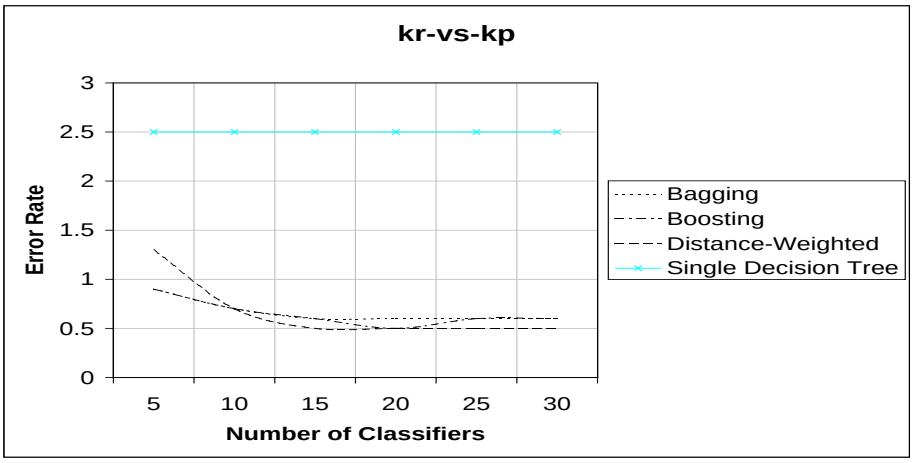


Figure 16: Bagging, Boosting, and Distance-Weighted test set error rates for the kr-vs-kp data set as the number of classifiers in the ensemble increases. The test set error rate for a single decision tree classifier is shown as a straight line across the x-axis for comparison purposes.

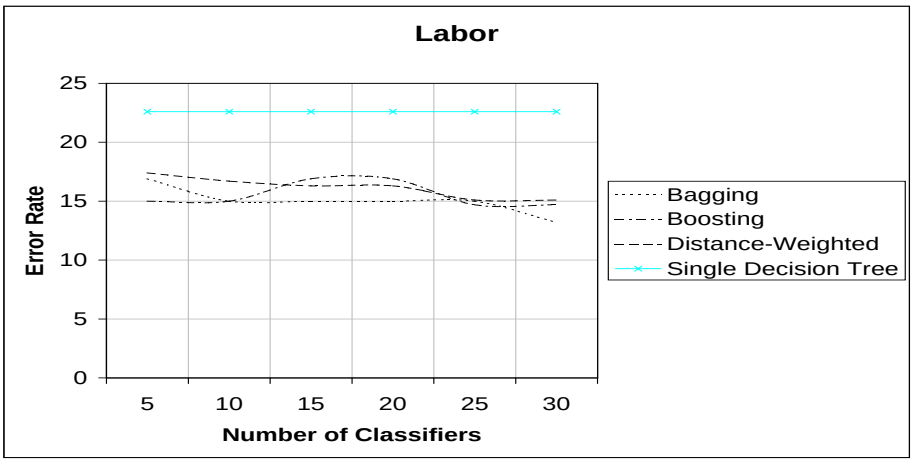


Figure 17: Bagging, Boosting, and Distance-Weighted test set error rates for the labor data set as the number of classifiers in the ensemble increases. The test set error rate for a single decision tree classifier is shown as a straight line across the x-axis for comparison purposes.

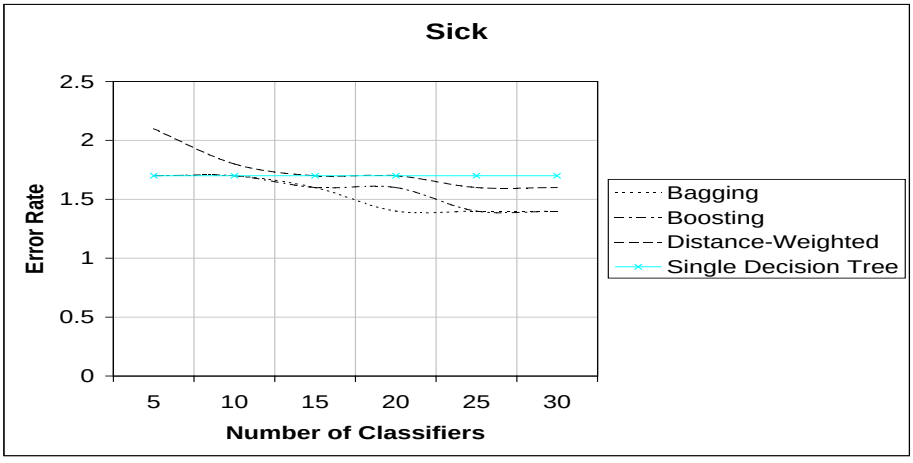


Figure 18: Bagging, Boosting, and Distance-Weighted test set error rates for the sick data set as the number of classifiers in the ensemble increases. The test set error rate for a single decision tree classifier is shown as a straight line across the x-axis for comparison purposes.

4.5 Discussion

The graphs in Figures 8-18 suggest that each of the three ensemble building mechanism outperforms the single decision tree classifier. Much of the reduction in error for the three ensemble building mechanisms appears to have occurred after adding 15 classifiers to the ensemble. For each of the three ensemble building technique we nearly get a straight line after adding 20 classifiers. The graphs also indicate that after about 25 classifiers in the ensemble there is no significant reduction in the test set error rate. This proves that we get fairly accurate learners with 25 classifiers in each of the three ensemble building methods. All the three methods for building ensembles produces similar shaped curves after adding 15 classifiers to the ensemble. The new distance-weighted method for building ensembles outperforms the Boosting method on 5 of the 11 data sets. It performs better than Bagging on 4 data sets. On the remaining data sets the results of the new method is comparable to Bagging and Boosting.

5 Conclusions

In this thesis I have implemented a new ensemble building technique, which I call the distance-weighted method and compared it with the two popular techniques of building ensembles (i.e., Bagging and Boosting). I compare the performance of the three ensemble building techniques and the single decision tree classifier. The new technique uses C4.5 decision trees as component classifiers. In the preliminary tests, I first determined the baseline results with which to compare the new technique for building ensembles. The baseline results included tests from Bagging, Boosting and the standard method for building single decision trees from the data set. All the decision trees were built using the 10-fold cross validation approach and the results were averaged over 5 execution cycles. The new method for building ensembles outperforms Bagging on four data sets and is better than Boosting on five data sets, and the results over the other data sets are comparable.
Based on the experiments that we performed on our implementations, we can answer the questions posed in Chapter 1.

- *Q*uestion 1: How does the new ensemble building method compare to Bagging and Boosting in terms of test set error rates?
  From the results we can conclude that the new distance-weighted ensemble building technique is generally as good as Boosting. This technique for building ensembles gives us better results than Bagging on four data sets and proves to be better than Boosting on five data sets. The results over the other data sets are comparable to the results produced by Boosting and Bagging.

- *Q*uestion 2: What is the optimal size of the ensemble (i.e., the number of classifiers making up the ensemble)?
  For each of the three ensemble building mechanisms, most of the reduction in error appears to have occurred after adding 15 classifiers to the ensemble. For each of the three ensemble building technique we nearly get a straight line after adding 20 classifiers. The results indicate that after about 25-classifiers in the ensemble there is no significant reduction in the test set error rate. This proves that we get fairly accurate learners with 25 classifiers in each of the three ensemble building methods.

- *Q*uestion 3: Is the new method faster than the Boosting?
  Boosting ensembles cannot be built in parallel as we need to take into account the error from the previous classifier in order to sample data for the next classifier. The distance-weighted method for building ensembles does not take into

account the errors from the previous classifier. Instead it randomly selects data points that act as central data points and stochastically selects points which are closer to the central data point to train the individual classifiers that make up the ensemble. The process of selecting data points for each ensemble and building the ensemble can be carried out in parallel and hence the new distance-weighted method is potentially faster than the Boosting method.

The distance-weighted method for building ensembles is unique as compared to the earlier approaches for building ensembles. The data sampling procedure for the distance-weighted method makes "intelligent" choices for which data points for building the individual classifiers by considering only data points that are near by in the Euclidean distance. Hence the distance-weighting method can be considered as an intelligent Bagging technique. Using this method we are building piece-wise classifiers to approximate the global target function. Ideally the classification of an instance will be most similar to the classification of other instances that are close to it in the problem space. In this aspect the ensemble classifier that we are building mimics the K-Nearest Neighbor algorithm.

5.1 Future Work

Our results suggest that the new distance-weighted method of building ensembles is almost equivalent to the Boosting technique. However much works needs to be done in the following areas before we can be sure whether this new method is better than Boosting:

- **Noise:** The impact of noisy data points (in the training set) on the ensemble needs to be studied. The extent to which the ensemble overfits the training set as the number of classifiers increases needs a more thorough investigation.

- **Other distance computing metrics:** In this thesis work we have built classifiers by using the Euclidean distance as our metric. There are other distance computing functions such as Mahalanobis distance, Kullback-Leibler distance, etc. The performance of the system with other distance computing functions needs to be studied.

- **Incorporating attribute priority:** Not all the attributes in the data set have the same priority. Some of the attributes will be more important to the target concept than the rest. Hence we need to assign more weights to such attributes when we compute the distance between two data points. If we have

prior knowledge of the importance of certain attribute features, we can weigh them accordingly.

- **Using feature selection:** In many applications, the size of a dataset is so large that learning might not work as well before removing the unwanted features. Reducing the number of irrelevant features before we sample data drastically reduces the running time of a learning algorithm and yields a more general concept. This helps in getting better insight into the underlying concept. Feature selection methods try to pick a subset of features that are relevant to the target concept.

- **Weighing of individual classifiers:** Since the classification of an instance will ideally be most similar to the classification of other instances that are close to it, we need to assign more weight to those classifiers whose central data points are close to the query instance. We need to come up with a effective weighing scheme so that classifiers whose central data point is closer to the query instance are assigned more weight than classifiers whose central data point is further away from the query instance.

- **Picking region centers in poorly performing areas of problem space:** Sometimes it might happen that the distance-weighted ensemble building technique does not perform well on certain regions in the problem space. In such situations we would like to pick out those regions and build local classifiers and add them to the ensemble to improve the overall accuracy of the system.

References

[Agarwal and Shim, 1996] Agarwal, R. A. and Shim, K. (1996). Developing tightly-coupled data mining applications on a relational database system. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining(KDD–96)*, pages 287–290, Menlo Park, CA.

[Aha et al., 1991] Aha, D., Kibler, D., and Albert, M. (1991). Instance-based learning algorithms. *Machine Learning, 6*, pages 37–66.

[Atkeson et al., 1997] Atkeson, C., Moore, A., and Schaal, A. (1997). Locally weighted training. *Artificial Intelligence Review.*

[Bauer and Kohavi, 1999] Bauer, E. and Kohavi, R. (1999). An empirical comparision of voting classification algorithms: Bagging, Boosting a and variants. pages 36, 105–139.

[Breiman, 1996a] Breiman, L. (1996a). Bagging predictors. *Machine Learning*, pages 123–140.

[Breiman, 1996b] Breiman, L. (1996b). *Bias, variance and arcing classifiers.* UC-Berkeley, Berkeley, CA.

[Breiman et al., 1984] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, P. J. (1984). *Classification and Regression Trees.* Wadsworth International Group, CA.

[Chauvin and Rumelhart, 1995] Chauvin, Y. and Rumelhart, D. (1995). *Backpropagation: Theory, architectures, and applications.* Lawrence Erlbaum Assoc., Hillsdale NJ.

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, pages 21–27.

[Efron and Tibshirani, 1993] Efron, B. and Tibshirani, R. (1993). *An Introduction to Bootstrap.* Chapman and Hall, New York.

[Fayyad et al., 1995] Fayyad, U., Smyth, P., Weir, N., and Djorgovski, S. (1995). Automated analysis and exploration of image databases: Results,progress, and challenges. *Journal of Intelligent Information Systems*, pages 1–19.

[Freund and Schapire, 1996] Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–156, Bari, Italy.

[Jensen, 1996] Jensen, F. (1996). *An Introduction to Bayesian Networks.* New York: Springer Verlag.

[Kohavi and Wolpert, 1996] Kohavi, R. and Wolpert, D. (1996). Bias plus variance decomposition for zero-one loss functions. *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 275–283.

[Kononenko et al., 1984] Kononenko, I., Bratko, I., and Roskar, E. (1984). Experiments in automatic learning of medical diagnostic rules. (Technical Report), Jozef Stefan Institute, Ljubjjana. Yugoslavia.

[Maclin and Opitz, 1997] Maclin, R. and Opitz, D. (1997). An emprical evaluation of Bagging and Boosting. *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 546–551.

[Mitchell and Thrun, 1993] Mitchell, T. and Thrun, S. (1993). Explanation-based neural network learning for robot control. *Advances in neural information processing systems*, pages 287–294.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning*, chapter Decision Tree Learning. WCB/McGraw-Hill.

[Murphy and Aha, 1994] Murphy, P. and Aha, D. (1994). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[Pomerleau, 1989] Pomerleau, D. (1989). Alvinn: An autonomous land vehicle in a neural network. In *Tech. rep. CMU-CS-89-107.* Pittsburg, PA: Carnegie Mellon University.

[Provost and Kolluri, 1999] Provost, F. and Kolluri, V. (1999). A Survey of Methods for Scaling Up Inductive Algorithms. *Knowledge Discovery and Data Mining(1999).*

[Quinlan, 1990] Quinlan, J. R. (1990). Induction of Decision Trees. In *Readings in Machine Learning.* Morgan Kaufmann. Originally published in *Machine Learning* 1:81–106, 1986.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Mogran Kaufmann.

[Stolfo et al., 1997] Stolfo, S., W.Fan, D., Lee, W., Prodromidis, A., and Chan, P. (1997). Credit Card Fraud Detection Using Meta Learning: Issues and Initial Results. Issues and initial results. Working notes of AAAI Workshop on AI Approaches to Fraud Detection and Risk Management 1997.

[Tesauro, 1995] Tesauro, G. (1995). Temporal difference learning and td-gammon. *Communications of the ACM*, pages 58–68.

[Weiss and Indurkhya, 1997] Weiss, S. and Indurkhya, N. (1997). *Predictive Data Mining: A Practical Guide.* Morgan Kaufmann.