UNIVERSITY OF MINNESOTA

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

Hariprasad Bommaganti

and 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. Richard Maclin

Name of Faculty Adviser

Signature of Faculty Advisor

Date

GRADUATE SCHOOL

Acknowledgments

I am grateful to my advisor Dr. Richard Maclin for providing me an opportunity to work with him and the valuable guidance that he provided. I would also like to thank Dr. Ted Pedersen and Dr. Robert McFarland for their cooperation.

I am indebted to my parents, brother and friends, without their support nothing would have been possible.

Abstract

In inductive learning, a machine learning algorithm attempts to learn a model of a concept given examples of that concept. The examples are described by a set of features that may or may not be relevant to the concept being learned. Feature subset selection deals with the identification of those features that are relevant to the learning process. Good feature subset selection increases the performance of the learned concept and also reduces the time required for learning. Some of the existing techniques for feature subset selection include embedded techniques, filter techniques and wrappers.

This work investigates the use of ensembles in feature subset selection. An ensemble uses multiple component classifiers to represent a model. We present a novel feature selection method we call feature boosting that generates a weighted distribution of features. Tests with this method suggest that it is very effective. In many cases, the accuracy of such an ensemble was comparable to one that used all features. We further compare the performance of this new feature boosting methodology with some simpler techniques. Our results indicate that feature subset selection using ensembles is a useful idea and has potential for further research.

Contents

List of Figures

List of Tables

1 Introduction

Plurality should not be posited without necessity.

- William of Ockham (1284 - 1347)

In inductive learning, a machine learning algorithm attempts to learn a model of a concept given examples of that concept. For example, an algorithm might be provided with the information about a set of patients in a hospital (e.g., information from the doctor's interview of the patient, results from lab tests, etc.). The algorithm would then be told which of the patients have a particular cancer and which do not. Based on the information about the patients, the machine learning algorithm constructs a model trying to identify which information is predictive of cancer. One of the key problems for this task is to determine which information (which features) describing the patient are important and which are not.

As the number of features describing a patient increases, the amount of data needed to differentiate between useful and useless features increases. Thus, developing methods that can be used to find or select an appropriate subset of features from a larger set of features is very important. This problem is called the *feature subset selection* problem.

A simple learning process might involve presenting the learning algorithm with a set of examples, each of which contain a set of (input) feature values and a corresponding task. This provides the *experience* to the learner. Later, the learner uses this experience to perform the required task. Its performance determines the effectiveness in learning the task.

Consider a learner that is trying to acquire the skill of predicting the next day's weather based upon today's climatic conditions. In this case, the set of feature values would be the climatic conditions recorded on a particular day in the past. They might contain that day's high and low temperatures, humidity, rainfall, etc. and the prediction value would be the corresponding weather for the next day. We want learners that are effective for future examples. Feature subset selection involves choosing a subset of the input features and

training the learner using only those features while still achieving acceptable accuracy for the learner. This results in a reduction of the computational time needed by the learner.

1.1 Feature Subset Selection

There are various techniques for feature subset selection. They can be broadly classified into three categories [Blum and Langley, 1997]: *embedded selection techniques, filter techniques* and *wrapper techniques*. A detailed discussion of the above feature selection techniques is given in Chapter 2.

1.1.1 Embedded Selection Techniques

In the case of embedded feature selection, feature subset selection is a part of the induction algorithm itself. An induction algorithm is a learning algorithm used to capture the concept behind the examples. Most of the commonly known induction algorithms (ID3 [Quinlan, 1990], Version Spaces [Mitchell, 1977], CART [Breiman et al., 1984], etc.) come under this category. These algorithms require lots of data to effectively select an appropriate subset.

1.1.2 Filter Techniques

In filter techniques [Kira and Rendell, 1992, Almuallim and Dietterich, 1991], subset selection is a process that is applied prior to induction. The selected subset serves as an input to the induction algorithm. This subset is used by the algorithm during the training process.

1.1.3 Wrapper Techniques

Wrapper techniques [Kohavi and John, 1997] search for an *optimal* subset of features by starting with an empty set and adding or removing a feature from the current set. Each addition or deletion is governed by the performance of the induction algorithm using the present subset of features. The induction algorithm is a part of the feature search engine itself.

1.2 Our Approach

We use an ensemble learning method to carry out the process of feature subset selection. We use C4.5 as our basic decision tree model. Bagging and boosting are two popular ensemble techniques that are known to work very effectively with decision trees [Maclin and Opitz, 1997]. We use these techniques as a part of our system. We use a bagging ensemble to evaluate our feature selection technique. We evaluate four different strategies for feature subset selection. We briefly describe them below:

- *Random Subset Selection*: We randomly pick a subset of attributes from the original set of input attributes. Every feature has equal probability in being selected.

- *Information-Gain Based Selection*: In this approach, we find the information gain for each of the attributes in the training dataset. The features with the best information gain are selected to form the subset.

- *Cumulative Information-Gain Based Selection*: In this approach, we consider the fact that an attribute that splits the whole of the training set very well might not split a part of the dataset effectively. We select an attribute at each level of the tree using a new metric called *cumulative information gain.*

- *Feature Boosting*: In this approach, we use a technique similar to boosting, where we alter the distribution of the weights for each feature depending upon the performance of a classifier that used it. The feature weight distributions are changed over several iterations by building a classifier in each iteration and using its accuracy to update the weights.

1.3 Thesis Statement

This thesis evaluates the use of ensembles for feature subset selection. The weight distribution of the features which is the output of the feature subset selector will be used to train a bagging ensemble. Our hypothesis is that the feature boosting technique that encompasses

a boosting classifier would produce good feature subsets for the individual classifiers of the bagging ensemble.

The accuracy of ensembles using the feature subsets (generated using feature boosting) will be compared with the baseline versions including a C4.5 decision tree classifier and bagging ensemble that uses all the features to verify the above hypothesis.
The thesis will attempt to answer the following questions:

1. *Can the ensemble methodology be exploited to do feature subset selection?*

2. *How well does the feature selection algorithm perform?* We compare its accuracies with the baselines including a single C4.5 decision tree algorithm and a bagging ensemble that uses all the attribute information.

3. *Are there some simple techniques to achieve good feature subset selection for ensemble learning?* This question will be answered by the development and comparison of accuracies of other simpler techniques (like random subset selection, information-gain based approaches) to the feature boosting methodology.

1.4 Outline of Thesis

In the next chapter, we present background on feature subset selection, decision trees and ensemble learners and the new ensemble based feature subset selection approach. We then discuss the results of our experiments followed by concluding remarks and attempt to answer the questions that we have proposed above. A brief survey of the chapters follows:

- Chapter 2 lays out the basic framework with details about determining the relevance of a feature, some feature subset selection approaches, ID3 and C4.5 decision tree building algorithms, ensemble learning and popular ensemble learning techniques.

- Chapter 3 describes our new approach for feature subset selection using ensembles. It further describes other ensemble approaches that we used as baselines.

- Chapter 4 elaborates on the experiments conducted to compare the new feature subset selection algorithm. We conclude the chapter with a discussion on the results obtained

for the various test datasets and also compare the accuracies obtained for baseline algorithms and the other relatively simple feature subset selection strategies.

- Chapter 5 presents conclusions for the thesis based upon the results and also answers the questions that we posed above. We end this thesis with some discussion of possible directions for future research.

2 Background

In this chapter, we give an introduction to machine learning and the various relevant underlying concepts. We further discuss the feature subset selection problem and introduce the existing approaches to feature subset selection. We give a description of ID3 and C4.5 decision tree models that we have used in our work. Finally, we provide an introduction to ensemble learning with an insight into two popular ensemble learning techniques - Bagging and Boosting.

2.1 Machine Learning

Learning can be defined as the acquisition of knowledge or skill through experience. The experience could come from teacher or from self-study. A more concrete definition of learning which is widely accepted by most psychologists is:

> "Learning is a process in which behavior capabilities are changed as the result of experience, provided the change cannot be accounted for by native response tendencies, maturation, or temporary states of the organism due to fatigue, drugs, or other temporary factors" [Runyon, 1977].

The simplest form of learning is the association of events. For example, a normal person, who gets stung by bees when he goes near a bee-hive with honey in his hands, will take precautionary measures before he goes near the bee-hive again. The measures he takes to avoid such an accident may vary. He might go there without the honey or wear protective garments and still take the honey or he might avoid going near it. The most important observation is the association of the event of being stung by the bees with the two events of taking honey and going near the bee-hive. This association is the *knowledge* acquired by the individual through his rough *experience* with bees, which might motivate him to *act* more wisely in future.

Machine Learning refers to the study of computer programs that *learn* automatically through experience [Mitchell, 1997]. Learning in this context refers to changes to the

system (brought about during the process of learning a particular task T) that enable the system to perform the task T or similar tasks more effectively in the future.

2.1.1 Concept Learning and Classifiers

Learning is a process of *identifying* the underlying concept. More often than not, a concept can be viewed as a boolean-valued function. Consider an example of automatically recognizing a handwritten character (say an A) after gaining *experience* through a set of training examples which are a set of pre-labeled handwritten character samples. The labels simply specify whether the sample is an A or not. The corresponding boolean-valued function is whether a particular example is an A or not. Mathematically, it could be done by building a hyper-plane in the sample of input space that splits the examples into two classes - A and *not* A. The learned hyper-plane can then be used to classify new examples (to mark them as As or not). The process of extending limited information provided by some set of examples to the whole sample space (comprised of the training examples and the new examples) is *Concept Learning* or *Inductive Learning*. Concept learning is an important and diverse area of interest in machine learning. It involves the development of algorithms that can classify future (test) instances based on the concept learned from the existing training examples. A model capable of doing such classification is called a *classifier*.

For example, a learner may want to learn when and what type of contact lenses to prescribe given various characteristics of a patient. There are 2 types of lenses: *hard* and *soft*. Further, the learner may also declare that the individual need not wear any glasses depending upon the various features of the individual. We list below the information that the learner has available about patients.

- **Age**: Describes the age of the patient in discrete terms.
  Possible Values: *young, pre-presbyopic, presbyopic*.

- **Prescription**: Describes the kind of prescription received by the patient.
  Possible Values: *myope, hypermetrope*.

- **Astigmatic**: Indicates if the patient suffers from astigmatism.

  Possible Values: *no, yes.*

- **Tears**: Indicates if the tear glands of the patient work normally or not.

  Possible Values: *reduced, normal.*

- **Lens Type**: Indicates the type of lens that the individual would be prescribed depending on the values for the input features.

  Possible Values: *hard, soft, none.*

Each of the above characteristics takes certain possible values (as shown). In machine learning terminology, each such characteristic is known as *a feature* or *attribute*. Of the above specified five characteristics, the first four characteristics comprise the information that the learner needs to know in order to prescribe the correct pair of lenses. These four characteristics taken together are the *input feature set*. Further, there are features (called *output features*), whose values are associated with the different combinations of the above mentioned input features. The feature *Lens Type* is an output feature in the above example. Examples are presented to the concept learner during the training phase. The output values for the examples are generally provided by a human expert, in this case, perhaps by an ophthalmologist. The learner uses the mappings between the input and output feature values to generate a concept. After the training process, given a test case (which is a combination of input feature values), the concept learner should be able to predict the correct value for the output feature.

Before we move onto a discussion on feature subset selection techniques, let us derive more formally a relationship between the input features, the output feature(s) and the concept learning.

Assume that we have n input features.

Let X_i be the set of possible feature values for input feature i.

The complete input space is a cross-product:

$$I \quad = \quad (X_1 \times X_2 \times ... \times X_n)$$

Let Y represent the set of possible values for the output feature.

Note: *Actually, we could have more than one output feature, in which case Y would be a cross-product like I.*

An example e can be represented by a tuple $(\vec{x}, y)$ where,

$\vec{x} = (x_1, x_2, ..., x_n)$ such that $x_i \in X_i$ and y is an output feature value ($y \in Y$).

A dataset D is a set of examples drawn from the possible distribution of input vectors and output values. Therefore, we can say that

$$D \quad = \quad \{(\vec{x}, y), \ \ldots\}$$

Our goal in concept learning is to find a hypothesis h such that

$$\forall \vec{x} \in I, \quad h(\vec{x}) = y \tag{1}$$

In the above equation, we require a perfect hypothesis for the input space I (in the sense that every data point must be perfectly classified). This is almost always impossible to achieve. Thus, we modify our goal by defining an error function $\epsilon(\vec{x})$ such that

$$\epsilon(\vec{x}) \quad = \quad \begin{cases} 1 & \text{if } h(\vec{x}) \neq y \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

For some test data, we try to find a hypothesis h that minimizes $\sum_{\forall \vec{x} \in I} \epsilon(\vec{x})$.

A good concept learning algorithm should tend towards classifying the test instances with a minimal error. This minimization in a learning algorithm is called the ability to *generalize* the concept (that was learned) using the given set of examples. In the case of a feature subset selection algorithm, we add a new constraint of finding a subset of the input features that still builds a good concept learner. We now turn to a discussion of the actual problem of feature subset selection and some of the available solutions.

2.2 Feature Subset Selection

Kohavi and John [1997] define the problem of feature subset selection to be that of finding a subset of the original set of features of a dataset, such that the induction algorithm that is run on the data containing only the features from the subset generates a classifier with the highest possible accuracy.

To generate a useful subset, we need to identify a few features as *important* from the set of input features in order to form a subset of features that still achieves high accuracy. Other features may be discarded as their information adds to the complexity of learning the associated concept without increasing accuracy. In theory, the former type of feature is termed a *relevant feature* while the latter is termed an *irrelevant feature*. We now briefly discuss some of the popular definitions associated with the terms *relevance* and *irrelevance*.

2.2.1 Relevance and Irrelevance

The simplest definition of relevance is as follows:

> *A feature* **f** *is* relevant *to the target concept* **c** *if there exist examples A and B in the instance space such that they differ only in their values for this feature and the output feature values.* [Blum and Langley, 1997]

This definition is simple, but we may not be able to determine whether a given feature is *relevant* from the given small (sub)set of the possible examples. For example, imagine a dataset where every feature appears twice. Then, according to the above definition, each feature would become *irrelevant* as no single pair of examples would have differing values for exactly one feature. Kohavi and John [1997] introduce a useful notion of strong and weak relevance.

Strong Relevance

A feature f is said to be strongly relevant to the *sample* S if there exist examples A and B in the sample space such that they differ only in their values for this feature and the output feature values. A slight variation to the above definition can be obtained by associating a

non-zero probability over the distribution $\mathcal{D}$ of the examples and not assuming the existence of the examples A and B in the sample space. If A and B have such a non-zero probability and they differ only in their values for this feature and the output feature values then the feature can be deemed relevant.

Weak Relevance

A feature f is weakly relevant to the sample S if it is possible to remove a subset of features so that f becomes strongly relevant. This can be important in subset selection since if the subset of features whose removal makes the feature f strongly relevant were to be actually removed from the feature set during subset selection, then we surely do not want to *remove* this feature.

A feature that falls into one of the above categories (strong or weak) is considered relevant and others are considered irrelevant features.

Most concept learning algorithms tend to capture the concept poorly (and hence generalize poorly) in the presence of many irrelevant features in the input data. Irrelevant features also add to the complexity of the input space, which increases the computation time for learning. We now look at the various popular feature subset selection techniques.

2.2.2 Feature Subset Selection Techniques

Current feature subset selection techniques can be broadly classified [Blum and Langley, 1997] into three categories:

- Embedded Selection Techniques - These are induction algorithms that involve feature subset selection as a part of the induction algorithm itself. Many induction algorithms fall into this category.

- Filter Techniques - These approaches use a feature selection algorithm to select the subset which is then passed on to the induction algorithm.

- Wrapper Techniques - The feature subset selection algorithm forms a wrapper on top of the induction algorithm. This is a search strategy to find an *optimal* subset of

features by adding or deleting features from the input feature set depending on the accuracy of the induction algorithm itself.

2.2.2.1 Embedded Selection Techniques

Embedded selection algorithms include induction algorithms that implicitly carry out feature subset selection. Methods that induce logical descriptions are a simple form of such selection. These methods search for a subset of features in the feature space by adding or removing a feature from the generated set, so that the prediction errors are lower for newer instances. An example of such an approach is Version Spaces [Mitchell, 1977].

Decision tree algorithms such as ID3 [Quinlan, 1990], CART [Breiman et al., 1984], etc. are another form of embedded techniques. But in this case, the features forming the subset have complex relationships among themselves. The complexity arises due to the recursive partitioning approach taken by the induction algorithm. At each level of recursion, the existing partition is split using a feature f where f is supposed to be the best possible attribute to split the partition. Information Gain values (defined later) are used to decide which feature is selected. In the later sections of this chapter, we review two popular decision tree models, ID3 and its successor C4.5.

We tend not to use such techniques to do feature selection because we need large amounts of data to search the complete feature space for an optimal feature subset. Further, interactions among the relevant features, which can make a relevant feature in isolation *seem* irrelevant, might also lead to poor generalization.

2.2.2.2 Filter Techniques

In filter techniques, the feature subset selection and the induction are two different processes. The feature subset selection process is carried out before the induction. The output of the feature subset selection process serves as one of the inputs for induction. Feature subset selection acts as a preprocessing step that uses certain characteristics of the training set to generate the relevant subset. This technique (preprocessing step) can be used in combination with any of the induction techniques.

A very simple filter approach would be to select k attributes that have the highest correlation with the output (target) class. One metric that would measure such correlation would be the mutual information between the corresponding input feature and the output feature. The RELIEF algorithm [Kira and Rendell, 1992] uses such a mechanism with added complexity to the feature evaluation function.

FOCUS [Almuallim and Dietterich, 1991], another filter based approach, is an algorithm that searches for the smallest possible subset of features that will completely split up the training set without any error. This is achieved by searching in the feature space starting with an empty set of features. It starts by looking at a single feature (in isolation), and then looking at pairs and so on. The search stops when it finds a (sub)set that perfectly splits the training data with respect to their output class values.

A disadvantage of filter techniques is that the feature subset selection generally does not take the induction algorithm (that will use this feature subset) into consideration.

2.2.2.3 Wrapper Techniques

The wrapper approach [Kohavi and John, 1997] also searches the feature space for the optimal subset by starting with an empty set of selected features. But in this case, the evaluation measure for a particular subset is the estimated accuracy of the classifier built using an induction algorithm. The induction algorithm is wrapped inside the feature search engine and hence, they are called wrapper mechanisms. Figure 1 illustrates the wrapper approach to feature subset selection. The induction algorithm acts as a black box. The induction algorithm is run on the dataset using different feature subsets, the feature subset with the highest evaluation is selected as the final feature subset. The dataset is initially split into a training set and a test set. The training set is used by the feature subset selector in searching for the *optimal* subset. The training set is again partitioned in the feature subset selector so that it can estimate the performance of a particular subset of features. Kohavi and John [1997] discuss many different strategies of selecting or deleting features in the process of subset selection.

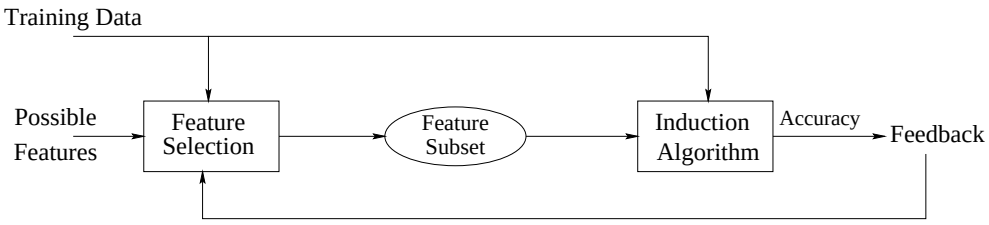


Figure 1: The wrapper approach to feature subset selection. The feature selection component searches for the various subsets in the feature subset space. It uses the induction algorithm to obtain a performance estimate. Feature evaluation evaluates each particular subset.

The main idea behind the use of the induction algorithm itself in the evaluation of a subset of features is that the induction algorithm should eventually use a subset that would give a better estimate than any other metric that might have an entirely different inductive bias. A major disadvantage associated with the wrapper mechanism is the computational cost involved. Faster evaluation techniques have been evolved to reduce the computation. Caruana and Freitag [1994] evolved a technique where the decision trees are cached for later use in the process of search.

We now look at the basic induction algorithm (Decision Trees) that was used in this thesis work.

2.3 Decision Trees

Decision trees are one of the most popular methods used for inductive inference. They are a method for approximating discrete-valued functions which are functions whose range is a finite set of values. They are robust for noisy data (data that is inconsistent to the underlying concept) and capable of learning disjunctive expressions. They have been applied to a variety of problems ranging from classification of celestial objects in satellite images to medical diagnosis and credit risk assessment.

2.3.1 Tree Representation

A decision tree is a k-ary tree where each of the internal nodes specifies a test on some attribute from the input feature set used to represent the data. Each branch descending from a node corresponds to one of the possible values of the feature specified at that node. An instance is classified by recursively testing (starting from the root) the attribute value of the instance for the attribute specified by that node and then moving down the corresponding branch until a leaf node is found. A classification label, which is one of the possible values of the output feature, is associated with every leaf in the tree and every test instance receives the label of the corresponding leaf.

In our decision tree explorations, we will continue to use the "lenses" dataset described in Section 2.1.1. This dataset was obtained from the UCI repository [Blake and Merz, 1998]. Table 1 shows the training instances used in our discussion.

Figure 2 depicts a decision tree formed using the data in Table 1. There are three internal nodes that use tears, astigmatic and prescription as the testing features respectively. Further, there are 4 leaves. Each leaf is labeled with a unique classification label (*hard, soft* or *none*).

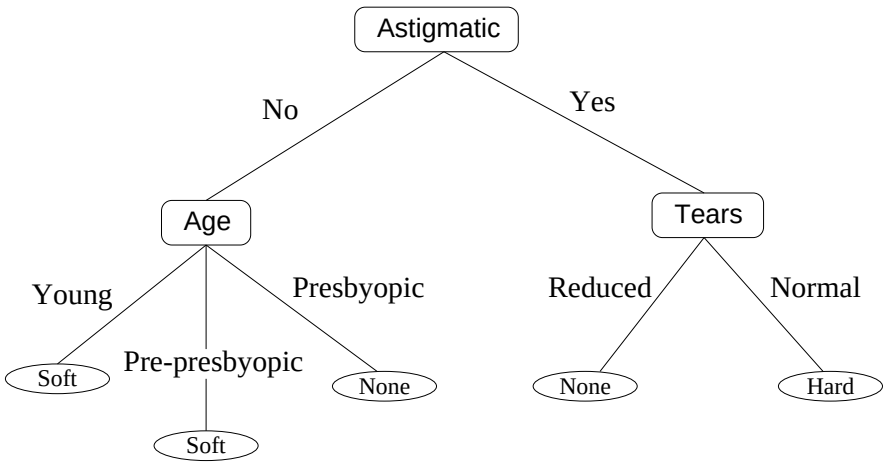


Figure 2: A decision tree for the concept represented by *lenses* data.

Table 1: List of the examples comprising the "lenses" dataset. The dataset has four input features and one output feature (shown last). The lenses dataset originally consists of 24 instances covering all the possible combinations. For our explanation, we use only 17 instances.

young, myope, no, reduced, soft
young, myope, no, normal, soft
young, myope, yes, reduced, none
young, hypermetrope, no, normal, soft
young, hypermetrope, yes, reduced, hard
pre-presbyopic, myope, yes, reduced, none
pre-presbyopic, myope, yes, normal, hard
pre-presbyopic, hypermetrope, no, reduced, none
pre-presbyopic, hypermetrope, no, normal, soft
pre-presbyopic, hypermetrope, yes, reduced, none
presbyopic, myope, no, reduced, none
presbyopic, myope, no, normal, none
presbyopic, myope, yes, reduced, none
presbyopic, myope, yes, normal, hard
presbyopic, hypermetrope, no, reduced, none
presbyopic, hypermetrope, yes, reduced, none
presbyopic, hypermetrope, yes, normal, none

2.3.2 Classifying an instance

Consider the following test instance with the following values for various features:

$e1 = (young, hypermetrope, yes, normal)$

This instance will be sorted down the rightmost branch of the decision tree in Figure 2 and would therefore be classified as a *hard* instance.

In general, decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances. Each path from the root to a leaf corresponds to a conjunction of attribute tests, and the tree itself is a disjunction of such conjunctions. For example, the decision tree in Figure 2 corresponds to the following expressions for each type of leaf:

If (Astigmatic = no)
$\wedge$ (Age = $Pre - Presbyopic \vee$ Age = $Young$)
Then (Type of Lens = $soft$)

If (Astigmatic = no) $\wedge$ (Age = $Young$)
Then (Type of Lens = $none$)

If (Astigmatic = yes) $\wedge$ (Tears = $reduced$)
Then (Type of Lens = $none$)

If (Astigmatic = yes) $\wedge$ (Tears = $normal$)
Then (Type of Lens = $hard$)

2.3.3 ID3: The basic algorithm and further

Most of the algorithms that have been developed for learning decision trees are variants of the basic algorithm that employs a greedy, top-down search through the space of possible decision trees. The ID3 algorithm [Quinlan, 1990], is one of the first decision tree algorithms.

ID3 builds a tree starting with the root node. An attribute is selected from the attribute set using a statistical test that classifies the entire training set in the *best* possible way with one attribute (*best* defined below). The root is associated with this attribute. A descendent

Table 2: The basic ID3 [Quinlan, 1990] decision tree learning algorithm for learning boolean-valued functions.

`ID3algorithm` (E, T, I)
Inputs: example set E, target class T, set of available input features I

- Create a node called *root* (corresponding to the given examples E).
- If all the examples in E belong to the same class (say p) Then
  - Assign the target class value (p) to the *root* node and return *root*.
- Else
  - For each available attribute $a \in I$:
    * Calculate $Gain(E, a)$
  - Select the attribute (a_{max}) such that $Gain(E, a_{max})$ is maximum.
  - For each possible value v of a_{max}:
    * Create a new branch for v below the *root* node.
    * Let E_a be the subset of examples in E that have a value of v for the attribute a_{max}.
    * If the set of examples E_a is *empty*, Then
      . Add an empty node K below the newly formed branch.
      . Assign to node K, the most common target attribute value observed at *root*.
    * Else
      . Call `ID3algorithm`(E_a, T, I - $\{a_{max}\}$)
  - Return *root*

---

of the root node is created for each possible values of this attribute. Examples are sorted to the appropriate descendent node in accordance with the value of the example for this attribute. The above process is repeated for each of these descendent nodes. Table 2 presents a formal version of the ID3 algorithm that specifically learns boolean-valued functions.

2.3.4 Finding the Best Attribute

As noted above, the central choice in the ID3 algorithm is selecting which attribute to test at each node. ID3 uses a statistical measure called *information gain* to decide on which attribute is the most useful in classifying examples. Information gain has its roots in information theory and is closely related with another term called *entropy*. An explanation of entropy follows before we discuss information gain.

A crude definition of entropy is that it is a measure of the (im)purity in an arbitrary collection of examples. Entropy, from an information theoretic perspective is termed as a lower bound on the average number of bits required to transmit the message without loss across a communication channel.

For a given arbitrary sample S, consisting of s_+ positive examples and s_- negative examples, the entropy of the sample is calculated as:

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

where:

$$p_{\oplus} = \frac{s_+}{|S|} \quad and \quad p_{\ominus} = \frac{s_-}{|S|}$$

$p_{\oplus}$ = proportion of the positive examples in the sample S.

$p_{\ominus}$ = proportion of the negative examples in the sample S.

$|S|$ = number of examples in the sample S.

Entropy is zero when the sample is *pure*, which means that all the examples in the sample S belong to one class (i.e., all the examples in S are either positive or negative). Entropy attains a maximum value of 1 when the sample is *maximally* impure (i.e., there are equal proportions of positive and negative examples in the sample). Entropy when plotted for different proportions of positive and negative examples might look as shown in Figure 3.

Though we have considered a boolean classification in the above discussion, entropy as a measure can be generalized for more output values. Hence, for a *c-wise* classification, the formula for entropy would be:

$$Entropy(S) \quad \equiv \quad \sum_{i=1}^{c} -p_i log_2 p_i \tag{3}$$

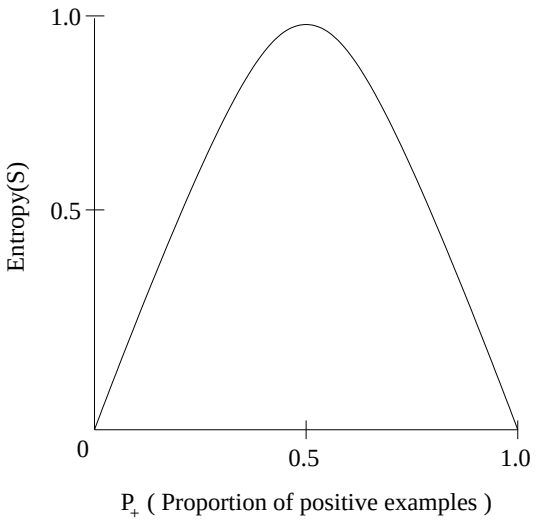


Figure 3: The entropy function plotted against the proportion of the positive examples in a sample. Entropy of the sample is highest when P_+ is is 0.5, which means the positive and negative examples are equally distributed in the sample.

---

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

We use the above formula for entropy on our *lenses* dataset. The lenses dataset has a total of 17 instances with a distribution of three, four and ten for the output feature (*type of lens*) values *hard*, *soft* and *none* respectively. In this case, the entropy is:

$$Entropy([3_h, 4_s, 10_n]) \quad = \quad -\frac{3}{17}log_2(\frac{3}{17}) - \frac{4}{17}log_2(\frac{4}{17}) - \frac{10}{17}log_2(\frac{10}{17})$$

The subscripts 3, 4 and 10 denote the respective output feature values. For example, in the above equation, we calculate the entropy of 17 instances with a distribution of three *hard*, four *soft* and ten *none* instances respectively.

The Information gain of an attribute A is the expected decrease in entropy caused by partitioning the examples according to that attribute. It is used to compare the effectiveness of an attribute on a given set of examples. More precisely, the information gain, $Gain(S, A)$,

for a given sample S over an attribute A is defined as:

$$Gain(S,A) \quad = \quad Entropy(S) \; - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{4}$$

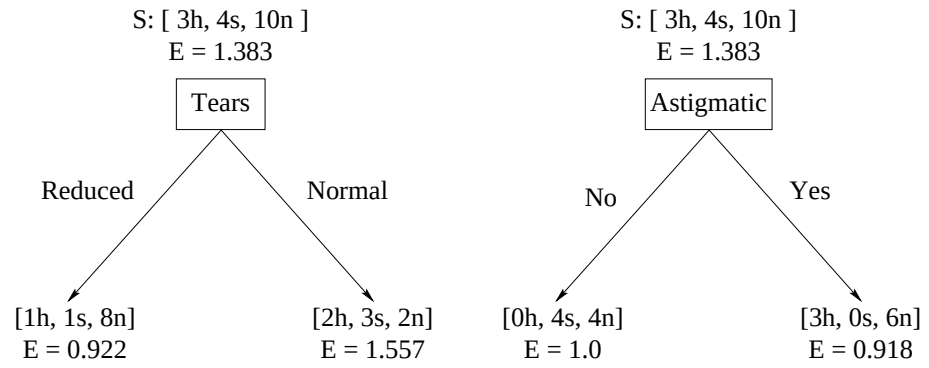
where Values(A) is the set of all possible values for Attribute A, and S_v is the subset of S for which attribute A has value v (i.e., $S_v = s \in S | A(s) = v$). The first term is the entropy of the original collection S. The second term is the expected value of the entropy for the sample S after it has been partitioned by the attribute A. It can also be viewed as the decrease in the number of bits required to encode the target value of an arbitrary example in S by knowing the value of the attribute A.

2.3.5 An Example

Consider the lenses dataset with the above output class distribution. To calculate the information gain associated with the Astigmatic attribute for the above example set, we first determine that there are two possible values of Astigmatic: *no* and *yes*. Assuming that out of the 17 examples, one *hard*, two *soft* and six *none* examples have a value of *no* and the rest have a value of *yes* for Astigmatic attribute, we can calculate the information gain as follows:

$$\begin{aligned}
Values(Astigmatic) &= no, yes \\
S &= [3_h, 4_s, 10_n] \\
S_{Astigmatic=no} &= [0_h, 4_s, 4_n] \\
S_{Astigmatic=yes} &= [3_h, 0_s, 6_n] \\
Gain(S, Astigmatic) &= Entropy(S) - \sum_{v \in \{no,yes\}} \frac{|S_v|}{|S|} Entropy(S_v) \\
&= Entropy(S) - (\frac{8}{17}) Entropy(S_{Astigmatic=no}) \\
&\quad -(\frac{9}{17}) Entropy(S_{Astigmatic=yes}) \\
&= 0.426
\end{aligned}$$

At any given node in the decision tree, the information gain values are calculated for all the available attributes at that node, as done above for the attribute Astigmatic. The



$Gain(S, Tears)$
$= 1.383 - \frac{10}{17}0.922 - \frac{7}{17}1.557$
$= 0.2$

$Gain(S, Astigmatic)$
$= 1.383 - \frac{8}{17}1.0 - \frac{9}{17}0.918$
$= 0.426$

Figure 4: Selecting the best attribute: The two attributes Tears and Astigmatic are being evaluated at the root node. The information gain calculations involved are also shown above.

---

attribute that gives the highest information gain is selected as the attribute for testing at that node. In other words, the selected attribute is associated with that node.

For example, in the above example set, the attribute *Astigmatic* was found to give an information gain value of 0.426 which was the highest gain of all the attributes. Hence, it was selected as the attribute used for testing at the root. Figure 4 illustrates the use of information gain in making a choice of attribute at a particular node.

2.3.6 What does a decision tree algorithm search for?

Induction algorithms, in general, can be viewed as searching a hypothesis space to find the best hypothesis that might explain the training examples. The hypothesis space in the case of decision trees is the set of all possible decision trees that can be formed. ID3 performs a simple-to-complex hill-climbing search through this hypothesis space, beginning with an empty tree and gradually considering more complex hypotheses. The hill-climbing search involves moving from a decision tree in the hypothesis space to a tree with one additional node that leads to better classification of the training set. ID3 does not use any backtracking

and hence, does not search this hypothesis space in its entirety. The evaluation schema used in this hill-climbing search is the information gain measure.

Some of the important features of ID3 algorithm with respect to its search space are as follows:

- The hypothesis search space of ID3 is a complete space of discrete-valued functions for the given set of attributes as parameters. This avoids the problems of searching incomplete hypothesis spaces which might lead to biased and non-optimal solutions.
- ID3 in its purest form does not support backtracking. This also brings with it all the anomalies of any hill climbing algorithm which does not involve backtracking. This might result in finding a locally optimized solution that does not provide the best possible global optimization. C4.5, an improved version of ID3 does provide a mechanism similar to backtracking in the form of post-pruning [Quinlan, 1993].

2.3.7 The Inductive Bias of ID3

The ID3 algorithm selects only one decision tree given an example set, though there may be many different trees consistent with the data. The bias of ID3 is mainly towards finding trees that have attributes with higher information gain nearer to the root. This indirectly implies the inclination of the algorithm towards selection of shorter trees than longer and complex ones. It can be supported by the fact that by the selection of attributes that split the data in the best possible way at every node, the algorithm tries to ensure that it encounters a leaf at the very earliest possible point and hence explains the bias for shorter trees.

2.3.8 C4.5: Extending ID3

ID3, one of the basic forms of decision tree learning algorithms, does not handle many practical issues like allowing continuous values, determining how deeply the tree should grow, handling examples with missing attribute values, cost-dependent attributes and improving computational efficiency. An extension of ID3 that addresses most of the above issues

is C4.5 [Quinlan, 1993]. In this section, we will discuss the above issues in more detail and look at the popular existing approaches to solving them and in particular, the C4.5 mechanism of resolving them.

2.3.8.1 Continuous Values for Attributes

It is not uncommon to have an attribute that takes continuous *numeric* values. Consider a simple classification problem of classifying cities as being polluted or not. An input feature that might be of interest would be the density of population in that city. If we consider the raw densities as input feature (values), then we are dealing with a continuous attribute. One of the basic constraints of ID3 (discussed in Section 2.3) is that it requires the output and input features to take only discrete values. For the output features, it is an inherent requirement of the decision tree algorithm. But, the case of discretization of the continuous input feature values is very easily attainable. A simple technique is to transform the continuous attribute (say A) into a discrete attribute by partitioning the range of the possible continuous values into a set of discrete intervals.

For example, consider a continuous attribute that measures temperature (say T). Let us assume that the range of possible values is 0 to 100 F. We can transform it into a discrete attribute (say T_d) that takes 3 different attribute values (cold, warm and hot). The partitions are such that 0-25 F are equivalent to cold, 25-75 F are considered warm, while the rest of the continuous values are considered hot.

$T \quad = \quad continuous\ temperature\ attribute.$

$T_d \quad = \quad discrete\ temperature\ attribute.$

$t \in T, \quad t_d \in T_d$

$if \quad 0 \leq t < 25 \quad then \quad t_d = cold$

$if \quad 25 \leq t < 75 \quad then \quad t_d = warm$

$if \quad 75 \leq t < 100 \quad then \quad t_d = hot$

One important question to consider is how do we decide the number of partitions that we make? C4.5 considers a simple single split for any continuous attribute (i.e., we have only two partitions and hence only one split point). For example, in the case of the above

illustrated temperature example, C4.5 always transforms the continuous feature T into a discrete boolean valued feature T_d with respect to a threshold K such that:

$T \quad = \quad$ *continuous temperature attribute.*

$T_d \quad = \quad$ *discrete temperature attribute.*

$t \in T, \quad t_d \in T_d$

$if \quad 0 \leq t < K \quad then \quad t_d = 0$

$if \quad K \leq t < 100 \quad then \quad t_d = 1$

C4.5 only considers single splits, but can imitate the multiple split point method by splitting more than once (along a path from root to a leaf) on the same continuous attribute. Figure 5 illustrates such a possibility in C4.5. The possibility of imitation of a multiple split in C4.5 arises due to the fact that continuous attributes always get maintained in the list of attributes available for splitting during the process of building the tree. This is because a continuous attribute can itself be considered as a set of many possible discrete-valued attributes. By selecting one of these attributes at some node, the algorithm does not eliminate the other possible discrete-valued attributes to be selected later on in the tree.

Finding the Split Point

A well-chosen split point should help in splitting the data to the best possible extent. After all, a main criterion in the greedy decision tree approach is to build *shorter trees.* The best split point can be easily evaluated by considering each unique value for that feature in the given data as a possible split point and calculating the associated information gain. A simpler, less computationally intensive method due to Fayyad [1991] is used in C4.5. In this method, the data is first sorted according to its value for the corresponding continuous attribute. Now, adjacent examples that differ in their classification value are identified. The mid-point between the corresponding values of such adjacent examples for the attribute under consideration form a possible splitting threshold [Fayyad, 1991]. The threshold that gives the highest value for the information gain is selected to be the split point.

2.3.8.2 Avoiding Overfitting - Pruning the tree

ID3 builds a tree so that all the training examples are classified correctly. While this

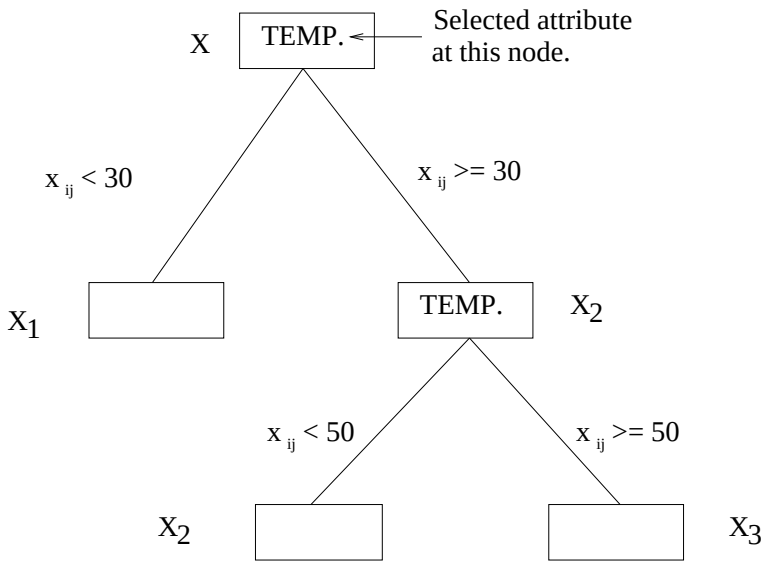


Figure 5: The continuous attribute *temp* is selected at two nodes along a path from root to a leaf. The splitting at these two points imitates a two-point split. Note that X_1, X_2 and X_3 are the three partitions created by splitting twice on temperature. The temperature ranges corresponding to these partitions are [0, 30), [30, 50) and [50, 100] respectively.

---

approach is correct when there is no noise, lower accuracies might be obtained in cases where there is noise in the data and/or the number of training examples is very small for the decision tree to decipher the target function. This phenomenon is called *overfitting*. A more formal definition for overfitting provided by [Mitchell, 1997] is as follows:

> *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' on the training data but h has a larger error than h' over the entire distribution.*

The accuracy of a concept learning algorithm L over a concept C can have different values depending on the examples we consider for training. The accuracy might degrade if there is a lot of noise associated with the training examples. We can estimate the accuracy using

a set of unseen examples. Clearly, the examples should *support* the cause of the underlying concept to provide good estimation. This form of evaluation gives a better estimate of the generalization achieved by the concept learner since the learner is predicting unseen instances. Hence, it is more important that the learner succeeds in finding a hypothesis (*in the hypothesis space it searches*) that gives the least possible error for the entire distribution of instances (all examples). If the learner failed to capture such a hypothesis then it is deemed to have overfitted.

There are various techniques to avoid overfitting, they can be grouped into two classes:

- Approaches that *stop growing* the tree early, before it reaches a point where it perfectly classifies the training data. This might sound simple, but for practical purposes, it is difficult to estimate precisely when the tree building should actually stop.

- Approaches that allow the tree to overfit the data and then *post-prune* the tree. Pruning refers to the process of shortening the tree built in order to overcome overfitting. This generally involves removal of some of the nodes or subtrees from the original decision tree. A general approach to pruning involves the use of a separate unseen set of examples. The goal is to improve (by pruning) upon the accuracy on this unseen set of examples. There are two ways of doing post-pruning: *Reduced Error Pruning* and *Rule Post Pruning*.

Reduced Error Pruning [Quinlan, 1987], refers to the process of building a tree and then considering each of the decision nodes in the tree to be candidates for pruning. Pruning a node involves removal of the subtree rooted at that node, making it a leaf and assigning it the most common classification of the training examples associated at that node.

Rule-Post Pruning is a second powerful method of post-pruning. A variant of this method is used by C4.5 [Quinlan, 1993]. It involves the following steps:

- Grow the decision tree completely.

- Convert the tree thus grown into an equivalent set of rules, by creating one rule for each path from the root to a leaf.

- Prune each rule by removing any preconditions that result in improving the accuracy of the tree.

- Sort the rules according to their estimated accuracies and consider them in that order when classifying the test data.

C4.5 uses a pessimistic estimate of the accuracy to evaluate the performance based on the training set itself.

2.3.8.3 Alternate Measures for Selecting Attributes

Information gain has a natural bias towards selecting attributes with many values rather than those with few values. For example, consider a case where an attribute $date\_of\_birth$ is added to our lenses dataset. With many different values for $date\_of\_birth$, it will split the data very well (in terms of Information Gain). If all the patients in the example set were to be born on a different date, $date\_of\_birth$ provides a perfect split. But this is unlikely do well on future instances and surely not the concept we want to learn. C4.5 uses an alternative measure called *Gain Ratio*, that suppresses this bias. The gain ratio includes a newer term called *Split Information* that effectively takes into account the factor of an attribute having many values. We define split information as follows:

Let S be the set of examples and A be the attribute in concern.

Let $Values(A)$ be the set of all possible values of attribute A.

Let S_v be the subset of S for which attribute A has a value v.

Then the split information for the example set S and attribute A is defined as:

$$SplitInformation(S, A) \quad = \quad - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} log_2 \frac{|S_v|}{|S|} \qquad (5)$$

And the gain ratio is:

$$GainRatio(S, A) \quad = \quad \frac{InformationGain(S, A)}{SplitInformation(S, A)} \qquad (6)$$

By using split information, which is proportional to the number of values an attribute A can take, gain ratio effectively removes the bias of information gain towards features with many values. A practical issue that can arise by using the gain ratio formula is what if the split information is very small or equal to zero? C4.5 considers this issue and as a remedy it first lists the set of attributes with information gain above the average information gain for that node and then it uses the gain ratio to select the best attribute from the list [Quinlan, 1990].

2.3.8.4 Training examples with missing attribute values

It is not uncommon to find data that is incomplete and hence has missing values. For example, the sample training data may be missing values for some of the attributes. The decision tree algorithms discussed so far provide no facility to handle such a case. C4.5 provides an approach to handling missing values. Suppose $[x, c(x)]$ is the example that has a missing value for attribute A in the example set S. Some of the methods to deal with missing values are discussed below:

- Assign A the most common value observed at that node for that particular attribute. For example, consider an example set of 6 instances. The output feature can be a boolean value 0 or 1. One of the input attributes is *Temperature*. The examples with the corresponding values for *temperature* are listed below:

Temperature:	hot	cold	hot	?	cold	hot
Output Class:	1	0	1	0	0	1

  Then, the unknown value (shown as ?), is assigned a value *hot* according to the above explanation.

- Assign it the most common value among the examples at that node that have the same classification value $c(x)$. In the above example, the class associated with ? is 0. Hence, the most common value for examples with output class = 0 is *cold*. Hence, we assign a value *cold* to that example.

- A more complex procedure, used in C4.5, is to replace the example $[x, c(x)]$, with a corresponding set of examples, where each new example has a different possible value for the attribute A. Each of these new examples are assigned weights according to the ratio of examples possessing the same value for the attribute A at that node. Similarly, the classification of an example with unknown attribute values is done by splitting the test example along different paths. The final classification value is calculated by summing up the weights across the different leaf nodes for that example. For example, in the above example, if we were to select *temperature* as the best attribute and split on it, two new instances are created to replace the unknown instance. They will take values *hot* and *cold* with a corresponding weight of 3/5 and 2/5 respectively. Now, all the instances with the value *cold* and *hot* would move down the respective branches (including these newly generated fractional instances).

Now we move on to the other basic tool used in this thesis work - ensemble learning. We look at two popular ensemble learning techniques: Bagging and Boosting.

2.4 Ensemble Methods

One of the most effective learning methods is ensemble learning. In ensemble learning, we have a group of experts (concept learners that have been trained on this concept) at our disposal. We submit our question (test example) to each of these concept learners. To get our answer, instead of using the answer provided by one of these experts, we find a solution by combining the answers provided by the entire group. This is the central idea of an ensemble. An ensemble combines the predictions of a set of classifiers which are independently trained. Research in the past has shown that an ensemble leads to a better approximation of the target function than any of the constituent individual classifiers themselves [Bauer and Kohavi, 1999, Maclin and Opitz, 1997]. While the individual classifiers are not restricted to a particular class of learning algorithm(s), we are particularly interested in ensembles of decision tree learners. Two popular mechanisms for creating ensembles are Bagging and Boosting, which are known to work very effectively for decision

trees [Freund and Schapire, 1996, Breiman, 1996a, Breiman, 1996b, Quinlan, 1993]. These methods rely on re-sampling techniques to obtain different training sets for the individual classifiers forming the ensemble. It has been shown that an ensemble formed by highly accurate individual classifiers that disagree to the largest extent leads to good generalization of the target function [Krogh and Vedelsby, 1995]. Bagging and Boosting try to achieve this effect by training each classifier on differing training sets. In the next two sections, we look at these two mechanisms in further detail.

2.4.1 Bagging

Bagging is an ensemble approach where each classifier is trained on a different bootstrap sample of data. A bootstrap sample is generated by randomly drawing, with replacement, N examples - where N is the size of training set. Because of re-sampling, many of the examples may be repeated in the resulting sample while some may be left out.

Figure 6 illustrates a bagging ensemble. As shown in the figure, each individual classifier k learns from a subset of the original data.

Hence, the error of the classifier k by itself might be larger than the error of a classifier that learns from the whole of the sample data. In fact, this might be the case with each of the individual classifiers. But, when these classifiers are combined into an ensemble, they produce an error that is generally smaller than the error of a single classifier. Bagging uses a simple voting combiner although much more complex combining methodologies have evolved. Given a test instance, each classifier returns a corresponding output prediction to the combiner. Then, the output value that was selected by most of the classifiers will be the combiner's output. The reduction in the error might be accounted for by the diversity in the collection of constituent classifiers. Bagging is effective on unstable learning algorithms, that produce a large change in predictions for a corresponding small change in the training set [Breiman, 1996a]. Decision trees are a form of unstable learning algorithm.

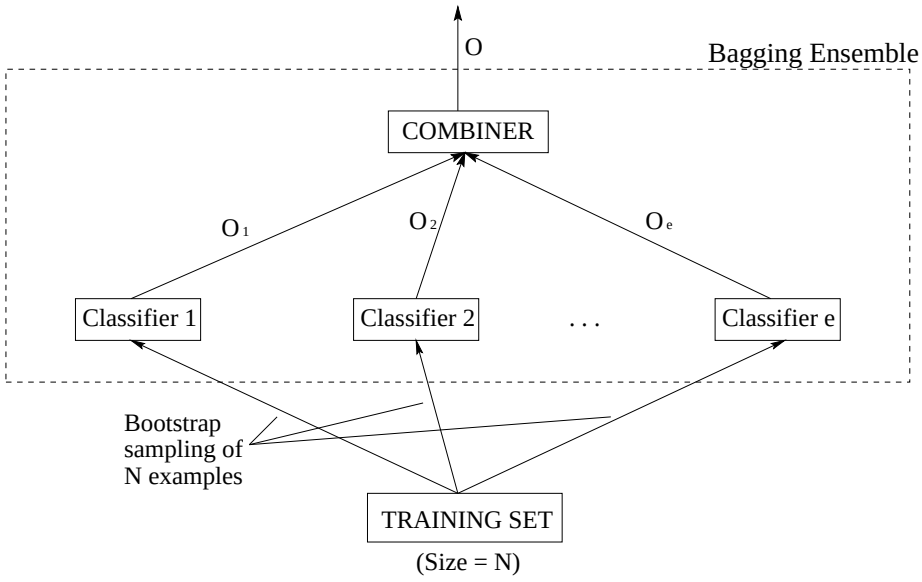


Figure 6: A bagging ensemble on size n. A bootstrap sample of the training set is used for training each individual classifier.

2.4.2 Boosting

A key criteria to forming a good ensemble is to achieve diversity among the classifiers that build up the ensemble. Thus the predictions for a given data point should vary. But at the same time, a majority of the component classifiers should classify the data point correctly. This ensures that the error rate of the ensemble will be lower than that of the individual classifiers. Boosting was proposed by Schapire [1990], as a method for converting a *weak* learning algorithm (an algorithm that performs slightly better than random guessing) to a *strong* learning algorithm (an algorithm that performs very well).

The main idea involved in boosting is to produce a set of classifiers sequentially. The weights for the various training examples used to create each classifier depends on the performance of the previous classifiers. Basically, the examples that were misclassified by the previous classifiers receive a *boost* and are selected more often than the other examples. This gives the advantage that boosting adaptively changes the distribution of the training

set based on the performance of the previous classifiers. Two popular forms of boosting are Ada-Boosting [Freund and Schapire, 1996] and Arcing [Breiman, 1996b]. The two methods (Ada-Boosting and Arcing) differ in the changes they produce to the training-set distributions. They also have different methodologies for combining the individual classifiers' outputs.

Both of these above boosting methodologies initially start by assigning a probability of 1/N to each examples (where N is the number of training examples). A classifier c is built by randomly generating a training sample using the probabilities assigned to each of the examples. The examples are now classified using the trained classifier. The probabilities of the mis-classified instances are changed in the two algorithms as follows:

In Ada-Boosting, the sum of the misclassified instance probabilities is E_k for a classifier C_k. Then, the probabilities (p_{ik}) of the mis-classified instances are modified by multiplying them by a factor $B_k = \frac{1-E_k}{E_k}$. Hence,

$$p_{ik+1} \quad \leftarrow \quad p_{ik}\frac{1-E_k}{E_k}$$

Further, all of the probabilities are normalized so that the sum equals 1. The new probabilities form the distribution for the next classifier $k+1$. Ada-Boosting combines the classifiers $C_1, ..., C_k$ using weighted voting where C_k has weight $log_2(B_k)$.

In Arcing, the updation of the probabilities occurs differently. For the i^{th} example in the training set, let m_i represent the number of times that example was misclassified by the previous K classifiers. The probability p_i for selecting example i for the $(K+1)^{th}$ classifier is

$$p_i \quad = \quad \frac{1+m_i^4}{\sum_{j=1}^{n} 1+m_j^4}$$

Bagging and boosting are both found to work well for ensembles involving decision trees and neural networks [Maclin and Opitz, 1997]. We focus mostly on bagging ensembles with modifications in our pursuit for feature subset selection. In the next chapter, we will discuss the methodology used to find plausible feature subsets.

3 Methods Tested For Feature Selection

In this chapter, we discuss the various feature subset selection approaches that were evaluated as a part of this thesis work. Our approaches are built on ensemble learning. A basic setup of a bagging ensemble of C4.5 decision trees was used to estimate the relative performances of the various approaches. The estimated accuracies of C4.5 decision tree learners and the bagging ensemble were used as baselines in the evaluation of the various subset selection techniques.

Figure 7 illustrates the complete setting that we use for evaluating our feature subset selection techniques. The subset selection approaches basically create a probabilistic distribution of features ($\mathcal{D}_F$) that will be used by the random-picker to generate a subset (of the pre-determined size k) of features (F_k) for each of the individual classifiers of the ensemble. Each classifier will now have access to that subset of input features, namely F_k.

For example, if we were to use the "lenses" dataset (discussed in Section 2.1.1) which originally has four input features, and our random subset selector picks two features, namely *age* and *prescription* for use by a classifier, then an example of the form:

$$e = < young,\ myope,\ no,\ normal >$$

would actually be represented as $< young, myope >$. The other two features are not considered at all in the learning process of the classifier. They are simply "removed" from the vision of that classifier. The ensemble consists of a combination of such classifiers. It is evaluated on the test set and the corresponding results signify the performance of a particular feature subset selection algorithm for a given subset size k. The ensemble classifiers are provided with the subset of features by a feature subset generator. The various feature subset selection techniques that we have tested include:

- *Random Subset Selection* - In this approach, every feature is considered equally probable. This method is trivial but vital as a baseline.

- *Information-Gain Based Selection* - Features that have a high information gain for the training set are selected. This is motivated by the decision tree approach.

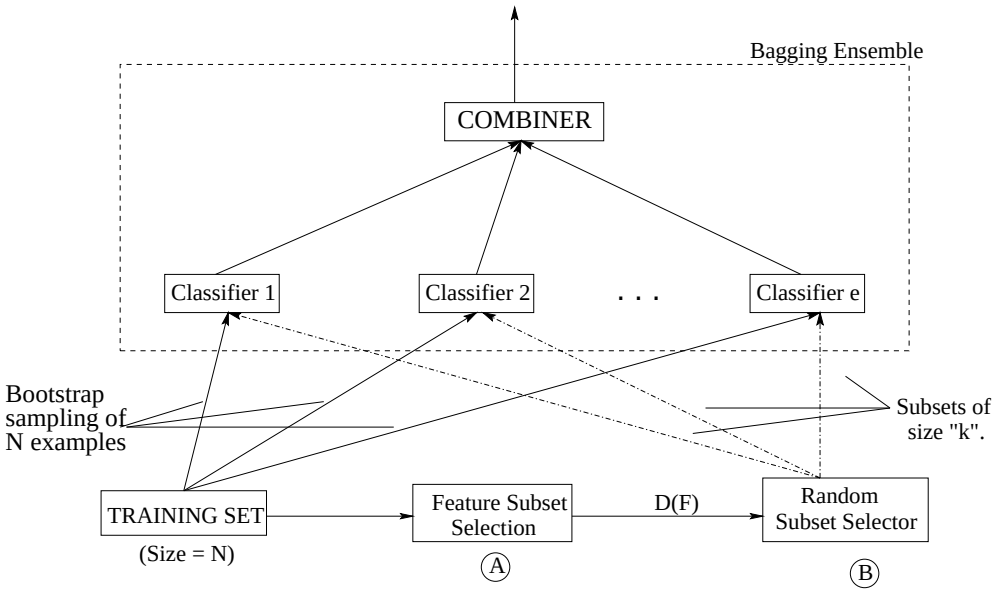


Figure 7: The general feature subset selection setup is shown above. *Component A* (Feature subset selection) produces a distribution of features (i.e., weights associated with each feature). *Component B* (Random subset selector) produces e (number of components in the ensemble) different randomly generated subsets of features from the distribution $\mathcal{D}(F)$, which is the output of component A.

- *Cumulative Information-Gain Based Selection* - Features are selected based on their *cumulative information gain*. This quantity that shows how well a feature splits the data (all of it) at a given level in the decision tree (see Section 3.5 for more details).

- *Feature Boosting* - This methodology uses a technique similar to boosting in order to find a better distribution of the features. It uses decision tree classifiers as a part of the boosting technique and updates the feature weights over the sequence of classifiers.

We now move onto a brief description of the "tools" used in this process, namely C4.5 decision trees and the bagging ensemble prior to the discussion of the various subset selection techniques.

3.1 Basic Setup

Our solution to the feature subset selection uses an ensemble of decision trees. Hence, the two basic tools are *decision tree classifiers* and *an ensemble* built using decision trees as components. We use C4.5 as our decision tree model. It is a successor of ID3 decision tree model (see Section 2.3.8 for details). We use bagging and boosting ensemble methods in our subset selection approach. A brief description of these models follow.

3.1.1 C4.5 - Base Evaluation

All decision tree models encapsulate a feature subset selection strategy. Their bias for shorter trees results in the probable elimination of some features from the resulting decision tree. The C4.5 model is no exception. In fact, C4.5 achieves further elimination of features through pruning. C4.5 uses rule-post pruning to *remove* some of the insignificant nodes (and hence, some *not so relevant features*) from the tree. This basic form of subset selection algorithm will be used as a baseline in our evaluation process.

3.1.2 Bagging Ensemble

As we said earlier, our approach to subset selection will involve generation of a probabilistic distribution (weights) for the various features in the input feature set. The new distribution of the features $\mathcal{D}_F$ thus obtained with respect to its weights will be used to randomly pick a sample subset F_k (where k is the size of the subset). Each such sample subset will govern an individual classifier's learning process (i.e., the features whose information is visible to that classifier). The sample data required to train the classifier is selected from the ensemble's training dataset using *bootstrapping* (explained in Section 2.4.1). The accuracy of the bagging ensemble using all the features (which is actually a simple bagging ensemble) serves as our second baseline. Before we develop our mechanisms using the above mentioned *basic tools*, we take a closer look at the evaluation procedure.

3.1.3 A Note on Evaluation

Throughout this thesis work, the evaluation of an induction algorithm has been done using ten fold cross validation. Ten-fold cross validation works as follows:

---

1. Divide the given dataset D randomly into 10 subsets ($D_1 \ \ldots \ D_{10}$) of approximately equal size called *folds*.
2. $correct \ = \ 0$
3. For each of the folds D_i,
   - Select D_i to be the test set and train the classifier using the remaining nine folds.
   - Test the trained classifier using the examples of D_i. Add the number of correctly classified instances to *correct*.
4. Accuracy of the classifier $= \frac{correct}{|D|}$, where $|D|$ refers to the total number of examples in the dataset D.

This has the advantage of having each instance in the dataset used as a test instance exactly once. Another closely related evaluation technique is the *leave-one-out validation* [Kohavi, 1995]. In leave-one-out validation, only one example is separated out as a test instance and the classifier is trained on the rest of the instances and then tested using the left out instance. This process is then repeated over all the instances in the dataset so that all the instances serve as a test instance exactly once. Leave-one-out, while providing a better estimate of generalization does not seem to be a good testing method in terms of the computational overhead involved.

3.2 Modification to the bagging ensemble

A small modification was made to the method of combining the outputs of the individual classifiers in the original bagging ensemble. Since each of the classifiers learn based on

a subset of the actual input features F, the information available to these classifiers is restricted to this subset of features, leading to poor performance of some of the classifiers. We try to identify such classifiers by estimating their accuracies on the training set itself. This gives a rough estimate of the classifier's performance (although this estimate is biased). We normalize the accuracies of all the individual classifiers and use these normalized weights to determine a classifier's contribution to the ensemble.

For example, if the accuracy of the classifier i (in the ensemble) is a_i, then the normalized weight corresponding to this classifier would be

$$w_i \quad = \quad \frac{a_i}{\sum_{j=1}^{e} a_j}$$

where e = number of components in the ensemble.
Further, if a data instance y was tested on the ensemble, then the calculation of the output feature value will occur as follows:

Let h_i be the i^{th} classifier in the ensemble of size e.

Let D represent the training data set.

Assume that the output feature has k different values o $= \{o_1, o_2, ..., o_k\}$.

Further, let $h_i(D, y) \in \{o_1, o_2, ..., o_k\}$ represent the output feature value of the i^{th} classifier in the ensemble for a test instance y. Then, the output of the ensemble is

$$\underset{o_j \in o}{\text{argmax}} \quad [\sum_{\forall i,\ h_i(D,y)=o_j} w_i] \tag{7}$$

Some Definitions

There are some terms that we will frequently use in the next few sections. We mention below their specific purpose in our setup to avoid any misinterpretations.

Random subset selector: It is a component of the basic setup (component B in Figure 7). It produces a random subset of features from a given probability distribution of the features.

Attribute subset: This is the subset generated by the random subset selector for a particular component classifier. The classifier would have access to the feature values corresponding to these features (only) during the next training phase. For example, if we consider an input feature set $\{a, b, c, d, e\}$, the random subset selector could select features $\{b, d\}$ as the subset to be used.

In the following four sections, we discuss the four different variations that we have used for our feature subset selection process in Figure 7. The random subset selector uses this distribution ($\mathcal{D}(F)$) to select attribute sets for the various individual classifiers of the ensemble. This completes the process of creating the ensemble. We later discuss the process of evaluating the feature subset selection algorithm based on the performance of this ensemble.

3.3 Random Subset Selection

The bagging ensemble discussed above provides the basic infrastructure to build our feature subset selection approaches. One of the simplest techniques implemented was to randomly select a subset of features to be used by each of the individual classifiers forming the ensemble.

In this simplest form of subset selection, the probability distribution of the input features after *subset selection* does not change (i.e., all the features have an equal probability of getting selected by the random subset selector). If F represents the set of input features, $|F|$ represents the number of input features and $\mathcal{D}(f)$ represents the probability of selecting feature f from F,

$$\forall f \in F, \quad \mathcal{D}(f) \quad = \quad \frac{1}{|F|}$$

For example, consider the previously discussed *lenses* dataset. It has four input attributes namely *age, prescription, astigmatic* and *tears*. If the size of the subset was supposed to be two, then random subset selector would randomly select any combination of two attributes as the attribute subset for a component classifier of the ensemble.

3.4 Information-Gain Based Selection

This form of subset selection derives its motivation from the attribute selection criteria in ID3. Information gain is a measure of the change in entropy associated with a dataset by *knowing* the correspondence between an input feature and the output feature. At a given node in the decision tree, information gain provides a measure for deciding on which input attribute to use in splitting. As we use a decision tree as the basic classifier in our ensemble, an attribute with high information gain for the dataset might also be equally relevant in the process of concept generalization by the ensemble.

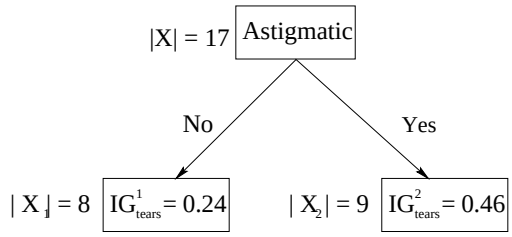
In this process of feature subset selection, we select the k attributes that have the largest information gain values over the entire dataset. In terms of the probability distribution generated in this case, the selected k attributes have a weight of $1/k$ associated with them and the remaining $|F| - k$ attributes have a zero weight. In this case, the use of random subset selector becomes redundant. It selects k features according to the distribution D that only has k features with positive weights. Hence, all the top k attributes are selected to form the attribute set for each of the member classifiers of the ensemble.

3.5 Cumulative Information-Gain Based Selection

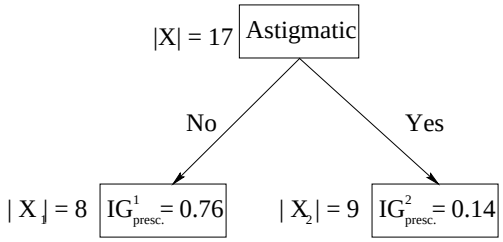
Information-gain based selection is a good approach because of the fact that the features selected will probably be the ones that each of the ensemble members would probably select as the root node. But there is an inherent pitfall. An attribute that provides good information gain for the complete set of examples might not do well on a fraction of the sample. We consider this case because, when each individual classifier is trained, it forms the root node and then the data is split up. Now, we look for an attribute that performs well on the split subsets of data. It might so happen that there were attributes that really perform well on these portions of data but perform poorly on the entire dataset and hence, were not even selected by the subset selection algorithm that was discussed in the previous section.

We modify our method of *picking* the attributes slightly in this case to account for the

1. Calculating Cumulative Info Gain for attribute Tears given the following values.



2. Calculating Cumulative Info Gain for attribute Prescription.



$$
\begin{aligned}
&Cumulative\ Gain(Tears): \\
&= \tfrac{1}{|X|}(|X_1| * IG^1_{tears} + |X_2| * IG^2_{tears}) \\
&= \tfrac{1}{17}(0.24 * 8 \ + \ 0.46 * 9) \\
&= 0.3565
\end{aligned}
\qquad
\begin{aligned}
&Cumulative\ Gain(Prescription): \\
&= \tfrac{1}{|X|}(|X_1| * IG^1_{presc} + |X_2| * IG^2_{presc}) \\
&= \tfrac{1}{17}(0.76 * 8 \ + \ 0.14 * 9) \\
&= 0.4318
\end{aligned}
$$

Figure 8: An example of calculating the cumulative information gain for some hypothetical values of information gains. In the above diagram, we consider calculating the cumulative gain values for attributes Tears and Prescription at level 1 of the decision tree. IG represents the corresponding feature's information gain for the examples at that node. X, X_1 and X_2 denote the set of examples associated with the respective node.

---

above problem. Figure 8 illustrates our modification. For our description of the refinement, we define a term called *cumulative information gain* that is associated with an attribute at a given *level* of the decision tree.

We can define *cumulative information gain* of an attribute A as the weighted sum of the information gains for an attribute A at each of the nodes at level k. The weights are

actually the proportion of examples at that particular node. Formally, it can be stated as follows:

Let p be the number of nodes at a particular level (say k) of the decision tree.

Let the set of examples associated with a particular node i at level k be D_i.

$$Cumulative\_Info\_Gain(D, A) \quad = \quad \frac{1}{\sum_{j=1}^{p} D_j} \sum_{i=1}^{p} D_i \; Gain(D_i, A)$$

An attribute that has the highest *cumulative information gain* at a given level of the decision tree is selected from that level of the decision tree. Then each of the selected k (assuming that the subset size is k) attributes are given a weight of $1/k$ while the rest of the attributes have a weight of zero. As in the previous approach, the random subset selector becomes redundant as it picks all of the above k features for each of the individual classifiers.

3.6 Feature Boosting

In this section, we discuss our method for assigning weights to the various features similar to the modification of the probability distribution of a training dataset in ensemble boosting. We will use the term *feature boosting* to identify this approach. Table 3 lists the algorithm for feature boosting. It is an iterative process of redistributing the weights of the input features based upon the accuracy of a classifier that was trained and tested on the training set ($\mathcal{D}_{train}$) using a subset of features (S_F) selected randomly based upon the current distribution of the feature weights. Figure 9 gives an illustration of the different *components* involved in this technique. A simpler way to explain the process is as follows:

A feature $f \in F$ has been selected for some iteration. The classifier is trained and tested on $\mathcal{D}_{train}$. If the resulting accuracy is comparatively higher (lower) than the average accuracy ($\overline{A}$) of the previous iterations, increase (decrease)

the weight of feature f so that it can be selected more (less) often in future iterations.

3.6.1 Criterion Function

The criterion function is used in the evaluation of the change in the weights of the various features. It has been formulated such that the weights of the features in the subset F_k will increase (decrease) depending on the accuracy of the classifier being better (worse) than the average accuracy of all the classifiers so far. We define a few parameters below before defining the function itself.

- $\overline{A}$ and $\sigma_{\overline{A}}$ are the mean and the standard deviation of the accuracies of all the classifiers built in the previous iterations.
- acc is the accuracy of the classifier in the present iteration.
- $|F|$ is the number of original input features and k is the size of the subset size.

The criterion function $\delta w(i)$ defined for a particular feature i is as follows:

$$\delta w(i) \quad = \quad (\frac{|F| - k}{|F| - 1})\ \psi(\frac{acc - \overline{A}}{\sigma_{\overline{A}}}) \tag{8}$$

where

$$\psi(x) \quad = \quad \frac{1 - e^{-4x}}{1 + e^{-4x}}$$

Once, the criterion function value for a given iteration is found out, the weights of features $f_k \in F_k$ are updated as follows:

$$\forall f_i \in F_k, \quad w(i) \quad = \quad (1 + \delta w(i))\ w(i)$$

The process of boosting was iterated a varying number of times and finally we fixed it to the size of the ensemble. The initial weight changes are made after the first few (*init*, see Table 3) iterations. These newly generated weights are normalized, so that they can represent a probability distribution, which is passed onto the random subset selector for

Table 3: Algorithm for feature boosting that would generate a weighted distribution of features.

- Let $\mathcal{D}_{train}$ be the training dataset, F be the set of features, k be the required subset size and $\mathcal{D}_F$ be the initial weight distribution of the features in the set F. Initially, each feature $f \in F$ has an equal chance $(1/|F|)$ of getting selected in the subset.
- For some (pre-defined) i iterations,
  - Select a random feature subset (F_k) of size k from F using the probability distribution $\mathcal{D}_F$.
  - Build a classifier (in our case, a C4.5 decision tree) using the training dataset D_{train} and estimate its accuracy on D_{train} itself. Add the accuracy to the list of accuracies.
  - If the present iteration count is above *init* then
    . Use the criterion function (defined below) to re-evaluate the weights associated with each of the features in F_k. The criterion function calculates the mean and standard deviation of the values in the list of accuracies and uses those values in the formula. Further, normalize the weights to obtain a new distribution $(\mathcal{D}_F)$ for the input features (F).
- Return the final distribution of the features $(\mathcal{D}_F)$.

---

further evaluation. The random subset selector (Component B in Figure 7) then generates the attribute sets for each individual classifier of the ensemble.

How the criterion function works

There are two factors that make up our criterion function. We discuss below briefly what they signify and how they change the weights.

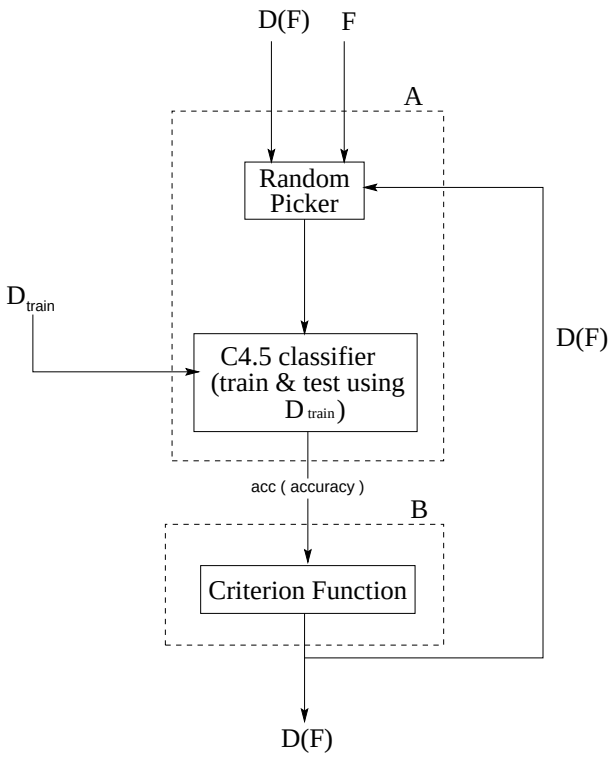


Figure 9: Feature Boosting based approach to subset selection. The above diagram illustrates the two components A and B used in the feature boosting technique. *Component A* includes the random subset selector. *Component B* contains the criterion function. This is used to update the weights (or distribution) of the input features based upon the accuracy (acc) of the classifier. D_{train} is the current training set and $D(F)$ is the probability distribution of the various input features. F is the initial set of input features.

- $(\frac{|F|-k}{|F|-1})$: This factor modulates the increase and decrease in weights according to the size of the ensemble. Our assumption is that in an ensemble of relatively larger size, there is a high chance that a feature might be used in the ensemble. So, the weights might be changed slowly. But in the case of a small ensemble, we want *bad* features to be replaced at the earliest. This will help in the convergence of the features

quickly in the case of smaller ensembles.

- $\psi(\frac{acc-\overline{A}}{\sigma_{\overline{A}}})$: This is the actual update factor. It takes into account how well the present classifier has performed relative to the available *history*. This is done by subtracting the mean ($\overline{A}$) from the classifier's accuracy and scaling it with the standard deviation (of the accuracies of classifiers in past). The resulting value will be proportional to how well the present classifier has performed. $\psi(x)$ function is used to scale the values obtained into a range of -1 to +1. We use the following formula for $\psi(x)$:

$$\psi(x) \quad = \quad \frac{1 - e^{-4x}}{1 + e^{-4x}}$$

  The above function is more commonly known as a *sigmoid function*. A constant 4 has been added to the exponent in order to increase the steepness of the resulting curve. This would increase the output value ($\psi(x)$) for a particular value of x. We decided on this constant (4) after some empirical evaluation. The use of this function ensures that the change in weight we make ($\delta w(i)$) is always less than the weight of a feature.

This method, along with the three previous approaches are tested in our experiments. In the following section, we briefly discuss the process of evaluating the generated ensemble by training and testing it using cross-validation.

3.7 Evaluating the Ensemble

Once the attribute subsets have been picked for the classifiers of the ensemble, the ensemble is trained using the training set ($\mathcal{D}_{train}$). We should recollect at this point that this was the set used to generate the attribute sets (see Sections 3.4, 3.5, 3.6). The training of the ensemble occurs by training each of the individual classifiers using a bootstrap sample (of size equal to the training set). Further evaluation occurs by testing the ensemble using the test set ($\mathcal{D}_{test}$). This evaluation process is repeated with some other fraction of the dataset serving as the test set.

The resulting accuracies directly signify the performance of a particular subset selection strategy. The results were collected for various subset sizes and various ensemble sizes. In

the next chapter, we discuss the results obtained and compare these four strategies and the baseline (bagging ensemble) that uses all the features.

4 Experiments and Discussion

In this chapter, we describe our experiments including the datasets that were chosen and the learning methodology used on our various feature subset selection algorithms. We further present the various results and discuss their significance.

4.1 Datasets

Table 4 provides a summary of the characteristics of the various datasets that were chosen for evaluation. All the datasets used in our evaluation were drawn from the UCI data set repository [Blake and Merz, 1998]. The chosen datasets come from diverse real-world situations. As can be observed from the table, the datasets show a variety of characteristics. All of the datasets were chosen to assist in the comparison of results obtained by other subset selection algorithms using the same datasets. The various characteristics described in the Table 4 include the number of example instances, the number of input and output features, the distribution of features with respect to their types (continuous or discrete).

4.2 Methodology

We now discuss the procedure used in our empirical evaluation of the different feature subset selection approaches.

4.2.1 General Procedure

The various algorithms used in the learning process include the two baseline algorithms (C4.5 decision tree and the bagging ensemble that uses all of the features) and the four different feature subset selection strategies (explained in the previous chapter). All of the algorithms were tested using ten-fold cross-validation. The final results are averaged over 10 (ten-fold) cross-validation experiments. Since we are using ensembles, the performance is evaluated for varying sizes of the ensemble. In our evaluation procedure we have considered sizes ranging from 5 to 25.

Table 4: The following datasets were chosen for the experiments. For each of the datasets, the number of discrete and continuous attributes, the number of possible output classes and the size of the dataset are shown. These datasets are all available from the UCI Machine Learning dataset repository.

Name of Dataset	Number of Attributes		Number of Output Classes	Number of Instances
	Discrete	Continuous		
breast-cancer-wisconsin	-	9	2	699
cleveland-heart	5	8	2	303
credit-a	9	6	2	690
credit-g	13	7	2	1000
glass	-	9	7	214
hepatitis	13	6	2	155
house-votes-84	16	-	2	435
hypo	22	7	5	3772
ionosphere	-	34	2	351
kr-vs-kp	36	-	2	3196
labor	8	8	2	57
pima-indians-diabetes	-	8	2	768
sick	22	7	2	3772

4.2.2 Evaluating Feature Subset Selection

The procedure used for the evaluation of various feature subset selection techniques involves ten-fold cross-validation (see Section 3.1.3 for details). Again, as with the baseline algorithms, the final results are averaged over 10 such (ten-fold) cross-validations.

In the next section we present the error comparison graphs and tables for the different feature subset selection methods discussed in this thesis work.

4.3 Results

Figures 10 - 22 present the plots for the error estimates obtained for the various subset selection mechanisms on the different datasets that were described above. We also include the corresponding error estimates obtained for the baseline C4.5 and bagging ensemble (with no feature selection). The number of classifiers used is represented on the *x-axis* while the error estimates are represented on the *y-axis*.

- *Breast-cancer-wisconsin* - For this dataset (see Figure 10), almost all the subset selection techniques outperformed the baseline bagging ensemble approach. With a subset size of two, both the information-gain based approaches performed worse than the baseline bagging approach. Feature boosting with a subset size of two was as good as the baseline bagging approach. Random subset selection using just three features produced the best results, though most of the methods using subsets of seven features were close.

- *Cleveland-heart* - For cleveland-heart (see Figure 11), the cumulative information gain based approach performed poorly. All of the feature subset selection techniques required more than two features to match the baseline bagging methodology. Feature boosting with a subset size of five produced accuracies almost equal to that of the baseline bagging. The information-gain based approach using a subset of twelve features produced the best results.

- *Credit-a* - The results obtained (see Figure 12) were as expected but they were not very interesting. There were no significant differences in the performances of various subset selection methodologies. Exceptions were the poor performance of the random subset selection and feature boosting for a subset size of 2. Feature boosting produced good results for larger subset sizes. Feature boosting was the best of all the subset selection techniques.

- *Credit-g* - For credit-g (see Figure 13), the information-gain based approach performed very well. Baseline bagging was very close to the best classifier. Feature boosting

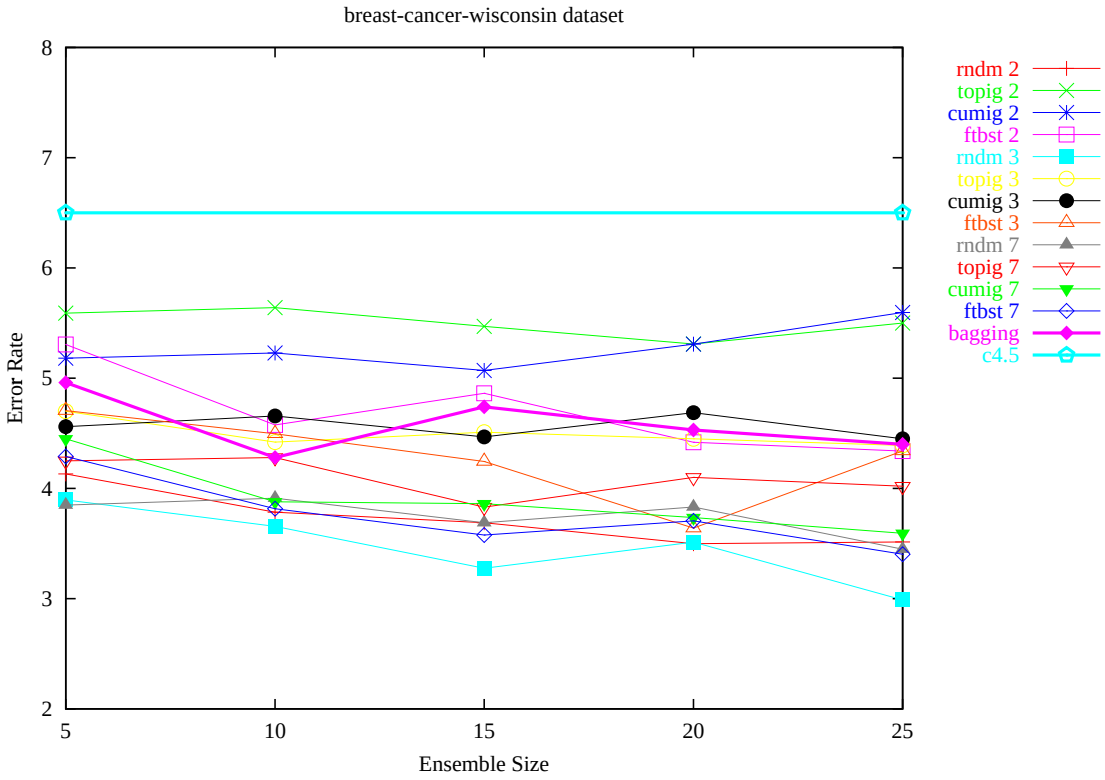


Figure 10: Comparison of the error estimates for the four different feature selection techniques: random subset selection (rndm), information-gain based selection (topig), cumulative information-gain based selection (cumig) and feature boosting (ftbst) on *breast-cancer-wisconsin* dataset. In each feature selection case, the error estimates for different subset sizes are plotted. The error estimates for the baseline C4.5 and bagging ensemble using all the features (bagging) on this dataset are also plotted for comparison. The maximum number of possible features (used in a subset) is ten. Note: in these figures, the ensemble size shown on the x-axis does not start from 0. Further, in many cases the error rate which is shown on the y-axis might not start from 0.

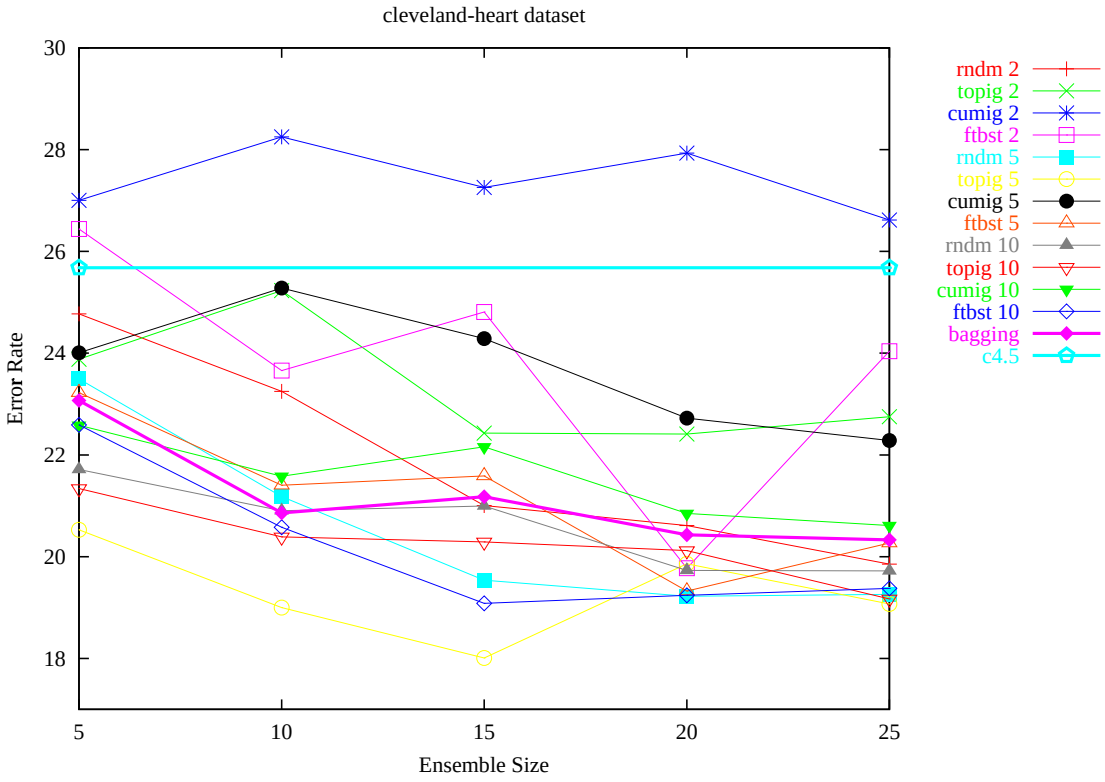


Figure 11: Comparison results similar to Figure 10 for *cleveland-heart* dataset. The maximum number of possible features (used in a subset) is seven. Note: the error rate shown on y-axis does not start from 0.

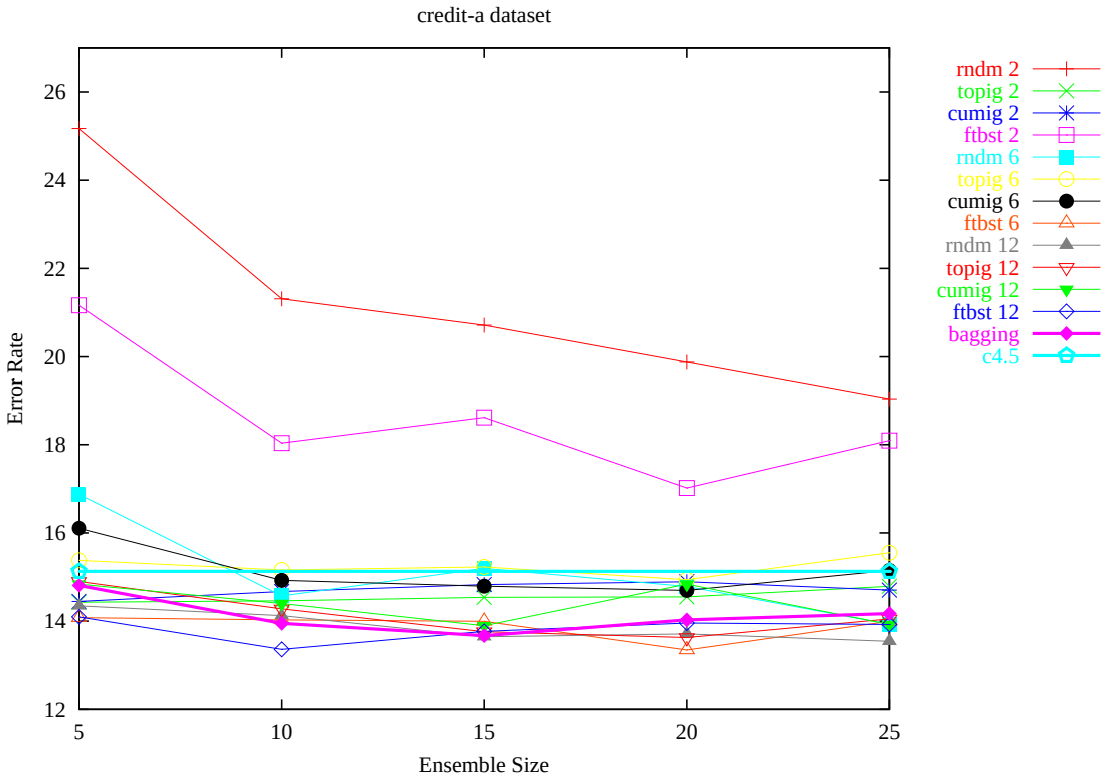


Figure 12: Comparison results similar to Figure 10 for *credit-a* dataset. The maximum number of possible features (used in a subset) is twelve. Note: the error rate shown on y-axis does not start from 0.

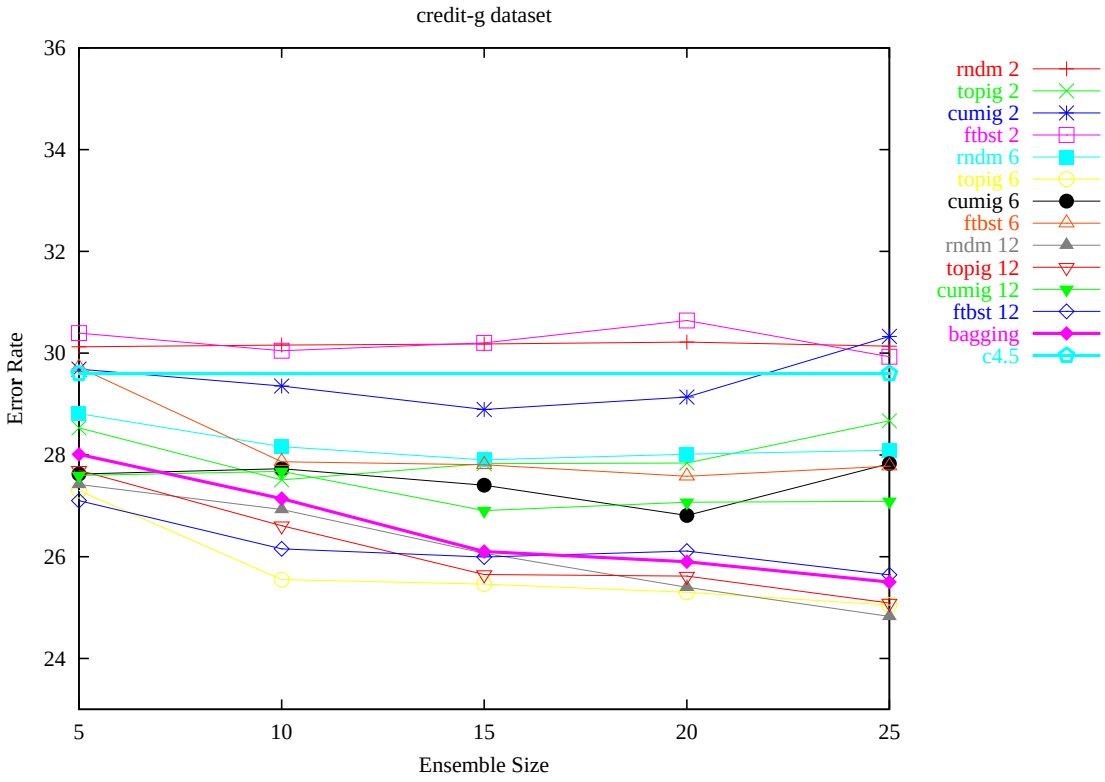


Figure 13: Comparison results similar to Figure 10 for *credit-g* dataset. The maximum number of possible features (used in a subset) is twelve. Note: the error rate shown on y-axis does not start from 0.

worked better for larger subset sizes. The information-gain based approach produced the best classifier for a subset size of six.

- *Glass* - This dataset (see Figure 14) contains nine input features. The random subset selection and feature boosting techniques were as good as baseline bagging for a subset size of seven. Cumulative information gain technique performed poorly for this dataset.

- *Hepatitis* - For this dataset (see Figure 15), random subset selection with six features produced the best results. The baseline bagging was not the best classifier. Feature boosting also produced good results especially for subset sizes of six and twelve. All feature subset selection algorithms (even for a subset size of two) outperformed the baseline C4.5 method.

- *House-votes-84* - The baseline algorithms (C4.5 and bagging) produce the best results in this case (see Figure 16). Feature boosting and cumulative information-gain based approaches using subset sizes of twelve produced comparable results. In general, most of the subset selection algorithms produced good results for larger subset sizes.

- *Hypo* - The baseline algorithms performed well in this case (see Figure 17). This was a case where feature boosting was not the best subset selection mechanism. Both the information-gain based approaches performed very well. They produced accuracies comparable to the baselines for larger subset sizes.

- *Ionosphere* - In this dataset (see Figure 18), the baseline bagging, the random subset selection and the feature boosting (both with a subset size of twelve) methods gave best results. For a subset size of two, all the subset selection techniques performed worse than the baseline C4.5 algorithm. Information-gain and cumulative information gain based feature selection did not work very well.

- *Kr-vs-kp* - This dataset consists of 36 features. The best performers were the two baseline algorithms, C4.5 and bagging with no feature subset selection (see Figure 19).

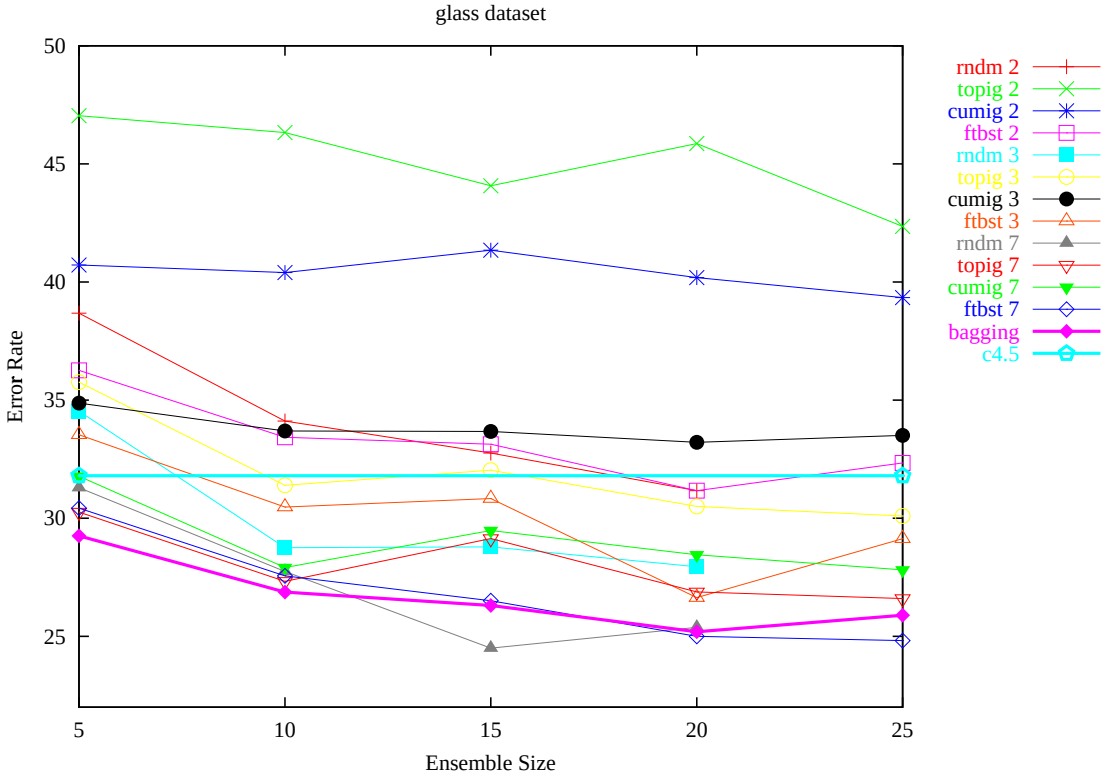


Figure 14: Comparison results similar to Figure 10 for *glass* dataset. The maximum number of possible features (used in a subset) is seven. Note: the error rate shown on y-axis does not start from 0.

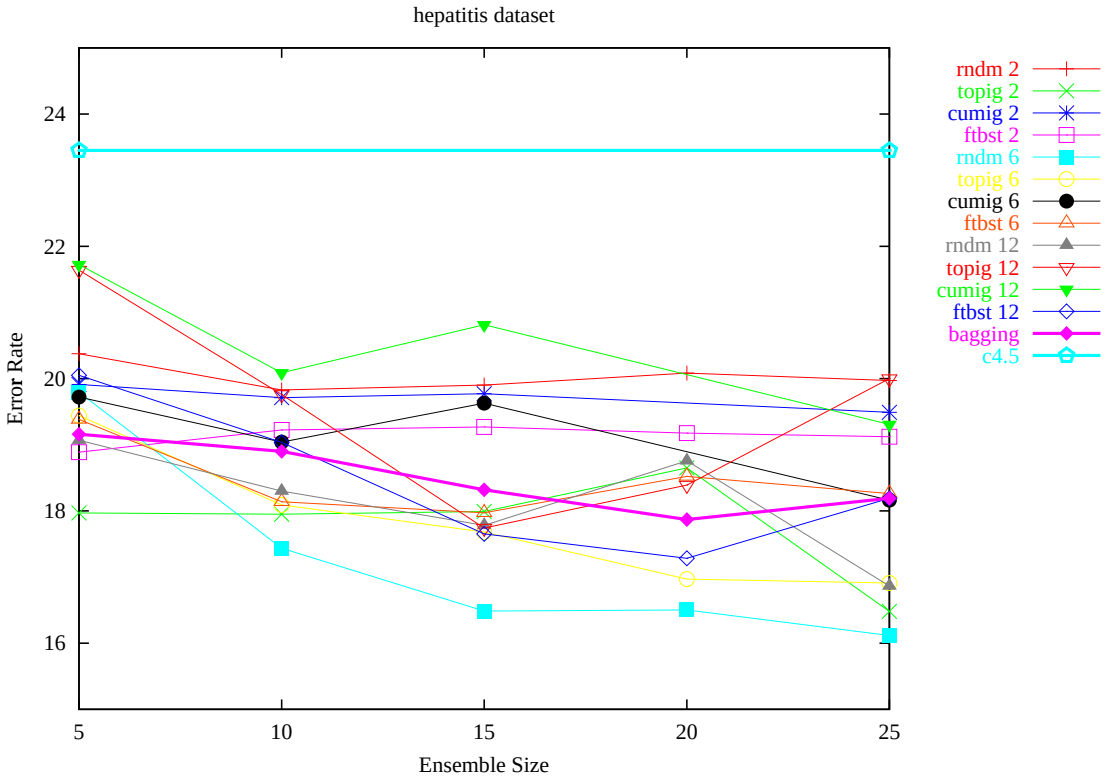


Figure 15: Comparison results similar to Figure 10 for *hepatitis* dataset. The maximum number of possible features (used in a subset) is twelve. Note: the error rate shown on y-axis does not start from 0.

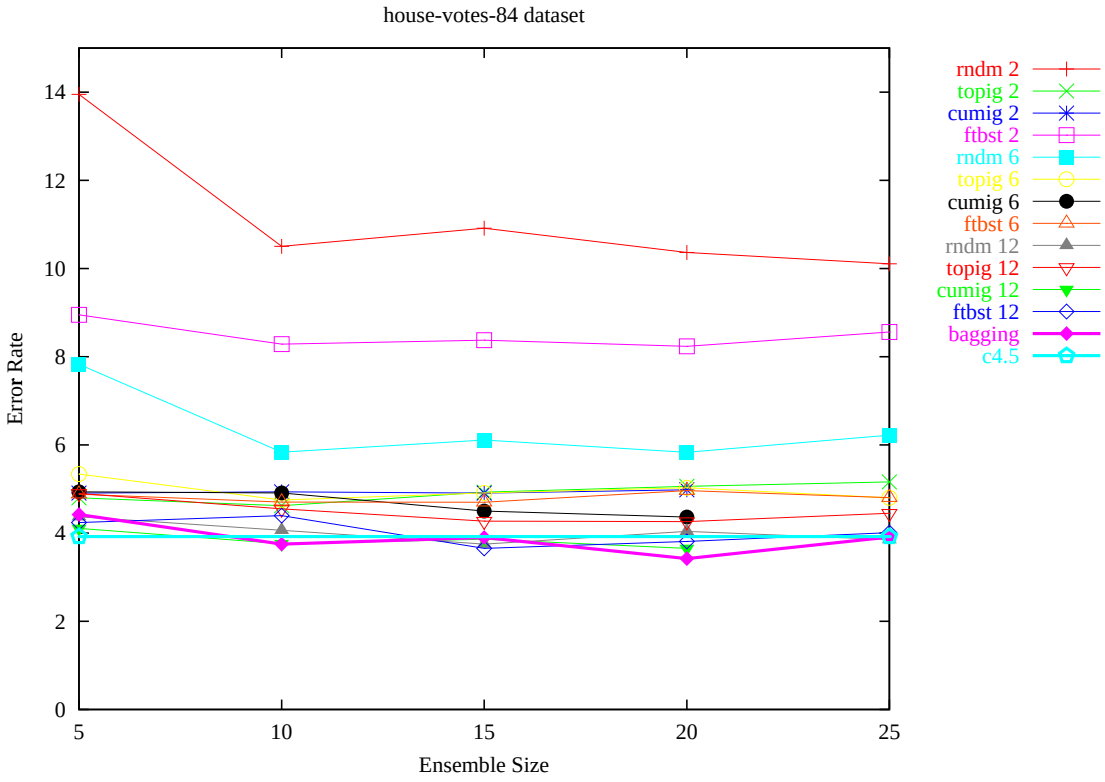


Figure 16: Comparison results similar to Figure 10 for *house-votes-84* dataset. The maximum number of possible features (used in a subset) is twelve.

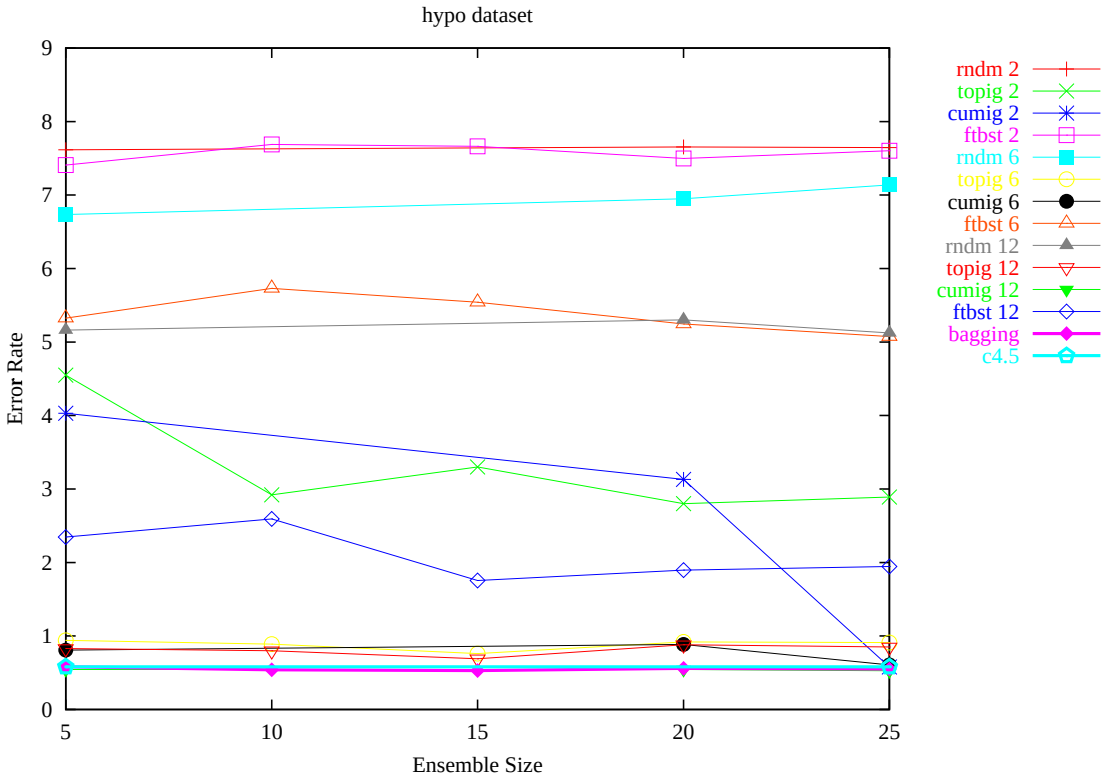


Figure 17: Comparison results similar to Figure 10 for *hypo* dataset. The maximum number of possible features (used in a subset) is twelve.

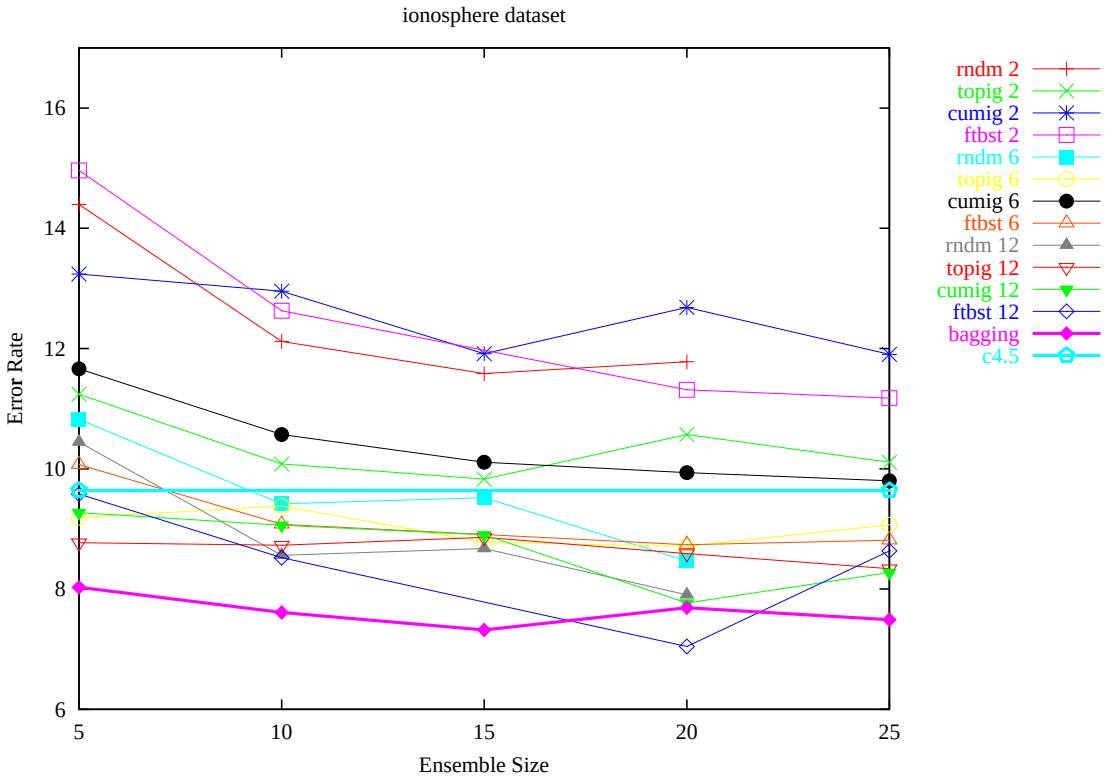


Figure 18: Comparison results similar to Figure 10 for *ionosphere* dataset. The maximum number of possible features (used in a subset) is twelve. Note: the error rate shown on y-axis does not start from 0.

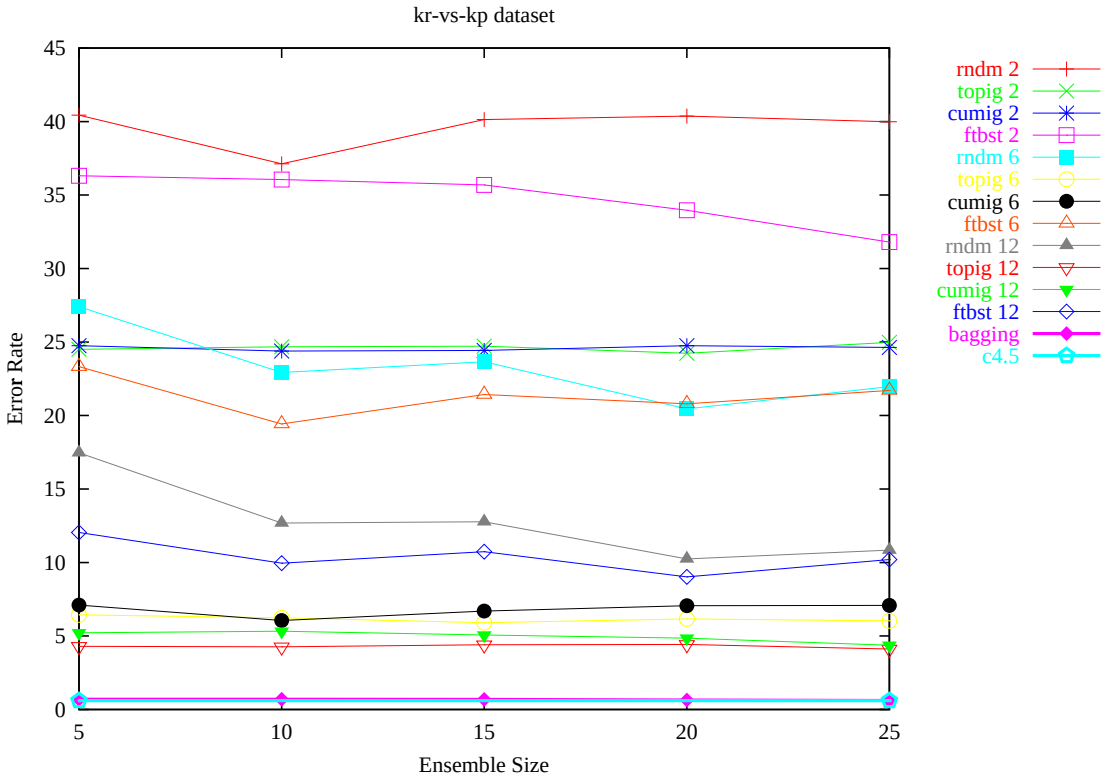


Figure 19: Comparison results similar to Figure 10 for *kr-vs-kp* dataset. The maximum number of possible features (used in a subset) is twelve.

All subset selection techniques gave much lower accuracies. Relatively though, cumulative information-gain based approach produced the best results of all the subset selection techniques. The information-gain based approach also performed relatively well. Random subset selection performed very badly on this dataset. Feature boosting performed moderately for larger subset sizes.

- *Labor* - This is a very small dataset (by the number of examples) with 16 features. For smaller subset sizes all the subset selection techniques did worse than C4.5 baseline (see Figure 20). The baseline bagging produced the best results. The feature boosting approach and cumulative information-gain based approaches produced good results for larger subsets.

- *Pima-indians-diabetes* - The results for random subset selection were interesting for this dataset (see Figure 21). Random subset selection for a subset size of two worked much better than many other approaches, in fact it was better than the selection of three features using random subset selection. Baseline bagging produced the best results. The information-gain based approaches and feature boosting produced comparable results to the baseline bagging.

- *Sick* - For this dataset (see Figure 22), baseline bagging produced the best results. Cumulative information-gain based feature selection produced accuracies comparable to the baseline bagging. Feature boosting failed to produce good results for this dataset. Random subset selection performed very poorly on this dataset.

Tables 5 - 7 show results of the various feature subset selection algorithms with the two baseline algorithms for a fixed ensemble size of twenty five. Each of the three tables show the error estimates for the four different feature subset selection algorithms on each of the datasets discussed in Table 4. Further, for comparison purposes we reproduce the error estimates of the two baseline algorithms in each of the three tables.

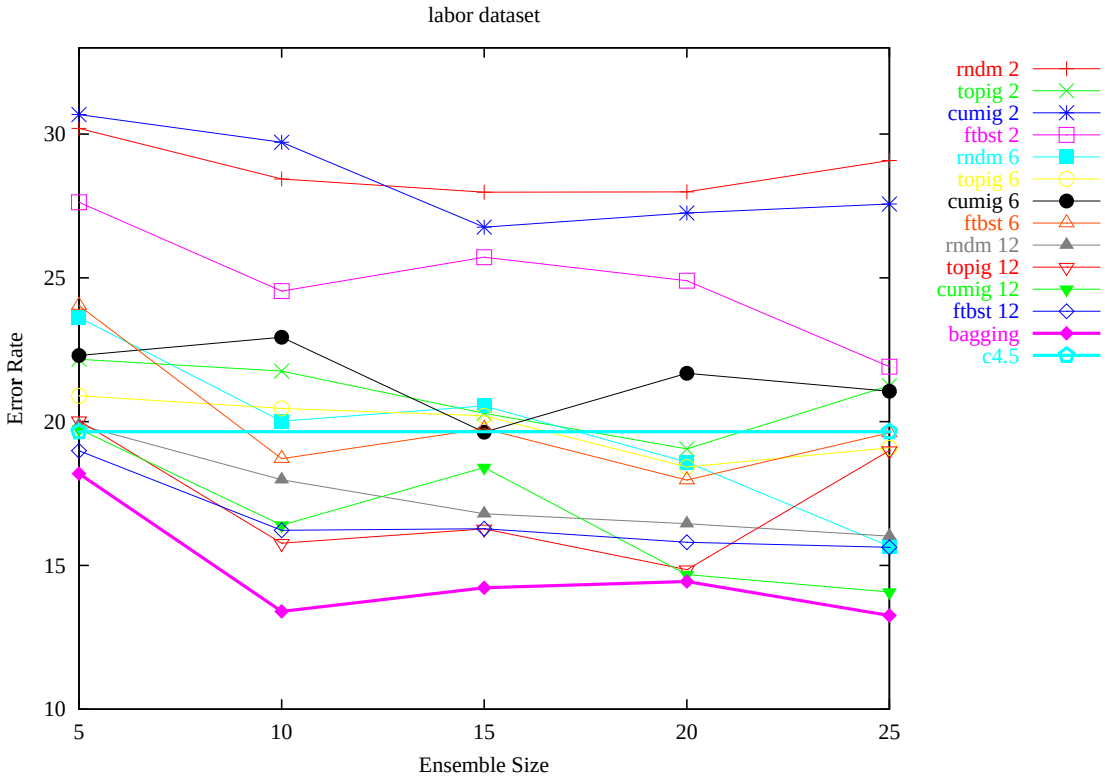


Figure 20: Comparison results similar to Figure 10 for *labor* dataset. The maximum number of possible features (used in a subset) is twelve. Note: the error rate shown on y-axis does not start from 0.

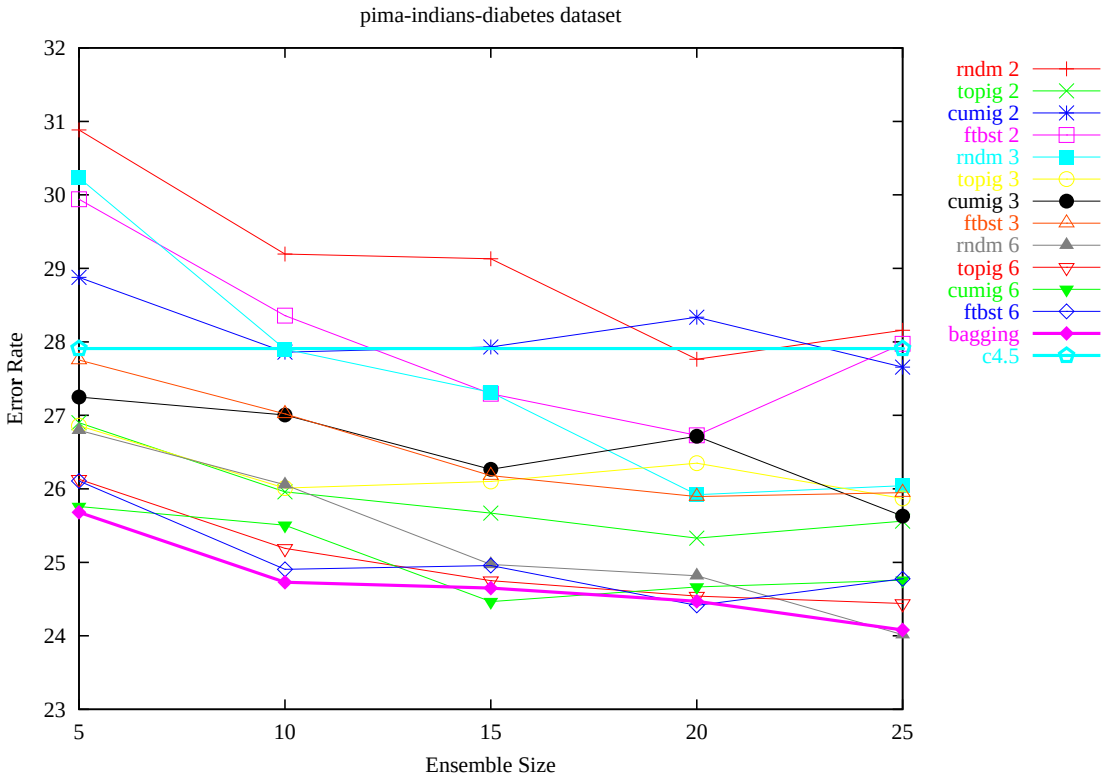


Figure 21: Comparison results similar to Figure 10 for *pima-indians-diabetes* dataset. The maximum number of possible features (used in a subset) is seven. Note: the error rate shown on y-axis does not start from 0.

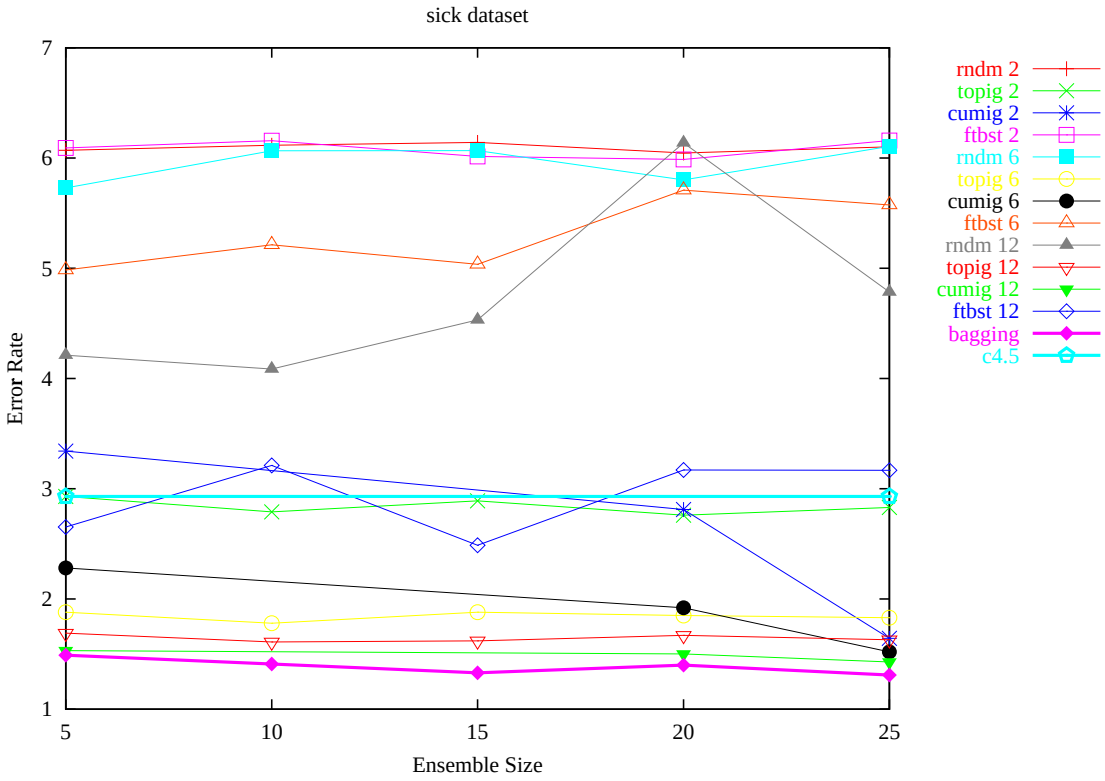


Figure 22: Comparison results similar to Figure 10 for *sick* dataset. The maximum number of possible features (used in a subset) is twelve. Note: the error rate shown on y-axis does not start from 0.

Table 5: Error estimates of the four subset selection techniques for the minimum possible subset size (number of features = 2). Ensemble size used for the following estimations is twenty five. The four feature selection techniques compared below are random subset selection (Random), information-gain based approach (I. Gain), cumulative information-gain based approach (C. Gain) and feature boosting (F. Boost). The asterisk (*) represents the best classification technique (baseline or feature subset selection technique) for a particular dataset.

Name of the Dataset	All Features		Subset of minimum size			
	Single DT	Bagging Ensemble	Random	I. Gain	C. Gain	F. Boost
cancer	6.50±0.53	4.40±0.68	3.51±0.37*	5.50±0.40	5.60±0.63	4.34±0.81
heart	25.68±1.75	20.33±1.02	19.86±2.53*	22.75±1.92	26.62±1.23	24.04±1.91
credit-a	15.13±0.90	14.17±0.95*	19.04±1.94	14.79±0.28	14.70±0.64	18.09±2.02
credit-g	29.60±0.66	25.50±0.65*	30.20±0.91	28.67±1.19	30.33±0.93	29.93±0.56
glass	31.80±2.62	25.89±1.92*	31.17±2.76	42.35±3.42	39.34±4.34	32.34±3.39
hepatitis	23.45±2.47	18.19±1.18	19.97±1.52	16.48±2.43*	19.49±2.83	19.12±1.65
vote	3.92±0.32	3.91±0.61*	10.11±0.47	5.16±0.21	4.98±0.37	8.56±1.61
hypo	0.58±0.06	0.55±0.05*	7.65±0.09	2.89±0.21	0.58±0.06	7.60±0.11
ionosphere	9.64±1.03	7.49±0.27*	11.78±1.08	10.11±0.88	11.9±1.52	11.18±1.21
kr-vs-kp	0.60±0.09*	0.61±0.09	39.99±1.57	24.97±0.72	24.63±0.56	31.80±1.32
labor	19.65±2.16	13.26±2.68*	29.08±3.12	21.25±2.33	27.57±4.85	21.91±4.83
diabetes	27.91±1.16	24.08±0.81*	28.16±1.05	25.56±0.96	27.66±1.92	27.97±1.43
sick	2.93±0.75	1.31±0.08*	6.10±0.12	2.83±0.14	1.64±0.57	6.16±0.10

Table 6: Error estimates of the various subset selection techniques for a medium sized subset (number of features ≤ 6). Results shown below are similar to those in Table 5.

Name of the Dataset	All Features		Subset of medium size			
	Single DT	Bagging Ensemble	Random	I. Gain	C. Gain	F. Boost
cancer	6.50±0.53	4.40±0.68	2.99±0.34*	4.39±0.64	4.45±0.49	4.34±0.44
heart	25.68±1.75	20.33±1.02	19.26±1.45	19.07±0.82*	22.29±2.37	20.27±2.14
credit-a	15.13±0.90	14.17±0.95	13.93±0.66*	15.55±0.57	15.15±0.85	14.00±0.77
credit-g	29.60±0.66	25.50±0.65	37.73±1.23	25.05±0.76*	27.83±1.23	27.78±0.94
glass	31.80±2.62	25.89±1.92*	27.95±1.60	30.09±1.63	33.51±2.68	29.13±2.22
hepatitis	23.45±2.47	18.19±1.18	16.12±2.10*	16.91±1.89	18.16±2.21	18.26±1.83
vote	3.92±0.32	3.91±0.61*	6.22±0.41	4.80±0.54	4.37±0.68	4.80±0.38
hypo	0.58±0.06	0.55±0.05*	7.14±0.26	0.91±0.13	0.61±0.07	5.08±0.72
ionosphere	9.64±1.03	7.49±0.27*	8.47±0.96	9.07±0.94	9.80±1.17	8.81±0.55
kr-vs-kp	0.60±0.09*	0.61±0.09	21.96±2.00	6.03±0.27	7.08±1.51	21.70±1.99
labor	19.65±2.16	13.26±2.68*	15.67±3.60	19.09±3.51	21.06±3.57	19.61±3.06
diabetes	27.91±1.16	24.08±0.81*	26.04±1.13	25.87±1.58	25.63±1.04	25.95±1.60
sick	2.93±0.75	1.31±0.08*	6.11±0.20	1.83±0.12	1.52±0.21	5.57±0.47

Table 7: Error estimates of the various subset selection techniques for the maximum possible subset size that was used during the evaluation (number of features ≤ 12). Results shown below are similar to those in Table 5.

Name of the Dataset	All Features		Subset of maximum size			
	Single DT	Bagging Ensemble	Random	I. Gain	C. Gain	F. Boost
cancer	6.50±0.53	4.40±0.68	3.45±0.4	4.02±0.34	3.59±0.32	3.40±0.31*
heart	25.68±1.75	20.33±1.02	19.72±1.09	19.17±1.57*	20.61±1.69	19.38±1.16
credit-a	15.13±0.90	14.17±0.95	13.54±1.04*	14.05±0.70	13.93±0.63	13.93±0.27
credit-g	29.60±0.66	25.50±0.65	30.99±0.99	25.38±0.59*	27.09±1.39	25.64±0.60
glass	31.80±2.62	25.89±1.92	25.38±1.91	26.60±1.60	27.81±3.46	24.82±1.63*
hepatitis	23.45±2.47	18.19±1.18	16.87±2.33*	20.00±3.19	19.31±2.18	18.19±1.94
vote	3.92±0.32	3.91±0.61	3.86±0.63	4.45±0.63	3.66±0.50*	4.01±0.52
hypo	0.58±0.06	0.55±0.05	5.12±0.22	0.85±0.12	0.53±0.15*	1.95±0.57
ionosphere	9.64±1.03	7.49±0.27*	7.91±0.93	8.34±0.87	8.28±1.05	8.63±0.95
kr-vs-kp	0.60±0.09*	0.61±0.09	10.85±1.15	4.11±0.37	4.38±0.76	10.20±1.59
labor	19.65±2.16	13.26±2.68*	16.02±2.79	18.99±4.73	14.08±2.79	15.63±2.80
diabetes	27.91±1.16	24.08±0.81	24.02±1.12*	24.44±1.11	24.76±1.17	24.78±1.33
sick	2.93±0.75	1.31±0.08*	4.78±0.21	1.63±0.10	1.43±0.21	3.17±0.53

4.4 Discussion

An overall analysis of the results reveals the fact that in many cases we can obtain more accurate ensemble learners by the use of feature subset selection. In many cases, the ensemble learner that uses all the features is not the best learner.

We now discuss below the relative performances of each of the four different feature selection techniques. We further discuss the possible implications of their performances.

- *Random Subset Selection* - This is the most trivial form of feature subset selection technique used. Its results were good for some datasets (cleveland-heart, breast-cancer-wisconsin, hepatitis and glass). Random selection performs poorly in some other cases (kr-vs-kp, sick and hypo).

  The randomly selected subsets of features for each individual classifiers of the bagging ensemble might result in an increase in the diversity among these classifiers which results in the formation of a good ensemble. This in turn may result in lower error. This observation assumes that each randomly generated subset contains some relevant features that lead to the creation of *good* individual classifiers. This might not be the case in the presence of many irrelevant features. This can be observed by the bad performance of this technique on datasets like kr-vs-kp. But, as we have used many datasets that have relatively few features, random selection performed very well.

- *Information-Gain Based Subset Selection* - This approach performed moderately well on some datasets (credit-g and pima-indian-diabetes). Its performance was very good on large datasets like kr-vs-kp and sick. Most of the times, better results were produced using the cumulative information-gain approach. In some of the datasets, this approach produced good results for larger subset sizes.

- *Cumulative Information-Gain Based Subset Selection* - This approach performed well on some of the datasets (house-votes-84 and hypo). It performed very well on sick, kr-vs-kp and labor. Datasets for which cumulative information-gain based approach failed to perform well are glass, cleveland-heart and breast-cancer-wisconsin. This

technique performs better for larger subset sizes than for smaller ones. But there were many cases when the accuracies of the ensemble using this feature selection technique were better than the accuracies of the baselines. By observing the results, we can find a close relation between this approach and the information-gain based approach. This is plausible for two reasons: (1) both these approaches are based upon the common notion of information gain and (2) these approaches select a particular subset that will be used by all the classifiers of the ensemble. This may reduce the degree of diversity among the classifiers.

- *Feature Boosting Methodology* - Our approach to feature selection did well in a number of cases. It outperformed the bagging ensemble that used all the features in many cases (credit-a, cleveland-heart, hepatitis, ionosphere and labor). But feature boosting performed poorly on kr-vs-kp and sick. Our feature boosting approach was more consistent (independent of the dataset) than the above simpler techniques. In many cases, feature boosting performed better than other techniques for a smaller number of features. This might be partially due to the use of a feature weighting technique in this approach while in other cases we were selecting a definitive subset to be used by the ensemble.

We have elicited some of the important observations with regard to each of the subset selection techniques. We also infer that subset selection using ensembles in many cases result not only in the reduction of computation time but also a reduction in error. In general, our feature boosting method performed well when compared to the other feature selection techniques that we developed. In the next chapter, we will attempt to answer the questions that we posed in our introduction. We will further discuss the possible future work in this area.

5 Conclusions

Feature subset selection is an important problem in the field of machine learning. The question dealt with in this thesis is to investigate the use of ensemble learning in feature subset selection. We used bagging ensemble learners for our work. Further details of the different methodologies implemented are provided in the next section.

5.1 Thesis Questions

In this thesis we defined an ensemble-based approach to feature subset selection. The basic idea is to provide each of the individual classifiers in the ensemble a subset of features along with the training data. Each classifier is trained using only the information provided by the corresponding subset of features. One important aspect of this thesis involved developing strategies for the selection of the subsets for the ensemble components. We implemented four different feature selection methods: (1) random subset selection, (2) an information gain based approach, (3) a more complex method that we call cumulative information gain and (4) a feature boosting methodology.

We compared these approaches to each other and to a baseline C4.5 and bagging ensemble approach. All the experiments were conducted using datasets that were drawn from the UCI repository.

Though it was expected that random selection would perform badly, to our surprise it performed comparatively well in some cases but this was dependent on the dataset being tested. In larger ensembles, the effect of increasing the size of the ensemble had less impact on the accuracy. Feature boosting seemed to achieve considerable improvements for smaller values of subset size in many cases.

Now, we will try to answer the questions that we posed in the first chapter.

1. *Can the ensemble methodology be exploited to do feature subset selection?*

   Ensemble learning can often be used to achieve better results in the area of feature subset selection. An ensemble can use a very small subset of features and still produce good results. Since the ensembles can use a small subset of features, there can be more

differences between the individual classifiers of the ensemble, a requirement for *good ensembles*. This can be clearly observed from the results obtained for the various feature selection techniques.

2. *How well does the feature boosting selection algorithm perform? Compare its accuracies with the baseline versions like the single C4.5 decision tree algorithm and a bagging ensemble that uses all the attribute information.*

   Figures 10 - 22 show the comparisons of the different feature subset selection algorithms with the baseline C4.5 and bagging ensembles. The four feature subset selection strategies seemed to perform better than the baseline algorithms in many cases for a nominal size of the feature subsets. Feature boosting appears to perform consistently well for larger subset sizes.

3. *Are there some simple techniques to achieve good feature subset selection for ensemble learning?*

   To answer to this question, we developed three other simple subset selection techniques: random subset selection, an information-gain based approach and a cumulative information-gain based approach. The selected feature distributions were tested on the bagging ensemble and they were compared with the results of feature boosting. Figures 10 - 22 depict the comparisons on each of the datasets. On average, the random subset selection and feature boosting approach perform well. The reasons for the *good* performance of the random selection technique might be credited to the ability to create *diverse* individual classifiers as a part of the ensemble (see section 4.4 for more details). The cumulative information-gain based approach performed very well on some of the datasets. So, simple techniques do work decently on ensembles but their performance is dependent on the dataset.

5.2 Future Work

Our new approach to feature subset selection suggests that even with a small number of features, an ensemble approach can be modified to achieve accuracies that are comparable

to the ensemble that uses no subset selection. This suggests that ensemble learning can be used for feature subset selection. Further, the use of a *judiciously* chosen distribution of features can lead to ensembles with better performance than the original baseline bagging ensemble with all the features chosen. A number of areas of future research are suggested from this work:

- Evaluation on larger datasets with larger number of features - the datasets that we have considered in the course of evaluation have a maximum of 36 features (kr-vs-kp dataset). This method can be tested on much larger datasets that have a larger number of features. This might provide more insight into the performance of each of these subset selection techniques.

- Comparison to the wrapper approach - in the present thesis, we have not been able to evaluate the performance of the ensemble method with any of the existing techniques. This partly is due to the lack of a proper *common* scale of performance. Existing wrapper techniques do not use ensemble learning (which is a powerful form of learning) and it was generally observed that in many cases, our approach achieved higher accuracies than the existing techniques. But this was possible because of the use of ensemble learning approach. A wrapper approach is closely related to our approach. It does not keep the intermediate classifiers generated or maintain a history of the accuracies obtained during the previous iterations of its feature search process. In our approach, we keep all the classifiers and maintain a history and we use the history to calculate the new mean and standard deviation for each iteration. We use this mean and standard deviation in our process of weight update. Hence, it might be possible to compare our approach to the wrapper technique.

- Optimal subset size determination - our present learning model does not attempt to determine an appropriate feature subset size. It is rather a system where the user specifies the required subset size. It would be interesting to see how the present feature boosting technique can be modified to find the appropriate subset size.

- Boosting pairs (or higher) combinations of features - features in a feature set might not be independent. Recognizing features that interact is another important problem area in feature subset selection. In our present approach, we create the weighted distribution of features independent of the fact that such relationships might exist. Our present approach can be thought of as a framework for research in this direction. Creation of such combinations of features and assignment of a combined weight to them would lead to better feature subset selection. This could be further be extended to finding all such possible correlated feature subsets.

References

[Almuallim and Dietterich, 1991] Almuallim, H. and Dietterich, T. G. (1991). Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, volume 2, pages 547–552, Anaheim, California.

[Bauer and Kohavi, 1999] Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105.

[Blake and Merz, 1998] Blake, C. and Merz, C. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[Blum and Langley, 1997] Blum, A. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97(1–2):245–271.

[Breiman, 1996a] Breiman, L. (1996a). Bagging predictors. *Machine Learning*, 24(2):123–140.

[Breiman, 1996b] Breiman, L. (1996b). Bias, variance and arcing classifiers. Technical report, University of California at 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, Belmont, CA.

[Fayyad, 1991] Fayyad, U. M. (1991). *On the induction of decision trees for multiple concept learning*. PhD thesis, Electrical Engineering and Computer Science Department, University of Michigan.

[Freund and Schapire, 1996] Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 148–146, San Francisco, CA.

[Kira and Rendell, 1992] Kira, K. and Rendell, L. A. (1992). A practical approach to feature selection. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 249–256, Aberdeen, Scotland.

[Kohavi, 1995] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 1137–1145, San Francisco, CA.

[Kohavi and John, 1997] Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1–2):273–323.

[Krogh and Vedelsby, 1995] Krogh, A. and Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In Tesauro, G., Touretzky, D., and Leen, T., editors, *Advances in Neural Information Processing Systems*, volume 7, pages 231–238, Cambridge, MA. MIT Press.

[Maclin and Opitz, 1997] Maclin, R. and Opitz, D. (1997). An empirical evaluation of bagging and boosting. *Journal of AI Research*, 11:169–198.

[Mitchell, 1977] Mitchell, T. M. (1977). Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 305–310, Cambridge, MA.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine Learning.* WCB/McGraw-Hill, Boston, MA.

[Quinlan, 1987] Quinlan, J. R. (1987). Rule induction with statistical data–A comparison with multiple regression. *Journal of the Operational Research Society*, 38:347–352.

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

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA.

[Runyon, 1977] Runyon, K. E. (1977). *Consumer Behavior and the Practice of Marketing.* Charles E. Merrill Publishing Company, Columbus, Ohio.