University of Alberta

DECISION TREE INSTABILITY AND ACTIVE LEARNING

by

Kenneth David Dwyer

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Spring 2007

Your file Votre référence
ISBN: 978-0-494-29954-8
Our file Notre référence
ISBN: 978-0-494-29954-8

Use what talents you possess; the woods would be very silent
if no birds sang except those that sang best.

– Henry van Dyke

Abstract

The instability of learning algorithms is an important topic that is often overlooked in the field of machine learning. An unstable learner is one that may yield dramatically different classifiers from training sets that differ just slightly. Decision tree learning algorithms produce accurate models that can be interpreted by a domain expert with relative ease. However, these algorithms are known to be highly unstable, which undermines the common objective of extracting knowledge from the tree structures they create.

This thesis examines the instability of the C4.5 decision tree learner in two contexts: passive learning and active learning. An alternative splitting criterion is tested with C4.5, in order to determine whether it improves the stability of the algorithm. Several active learning methods that use C4.5 as a component learner are compared with respect to the stability and accuracy of the decision trees they produce.

Acknowledgements

I would first of all like to take this opportunity to thank my supervisor, Dr. Robert Holte, for opening my mind to many new ideas and approaches to solving problems. It has been both a tremendous learning experience and a true pleasure to work with Rob; I invariably left his office each week with more enthusiasm and a clearer sense of purpose than I had possessed prior to our discussion. I could not have asked for more than this.

David Woloshuk at the Alberta Ingenuity Centre for Machine Learning (AICML) has gone out of his way on a number of occasions to ensure that any obstacles that were encountered while conducting these experiments were dealt with promptly and effectively; for this excellent service, I offer my sincere thanks.

Finnegan Southey was involved in my research for several months before leaving the department, and provided some valuable input during this time. More importantly though, I wish to thank Finnegan for initially meeting, at my request, to tell me about his work and his interest in machine learning. I look back on this short chat as having considerable influence on my eventual decision to pursue research in this field.

I would like to thank Milan Krneta, Gary Paterson, and Scott Wardle at Electronic Arts for guiding me through the labyrinth that is the FIFA 2004 source code. A special thank you also goes out to Vadym Voznyuk at Electronic Arts, who contributed in a number of ways to the production of the two soccer game datasets.

Last, but certainly not least, my family, friends, and loved ones have been incredibly supportive; I could not have reached this point without their help and encouragement. In particular, Irina has been my inspiration, and gave me strength at times when my morale was sinking. I cannot put into words the positive effect of her kindness and consideration, as well as her "go get 'em" attitude, on my ability to complete this thesis.

The financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC), the Informatics Circle of Research Excellence (iCORE), and the University of Alberta is gratefully acknowledged.

Table of Contents

1 Introduction 1
- 1.1 Problem Definition . . . 1
- 1.2 Approach to the Problem . . . 2
- 1.3 Contributions of this Research . . . 3
- 1.4 Outline . . . 4

2 Decision Tree Instability 5
- 2.1 Decision Tree Classifiers . . . 5
- 2.2 Decision Tree Splitting Criteria . . . 7
- 2.3 Instability of Decision Tree Learning Algorithms . . . 9
  - 2.3.1 Learner Instability and its Effects . . . 9
  - 2.3.2 Causes of Decision Tree Instability . . . 12
- 2.4 Quantifying Stability . . . 13
  - 2.4.1 Variance in Error Rate Estimates as Instability . . . 14
  - 2.4.2 Measuring the Semantic Stability of Learners . . . 14
  - 2.4.3 Measuring the Structural Stability of Decision Trees . . . 14
  - 2.4.4 The Region Stability Metric . . . 16
- 2.5 Decision Tree Stability Experiments . . . 19
  - 2.5.1 Datasets . . . 19
  - 2.5.2 Experimental Procedure . . . 20
  - 2.5.3 C4.5 and the DKM Splitting Criterion . . . 22
- 2.6 Experimental Results . . . 24
  - 2.6.1 Predictive Accuracy, Tree Size, and Semantic Stability . . . 24
  - 2.6.2 Absolute Structural Stability . . . 28
  - 2.6.3 Approximate Structural Stability . . . 34
  - 2.6.4 Discussion . . . 37
- 2.7 Conclusions . . . 39

3 Stability in Active Learning 41
- 3.1 Selective Sampling and Stability . . . 42
- 3.2 Selective Sampling Methods . . . 48
- 3.3 Experimental Procedure . . . 54
- 3.4 Experimental Results . . . 57
  - 3.4.1 Predictive Accuracy . . . 57
  - 3.4.2 Semantic Stability . . . 62
  - 3.4.3 Absolute Structural Stability . . . 66
  - 3.4.4 Approximate Structural Stability . . . 72
  - 3.4.5 Comparison of Splitting Criteria . . . 79
- 3.5 Conclusions . . . 82

4 Region-based Active Learning 84
- 4.1 Region-based Selective Sampling with Decision Trees . . . 85
- 4.2 Merging Decision Trees . . . 86
  - 4.2.1 The TreeMerge Algorithm . . . 87
  - 4.2.2 Summary . . . 91
- 4.3 Simplifying the Joint Committee Hypothesis . . . 91
  - 4.3.1 Issues with Tiny Decision Regions . . . 92
  - 4.3.2 Threshold Alignment . . . 93
- 4.4 Experimental Procedure . . . 98
- 4.5 Experimental Results . . . 99
  - 4.5.1 Classification Accuracy . . . 100

4.5.2 Execution Time and Hypothesis Complexity . . . 106
4.5.3 Increasing the Minimum Allowable Decision Region Size . . . 110
4.6 Region-based Query Learning . . . 112
4.6.1 Query Learning Versions of Region-based Algorithms . . . 112
4.6.2 Experiments and Discussion . . . 113
4.7 Conclusions . . . 115

5 Related Work **117**
5.1 Improving the Stability of Decision Trees . . . 117
5.2 Active Learning . . . 119

6 Conclusions **120**
6.1 Summary . . . 120
6.2 Limitations and Future Work . . . 121
6.3 Final Word . . . 124

Bibliography **125**

A Dataset Information **128**
A.1 UCI Datasets . . . 128
A.2 FIFA Soccer Datasets . . . 130

B Estimating a C4.5 Penalty Term for the DKM Criterion **132**

C Additional Experimental Data **137**
C.1 Data from Selective Sampling Experiments . . . 137
C.2 Data from Query Learning Experiments . . . 147

List of Tables

2.1 UCI datasets containing discrete attributes only 20
2.2 UCI datasets containing continuous attributes only 21
2.3 UCI datasets containing containing a mixture of continuous and discrete attributes . 21
2.4 Mean semantic stability scores, error rates, and tree sizes for the entropy and DKM splitting criteria 26
2.5 Mean stability scores for datasets containing discrete attributes only 29
2.6 Mean stability scores for datasets containing at least one continuous attribute (1) . . 30
2.7 Mean stability scores for datasets containing at least one continuous attribute (2) . . 31

3.1 Weighted error rates (entropy) 58
3.2 Weighted error rates (DKM) 58
3.3 Weighted semantic FinalStab scores (entropy) 63
3.4 Weighted semantic FinalStab scores (DKM) 63
3.5 Weighted semantic PrevStab scores (entropy) 65
3.6 Weighted semantic PrevStab scores (DKM) 65
3.7 Weighted structural FinalStab scores (entropy, $\varepsilon_{pct} = 0$) 67
3.8 Weighted structural FinalStab scores (DKM, $\varepsilon_{pct} = 0$) 67
3.9 Weighted structural PrevStab scores (entropy, $\varepsilon_{pct} = 0$) 69
3.10 Weighted structural PrevStab scores (DKM, $\varepsilon_{pct} = 0$) 69
3.11 Weighted structural RunStab scores (entropy, $\varepsilon_{pct} = 0$) 71
3.12 Weighted structural RunStab scores (DKM, $\varepsilon_{pct} = 0$) 71
3.13 Structural FinalStab win-loss-draw counts versus Random (entropy) 76
3.14 DKM weighted FinalStab scores subtracted from entropy scores ($\varepsilon_{pct} = 5\%$) . . . 80
3.15 DKM weighted PrevStab scores subtracted from entropy scores ($\varepsilon_{pct} = 5\%$) 80
3.16 DKM weighted RunStab scores subtracted from entropy scores ($\varepsilon_{pct} = 5\%$) 81

4.1 The minimum permitted decision region widths (λ_{pct}) that were used for the experiments described in Section 4.4 100
4.2 Weighted error rates 101
4.3 Average fraction of examples in the training set that belong to the positive class . . 105
4.4 Average execution times for committee building and example selection 107
4.5 Average number of decision regions in the committee and in the merged tree 107
4.6 Average time taken to classify the unlabelled pool examples 110
4.7 Weighted error rates 111

A.1 Datasets generated using the FIFA 2004 soccer game 131

B.1 Comparison between no penalty and the estimated DKM penalty 135
B.2 Comparison between the entropy penalty and the estimated DKM penalty 135

C.1 Weighted structural FinalStab scores (entropy, $\varepsilon_{pct} = 5\%$) 138
C.2 Weighted structural FinalStab scores (DKM, $\varepsilon_{pct} = 5\%$) 138
C.3 Weighted structural PrevStab scores (entropy, $\varepsilon_{pct} = 5\%$) 139
C.4 Weighted structural PrevStab scores (DKM, $\varepsilon_{pct} = 5\%$) 139
C.5 Weighted structural RunStab scores (entropy, $\varepsilon_{pct} = 5\%$) 140
C.6 Weighted structural RunStab scores (DKM, $\varepsilon_{pct} = 5\%$) 140
C.7 Weighted structural FinalStab scores (entropy, $\varepsilon_{pct} = 10\%$) 141
C.8 Weighted structural FinalStab scores (DKM, $\varepsilon_{pct} = 10\%$) 141
C.9 Weighted structural PrevStab scores (entropy, $\varepsilon_{pct} = 10\%$) 142
C.10 Weighted structural PrevStab scores (DKM, $\varepsilon_{pct} = 10\%$) 142

C.11 Weighted structural RunStab scores (entropy, $\varepsilon_{pct} = 10\%$) 143
C.12 Weighted structural RunStab scores (DKM, $\varepsilon_{pct} = 10\%$) 143
C.13 Weighted structural FinalStab scores (entropy, $\varepsilon_{pct} = 15\%$) 144
C.14 Weighted structural FinalStab scores (DKM, $\varepsilon_{pct} = 15\%$) 144
C.15 Weighted structural PrevStab scores (entropy, $\varepsilon_{pct} = 15\%$) 145
C.16 Weighted structural PrevStab scores (DKM, $\varepsilon_{pct} = 15\%$) 145
C.17 Weighted structural RunStab scores (entropy, $\varepsilon_{pct} = 15\%$) 146
C.18 Weighted structural RunStab scores (DKM, $\varepsilon_{pct} = 15\%$) 146
C.19 Weighted error rates 147

List of Figures

2.1 Pruned tree for the lymphography dataset, trained on 106 examples 11
2.2 Pruned tree for the lymphography dataset, trained on 107 examples 11
2.3 Two trees grown using the DKM criterion from the glass dataset 18
2.4 ECDFs of the paired semantic stability score differences on the australian and anneal datasets 27
2.5 ECDFs of the paired semantic stability score differences on the heart-h and nursery datasets 28
2.6 ECDF of the paired structural stability score differences ($\varepsilon_{pct} = 0$) on the australian dataset 32
2.7 ECDFs of the paired semantic stability score differences and the paired structural stability score differences ($\varepsilon_{pct} = 0$) on the breast-cancer dataset 32
2.8 Structural stability scores on the vowel dataset as a function of ε_{pct}, and as an ECDF of the paired score differences ($\varepsilon_{pct} = 5\%$) 34
2.9 Structural stability scores on the pendigits dataset as a function of ε_{pct}, and as an ECDF of the paired score differences ($\varepsilon_{pct} = 5\%$) 35
2.10 Structural stability scores on the anneal dataset as a function of ε_{pct}, and as an ECDF of the paired score differences ($\varepsilon_{pct} = 12\%$) 36
2.11 Mean structural stability score versus ε_{pct} on the australian and letter datasets . . . 36
2.12 Pairs of trees grown on the lymphography dataset, first using DKM, and then using the entropy criterion 38

3.1 Active learning stability calculated using the FinalStab metric 45
3.2 Active learning stability calculated using the PrevStab metric 46
3.3 Active learning stability calculated using the RunStab metric 47
3.4 Median structural FinalStab scores on the kr-vs-kp dataset (DKM, $\varepsilon_{pct} = 0$) 56
3.5 Mean error rates on the wdbc and australian datasets (DKM) 59
3.6 Mean error rates forming a "banana" shape, indicating efficient use unlabelled data by the selective sampling methods 61
3.7 Median semantic FinalStab scores on the vowel and tic-tac-toe datasets (entropy) . 64
3.8 Median semantic PrevStab scores on the german and wdbc datasets (entropy) . . . 64
3.9 Median structural FinalStab scores on the nursery and australian datasets (entropy, $\varepsilon_{pct} = 0$) 68
3.10 Median structural PrevStab scores on the pendigits dataset (entropy, $\varepsilon_{pct} = 0$) . . . 70
3.11 Structural RunStab scores on the pendigits and vowel datasets (entropy, $\varepsilon_{pct} = 0$) . 72
3.12 Median structural FinalStab scores on the segmentation dataset (DKM, $\varepsilon_{pct} = 0$ and $\varepsilon_{pct} = 5\%$) 73
3.13 Median structural PrevStab scores of Uncertainty on the pendigits dataset (entropy, $\varepsilon_{pct} \in \{0, 5\%, 10\%\}$) 75
3.14 Median structural PrevStab and FinalStab scores on the letter dataset (entropy, $\varepsilon_{pct} = 5\%$) 77
3.15 Mean error rates on the letter dataset (entropy) 78
3.16 Median structural PrevStab and FinalStab scores on the german dataset when using the QBag method ($\varepsilon_{pct} = 5\%$) 81

4.1 The merging of two simple decision trees 92
4.2 Merged tree representations of bagging committees created from the corner-kick dataset, with and without λ-Alignment 97
4.3 Mean error rates on the segmentation dataset 101
4.4 Comparison of score distributions defined by BootM and BootM-rb-min on the vowel dataset 103
4.5 Mean error rates on the vowel dataset, for the BootM and BootM-rb-min methods . 103

4.6 Average fraction of examples in the training set belonging to the positive class, on the vowel and pima-indians datasets 105
4.7 Average execution times on the corner-kick and give-and-go datasets 108
4.8 Mean error rates on the letter and german datasets 111
4.9 Average fraction of examples in the training set belonging to the positive class, on the corner-kick and wdbc datasets 114

B.1 Estimated DKM penalty as a function of the entropy penalty 133

Chapter 1

Introduction

1.1 Problem Definition

In the field of machine learning, the instability of learning algorithms is an important, yet often overlooked issue. A learning algorithm is said to be unstable if it produces drastically different hypotheses from training sets that differ just slightly. Decision tree learners (e.g. [7, 42]) constitute one of the most widely used and well studied classes of learning algorithms. Unlike some learning representations, such as the artificial neural network, whose prediction model is difficult, if not impossible for a human to comprehend, a decision tree forms an intuitive series of "If-Then" rules describing the relationship between the features of an entity and its predicted class or category. Indeed, the relative ease with which a decision tree can be interpreted is one of its most attractive qualities. However, decision tree learners are known to be highly unstable procedures, which undermines the common objective of extracting knowledge from the classifiers that they produce.

Decision tree learners are also used as component learning algorithms within the active learning framework. Active learning is an iterative procedure, in which the learner plays a role in choosing the examples that it uses to train a classifier; this is in contrast to passive learning, in which training examples are presented to the learner at random. Decision tree instability is also a concern in the context of active learning, since the tree learner may alter its hypothesis substantially whenever new examples are labelled and added to the training set. One implication of this instability is that when the active learning procedure is finished, the resulting decision tree may be an artifact of the particular training set that was formed during the final iteration of learning. In other words, if a few additional examples had been labelled, a significantly different tree may have been produced instead. Once again, this impairs the goal of knowledge acquisition, since it is not known whether the tree actually represents the target concept, or if it is merely the result of variations in the training data.

This thesis studies the instability of the C4.5 decision tree learning algorithm [42] in both passive and active learning settings. An alternative splitting criterion is tested in order to determine whether it improves the stability of C4.5, and several active learning methods that use C4.5 as a base learner

are compared with respect to the accuracy and stability of the trees they produce. This research is important because decision tree algorithms are frequently used with little or no regard for their instability; it is therefore essential that this instability be quantified, and techniques for improving stability be explored.

In addition, there have been very few studies performed that have dealt with the issue of training a single decision tree classifier via active learning. More commonly, committees of decision trees have been induced in this manner (e.g. [1]). However, while a tree committee is generally more accurate than any single tree, such a classifier is incomprehensible, and thus sacrifices one of the attractive properties of the decision tree representation. In this work, various active learning algorithms are applied to the task of inducing a single tree, and some methods are shown to perform significantly better than others in this context – a result that has not previously been reported. Finally, a novel approach to active learning with decision trees is proposed and evaluated in this thesis that is competitive with, and in some cases superior to, existing algorithms. The introduction of this technique opens up a number of interesting directions for future research involving decision trees and active learning.

1.2 Approach to the Problem

The instability of decision tree learners is studied empirically in this thesis. Two types of stability are measured: *semantic* and *structural* stability. Semantic stability measures the degree to which two classifiers make the same predictions, whereas structural stability measures the similarity between particular structural properties of two trees. An existing measure, known as expected agreement [52], is used to quantify semantic stability, and a novel measure of structural decision tree stability is employed, which is called the region stability metric. All experiments are conducted using a representative collection of benchmark datasets from the UCI machine learning repository [39].

In the passive learning setting, a pair of decision trees is induced from a random division of a given dataset into two halves, and the stability of these two trees is measured. Within the context of active learning, the learner is presented with a small set of randomly chosen examples that serves as the initial training set. A separate set of examples have their labels removed; these constitute the pool of unlabelled examples that is made available to the learner. Then, for a fixed number of iterations, the learner trains a classifier on the current training data, and then requests that a batch of examples in the pool be labelled and incorporated into the training set for the subsequent iteration. Three different aspects of active learning stability are measured, each of which involves applying the expected agreement metric (semantic) or the region stability metric (structural) to the trees that are induced at specific iterations during the procedure, as well as to trees that are created when using different initial training sets.

One of the main hypotheses that is tested in this research is whether the use of an alternative splitting criterion improves the stability of the C4.5 algorithm. The splitting criterion plays an

important role in the construction of a decision tree, and it is has been demonstrated that certain splitting criteria are more sensitive to specific variations in the training data than others [19, 21]; thus, it is plausible that tree stability may increase if a different criterion is used. To this end, experiments are performed to compare C4.5's default splitting criterion, which is known as entropy, to another criterion, called DKM [14, 28]. Although the two splitting criteria are also compared in the active learning environment, the primary comparison in the latter case is between seven different active learning methods, including random sampling. Other statistics, such as error rates, are also recorded during both the passive and active learning experiments. Increasing stability is not the only goal, since a learner may exhibit high stability, yet at the same time be unacceptably inaccurate. Such factors must be weighed in determining the best performing algorithm.

1.3 Contributions of this Research

Three main contributions are made in this thesis. The first of these is a detailed study of decision tree instability, using a well-known decision tree learning algorithm and testing two different splitting criteria. This is a significant contribution, since the instability of decision tree learners has received relatively little attention, despite the fact that tree learners are one of the most commonly-used machine learning algorithms in existence. The second major contribution is an examination of several active learning methods that use the decision tree learner as a component. Not only is this the first assessment of decision tree instability in an active learning setting, but it also marks the first known instance where a variety of active learners are compared on the task of inducing a single decision tree. By contrast, existing studies have either grown committees of trees, or have focused exclusively on a single active learning method (e.g. [1, 30]). The third contribution of this research is the proposal and evaluation of a new mechanism for performing active learning with decision trees, which is termed region-based active learning.

The experiments conducted throughout this thesis yielded a number of important and interesting findings. First and foremost, the DKM splitting criterion was found to improve the stability of C4.5, particularly during active learning, without degrading classification accuracy. This is a noteworthy conclusion, as it states that the using DKM with C4.5 makes the algorithm less sensitive to subtle changes in the training data, and that trees produced during active learning with DKM are more consistent than those grown using entropy. In fact, DKM also caused the accuracy of C4.5 to improve significantly on the active learning tasks, which undoubtedly makes it the splitting criterion of choice in this setting. The active learning experiments also revealed that certain methods produced trees that were significantly more stable and accurate than those produced by other methods. To reiterate, this is an important result because the active learners used in these experiments had not been previously employed to induce a single tree. Yet, it was determined here that some of these methods perform remarkably well on this learning problem. Finally, experiments with region-based active learners demonstrated two things. The first is that a simplified committee of trees, when used

to select examples, tends to perform at least as well as a complex tree committee, and lowers the computational cost in some cases. Secondly, active learning methods that employ a strategy known as weight sampling additionally benefit from ranking decision regions, rather than explicitly ranking the unlabelled examples in the pool.

Overall, the contributions of this research are important because decision tree learning algorithms, such as C4.5, are widely used in practice. They are not merely employed as passive learners, but are used as active learners as well. The conclusions drawn in this thesis constitute significant progress toward understanding and improving the quality of decision trees that are induced in both of these learning environments.

1.4 Outline

This thesis is organized as follows. Chapter 2 examines decision tree instability in a passive learning setting. It introduces decision tree learning algorithms and splitting criteria, discusses decision tree instability, and describes the methods that are used to measure instability.

In Chapter 3, the focus is on active learning with decision trees. Several active learning methods are described, as is the approach taken to quantify decision tree instability in this setting.

Chapter 4 is also concerned with employing decision trees as active learners. In this chapter, however, an original approach to active learning with decision tree classifiers is proposed, and new methods are formulated on this basis.

Other research related to the instability of decision trees is reviewed in Chapter 5, as well as some studies that directly involve or are relevant to active learning.

The thesis is summarized in Chapter 6, in which conclusions are drawn and possible directions for future research are outlined.

Chapter 2

Decision Tree Instability

This chapter examines the instability of decision trees in a passive learning setting – a situation in which the learner has no control over the training examples that it receives. A novel measure of structural stability for decision trees is introduced, and two different decision tree splitting criteria are compared in terms of the stability and accuracy of the models that they produce.

The organization of this chapter is as follows: In Section 2.1, the decision tree induction algorithm is described. Two distinct splitting criteria are introduced in Section 2.2. In Section 2.3, the instability of decision tree learners is discussed, followed by a description of several methods for measuring instability, including a newly proposed structural stability metric, in Section 2.4. In Sections 2.5 and 2.6, experiments are conducted and results are presented that compare the performance of the two splitting criteria. Finally, conclusions are drawn in Section 2.7.

2.1 Decision Tree Classifiers

A classification tree or *decision tree* is a hierarchical model, composed of *decision nodes*, *branches*, and *leaf nodes*, which defines a function $h : X \rightarrow Z$. Here, X is a collection of attribute vectors and Z is a finite set of discrete-valued outputs, or classes. A is the set of attributes or features, and is defined by the particular learning problem; the attribute vectors X will be referred to as instances or examples, where each instance $\mathbf{x} \in X$ consists of $|A|$ attribute-value pairs. The pair $(\mathbf{x}, z)$ is called a *labelled example* if $\mathbf{x}$ is known to belong to a particular class $z \in Z$; otherwise, if the class for $\mathbf{x}$ is unknown, $\mathbf{x}$ is referred to as an *unlabelled example*.

Each decision node d in a decision tree T prescribes a test on a specific attribute $a \in A$. A given test has n disjoint and exhaustive outcomes corresponding with n branches, such that each branch connects d to exactly one of n descendant nodes. In turn, each of these descendants is either a decision node or a leaf node. When a decision node d specifies a test on a particular attribute a, d is said to perform a "split" on a, in reference to the effect that the test has of dividing a collection of examples into smaller-sized subsets; in this work, the words test and split are used interchangeably within this context. Finally, a leaf node has no branches, but is instead associated with a particular

output $z \in Z$, which is known as its label. More generally, the label may be a vector of probabilities, indicating the probability of membership in each class $z \in Z$.

A given attribute $a \in A$ must be of one of two types: *continuous* or *discrete*. The value of a continuous attribute can be any real number, whereas a discrete attribute is restricted to a value $v \in V_a$, where V_a is the finite set of legal values for a. An attribute's type determines the form of test that is performed on that attribute by a decision node. A test on a continuous attribute a forms exactly two branches, by specifying a real-valued threshold or cut-point t. Any examples that have a value for a that is less than or equal to t are sent along the left branch, while the remaining examples are sent along the right branch. On the other hand, a test on a discrete attribute a forms $|V_a|$ branches – one for each distinct value that a may take.

Given a training set consisting of N labelled examples $L = \{(\mathbf{x}_1, z_1), \ldots, (\mathbf{x}_N, z_N)\}$, a general algorithm for growing a decision tree can be summarized as follows:

1. At node d of the tree, if every example $\mathbf{x} \in L_d$ is a member of the same class z, then create a leaf node with label z and discontinue further splitting.

2. Otherwise, create a list of candidate splits S. Assign a score to each split $s \in S$, according to some *goodness measure*. Denote by s^* the split that receives the highest score, and let $a^* \in A$ be the attribute selected by s^*.

3. Create a decision node d that performs the split s^* on attribute a^*. Set $n = 2$ if a^* is continuous; otherwise, set $n = |V_{a^*}|$. Partition the examples into n subsets, namely, $L_{d_1}, \ldots, L_{d_n}$, by sending each instance $(\mathbf{x}, z) \in L_d$ along one of the n branches, according to the value of a^* that is specified by $\mathbf{x}$.

4. Repeat steps 1 to 3 for each subset L_{d_i}, for $1 \leq i \leq n$.

Step 1 states that the splitting procedure halts whenever the data L_d arriving at a node d are pure (i.e. all examples have the same class label). Depending on the specific decision tree algorithm, additional stopping criteria may be introduced. The algorithm used in this thesis is the C4.5 decision tree algorithm [42]. C4.5 stops splitting and creates a leaf unless at least two of the subsets produced by the best candidate split contain at least 2 examples.[1] Thus, a leaf node does not necessarily contain examples belonging to a single class. In C4.5, an impure leaf l receives the label of the majority class among the training examples that reach l. The goodness measure used in Step 2 is a function that assigns a value, or score, to each candidate split. C4.5 halts further splitting and creates a leaf node in the event that no candidate split achieves a higher score than that of the parent node. This condition will be referred to as the "zero-gain" stopping criterion, and is discussed further in Section 2.5.

[1] In fact, the minimum number of examples referred to here is a parameter of C4.5 whose default value is 2. Use of the default value is assumed throughout this thesis.

If the training examples L are noisy, or if L is not a representative sample of the target function, then a tree constructed by the above algorithm may "over-fit" the data and make poor predictions on unseen cases. In order to prevent over-fitting, many decision tree learners "prune" the initial tree, either by stopping the growing phase prematurely (pre-pruning) or removing portions of the tree after it has been grown (post-pruning). C4.5 takes the latter approach, employing a technique called *pessimistic pruning*. Based on the training set L, the true error rate of a given subtree or leaf is estimated according to the confidence interval of the binomial distribution. An existing subtree is then replaced by a leaf whenever the leaf has a lower predicted error rate.

Finally, a decision tree T classifies an example $\mathbf{x}$ according to the following recursive procedure:

1. If T is a leaf, then output the label of T.
2. Otherwise, T is a decision node. Subject $\mathbf{x}$ to the test that is specified by T, thereby producing an outcome i. Send $\mathbf{x}$ along branch i to the subtree T_i, and return the class label that T_i assigns to $\mathbf{x}$.

2.2 Decision Tree Splitting Criteria

An important element of the tree-growing procedure is the goodness measure that is used to rank candidate splits; this measure is known as the splitting criterion. While various strategies for choosing the best split have been proposed and studied (e.g. [9, 35]), the focus here is on the generic *gain* criterion, which utilizes an impurity function to evaluate candidate splits. The impurity function measures the heterogeneity of a collection of examples with respect to its class labels; by choosing different impurity functions for the gain criterion, it is possible to produce distinct splitting criteria. The basic gain criterion is a feature of the two most widely-used decision tree software packages, namely, C4.5 and CART [7]; a different impurity function is used by each of these.

Given a collection of labelled examples L and an impurity function $I : L \rightarrow [0, 1]$, the gain criterion measures the difference between the impurity of L before and after a candidate split s is performed. The impurity of the data before making the split s is computed as $I(L)$. The total impurity after performing the split is calculated as a weighted sum of the impurities of the n_s subsets induced by s, where the weight of subset L_i is $\frac{|L_i|}{|L|}$. Finally, the gain criterion, which calculates the change in impurity that results from performing split s, is defined as:

$$gain_I(L, s) = I(L) - \sum_{i=1}^{n_s} \frac{|L_i|}{|L|} \cdot I(L_i) \tag{2.1}$$

Thus, $gain_I(L, s)$ is the score assigned to candidate split s at a node that contains the examples L. Decision tree algorithms, such as C4.5, typically aim to grow the smallest tree (i.e. the tree containing the fewest nodes) that is consistent with the training data. According to this strategy, the best split s^* is chosen which produces the greatest decrease in impurity, or alternatively, which maximizes the gain. Formally, $s^* = \arg\max_{s \in S} gain_I(L, s)$.

Next, the two impurity functions that are used in the experiments in this chapter are introduced. A binary (i.e. positive and negative class) learning problem is assumed throughout this thesis.[2] The impurity function used by C4.5 is an information-theoretic measure called *entropy*, which is defined as follows. Given a collection of labelled examples L, let p_L^+ denote the fraction of examples in L that belong to the positive class (i.e. the minority class). The entropy of L is calculated as:

$$ent(L) = -(p_L^+)\log_2(p_L^+) - (1-p_L^+)\log_2(1-p_L^+) \tag{2.2}$$

where $-0 \cdot \log(0)$ is defined to be 0. It can be shown that the entropy of L is maximal (equal to 1) when exactly half of the examples belong to each class, and minimal (equal to 0) when all examples belong to the same class. The splitting criterion $gain_{ent}(L, s)$, which is obtained by combining the entropy impurity measure with the generic gain criterion, is known as the *information gain* criterion. Information gain is one of two splitting criteria that are used by C4.5. The other criterion, which is called the gain ratio, is introduced in Section 2.5.3.

The second impurity function that is used in these experiments is known as *DKM*[3] [14, 28]. Using the notation from Equation 2.2, the impurity of a collection of examples L is computed by DKM as:

$$dkm(L) = 2\sqrt{(p_L^+)(1-p_L^+)} \tag{2.3}$$

The splitting criterion $gain_{dkm}(L, s)$, which pairs the DKM impurity measure with the generic gain criterion, is called the *DKM gain* criterion. In this thesis, the gain is implicitly assumed; therefore, this splitting criterion is referred to simply as DKM.

There exists theoretical and experimental evidence in the machine learning literature that suggests that DKM is superior to information gain as a decision tree splitting criterion. The analysis conducted by Kearns and Mansour [28] established a relationship between the size of a tree (i.e. the number of decision nodes) and its error rate that was dependent on the splitting criterion used to grow the tree. The lower bound on the number of nodes required to achieve a given error rate using DKM was found to be superior to the bounds derived for other well-known splitting criteria; in fact, the DKM criterion was proven to be optimal in this respect. As an example, suppose that two *unpruned* decision trees, T_i and T_d, are induced using information gain and DKM, respectively. The theory predicts the following: If T_i and T_d contain the same number of nodes, then T_d has a lower error rate than T_i; or alternatively, if T_i and T_d have the same error rates, then T_d contains fewer nodes than T_i. These findings were supported empirically by Dietterich et al. [14], who conducted experiments on nine benchmark problems and observed that DKM consistently constructed better trees (i.e. trees that were either more accurate, or that were equally accurate but smaller) than did information gain. In addition, a later empirical study concluded that *pruned* trees built using

[2]This is a requirement of the DKM splitting criterion [14], as well as of the bootstrap-LV algorithm [47], which is introduced in Section 3.2.

[3]The name DKM refers the creators of the splitting criterion, namely, Dietterich, Kearns, and Mansour. It has also been referred to as the "Z criterion" in some publications (e.g. [18, 48]).

the DKM criterion achieved significantly lower error rates than trees constructed by the original (i.e. entropy-based) version of C4.5 [18].

2.3 Instability of Decision Tree Learning Algorithms

2.3.1 Learner Instability and its Effects

A learning algorithm is said to be *unstable* if it is sensitive to small changes in the training data. Given two training sets that differ by some small amount, an unstable learner Ω_{unstab} may produce two dramatically different classifiers. Put another way, hypotheses induced by Ω_{unstab} from several similar data samples, each taken from the same underlying distribution, are likely to exhibit considerable variance. By contrast, stable learning algorithms, such as nearest neighbour, tend to be less responsive to minor changes in the training data [2].

Learner instability can be problematic for a number of reasons. Estimates of predictive accuracy for a classifier produced by an unstable algorithm can be noisy, making it difficult to choose the best predictor from a set of candidates, for example. Note that this is not always the case, since a learner may induce a diverse set of models from different data samples, all of which achieve similar accuracy. In domains where knowledge extraction is a goal of learning (i.e. the classifier itself is interpreted, rather than just performance measures), it is difficult to gain insight into the prediction model if, as a consequence of instability, the learner generates vastly different hypotheses when small changes are made to the training set. An example of this problem is given by Turney, who describes a real-life situation in which decision trees are used by engineers to help them understand the sources of low yield in a manufacturing process [52, p. 2]: "The engineers frequently have good reasons for believing that the causes of low yield are relatively constant over time. Therefore the engineers are disturbed when different batches of data from the same process result in radically different decision trees. The engineers lose confidence in the decision trees, even when we can demonstrate that the trees have high predictive accuracy."

Decision tree induction algorithms are known to be unstable procedures [6, 15]. However, they are frequently used with little or no regard for their sensitivity to small perturbations in the training data. By failing to account for the instability of tree learners, one may unknowingly rely on a classifier that is highly dependent on the particular data sample provided to the algorithm. Not only might the predictive performance of such a model be poor, but in situations where users make inferences based on the structure of a single tree, as the engineers do in the manufacturing process example, their conclusions may be ill-founded, and poor decisions may be made as a result. Dannegger suggests that this practice "... may be just as dangerous and misleading as ignoring available confidence regions within other statistical estimation procedures." [12, p. 6]

In this thesis, two types of stability are examined: semantic stability and structural stability. A learner that is semantically stable will, given data samples drawn from the same process, tend to

produce hypotheses that make similar predictions. For a learner to be syntactically or structurally stable, a stronger condition must be satisfied, namely, the hypotheses that it creates from related data sets must themselves be similar. Thus, structural stability is a sufficient condition for semantic stability, but the converse is not true. It is also possible to formulate a measure of stability that is not purely semantic or purely structural, but which considers some characteristics of a classifier's structure. Several methods for measuring these two types of stability are introduced in Section 2.4. Unless otherwise noted, the word stability will be used to refer to structural stability throughout this work.

Next, a concrete example of decision tree instability is presented. Figures 2.1 and 2.2 depict two pruned trees generated by C4.5 from the lymphography dataset, which was obtained from the UCI repository of machine learning databases [39]. These figures display the raw output of C4.5, and are interpreted as follows: Each line of the output describes a single test (or split) that appears to the left of the colon; optionally, a class label is displayed on the right side of the colon, and is indicative of a leaf node. The parent-child relationship between certain nodes is represented using indentation. Each level of indentation corresponds to a particular tree depth, and is indicated by a vertical bar (|). For example, in Figure 2.1, the nodes 'block_of_affere = no:' and 'block_of_affere = yes:' are siblings, while 'no_of_nodes_in <= 3:' is the parent of these two nodes.

The lymphography dataset was converted from a multi-class to a two-class problem by deleting the examples associated with the 'normal find' and 'fibrosis' classes, which account for 6 of the 148 instances in the original dataset. Thus, the modified version of the dataset, which is denoted as L, contains 142 examples and has 18 attributes. A decision tree was created by randomly selecting 106 (roughly 75%) of the available examples, and applying C4.5 to the resulting dataset (default parameters for C4.5 are assumed unless otherwise noted). Let us denote this dataset as L_{106}, and the tree induced from this data as T_{106}. The tree T_{106}, which is displayed in Figure 2.1, contains 5 decision nodes and 8 leaves.

The data was then altered by adding to L_{106} a single randomly-chosen example $x \in L - L_{106}$, thus forming the set L_{107}. C4.5 was applied to L_{107}, and the resulting tree T_{107} is displayed in Figure 2.2. Containing 9 decision nodes and 13 leaves, T_{107} is considerably larger than T_{106}. Whereas T_{106} indicates that only 3 of the 18 attributes are relevant to the diagnosis, T_{107} considers 7 features when making predictions. The attribute selected for the root node is also inconsistent from one tree to the next. When interpreting a decision tree, the proximity of a feature to the root is often considered to be a measure of its relevance in predicting the class variable. Based on this view, if a physician were to examine either T_{106} or T_{107} in isolation, he/she would reach rather different conclusions regarding the relative importance of each factor to the diagnosis. To further examine the effect on tree structure of adding a single example to L_{106}, this step was repeated for each of the remaining 35 unused examples. In turn, each instance was added to L_{106} and a decision tree was trained on each set of size 107 that was formed. Of the resulting trees, 27 differed from T_{106}, and

```
no_of_nodes_in > 3: malign_lymph
no_of_nodes_in <= 3:
|   block_of_affere = no:
|   |   early_uptake_in = yes: malign_lymph
|   |   early_uptake_in = no:
|   |   |   no_of_nodes_in <= 1: metastases
|   |   |   no_of_nodes_in > 1: malign_lymph
|   block_of_affere = yes:
|   |   changes_in_node = no: malign_lymph
|   |   changes_in_node = lacunar: metastases
|   |   changes_in_node = lac_margin: metastases
|   |   changes_in_node = lac_central: malign_lymph
```

Figure 2.1: Pruned tree for the UCI lymphography dataset, trained on 106 examples.

```
changes_in_node = no: malign_lymph
changes_in_node = lac_central: malign_lymph
changes_in_node = lacunar:
|   exclusion_of_no = no: metastases
|   exclusion_of_no = yes:
|   |   no_of_nodes_in <= 1: metastases
|   |   no_of_nodes_in > 1: malign_lymph
changes_in_node = lac_margin:
|   block_of_affere = no:
|   |   extravasates = yes: malign_lymph
|   |   extravasates = no:
|   |   |   early_uptake_in = no: metastases
|   |   |   early_uptake_in = yes:
|   |   |   |   changes_in_lym = bean: malign_lymph
|   |   |   |   changes_in_lym = oval: malign_lymph
|   |   |   |   changes_in_lym = round: metastases
|   block_of_affere = yes:
|   |   no_of_nodes_in <= 3: metastases
|   |   no_of_nodes_in > 3:
|   |   |   bl_of_lymph_c = no: malign_lymph
|   |   |   bl_of_lymph_c = yes: metastases
```

Figure 2.2: Pruned tree for the UCI lymphography dataset, trained on 107 examples.

in total 6 unique trees were produced. This reveals that the changes in the hypothesis that were observed as a result of adding a single example to L_{106} were not simply a consequence of the particular x that was added.

2.3.2 Causes of Decision Tree Instability

Having presented an example of decision tree instability, the causes of this instability are now discussed. As was detailed in Section 2.1, at a given node, C4.5 ranks candidate splits according to the scores that are assigned by a goodness measure. Instability arises when the scores assigned to one or more candidates are sufficiently close to the score achieved by the best split, such that if a small change is made to the training data, a different candidate becomes the best split. Due to the recursive nature of the tree-growing procedure, changing the split at node T may cause the subtree below T to undergo significant transformation. Li and Belford [32] present theorems that relate the degree of change in the training data that is required to cause the best split at a node to change.

The recursive partitioning algorithm that C4.5 uses to grow a decision tree is greedy – it performs the split that is locally optimal, without considering its implications on future tree growth. This myopic strategy may contribute to the instability of C4.5; if the best split was instead selected using some level of lookahead, the algorithm could conceivably be more resistant to minor differences in training samples. However, Murthy and Salzberg [37] found that the use of one-level lookahead generated trees that did not differ significantly from those produced by the greedy approach, in terms of accuracy and size. Furthermore, the authors observed numerous cases where lookahead yielded trees that were inferior to the greedily-induced ones. In light of these findings, paired with the enormous increase in computation that is required for lookahead, it is difficult to justify its use when growing decision trees, even though it could potentially improve their stability. For these reasons, the effect of lookahead on the stability of decision tree learners is not explored in this thesis.

Several methods for improving learner stability have been proposed, including some that are designed specifically for decision tree algorithms. For example, techniques that aggregate multiple classifiers are able to improve the accuracy of an arbitrary base learner under a broad set of conditions, and often yield increased semantic stability as well; a well-known approach of this type is bagging [5], which is described in detail in Section 3.2. The predictions of a classifier ensemble formed by bagging are typically more accurate and stable than those of the base learner itself. Unfortunately, since bagging averages the predictions of several classifiers, the resulting model is no longer intelligible.

Other methods have been devised for improving the stability of decision trees while retaining the desirable property of interpretability; these are reviewed in Chapter 5. All of these methods induce and combine several trees, or otherwise significantly increase the complexity of the procedure used to grow a tree. In this chapter, an alternative, less elaborate approach to improving the stability of decision tree learners is investigated. The objective is to determine whether replacing

the entropy-based splitting criterion in C4.5 with the DKM criterion will make the algorithm more stable, without significantly degrading the accuracy or intelligibility of the trees that are produced. This is a fundamental modification to the decision tree algorithm, which, if successful, can be used in conjunction with other techniques that aim to improve tree stability.

Drummond and Holte [19] showed that, for attributes that are split into exactly two subsets (e.g. two-valued discrete attributes and continuous attributes), DKM is completely insensitive to the class priors in the training data. Elkan [21] generalized this result to multi-way splits, and also showed analytically that related impurity functions *are* sensitive to priors. Other research that was summarized in Section 2.2 suggests that DKM tends to grow more accurate, compact trees than the entropy criterion. Together, these findings provide a basis for hypothesizing that the DKM splitting criterion may increase the stability of C4.5, without reducing the quality of the decision trees that are induced.

It should be emphasized that algorithmic stability is not the only goal, but must be considered in conjunction with other performance measures, such as classification accuracy. For instance, perfect stability can be achieved by employing an algorithm that produces the same classifier regardless of the training set. It is therefore desirable to choose the most stable learner among those that achieve approximately the same accuracy. In tasks where the stability of the learner is a particularly high priority, one may be even willing to trade-off a small amount of predictive accuracy if it leads to a worthwhile increase in stability.

2.4 Quantifying Stability

A stable learning algorithm Ω_{stab} has been defined as one that is resistant to small perturbations in the training data. Given two "similar" training sets, Ω_{stab} will produce two models that are, on average, not significantly different. This characterization of algorithmic stability requires the specification of two distance metrics – one between data sets (i.e. samples taken from the same underlying distribution) and another between hypotheses – as well as a definition of what degree of similarity/dissimilarity constitutes stability based on theses metrics. The focus here is not on the problem of measuring the distance between sets of examples, as existing metrics are applicable. In fact, an explicit distance metric is not required if one assumes that the process used to generate the data sets inherently implies some degree of similarity between them. For example, if k sets of examples are obtained that are the k training sets produced by k-fold cross-validation[4] from a given dataset, then these k data samples may be considered to be "similar" for the purpose of measuring the stability of a learning algorithm. On the other hand, choosing a particular metric that is used to distinguish between hypotheses is the essence of quantifying stability. For this reason, several such measures are reviewed, and a novel measure is proposed that is specific to domain-partitioning

[4]The k-fold cross-validation estimate is obtained by dividing the available data L into k disjoint subsets, $L_1, \ldots, L_k$. Then, for $1 \leq i \leq k$, a classifier is trained from $L - L_i$ and is evaluated on the examples in L_i, thereby producing the i^{th} score. The final estimate is typically calculated as the average of these k scores.

hypotheses such as decision trees.

2.4.1 Variance in Error Rate Estimates as Instability

Before introducing specific stability metrics, the implications of interpreting the variance in a learner's error rate as a measure of its stability are considered, in order to demonstrate that such an approach is inadequate for quantifying either semantic or structural stability. Suppose that the error rate of learner Ω on dataset L is estimated via k-fold cross-validation; intuitively, high variance in the k error rates indicates that Ω is unstable with respect to L. For this to be valid, however, it would first be necessary to remove the variability in the estimate that arises due to the use of k different evaluation sets in cross-validation, since identical hypotheses could yield distinct error rates if evaluated on different data. For this reason, the cross-validation procedure is modified, such that a single evaluation set E is set aside and used to evaluate each hypothesis, and the examples in $L - E$ are partitioned according to the usual cross-validation procedure to form the k training sets. Under this scheme, the error rates for Ω are derived from (presumably) different training sets, but using identical evaluation data. Thus, any discrepancy between the scores can be attributed to the instability of Ω with respect to these training sets. That is to say, if the error rates vary, then it must be the case that the hypotheses differ in some capacity.

Although instability can be detected by the above procedure, stability cannot be inferred using this approach. If two hypotheses produce identical error rate estimates, this does not imply that the hypotheses themselves are identical. For example, if the error rate is found to be perfectly stable such that each classifier misclassifies exactly one example in E, it is possible that each errs on a different example, which would imply that the hypotheses are distinct. Thus, stable error rates are a necessary but not a sufficient condition for hypothesis stability.

2.4.2 Measuring the Semantic Stability of Learners

Relatively few strategies for measuring the stability of an arbitrary learning algorithm have been suggested. A learner-independent measure of semantic stability formulated by Turney [52] is perhaps the most well-known, as it has been utilized in a number of studies (e.g. [17, 29]). Given two training sets, sampled from the same distribution, the learning algorithm is applied to induce two hypotheses. Turney measures the stability of the learner as the *expected agreement* between the two hypotheses, where the agreement is the probability that a randomly chosen unlabelled example is assigned to the same class by both models. Details regarding the calculation of the expected agreement are presented in Section 2.5.2.

2.4.3 Measuring the Structural Stability of Decision Trees

In this section, two distance measures for decision trees are briefly reviewed, which can be used to quantify the structural stability of tree learners. Shannon and Banks [51] measure the distance

between two trees as the weighted sum of the *discrepant r-paths*; this metric will be referred to here as *Discrepant*. A discrepant r-path is a path of length r, beginning at the root and terminating at a leaf, that appears in exactly one of the two trees being compared. A path consists of r attributes and $r-1$ edges that each connect to either a right or left child node (i.e. binary splits are assumed). The Discrepant metric is flexible in that it allows one to specify an arbitrary weight function $\alpha(r)$. As an example, the authors suggest using the function $\alpha(r) = \frac{1}{r}$, which penalizes discrepancies that occur near the root of the tree more severely than differences that occur at greater tree depths. Unfortunately, Discrepant is quite limiting in other respects. Most notably, the actual cut-points for tests on continuous attributes are not considered. Although this may be appropriate for the medical diagnosis task that is the authors' focus, in some domains this simplification could cause important differences between tree structures to be ignored. Also, the assumption of binary test outcomes restricts the application of this measure to data consisting of only continuous and/or two-valued discrete attributes.

A more generally applicable distance measure for decision trees, named *Common*, simply counts the number of nodes that are shared by two trees [40]. Nodes in the two trees are compared in a breadth-first ordering; the count is incremented if the current pair of nodes test the same attribute, specify the same cut-point (in the case of a continuous feature), and also appear in the same position in both trees. If a node is encountered in one tree that is not present in the other, then the subtree below that node is not processed. Common represents the strongest possible measure of structural stability for decision trees – the maximum value (which is equal to the number of internal nodes) is achieved if and only if the two trees are structurally identical. In fact, it is possible for two trees to be logically equivalent, yet have zero structural stability according to Common.

Just as there is no universally accepted performance metric for classification tasks, there is no consensus on how to quantify the structural stability of a decision tree learner. In some cases, the demands of the learning task may suggest an appropriate metric. For example, both Discrepant and Common report minimal stability when the two trees being compared differ at the root node. However, it is possible for two trees to differ at the root, and yet be quite similar or even identical elsewhere; neither of these metrics are sensitive to such an occurrence. At the opposite end of the spectrum lies Turney's expected agreement, which measures the probability that two classifiers predict the same class label for a randomly chosen example. Turney argues that two properties of this measure of semantic stability are desirable: 1) it is not restricted to any particular classifier representation, and 2) it considers logically equivalent representations to be equal. This thesis focuses exclusively on the stability of decision trees, and so the first point is not applicable here. With regard to the second property, since it is possible for two logically equivalent trees to have remarkably different structure, the expected agreement measure is clearly not appropriate if one wishes to quantify and compare structural stability. Thus, although these two characteristics of expected agreement may be favourable for learning tasks in general, neither contributes to the objective of

measuring structural stability. Next, a novel measure of structural stability is proposed, which borrows elements from the metrics that have been reviewed here. This metric captures a weaker notion of stability than purely structural stability, yet a stronger notion than purely semantic stability.

2.4.4 The Region Stability Metric

The newly proposed *region stability* metric computes the probability that two decision trees, induced from different training sets, classify a randomly selected training example in the "same" leaf node. The term *decision region* is used to refer to a leaf node, and this metric computes the similarity between two sets of decision regions (or trees). For two decision regions r_1 and r_2 (residing in trees T_1 and T_2, respectively) to be considered equal, the path p_1 from the root of T_1 to r_1 must contain the same set of tests as the path p_2 from the root of T_2 to r_2. In a C4.5 decision tree, a test t on a discrete attribute a_d can appear at most once along any path from the root to a leaf; therefore, equality between r_1 and r_2 is preserved if: 1) neither p_1 nor p_2 contains the test t, or 2) both p_1 and p_2 contain t, and the same discrete value for a_d appears in p_1 and p_2. A continuous attribute, on the other hand, may be tested multiple times along a single path, at various thresholds. Thus, r_1 and r_2 are defined to be equal with respect to a continuous attribute a_c if the set of tests pertaining to a_c in p_1 and those in p_2 define the same range of values along the dimension a_c. In other words, r_1 and r_2 must specify the same lower and upper bounds for a_c. The final condition for r_1 and r_2 to be equal is that they must predict the same class label. The region stability of two decision trees T_1 and T_2 is estimated by having both trees classify a set of examples E, and recording the fraction of examples $x \in E$ for which T_1 and T_2 map x to decision regions that are considered to be equal according to the above conditions. This produces a value in $[0, 1]$, which is the region stability score, and serves as an estimate of structural stability.

Note that the decision region overlap between two trees T_1 and T_2 could be computed without the use of an evaluation set E, by calculating the fraction, in terms of total volume, of decision regions in T_1 for which an equivalent region exists in T_2. However, this scheme gives equal weight to each decision region or leaf, even though the distribution D of examples in the particular domain may be non-uniform. On the other hand, if the examples in E are drawn with respect to D, a decision region r_1 that classifies a larger portion of examples than another region r_2 has a correspondingly greater influence on the region stability score, which, it is argued here, is appropriate. The evaluation set E may be the same set of instances that is used to estimate the predictive performance of a decision tree, for example, and does not necessarily require that data be set aside for the sole purpose of measuring structural stability. Also, the examples in E need not be labelled, since it is the class labels predicted for $\mathbf{x} \in E$ by T_1 and T_2 that are ultimately compared – the true label of $\mathbf{x}$ is irrelevant to the calculation of the region stability score.

It should be noted that when checking two regions r_1 and r_2 for equality, the region stability metric compares the *unordered* set of tests that define these regions. This is contrary to Discrepant

and Common, which consider two decision regions to be different if the set of tests defining the regions are equal, but are ordered differently in the trees being compared. It is posited that the latter approach to measuring structural stability can fail to account for important similarities that exist between two trees. Another important difference between the region stability metric and Common is that Common assigns equal weight to every node, whereas decision regions are weighted according to the instances in the evaluation set E by the region stability metric, as discussed previously. Hence, by the region stability metric, two trees whose decision regions are similar only in densely sampled areas of the instance space will achieve a higher stability score than two trees that are similar only in sparse areas of the space. Consider the two decision trees in Figure 2.3, for example. For each leaf l in these trees, the number of training examples classified by l, as well as the estimated number of errors made by l, are displayed in parentheses. Each tree spawns a leaf from the split performed at the root node, which classifies 32% and 35% of the examples in each tree, respectively. By the Common metric, these decision regions contribute the same amount to the stability score as do the leaves that classify less than 5% of the training examples. Although Discrepant allows one to specify a weight function, this function must depend only on the length of a discrepant path, and therefore cannot be sensitive to the distribution of instances in the space.

As the region stability metric has been defined thus far, the unordered sets of tests defining two decision regions r_1 and r_2 must be exactly the same in order for the regions to be counted as equal. For tests on a continuous attribute a_c, region equality holds if and only if r_1 and r_2 specify precisely the same upper and lower bounds for a_c; this may be seen as an overly strict requirement in the context of some learning problems. Due to the fact that C4.5 only places a threshold for a_c at a value that exists in the training data, subtle differences between two training sets can make it impossible for the same thresholds to appear in two trees induced from these data samples. The implication is that the region stability for two structurally similar trees can be low, or even zero, if the thresholds for one or more continuous tests differ slightly between the two trees. Consider the two trees in Figure 2.3, for example, which were grown using the DKM criterion from a random division of the UCI glass dataset [39] into two equally-sized halves. Both trees select the same attributes, in the same order, for the first four tests that are performed, and the leaf nodes descending from these tests predict the same class labels. On average, the thresholds defined by the two trees on these four attributes differ by less than 3% of the range of values (that are present in the training data) for each feature. Nevertheless, the region stability score is zero for this pair of trees, since no example can be classified by two leaves that define identical decision thresholds.

For some learning tasks, it may be desirable to employ a stability measure which reflects the fact that two trees, such as the ones in the preceding example, have a considerable amount of structure in common, even though it may be the case that no two decision regions are identical. To accomplish this with the region stability metric, the manner in which decision regions are compared with respect to a continuous attribute a is modified. When checking two regions r_1 and r_2 for equality, instead

```
Al > 1.38 : non-float (29.0/7.1)
Al <= 1.38 :
|   Ca > 9.85 : non-float (6.0/1.2)
|   Ca <= 9.85 :
|   |   K <= 0.23 : float (12.0/1.3)
|   |   K > 0.23 :
|   |   |   Mg > 3.75 : non-float (7.0/1.3)
|   |   |   Mg <= 3.75 :
|   |   |   |   Al <= 1.08 : non-float (3.0/1.1)
|   |   |   |   Al > 1.08 : float (25.0/3.7)


Al > 1.41 : non-float (26.0/7.1)
Al <= 1.41 :
|   Ca > 10.17 : non-float (4.0/1.2)
|   Ca <= 10.17 :
|   |   K <= 0.19 : float (14.0/1.3)
|   |   K > 0.19 :
|   |   |   Mg > 3.66 : non-float (5.0/1.2)
|   |   |   Mg <= 3.66 :
|   |   |   |   RI > 1.51743 : float (20.0/1.3)
|   |   |   |   RI <= 1.51743 :
|   |   |   |   |   Al > 1.27 : float (5.0/1.2)
|   |   |   |   |   Al <= 1.27 :
|   |   |   |   |   |   Fe > 0.07 : non-float (2.0/1.0)
|   |   |   |   |   |   Fe <= 0.07 :
|   |   |   |   |   |   |   Si <= 72.81 : non-float (2.0/1.0)
|   |   |   |   |   |   |   Si > 72.81 : float (3.0/1.1)
```

Figure 2.3: Two trees grown using the DKM criterion from a random division of the UCI glass dataset into two equal-sized halves.

of insisting that the upper and lower bounds specified by r_1 be exactly the same as those of r_2, the bounds are permitted to differ by a small amount and still be considered equal. Formally, two regions r_1 and r_2 are considered to be equal with respect to a continuous attribute a_c if the bounds $\text{upper}_a(r_1)$ and $\text{upper}_a(r_2)$ are within ε_a units of one another, and the bounds $\text{lower}_a(r_1)$ and $\text{lower}_a(r_2)$ are within ε_a units of one another. Under this scheme, the lengths of the decision regions r_1 and r_2 along dimension a may differ by as much as $2 \cdot \varepsilon_a$, yet still be considered equal. Note that by setting $\varepsilon_a = 0$, the regions are required to be identical with respect to a. In practice, rather than specifying an absolute value ε_a for each continuous attribute a, a single parameter, ε_{pct}, is used. Given $\varepsilon_{pct} \in [0, 100]\%$, ε_a is calculated for attribute a as:

$$\varepsilon_a = \frac{\varepsilon_{pct}}{100} \cdot [\max(a) - \min(a)] \tag{2.4}$$

where $\min(a)$ and $\max(a)$ are the minimum and maximum allowable values for a. If these extrema are defined naturally in the domain, then the minimum and maximum values for each continuous attribute can be specified directly. For instance, the minimum and maximum values for an attribute that represents an 8-bit colour value are 0 and 255. If absolute limits are not defined, then the minimum and maximum values for a that appear in the entire dataset are used as the boundary values. The final detail is that if a region r_1 does not specify a lower bound for a, then r_1's lower bound for a is treated as if it were $\min(a)$. As a consequence, a region r_2 that specifies a lower bound in the interval $[\min(a), \min(a) + \varepsilon_a]$ is considered to be equal to r_1, even though the latter region does not explicitly constrain the value of a. Similar logic is applied when checking upper bounds for equality.

The application of the region stability metric is not restricted to decision trees, as the metric can be used with any learner that produces an axis-parallel domain partitioning hypothesis. Note that C4.5Rules [42], for example, does not fit this description, since it creates an ordered set or rules that may not be mutually exclusive. A method closely resembling the region stability metric was developed by Monk et al. [36] for the purpose of measuring the similarity between rule sets.

2.5 Decision Tree Stability Experiments

2.5.1 Datasets

In order to investigate whether using the DKM splitting criterion yields improved hypothesis stability over the entropy criterion when used with C4.5, a series of experiments were conducted using datasets from the UCI repository [39]. For datasets that included separate training and test sets (e.g. pendigits), the two sets were combined into a single collection of examples. As the region stability metric requires that each instance be classified by exactly one leaf, missing values were removed from the data.[5] Specifically, attributes with an unknown value rate of greater than 10%

[5]The strategy used by C4.5 to cope with missing attribute values is to divide an example whose value for attribute A is unknown. This occurs whenever a test is performed on A, and produces a number of weighted fractional instances that is

Name	Attributes	Instances	Min. Class	% Min.
breast-cancer	9	277	recurrence	29.2
car ~	6	1,728	acceptable	30.0
dermatology ~	33	366	1	30.6
kr-vs-kp	36	3,196	no-win	47.8
nursery ~	8	12,960	priority	32.9
tic-tac-toe	9	958	negative	34.7
vote	14	312	republican	41.7

Table 2.1: UCI datasets containing discrete attributes only.

were removed, and then any remaining instances that still contained missing values were deleted. Finally, multi-class problems were converted to two-class ones by designating a single class as the target concept, and aggregating the remaining classes into the class "other."

A collection of 29 datasets was initially obtained from the UCI repository. However, on 4 of these datasets, the decision trees produced by C4.5 during experimentation exhibited an average error rate that was greater than the default error rate – the error rate of the trivial classifier, which always predicts the majority class. Since neither of the two splitting criteria induced useful classifiers from these 4 datasets, they have been omitted from the study.[6] The remaining 25 datasets were divided into three categories, according to whether they contained: 1) discrete attributes only, 2) continuous attributes only, or 3) a mixture of discrete and continuous attributes; their properties are summarized in Table 2.1, Table 2.2, and Table 2.3, respectively. In the "Name" column of each table, the tilde symbol (~) indicates that a particular dataset has been converted from a multi-class to a two-class learning problem. The "Attribute" column specifies the number of discrete and/or continuous features that are present in a given dataset, while the "Instances" column reports the total number of examples. Finally, the "Min. Class" and "% Min." columns display the name of the minority (or positive) class, and the percentage of examples that belong to that class, respectively. The full names of these datasets are provided in Appendix A.1.

2.5.2 Experimental Procedure

The empirical procedure proposed by Turney [52], which was described in Section 2.4.2, was used for the purpose of estimating semantic stability. Specifically, the expected agreement was estimated by the following empirical procedure: For a given dataset L and learning algorithm Ω, v random attribute vectors (i.e. unlabelled examples) U were generated according to a uniform distribution. Then, for t trials, the following steps were repeated. On trial i, L was randomly divided into two equal-sized training sets L_1 and L_2. The learner Ω was applied to each of these, producing the hypotheses $h_\Omega(L_1)$ and $h_\Omega(L_2)$. Finally, the average agreement g_i was calculated with respect to

equal to the number of distinct test outcomes; each fractional instance is then propagated along the appropriate branch. If, during classification, a single example reaches more than one leaf in this manner, the class with the largest combined weight is predicted.

[6]The datasets that were omitted are (minority classes displayed in parentheses): abalone ("9"), contraceptive ("long"), hepatitis ("die"), and wpbc ("recurrence").

Name	Attributes	Instances	Min. Class	% Min.
glass ~	9	163	non-float	46.6
ionosphere	32	351	bad	35.9
letter ~	16	20,000	letter-k	3.7
pendigits ~	16	10,992	digit-9	9.5
pima-indians	8	768	positive	34.9
segmentation ~	18	2,310	cement	14.3
sonar	60	208	rock	46.6
vehicle ~	18	846	opel	25.1
vowel ~	10	990	hud	9.1
wdbc	30	569	malignant	37.3
yeast ~	6	1,484	nuclear	28.9

Table 2.2: UCI datasets containing continuous attributes only.

Name	Attributes Cont. / Discr.	Instances	Min. Class	% Min.
anneal ~	6 / 4	812	not-3	23.0
australian	6 / 9	653	+	45.3
german	7 / 13	1,000	bad	30.0
heart-c	6 / 7	296	>50%	45.9
heart-h	5 / 5	261	>50%	37.5
heart-statlog	7 / 6	270	present	44.4
lymphography ~	3 / 15	142	malign	43.0

Table 2.3: UCI datasets containing a mixture of continuous and discrete attributes.

U, where the agreement for $\mathbf{x} \in U$ was 1 if $h_\Omega(L_1)$ and $h_\Omega(L_2)$ predicted the same class label for $\mathbf{x}$, and was 0 otherwise. After repeating these steps t times, the expected agreement (i.e. semantic stability) was computed as $\frac{1}{t}\sum_{i=1}^{t} g_i$.

To estimate structural stability, the above procedure was modified to use the region stability measure in place of semantic agreement. Thus, the structural agreement $\mathbf{x} \in U$ was 1 if $h_\Omega(L_1)$ and $h_\Omega(L_2)$ classified $\mathbf{x}$ in "equal" decision regions, and was 0 otherwise. Equality between two regions was determined according to the specifications in Section 2.4.4. The average structural agreement and estimated structural stability were calculated analogously to Turney's semantic stability.

The C4.5 algorithm distinguishes between two types of attributes: continuous and discrete. However, many of the datasets used in these experiments contain attributes that are effectively integer-valued. For example, in the pima-indians dataset, 6 of the 8 attributes denoted as type continuous contain only integer values in the data. In order to improve the fidelity of the artificial inputs that are generated for estimating stability, when randomly assigning a value to a continuous attribute a, only integer values are considered if the values for a in the data are strictly integers; otherwise, a real number is randomly generated. Given these constraints, the value for a is chosen from the range $[\min(a), \max(a)]$, where $\min(a)$ and $\max(a)$ are the minimum and maximum allowable values for a, respectively, as defined in Section 2.4.4. The use of a uniform distribution when generating unla-

belled examples can be seen as approximating the true distribution of examples in a given domain. Since a number of the UCI datasets contain relatively few training examples (say, less than 500), it is not always feasible to set aside examples that will serve solely as weights for the region stability metric. When using the 2-fold cross-validation procedure proposed by Turney, the learning algorithm trains on only half of the available examples during a given trial; reducing the number of training examples further than this may severely restrict the learner's ability to capture the target concept.

Experiments were carried out using values of 10,000 for both t and v (the number of times the data was randomly split, and the number of attribute vectors that were generated) in the stability estimation procedure. The semantic and structural stability of C4.5 was estimated when using the default entropy splitting criterion, and then using the DKM criterion. For control, the same partitions of the data were presented to both splitting criteria. Furthermore, for all datasets that contained one or more continuous attributes, the ε_{pct} parameter was varied from 0% to 15%, in increments of 1%; for each continuous attribute a, the absolute discrepancy permitted between thresholds defined on a was calculated according to Equation 2.4. It was presumed that 15% would be a generous upper limit for ε_{pct}. As discussed, the purpose of incorporating the ε_{pct} parameter into the region stability metric was to recognize "approximately equal" tests on continuous attributes. It seems unlikely that one would want to interpret two thresholds that differ by an amount greater than 15% as being equal. At this setting, two decision regions could differ in length along dimension a by almost one-third (30%) of a's range, yet be considered equal by the stability measure. In addition to recording the stability of the induced classifiers, the error rate and model size (number of leaf nodes) were also tracked. The latter two quantities were estimated by 2-fold cross-validation, using the random divisions of the data that were produced by the stability estimation procedure. Each time the data was split, two classifiers were induced, and so two values each for error rate and model size were obtained. Hence, the final estimates for these statistics were computed from $2t$ values. On the other hand, the semantic and structural stability measures were computed each pair of classifiers, thereby yielding a single score for each split of the data. Hence, the final stability estimates are averaged over t values.

2.5.3 C4.5 and the DKM Splitting Criterion

The tree-growing procedure of C4.5 Release 8 [44] was modified to permit the use of DKM as the splitting criterion. Previous studies involving DKM and C4.5 made comparisons between the DKM gain and information gain criteria [14, 19]. However, the default criterion used by C4.5 is not information gain – it is the *gain ratio*. An objective of this work is to measure the stability of C4.5 as it is typically used in practice, and to determine whether an alternative splitting criterion can improve its stability; thus, the DKM results are compared to those obtained when using the default gain ratio criterion.

Instead of choosing the test that has the highest information gain among the candidate splits, the gain ratio criterion divides the gain by the so-called "split information." The latter value measures the total amount of information required to describe the outcome of the test t for each example, without considering class membership; it is minimal when all examples yield the same outcome, and is maximal when each example is propagated along a unique branch. The split information is calculated as $-\sum_{i=1}^{n} \frac{|L_i|}{|L|} \cdot \log_2\left(\frac{|L_i|}{|L|}\right)$, where L_i is the number of examples in the i^{th} subset that is produced by performing test t on examples L; the gain ratio is then computed as the gain divided by the split information. Ranking candidate tests according to this ratio serves to alleviate the bias that information gain exhibits in favour of splits that produce a large number of outcomes. The DKM gain criterion is also subject to this bias. An example offered by Quinlan [42, p. 23], which is paraphrased here, helps to illustrate the problem. Suppose that in a medical diagnosis problem, the dataset contains a discrete attribute that records a unique identifier for each patient. A test t on this attribute yields a maximal gain according to either splitting criterion, since t partitions the data into pure subsets, each containing a single example. Unfortunately, the resulting tree has little or no predictive power. On the other hand, the gain ratio criterion assigns a low score to the split t, which is appropriate in this case.

The DKM criterion was derived only for learning tasks involving two classes; an extension to multi-class domains is an open problem [14]. Although only binary-valued class attributes exist in the datasets used here, DKM's inability to measure the impurity of more than two subsets presents an obstacle in attempting to formulate a "DKM gain ratio" criterion that is analogous to the information gain ratio used by C4.5. The problem is that, as described in the previous paragraph, C4.5 calculates a quantity called the split information, which is an information-theoretic measure that is computed over the n subsets produced by a candidate test. Since a test on a discrete attribute can have $n > 2$ outcomes, a DKM-based measure that is applicable to multiple subsets is needed in order to mimic the calculation carried out by C4.5. In this research, the problem is side-stepped by simply using the DKM gain criterion, and comparing its performance to that of the information gain ratio criterion. For brevity, these will typically be referred to as DKM and entropy, respectively.

An additional issue must be addressed when using the DKM criterion with C4.5. The tree-growing algorithm penalizes the score of any candidate split made on a continuous attribute. Specifically, when considering a split on a continuous attribute a, a value equal to $\frac{1}{n}\log_2(u)$ is deducted from the gain, where n is the number of examples at the current tree node and u is the number of possible thresholds for a, based on these n examples (this is explained below). The application of this penalty term has three effects. First of all, it reduces the scores of candidate splits on continuous attributes relative to splits on discrete attributes. Secondly, since the value of the penalty term can vary for different continuous attributes, it can cause candidate splits on continuous attributes to be re-ranked. Finally, in cases where the unmodified gain associated with the best split on a continuous attribute a is less than or equal to the penalty term, applying the penalty drops the gain to zero. C4.5

does not perform a split when all candidate splits have zero gain; thus, the penalty can cause this stopping criterion to be invoked where a split would otherwise be made.

The penalty that C4.5 deducts from the scores of continuous splits is derived from information theory and the MDL principle. Its underlying motivation is that a split on a discrete attribute can be described by identifying the selected attribute, whereas for a split on a continuous attribute, one must additionally specify the threshold that is chosen. According to the MDL principle, the extra encoding cost for a continuous split must be taken into account when choosing between hypotheses. Due to the fact that the DKM criterion is not based on information theory, there is no theoretical justification for deducting this particular term from the DKM gain. Yet, C4.5's bias in favour of continuous attributes with many distinct values is due to the split-selection procedure that it employs, and is not a symptom of any particular splitting criterion. As C4.5 with entropy was shown to grow smaller, more accurate trees, on average, when using the penalty [44], it is plausible that DKM may benefit from a similar correction. In previous studies of DKM [14, 19], the penalty was disabled entirely, so that the unmodified gain was calculated for each criterion. However, as stated, a goal of this work is to study the stability of C4.5 as it is used in practice. Therefore, the penalty for the entropy criterion remains intact, and a penalty term that is appropriate for DKM is estimated empirically. The details of this estimation procedure, as well as experiments that demonstrate the effect of the two penalty terms, are documented in Appendix B.

The estimated penalty is used with the DKM criterion for the remaining experiments in this thesis. Deriving a penalty term for DKM that optimizes performance is not the focus of this research, and is left as future work. The estimation procedure described in Appendix B reduces the scores of candidate splits on continuous attributes in a manner that is comparable to the penalty applied by the unmodified C4.5 software; it is calibrated for DKM scores, as opposed to entropy scores. Also, the experimental data in Appendix B demonstrates that the estimated penalty succeeds in moderating the complexity of the trees induced by DKM. Ultimately, the objective here is to make the penalty term a neutral factor when comparing the two splitting criteria, thereby removing, or at least mitigating what might otherwise be an additional source of variance.

2.6 Experimental Results

2.6.1 Predictive Accuracy, Tree Size, and Semantic Stability

The experimental data gathered from the 25 datasets revealed little difference between DKM and entropy in terms of error rate (see Table 2.4). Entropy yielded more accurate trees 12 times, whereas DKM was superior 9 times on this performance metric. Only twice did the mean error rates of the two splitting criteria differ by more than .01; this occurred on the breast-cancer dataset, where entropy was better by .016, and on the sonar dataset, where DKM was better by .014. The standard deviation of the differences between the paired error rates (not shown) was less than .02 on 14 of

the datasets tested, less than .05 on 22 of the datasets, and was never more than .07. That the two criteria performed similarly with respect to classification accuracy is convenient for the purpose of studying stability, as it can be assumed that any improvement in stability that is observed on a particular dataset does not correspond with a significant loss in predictive accuracy. Similarly, neither criterion consistently grew smaller trees than the other; the trees produced by entropy were less complex (i.e. contained fewer leaf nodes) on 13 of the 25 datasets, and no draws were recorded. The largest ratio of tree sizes was observed on the breast-cancer dataset, where the trees grown by entropy and DKM contained a mean of 13.3 and 8.9 leaves, respectively. The relative size of the trees is noted only to reiterate that the penalty term estimated for DKM prevents excess tree growth that would otherwise have materialized on some of the datasets, as was discussed in Section 2.5.3.

Next, the semantic stability of the trees induced using the two splitting criteria is compared. In Table 2.4, for each dataset and splitting criterion, the mean of the semantic stability scores is listed in the "Agreement" column, followed by the standard deviation. One immediately notices that the standard deviations of the stability scores are uniformly larger than the standard deviations of the error rates. For the entropy criterion, the standard deviation of the semantic stability scores was .073, on average, while the average standard deviation of the accuracy scores was just .021 (the values for DKM were .07 and .021, respectively). This phenomenon can be explained by the fact that two hypotheses may be equally accurate, yet may make different predictions on two or more examples in the evaluation set. Given that the stability scores are highly variable, it is important to understand how the *differences* between the scores vary, in attempting to distinguish entropy from DKM. For instance, one criterion could have been systematically more stable than the other, even though the scores for both criteria varied significantly. This turns out not to be the case – the standard deviation of the differences between the scores (not shown) was less than the standard deviations of the scores for both splitting criteria on only 5 of the 25 datasets.

If one splitting criterion C_1 is to be considered more stable than another criterion C_2 on a particular dataset, then trees grown from different training samples using C_1 must consistently be more stable than the trees produced by C_2 from these samples. If the stability scores for two splitting criteria exhibit high variance, then it may be misleading to compare the averages of these scores, since the criterion having the higher average score could potentially be less stable, by a small margin, on the majority of training samples. In analyzing these results, to determine whether one splitting criterion was "significantly" more stable than the other on a given dataset,[7] the statistical median m of the score differences on the t trials, namely, $m = median(dkm_1 - ent_1, dkm_2 - ent_2, \ldots, dkm_t - ent_t)$, was calculated. For a given dataset, if m was greater than or equal to 0.05, then a win was recorded for DKM, whereas a win was awarded to entropy if m was less than or equal to -0.05; otherwise, a draw was declared. This standard ensures that C_1 is assessed as being superior to C_2 in terms of stability only if it is *consistently* more stable than C_2 (i.e. on at least half of the trials), and

[7]The word "significant" is enclosed in quotes for the remainder of this chapter, as a reminder that this is an ad hoc definition of significance, and does not imply classical statistical significance.

Dataset	Agreement						Error rate						Number of leaf nodes					
	Entropy			DKM			Entropy			DKM			Entropy			DKM		
anneal	.887	±	.066	.833	±	.079	.122	±	.018	.126	±	.017	25.6	±	5.2	25.3	±	4.8
australian	.864	±	.136	.975	±	.051	.146	±	.017	.146	±	.017	13.6	±	9.1	9.4	±	12.0
breast-cancer	.591	±	.075	.662	±	.147	.277	±	.033	.293	±	.029	13.3	±	9.2	8.9	±	9.5
car	.933	±	.019	.928	±	.020	.067	±	.010	.068	±	.009	42.3	±	6.0	41.7	±	6.0
dermatology	.753	±	.142	.696	±	.117	.021	±	.009	.022	±	.009	5.5	±	1.7	5.4	±	1.7
german	.656	±	.075	.764	±	.051	.289	±	.019	.291	±	.019	51.9	±	14.4	49.3	±	17.5
glass	.673	±	.117	.652	±	.112	.257	±	.053	.256	±	.055	7.6	±	2.0	8.3	±	2.3
heart-c	.593	±	.074	.568	±	.077	.241	±	.034	.238	±	.033	14.3	±	3.7	16.4	±	4.3
heart-h	.546	±	.114	.511	±	.086	.233	±	.031	.243	±	.030	8.8	±	4.2	10.9	±	5.0
heart-statlog	.585	±	.078	.570	±	.079	.246	±	.037	.240	±	.037	13.2	±	3.4	15.3	±	3.8
ionosphere	.750	±	.069	.737	±	.078	.116	±	.027	.124	±	.026	8.3	±	1.8	10.1	±	1.8
kr-vs-kp	.975	±	.018	.975	±	.010	.010	±	.003	.011	±	.003	24.6	±	1.9	25.1	±	2.6
letter	.887	±	.025	.931	±	.014	.013	±	.001	.013	±	.001	68.9	±	7.2	75.5	±	8.1
lymphography	.557	±	.090	.640	±	.145	.222	±	.047	.214	±	.050	8.2	±	2.6	7.2	±	2.9
nursery	.963	±	.006	.957	±	.006	.046	±	.004	.049	±	.004	218.9	±	17.9	221.2	±	18.5
pendigits	.766	±	.040	.795	±	.041	.013	±	.002	.013	±	.002	39.4	±	3.7	39.5	±	4.3
pima-indians	.666	±	.107	.699	±	.113	.272	±	.022	.272	±	.023	15.7	±	6.2	12.8	±	6.1
segmentation	.775	±	.071	.811	±	.056	.017	±	.005	.018	±	.005	13.2	±	2.5	14.1	±	2.3
sonar	.567	±	.079	.623	±	.070	.300	±	.050	.286	±	.052	9.3	±	1.2	9.7	±	1.2
tic-tac-toe	.735	±	.050	.723	±	.045	.188	±	.025	.192	±	.025	56.6	±	8.5	55.4	±	8.6
vehicle	.662	±	.092	.687	±	.091	.243	±	.021	.241	±	.022	30.8	±	8.1	29.8	±	8.8
vote	.846	±	.074	.868	±	.061	.046	±	.014	.044	±	.013	3.6	±	1.5	3.5	±	1.6
vowel	.860	±	.032	.878	±	.021	.046	±	.012	.045	±	.012	11.2	±	2.2	13.0	±	2.1
wdbc	.709	±	.077	.724	±	.087	.069	±	.016	.068	±	.015	7.8	±	1.6	8.3	±	1.7
yeast	.695	±	.088	.690	±	.087	.252	±	.015	.254	±	.015	19.6	±	7.7	16.7	±	7.3

Table 2.4: Mean semantic stability scores, error rates, and tree sizes for the entropy and DKM splitting criteria. The value to the right of the ± sign represents one standard deviation.

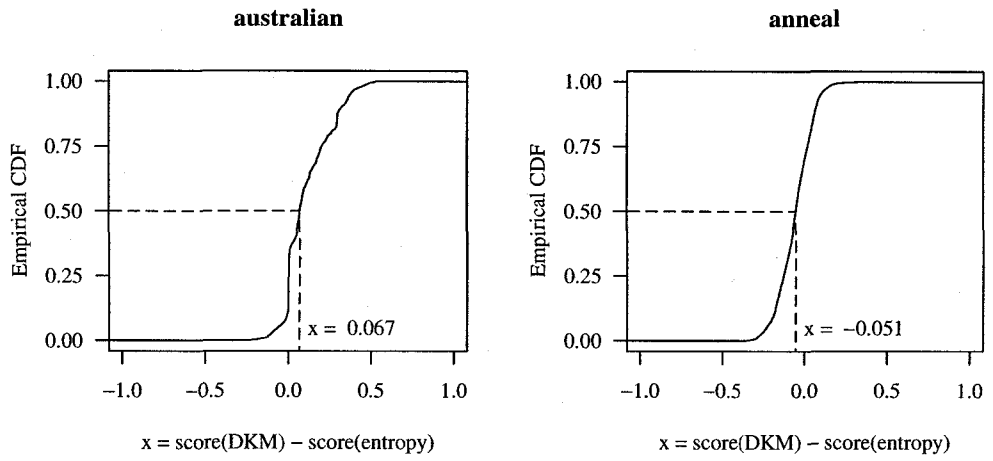

Figure 2.4: Empirical CDFs of the paired semantic stability score differences on the australian (left) and anneal (right) datasets. A vertical dashed line indicates the median score.

by a non-trivial amount (i.e. at least a 5% improvement).

The empirical cumulative distribution function (ECDF) is used here to visualize the results for a given dataset. The value of the ECDF at x is the proportion of the observations (i.e. score differences) that are less than or equal to x. Consider the ECDF pictured in Figure 2.4 (left), for example. The range of score differences is displayed along the x-axis; for a given x, the value on the y-axis indicates the fraction of observations that were at most x. The median of the score differences is the x value for $y = 0.50$, which in this case is $x = 0.067$, and is indicated by a vertical dashed line (the horizontal dashed line is included to indicate graphically that this is the median observation). Thus, on this dataset, half of the score differences were greater than 0.067; since the median is above 0.05, this is considered to be a win for DKM.

Applying the above standard to the semantic stability scores that were obtained on the 25 datasets, there were only 6 cases where one splitting criterion was declared to be "significantly" more stable than the other. Wins were recorded for DKM on the australian, breast-cancer, german, lymphography, and sonar datasets, whereas entropy was found to be superior on the anneal dataset. Figure 2.4 plots the ECDF of the score differences between DKM and entropy for the australian dataset (left) and the anneal dataset (right). On both of these datasets, the score differences were consistently distributed away from zero, and the magnitude of the differences was often substantial. In contrast, on most of the datasets, the paired score differences were either highly variable and fluctuated above and below zero, or else one criterion frequently scored higher than the other, but only marginally so. The former situation is depicted in Figure 2.5 (left), which plots the ECDF for the heart-h dataset. Here, the score differences varied across a wide range, from approximately -0.65 to 0.52. On the nursery dataset, for which the results are depicted in Figure 2.5 (right), entropy

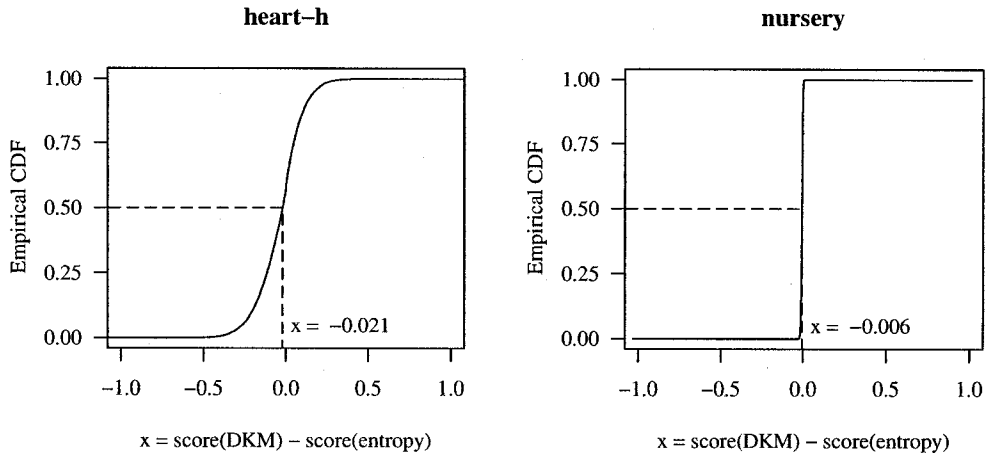

Figure 2.5: Empirical CDFs of the paired semantic stability score differences on the heart-h dataset (left) and nursery dataset (right). A vertical dashed line indicates the median score.

was more stable than DKM on 93% of the trials; however, the differences were minuscule. As a result, it is argued that there is no practical advantage to using the entropy criterion on this dataset, with respect to semantic stability. The method used to assess the "significance" of improvements in stability captures this intuition.

2.6.2 Absolute Structural Stability

In this section, the structural stability scores calculated using $\varepsilon_{pct} = 0$ are examined. Recall that by setting the ε_{pct} parameter of the region stability metric to zero, a maximal score for two trees T_1 and T_2 is realized if and only if the decision regions defined by T_1 and T_2 are exactly the same. That is, a perfect stability score of 1 does not imply that T_1 and T_2 are identical, since the order in which tests are performed by the two trees need not be the same. The standard applied for determining "significant" differences between structural stability score distributions is the same as the one applied when analyzing semantic stability. To reiterate, the median m of the score differences is computed; a win is assigned to DKM if $m \geq 0.05$, a win is recorded for entropy if $m \leq -0.05$, and a draw is declared otherwise. The results are summarized in Tables 2.5 and 2.6. These tables display the average stability scores of the two splitting criteria on each dataset. In each cell, the average score for DKM is shown on the left side of the slash, and the average score for entropy is presented on the right side. The column headers show the value of ε_{pct} that was used to obtain the stability scores in each column – the first column is an exception, as it contains the semantic stability scores, which are duplicated from Table 2.4. Bold entries indicate a "significant" win for the criterion with the higher score, based on the "median $\pm$ 0.05" standard described above. Table 2.5 shows the scores for datasets containing discrete attributes only, while Table 2.6 displays the scores for datasets containing at

Dataset	Semantic	0%
breast-cancer	**.662 / .591**	.109 / .036
car	.928 / .933	.297 / .375
dermatology	.696 / .753	.147 / .284
kr-vs-kp	.975 / .975	.862 / .837
nursery	.957 / .963	.785 / .767
tic-tac-toe	.723 / .735	.079 / .096
vote	.868 / .846	.568 / .553

Table 2.5: Mean stability scores (DKM / entropy) for datasets containing discrete attributes only. Bold entries indicate that the absolute value of the median difference between scores is at least 0.05. The "0%" column displays the structural stability scores for $\varepsilon_{pct} = 0$.

least one continuous attribute. Since absolute structural stability is currently being examined, only the "0%" columns in each table are relevant to this discussion.

There are 4 datasets, namely, australian, breast-cancer, german, and lymphography, on which the trees produced by DKM are more structurally stable than those of entropy when scores are calculated using $\varepsilon_{pct} = 0$; entropy never performs "significantly" better than DKM based on this measure. On these datasets, the increase in mean structural stability ranges from .214 to .305. These are substantial increases in relation to the semantic stability gains that were made, which were never greater than .111. Figure 2.6 shows the ECDF of the structural stability score differences on the australian dataset, where the average improvement exhibited by DKM over entropy was .305. On this dataset, although the discrepancies between the two splitting criteria are negligible on nearly half of the trials, the remaining scores almost exclusively favour DKM, usually by a margin of 0.5 or greater. It turns out that, for many divisions of the training data, DKM grew pairs of trees that consisted of a single split on the binary-valued attribute 'A9' (and two leaf nodes), thereby achieving a maximal structural stability score of 1. In contrast, from the same divisions of the data, entropy usually produced pairs of trees that contained more nodes, and which frequently achieved stability scores in the neighbourhood of 0.5. While DKM was less sensitive to variations in the training examples on this dataset, it is important to note that, as recorded in Table 2.4, the two splitting criteria achieved identical error rates in this case. Thus, the improvement in structural stability attributable to DKM is welcome, as it does not correspond with a loss of accuracy.

On the datasets that contain only discrete attributes, no "significant" differences between the two splitting criteria were identified when using the region stability metric. For example, on the breast-cancer dataset, DKM produced models that were semantically more stable than those created with entropy, yet these models exhibited no "significant" improvement in terms of structural stability. Figure 2.7 plots the ECDF of the paired semantic score differences (left), as well as the differences between the structural scores (right) on this dataset. These plots indicate that DKM grew trees that were more likely to agree on the class label of an unseen example $\mathbf{x}$ compared to trees grown using entropy, but that the former were no more likely than the latter to test the same set of features when

Dataset	Semantic	0%	1%	2%	3%	4%	5%	6%	7%
anneal	**.833 / .887**	.129 / .074	.143 / .075	.152 / .081	.158 / .081	.158 / .081	.158 / .085	.260 / .090	.261 / .090
australian	**.975 / .864**	**.656 / .351**	**.834 / .422**	**.835 / .424**	**.837 / .426**	**.839 / .427**	**.871 / .449**	**.872 / .450**	**.872 / .450**
german	**.764 / .656**	**.450 / .207**	**.451 / .207**	**.451 / .207**	**.451 / .207**	**.451 / .207**	**.451 / .210**	**.451 / .216**	**.452 / .217**
glass	.652 / .673	.000 / .000	.006 / .007	.020 / .023	.024 / .028	.028 / .033	.030 / .035	.032 / .038	.032 / .038
heart-c	.568 / .593	.017 / .010	.036 / .015	.036 / .015	.036 / .015	.037 / .015	.037 / .015	.037 / .015	.038 / .015
heart-h	.511 / .546	.002 / .002	.003 / .002	.003 / .002	.003 / .002	.006 / .006	.006 / .006	.006 / .006	.007 / .006
heart-statlog	.570 / .585	.024 / .016	.038 / .023	.039 / .023	.039 / .023	.039 / .023	.040 / .023	.040 / .023	.040 / .024
ionosphere	.737 / .750	.000 / .000	.000 / .000	.124 / .079	.223 / .134	.247 / .156	.293 / .204	.293 / .205	.332 / .247
letter	.931 / .887	**.349 / .135**	**.349 / .136**	**.349 / .136**	**.349 / .136**	**.349 / .136**	**.349 / .136**	**.349 / .136**	**.457 / .195**
lymphography	**.640 / .557**	**.299 / .020**	**.318 / .027**	**.318 / .027**	**.318 / .027**	**.318 / .027**	**.318 / .027**	**.318 / .027**	**.318 / .027**
pendigits	.795 / .766	.000 / .007	.000 / .082	.003 / .128	**.007 / .252**	**.010 / .278**	**.010 / .278**	**.023 / .326**	**.029 / .333**
pima-indians	.699 / .666	.000 / .000	.001 / .000	.003 / .001	.010 / .004	.015 / .006	.019 / .008	.028 / .013	.037 / .019
segmentation	.811 / .775	.000 / .000	.000 / .008	.010 / .024	.010 / .039	.010 / .040	.025 / .043	.035 / .047	.035 / .049
sonar	**.623 / .567**	.000 / .000	.000 / .000	.000 / .000	.000 / .000	.000 / .000	.001 / .000	.001 / .000	.001 / .000
vehicle	.687 / .662	.000 / .000	.000 / .000	.000 / .000	.001 / .002	.001 / .002	.003 / .009	.003 / .011	.004 / .014
vowel	.878 / .860	.000 / .000	.000 / .016	**.532 / .072**	**.572 / .079**	**.592 / .083**	**.604 / .086**	**.607 / .087**	**.613 / .100**
wdbc	.724 / .709	.000 / .000	.000 / .000	.008 / .002	.034 / .009	.039 / .011	.045 / .012	.050 / .013	.054 / .014
yeast	.690 / .695	.000 / .000	.004 / .001	.009 / .004	.013 / .007	.018 / .011	.022 / .013	.027 / .016	.034 / .020

Table 2.6: Mean stability scores (DKM / entropy) for datasets containing at least one continuous attribute, calculated using values of ε_{pct} ranging from 0% to 7%. Bold entries indicate that the absolute value of the median difference between scores is at least 0.05. (Continued in Table 2.7)

Dataset	8%	9%	10%	11%	12%	13%	14%	15%
anneal	.266 / .090	.267 / .090	.267 / .092	.268 / .092	**.313 / .446**	**.313 / .446**	**.365 / .452**	.440 / .469
australian	**.875 / .450**	**.877 / .450**	**.878 / .450**	**.879 / .451**	**.879 / .451**	**.879 / .451**	**.879 / .451**	**.880 / .451**
german	**.452 / .220**	**.452 / .220**	**.452 / .220**	**.453 / .221**	**.453 / .222**	**.453 / .222**	**.453 / .222**	**.453 / .222**
glass	.033 / .039	.035 / .040	.035 / .040	.038 / .042	.039 / .042	.040 / .043	.042 / .046	.044 / .048
heart-c	.038 / .015	.039 / .015	.042 / .016	.042 / .016	.042 / .016	.042 / .017	.042 / .017	.042 / .017
heart-h	.007 / .006	.007 / .006	.007 / .006	.012 / .017	.012 / .017	.012 / .017	.013 / .018	.013 / .018
heart-statlog	.040 / .024	.041 / .024	.044 / .024	.044 / .025	.045 / .025	.045 / .025	.045 / .026	.046 / .026
ionosphere	.332 / .247	.335 / .253	.358 / .299	.364 / .311	.383 / .352	.383 / .352	.383 / .352	.383 / .352
letter	**.457 / .195**	**.457 / .195**	**.457 / .195**	**.457 / .195**	**.457 / .195**	**.457 / .195**	**.520 / .272**	**.520 / .272**
lymphography	**.318 / .027**	**.318 / .027**	**.318 / .027**	**.318 / .027**	**.318 / .027**	**.318 / .027**	**.318 / .027**	**.322 / .030**
pendigits	**.044 / .337**	**.049 / .340**	**.049 / .340**	**.124 / .350**	**.128 / .353**	**.134 / .356**	**.139 / .359**	**.139 / .359**
pima-indians	.044 / .023	.056 / .030	.063 / .034	.076 / .042	.088 / .054	.096 / .059	.108 / .070	.114 / .073
segmentation	.035 / .051	.035 / .053	.035 / .055	.035 / .058	.036 / .062	.036 / .064	.038 / .068	.038 / .149
sonar	.002 / .000	.002 / .000	.003 / .000	.003 / .000	.004 / .000	.004 / .001	.005 / .001	.006 / .001
vehicle	.006 / .014	.006 / .021	.006 / .021	.006 / .023	.006 / .027	.006 / .027	.006 / .031	.006 / .031
vowel	**.618 / .199**	**.627 / .323**	.631 / .374	.632 / .380	.633 / .386	.634 / .390	.634 / .392	.635 / .396
wdbc	.055 / .014	.055 / .015	.056 / .015	.057 / .016	.057 / .017	.059 / .018	.061 / .018	.063 / .020
yeast	.047 / .029	.052 / .032	.056 / .035	.057 / .036	.060 / .038	.061 / .041	.063 / .042	.064 / .043

Table 2.7: Mean stability scores (DKM / entropy) for datasets containing at least one continuous attribute, calculated using values of ε_{pct} ranging from 8% to 15%. Bold entries indicate that the absolute value of the median difference between scores is at least 0.05.

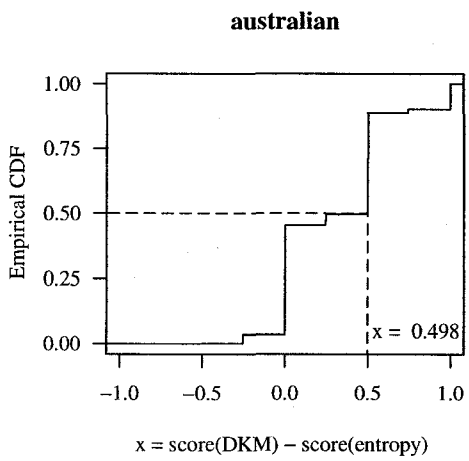

Figure 2.6: Empirical CDF of the paired structural stability score differences ($\varepsilon_{pct} = 0$) on the australian dataset. A vertical dashed line indicates the median score.

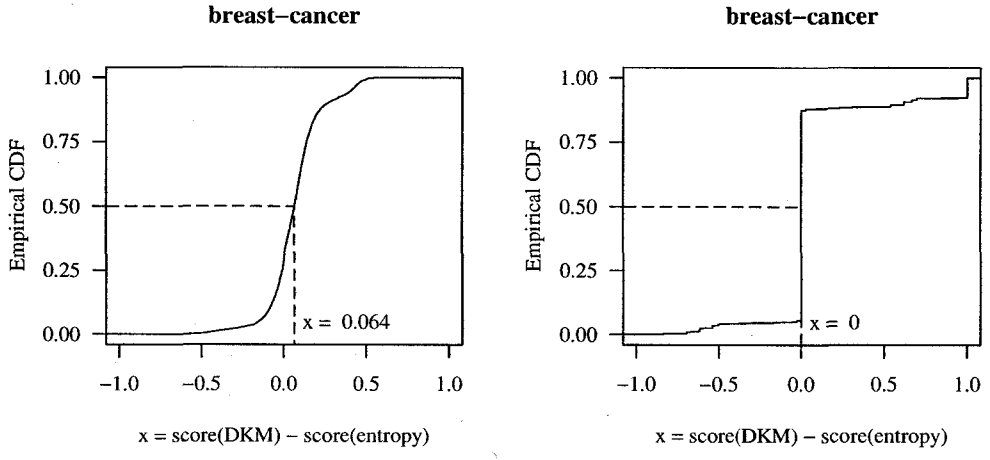

Figure 2.7: Empirical CDFs of the paired semantic stability score differences (left) and structural stability score differences (right, $\varepsilon_{pct} = 0$) on the breast-cancer dataset. A vertical dashed line indicates the median score.

classifying x. The characteristic that distinguishes these two score distributions is the prevalence of draws between the splitting criteria on the structural stability measure. In fact, for each of the breast-cancer, dermatology, tic-tac-toe, and vote datasets, the percentage of trials in which a draw was recorded exceeded 59%. Inspection of the scores for each splitting criterion, as opposed to their differences, revealed that, on the first three of these datasets, a large portion of these draws occurred when the trees grown by both DKM and entropy achieved zero structural stability.

The failure of both splitting criteria to consistently obtain positive structural stability scores in these cases was mainly due to the relatively low number of training examples that were presented to the learner. Among the datasets containing only discrete attributes, the ones on which C4.5 was the most unstable were three of the four smallest data sets in this group. Moreover, by the experimental procedure employed, only half of the examples in each dataset were made available to the learning algorithm, for the purpose of growing two trees from separate samples. As a consequence, pairs of trees induced from these samples often had very few, or no decision regions in common. The vote dataset was the lone exception to the trend of scoreless draws, as both splitting criteria consistently tested the 'physician-fee-freeze' attribute at the root of the tree. Although a large fraction of the trials resulted in draws, the stability scores for DKM and entropy were always positive due to the unanimous agreement on the root node split. This split invariably produced a leaf predicting the class 'democrat', thereby defining a decision region common to each tree. On the remaining datasets that have only discrete attributes, draws between DKM and entropy were less frequent, and the trees produced by both criteria were more generally more stable than the trees grown on the smaller datasets. However, neither criterion consistently outperformed the other, with respect to structural stability.

Finally, on 9 of the 11 datasets that consist entirely of continuous attributes, the structural stability scores for both splitting criteria were always zero. In many of these cases, the trees being compared closely resembled one another, with respect to the attributes tested and predictions made at the leaves; however, the thresholds that were defined by each tree tended to differ, often marginally. The two trees displayed in Figure 2.3, which were grown using the DKM criterion from a random division of the glass data, offer an example of such an occurrence. The fact that the region stability score for these two trees is zero scores reinforces the notion that some amount of leeway should be permitted when comparing tests on continuous attributes, since no information regarding the structural similarity of these trees may be communicated otherwise. Next, positive values ranging from 1% to 15% are assigned to ε_{pct}, in order to study the effect on the structural stability scores. Since comparisons between tests on discrete attributes are not affected by this parameter, no further analysis is conducted on the datasets that contain only discrete attributes.

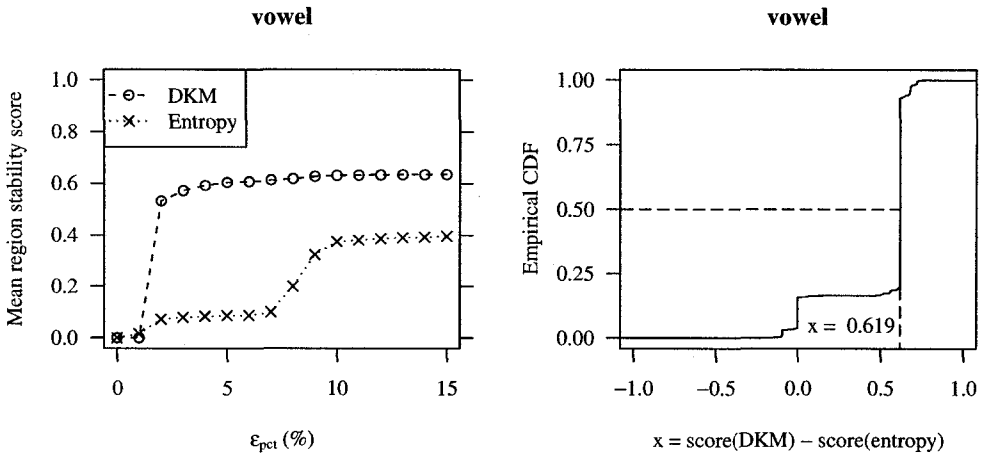

Figure 2.8: Structural stability scores on the vowel dataset. Left: Mean score as a function of ε_{pct}. Right: Empirical CDF of the paired score differences ($\varepsilon_{pct} = 5\%$).

2.6.3 Approximate Structural Stability

On some of the datasets that contained one or more continuous attributes, increasing ε_{pct} by a small amount caused the mean stability scores to rise considerably, indicating that trees were structurally similar, but differed marginally with respect to their decision thresholds. In other cases, increasing ε_{pct} had little or no impact on structural stability scores; in these situations, either the decision regions formed by two trees test different sets of attributes when classifying most examples, or the thresholds differ by amounts greater than the margin of error that was permitted. On many of the datasets, increasing the value of ε_{pct} had a similar effect on the scores of both splitting criteria, whereas it occasionally revealed "significant" differences between DKM and entropy. The vowel dataset presented an example of the latter outcome; with $\varepsilon_{pct} = 0$, the average structural stability for both criteria was 0, as was the median of the paired score differences. However, when the thresholds were permitted to differ by at most 2% of their respective attribute ranges, the trees constructed by DKM scored "significantly" higher on the region stability metric than did the trees built using entropy. Figure 2.8 (left) plots the average region stability of each splitting criterion as a function of the value of ε_{pct}. The stability scores of DKM bettered those of entropy by a margin of approximately .5 for values of ε_{pct} ranging from 2% to 7%. When ε_{pct} was between 8% and 10%, the stability for entropy increased sharply, and subsequently plateaued at a level that was just over .2 below the scores for DKM. The median paired difference between the scores was greater than .05 for values of ε_{pct} in the range of 2% to 9%; for all other values of ε_{pct}, the differences were "insignificant." Figure 2.8 (right) shows the score distribution for $\varepsilon_{pct} = 5\%$, which was heavily skewed in favour of DKM; the median difference in this case was .619.

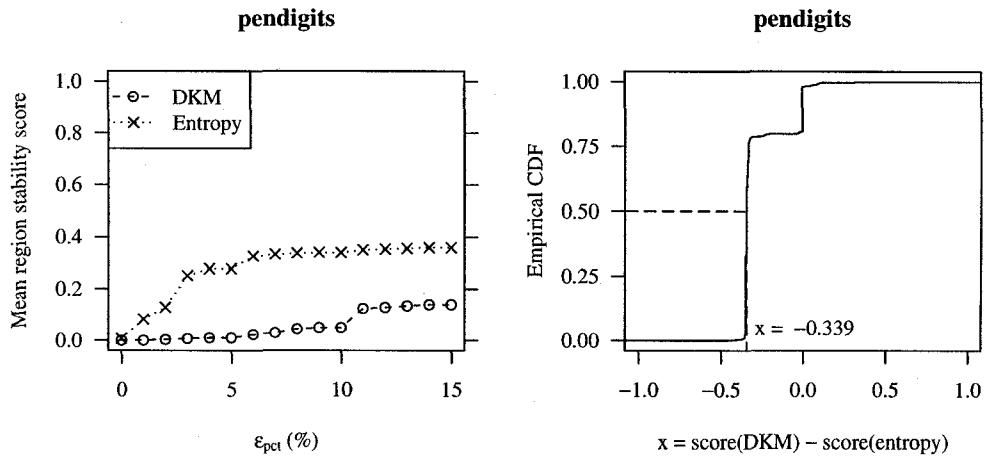

Figure 2.9: Structural stability scores on the pendigits dataset. Left: Mean score as a function of ε_{pct}. Right: Empirical CDF of the paired score differences ($\varepsilon_{pct} = 5\%$).

A similar phenomenon was observed on the pendigits dataset, except that in this case, it was the trees grown using entropy, rather than with DKM, whose thresholds were closer to one another. The plot of region stability versus ε_{pct} is displayed in Figure 2.9 (left) - the stability gains for entropy are "significant" for all values of ε_{pct} at or above 3%. In addition, Figure 2.9 (right) shows the ECDF for the paired score differences that were calculated using $\varepsilon_{pct} = 5\%$. When measuring and comparing the structural stability of decision trees, the behaviour of the splitting criteria on the vowel and pendigits datasets illustrates the potential insight that can be gained by ignoring small discrepancies between the thresholds that are defined by tests on continuous attributes.

The last notable finding that resulted from increasing the value of ε_{pct} was observed on the anneal dataset. In this case, there were no "significant" differences between the stability of trees produced by DKM and entropy while ε_{pct} was less than 12%, although the average region stability scores favoured DKM in all these cases (see Figure 2.10 (left)). At $\varepsilon_{pct} = 12\%$, however, the average score for entropy increased dramatically from its previous value of .092, to .446. This constituted a "significant" improvement over DKM, as the median of the score differences was -0.06 at this point. Figure 2.10 (right) displays the ECDF of the score differences for $\varepsilon_{pct} = 12\%$. The median remained at -0.06 until ε_{pct} reached 15%, at which point the gap between the stability scores of the two criteria narrowed once again. The sudden jump in the average stability for entropy was the result of three large decision regions that were consistently defined using the same pair of continuous attributes, but whose thresholds varied considerably. Although DKM defined these thresholds within a fairly close distance of one another on about a quarter of the trials, entropy placed them slightly farther apart (i.e. just over 10% of the respective attribute ranges) on nearly half the trials.

With the exception of the anneal, pendigits, and vowel datasets, no additional "significant" dif-

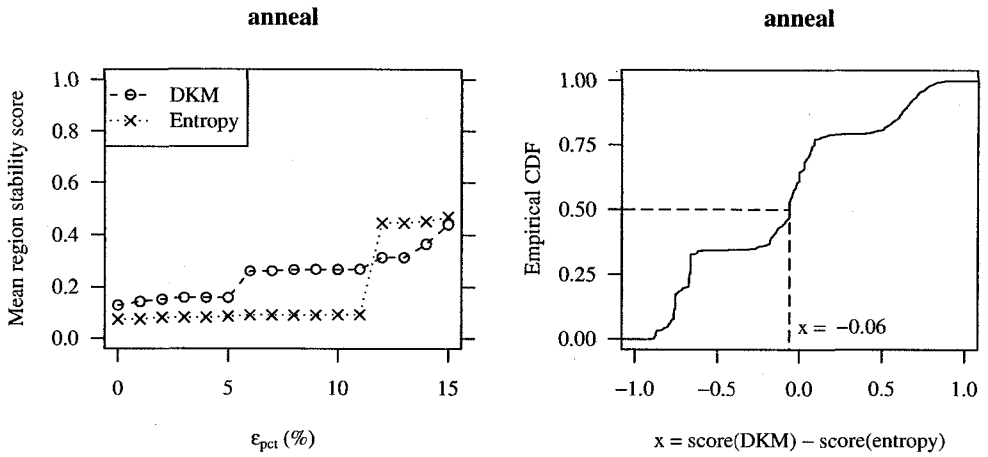

Figure 2.10: Structural stability scores on the anneal dataset. Left: Mean score as a function of ε_{pct}. Right: Empirical CDF of the paired score differences ($\varepsilon_{pct} = 12\%$).

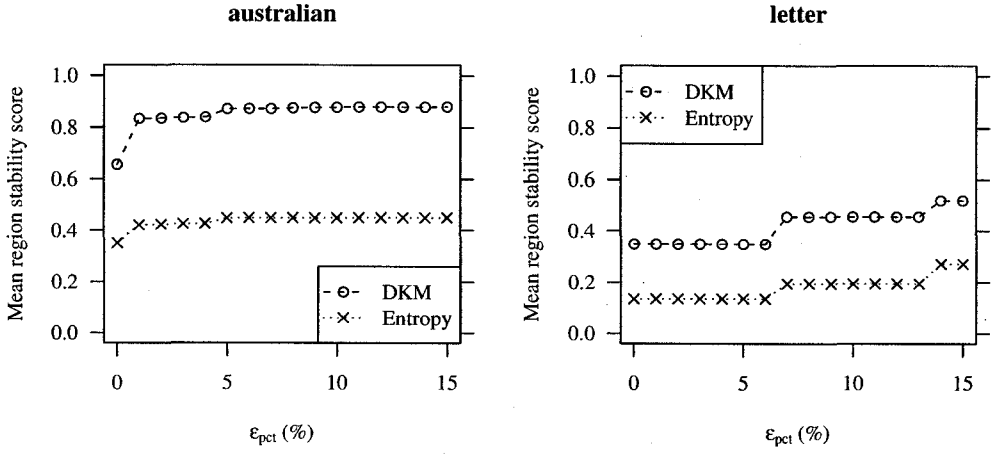

Figure 2.11: Mean structural stability score as a function of ε_{pct} on the australian (left) and letter (right) datasets.

ferences between the structural stability scores of the two splitting criteria were observed as a result of increasing the value of the ε_{pct} parameter above 0%. That being said, there were cases where the average difference between the stability of DKM and entropy varied as ε_{pct} was increased; however, the changes were not sufficient to push the median of the score differences outside of the range $(-0.05, 0.05)$. In terms of "significant" increases in structural stability, DKM maintained its superiority on each of the australian, german, letter, and lymphography datasets for all values of ε_{pct} that were used. The average region stability scores are plotted as a function of ε_{pct} in Figure 2.11 for the australian and letter datasets. In total, then, the number of "significant" stability improvements attributable to each splitting criterion was 5 for DKM and 2 for entropy.

2.6.4 Discussion

In this section, the practical implications of the improvements in structural stability that were observed on 7 of the 25 datasets are discussed. By inspecting the individual trials of the datasets on which one criterion was found to be convincingly more stable than the other, the "significant" increases in stability were generally associated with one of two sets of circumstances. The first possibility was that the trees achieving the high stability scores actually did have many or most of their decision regions in common. This is the primary motivation for using a stable decision tree algorithm; a learner that is less sensitive to small variations in the training data produces hypotheses that are consistently similar, and are therefore a more reliable source for knowledge extraction. As an illustration, consider the lymphography dataset, on which DKM achieved "significantly" higher region stability scores than entropy. In Figure 2.12, the two trees grown by each criterion during a typical trial are displayed. The trees grown by DKM were identical, except that the tree created from the second partition of the training data performed a test on the 'lymphatics' attribute that was not present in the first tree. In contrast, the trees grown by entropy differed considerably. Of the combined 8 attributes that are tested by the two entropy trees, only one feature, namely, 'block_of_affere,' appeared in both. Whereas the trees grown using DKM largely agreed on the attributes that were relevant to the diagnosis, the trees produced by entropy painted two very different pictures, making the task of interpreting these trees rather difficult. Hence, on this dataset, the higher structural stability scores achieved by DKM were indicative of a practical increase in model consistency.

The second state of affairs associated with one splitting criterion producing higher structural stability scores than the other was a situation in which a small fraction of the total number of decision regions in a tree tended to classify a large percentage of the unseen examples that were generated for the purpose of measuring stability. This trend was observed on the german and letter datasets, for example, where frequently a small number of leaf nodes, which were children of the root node or of decision nodes at relatively low tree depths, covered a large portion of the instance space. On both datasets, DKM defined these decision regions consistently across different training samples, whereas

Tree grown from 1st data partition using DKM:

```
changes_in_node = no: malign_lymph (2.0/1.8)
changes_in_node = lac_margin: metastases (38.0/6.1)
changes_in_node = lac_central: malign_lymph (13.0/3.6)
changes_in_node = lacunar:
|   special_forms = no: metastases (7.0/3.4)
|   special_forms = chalices: metastases (1.0/0.8)
|   special_forms = vesicles: malign_lymph (10.0/3.5)
```

Tree grown from 2nd data partition using DKM:

```
changes_in_node = no: malign_lymph (1.0/0.8)
changes_in_node = lac_margin: metastases (37.0/10.4)
changes_in_node = lac_central: malign_lymph (12.0/1.3)
changes_in_node = lacunar:
|   special_forms = no: metastases (4.0/1.2)
|   special_forms = vesicles: malign_lymph (11.0/1.3)
|   special_forms = chalices:
|   |   lymphatics = normal: malign_lymph (0.0)
|   |   lymphatics = arched: malign_lymph (3.0/2.1)
|   |   lymphatics = deformed: malign_lymph (1.0/0.8)
|   |   lymphatics = displaced: metastases (2.0/1.0)
```

Tree grown from 1st data partition using entropy:

```
no_of_nodes_in <= 3 :
|   block_of_affere = yes: metastases (40.0/4.9)
|   block_of_affere = no:
|   |   lym_nodes_enlar <= 1 : metastases (2.0/1.0)
|   |   lym_nodes_enlar > 1 : malign_lymph (14.0/5.8)
no_of_nodes_in > 3 :
|   bl_of_lymph_c = no: malign_lymph (10.0/1.3)
|   bl_of_lymph_c = yes:
|   |   regeneration_of = no: metastases (2.0/1.0)
|   |   regeneration_of = yes: malign_lymph (3.0/1.1)
```

Tree grown from 2nd data partition using entropy:

```
changes_in_node = no: malign_lymph (1.0/0.8)
changes_in_node = lac_central: malign_lymph (12.0/1.3)
changes_in_node = lacunar:
|   special_forms = no: metastases (4.0/1.2)
|   special_forms = vesicles: malign_lymph (11.0/1.3)
|   special_forms = chalices:
|   |   block_of_affere = no: malign_lymph (4.0/2.2)
|   |   block_of_affere = yes: metastases (2.0/1.0)
changes_in_node = lac_margin:
|   block_of_affere = yes: metastases (23.0/3.7)
|   block_of_affere = no:
|   |   special_forms = no: metastases (2.0/1.0)
|   |   special_forms = chalices: metastases (2.0/1.0)
|   |   special_forms = vesicles:
|   |   |   changes_in_lym = bean: malign_lymph (0.0)
|   |   |   changes_in_lym = oval: malign_lymph (4.0/1.2)
|   |   |   changes_in_lym = round: metastases (6.0/3.3)
```

Figure 2.12: Pairs of trees grown from the same random split of the lymphography dataset, first using DKM, and then using the entropy criterion.

entropy did not. Due to the fact that many of the unlabelled examples were classified by these few regions, the structural stability scores for DKM tended to be much higher than those of entropy, even though both splitting criteria grew large trees whose structure often varied considerably at greater tree depths. Thus, in these cases, the higher structural stability scores reflected the consistency with which DKM identified the most heavily-weighted decision regions, rather than being an indication that the trees produced by DKM tended to share many common regions. On the anneal dataset, it was the entropy criterion that was able to consistently define several large decision regions, albeit at rather high values of ε_{pct}. These occurrences highlight a fundamental difference between the proposed region stability metric and metrics such as *Common* [40], which give equal weight to each decision node. The latter type of measure would have assigned low stability scores to both splitting criteria on these two datasets, since much of the structure in these trees was highly variable. Unfortunately, such scores would not have reflected the fact that one criterion defined the largest, and arguably most important decision regions consistently, while the other failed to do so.

2.7 Conclusions

In this chapter, a methodology for measuring the structural stability of decision trees was proposed, which features the novel region stability metric, and employs a 2-fold cross-validation procedure for estimating stability [52]. Using the C4.5 decision tree algorithm, two different splitting criteria were compared in order to determine whether the use of an alternate criterion, known as DKM, would lead to the creation of trees that were more stable than the ones induced using the default gain ratio criterion, which has been referred to here as entropy.

The results of the experiments revealed that the two splitting criteria tested, namely, DKM and entropy, performed similarly with respect to classification accuracy and tree size on this collection of 25 benchmark datasets. Using the novel region stability measure, "significant" differences in structural stability were observed on 7 of these datasets – the DKM criterion grew more stable trees in 5 cases, whereas entropy did so on 2 occasions. An examination of the individual trials for the datasets where substantial stability improvements were achieved by one splitting criterion, according to the region stability metric, led to the identification of two causes for the "significant" differences between the scores, which were explored in Section 2.6.4. The semantic stability of the decision trees induced from these datasets was also recorded, using Turney's measure of expected agreement. Again, few "significant" differences between the two splitting criteria were identified, as DKM was more stable on 5 datasets compared to 1 victory for entropy. Superior semantic stability was an indicator of superior structural stability in only three cases: the australian, german, and lymphography datasets.

Even though 5 of the 7 substantial improvements in structural stability were achieved by the DKM splitting criterion, it cannot be concluded from these results that DKM produces "significantly" more stable trees than the default entropy criterion across a wide range of learning problems.

On many of the datasets that were used in these experiments, both criteria produced trees that were very unstable. With ε_{pct} set to 5%, which allows for a reasonable amount of discrepancy between thresholds for continuous tests, there were 10 datasets for which the average region stability of both criteria was less than .05. In other cases, where some appreciable level of stability was realized, it was either the case that 1) there was little or no difference between the scores achieved by DKM and those obtained by entropy, or 2) the differences were inconsistent, such that one criterion was superior on some of the trials, but performed worse on others.

Nevertheless, when taken as a whole, the experimental results do suggest that DKM is the preferable splitting criterion, since it is likely to produce trees that are at least as stable as those grown using entropy, and sometimes much more so. Consider, once again, the scores obtained using $\varepsilon_{pct} = 5\%$. Here, the average region stability score for DKM was greater than that of entropy on 17 datasets, and there was one draw. The tendency of DKM to obtain slightly better structural stability scores than entropy, combined with the "significant" increases in stability that were achieved by the former on 5 datasets, versus 2 for entropy, indicate that one should construct trees using DKM rather than with entropy if the aim is to maximize structural stability.

Chapter 3

Stability in Active Learning

In this chapter, the stability of active learning procedures is studied empirically. In active learning, the learner has the ability to choose points from the instance space on which to train a classifier. This is contrary to passive learning, such as in the previous chapter, whereby the learner plays no role in specifying the set of training examples; typically, examples are drawn at random from a fixed, but unknown distribution. By requesting labels only for those points that are expected to be particularly informative, active learning aims to reduce the number of labelled examples that are needed to achieve a given level of predictive accuracy. Although the ability of active learning methods to make more efficient use of unlabelled data has been well documented in the machine learning literature (e.g. [1, 11, 31, 47]), very little attention has been given to the stability of these techniques.

Active learning is an iterative procedure, in which a single iteration consists of inducing a classifier and then requesting that one or more new examples be labelled and added to the training set of the subsequent iteration. Depending on the particular active learning technique, these new examples are either removed from a given set of unlabelled instances or are generated by the active learner; class labels are then assigned by a human expert or an oracle. In this manner, the training set expands incrementally until the available labelling and/or computational resources are exhausted, or until some other stopping condition is satisfied. Although many classifiers are induced during the active learning procedure (i.e. one per iteration), it is typically the model created during the final iteration of active learning that is employed for subsequent classification tasks and/or interpreted in order to understand the prediction model.

The majority of active learning methods fall into one of three categories: *pool-based*, *stream-based*, and *membership queries*. In pool-based active learning (e.g. [1, 30]), a fixed pool of unlabelled examples is available, which is typically a tiny subset of the possible examples in the space. The learner may examine each example in the pool and then request the labels of the ones that are expected to be the most informative. The stream-based approach [10] differs in that unlabelled examples are presented sequentially in the form of a random stream, and the learner must decide whether or not to label each example as it is appears. Finally, learning by membership queries [3]

does not involve the use of an unlabelled collection of examples. Instead, the learner generates a number of attribute vectors and queries the oracle at these points to determine the class membership of each.

This chapter is concerned with the stability of active learning methods. By studying the stability of such procedures, the basic objective is to investigate the degree to which classifiers induced at various iterations of active learning, as well as across distinct initial training sets, differ structurally. The stability of several active learning methods is measured and compared, based on experiments conducted on several datasets. Each active learner studied in these experiments makes use of an arbitrary base learning algorithm, which in this case is the C4.5 decision tree inducer. In addition, the stability of C4.5 when using the DKM and entropy splitting criteria is compared within the context of active learning. The study is restricted to pool-based active learning methods, which are the most common among the three active learning paradigms. In this work, the class of pool-based active learners will be referred to as *selective sampling* procedures, as they are frequently described in the machine learning literature.

This chapter proceeds as follows: A generic selective sampling procedure is introduced in Section 3.1, as well as the approach that is taken in order to measure stability in this context. In Section 3.2, the particular selective sampling methods used in these stability experiments are described, followed by a description of the experimental procedure in Section 3.3. The results with respect to semantic and structural stability, as well as predictive accuracy, are presented in Section 3.4, which concludes by comparing the stability of the entropy criterion on the active learning tasks to that of DKM. Finally, the findings are summarized in Section 3.5.

3.1 Selective Sampling and Stability

In this section, the general selective sampling procedure is described, and the issue of stability is discussed in this context. The training set at the i^{th} iteration of selective sampling is denoted as L_i, and let $h_\Omega(L_i)$ be the hypothesis that the base learner Ω induces from L_i. The learner Ω is assumed to be deterministic; that is, running Ω on a given training set L_i produces a specific hypothesis $h_\Omega(L_i)$ with probability 1. Under this assumption, which is satisfied by the C4.5 algorithm, any differences between two hypotheses induced by Ω are attributable to differences in the training data used for each. Given a set of unlabelled data U_i and a (possibly empty) set of labelled data L_i, a given iteration i of selective sampling consists of the following three steps:

1. Train a classifier $h_\Omega(L_i)$ on the data L_i using the learning algorithm Ω.
2. Rank the examples in U_i according to some measure of expected informativeness (i.e. a scoring function). This measure is typically a function of $h_\Omega(L_i)$.
3. Remove the m highest-ranking examples from U_i, acquire their labels, and add the resulting labelled examples to L_i. This forms the sets U_{i+1} and L_{i+1}.

Henceforth, the subscript Ω is implicitly assumed for any hypothesis h. The parameter m is the batch size – the number of new labelled examples that are added to the training set during each iteration – and is often set to 1. However, in some applications the cost of training $h(L_i)$ and/or the cost of ranking the examples in U_i may be high relative to the cost of acquiring a label, in which case it may be favourable to label a batch of $m > 1$ examples during each iteration. Moreover, selective sampling procedures exist (e.g. [34, 47]) that define a probability distribution over U_i, and then choose examples to be labelled by sampling from this distribution; such methods typically assume that m is larger than 1. The potential drawback of using a large batch size is that some of the examples within the batch may be redundant, in which case labelling effort is wasted.

Before the first iteration of selective sampling, it is common practice to draw and label a number of examples from U to serve as the initial training data L_1. The examples that compose L_1 are usually selected at random from U, which is the approach taken here. It is worth noting, however, that other strategies have been proposed (e.g. [16]) that aim to minimize redundancy within the initial training set, and/or ensure that the minority class is adequately represented. It is assumed here that the batch size m remains fixed throughout the selective sampling procedure, and thus the training set L_i contains $|L_1| + i \cdot m$ labelled examples.

The stability of a selective sampling procedure S is measured with respect to the particular base learner that it uses. In this research, C4.5 is employed as the base learner. A method for estimating the stability of C4.5 was described in Section 2.5.2 that involved repeatedly applying the tree inducer to pairs of non-identical training sets. Then, a novel structural stability metric, referred to as the region stability metric, was applied to each pair of decision trees that were produced, and the scores were averaged. In order to measure the stability of a selective sampler S that uses C4.5 as its base learner, the region stability metric is instead applied to the hypotheses that are induced during different iterations of selective sampling. At each iteration i, the examples forming training set L_i, from which hypothesis $h(L_i)$ is produced, have been chosen by S. Thus, by calculating the stability of these hypotheses, a measure of the stability of S when using C4.5 is obtained.

The above approach to quantifying the stability of a selective sampling procedure S is not complete, as it remains to be defined exactly which pairs of hypotheses are to be assessed by the region stability measure. In passive learning experiments, the available data was randomly divided into two halves, and the decision trees induced from these training samples were compared. However, during a single run of selective sampling, dozens or even hundreds of trees may be grown from a single dataset. In order to procure relevant and meaningful information regarding the stability of S from this collection of hypotheses, three forms of active learning stability are proposed and measured. The first quantity that is measured is the hypothesis stability on consecutive iterations, and is called *PrevStab*. At iteration i, for $i > 1$, the region stability metric is applied to the current and previous hypotheses, which is the pair $(h(L_i), h(L_{i-1}))$. Calculating this score for all consecutive pairs of hypotheses produced during active learning reveals the amount of structural change that occurs as

a result of adding each new batch of examples to the training set. The second stability computation, called *FinalStab*, measures the similarity between the hypothesis at each iteration i and the hypothesis that is induced at the final iteration of selective sampling. Its purpose is to evaluate the manner in which the series of hypotheses progresses toward the final model, in terms of structural similarity. To reiterate a point made in Section 2.4, this is distinct from monitoring the error rate of each hypothesis and comparing these values to the error rate of the final model, since a stable error rate is a necessary, but not a sufficient condition for structural stability. The last type of stability that is considered is the stability across different initial training sets (or selective sampling runs), which is called *RunStab*. Given n runs of selective sampling, each using a distinct initial training set, the RunStab score at iteration i is obtained by calculating the region stability metric over each pair of hypotheses $(h(L_i^j), h(L_i^k))$, for runs $\{j, k\} \leq n, j \neq k$, and then taking the average of these scores. Here, L_i^j is the set of labelled examples formed at iteration i when using the j^{th} initial training set; also, the region stability scores are all computed with respect to the same evaluation set E. The RunStab metric quantifies the structural similarity between the decision trees induced at a particular iteration of different selective sampling runs. This may be interpreted as the degree to which a given selective sampling method produces structurally similar models from different initial training sets.

In active learning, the model induced during the final iteration warrants special consideration, as it is the end result or product of the sampling procedure. In contrast, the intermediate classifiers are typically discarded. As an example of how the proposed stability measures for active learning might be used and interpreted, consider the graphs displayed in Figure 3.1. These plot the FinalStab scores at each iteration of active learning against the fraction of pool examples that have been labelled and added to the training set, for the german and kr-vs-kp datasets. At each iteration, a C4.5 decision tree was induced using the entropy splitting criterion from the current training set; random sampling was used to select the next batch of examples to be labelled and added to the training set. This is identical to the generic selective sampling procedure described earlier in this section, except that the examples in the unlabelled pool are not ranked, but are chosen at random. Each FinalStab score was calculated by applying the region stability metric to the hypothesis induced from the given training set and the hypothesis induced from two-thirds of the available pool examples (this was the final hypothesis). A holdout set containing one-third of the examples in the entire dataset served as the evaluation set for the region stability metric, and the ε_{pct} parameter was set to 0. Finally, 10 runs were performed for each dataset using 10 different initial training samples, and the median score at each iteration was plotted. Note that the FinalStab score for the last iteration of active learning is necessarily 1, as it represents the similarity between the final model and itself; this point has been omitted from both graphs.

On the german dataset, the intermediate hypotheses tend to be structurally dissimilar from the final hypothesis. It is not until almost half of the examples in the pool have been labelled that the classifier begins to resemble the final model. The hypothesis induced during the second-last iteration

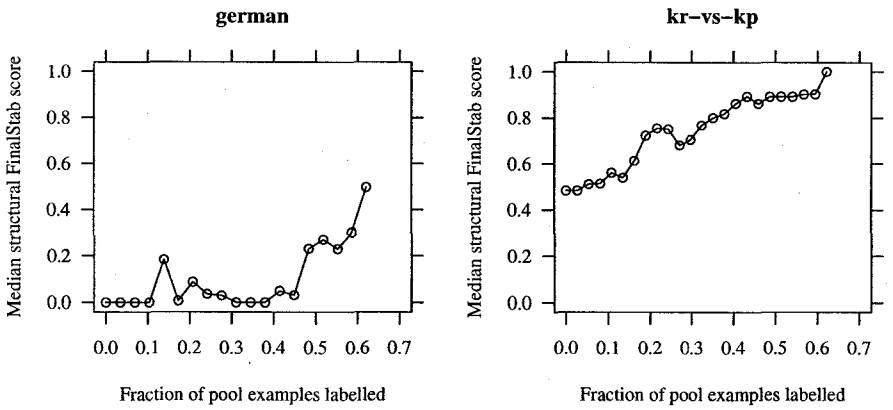

Figure 3.1: Active learning stability calculated using the FinalStab metric.

of active learning achieves a FinalStab score of approximately 0.5, indicating that the final batch of examples that is selected causes nearly half of the decision regions to change. By contrast, on the kr-vs-kp dataset, the sequence of hypotheses induced during active learning exhibit a relatively smooth progression toward the final model. The last 10 classifiers achieve FinalStab scores of at least 0.8, and the last batch of training examples that is added has no effect on the existing decision regions. Hence, in the case of the kr-vs-kp dataset, the FinalStab scores suggest that the final model is a stable representation of the target concept, and therefore may be useful for knowledge extraction. On the other hand, the final model induced from the german data is likely to be an artifact of the particular training sample from which it was induced; it may be misleading to interpret this model, as its decision regions are volatile and may not represent the underlying concept well. In practice, the error rate of these models would also have to be considered; however, for demonstration purposes, predictive accuracy is not discussed here.

In Figure 3.2, the median PrevStab scores for the same 10 runs (i.e. the same sequences of hypotheses) are plotted for the two datasets. The leftmost point in each graph is the region stability score comparing the model induced from the initial training data to the model induced after the first batch of examples has been labelled. In the case of the german data, the decision regions often change considerably between consecutive iterations, particularly in the early stages of active learning. On the contrary, consecutive models induced on the kr-vs-kp dataset never yield a region stability score of less than 0.7, and only slight adjustments are made to the hypothesis during each of the final 9 iterations. Thus, the PrevStab scores suggest that the decision trees induced from the kr-vs-kp data are generally resistant to the small changes in the training data that are made as new examples are incorporated, whereas the models produced from the german data tend to change significantly whenever a new batch of examples is labelled.

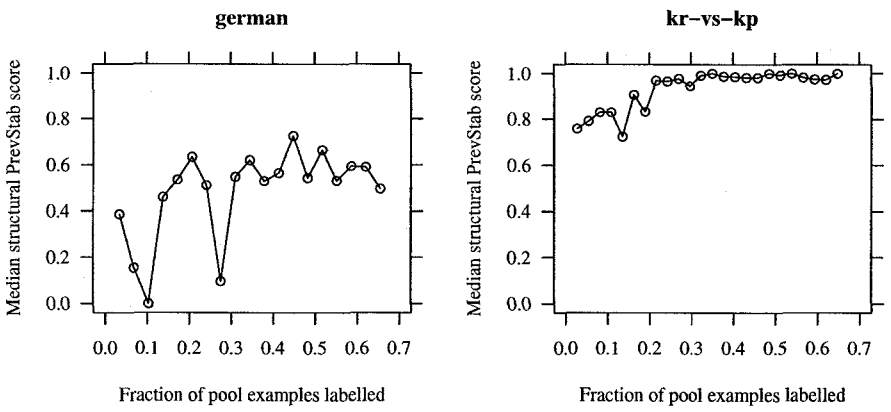

Figure 3.2: Active learning stability calculated using the PrevStab metric.

Plots of the structural RunStab scores for the two datasets are displayed in Figure 3.3. In each graph, the leftmost point is the RunStab score for the models induced from the 10 different initial training sets, and the rightmost point represents the structural similarity between the final models that were created. This metric highlights yet another dimension of C4.5's instability on the german dataset, as the models induced during different runs at a given iteration have little structure in common. The picture is quite different on the kr-vs-kp dataset, however, as the trees induced from the initial training sets share slightly fewer than half of their decision regions, and the similarity between the models increases at virtually every iteration. The final models produced by each run have approximately 80% of their decision regions in common.

In light of the structural stability estimates that were obtained via the experiments conducted in Section 2.6, the results presented in the preceding example are somewhat predictable. To recapitulate, in the experiments measuring structural stability with $\varepsilon_{pct} = 0$, C4.5 was found to be fairly stable on the kr-vs-kp dataset, but was considerably less stable on the german dataset. Thus, given the randomly drawn training samples from which the scores in this example were derived, it is not surprising that the base learner performs similarly. However, just as various selective sampling methods have been shown to improve the accuracy of C4.5 (e.g. [1, 30]), it is possible that the stability of active learning may improve when using C4.5 as the base learner in combination with certain selective samplers. Active learning often alters the distribution of training examples significantly, and the effect of such procedures on learner stability has not previously been studied.

When comparing the stability of different selective sampling procedures, some properties of the stability learning curves (i.e. the stability scores as a function of the training set size) are desirable. Ideally, the FinalStab scores increase as more examples are labelled and added to the training set, reaching reasonably high levels in the later stages of active learning. This is typically true of accuracy learning curves, for example. Whereas a consistently increasing level of accuracy indicates

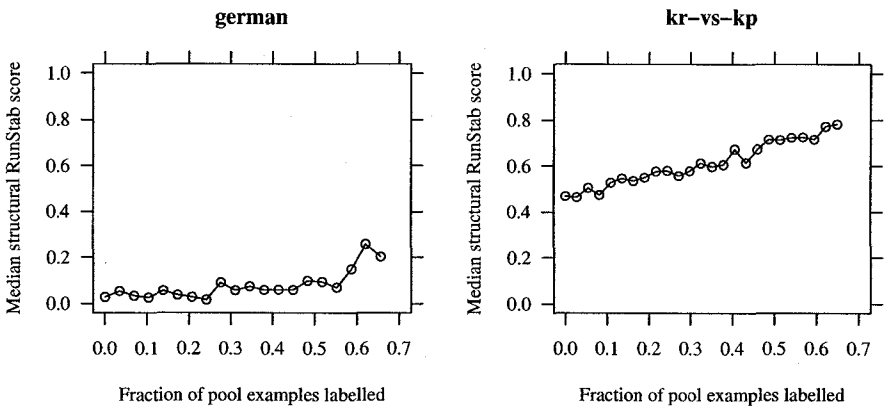

Figure 3.3: Active learning stability calculated using the RunStab metric.

that the base learner is inducing hypotheses that are better predictors than those learned during previous iterations, an increasing sequence of FinalStab scores implies that the learner is progressively producing models that have more structure in common with the final model. In other words, the examples chosen by a stable selective sampler have the effect of steadily guiding the base learner toward a particular hypothesis; on the contrary, the examples selected by an unstable sampler may cause the learner to induce increasingly accurate hypotheses, but these models may bear little resemblance to one another. With regard to PrevStab scores, which measure the structural similarity of models induced during consecutive iterations, these are generally expected to be low in the early stages of active learning, as the selective sampler explores the instance space. Ideally, as the sampling method begins to refine the hypothesis, the PrevStab scores increase and maintain a fairly high level. Put another way, if the selective sampler is successful in identifying the most informative examples during the early iterations of active learning, the remaining batches will tend to exert less influence on the hypothesis. This situation is characterized by a sharply increasing PrevStab curve that plateaus as the structural differences between consecutive models lessen. Finally, it is desirable for the RunStab scores yielded by a selective sampling method to be high, especially in the late stages of learning. If this is not the case, then the implication is that the selective sampler is highly sensitive to the particular examples that form the initial training set, as is the final model that is produced.

To summarize, structural stability is desired because it facilitates knowledge extraction. The three active learning stability metrics proposed here, which use the region stability metric described in Section 2.4 as a component, serve to quantify the stability of hypotheses that are produced during active learning.

Algorithm 1 Uncertainty-sampling

Require: Learning algorithm Ω, initial training set L_1, unlabelled pool U, batch size m, maximum number of iterations n

1: Initialize $U_1 \Leftarrow U$
2: Induce hypothesis $h(L_1)$ by applying Ω to the examples in L_1
3: **for** $i = 1$ to n **do**
4: Assign a score to each example $\mathbf{x} \in U_i$ which is inversely proportional to $h(L_i)$'s confidence in predicting the class label of $\mathbf{x}$
5: Label the examples $X_i^* \subseteq U_i$ which have the m highest scores
6: Set $U_{i+1} \Leftarrow U_i - X_i^*$ and $L_{i+1} \Leftarrow L_i + X_i^*$
7: Induce hypothesis $h(L_{i+1})$ by applying Ω to the examples in L_{i+1}
8: **end for**

3.2 Selective Sampling Methods

The selective sampling methods that are included in this study can generally be described as uncertainty-based approaches, as they each request the labels for examples whose class membership (or class probabilities) cannot be predicted confidently based on the currently available training data. In contrast, active learning techniques have been developed that optimize other criteria, such as the expected future error [46]. Note that each selective sampling procedure described here contains an implicit stopping condition, which is triggered when no examples remain in the unlabelled pool. For brevity, this detail has been omitted from the sampling method descriptions and pseudo-code; moreover, the experiments conducted in this work never exhaust the example pool, but instead terminate after a predetermined number of iterations.

The most simplistic selective sampling approach analyzed here is uncertainty sampling [30, 31]. It can be used with any base learner, such as C4.5, that is able to produce a confidence estimate when making a prediction on an unseen case. A decision tree classifies an unseen example $\mathbf{x}$ in a particular leaf l,[1] which contains n_+ training examples that are members of the positive (minority) class, and n_- examples from the negative (majority) class. The confidence in class $k \in \{+, -\}$ can be calculated as $\frac{n_k}{n_+ + n_-}$, and is defined to be 0.5 if l contains no training examples. During a given iteration i of selective sampling, the uncertainty sampling method induces a hypothesis $h(L_i)$ from the current training set L_i, which is then used to classify each unlabelled example $\mathbf{x} \in U_i$, in order to determine the confidence of the prediction for each unseen case. Uncertainty sampling requests the labels of the m examples on which $h(L_i)$ is least confident. Pseudo-code for uncertainty sampling is shown in Algorithm 1.

The remaining selective sampling methods examined in this study are committee-based methods, all of which are variations of the query-by-committee algorithm [50]. These procedures induce $c > 1$ classifiers, namely, $h_1, \ldots, h_c$, whose predictions are combined in some way for the purpose of ranking the unlabelled examples. Each committee member h_i has an associated vote weight

[1] This may not be the case if $\mathbf{x}$ has missing values for one or more of its attributes, although a confidence estimate can nevertheless be produced based on the total weight assigned to each class by the leaves that classify $\mathbf{x}$. As was explained in Section 2.5.1, missing values were removed from the datasets used in these experiments.

Algorithm 2 Bagging-committee

Require: Learning algorithm Ω, training set L, committee size c

1: Initialize $N \Leftarrow |L|$
2: **for** $i = 1$ to c **do**
3: Create L^i by randomly drawing N examples from L, with replacement, according to a uniform distribution
4: Induce hypothesis $h(L^i)$ by applying Ω to the examples in L^i
5: **end for**
6: **return** $h(L^1), \ldots, h(L^c)$

v_i that determines the amount of influence it exerts on predictions made by the committee. The committee-based methods that are considered each use one of two approaches (or variants thereof) to construct a committee from the training data: bagging [5] or boosting [23]. Next, these two committee-building techniques are introduced, and then five committee-based selective sampling methods that utilize these techniques are described.

Given a training set L consisting of N examples, bagging (short for bootstrap aggregation) creates c bootstrap replicates of L, denoted as $L^1, \ldots, L^c$, and then applies the learning algorithm to each training set L^i to produce $h(L^1), \ldots, h(L^c)$. The bootstrap technique [20] employed by bagging involves creating each L^i by drawing $|L|$ instances from L with replacement, according to a uniform distribution. Each committee member has unit vote weight in bagging; thus, given an unseen example $\mathbf{x}$, the committee outputs as its prediction the class that is assigned to $\mathbf{x}$ by the majority of its members. The procedure used to construct a bagging committee from a set of examples is displayed in Algorithm 2.

Boosting uses an iterative procedure to perturb the training data and form a committee. Instead of resampling, boosting assigns weights $\mathbf{w}_i$ to the training examples in L at iteration i, and changes the input data for the next committee member by altering these weights. Pseudo-code for the committee-building procedure used by Adaboost.M1 [22], which is the particular boosting algorithm used in this work, is shown in Algorithm 3. Initially, uniform weights are assigned to the instances. Then, for each iteration i, $1 \leq i < c$, hypothesis $h(L, \mathbf{w}_i)$ is induced from L with respect to $\mathbf{w}_i$. Subsequently, the weight $\mathbf{w}_{i+1}^{[j]}$ is decreased if $h(L, \mathbf{w}_i)$ predicts the correct class for $\mathbf{x}_j$, and is increased otherwise (see lines 10-14 of Algorithm 3). At iteration $i+1$, this scheme has the effect of focusing the learner on the examples that the previous hypothesis $h(L, \mathbf{w}_i)$ had difficulty explaining. The boosting committee also predicts according to the majority vote; however, unlike bagging, the vote weights are not uniform. Instead, boosting assigns a vote weight v_i to each committee member $h(L, \mathbf{w}_i)$, which is a function of the committee member's error rate.

Some learning algorithms cannot accommodate weighted training examples, as is required for building a boosting committee. However, each L^i can be created by resampling from L according to the weighted distribution $\mathbf{w}_i$ [45]. This way, L^i approximates the true weights $\mathbf{w}_i$, since the effective weight of each example in L^i is now a multiple of $\frac{1}{N}$. Fortunately, the C4.5 algorithm

Algorithm 3 Adaboost.M1-committee

Require: Learning algorithm Ω, training set L, committee size c

Let $I(\cdot)$ be the indicator function, which returns 1 if its argument is true, and 0 otherwise
Initialize $N \Leftarrow |L|$
Initialize $\mathbf{w}_i^{[j]} \Leftarrow \frac{1}{N}$, for $1 \leq j \leq N$
for $i = 1$ to c **do**
  Induce hypothesis $h(L, \mathbf{w}_i)$ by applying Ω to the examples in L, using weights $\mathbf{w}_i$
  Let y_j be the class label predicted by $h(L, \mathbf{w}_i)$ for example $\mathbf{x}_j \in L$, whose true label is z_j
  Compute the weighted error rate ϵ_i of $h(L, \mathbf{w}_i)$ as: $\epsilon_i \Leftarrow \sum_{j=1}^{N} \mathbf{w}_i^{[j]} \cdot I(y_j \neq z_j)$
  Set $\beta_i \Leftarrow \frac{\epsilon_i}{1-\epsilon_i}$
  Set $v_i \Leftarrow \log(\frac{1}{\beta_i})$
  for $j = 1$ to N **do**
    if $y_j = z_j$ **then**
      Set $\mathbf{w}_{i+1}^{[j]} \Leftarrow \mathbf{w}_i^{[j]} \cdot \beta_i$
    end if
  end for
  Normalize the weights such that $\sum_{j=1}^{N} \mathbf{w}_i^{[j]} = 1$
end for
return $h(L, \mathbf{w}_1), \ldots, h(L, \mathbf{w}_c)$

can be modified in a straightforward manner to handle instance weights, and so no approximation is required here. C4.5 maintains a weight for each example, as part of its strategy for handling missing attribute values. In order to grow a decision tree from a weighted training set, it is necessary to replace the existing weight vector, in which each weight is initialized to 1 by default, with the desired weight vector. Additionally, the instance weights must sum to N (i.e. they are not normalized to 1) in order for the "minimum subset size" stopping criterion (see Section 2.1) to function properly. Finally, as a technical note, C4.5's "CountItems" function has to be modified so that it knows when non-uniform instance weights are present. Normally, when growing a tree, if there are no missing values for the attribute under consideration, the function simply counts the number of examples without explicitly considering the weight vector. Hence, it will misreport the total weight whenever the true weights do not sum to the number of training examples.

Next, the five committee-based selective sampling methods analyzed in this study are described. They are: query-by-bagging, query-by-boosting, confidence-based query-by-boosting, bootstrap-LV, and bootstrap-M. In query-by-bagging [1], a committee of classifiers is formed by applying the bagging procedure described above to the training data L, as specified in Algorithm 2. Next, each committee member makes a prediction for each unlabelled example $\mathbf{x} \in U$, and labels are requested for the m examples for which the committee vote is most evenly split. The query-by-boosting procedure [1] is defined analogously, except that a boosting committee is built instead of a bagging committee; also, a weighted vote is computed for each $\mathbf{x} \in U$. Then, the m examples are selected for which the difference in the total weight assigned to the each class by the committee is minimal. Pseudo-code for query-by-bagging (QBag) and query-by-boosting (QBoost) is displayed in Algorithm 4 and Algorithm 5, respectively.

Algorithm 4 Query-by-bagging

Require: Learning algorithm Ω, initial training set L_1, unlabelled pool U, batch size m, maximum number of iterations n, committee size c

1: Initialize $U_1 \Leftarrow U$
2: Initialize $h(L_1^1), \ldots, h(L_1^c) \Leftarrow$ Bagging-committee(Ω, L_1, c)
3: **for** $i = 1$ to n **do**
4: **for all** $\mathbf{x} \in U_i$ **do**
5: Let $y^j \in \{-1, 1\}$ be the class predicted by committee member $h(L_i^j)$ for example $\mathbf{x}$
6: Set the score for $\mathbf{x}$ to be $\left|\sum_{j=1}^{c} y^j\right|$
7: **end for**
8: Label the examples $X_i^* \subseteq U_i$ which have the m lowest scores
9: Set $U_{i+1} \Leftarrow U_i - X_i^*$ and $L_{i+1} \Leftarrow L_i + X_i^*$
10: Set $h(L_{i+1}^1), \ldots, h(L_{i+1}^c) \Leftarrow$ Bagging-committee(Ω, L_{i+1}, c)
11: **end for**

Algorithm 5 Query-by-boosting

Require: Learning algorithm Ω, initial training set L_1, unlabelled pool U, batch size m, maximum number of iterations n, committee size c

1: Initialize $U_1 \Leftarrow U$
2: Initialize $h(L_1, \mathbf{w}_1), \ldots, h(L_1, \mathbf{w}_c) \Leftarrow$ Adaboost.M1-committee(Ω, L_1, c)
3: **for** $i = 1$ to n **do**
4: **for all** $\mathbf{x} \in U_i$ **do**
5: Let $y^j \in \{-1, 1\}$ be the class predicted by committee member $h(L_i, \mathbf{w}_j)$ for example $\mathbf{x}$
6: Let v^j be the vote weight for committee member $h(L_i, \mathbf{w}_j)$
7: Set the score for $\mathbf{x}$ to be $\left|\sum_{j=1}^{c} v^j \cdot y^j\right|$
8: **end for**
9: Label the examples $X_i^* \subseteq U_i$ which have the m lowest scores
10: Set $U_{i+1} \Leftarrow U_i - X_i^*$ and $L_{i+1} \Leftarrow L_i + X_i^*$
11: Set $h(L_{i+1}, \mathbf{w}_1), \ldots, h(L_{i+1}, \mathbf{w}_c) \Leftarrow$ Adaboost.M1-committee(Ω, L_{i+1}, c)
12: **end for**

Confidence-based query-by-boosting (QBoostC) is a novel adaptation of the algorithm Adaboost.MH [48] to the task of selective sampling. In Adaboost.MH, each committee member makes a "soft" prediction in the form of a real number, the sign of which specifies the predicted class, and whose magnitude indicates the degree of confidence in the prediction (i.e. a large magnitude implies high confidence). By contrast, in Adaboost.M1, which is used by QBoost, the individual committee members make binary-valued predictions. When using a decision tree as the base learner, the soft prediction $y_j \in \mathbb{R}$ is computed for example $\mathbf{x}_j$ as follows: Let W_+^j and W_-^j be the fractions of positive and negative training examples in the decision region that classifies $\mathbf{x}_j$. Then, $y_j = \frac{1}{2} \ln\left(\frac{W_+^j + \epsilon}{W_-^j + \epsilon}\right)$, where ϵ is a smoothing constant, introduced to prevent extremely large or small confidence values from occurring. In these experiments, the value $\epsilon = \frac{1}{N}$ was used. The procedure "Adaboost.MH-committee" is obtained by replacing line 12 in Algorithm 3 with the weight update rule $w_{i+1}^{[j]} = w_i^{[j]} \cdot \exp(-z_j \cdot y_j)$, and then deleting lines 7, 8, 9, 11, and 13.

Pseudo-code for the QBoostC selective sampler proposed here is shown in Algorithm 6. The procedure first builds a boosting committee according to the Adaboost.MH algorithm described

Algorithm 6 Confidence-based query-by-boosting

Require: Learning algorithm Ω, initial training set L_1, unlabelled pool U, batch size m, maximum number of iterations n, committee size c, smoothing constant ϵ

1: Initialize $U_1 \Leftarrow U$
2: Initialize $h(L_1, \mathbf{w}_1), \ldots, h(L_1, \mathbf{w}_c) \Leftarrow$ Adaboost.MH-committee(Ω, L_1, c)
3: **for** $i = 1$ to n **do**
4: **for all** $\mathbf{x} \in U_i$ **do**
5: Let W_+ and W_- be the fractions of positive and negative training examples in the decision region in which committee member $h(L_i, \mathbf{w}_j)$ classifies example $\mathbf{x}$
6: Let $y^j \Leftarrow \frac{1}{2} \ln \left(\frac{W_+ + \epsilon}{W_- + \epsilon} \right)$ be the soft prediction made by $h(L_i, \mathbf{w}_j)$ for $\mathbf{x}$
7: Set the score for $\mathbf{x}$ to be $\left| \sum_{j=1}^{c} y^j \right|$
8: **end for**
9: Label the examples $X_i^* \subseteq U_i$ which have the m lowest scores
10: Set $U_{i+1} \Leftarrow U_i - X_i^*$ and $L_{i+1} \Leftarrow L_i + X_i^*$
11: Set $h(L_{i+1}, \mathbf{w}_1), \ldots, h(L_{i+1}, \mathbf{w}_c) \Leftarrow$ Adaboost.MH-committee(Ω, L_{i+1}, c)
12: **end for**

above, and then employs essentially the same procedure that is used by QBoost to select examples for labelling. In particular, QBoostC chooses to label the m examples for which the sum of the committee members' votes is nearest to zero; however, the vote weight for a given committee member is no longer fixed, but instead may vary for each unlabelled example $\mathbf{x}$, according to its confidence in predicting the label of $\mathbf{x}$. Thus, QBoostC changes the manner in which the committee is built, as well as the voting mechanism, which in turn can alter the ranking of the unlabelled examples. Schapire and Singer [48] found that the use of real-valued confidence estimates in Adaboost.MH led to lower generalization error rates in experiments conducted on several benchmark datasets, compared to a version of the algorithm that permitted only binary predictions. The base learner they used was a *decision stump*, which is a decision tree containing exactly one decision node. In light of these findings, it would be interesting to determine whether a selective sampling procedure based on the same principles, namely, QBoostC, can make more efficient use of unlabelled data than QBoost, which uses the binary-valued prediction model. It may also be the case that the models produced by these two selective sampling methods differ in terms of their stability.

The bootstrap-LV (BootLV) selective sampling method [47], as its name suggests, creates bootstrap replicates of the training data from which it induces committee members; that is, it builds a bagging committee. A fundamental difference between BootLV and the selective sampling methods described thus far is that the former is designed to improve class probability estimates (CPE), whereas the others aim to improve classification performance. Accordingly, for a given example $\mathbf{x}$, BootLV requires that the base learner be able to estimate the probability of $\mathbf{x}$'s membership in each class. Given a C4.5 decision tree, the CPE for class y is calculated as the fraction of training examples with label y among the examples in the decision region that classifies $\mathbf{x}$. Next, BootLV computes the estimated "local variance" (LV) – the variance in the CPEs produced by the committee members – for each unlabelled example $\mathbf{x} \in U$. A high LV for $\mathbf{x}$ indicates that the example is not

Algorithm 7 Bootstrap-LV

Require: Learning algorithm Ω, initial training set L_1, unlabelled pool U, batch size m, maximum number of iterations n, committee size c

1: Initialize $U_1 \Leftarrow U$
2: Initialize $h(L_1^1), \ldots, h(L_1^c) \Leftarrow$ Bagging-committee(Ω, L_1, c)
3: **for** $i = 1$ to n **do**
4: **for all** $\mathbf{x} \in U_i$ **do**
5: Let p^j be the probability estimated by $h(L_i^j)$ that $\mathbf{x}$ belongs to the positive (minority) class
6: Let $\bar{p}$ be the average of these estimates, formally: $\bar{p} = \frac{1}{c}\sum_{j=1}^{c} p^j$
7: Set $D_i(\mathbf{x}) \Leftarrow \frac{1}{\bar{p}}\sum_{j=1}^{c}(p^j - \bar{p})^2$
8: **end for**
9: Normalize the distribution D_i such that $\sum_{j=1}^{|L_i|} D_i(\mathbf{x}_j) = 1$
10: Draw and label m examples X_i^* from U_i, without replacement, according to D_i
11: Set $U_{i+1} \Leftarrow U_i - X_i^*$ and $L_{i+1} \Leftarrow L_i + X_i^*$
12: Set $h(L_{i+1}^1), \ldots, h(L_{i+1}^c) \Leftarrow$ Bagging-committee(Ω, L_{i+1}, c)
13: **end for**

well understood, and that it may be beneficial to obtain its label. Rather than simply requesting the label for the m instances that exhibit the greatest LV, however, BootLV defines a probability distribution D over the examples in the unlabelled pool U, in which the weight of each example is proportional to its estimated LV. BootLV then selects the examples to be queried by sampling m times, without replacement, according to D. Pseudo-code for this method is displayed in Algorithm 7.

Obtaining labels for examples that are expected to improve CPEs can also benefit classification performance. However, as Saar-Tsechansky and Provost point out, the pursuit of accurate CPEs can be detrimental to classification performance in some situations [47, p. 19]. It has been argued that a scoring function based on margins may be more appropriate for classification tasks than a function that considers the variance in class estimates [34, 33]. In order to test this hypothesis, a version of BootLV that uses margins in place of local variance was formulated; this method is referred to as bootstrap-M (BootM). In a two-class problem, the margin is computed as the difference between the average weight assigned to the positive (minority) class by the committee members and that of the negative (majority) class. In fact, this is exactly the scoring function that is computed by QBag, QBoost, and QBoostC, albeit using different voting schemes in each case.[2] As in BootLV, the prediction made by each committee member in BootM for an example $\mathbf{x} \in U$ takes the form of a probability, and is the estimated probability that $\mathbf{x}$ belongs to the positive class. Given the c predictions for $\mathbf{x}$, the weight assigned to $\mathbf{x}$ in the distribution D is inversely proportional to the margin, so that examples yielding smaller margins have a higher probability of being selected. Pseudo-code for BootM is shown in Algorithm 8. Note that on line 7, a smoothing constant ϵ was used to prevent division by zero; its value was set to 0.001 for these experiments.

Although a decision tree is generally considered to be an intelligible model, a committee of de-

[2] These methods actually compute the difference between the *total* weight assigned to each class, which, for the purpose of ranking unlabelled examples, is the same as computing the difference in the average weights.

Algorithm 8 Bootstrap-M

Require: Learning algorithm Ω, initial training set L_1, unlabelled pool U, batch size m, maximum number of iterations n, committee size c, smoothing constant ϵ

Initialize $U_1 \Leftarrow U$
Initialize $h(L_1^1), \ldots, h(L_1^c) \Leftarrow$ Bagging-committee(Ω, L_1, c)
for $i = 1$ to n **do**
  for all $\mathbf{x} \in U_i$ **do**
    Let p^j be the probability estimated by $h(L_i^j)$ that $\mathbf{x}$ belongs to the positive (minority) class
    Let q be the margin for $\mathbf{x}$, where $q = \left| \left(\frac{1}{c} \sum_{j=1}^{c} p^j \right) - 0.5 \right|$
    Set $D_i(\mathbf{x}) \Leftarrow \frac{1+\epsilon}{q+\epsilon}$
  end for
  Normalize the distribution D_i such that $\sum_{j=1}^{|L_i|} D_i(\mathbf{x}_j) = 1$
  Draw and label m examples X_i^* from U_i, without replacement, according to D_i
  Set $U_{i+1} \Leftarrow U_i - X_i^*$ and $L_{i+1} \Leftarrow L_i + X_i^*$
  Set $h(L_{i+1}^1), \ldots, h(L_{i+1}^c) \Leftarrow$ Bagging-committee(Ω, L_{i+1}, c)
end for

cision trees tends to be incomprehensible [5, 17]. Since intelligibility is of concern in this thesis – it is one of the main motivations for using a decision tree learner, and for being interested in stability – in the experiments conducted here, the committee-based selective sampling methods choose the examples that are to be labelled. However, these examples are then used to induce a single decision tree classifier, which is the model that is ultimately evaluated on the various performance metrics. Thus, the error rates and stability scores reported for a given committee-based method apply to the decision tree that is induced from the data L_i at each iteration i of active learning. In contrast, previous studies have trained a committee of trees from examples that were also chosen by a committee of trees [1]. This topic is discussed further in Section 3.4.1.

3.3 Experimental Procedure

Regarding the data used for these experiments, several of the UCI datasets used in the experiments conducted in Section 2.5 contain a relatively small number examples. This can be problematic when evaluating selective sampling methods, as after appropriate fractions of the data have been assigned to the evaluation set and the initial training set, few examples remain to form the unlabelled pool. In the active learning experiments performed in this section, only the datasets listed in Tables 2.1, 2.2, and 2.3, which contain a minimum of 500 examples are used. Of the 25 datasets listed in these tables, 15 meet this condition.

The active learning experiments involved the use of the six selective sampling methods described in Section 3.2, as well as random sampling (Random), which served as a basis for comparison. For the committee-based methods, the number of committee members was set to 10. The C4.5 algorithm was used as the base learner, and all experiments were duplicated using the entropy splitting criterion and then the DKM criterion. For each UCI dataset, one third of the data was randomly set aside as the evaluation set, and was used to measure both accuracy and structural stability (i.e. these

examples served as weights for the region stability metric). Stratification was employed to ensure that the proportion of positive examples (i.e. examples belonging to the minority class) appearing in the evaluation set matched that of the training set. Of the remaining instances, one-fifteenth (or 10% of the entire dataset) were randomly selected to form the initial training set L_1, while those not selected formed the unlabelled example pool. The examples in the initial training set and the unlabelled pool were not stratified, since their class labels are technically unknown when the random selection is performed. The batch size m was set to be 2% of the number of examples in the entire dataset, to a minimum of 10 and a maximum of 50 examples. Since the training sets formed by the various selective sampling methods grow increasingly similar as the number of unlabelled examples remaining in the pool becomes low, active learning was halted once two-thirds of the pool examples were labelled. For a given dataset, 10 runs were performed, each using a different initial training set; specifically, the random number generator was seeded with a different value when choosing L_1 for each run. The same evaluation set was used for all 10 runs, in order to remove a source of variation between the accuracy and stability scores obtained during each run. Finally, the region stability scores were computed using values for ε_{pct} of 0, 5, 10, and 15 percent.

In order to assess whether one selective sampling method is superior to another on a given dataset, a summary statistic was devised to convert a learning curve into a single value. It was argued in Section 3.1 that structural stability is most desirable in the later stages of active learning; ideally, the learner will have a reasonable grasp of the target concept at this point, and new examples will serve mainly to refine the model, rather than to change it dramatically. A high degree of stability during earlier iterations of active learning is of little value if stability deteriorates in the later rounds, since it is the model produced at the last iteration of active learning that is ultimately used for classification tasks and/or knowledge extraction. Based on this reasoning, a weighted average of the scores was computed in order to compare the performance of the selective sampling methods, as well as that of the two splitting criteria.

Given 10 runs of active learning on a particular dataset, the median region stability score was calculated at each iteration. The reason for using the median rather than the mean was that structural stability often varied significantly between runs – trials exhibiting very low or very high scores at certain iterations would exert undue influence on the average score, whereas the median was robust to these anomalies. Once the learning curve consisting of the median score s_i at each iteration i had been obtained, the weighted average was computed as $\frac{1}{n}\sum_{i=1}^{n} w_i \cdot s_i$, where n was the total number of selective sampling iterations. The weights w_i were increased linearly as a function of the iteration number i, and were calculated using the equation $w_i = \frac{2i}{n(n+1)}$. By this formula, the weights summed to 1, and the weight for the i^{th} iteration was i times larger than the weight for iteration 1. Semantic stability was measured by replacing the region stability metric used by PrevStab and FinalStab with the expected agreement measure (see Section 2.4), and weighted averages were computed from the resulting sequences of scores. Note that for the RunStab metric, only structural

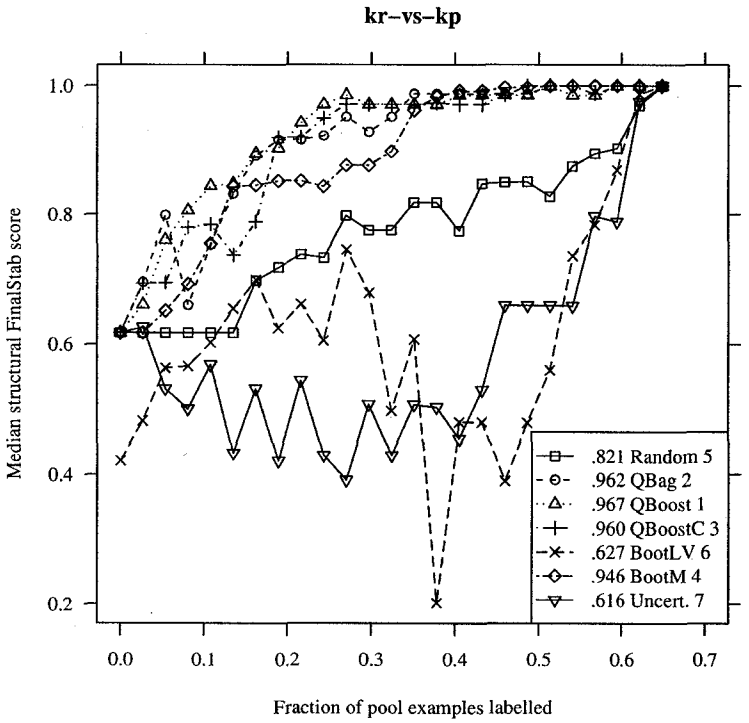

Figure 3.4: Median structural FinalStab scores on the kr-vs-kp dataset (DKM criterion, $\varepsilon_{pct} = 0$).

stability was measured. The learning curve for classification accuracy was summarized using the same approach as for stability, since it can also be argued that higher accuracy is most desirable in the later stages of active learning. However, since the accuracy scores recorded during different runs exhibited less variance than did the stability scores, the learning curve consisted of the mean accuracy score at each iteration.

As an example of how the weighted averaging and ranking schemes characterize learning curves produced by the various selective sampling procedures, consider the example in Figure 3.4. In this graph, the structural FinalStab scores recorded for the 7 sampling methods on the kr-vs-kp dataset are plotted as a function of the training set size. To the left of each sampling method's name in the graph legend, the weighted average of its scores is displayed; to the right, the method's rank is shown. QBoost, QBag, and QBoostC were the highest-ranked methods in this case, as all three were highly and consistently stable beyond the point where one-fifth of the pool examples had been labelled. Although the performances of these methods were virtually indistinguishable after 40% of the pool examples had been labelled, QBag and QBoostC were slightly less stable than QBoost during the early rounds of learning, which explains their marginally lower scores. BootM trailed behind this group just slightly. In the latter half of the learning iterations, BootM achieved levels of stability that matched those of the top-ranked methods; however, a lapse in stability between

iterations 7 and 13 (.16 to .32 on the x-axis) was substantial enough to place it in fourth position. The rank of 5 assigned to random sampling requires little explanation, as the majority of its scores were almost directly in between those of its two nearest competitors, particularly in the last third of the curve. Finally, BootLV and Uncertainty were clearly the most unstable methods. Despite taking a distinct plummet at iteration 15 (.38 on the x-axis), BootLV eventually recovered and kept pace with Uncertainty during the final iterations of active learning. Apparently, when the weighted averages were computed, the increased stability of BootLV over Uncertainty in the early stages of learning was enough to offset its brief period of instability during the middle rounds.

The following methodology, which is described by Demsar [13], was adopted for the purpose of determining whether differences in performance between the various methods that were observed on this collection of datasets were statistically significant. The null hypothesis is that all the methods are equivalent. For each dataset, ranks were assigned to the sampling methods;[3] next, the non-parametric Friedman test [24] was applied to the average rank that was calculated for each method, and the critical value was computed using the F_F statistic proposed by Iman and Davenport [26]. If the null hypothesis could be rejected based on this value (at a chosen significance level α), the Nemenyi test [38] was subsequently used to determine whether significant differences existed between given pairs of sampling methods. Specifically, the Nemenyi test calculates the critical difference (CD) – the minimum amount by which the average ranks of two methods must differ in order for the discrepancy to be considered statistically significant.

As a final methodological note, the stability curves for all seven sampling procedures were displayed in Figure 3.4. Unfortunately, it is often the case that several curves frequently overlap and/or cross one another, making it impractical to display all the active learning data. In order to improve the intelligibility of such plots, only a subset of the methods are typically displayed. Also, the weighted averages and ranks are not included in the graph legends due to space limitations.

3.4 Experimental Results

3.4.1 Predictive Accuracy

In this section, an overall assessment of the classification accuracy achieved by the various sampling methods is presented. Table 3.1 displays the weighted error rates for trees grown using the entropy criterion, while Table 3.2 shows the weighted error rates for DKM. The last row in each table reveals the average rank that was computed for each sampling method.

Regardless of the splitting criterion that was used, QBag achieved the highest average rank, and Random achieved the lowest average rank. Uncertainty also exhibited consistently poor performance, as it achieved the second-lowest average rank in both cases. In terms of statistical significance, at $\alpha = .05$, the null hypothesis was rejected for both splitting criteria, based on the Friedman

[3]The standard ranking procedure was followed. When assigning ranks, the method with the best performance received a rank of 1, a rank of 2 was assigned to the next best method, and so on. In the event of a tie, average ranks were assigned.

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.140 (6)	.120 (1)	.131 (4)	.137 (5)	.125 (2)	.127 (3)	.149 (7)
australian	.126 (1)	.129 (4)	.132 (7)	.127 (2)	.130 (5.5)	.130 (5.5)	.128 (3)
car	.095 (7)	.076 (1)	.083 (5.5)	.083 (5.5)	.078 (2)	.080 (3)	.081 (4)
german	.291 (7)	.271 (1)	.284 (4)	.287 (5.5)	.287 (5.5)	.279 (2)	.281 (3)
kr-vs-kp	.013 (7)	.007 (1.5)	.008 (3.5)	.008 (3.5)	.007 (1.5)	.009 (5)	.010 (6)
letter	.016 (7)	.011 (2.5)	.011 (2.5)	.011 (2.5)	.011 (2.5)	.013 (5.5)	.013 (5.5)
nursery	.057 (7)	.038 (2.5)	.039 (4)	.037 (1)	.038 (2.5)	.040 (5)	.045 (6)
pendigits	.015 (7)	.009 (1)	.010 (2)	.011 (3.5)	.012 (5.5)	.012 (5.5)	.011 (3.5)
pima-indians	.282 (4)	.284 (7)	.278 (1)	.283 (5.5)	.281 (3)	.279 (2)	.283 (5.5)
segmentation	.022 (7)	.011 (1)	.012 (2.5)	.013 (4)	.012 (2.5)	.019 (5)	.020 (6)
tic-tac-toe	.210 (5)	.195 (1.5)	.204 (3)	.195 (1.5)	.205 (4)	.212 (6)	.217 (7)
vehicle	.228 (3.5)	.228 (3.5)	.227 (2)	.230 (6)	.232 (7)	.225 (1)	.229 (5)
vowel	.056 (7)	.033 (1)	.034 (2)	.035 (3)	.037 (4)	.046 (5)	.049 (6)
wdbc	.078 (6)	.067 (2)	.066 (1)	.068 (3.5)	.068 (3.5)	.070 (5)	.079 (7)
yeast	.257 (7)	.251 (1)	.253 (3)	.254 (4.5)	.256 (6)	.252 (2)	.254 (4.5)
Avg. rank	(5.900)	(2.100)	(3.133)	(3.767)	(3.800)	(4.033)	(5.267)

Table 3.1: Weighted error rates (entropy criterion).

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.137 (6)	.122 (1)	.127 (3)	.132 (5)	.129 (4)	.126 (2)	.141 (7)
australian	.125 (1.5)	.128 (4)	.134 (6)	.129 (5)	.125 (1.5)	.127 (3)	.142 (7)
car	.096 (7)	.077 (1)	.083 (5)	.084 (6)	.078 (2)	.079 (3)	.081 (4)
german	.297 (7)	.287 (2)	.292 (5)	.290 (3)	.286 (1)	.291 (4)	.295 (6)
kr-vs-kp	.014 (7)	.007 (1)	.008 (3)	.008 (3)	.008 (3)	.009 (5)	.010 (6)
letter	.016 (7)	.012 (3.5)	.012 (3.5)	.011 (1)	.012 (3.5)	.013 (6)	.012 (3.5)
nursery	.062 (7)	.040 (3.5)	.040 (3.5)	.039 (1.5)	.039 (1.5)	.044 (5)	.045 (6)
pendigits	.015 (7)	.012 (3)	.012 (3)	.012 (3)	.012 (3)	.013 (6)	.012 (3)
pima-indians	.284 (7)	.280 (3.5)	.276 (2)	.282 (5)	.280 (3.5)	.283 (6)	.274 (1)
segmentation	.024 (7)	.017 (3.5)	.016 (1.5)	.017 (3.5)	.016 (1.5)	.019 (5)	.021 (6)
tic-tac-toe	.215 (6)	.203 (1)	.204 (2)	.211 (4)	.212 (5)	.207 (3)	.222 (7)
vehicle	.231 (5)	.229 (3)	.232 (6)	.227 (2)	.234 (7)	.230 (4)	.225 (1)
vowel	.053 (7)	.037 (1.5)	.037 (1.5)	.038 (3)	.039 (4)	.043 (5)	.046 (6)
wdbc	.074 (7)	.054 (1)	.060 (3)	.055 (2)	.064 (4.5)	.064 (4.5)	.069 (6)
yeast	.260 (7)	.252 (1.5)	.252 (1.5)	.255 (3)	.259 (6)	.258 (5)	.256 (4)
Avg. rank	(6.367)	(2.267)	(3.300)	(3.333)	(3.400)	(4.433)	(4.900)

Table 3.2: Weighted error rates (DKM criterion).

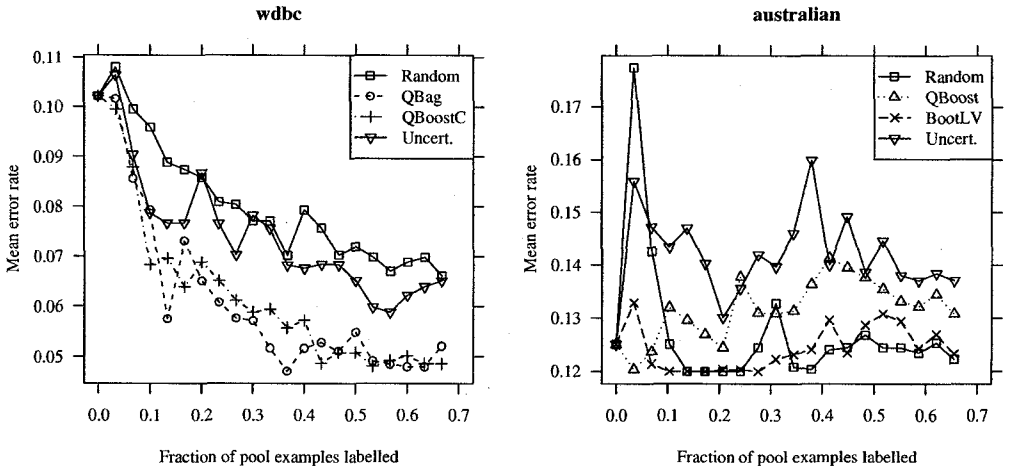

Figure 3.5: Mean error rates on the wdbc dataset (left) and australian dataset (right) when using the DKM criterion. Note that the scale of the y-axis differs between the two plots.

test. The critical difference (CD) computed by the Nemenyi test for 7 methods, 15 datasets, and $\alpha = .05$ is 2.329.[4] Thus, it can be asserted that QBag is significantly more accurate than both Random and Uncertainty for either splitting criterion. A representative example is the wdbc dataset, where for both splitting criteria, Random and Uncertainty claimed the two lowest ranks. The data obtained when using the DKM splitting criterion is shown in Figure 3.5 (left). On this dataset, Random produced the highest error rates on most iterations, with Uncertainty improving slightly upon these until the final round, at which point the error rates were almost identical. QBag and QBoostC also finished with virtually the same error rates, although the former was slightly more accurate between iterations 7 and 13 (.20 to .40 on the x-axis), which was sufficient to earn it the top position. Overall, there were few distinctions to be made between the remaining methods, as the differences in their error rates were often minute, and were never statistically significant.

On the australian and pima-indians datasets, the correlation between larger training set sizes and lower error rates was rather weak, as the predictive performance of many methods fluctuated significantly throughout the learning curve, without exhibiting any clear downward trend. It was on these datasets that random sampling was competitive, and occasionally more accurate than some of the selective sampling techniques. On the australian dataset, for instance, Random claimed or tied for the highest rank regardless of the splitting criterion that was used; the results for DKM are displayed in Figure 3.5 (right). In this particular case, Random and BootLV shared the top rank, while QBoost and Uncertainty ranked 6th and 7th, respectively. Although Random had a

[4]It should be emphasized that the CD does not depend on the actual ranks that are recorded, but depends only on the number of methods and datasets, as well as α. Thus, the same CD is implicitly assumed when analyzing the results for other statistics, whenever these parameters are the same. Of course, the CD is irrelevant unless the null hypothesis is first rejected based on the Friedman test, which does consider the ranks.

poor start, its error rate decreased rapidly during the 3rd and 4th iterations (.07 and .10 on the x-axis), and then remained roughly between 0.12 and 0.13 up to and including the final round of active learning. Interestingly, of the seven methods, only Random and BootLV were able to improve upon the classifier that was induced from the initial training sample. Random was also somewhat competitive on the tic-tac-toe dataset, where it achieved mediocre ranks of 5 and 6 when using entropy and DKM, respectively, but outperformed Uncertainty in both cases.

In Section 3.2, variants of the bootstrap-LV (BootLV) and query-by-boosting (QBoost) methods were proposed, called bootstrap-M (BootM) and confidence-based query-by-boosting (QBoostC). Based on the weighted average error rates that were computed from the experimental data, BootM did not yield improved accuracy over BootLV. When using the entropy splitting criterion, BootM was more accurate on 4 of the 15 datasets, but was less accurate in 8 cases (there were 3 draws). For the experiments involving DKM, the win-loss-draw count for BootM was 4-10-1. Although the number of wins in each case favoured BootLV, it is worth noting that the weighted average error rates of the two methods differed by more than .005 on only 5 of the 30 trials. For both splitting criteria, the differences between the average ranks of these two methods were statistically insignificant.

Hence, these results do not support the conjecture made by Melville and Mooney [33], namely, that selecting examples according to the minimum margin is a more effective method for reducing the error rate than using a criterion that measures the variance in class probability estimates (CPEs). Recall that the sole difference between the two sampling methods is that BootM chooses examples for which the difference between the total weight assigned to each class by the committee (i.e. the margin) is minimal, whereas BootLV selects examples for which the predictions made by the committee members exhibit the highest variance. Although a different selective sampling method, called ActiveDecorate, was used in the cited study, and the batch size ranged from just one to three examples per iteration, it seemed plausible that the use of margins could improve classification performance of any sampling method that focused on CPE variance, as was discussed here in Section 3.2. However, the results presented here indicate that this is not the case, at least concerning the BootLV method.

Similarly, it was determined that QBoostC was unable to reduce the error rates of QBoost. The win-loss-draw count for QBoostC versus QBoost, based on the weighted average error rates, was 3-9-3 when using the entropy criterion, and 6-7-2 with DKM. The average ranks favoured QBoost by margins of .634 and .033, respectively; however, these differences were statistically insignificant. Thus, allowing the members of the boosting committee to make real-valued confidence estimates, instead of binary predictions, did not cause the committee to select examples that were more informative for the purpose of training a C4.5 decision tree. The effect on stability of the modifications made to BootLV and QBoost is analyzed in the next two sections.

To summarize the classification accuracy results, the relative performances of the 7 selective sampling procedures observed in these experiments were on par with findings made in previous

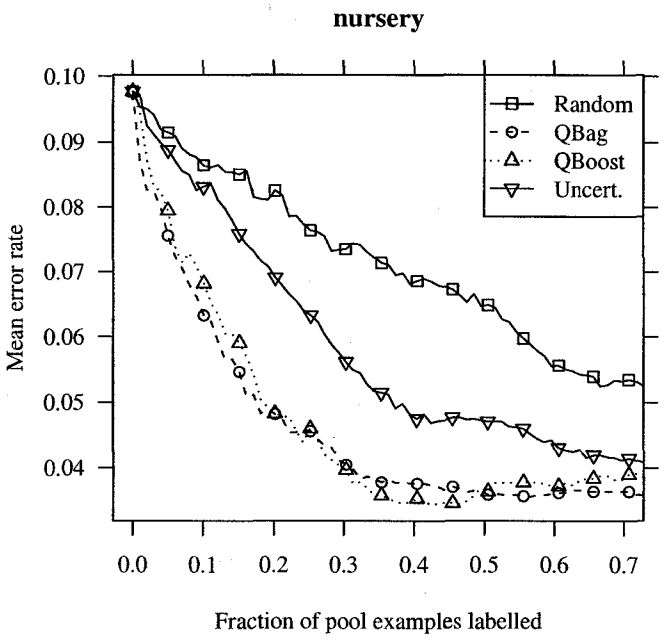

Figure 3.6: Mean error rates forming a "banana" shape, indicating efficient use unlabelled data by the selective sampling methods (entropy criterion).

active learning studies that involved these particular methods. It is not surprising that random sampling tended to produce the highest error rates among the 7 sampling methods that were tested. The goal of active learning, after all, is to build an accurate classifier using fewer training examples than are required when learning passively, and the inferiority of random selection has been demonstrated in numerous selective sampling studies (e.g. [1, 30]). Furthermore, the results obtained here confirm that committee-based methods usually produce lower error rates than uncertainty sampling (e.g. [16, 47]). On many of the datasets used in these experiments, a distinct "banana" shape was visible when the error rates of the selective sampling methods were plotted alongside those of random sampling. This is a hallmark of efficient selective sampling, and a prime example occurred on the nursery dataset (see Figure 3.6). In this case, Uncertainty decreased the mean error rate relative to Random; yet, the committee-based sampling methods, QBag and QBoost, achieved even larger reductions.

A key factor distinguishing the experiments performed in this study from most existing research involving active learning with decision trees is that here, the committee-based sampling methods select examples that are used to train a single decision tree classifier. That is, the committee-based results reported in these experiments represent the performance of a single C4.5 decision tree that is trained on examples that are selected by a committee of C4.5 decision trees. By contrast, in the original experiments involving QBag and QBoost, for instance, a committee of C4.5 trees was

employed to select unlabelled examples that were then used to train another bagged or boosted committee of C4.5 trees, respectively [1]. Although a single decision tree trained from a particular data sample L is likely to be less accurate than a bagged or boosted committee trained from L [5, 43], the committee of trees is no longer intelligible [5, 17]. Therefore, for learning problems in which comprehensibility is a goal, it is desirable to identify the selective sampling procedures(s) whose choice of examples elicits the best performance when used to train a single decision tree.

Of course, it is possible that superior performance could be achieved by having a single decision tree select the examples that it expects will be the most informative for learning; this is precisely what takes place when using the Uncertainty sampling procedure. However, the experimental results obtained here revealed that Uncertainty tended to grow trees that were less accurate than those constructed by the committee-based sampling methods; in particular, QBag achieved significantly lower error rates than Uncertainty when trees were grown using the entropy criterion. Based on these results, the QBag algorithm is recommended for choosing training examples that are used to induce a single decision tree classifier. I am aware of only two other active learning studies in which the type of classifier that was trained and evaluated differed from the type of classifier that selected the examples. Lewis and Catlett [30] employed a probabilistic classifier to select examples that were then used by a modified version of C4.5 to induce a decision tree. Xiao et. al [56] used a committee of rule sets to choose examples for a single C4.5Rules [42] production rule learner.

3.4.2 Semantic Stability

With regard to semantic stability, QBag achieved the highest average rank on the FinalStab metric, independent of the splitting criterion that was used. The weighted averages for the entropy and DKM criteria are displayed in Table 3.3 and Table 3.4, respectively. On the vowel dataset, for which the entropy results are illustrated in Figure 3.7 (left), BootM and Random occupied the two lowest positions. QBag recorded the highest weighted average on this dataset as a result of its superior stability during the middle rounds of active learning, which distinguished it from its closest competitors, namely, QBoost and QBoostC. Overall, the average rank for Random was considerably lower than the average ranks of the other selective sampling methods; in fact, it achieved the lowest rank on 19 of the 30 trials. Nevertheless, Random did not perform poorly across the board, as it was more stable than QBag, for example, on the australian, tic-tac-toe, and vehicle datasets. The results on the tic-tac-toe dataset are plotted in Figure 3.7 (right). Here, the levels of semantic stability achieved by Random and Uncertainty were superior to the those of the committee-based methods.

In terms of statistical significance, at $\alpha = .05$, QBag is significantly more stable than Random for both splitting criteria, and is also significantly better than BootM when using entropy. The differences between the remaining methods are insignificant.

The median semantic PrevStab scores were more highly concentrated than the median FinalStab scores (see Table 3.5 and Table 3.6). As a result, the differences between the various methods were

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.926 (7)	.954 (1)	.935 (4)	.929 (5)	.928 (6)	.942 (2)	.941 (3)
australian	.969 (4)	.974 (2)	.959 (7)	.966 (5)	.970 (3)	.962 (6)	.989 (1)
car	.963 (7)	.994 (1)	.989 (4)	.990 (3)	.992 (2)	.983 (6)	.987 (5)
german	.830 (7)	.865 (2)	.854 (4)	.844 (5)	.838 (6)	.860 (3)	.920 (1)
kr-vs-kp	.995 (7)	.999 (1.5)	.998 (4)	.998 (4)	.999 (1.5)	.998 (4)	.997 (6)
letter	.987 (7)	.997 (2.5)	.998 (1)	.996 (4.5)	.996 (4.5)	.991 (6)	.997 (2.5)
nursery	.970 (7)	.992 (1)	.987 (3)	.986 (4)	.990 (2)	.985 (5.5)	.985 (5.5)
pendigits	.986 (7)	.994 (1)	.993 (2.5)	.992 (4.5)	.992 (4.5)	.989 (6)	.993 (2.5)
pima-indians	.861 (5)	.874 (2)	.877 (1)	.854 (6)	.846 (7)	.867 (3)	.862 (4)
segmentation	.984 (7)	.997 (1)	.995 (3)	.992 (5)	.996 (2)	.993 (4)	.988 (6)
tic-tac-toe	.889 (2)	.840 (6)	.834 (7)	.849 (4)	.863 (3)	.845 (5)	.890 (1)
vehicle	.847 (2)	.829 (6.5)	.829 (6.5)	.841 (4)	.836 (5)	.842 (3)	.850 (1)
vowel	.965 (7)	.987 (1)	.984 (3)	.986 (2)	.982 (4)	.973 (6)	.980 (5)
wdbc	.953 (7)	.974 (1)	.964 (3)	.970 (2)	.963 (4)	.960 (5.5)	.960 (5.5)
yeast	.906 (4)	.920 (2)	.904 (6)	.905 (5)	.898 (7)	.908 (3)	.925 (1)
Avg. rank	(5.800)	(2.100)	(3.933)	(4.200)	(4.100)	(4.533)	(3.333)

Table 3.3: Weighted semantic FinalStab scores (entropy criterion).

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.932 (7)	.954 (1)	.943 (4)	.935 (6)	.936 (5)	.950 (2)	.944 (3)
australian	.981 (1)	.952 (5)	.948 (6.5)	.948 (6.5)	.974 (3)	.958 (4)	.978 (2)
car	.968 (7)	.994 (1)	.992 (2.5)	.989 (4)	.992 (2.5)	.987 (5)	.985 (6)
german	.857 (7)	.889 (2)	.867 (6)	.870 (5)	.881 (3)	.877 (4)	.936 (1)
kr-vs-kp	.994 (7)	1.00 (2)*	1.00 (2)*	1.00 (2)*	.999 (4.5)	.999 (4.5)	.997 (6)
letter	.988 (7)	.999 (1)	.998 (2.5)	.997 (4)	.998 (2.5)	.992 (6)	.995 (5)
nursery	.966 (7)	.991 (1)	.990 (2)	.985 (4)	.989 (3)	.983 (6)	.984 (5)
pendigits	.986 (7)	.995 (1.5)	.991 (4)	.992 (3)	.995 (1.5)	.987 (6)	.990 (5)
pima-indians	.859 (2)	.856 (3.5)	.852 (6)	.842 (7)	.856 (3.5)	.855 (5)	.874 (1)
segmentation	.985 (7)	.993 (1)	.989 (4.5)	.991 (3)	.989 (4.5)	.992 (2)	.988 (6)
tic-tac-toe	.891 (1)	.836 (7)	.843 (4)	.837 (5.5)	.856 (3)	.837 (5.5)	.874 (2)
vehicle	.854 (2)	.832 (3)	.818 (6)	.811 (7)	.831 (4)	.828 (5)	.858 (1)
vowel	.961 (7)	.977 (2)	.975 (3.5)	.972 (5.5)	.986 (1)	.972 (5.5)	.975 (3.5)
wdbc	.960 (6)	.971 (4)	.973 (3)	.974 (1.5)	.974 (1.5)	.969 (5)	.957 (7)
yeast	.906 (6)	.920 (2)	.909 (5)	.915 (3)	.899 (7)	.912 (4)	.929 (1)
Avg. rank	(5.400)	(2.467)	(4.100)	(4.467)	(3.300)	(4.633)	(3.633)

Table 3.4: Weighted semantic FinalStab scores (DKM criterion). Entries preceded by an asterisk (*) are not perfect stability scores, but are the result of rounding the true weighted average scores to three decimal places.

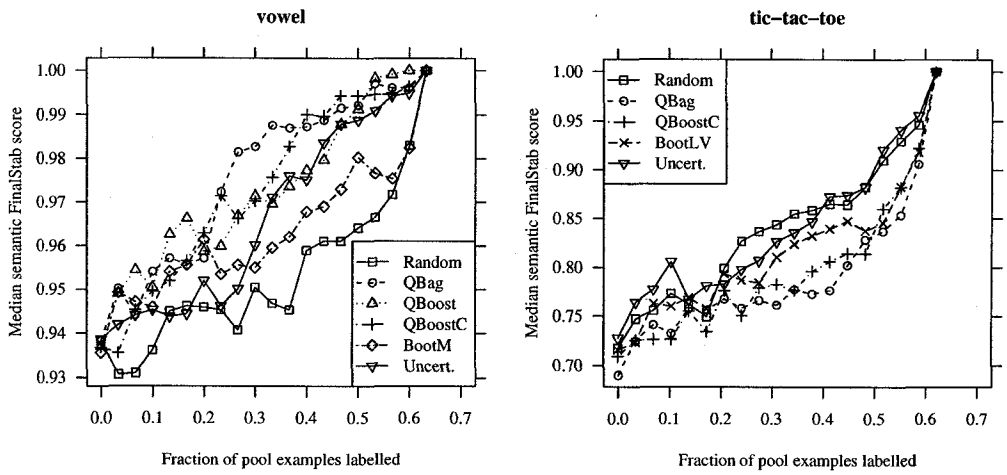

Figure 3.7: Median semantic FinalStab scores on the vowel dataset (left) and tic-tac-toe dataset (right) when using the entropy criterion. Note that the scale of the y-axis differs between the two plots.

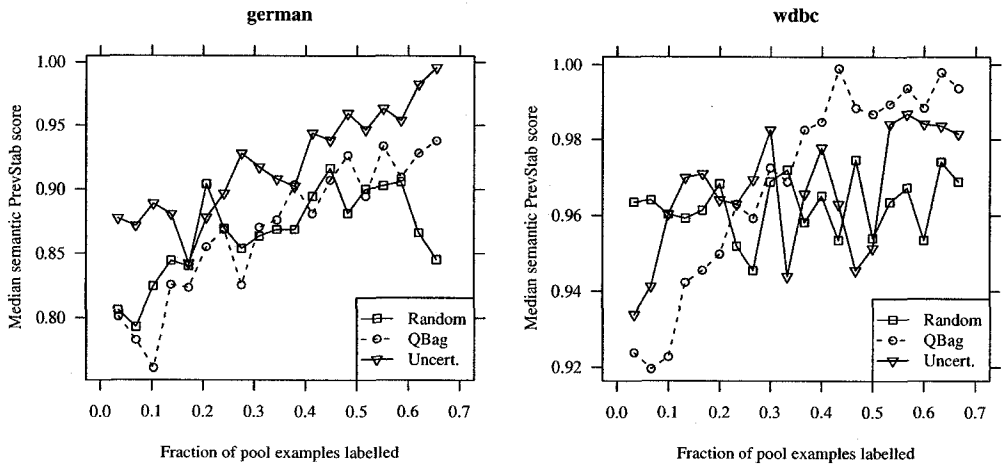

Figure 3.8: Median semantic PrevStab scores on the german dataset (left) and wdbc dataset (right) when using the entropy criterion. Note that the scale of the y-axis differs between the two plots.

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.943 (6)	.952 (2)	.946 (5)	.940 (7)	.948 (4)	.950 (3)	.955 (1)
australian	.968 (3)	.972 (2)	.959 (7)	.964 (4.5)	.962 (6)	.964 (4.5)	.985 (1)
car	.977 (7)	.987 (2)	.985 (4.5)	.985 (4.5)	.988 (1)	.981 (6)	.986 (3)
german	.873 (4.5)	.889 (2)	.867 (7)	.868 (6)	.873 (4.5)	.877 (3)	.934 (1)
kr-vs-kp	.995 (5)	.996 (2)	.995 (5)	.996 (2)	.996 (2)	.995 (5)	.994 (7)
letter	.997 (6.5)	.999 (3)	.999 (3)	.999 (3)	.999 (3)	.997 (6.5)	.999 (3)
nursery	.995 (7)	.998 (1.5)	.997 (4)	.997 (4)	.997 (4)	.996 (6)	.998 (1.5)
pendigits	.994 (7)	.997 (3)	.997 (3)	.997 (3)	.997 (3)	.995 (6)	.997 (3)
pima-indians	.874 (3)	.871 (5)	.884 (2)	.865 (6)	.861 (7)	.873 (4)	.887 (1)
segmentation	.988 (6)	.992 (1.5)	.990 (3)	.989 (4.5)	.992 (1.5)	.989 (4.5)	.987 (7)
tic-tac-toe	.913 (2)	.869 (5)	.851 (7)	.858 (6)	.899 (3)	.877 (4)	.924 (1)
vehicle	.863 (1)	.843 (4.5)	.840 (7)	.843 (4.5)	.842 (6)	.859 (3)	.862 (2)
vowel	.974 (6)	.983 (1)	.980 (3.5)	.982 (2)	.977 (5)	.971 (7)	.980 (3.5)
wdbc	.959 (6.5)	.976 (1)	.969 (4)	.972 (2)	.970 (3)	.959 (6.5)	.966 (5)
yeast	.911 (3)	.917 (2)	.902 (7)	.907 (5)	.906 (6)	.910 (4)	.923 (1)
Avg. rank	(4.900)	(2.500)	(4.800)	(4.267)	(3.933)	(4.867)	(2.733)

Table 3.5: Weighted semantic PrevStab scores (entropy criterion).

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.957 (6)	.963 (1.5)	.958 (4.5)	.951 (7)	.958 (4.5)	.960 (3)	.963 (1.5)
australian	.976 (1.5)	.965 (6)	.961 (7)	.969 (5)	.974 (3)	.976 (1.5)	.970 (4)
car	.978 (7)	.988 (1.5)	.986 (3)	.985 (4)	.988 (1.5)	.983 (5.5)	.983 (5.5)
german	.890 (6)	.906 (3)	.886 (7)	.902 (4)	.901 (5)	.908 (2)	.951 (1)
kr-vs-kp	.994 (7)	.996 (3)	.996 (3)	.996 (3)	.996 (3)	.996 (3)	.995 (6)
letter	.997 (7)	.999 (3)	.999 (3)	.999 (3)	.999 (3)	.998 (6)	.999 (3)
nursery	.995 (7)	.998 (1.5)	.997 (3.5)	.996 (5.5)	.997 (3.5)	.996 (5.5)	.998 (1.5)
pendigits	.994 (7)	.997 (2)	.996 (4)	.997 (2)	.997 (2)	.995 (5.5)	.995 (5.5)
pima-indians	.890 (2)	.868 (5)	.875 (3)	.847 (7)	.863 (6)	.873 (4)	.891 (1)
segmentation	.988 (4.5)	.991 (1)	.987 (7)	.988 (4.5)	.988 (4.5)	.990 (2)	.988 (4.5)
tic-tac-toe	.915 (1)	.855 (7)	.859 (5.5)	.859 (5.5)	.905 (3)	.873 (4)	.914 (2)
vehicle	.869 (1)	.849 (3)	.828 (7)	.833 (6)	.838 (5)	.843 (4)	.864 (2)
vowel	.971 (6.5)	.982 (1.5)	.980 (4)	.981 (3)	.982 (1.5)	.971 (6.5)	.975 (5)
wdbc	.968 (5)	.970 (4)	.972 (2)	.971 (3)	.974 (1)	.966 (6.5)	.966 (6.5)
yeast	.919 (5)	.922 (3)	.909 (7)	.924 (2)	.911 (6)	.921 (4)	.934 (1)
Avg. rank	(4.900)	(3.067)	(4.700)	(4.300)	(3.500)	(4.200)	(3.333)

Table 3.6: Weighted semantic PrevStab scores (DKM criterion).

rather small according to this metric, and the average ranks of any two methods were always within 2.4 of one another. To put this into context, on the semantic FinalStab metric, the average ranks ranged by as much as 3.7. Once again, QBag garnered the highest average rank, independent of the splitting criterion employed; in both cases, Uncertainty was a close second. In fact, Uncertainty was the most stable method on 9 of the 30 trials, whereas QBag achieved a rank of 1 only 3 times. On the other hand, the stability of Uncertainty was more variable across different datasets, as it achieved a rank of 5 or lower 8 times, compared to only 3 times for QBag, whose performance was more consistent on this metric. On the german dataset, for example, Uncertainty scores higher than QBag on almost every iteration, and the methods rank 1st and 2nd, respectively. By contrast, Uncertainty's rank slips to 5th on the wdbc dataset, while QBag is the most stable method. The scores for these two datasets are plotted in Figure 3.8. Both QBag and Uncertainty were significantly more stable than Random when the entropy criterion was used to grow trees; however, the null hypothesis could not be rejected when DKM was used.

3.4.3 Absolute Structural Stability

First, the FinalStab scores, which measure the structural similarity between the models induced at each iteration of learning and the final model, are compared for the various sampling methods. Although QBag was found to be significantly more stable than Random according to the semantic FinalStab metric, no clear winner(s) emerged with respect to structural FinalStab. The results obtained when using $\varepsilon_{pct} = 0$ are presented in Table 3.7 and Table 3.8. At this setting of ε_{pct}, no method achieved an average rank that was better than 3. Random sampling ranked the lowest, on average, when entropy was used, yet its average performance for DKM was comparable to that of the selective sampling methods. Switching from entropy to DKM had the opposite effect on QBoost, as its average rank declined from 3.2 to 4.267. Only QBoostC and Uncertainty were among 3 most stable procedures regardless of the splitting criterion employed.

As the relatively small ranges of the average rankings would suggest, superiority in terms of structural stability was fairly evenly divided among the different sampling techniques. The Friedman test did not detect significant differences between the methods at $\alpha = .05$ for either splitting criterion. As an example of the general inconsistency of the structural stability of these sampling methods, consider the results for the nursery and australian datasets, which are displayed in Figure 3.9 (left) and Figure 3.9 (right), respectively. Of the methods displayed in the nursery graph, QBag is consistently the most structurally stable, Random is almost invariably the least stable, and the scores for QBoost and QBoostC lie almost directly in the middle of these two. On the australian dataset, however, Random is notably more stable than QBoost, and is more stable than QBag on most iterations.

When compared with error rate, and to a lesser degree semantic stability, structural stability scores often exhibited much more variance within a given run of active learning; also, large discrep-

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.112 (7)	.278 (3)	.360 (1)	.319 (2)	.154 (6)	.157 (5)	.261 (4)
australian	.737 (3)	.663 (5)	.447 (7)	.498 (6)	.732 (4)	.742 (2)	.881 (1)
car	.782 (7)	.836 (4)	.821 (5)	.837 (3)	.844 (2)	.862 (1)	.790 (6)
german	.157 (7)	.281 (5)	.354 (2)	.287 (4)	.196 (6)	.315 (3)	.564 (1)
kr-vs-kp	.825 (5)	.842 (4)	.853 (3)	.854 (2)	.555 (7)	.880 (1)	.650 (6)
letter	.475 (6)	.730 (3)	.833 (1)	.778 (2)	.714 (4)	.471 (7)	.606 (5)
nursery	.841 (7)	.942 (1)	.899 (5)	.894 (6)	.935 (2)	.906 (4)	.908 (3)
pendigits	.183 (7)	.645 (3)	.730 (2)	.749 (1)	.507 (4)	.427 (6)	.457 (5)
pima-indians	.035 (6)	.114 (1)	.071 (3)	.061 (4)	.027 (7)	.055 (5)	.086 (2)
segmentation	.126 (6)	.375 (3)	.447 (1)	.086 (7)	.398 (2)	.234 (4)	.163 (5)
tic-tac-toe	.455 (1)	.254 (7)	.264 (6)	.307 (4)	.276 (5)	.333 (3)	.416 (2)
vehicle	.073 (1)	.014 (5)	.010 (6)	.041 (2)	.006 (7)	.036 (3)	.020 (4)
vowel	.090 (7)	.595 (4)	.650 (2)	.691 (1)	.640 (3)	.303 (6)	.414 (5)
wdbc	.000 (6.5)	.185 (2)	.147 (3)	.259 (1)	.000 (6.5)	.108 (4)	.105 (5)
yeast	.111 (3)	.014 (7)	.151 (1)	.019 (6)	.022 (5)	.024 (4)	.138 (2)
Avg. rank	(5.300)	(3.800)	(3.200)	(3.400)	(4.700)	(3.867)	(3.733)

Table 3.7: Weighted structural FinalStab scores (entropy criterion, $\varepsilon_{pct} = 0$).

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.253 (3)	.194 (4)	.137 (7)	.143 (6)	.320 (1)	.187 (5)	.263 (2)
australian	.959 (1)	.637 (4)	.618 (5)	.571 (7)	.677 (3)	.574 (6)	.778 (2)
car	.798 (5)	.819 (4)	.827 (3)	.891 (1)	.684 (7)	.731 (6)	.841 (2)
german	.465 (7)	.595 (2)	.540 (5)	.553 (3)	.506 (6)	.545 (4)	.777 (1)
kr-vs-kp	.821 (5)	.962 (2)	.967 (1)	.960 (3)	.627 (6)	.946 (4)	.616 (7)
letter	.485 (6)	.793 (4)	.880 (3)	.888 (1)	.881 (2)	.538 (5)	.457 (7)
nursery	.822 (7)	.942 (1)	.926 (4)	.872 (6)	.935 (2)	.903 (5)	.929 (3)
pendigits	.239 (6)	.732 (1)	.603 (3)	.574 (4)	.646 (2)	.306 (5)	.182 (7)
pima-indians	.069 (3)	.064 (4)	.054 (5)	.071 (2)	.029 (7)	.033 (6)	.100 (1)
segmentation	.285 (4)	.404 (2)	.138 (7)	.330 (3)	.162 (5)	.464 (1)	.157 (6)
tic-tac-toe	.449 (1)	.253 (5)	.307 (3)	.304 (4)	.245 (7)	.252 (6)	.376 (2)
vehicle	.109 (1)	.000 (7)	.022 (6)	.038 (4)	.060 (2)	.023 (5)	.050 (3)
vowel	.157 (7)	.653 (4)	.713 (3)	.733 (1.5)	.733 (1.5)	.647 (5)	.641 (6)
wdbc	.127 (6)	.146 (5)	.153 (4)	.203 (2)	.392 (1)	.177 (3)	.044 (7)
yeast	.133 (3)	.121 (4)	.109 (5)	.108 (6)	.031 (7)	.269 (1)	.134 (2)
Avg. rank	(4.333)	(3.533)	(4.267)	(3.567)	(3.967)	(4.467)	(3.867)

Table 3.8: Weighted structural FinalStab scores (DKM criterion, $\varepsilon_{pct} = 0$).

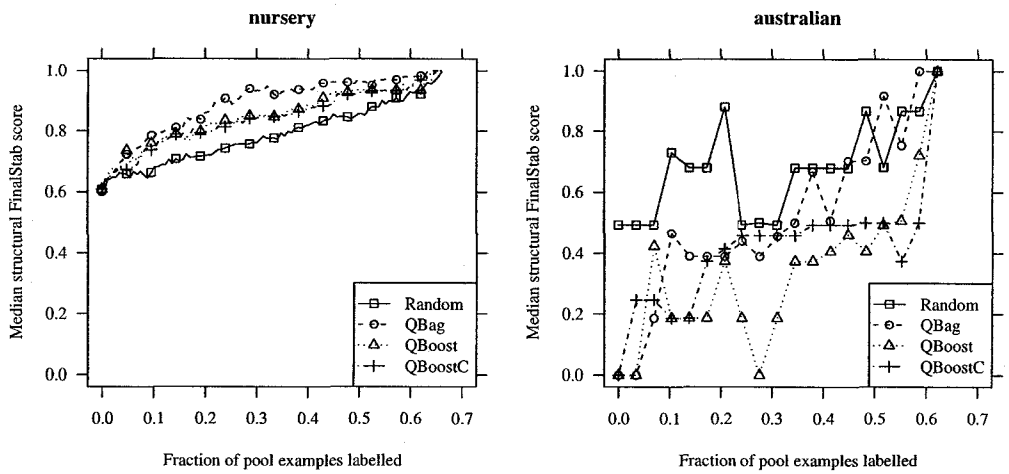

Figure 3.9: Median structural FinalStab scores on the nursery dataset (left) and australian dataset (right) when using the entropy criterion ($\epsilon_{pct} = 0$).

ancies between the performance of the various sampling procedures were more common for this metric. The results on the australian dataset in Figure 3.9 provide a good example of both phenomena, as the median stability scores for QBoost, for example, dropped and then increased sharply during the middle iterations; also, the stability of Random differed from that of QBoost by more than 0.5 in some cases. High variance in the region stability scores was also observed on many of the datasets in the decision tree stability experiments that were conducted in Section 2.5.

With regard to the structural PrevStab measure, it was again the case that no selective sampling method produced models that were consistently more stable than those produced by the other methods, when region stability scores were calculated using $\epsilon_{pct} = 0$. All the average ranks were within the range $[3.3, 4.7]$, independent of whether entropy or DKM was used, and the null hypothesis could not be rejected in either case. On individual datasets, however, there were marked differences between the stability of some methods. Consider the graph in the Figure 3.10, which displays the results for three of the sampling methods on the pendigits dataset. Here, the models induced on consecutive iterations by Random differed by a moderate but fairly consistent amount – in the neighbourhood of 0.2 by the region stability metric – across virtually the entire learning curve. On the other hand, for both QBoostC and Uncertainty, the hypotheses often changed dramatically during the first half of the iterations as the result of adding new batches of examples to the training set. However, in the second half of the learning curve, both of these methods surpassed the stability of Random, and the decision regions of the classifiers produced after approximately 45% of the pool examples had been labelled did not change at all between iterations. It is important to note that both selective sampling methods achieved lower weighted average error rates than Random on this dataset.

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.524 (2)	.369 (7)	.471 (6)	.518 (3)	.501 (4)	.491 (5)	.631 (1)
australian	.845 (4)	.892 (2)	.784 (7)	.789 (6)	.839 (5)	.861 (3)	.978 (1)
car	.933 (3)	.928 (5)	.821 (7)	.929 (4)	.941 (2)	.927 (6)	.945 (1)
german	.533 (4)	.573 (2)	.473 (7)	.520 (5)	.477 (6)	.539 (3)	.777 (1)
kr-vs-kp	.964 (2)	.965 (1)	.955 (5)	.958 (3)	.928 (6)	.957 (4)	.915 (7)
letter	.959 (6)	.981 (1)	.979 (2)	.978 (3)	.968 (5)	.969 (4)	.943 (7)
nursery	.978 (7)	.989 (2)	.988 (3.5)	.988 (3.5)	.987 (5)	.985 (6)	.993 (1)
pendigits	.860 (7)	.929 (3)	.943 (2)	.948 (1)	.922 (4)	.894 (6)	.908 (5)
pima-indians	.201 (5)	.245 (3)	.309 (2)	.187 (6)	.161 (7)	.235 (4)	.403 (1)
segmentation	.674 (3.5)	.755 (1)	.674 (3.5)	.635 (6)	.718 (2)	.663 (5)	.497 (7)
tic-tac-toe	.718 (2)	.598 (5)	.528 (6)	.523 (7)	.622 (3)	.619 (4)	.768 (1)
vehicle	.223 (1)	.093 (7)	.123 (4)	.121 (5)	.114 (6)	.219 (2)	.178 (3)
vowel	.669 (7)	.770 (1)	.756 (2.5)	.756 (2.5)	.721 (5)	.735 (4)	.708 (6)
wdbc	.262 (7)	.603 (3)	.610 (2)	.660 (1)	.489 (4)	.353 (6)	.457 (5)
yeast	.268 (4)	.131 (6.5)	.292 (2)	.183 (5)	.131 (6.5)	.297 (1)	.279 (3)
Avg. rank	(4.300)	(3.300)	(4.100)	(4.067)	(4.700)	(4.200)	(3.333)

Table 3.9: Weighted structural PrevStab scores (entropy criterion, $\varepsilon_{pct} = 0$).

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.698 (2)	.502 (5.5)	.502 (5.5)	.420 (7)	.651 (3)	.586 (4)	.738 (1)
australian	.986 (1)	.846 (7)	.882 (6)	.946 (2)	.935 (3)	.932 (4)	.929 (5)
car	.932 (3)	.923 (5)	.905 (7)	.948 (1)	.919 (6)	.945 (2)	.929 (4)
german	.669 (7)	.737 (2)	.680 (6)	.730 (3)	.711 (4.5)	.711 (4.5)	.895 (1)
kr-vs-kp	.960 (5)	.982 (3)	.981 (4)	.984 (2)	.868 (7)	.989 (1)	.894 (6)
letter	.962 (6)	.989 (2.5)	.991 (1)	.989 (2.5)	.983 (4)	.976 (5)	.918 (7)
nursery	.981 (7)	.989 (3)	.989 (3)	.986 (5.5)	.989 (3)	.986 (5.5)	.992 (1)
pendigits	.888 (7)	.914 (4)	.933 (1)	.920 (3)	.921 (2)	.909 (5)	.896 (6)
pima-indians	.301 (2)	.242 (4)	.264 (3)	.168 (6)	.147 (7)	.236 (5)	.419 (1)
segmentation	.707 (3)	.801 (1)	.497 (7)	.631 (4)	.588 (6)	.796 (2)	.593 (5)
tic-tac-toe	.673 (2)	.519 (7)	.547 (6)	.566 (5)	.652 (3)	.581 (4)	.720 (1)
vehicle	.359 (1)	.145 (5)	.090 (7)	.153 (4)	.139 (6)	.183 (3)	.302 (2)
vowel	.759 (7)	.866 (2.5)	.862 (4)	.866 (2.5)	.882 (1)	.811 (6)	.840 (5)
wdbc	.576 (2)	.499 (5)	.492 (6)	.477 (7)	.672 (1)	.549 (3)	.522 (4)
yeast	.324 (6)	.385 (4)	.434 (2)	.393 (3)	.138 (7)	.450 (1)	.368 (5)
Avg. rank	(4.067)	(4.033)	(4.567)	(3.833)	(4.233)	(3.667)	(3.600)

Table 3.10: Weighted structural PrevStab scores (DKM criterion, $\varepsilon_{pct} = 0$).

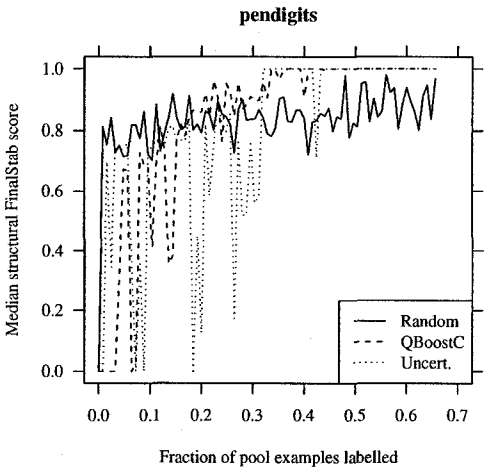

Figure 3.10: Median structural PrevStab scores on the pendigits dataset when using the entropy criterion ($\varepsilon_{pct} = 0$).

On some of the datasets that consisted entirely of continuous attributes, such pima-indians and vehicle, the weighted averages of the structural FinalStab and PrevStab scores were rather low for all the sampling methods – the decision regions underwent significant changes even after a large fraction of the pool examples had been labelled. This is a symptom of requiring that decision thresholds be identical when computing region stability by having $\varepsilon_{pct} = 0$. Although other datasets that are composed of only continuous attributes, including letter and pendigits, yielded relatively high scores in spite of this parameter setting, these typically contained a much larger number of training examples, which were sufficient to allow C4.5 to consistently define the heavily weighted decision regions in the early stages of active learning.

Whereas the structural FinalStab and PrevStab scores did not reveal statistically significant differences between the sampling methods based on the Friedman test, the differences observed on the RunStab metric were highly significant. The weighted averages for this measure when using entropy and DKM are displayed in Table 3.11 and Table 3.12, respectively. Random and Uncertainty received the two lowest average ranks, regardless of the splitting criterion employed, while QBag, QBoost, and QBoostC occupied the top 3 positions in both cases. The Friedman test found that the discrepancies between the average ranks were significant at $\alpha = .01$, and the critical difference calculated by the Nemenyi for this α was 2.727. Thus, for the entropy criterion QBag, QBoost, and QBoostC were each significantly more stable than both Random and Uncertainty; the only change in this regard when using DKM was that QBoost's average rank dropped to the point where it was no longer significantly better than any other method. Although BootLV and BootM ranked 4th and 5th, respectively, for both splitting criteria, no strong conclusions could be made regarding these two methods.

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.127 (4)	.164 (3)	.237 (2)	.263 (1)	.109 (5)	.091 (6)	.040 (7)
australian	.495 (5)	.594 (2)	.464 (6)	.455 (7)	.519 (4)	.580 (3)	.652 (1)
car	.533 (7)	.598 (5)	.617 (3)	.739 (1)	.710 (2)	.577 (6)	.601 (4)
german	.103 (7)	.198 (2)	.208 (1)	.166 (4)	.187 (3)	.139 (5)	.126 (6)
kr-vs-kp	.662 (5)	.843 (2)	.837 (3)	.870 (1)	.470 (6)	.803 (4)	.343 (7)
letter	.329 (6)	.714 (3)	.797 (1)	.725 (2)	.709 (4)	.420 (5)	.318 (7)
nursery	.767 (7)	.915 (3.5)	.927 (2)	.915 (3.5)	.933 (1)	.850 (5)	.827 (6)
pendigits	.089 (7)	.616 (3)	.680 (1)	.661 (2)	.497 (4)	.217 (5)	.137 (6)
pima-indians	.007 (6)	.035 (1)	.033 (2)	.019 (3)	.005 (7)	.009 (4.5)	.009 (4.5)
segmentation	.010 (7)	.222 (2)	.236 (1)	.117 (4)	.218 (3)	.063 (5)	.016 (6)
tic-tac-toe	.189 (5)	.203 (2)	.207 (1)	.194 (4)	.153 (7)	.198 (3)	.166 (6)
vehicle	.005 (5.5)	.011 (3)	.010 (4)	.015 (1)	.003 (7)	.012 (2)	.005 (5.5)
vowel	.055 (7)	.453 (4)	.529 (3)	.595 (1)	.558 (2)	.066 (6)	.092 (5)
wdbc	.002 (7)	.151 (2)	.123 (3)	.176 (1)	.100 (4)	.014 (5)	.009 (6)
yeast	.020 (6)	.035 (3)	.056 (1)	.022 (5)	.027 (4)	.045 (2)	.013 (7)
Avg. rank	(6.100)	(2.700)	(2.267)	(2.700)	(4.200)	(4.433)	(5.600)

Table 3.11: Weighted structural RunStab scores (entropy criterion, $\varepsilon_{pct} = 0$).

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.111 (5)	.132 (2)	.106 (6)	.120 (3)	.198 (1)	.115 (4)	.082 (7)
australian	.762 (1)	.577 (6)	.597 (5)	.642 (3)	.715 (2)	.610 (4)	.543 (7)
car	.542 (7)	.631 (3)	.598 (5)	.700 (1)	.649 (2)	.614 (4)	.548 (6)
german	.329 (7)	.499 (1)	.397 (4)	.461 (2)	.395 (5)	.365 (6)	.430 (3)
kr-vs-kp	.737 (5)	.920 (2)	.921 (1)	.894 (3)	.449 (7)	.865 (4)	.460 (6)
letter	.298 (7)	.705 (3)	.900 (2)	.917 (1)	.660 (4)	.330 (5)	.313 (6)
nursery	.768 (7)	.921 (3)	.926 (2)	.911 (4)	.944 (1)	.858 (5)	.841 (6)
pendigits	.046 (7)	.613 (1)	.452 (3)	.407 (4)	.590 (2)	.168 (5)	.054 (6)
pima-indians	.008 (5)	.037 (2)	.014 (4)	.040 (1)	.005 (7)	.006 (6)	.021 (3)
segmentation	.122 (5)	.187 (2)	.129 (3)	.207 (1)	.084 (6)	.128 (4)	.021 (7)
tic-tac-toe	.176 (5)	.201 (3)	.221 (2)	.223 (1)	.171 (6)	.183 (4)	.153 (7)
vehicle	.014 (2)	.020 (1)	.007 (4.5)	.007 (4.5)	.012 (3)	.005 (6)	.003 (7)
vowel	.310 (6)	.499 (3)	.632 (2)	.678 (1)	.474 (4)	.389 (5)	.214 (7)
wdbc	.008 (7)	.207 (2)	.104 (4)	.169 (3)	.357 (1)	.021 (5)	.014 (6)
yeast	.038 (5)	.125 (1)	.103 (3)	.098 (4)	.021 (7)	.120 (2)	.033 (6)
Avg. rank	(5.400)	(2.333)	(3.367)	(2.433)	(3.867)	(4.600)	(6.000)

Table 3.12: Weighted structural RunStab scores (DKM criterion, $\varepsilon_{pct} = 0$).

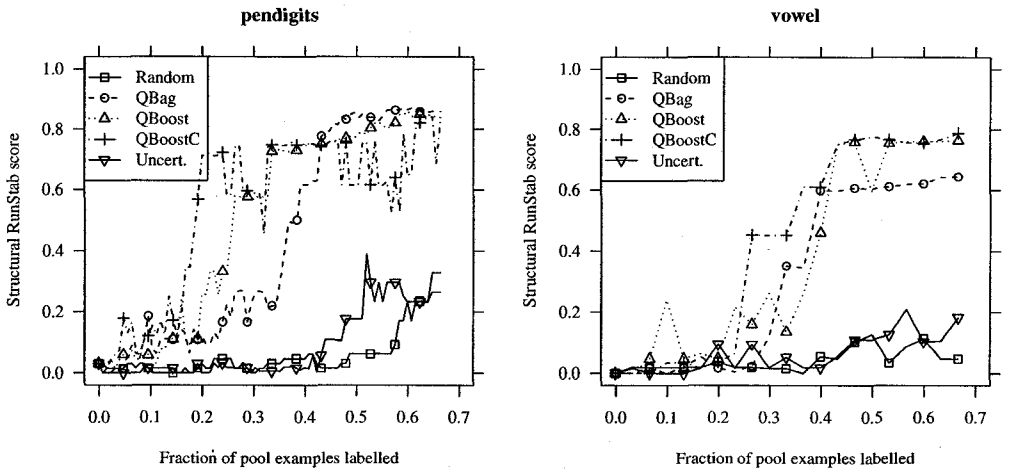

Figure 3.11: Structural FinalStab scores on the pendigits and vowel datasets (right) when using the entropy criterion ($\varepsilon_{pct} = 0$).

These RunStab results provide substantial evidence that Random and Uncertainty tend to produce models from different initial training data that have relatively little structure in common with one another. Not only did these two methods consistently rank lower than the other selective sampling procedures, but the magnitudes of the differences between their scores and those of the top-ranked samplers were rather large in many cases. The pendigits and vowel datasets offer two such examples (see Figure 3.11). On both datasets, there were clear gaps separating the top-ranked methods from Random and Uncertainty. After approximately 40% of the pool examples were labelled, the three committee-based selective samplers shown here each produced models whose structure was fairly consistent across different runs; on the other hand, the trees induced from different initial training sets by Random and Uncertainty bore little resemblance to one another, even in the final iterations of selective sampling. Thus, Random and Uncertainty were very sensitive to the particular instances that composed the initial training sets, whereas QBag, QBoost, and QBoostC tended to yield similar decision trees regardless of the starting point, once they had labelled a moderate number of examples.

3.4.4 Approximate Structural Stability

Next, the effect on structural stability of increasing ε_{pct} to values of 5, 10, and 15 percent is examined. The car, kr-vs-kp, nursery, and tic-tac-toe datasets each contain only discrete attributes. Although varying the ε_{pct} parameter has no effect on the region stability scores that are calculated for these datasets, they are included in each table for the purpose of computing average ranks for the sampling methods. The weighted averages of the FinalStab and PrevStab scores for these values of ε_{pct} can be found in Appendix C.1.

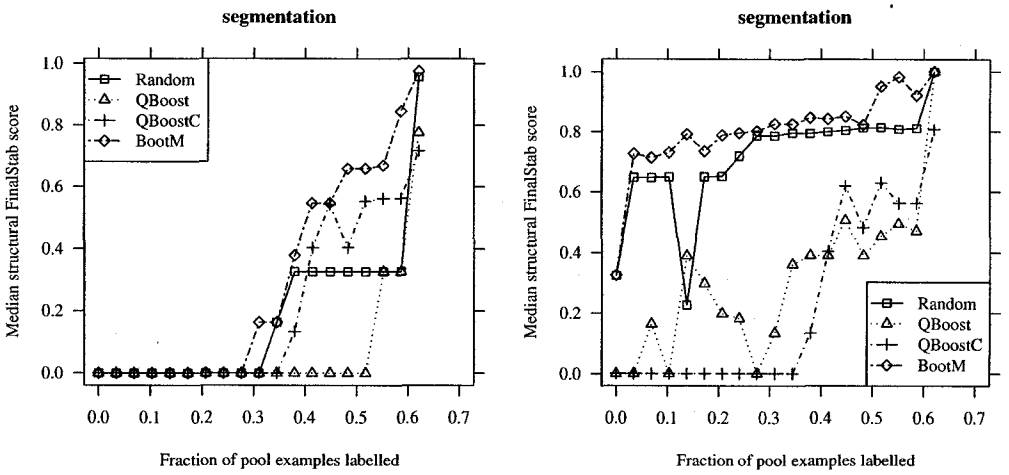

Figure 3.12: Median structural FinalStab scores on the segmentation dataset when using the DKM criterion. In the left graph, the region stability metric was calculated using $\varepsilon_{pct} = 0$, whereas in the right figure, $\varepsilon_{pct} = 5\%$ was used.

When using the entropy splitting criterion, Random maintained the worst average rank on the FinalStab metric for all values of ε_{pct}. With DKM, Random and BootM always occupied the bottom two positions. However, as with $\varepsilon_{pct} = 0$, none of the differences were significant for any positive values of ε_{pct}. Overall, increasing ε_{pct} had little impact on the relative stability of the selective sampling methods. For example, when using the entropy criterion, the top-ranked method(s) were the same across all values of ε_{pct} for 7 of the 12 datasets that contained at least one continuous attribute. When DKM was the splitting criterion, this was the case for 9 of these datasets. Thus, the structural FinalStab scores computed using $\varepsilon_{pct} = 0$ were very good indicators of the relative performance of the methods as the allowable margin of error for "almost equal" cut-points was raised.

Although the relative stability of the sampling methods changed little as the result of varying ε_{pct}, the absolute stability scores sometimes underwent profound increases as the result of permitting continuous thresholds that differed by small amounts to be considered equal by the region stability metric. For example, when using DKM on the segmentation dataset, the weighted FinalStab scores of all the methods increased by an average of .252 when ε_{pct} was increased from 0 to 5%. Individually, the largest gain was observed for Random, whose score jumped from .285 to .783. Figure 3.12 shows the results for BootM, QBoost, QBoostC, and Random on this dataset. None of these methods produced trees that had even a single decision region that was identical to a region defined by the final models they eventually created, until at least 30% of the pool examples were labelled. In fact, QBoost required the labels of more than half the pool examples before its models began to even remotely resemble the classifier produced during the final iteration. Yet, when structural stability

was assessed using $\varepsilon_{pct} = 5\%$, the scores for 3 of the 4 methods shown here increased substantially. BootM retained the top ranking, while Random improved from the 4th to the 2nd position, and as mentioned, achieved the largest increase of any method, including those not plotted here. It turns out that the models induced by BootM and Random in the early stages of learning actually did have a considerable amount of structure in common with their respective final models, but many of the thresholds defined for continuous attributes differed from those specified by the final classifiers by less than 5% of their respective attribute ranges. By contrast, QBoostC hardly benefitted from the allowable margin of error when comparing cut-points, as it remained completely unstable during the first half of the iterations; moreover, its stability scores beyond that point were virtually indistinguishable from those attained when using $\varepsilon_{pct} = 0$. The models induced by QBoostC during the first 11 iterations (0 to .34 on the x-axis) either performed tests on different attributes or defined cut-points that were distant from those appearing in the final model – even at $\varepsilon_{pct} = 15\%$, which was the largest margin of discrepancy used to compute stability scores, the weighted average score for QBoostC on the segmentation dataset was .372 (up from .330 at $\varepsilon_{pct} = 0$), and the characteristics of its FinalStab curve changed very little.

In applying the same analysis to the structural PrevStab scores for different choices of ε_{pct}, it was found that the scores calculated using $\varepsilon_{pct} = 0$ were also good indicators of the relative stability of the sampling methods when "almost equal" cut-points were acknowledged. Regardless of the splitting criterion employed, the top ranking did not change on the anneal, car, german, letter, pima-indians, or vehicle datasets. Additionally, the top-ranked method did not change on either of the australian, pendigits, or segmentation datasets when using entropy, whereas BootLV was always superior on the wdbc dataset when DKM was used. Although the FinalStab scores for Random were consistently lower than those of the selective sampling procedures, its PrevStab scores were generally competitive with the other methods across different values of ε_{pct}. Uncertainty and QBag were the most stable methods, on average, when the entropy criterion was used. However, Uncertainty struggled on datasets that had only continuous attributes and many training examples, particularly on the letter and segmentation datasets, where it ranked last for all values of ε_{pct}. For the DKM criterion, QBoostC, BootM, and Uncertainty competed for the highest average ranks. Still, no algorithm was significantly better than any other at the values of ε_{pct} that were used, for either splitting criterion.

With respect to the RunStab metric, the average ranks underwent few changes when scores were calculated using positive values of ε_{pct}. Uncertainty was always significantly inferior to QBag, QBoost, and QBoostC; the only exception was at $\varepsilon_{pct} = 15\%$ when using the DKM criterion, where Uncertainty obtained the worst average rank, but by amounts that were less than the Nemenyi critical difference versus all but the QBoostC method. While the average rank for Random was better than 5 in only one case ($\varepsilon_{pct} = 5\%$, DKM criterion), the differences between it and each of the 3 top-ranked methods were not always statistically significant, as they were at $\varepsilon_{pct} = 0$. The BootLV

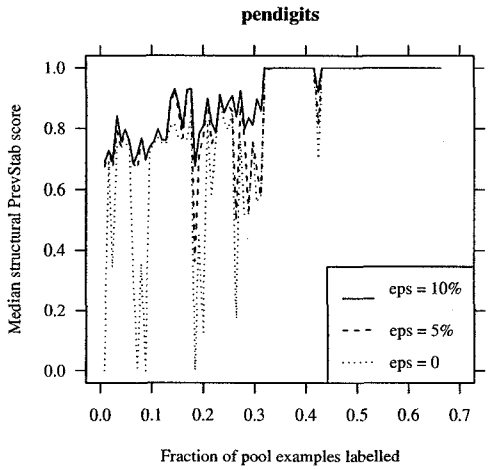

Figure 3.13: Median structural PrevStab scores of Uncertainty on the pendigits dataset when using the entropy criterion, for $\varepsilon_{pct} \in \{0, 5\%, 10\%\}$. In the legend, "eps" represents ε_{pct}.

and BootM methods, occupied the 4th and 5th positions in all but one situation ($\varepsilon_{pct} = 15\%$, DKM criterion). Finally, the best overall performance was delivered by QBoostC, whose average ranks placed it no lower than 2nd position for any combination of splitting criterion and choice of ε_{pct}. In fact, if significant differences are assessed at $\alpha = .05$ rather than at $\alpha = .01$, QBoostC is always significantly more stable than Random and Uncertainty, and is the only method to accomplish this feat.

On some datasets, increasing the value of ε_{pct} revealed that the instability observed when using $\varepsilon_{pct} = 0$ was, to a large degree, superficial. Figure 3.13 compares the median PrevStab stability scores for Uncertainty, using $\varepsilon_{pct} \in \{0, 5\%, 10\%\}$ (recall that the plot for $\varepsilon_{pct} = 0$ also appears in Figure 3.10). When stability was calculated using $\varepsilon_{pct} = 0$, Uncertainty appeared to be highly unstable during many of the early iterations of active learning – the models induced on consecutive iterations frequently had few or no decision regions in common. However, by allowing some leeway between the continuous thresholds specified by these models, it became clear that this instability was due to moderate discrepancies between the thresholds, and that many of the decision regions defined on consecutive iterations were actually very similar. In fact, the majority of the thresholds defined by these regions were separated by no more than 10% of their respective attributes ranges.

Although several examples were presented in which at least one sampling method was substantially more stable than another on a particular dataset, according to the Friedman test, no method was significantly better than any other on either of the FinalStab or PrevStab stability metrics, for any combination of splitting criterion and ε_{pct} value that was employed. Yet, when the entropy criterion was used, Random was invariably the lowest-ranked method on the FinalStab measure, regardless of the value of ε_{pct} used to calculate structural stability. Suppose that instead of com-

ε_{pct} (%)	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
0	11-4-0	12-3-0	10-5-0	8-6-1	11-4-0	12-3-0
5	11-4-0	11-4-0	9-6-0	9-6-0	11-4-0	9-6-0
10	11-4-0	11-4-0	9-6-0	9-6-0	11-4-0	9-6-0
15	11-4-0	11-4-0	9-6-0	9-6-0	10-5-0	9-6-0

Table 3.13: Structural FinalStab win-loss-draw counts versus Random, when using the entropy criterion.

paring the average ranks of all the methods, a direct comparison between the scores obtained by Random and QBag was conducted (QBag achieved the lowest rank when averaged over all values of ε_{pct}). By this token, the win-loss count for QBag was 11-4 for each value of ε_{pct}. The one-sided Wilcoxon signed-ranks test [54], which is appropriate when comparing two sampling methods [13], determined that at $\alpha = .05$, QBag was significantly more stable than Random when ε_{pct} was 0 or 10% (significance was narrowly missed at this α level when ε_{pct} was 5%). The drawback when comparing several methods simultaneously is that the highest ranks may be fairly evenly divided among a subset S of the methods, while individually, each method in S tends to be superior to the methods not in S. As a consequence, the differences between the ranks of the methods in S and those not in S are relatively smaller than they would be if pairwise comparisons were made instead.

Based on the direct comparison between QBag and Random, it can be concluded that the former is more structurally stable on the FinalStab metric. Two other methods, QBoost and BootM, were also more stable than Random on at least two-thirds of the datasets for all values of ε_{pct}, and none of the methods posted losing records versus Random (see Table 3.13). These results suggest that, overall, the selective sampling procedures are more stable than random sampling. When the same comparisons were made using the DKM data, however, Random fared much better against the selective samplers, sometimes winning on more than half of the datasets. Thus, only when entropy was used to grow decision trees did selecting examples at random appear to have a negative effect on structural stability.

When examples are chosen randomly while using the entropy criterion, the progression toward the final model is comparatively less smooth than it is with other selection methods. Put another way, if the sampling process is terminated at a given iteration, the model produced at this point is less likely to have structure in common with the other classifiers in the sequence when random sampling is used. Although Random was competitive on the PrevStab metric, it is important to put these statistics into perspective. Given that it achieves relatively low FinalStab scores, the modest PrevStab levels recorded for Random indicate that it tends to produce classifiers that do not differ significantly between consecutive iterations, but which nevertheless remain dissimilar from the final model. In this situation, good stability on consecutive iterations is of little value if the knowledge extraction is the ultimate goal, since the simultaneously low FinalStab scores imply that the classifiers induced during most iterations have little structure in common with the final model.

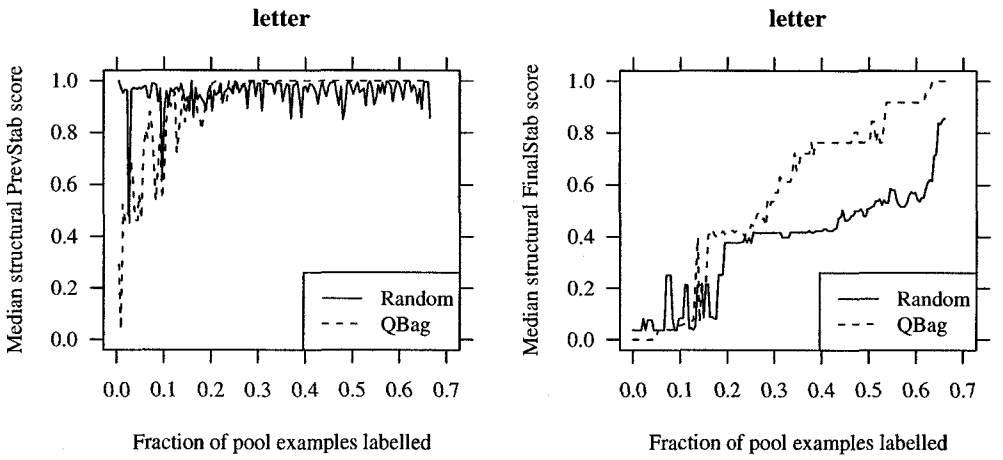

Figure 3.14: Median structural PrevStab scores (left) and FinalStab scores (right) on the letter dataset when using the entropy criterion. The region stability metric was calculated using $\varepsilon_{pct} = 5\%$.

The reason that Random is often less stable than the selective sampling methods according to the FinalStab metric, but comparatively stable on the PrevStab metric, is that the latter methods are designed to seek out the most informative examples in the unlabelled pool. When successful, the selective samplers add many examples to the training set in the early stages of active learning that are essential for learning the target concept accurately. Once the majority of these particularly informative examples have been labelled, the examples that are added to the training set in later iterations convey relatively little new information, and therefore exert less influence on the hypothesis. In contrast, if examples are selected at random, the highly informative cases are acquired gradually, rather than predominantly during the early iterations. The result is that there is less instability on consecutive iterations in the early stages of learning, but that the hypothesis continues to change as informative examples are labelled later in the active learning process, resulting in lower FinalStab scores.

A typical example of this behaviour was observed on the letter dataset. Figure 3.14 displays plots of the PrevStab and FinalStab scores recorded on this dataset for QBag and Random. The PrevStab scores, shown in the left picture, reveal that the consecutive models produced by Random during the early iterations almost always had more structure in common with one another than did the classifiers induced using QBag. The hypotheses formed by QBag changed significantly in the early rounds, as many highly informative examples were labelled, whereas a large portion of the examples obtained by Random were redundant, and exerted little influence on the existing model. The error rate plots displayed in Figure 3.15 support this interpretation of the stability scores. Referring to the FinalStab curve, at the point where 40% of the pool examples had been labelled, the model induced by QBag had much structure in common with its final model. As a result of labelling the most

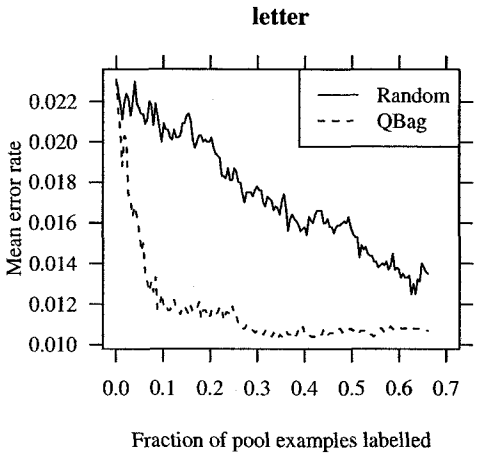

Figure 3.15: Mean error rates on the letter dataset when using the entropy criterion.

informative examples in the early rounds, there were fewer "surprises" – examples that were not adequately explained by the existing hypothesis – in the later stages of learning. Comparatively, the models induced by Random tended to have fewer decision regions in common with the final model that was formed. This is mainly due to the abrupt changes that were made to the hypothesis during the final iterations of learning. The batches of instances labelled in the final rounds contained some influential examples that caused a significant fraction of the decision regions to be adjusted. This is precisely the type of situation that one wishes to avoid if the final model produced by selective sampling is to be interpreted. Due to the instability of Random on this dataset, if a few less iterations had been performed, the structure of the resulting model would have been very different.

Random also performed poorly on the RunStab measure, although Uncertainty was even more sensitive to the particular initial training samples that were provided, in terms of the models it produced. As well, BootLV and BootM were mediocre in this regard, they were generally uncompetitive with the top-ranked sampling methods. Interestingly, it was the 3 committee-based methods that employed the direct selection strategy when choosing examples, which tended to build models that were structurally similar to one another across different runs of selective sampling. With the exception of Uncertainty, the stability across different initial training sets appears to be related to the amount of stochasticity that is present in each method. The procedures that use direct selection always choose the highest scoring examples; therefore, a given example will be added to the training set provided that it receives a sufficiently high score at some iteration. The weight sampling methods do not necessarily choose the highest scoring examples, although the selection procedure uses the scores as weights, so that the distribution over the examples is typically non-uniform. On the other hand, Random is completely stochastic, and this is reflected by the fact that it formed

training sets that were less similar across runs than those formed by the aforementioned methods. As for Uncertainty, its failure to produce structurally similar trees from different initial training sets is due to the fact that it assigns scores to unlabelled examples based on the hypothesis of a single decision tree. Since this tree is generally unstable, the score assigned to a given example is likely to change considerably when the tree is induced from different training data. This is less of a problem with the committee-based methods because a voted hypothesis tends to be more stable than a single tree [5]. Thus, despite being deterministic in nature, Uncertainty is very sensitive to the particular examples that form the training set, and as a consequence, it tends to produce dissimilar models across different selective sampling runs.

3.4.5 Comparison of Splitting Criteria

An examination of the relative performance of the two splitting criteria revealed that DKM frequently produced models that were more structurally stable. This trend was observed for all three active learning stability metrics, and for all values of ε_{pct} that were tested. With 15 datasets and 7 sampling methods, 105 FinalStab, PrevStab, and RunStab scores were obtained for each splitting criterion, for a given choice of $\varepsilon_{pct} \in \{0, 5\%, 10\%, 15\%\}$. For each combination of sampler, dataset, and ε_{pct}, the weighted average score achieved by DKM for the given statistic was compared to that of entropy. If the score for entropy was greater than the DKM score, a win was recorded for entropy, and vice versa.

On the FinalStab measure, DKM always accounted for at least 58% of the wins, independent of the ε_{pct} value used. For example, at $\varepsilon_{pct} = 5\%$, 63 wins were recorded for DKM compared to 40 for entropy, and there were 2 draws (see Table 3.14). Regardless of the sampling method used, DKM virtually always had at least as many wins as entropy; thus, it was not the case that DKM was more stable than entropy when using certain samplers, but less stable with others. The only instance in which entropy surpassed DKM's number of wins was with Uncertainty at $\varepsilon_{pct} = 0$, where entropy recorded 8 wins compared to 7 for DKM.

The differences between the PrevStab scores were even more in favour of DKM, which accounted for at least 65% of the wins in each case. The PrevStab data for $\varepsilon_{pct} = 5\%$ is displayed in Table 3.15. DKM always recorded more wins than entropy when using 6 of the 7 sampling methods – only when using Uncertainty with $\varepsilon_{pct} \in \{5\%, 10\%, 15\%\}$ did entropy match or surpass the performance of DKM, and the number of wins in these cases never differed by more than 1. Figure 3.16 displays the structural stability results for the german dataset when the QBag method was used. In this situation, DKM was superior with respect to both stability metrics.

A similar picture emerged when comparing the RunStab scores obtained by entropy and DKM. On this measure, the percentage of cases in which DKM was more stable than entropy ranged from 60 to 67 percent, depending on the value of ε_{pct} that was used to calculate the scores. Table 3.16 displays the score differences for $\varepsilon_{pct} = 5\%$. Only when used with the QBoost method did entropy

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	-.187	.067	.228	.201	-.207	-.140	-.041
australian	-.177	.073	-.271	-.201	.140	.096	.143
car	-.016	.018	-.005	-.054	.160	.132	-.051
german	-.304	-.306	-.183	-.263	-.289	-.200	-.194
kr-vs-kp	.004	-.120	-.114	-.105	-.072	-.067	.034
letter	-.077	-.063	-.047	-.110	-.168	-.094	.149
nursery	.019	.000	-.027	.022	.001	.002	-.020
pendigits	.258	.052	.158	.096	.076	.407	.428
pima-indians	-.014	.144	.066	-.039	-.012	.079	-.034
segmentation	-.455	-.138	.131	-.168	.269	-.518	-.026
tic-tac-toe	.007	.002	-.043	.003	.030	.081	.040
vehicle	-.045	.015	-.012	.029	-.077	.015	-.023
vowel	-.651	-.067	-.041	.022	-.160	-.186	-.206
wdbc	-.140	-.082	-.052	-.057	-.341	-.182	.201
yeast	.043	-.120	-.080	-.291	-.019	-.176	-.037
DKM wins	10	7	11	9	9	8	9
Entropy wins	5	7	4	6	5	7	6
Draws	0	1	0	0	1	0	0

Table 3.14: DKM weighted FinalStab scores subtracted from entropy weighted FinalStab scores, for $\varepsilon_{pct} = 5\%$.

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	-.213	-.090	-.024	.064	-.135	-.113	-.100
australian	-.089	-.032	-.087	-.108	-.068	-.055	.037
car	.001	.006	-.084	-.019	.022	-.018	.016
german	-.134	-.152	-.203	-.190	-.204	-.167	-.088
kr-vs-kp	.004	-.017	-.026	-.025	.059	-.032	.021
letter	-.003	-.008	-.012	-.011	-.015	-.007	.024
nursery	-.003	.000	-.001	.002	-.003	-.001	.001
pendigits	-.016	.035	.017	.024	.017	-.005	.011
pima-indians	-.061	.002	.048	.037	.048	.003	-.041
segmentation	-.060	.023	.124	.044	.153	-.105	-.089
tic-tac-toe	.045	.080	-.019	-.042	-.029	.038	.047
vehicle	-.098	-.064	.061	-.029	-.027	.068	-.125
vowel	-.110	-.081	-.070	-.018	-.086	-.015	-.038
wdbc	-.147	.096	.034	-.028	-.117	-.147	.036
yeast	.004	-.254	-.162	-.208	-.037	-.131	-.108
DKM wins	11	8	10	10	10	12	7
Entropy wins	4	6	5	5	5	3	8
Draws	0	1	0	0	0	0	0

Table 3.15: DKM weighted PrevStab scores subtracted from entropy weighted PrevStab scores, for $\varepsilon_{pct} = 5\%$.

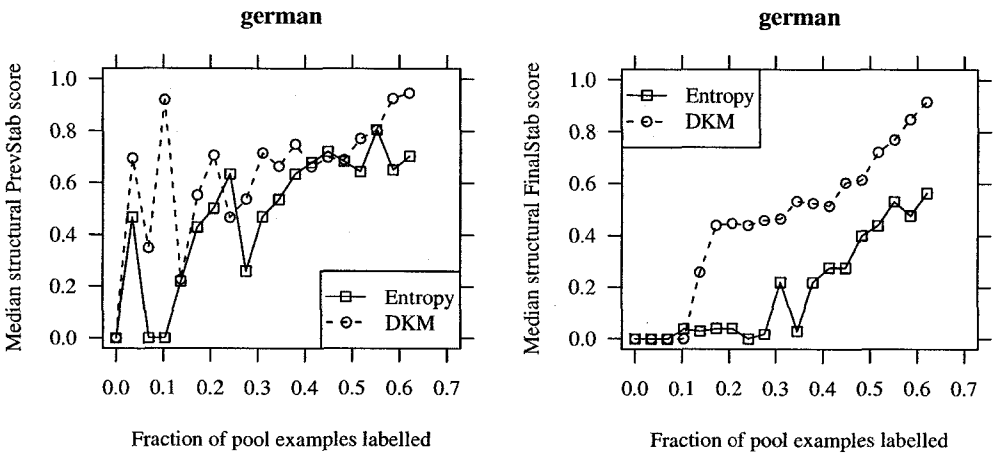

Figure 3.16: Median structural PrevStab scores (left) and FinalStab scores (right) on the german dataset when using the QBag method ($\varepsilon_{pct} = 5\%$).

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.002	.016	.138	.156	-.152	-.082	-.079
australian	-.232	-.066	-.117	-.199	-.175	-.050	.180
car	-.009	-.033	.019	.039	.061	-.037	.053
german	-.225	-.296	-.188	-.287	-.197	-.206	-.292
kr-vs-kp	-.075	-.077	-.084	-.024	.021	-.062	-.117
letter	-.031	.002	-.102	-.192	.032	-.017	.006
nursery	-.001	-.006	.001	.003	-.011	-.008	-.014
pendigits	.297	.050	.247	.150	.035	.142	.234
pima-indians	.012	.050	.035	-.005	-.029	.037	.000
segmentation	-.448	-.201	.226	-.038	.318	-.404	-.042
tic-tac-toe	.013	.002	-.015	-.029	-.017	.015	.013
vehicle	-.025	-.012	.006	.013	-.007	.006	.002
vowel	-.413	-.188	-.082	-.810	.055	-.481	-.410
wdbc	-.057	-.025	.046	-.068	-.270	-.066	-.010
yeast	-.046	-.087	-.111	-.155	.000	-.120	-.060
DKM wins	10	10	7	9	8	11	8
Entropy wins	3	3	7	4	6	4	5
Draws	2	2	1	2	1	0	2

Table 3.16: DKM weighted RunStab scores subtracted from entropy weighted RunStab scores, for $\varepsilon_{pct} = 5\%$.

equal the number of wins recorded by DKM; for the remaining sampling methods, DKM produced models that were more structurally stable across different training sets than those created using entropy on the majority of the datasets.

The Wilcoxon signed-ranks test revealed that the differences between the FinalStab, PrevStab, and RunStab scores acquired using entropy and DKM were highly significant ($\alpha = .01$) for all values of ε_{pct}. It is important to point out that the higher structural stability achieved by DKM did not correspond with a decrease in predictive accuracy; in fact, DKM achieved lower error rates, on average, than did entropy. Specifically, the win-loss-draw counts for DKM were 58-31-16 on this statistic. Although the majority (82) of these discrepancies were less than .005 in magnitude, the consistency with which DKM produced more accurate models was statistically significant at $\alpha = .05$, according to the Wilcoxon test.

The findings from the decision tree stability experiments in Section 2.5 were that DKM tended to grow trees that were more structurally stable than those produced when using the entropy splitting criterion. In those experiments, stability was measured by dividing a given dataset into two disjoint halves, inducing trees from these partitions of the data, and applying the region stability metric to the resulting trees. Results from the active learning experiments conducted in this chapter provide even stronger evidence to suggest that DKM produces more stable models than entropy, based on three different applications of the region stability metric, namely, FinalStab, PrevStab, and RunStab, which were defined in Section 3.1.

3.5 Conclusions

In this chapter, the stability of the C4.5 decision tree algorithm was studied within the context of active learning. The region stability metric formulated in Section 2.4 was used to measure the structural similarity of specific pairs of hypotheses that were induced during the active learning procedure; these adaptations of the metric were named FinalStab, PrevStab, and RunStab. Turney's expected agreement measure (see Section 2.4) was also employed in this manner in conjunction with the first two of these metrics, in order to measure semantic stability. In total, 7 selective sampling methods, including random sampling, were compared on these stability measures.

With respect to the error rates that were recorded for the various sampling methods, Random and Uncertainty were the least accurate, while QBag was the superior method. These findings are unique in that they are the first known active learning experiments in which a committee of decision trees was employed to select examples for training a single tree. One reason that a single tree might be desired instead of a tree committee is that the former tends to be easier to interpret, and therefore aids knowledge extraction.

On the semantic FinalStab measure, QBag ranked significantly higher than Random, on average, and also outperformed BootLV when the entropy criterion was used to grow trees. Similarly, on the semantic PrevStab measure, QBag and Uncertainty were both superior to Random, but only when

using entropy – no significant differences were observed for DKM. The structural stability scores of the sampling methods were more difficult to differentiate, as the Friedman significance test did not report any significant differences. However, a direct comparison between QBag and Random, the methods having the highest and lowest average rankings, respectively, did declare the former to be significantly more stable on the FinalStab metric when using the entropy criterion. By contrast, the differences between the RunStab scores recorded for the various methods were highly significant, as Random and Uncertainty were always the least stable of the methods. The QBag, QBoost, and QBoostC methods typically competed for the top 3 average rankings; however, it was QBoostC that demonstrated the best ability to grow structurally similar trees from distinct initial training samples.

Overall, while Random and Uncertainty were undoubtedly the inferior methods, due to their relatively high error rates and generally low structural stability, no single selective sampling procedure performed exceptionally better than the others. When the entropy criterion was used to grow trees, QBag was the most accurate method, and also scored highly on all the stability measures; thus, its use is recommended in conjunction with this splitting criterion. The results obtained when using the DKM criterion were less consistent across each of the stability metrics; however, since QBag performed on par with the other methods in terms of stability, and was clearly the most accurate method, it is deemed to be the method of choice for DKM as well. Nevertheless, QBoostC was a close runner-up when paired with DKM, as it was almost as accurate as QBag, and was more stable in many cases, particularly on the RunStab measure. It must be emphasized that these conclusions are based on average performance. No combination of sampling method and splitting criterion was able to achieve a consistent level of stability on all, or even the majority of the datasets that were tested – most ranked very high in some cases and very low in others.

The final conclusion drawn from these experiments is that, for most of the selective sampling methods that were studied, use of the DKM criterion yielded trees that were more accurate and structurally stable than those grown using entropy. These findings lend further credence to the experimental results presented in Section 2.6, which showed that DKM improved the stability of C4.5 in a passive learning environment.

Chapter 4

Region-based Active Learning

Each of the selective sampling methods studied in Chapter 3 employs one of two strategies when choosing the next batch of examples to label: direct selection or weight sampling. Direct selection (QBag, QBoost, QBoostC, Uncertainty) simply chooses the m examples having the highest scores, whereas weight sampling (BootLV, BootM) defines a probability distribution over the unlabelled examples and then randomly draws m examples, without replacement, according to this distribution.[1] Each selection scheme has its strengths and weaknesses. Direct selection ensures that the highest-scoring examples are labelled; however, if these are similar, then the batch may contain redundant examples, which would mean that labelling effort is wasted [25]. Weight sampling, on the other hand, encourages diversity among the examples in the batch; yet, the highest-scoring examples may be outweighed by a disproportionate number of low-scoring ones, such that a high-scoring example is unlikely to be drawn [56].

In this chapter, an alternative method is proposed for performing selective sampling with a committee of decision trees that is compatible with direct selection and weight sampling, but which serves to alleviate the downside associated with each of these. This approach is called *region-based* selective sampling, in contrast to the now familiar *example-based* class of selective sampling procedures. Instead of explicitly scoring and ranking the unlabelled examples in the pool, region-based selective sampling assigns scores to the decision regions that constitute the joint committee hypothesis. This hypothesis is represented by a so-called "merged" tree that is equivalent to the intersection of the tree classifiers in the committee. After mapping each unlabelled example to a particular decision region in the merged tree, either direct selection or weight sampling is performed on the scored regions, depending on the particular selective sampling method that is used, in order to choose a batch of examples to label.

The effect of this procedure on weight sampling is that the distribution of scores over the decision regions typically differs from the distribution of scores over the individual examples, even though a given example effectively receives the same score from both methods. In particular, the probability of selecting a high scoring example tends to be greater when using the region-based

[1] Descriptions of these methods, as well as their abbreviations, are contained in Section 3.2.

approach, sometimes resulting in a more rapid decrease in error rate as the training set grows. Direct selection is unaffected by this new selective sampling method, as it has been described thus far. However, an additional modification to the region-based approach does influence its choice of examples. A committee of decision trees can often form a very fine-grained partition of the instance space, such that the number of decision regions in the merged tree greatly outnumbers the sum of the regions defined by the individual committee members [45]. In order to limit the complexity of the joint committee hypothesis, a minimum decision region size is enforced. By preventing tiny decision regions from forming, the diversity of the examples in a given region is likely to increase, which can in turn reduce the amount of redundancy in a batch of examples chosen using direct selection. Finally, for either selection strategy, the amount of computation required to select a batch of examples may be reduced under certain circumstances by the region-based approach, relative to example-based selective sampling.

This chapter is organized as follows. In Section 4.1, the region-based approach to selective sampling with decision trees is described. An essential component of this method – an algorithm for combining decision trees – is introduced in Section 4.2. Next, in Section 4.3, a procedure is described for restricting the minimum size of the decision regions in the joint committee hypothesis (and therefore in the merged tree). Experiments are conducted in Section 4.4 that compare the performance of two example-based selective sampling methods to that of their region-based equivalents. In Section 4.6, an attempt to formulate the region-based selective sampling approach as a query learning method is described. Finally, in Section 4.7, conclusions are drawn.

4.1 Region-based Selective Sampling with Decision Trees

The region-based approach to selective sampling with tree committees is best illustrated by first describing the approach in terms of a sampling method that uses a single classifier, such as uncertainty sampling (Uncertainty). Recall from Section 3.2 that Uncertainty induces a single decision tree T_i during each iteration i of active learning. The score that Uncertainty assigns to an example $\mathbf{x}$ in the unlabelled pool U_i is $|t - \max(p, 1-p)|$, where t is the decision threshold (e.g. 0.5), and p is the probability of a positive example (i.e. in the leaf in T_i that classifies $\mathbf{x}$). After performing this calculation for each $\mathbf{x} \in U_i$, Uncertainty requests labels of the m lowest-scoring examples, as these are the examples for which p is closest to the decision threshold, and whose predictions are therefore the most uncertain.

Instead of assigning an effectiveness score to each example $\mathbf{x} \in U_i$, the region-based version of Uncertainty assigns a score to each decision region $r \in R_i$, where R_i is the set of mutually exclusive decision regions defined by T_i. The same scoring function is used; that is, the class distribution of the examples in r determines its score. Next, the unlabelled examples are classified by T_i; in doing so, each decision region r maintains a list of the examples that it classifies. Finally, the regions are sorted according to their scores, and the sorted list of regions is traversed in ascending order until

m examples have been encountered. These m examples are subsequently presented to the expert or oracle to be labelled. Note that the examples selected by the region-based approach may not be the very same examples that are chosen by Uncertainty. If one or more examples receive the same score, then sorting the examples explicitly may produce a different ordering than that which results from sorting the regions. However, it is always the case that the scores of the m examples selected by both methods are identical.

The computational cost of selecting examples can be reduced in situations where the number of unlabelled examples $|U_i|$ is greater than the number of decision regions $|R_i|$ defined by T_i. If the average cost of classifying an arbitrary example using T_i is K, then the cost of selecting the examples to be labelled is $K\,|U_i| + O(|U_i|\log(|m|))$. The latter term accounts for selecting the m best examples by score; also, the computation performed by the scoring function is assumed to incur negligible cost. Since, by definition of the scoring function, every example that is classified by a particular decision region necessarily receives the same score, mapping examples to scored decision regions and then sorting these regions has the effect of ranking the examples. The cost of the region-based selection procedure is $K\,|U_i| + O(|R_i|\log(|m|))$. Thus, if the number of decision regions is less than the number of unlabelled examples, the complexity of the example-selection procedure is reduced by a constant factor. Generalization of this cost analysis to a committee-based procedure is straightforward, with the only difference being that each example is classified c times instead of just once (c is the number of committee members). It should be pointed out, however, that the cost of constructing the tree or committee of trees tends to dominate the cost of actually selecting examples to label; hence, reductions of this sort are likely to have little impact on the overall complexity of selective sampling. The discussion here serves mainly to illustrate the region-based approach. Some empirical results regarding execution time and complexity are presented in Section 4.5.2.

The chief focus here is not on reducing the computational cost of selective sampling, but on improving the predictive accuracy, and possibly the stability, of the classifiers that are produced during selective sampling. Since committee-based selective sampling methods were shown in Chapter 3 to be superior, in terms of classification accuracy, to Uncertainty (see also [16, 47]), the remaining sections in this chapter study the application of the region-based active learning approach to committee-based methods.

4.2 Merging Decision Trees

In order for the region-based approach to be applicable to committee-based selective sampling methods, the committee of c decision trees induced at iteration i must first be represented as a set of mutually exclusive decision regions R_i. Once this is accomplished and scores are assigned to the regions, each unlabelled example $\mathbf{x} \in U_i$ is mapped to a particular decision region $r \in R_i$, the regions are sorted by score, and m examples are selected from the region(s) having the highest score(s). The

key issue, then, is how to efficiently convert the c decision trees into a set of decision regions, as is desired. The solution adopted here is to merge the c trees into a single tree, whose leaf nodes are the decision regions.

A procedure for merging a decision tree ensemble into a single tree classifier was first proposed by Quinlan [45]. Quinlan determined that a small boosted committee consisting of 3 decision trees that was formed by a novel method called *MiniBoosting* (a variant of AdaBoost [23]) achieved lower error rates than a single tree, on average. The committee members were merged in order to find out whether a single, interpretable decision tree could be constructed, which retained the improved accuracy of the voted committee. However, the merge operation tended to produce extremely large trees that were incomprehensible. Says Quinlan [45, p. 10]: "The partition of the instance space induced by voting even a small ensemble of classifiers is inherently complex. The effect on complexity of adding more classifiers to the set is multiplicative rather than additive."

The motivation for merging decision trees is quite different in this thesis. The merged tree is used for selective sampling, and is not meant to be interpreted. Thus, the complexity of the merged tree is a concern only with respect to computational resources that are required to compute and store it. The tree can be expensive to compute, and, as Quinlan noted, can have a large memory footprint. Any improvements in performance that are due to the region-based approach must therefore be weighed against these factors.

The merge algorithm formulated by Quinlan involves "concatenating" the set of trees and then executing a series of simplification steps on the resulting tree, in order to remove any redundant structure that may have been introduced. In this section, an algorithm for merging a pair of decision trees is presented, which is called *TreeMerge*. TreeMerge is functionally equivalent to the procedure described by Quinlan; however, TreeMerge is formulated such that redundant nodes are never introduced during the merge process, which eliminates the need for explicit simplification afterward. Since the product of combining two trees using TreeMerge is also a decision tree, n trees can be merged by running the algorithm $n-1$ times, each time using the output of the i^{th} merge as one of the two inputs for the $(i+1)^{st}$ merge.

4.2.1 The TreeMerge Algorithm

A decision tree T classifies an example $\mathbf{x}$ by recursively subjecting $\mathbf{x}$ to the test performed by the current decision node and then following the appropriate branch; this process begins at the root node of T, and terminates when a leaf is encountered, at which point the class predicted for $\mathbf{x}$ is the label associated with this leaf. In a similar manner, a decision region r can be "classified" by T. A decision region differs from an example (or attribute vector) in that the former may specify a range of values for a continuous attribute, whereas the latter specifies a particular value for any given attribute; furthermore, a decision region need not constrain the value of a given attribute at

Algorithm 9 TreeMerge

Require: Decision trees T_i and T_j

1: Initialize $R_j \Leftarrow$ set of decision regions defined by T_j
2: **for all** $r \in R_j$ **do**
3: AddRegionToTree(r, T_i)
4: **end for**
5: **return** T_i

all.[2] Setting aside these details for the time being, one reason for propagating a decision region r through a decision tree T is to determine whether the two entities agree on the class that is predicted within the region r in the instance space. Also, r can be "added" to T by incorporating the tests that are specified by r into T (if they are not already present), and somehow combining the class label predicted by r with the label(s) predicted by T in that same region. This "add" operation, which is called *AddRegionToTree,* is the essence of the TreeMerge algorithm, the details of which are now described.

Given two trees T_i and T_j, TreeMerge(T_i, T_j) is executed by adding each decision region in T_j to the tree T_i. Pseudo-code for the TreeMerge and AddRegionToTree procedures are displayed in Algorithms 9 and 10, respectively. The line numbers cited throughout the following description all refer to AddRegionToTree, unless otherwise noted. Let R_j be the set of decision regions defined by T_j; each region $r \in R_j$ is added to T_i as follows. Beginning at the root node of T_i, which specifies a test on attribute a, perform the test at the current node based on the value or range of values that r "covers" along dimension a (the possibility that T_i is a leaf node is covered below). If a is a discrete attribute, then r either specifies a single value v_a for a, or else it leaves a unconstrained. If r specifies a value v_a for a, then r is sent along the branch corresponding to v_a. Otherwise, for each of the n legal values $v_1, \ldots, v_n$ that exist for a, r is modified to constrain the value of a to v_i and is then sent along branch v_i (lines 19-22). On the other hand, if a is a continuous attribute, then r either specifies a range of values (in the form of lower and upper bounds) for a, or it leaves a unconstrained. Let $v_a^{\min}$ and $v_a^{\max}$ be the lower and upper bounds, respectively, that r sets for the continuous attribute a, and let t_a be the threshold specified by the test at the current decision node in T_i. If $v_a^{\max}$ is less than or equal to t_a, then r is sent along the left branch; otherwise, if $v_a^{\min}$ is greater than t_a, then r is sent along the right branch. In the event that t_a lies between the lower and upper bounds set by r along dimensions a (i.e. $v_a^{\min} \leq t_a < v_a^{\max}$), then r is first modified to constrain the value of a to the range $[v_a^{\min}, t_a]$ and is sent along the left branch. Next, r is modified to constrain the value of a to $(t_a, v_a^{\max}]$ and is sent along the right branch (lines 26-37).

This classification process repeats recursively until the region r (or a fraction thereof) reaches a leaf node in T_i. This leaf node is also a decision region – denote this region as l. At this point, one of two possibilities exist: either 1) r and l are equal, in that they each specify the same values

[2]This is also true of an attribute vector that is missing values for one or more attributes. However, it is assumed throughout this research that all attribute values are known.

Algorithm 10 AddRegionToTree

Require: Decision tree T, decision region r

if T is a leaf node **then**
  Let l denote this leaf node
  if r is equal to l **then**
    Compute the intermediate score for l
    return
  else {r is more specific than l}
    Let S be the set of tests performed by r that are not contained in l
    Replace l by a subtree T' which performs each test in S, and appends a new leaf node labelled with the class predicted by l to any branch that is not satisfied by r
    Connect r to the remaining unterminated branch, and compute its intermediate score
    return
  end if
else {T is a decision node}
  Let a be the attribute tested at the root of T
  if a is discrete **then**
    Let T_v be the subtree of T connected to branch v
    if r specifies a value v for a **then**
      AddRegionToTree(r, T_v)
    else
      Let $v_1, \ldots, v_n$ be the n legal values for a
      for $i = 1$ to n **do**
        Set $r' \Leftarrow r$ modified to constrain the value of a to v_i
        AddRegionToTree(r', T_{v_i})
      end for
    end if
  else {a is continuous}
    Let t_a be the threshold defined by T
    Let $[v_a^{\min}, v_a^{\max}]$ be the range of values covered by r along dimension a
    Let T_L and T_R be the right and left subtrees of T, respectively
    if $v_a^{\max} \leq t_a$ **then**
      AddRegionToTree(r, T_L)
    else if $v_a^{\min} > t_a$ **then**
      AddRegionToTree(r, T_R)
    else {$v_a^{\min} \leq t_a < v_a^{\max}$}
      Set $r' \Leftarrow r$ modified to constrain the value of a to $[v_a^{\min}, t_a]$
      AddRegionToTree(r', T_L)
      Set $r'' \Leftarrow r$ modified to constrain the value of a to $(t_a, v_a^{\max}]$
      AddRegionToTree(r'', T_R)
    end if
  end if
end if

(or ranges of values) for each attribute, or 2) r is more specific than l; that is, r performs one or more tests that are not performed by l. If r and l are equal, then their class labels are combined and AddRegionToTree returns (the method used to combine the labels is described below). However, if r is more specific than l, then the tests in r that do not appear in l are added to T_i in place of l (lines 7-9). Let S denote this set of tests. The leaf node l is removed from T_i and is replaced by a decision node that performs the first test in $s \in S$. Regardless of the type of attribute that s tests, exactly one outcome (branch) will be satisfied by r. Each of the remaining branches are connected to new leaf nodes that predict the class label specified by l. The branch that is satisfied by r connects to yet another decision node that performs the next test in S, and so on. This process continues until all the tests in S have been performed, at which point r is connected to the remaining unterminated branch. The label for the new leaf r is set to be combination of the class labels predicted by r and l, and AddRegionToTree returns.

Thus, the result of adding a region r to tree T is either that an existing leaf l in T has its class label combined with that of r, or that l is replaced by a new subtree that contains exactly one leaf that combines the predictions of r and l; the remaining leaves in this subtree predict the same class as does l. In either case, the region (or leaf) in T that contains the new combined label is referred to as the *region of overlap* between r and T.

The one detail that has been omitted from the description of AddRegionToTree is the mechanism by which the class label predicted by a region r is combined with the class label of a leaf in the destination tree T. Actually, it is not the leaf's class label that is updated, but its score. To reiterate, the purpose of TreeMerge is to combine a committee of decision tree classifiers into a single tree T', such that T' replicates the joint committee hypothesis. The score assigned to each decision region r in T_i (the tree representing the i^{th} committee member) is assumed to be a function of the class distribution of the training examples in r; this is referred to as the *basic score* for r. Note that the function used to compute the basic scores is independent from the function employed by C4.5 for classification, and may differ depending on the particular committee-building method that is employed (e.g. bagging, boosting). Prior to the execution of TreeMerge(T_i, T_j), the basic scores for the decision regions in the two trees are calculated and stored. Then, each time a region r from T_j is added to T_i, the region of overlap r_o between r and T_i is assigned an *intermediate score*; this score is computed using the basic score from r and from the leaf in T_i that overlaps with r (line 9). The intermediate score assigned to r_o must maintain sufficient information for the *combined score* of r_o to be calculated after all c trees have been merged together (i.e. after TreeMerge has been called $c - 1$ times). The combined score is the score that is assigned to a region by the entire committee.

For example, in query-by-bagging (QBag), the basic score $b_i(r)$ assigned to region r by the i^{th} committee member is a unit vote in $\{-1, 1\}$ whose sign indicates the predicted class. If there are p_r positive training examples and n_r negative examples in r, then $b_i(r) = 1 \cdot \text{sign}(p_r - n_r)$. The

intermediate score in this case is calculated as the sum of the pair of scores being combined. This is because the combined score for QBag is computed as the margin, or the difference between the number of votes assigned to each class; hence, the absolute value of the intermediate score for r after all c trees have been merged yields the desired combined score, which equals $|\sum_{i=1}^{c} b_i(r)|$.

If the combined score cannot be calculated incrementally for a particular committee-based method, then a generic intermediate score for r_o may simply be a list of the n basic scores that have been combined in r_o after n trees have been merged together. Then, after all c trees have been merged, the desired combined score is computed from these values. The main point being made here is that the combined score for a given region is not computed for each pair of trees that is merged, but is calculated from the region's intermediate score after all the trees in the committee have been merged.

4.2.2 Summary

To summarize, the TreeMerge procedure takes two decision trees as input, namely, T_i and T_j, and produces a single tree whose decision regions each contain an intermediate score. The intermediate score for a region r is calculated by combining the basic scores that are assigned to r by the two trees; the calculation performed when combining the basic scores, as well as the basic scores themselves, is determined by the particular committee-based sampling method that is used. By running TreeMerge $c - 1$ times, a committee of c decision trees can be combined into a single tree T'. The combined score for each region in T' is then computed from the region's intermediate score. As a result, the tree T' represents a hypothesis that is identical to the hypothesis that is formed by voting the trees in the committee; that is, T' assigns the same score to an arbitrary unlabelled example $\mathbf{x}$ that the committee assigns to $\mathbf{x}$.

4.3 Simplifying the Joint Committee Hypothesis

In his experiments with MiniBoosting, Quinlan [45] observed that committees consisting of just 3 C4.5 decision trees tended to represent very complex hypotheses; in fact, the merged tree computed from such a committee was sometimes too large to fit into the available memory. Preliminary experiments conducted here using the TreeMerge algorithm confirmed this trend for committees of size 10 (see Section 4.5.2 for details), which is the number of committee members used throughout this thesis. If the region-based selective sampling approach is to be useful in practice for a wide variety of datasets, there is clearly a need to limit the complexity of the hypothesis formed by the committee, in order to prevent the size of the merged tree from exploding. However, the goal of reducing the size of the merged tree does not conflict with the objective of making efficient use of unlabelled examples using the region-based approach; in fact, it is argued here that the two can be complementary.

In this section, a pre-processing step to the TreeMerge algorithm is proposed that institutes mandatory minimum size requirements for the decision regions contained in the joint committee

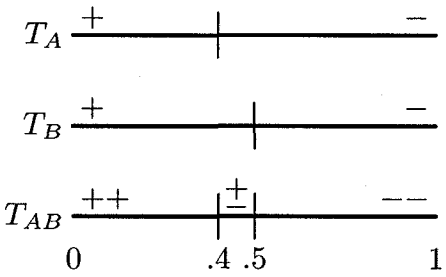

Figure 4.1: Two decision trees, T_A and T_B, which each perform a single split on the same continuous attribute (indicated by a vertical line), are merged to form the tree T_{AB}. The + and − signs represent votes for the positive and negative classes, respectively.

hypothesis. More specifically, no decision region is permitted to form whose width along a continuous dimension a is less than some predetermined amount. The method that is used to prevent such regions from occurring is formulated so as to reduce the complexity of the merged tree without significantly altering the hypothesis that it would otherwise define. The benefits of this enhancement to the region-based selective sampling approach include a reduction in the computational cost and space requirements of TreeMerge, as well as potentially improved example selection relative to existing example-based methods.

4.3.1 Issues with Tiny Decision Regions

When two or more decision trees are combined via the TreeMerge procedure, tiny decision regions may be formed by thresholds on a given continuous attribute that differ slightly between the two trees. Consider the example depicted in Figure 4.1. In this illustration, two trees T_A and T_B each define a single threshold on the same continuous attribute a. T_A places the threshold at 0.4, whereas T_B places it at 0.5; both trees predict the positive class $(+)$ for values of a that are less than or equal to their respective thresholds, and predict the negative class $(-)$ otherwise. If these two trees are merged by the TreeMerge procedure, the result is T_{AB}, which is shown at the bottom of the figure. Since the thresholds in the two trees are not equal, T_{AB} contains both thresholds, thereby producing three decision regions. The region $[0, 0.4]$ receives two votes for the positive class, region $(0.5, 1]$ gets two votes for the negative class, and the region $(0.4, 0.5]$ receives one vote for each class. Now, suppose that T_A and T_B are trees in a two-member bagging committee, and that the QBag method is being used to choose the next example to be labelled. Since the vote is split in the middle interval (i.e. the margin is zero), the label for an example in the range $(0.4, 0.5]$ will be requested. This choice might seem appropriate in this situation, even though only a small area of the committee's hypothesis will be benefit from this knowledge. However, suppose instead that the original thresholds were 0.499 and 0.5 – would labelling an example in such a tiny region be worthwhile?

The QBag method makes no distinction between the dimensions of the decision regions de-

fined by the committee – it simply looks for disagreement among the members. Consequently, this method is prone to selecting examples that reside in tiny regions, even though these may do little to improve the hypothesis, possibly resulting in wasted labelling effort. With respect to the TreeMerge algorithm, small discrepancies between the thresholds specified by trees in the committee lead to complex merged trees; such trees require extra computation to build, and their decision regions take longer to rank during the example selection phase. If, in fact, the tiny decision regions formed by the committee are not beneficial for selective sampling, then it is desirable to eliminate them from consideration. One way to accomplish this is as follows: During the TreeMerge procedure, if the overlap of two decision regions along dimension a is non-zero, but less than some preset width, then the resulting decision region is discarded, rather than being added to the merged tree. The problem with this approach, however, is that if a large portion of the unlabelled examples reside in tiny regions, then the selective sampling method will be unaware of a significant fraction of the available candidates. Instead, the procedure described in the following section "aligns" nearby thresholds that are defined by the committee trees, so that when the trees are merged, no tiny regions are formed; by this method, no areas of the space are overlooked during selective sampling.

4.3.2 Threshold Alignment

In this section, a procedure is described for "aligning" a collection of n thresholds defined on a continuous attribute a, namely, $t_a^1, \cdots, t_a^n$, which are within some distance λ_a of one another. This procedure is referred to as λ-*Alignment*. The parameter λ_a is the minimum width along dimension a that is permitted for any decision region that is contained in the merged tree. Analogously to the ε_{pct} parameter defined in Section 2.4, which was used in the calculation of the region stability metric, λ_{pct} is introduced as a single parameter, from which the minimum region width for any continuous attribute a in a particular dataset can be determined. That is, $\lambda_a = \frac{\lambda_{pct}}{100} \cdot [\max(a) - \min(a)]$, where $\min(a)$ and $\max(a)$ are the minimum and maximum legal values for a, and are either specified explicitly or are derived from the entire dataset. By aligning two or more thresholds that are very close to one another, slight discrepancies between them are eliminated, which prevents tiny decision regions from being created in the merged tree. Any unlabelled examples that would otherwise have resided in the small gap(s) between the thresholds are instead mapped to larger decision regions, and are therefore still considered by the selective sampling method; this would not be the case if the tiny regions were simply discarded, for example.

A collection of thresholds $t_a^1, \cdots, t_a^n$ that lie in a range of width less than λ_a, are aligned by setting the value of each t_a^i to the mean of these thresholds, or $\frac{1}{n}\sum_{i=1}^{n} t_a^i$. For instance, in the example depicted in Figure 4.1, if λ_a were set to 0.2 units, then the thresholds defined by T_A and T_B along the attribute a, which differ by just 0.1 units, would be aligned prior to running TreeMerge. After the alignment, T_A and T_B would each contain a threshold at $\frac{0.4+0.5}{2} = 0.45$, and the resulting merged tree T_{AB} would contain only this threshold. In essence, the positive decision region in T_A is

widened by 0.05 units, the positive region in T_B is narrowed by 0.05 units, and the negative regions in each tree are adjusted accordingly so that the entire range is covered. Then, when T_A and T_B are merged together, the votes are simply tallied, as any pair of regions either overlap completely or are disjoint (i.e. no new regions are formed). In general, a given pair of decision regions in the trees of two different committee members may partially overlap after thresholds have been aligned; however, it is ensured that the intersection of these regions has a width of at least λ_a along each continuous dimension a.

Next, the details of the λ-Alignment procedure are described. Pseudo-code for the method is displayed in Algorithm 11, and the line numbers referenced in the following description are lines in this code. Given a committee of c decision trees, induced from a dataset that contains at least one continuous attribute, the continuous thresholds are extracted from each tree, producing a single bag[3] of thresholds for each continuous attribute a (lines 4-7). For example, if the threshold 0.5 appears in two trees for attribute a, then among the elements in the bag for a are two thresholds having the value 0.5. The next step is to determine, for each attribute, the subsets of these thresholds that are to be aligned, or averaged together. Rather than simply scanning the thresholds defined on a in ascending or descending order, and grouping together those that are within λ_a of one another, a complete linkage hierarchical clustering algorithm [27] is employed, so that the order in which thresholds are considered does not affect the choice of subsets that are to be aligned. A cluster is defined here as a bag of thresholds; initially, for attribute a, a cluster of size 1 is created for each threshold that was extracted from the trees in the committee (lines 12-14). Let this bag of clusters be denoted by K_a. A list of cluster pairs, namely, $Pairs(K_a)$ is maintained, which records the distance between each pair of clusters $k_a^i, k_a^j \in K_a, i \neq j$, using a distance measure $d : (k_a^i, k_a^j) \rightarrow \mathbb{R}$. In complete linkage clustering, the distance between two clusters k_a^i and k_a^j is calculated as the maximum pairwise distance between the (threshold) values contained in k_a^i and those contained in k_a^j. For example, if $k_a^i = \{0.5, 1, 2\}$ and $k_a^j = \{-1, 4\}$, then:

$$\begin{aligned} d(k_a^i, k_a^j) &= \max(|-1-0.5|, |4-0.5|, |-1-1|, |4-1|, |-1-2|, |4-2|) \\ &= \max(1.5, 3.5, 2, 3, 3, 2) \\ &= 3.5 \end{aligned}$$

When initially forming or later updating the list of cluster pairs, the pair (k_a^i, k_a^j) is added to $Pairs(K_a)$ only if $d(k_a^i, k_a^j) < \lambda_a$ (lines 15-21). Next, the nearest pair of clusters (k_a^i, k_a^j) that are separated by a distance of less than λ_a are combined (ties are broken arbitrarily; lines 23-30). Specifically, clusters k_a^i and k_a^j are combined by replacing removing them from K_a and replacing them with a single cluster k_a^{ij} that contains the join of the elements k_a^i and k_a^j (i.e. $k_a^{ij} = k_a^i \uplus k_a^j$). After updating the list of cluster pairs to include k_a^{ij}, the current closest pair of clusters are once again combined, and so on.

[3] A bag, or multiset, is a set in which duplicate elements are permitted.

The process terminates when every pair of clusters is separated by a distance of at least λ_a (line 23). At this point, the mean $m(k_a^i)$ of the thresholds in each cluster k_a^i is computed, and every threshold in k_a^i is set to the value of $m(k_a^i)$ (lines 31-36). The reason that multiple copies of the same threshold are permitted during the λ-Alignment procedure is that multiple thresholds having the same value t_a provide stronger evidence that a true division occurs at t_a in the target concept; hence, the more instances of t_a that appear in a given cluster, the greater is the influence of t_a on the value of the mean that is computed over the thresholds in the subset.

By setting the thresholds contained in each cluster to their mean value, it is possible for a distinct pair of the aligned thresholds to be within λ_a of one another, even though every pair of clusters was originally separated by a distance of more than λ_a, according to the metric d. For example, consider the clusters $k_a^i = \{53, 53, 53, 54\}$ and $k_a^j = \{42, 45\}$, where $\lambda_a = 10$. The maximum pairwise distance between k_a^i and k_a^j is $|54 - 42| = 12$; however, the distance between the means of these clusters is $|53.25 - 43.5| = 9.75$. Due to this possibility, after the thresholds have been modified, a single pass is made through the sorted list of aligned thresholds, and if two consecutive thresholds differ by less than λ_a, the greater of the two is increased so that they differ by exactly λ_a (lines 37-45).

Whenever a particular threshold is modified during λ-Alignment, the actual decision node in the tree T_i that specifies this threshold is updated to reflect the change (line 46). However, modifying thresholds in a decision tree in this manner is likely to cause the distribution of training examples in one or more of its leaves to change – a training example whose value v_a for attribute a was less than or equal to the original threshold t_a will be classified differently if v_a is greater than the modified threshold t'_a, and vice versa. Since the correct class distributions are needed to compute basic scores for the decision regions in each committee member T_i, the examples that were used to train T_i are classified by the modified tree T'_i, and the class frequencies at the leaves of T'_i are updated accordingly. This step is performed only once, after the λ-Alignment procedure has processed all the continuous attributes. Finally, TreeMerge is called $c - 1$ times to compute the merged tree representing the modified committee. Note that no changes are made to the TreeMerge algorithm; λ-Alignment ensures that no decision region in the merged tree are of length less than λ_a along any continuous dimension a.

There is one additional modification to the committee-building procedure that has not yet been aired. Since λ_a is the minimum allowable width along dimension a for any decision region in the merged tree, a decision region contained in a committee member that is smaller than λ_a will necessarily have its thresholds along dimension a modified during λ-Alignment. In fact, such a region could have its lower and upper bounds along a averaged together, thereby "collapsing" the region into an interval having zero width along dimension a. To avoid this complication, the following pre-pruning rule is applied when inducing committee member T_i: If the best split s^* at a given node nominates a continuous attribute a, and the addition of a decision node to T_i that performs the test

Algorithm 11 LambdaAlign

Require: Set of c decision trees $T_1, \ldots, T_c$; minimum decision region size λ_{pct}

1: Let A_{cont} denote the set of continuous attributes
2: **for all** attributes $a \in A_{cont}$ **do**
3: Let $\lambda_a = \frac{\lambda_{pct}}{100} \cdot [\max(a) - \min(a)]$
4: Let B_a be the bag (i.e. multiset) of thresholds for attribute a
5: **for** $i = 1$ to c **do**
6: Extract the continuous thresholds from tree T_i and add them to B_a
7: **end for**
8: **if** B_a is empty **then**
9: Proceed to the next attribute a
10: **end if**
11: Let K_a be the bag of clusters for attribute a
12: **for all** thresholds $t_a \in B_a$ **do**
13: Create a cluster k_a containing a single threshold t_a, and add k_a to K_a
14: **end for**
15: Let $Pairs(K_a)$ be a list of cluster pairs and the distances between them
16: **for all** pairs of clusters $(k_a^i, k_a^j), i \neq j$ **do**
17: Compute the distance d between k_a^i and k_a^j
18: **if** $d < \lambda_a$ **then**
19: Add the pair of clusters (k_a^i, k_a^j) to $Pairs(K_a)$ with distance d
20: **end if**
21: **end for**
22: Sort $Pairs(K_a)$ in order of increasing distance
23: **while** $Pairs(K_a)$ not empty AND distance between closest pair $> \lambda_a$ **do**
24: Let (k_a^i, k_a^j) be the closest pair in $Pairs(K_a)$; if two or more pairs have the same lowest distance, select one of these pairs at random
25: Let $k_a^{ij} = k_a^i \uplus k_a^j$
26: Remove k_a^i and k_a^j from K_a; add k_a^{ij} to K_a
27: Remove from $Pairs(K_a)$ any pair that contains k_a^i and/or k_a^j
28: Repeat lines 17-20 with each pair (k_a^{ij}, k_a), for $k_a \in K_a, k_a \neq k_a^{ij}$
29: Sort $Pairs(K_a)$ in order of increasing distance
30: **end while**
31: **for all** clusters $k_a \in K_a$ **do**
32: Let m be the mean of the thresholds contained in k_a
33: **for all** thresholds $t_a \in k_a$ **do**
34: Set $t_a \leftarrow m$
35: **end for**
36: **end for**
37: There are now n clusters $k_1, \ldots, k_n$; each contains thresholds having a single value $v(k_i)$
38: Sort the clusters in order of increasing value (i.e. k_1 has the lowest value)
39: **for** $i = 2$ to n **do**
40: **if** $\left|v(k_a^i) - v(k_a^{i-1})\right| < \lambda_a$ **then**
41: **for all** thresholds $t_a \in k_a^i$ **do**
42: Set $t_a \leftarrow v(k_a^{i-1}) + \lambda_a$
43: **end for**
44: **end if**
45: **end for**
46: Update the value of each threshold in the tree T_i from which it was extracted
47: **end for**

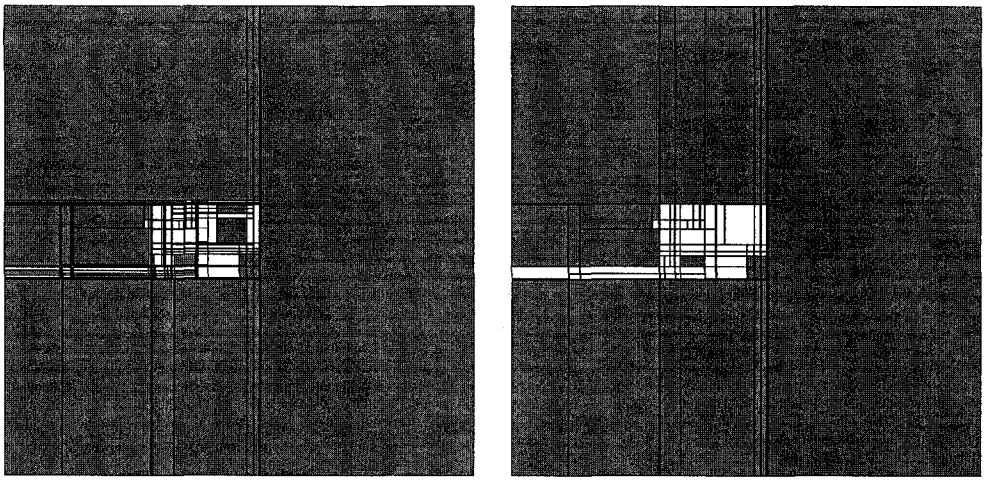

Figure 4.2: Merged tree representations of bagging committees created from the corner-kick dataset, with (right) and without (left) λ-Alignment.

s^* would lead to the creation of a region whose length along dimension a is less than λ_a, then s^* is not performed and a leaf node is added to T_i instead. In the presence of this pre-pruning rule, the tree is still post-pruned by C4.5 as usual (see Section 2.1).

To demonstrate the effect of performing λ-Alignment on a committee of decision trees, consider the two merged trees displayed in Figure 4.2. In each case, a bagging committee containing 10 trees was created using 2000 randomly chosen examples from the corner-kick dataset (see Appendix A.2). The only knowledge of the corner-kick dataset required for this particular example is that it consists of 2 continuous attributes; as such, the decision regions formed by the tree committee can be visualized on a two-dimensional plane. In Figure 4.2, the attributes have been normalized so that their ranges of values are the same. Each rectangle represents a decision region, and the shading indicates the score that it has been assigned by the committee; darker shades of grey indicate lower scores (i.e. larger margins for QBag), while brighter shades mean that the region is believed to contain informative examples. The left picture shows the merged tree that was constructed without first subjecting the committee to the λ-Alignment procedure. There are several large peripheral decision regions that are of believed to be uninformative. Near the middle of the two attribute ranges, however, the space is partitioned into a large number of high-scoring and low-scoring regions, some of which are very tiny. The right picture displays the merged tree that was built after performing λ-Alignment, using $\lambda_{pct} = 1\%$. This tree closely resembles the one grown in the absence of λ-Alignment, in that the areas of the space that are believed to be informative are essentially the same; however, it contains 274 fewer decision regions (193 compared to 467). It is clear that the scores assigned to some areas of the space changed as the result of performing λ-Alignment; yet, these changes were not substantial. The experiments conducted in Section 4.4 evaluate the effect of λ-Alignment on the examples that are chosen by the selective sampler.

4.4 Experimental Procedure

In order to determine whether the region-based selective sampling approach could improve performance over the traditional example-based approach, experiments were conducted using two methods from each genre. The two example-based methods were query-by-bagging (QBag) and bootstrap-M (BootM); descriptions of these can be found in Section 3.2. Region-based versions of these two procedures were formulated, named region-based query-by-bagging (QBag-rb-min) and region-based bootstrap-M (BootM-rb-min), respectively. The "min" suffix in these two names indicates that minimum decision region widths were enforced during the construction of the merged tree. Next, the two region-based selective sampling methods are described.

The QBag-rb-min method uses the same procedure as QBag to build a committee from the training set L_i at iteration i of selective sampling, except that the former employs the λ-Alignment procedure (including the pre-pruning rule) described in Section 4.3. Next, the committee members are combined into a single tree T_i' using the TreeMerge algorithm (see Section 4.2); the combined score for each decision region is computed as the margin, or the difference between number of positive and negative votes assigned to the region by the committee.[4] The unlabelled examples U_i are classified by T_i', thereby mapping each $\mathbf{x} \in U_i$ to a particular decision region in T_i'. Finally, m examples are selected from the decision regions that have the best combined scores.

The other region-based method, BootM-rb-min, builds a committee and subsequent merged tree from a given training set that are identical to those built by QBag-rb-min. The difference between these two methods is that BootM-rb-min chooses the next example to be labelled by choosing a decision region r in the merged tree at random, according to the distribution of scores assigned to the regions, and then randomly selecting an unlabelled example from r. The regions are sampled with replacement, although a particular example may be selected at most once, and therefore a region that contains no unselected examples has zero probability of being chosen. By contrast, QBag-rb-min repeatedly chooses an example from the region r that has the best score, and continues to do so until there are no more unselected examples in r, at which point it considers the second-highest scoring region, and so on.

The reason that this suite of sampling methods was chosen for these experiments is that the objective here was to identify the differences in performance between the example-based selective sampling approach and the region-based approach that employs λ-Alignment. QBag and BootM feature the two example-selection mechanisms that are used by the selective sampling methods described in Section 3.2, namely, direct selection and weight sampling, respectively. Query-by-boosting (QBoost) and confidence-based query-by-boosting (QBoostC) differ from QBag only in the procedures they use to construct a committee, and in the voting schemes that they employ. The weight sampling methods bootstrap-LV (BootLV) and BootM are identical except that the former's scoring function is based on the variance in the committee vote, whereas the latter assigns scores

[4]This is the same score that QBag computes for a given unlabelled example.

according to the margin. Moreover, since QBag also computes the margin, a comparison between QBag and BootM isolates the differences between direct selection and weight sampling, as does a comparison between QBag-rb-min and BootM-rb-min.

The same 15 datasets from the UCI repository that were used in the experiments performed in Section 3.3 were also used here, and the experimental procedure described in that same section was replicated.[5] In addition to the UCI datasets, two datasets that were generated from the FIFA 2004 commercial soccer game by Electronic Arts were also included. These datasets, which are named *corner-kick* and *give-and-go*, are described in Appendix A.2. The purpose of experimenting with these datasets was to compare the example-based and region-based approaches on datasets that were very large – these contained approximately 66,000 examples each. By comparison, the largest of the UCI datasets used here was letter, which had 20,000 instances. Due to their relatively large sizes, the experimental parameters used for the soccer datasets differed slightly from those used for the UCI datasets. Specifically, for both soccer datasets, the evaluation set consisted of 22,000 examples, and the batch size m was set to 100. The number of examples forming the initial training sets were 2000 for corner-kick and 5000 for give-and-go.[6] Finally, selective sampling was halted once the size of the training set reached 22,000 examples.

For the QBag-rb-min and BootM-rb-min methods, a minimum allowable decision region size was specified for each dataset, as is required for the λ-Alignment procedure. For a given dataset, this size was determined by first using $\lambda_{pct} = 5\%$; if 10 runs of selective sampling, each using different initial training samples, could not be completed at this setting (due to memory exhaustion), then the value of λ_{pct} was incremented by 5% until a λ_{pct} value was found for which all 10 runs were completed. The values that were arrived at by this procedure are listed in Table 4.1. No value for λ_{pct} was specified for datasets that contained discrete attributes only, as trees produced from these datasets are unaffected by this parameter. In these experiments, all trees were grown using the DKM splitting criterion.

4.5 Experimental Results

In this section, results pertaining to predictive performance, trends in example-selection, and computational cost are presented and discussed. As for the relative structural stability of the four methods, no method was consistently more stable than the others. For example, on both the FinalStab and PrevStab structural stability metrics, when using $\varepsilon_{pct} = 0$, the average ranks for the methods were all within a range of 0.47. In fact, the differences in these stability scores were not statistically significant at any values of ε_{pct} that were used; hence, no further analysis of stability is undertaken

[5]The results reported here for QBag and BootM are therefore identical to the figures reported in Section 3.4.

[6]The initial training set for give-and-go was made larger than that of corner-kick due to the higher dimensionality of the former. Unfortunately, the total quantity of data that could be generated using the game simulator was limited by computational and time constraints; for this reason, the size of the initial training set for give-and-go was increased linearly relative to that of corner-kick (rather than exponentially) as a function of the number of attributes in each dataset.

Dataset	$\lambda_{pct}(\%)$
anneal	5
australian	5
corner-kick	5
german	5
give-and-go	5
letter	10
pendigits	10
pima-indians	5
segmentation	5
vehicle	10
vowel	5
wdbc	10
yeast	5

Table 4.1: The minimum permitted decision region widths (λ_{pct}) that were used for the experiments described in Section 4.4. Datasets containing only discrete attributes are not listed here.

here.

4.5.1 Classification Accuracy

The weighted averages of the error rates for the four methods are displayed in Table 4.2. Looking first at the average ranks, QBag and QBag-rb-min tied with average ranks of 2.029 each, while the average rank for BootM-rb-min bettered that of BootM by exactly 1. The difference between the latter two methods was statistically significant at $\alpha = .10$, based on the Friedman and Nemenyi tests (see Section 3.3). In a head-to-head comparison, BootM recorded 4 wins, 11 losses, and 2 draws versus its region-based version, BootM-rb-min. On the other hand, the two bagging methods were very evenly matched, with 7 wins going to QBag, 6 to QBag-rb-min, and 4 datasets resulting in draws. Needless to say, the bagging methods were also significantly more accurate than BootM according to the same statistical tests. However, the difference between BootM-rb-min and the bagging methods was not statistically significant.

The results indicate that BootM-rb-min is superior to BootM, yet slightly less accurate than both QBag and QBag-rb-min, which tied for the best overall performance. Recall that QBag exhibited the lowest error rates among the 7 sampling methods that were studied in Section 3.4.1. Figure 4.3 plots the error rates of the four sampling methods on the segmentation dataset. In this case, BootM-rb-min outranked BootM, and also achieved lower error rates than both QBag and QBag-rb-min. Also, QBag-rb-min was more accurate than QBag on this dataset. That QBag-rb-min was competitive with QBag on most of the datasets suggests that performing λ-Alignment – pre-pruning each committee member's tree and averaging nearby decision thresholds – has little impact on the effectiveness of the examples that are selected. A complex hypothesis formed by the unmodified bagging committee did not necessarily lead to the selection of more informative examples than did a hypothesis defined by QBag-rb-min, which tends to be simpler.

Dataset	BootM	QBag	BootM-rb-min	QBag-rb-min
anneal	.125 (4)	.120 (2)	.123 (3)	.117 (1)
australian	.126 (2.5)	.126 (2.5)	.125 (1)	.131 (4)
car	.079 (3)	.077 (1)	.084 (4)	.078 (2)
corner-kick	.009 (2.5)	.009 (2.5)	.009 (2.5)	.009 (2.5)
german	.291 (2)	.287 (1)	.309 (4)	.292 (3)
give-and-go	.194 (4)	.193 (2)	.193 (2)	.193 (2)
kr-vs-kp	.009 (4)	.007 (1.5)	.007 (1.5)	.008 (3)
letter	.013 (4)	.012 (2)	.012 (2)	.012 (2)
nursery	.044 (3)	.040 (2)	.090 (4)	.039 (1)
pendigits	.013 (4)	.012 (3)	.011 (1.5)	.011 (1.5)
pima-indians	.282 (3.5)	.281 (2)	.282 (3.5)	.277 (1)
segmentation	.018 (3.5)	.018 (3.5)	.012 (1)	.014 (2)
tic-tac-toe	.207 (3)	.204 (1)	.223 (4)	.206 (2)
vehicle	.233 (4)	.227 (3)	.222 (1)	.224 (2)
vowel	.044 (4)	.037 (1.5)	.038 (3)	.037 (1.5)
wdbc	.065 (4)	.054 (1)	.056 (3)	.055 (2)
yeast	.256 (4)	.254 (3)	.252 (1)	.253 (2)
Avg. rank	(3.471)	(2.029)	(2.471)	(2.029)

Table 4.2: Weighted error rates.

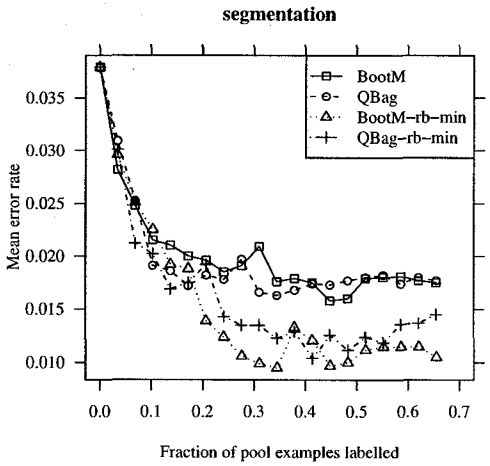

Figure 4.3: Mean error rates on the segmentation dataset.

The lower error rates achieved by BootM-rb-min compared to BootM are partially attributable to the fact that the former draws examples randomly from a distribution that is defined over the region scores, whereas the latter uses a distribution that is defined over the scores of the examples themselves. Xiao et. al [56] previously identified a weakness in BootLV that also applies to BootM, since both methods employ weight sampling. They observed that when there are relatively few examples that receive high scores, while most of the examples are assigned low scores, BootLV tends to select low-scoring examples. As an illustration, they provide the following example: "Suppose there are 100 unlabelled points, of which 4 have fairly high scores, e.g. x, and 96 have substantially lower scores, e.g. $x/10$. Because there is a 24:1 ratio of lows to highs, but their score ratio is only 1:10, a low-scoring point is 2.4 times more likely to be chosen than a high-scoring point each time bootstrap-LV draws a sample." [56, p. 6]

Compared to BootM, BootM-rb-min tended to produce distributions of scores that were less skewed, instead of being unduly biased toward the majority class. Consider the relative performance of these two methods on the vowel dataset, for example. The vowel dataset is rather imbalanced, with only 9.1% of the examples belonging to the minority class. Based on the analysis of weight sampling offered by Xiao and colleagues [56], this dataset is a likely candidate for BootM to perform poorly on, since it may not adequately represent the minority class in the training sets that it forms. Indeed, BootM's weighted average error rate on the vowel dataset was .006 higher than that of BootM-rb-min (see Figure 4.5). To gain some insight into the behaviour of the two methods on this dataset, a single iteration of selective sampling was performed using the initial training set from the first of the 10 runs. In particular, the initial training set consisted of 99 examples; BootM constructed a committee from this data and assigned a score to each of the 562 unlabelled instances in the pool, whereas BootM-rb-min built a merged tree and assigned a score to each decision region contained in the tree. In this particular merged tree, there were 112 decision regions, 32 of which classified at least one unlabelled example (empty decision regions are not included in the weight sampling procedure by BootM-rb-min). Figure 4.4 displays the total weight that was designated to scores that were greater than or equal to x, for values of x ranging from 0.1 to 0.95.[7] As predicted, BootM assigned relatively little weight to the highest-scoring examples – the examples that received scores of less than 0.2 had a combined weight of 0.904, while the total weight for scores of at least 0.9 was just 0.02. By contrast, BootM-rb-min was more likely to select a high-scoring region, as the scores that were less than 0.2 combined for a weight of only 0.375, while scores that were greater than or equal to 0.9 had a total weight of 0.062.

The example depicted in Figure 4.4 indicates that, during a given iteration of selective sampling, BootM-rb-min is more likely than BootM to choose an unlabelled example that has a reasonably high score. Note that although the two methods use the same scoring function, the score that each assigns to a particular example may differ due to the fact that the committee formed by BootM may

[7]No scores less than 0.10 were assigned by either method. There were 11 scores (9 for BootM and 2 for BootM-rb-min) whose values were greater than 1; these were counted in their respective "$\geq$ 0.95" bins.

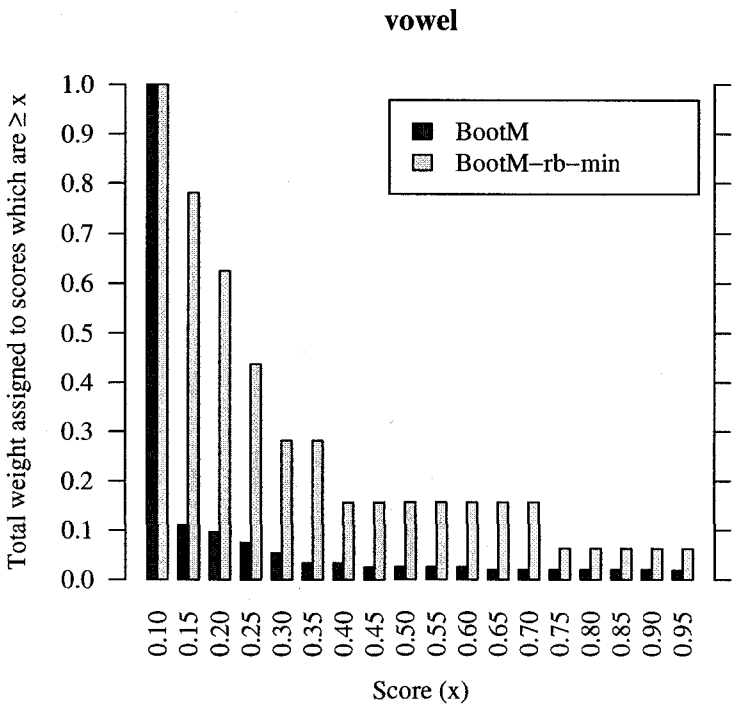

Figure 4.4: Comparison of score distributions defined by BootM and BootM-rb-min on the vowel dataset.

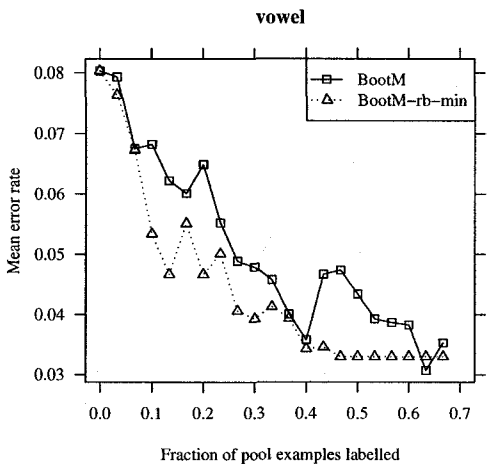

Figure 4.5: Mean error rates on the vowel dataset, for the BootM and BootM-rb-min methods.

constitute a different hypothesis than that of BootM-rb-min. However, since the scoring function is heuristic – it does not directly optimize any performance criterion – increasing the probability of choosing a high-scoring example does not necessarily lead to lower error rates. That is, an example that is believed to be informative may not actually improve the hypothesis once it is labelled and added to the training set.

BootM-rb-min achieved lower error rates, on average, than BootM; yet, another difference between the two methods was even more palpable: the former tended to label a greater number of positive examples (i.e. examples from the minority class) than did the latter. At each iteration, the fraction of examples in the training set belonging to the positive class was recorded. The (unweighted) averages of these values for each dataset are displayed in Table 4.3, for all four sampling methods. BootM-rb-min sought out more positive examples than BootM on 13 of the 17 datasets that were tested. The former method also labelled more positive examples, on average, than QBag; yet, QBag-rb-min incorporated more positive examples into the training set than any other method. Figure 4.6 plots the fraction of the examples in the training set that were positive as the number of labelled examples increased, for the vowel (left) and pima-indians (right) datasets. On the vowel dataset, QBag and both region-based methods requested the labels of many positive examples in the early stages of learning, whereas BootM maintained approximately the same ratio of positive to negative examples during most iterations. On the pima-indians dataset, BootM once again labelled the fewest positive examples; however, in this case there were clear differences between the other three methods, as both region-based methods selected a larger proportion of positive examples in the early rounds of learning than did QBag. In the end, the final training set produced by QBag-rb-min contained the most positive instances of any method.

Based on these observations, it is not solely the region-based weight sampling procedure employed by BootM-rb-min that causes it to choose a greater portion of positive examples than BootM; the λ-Alignment step performed by both BootM-rb-min and QBag-rb-min appears to be an important factor in this trend. One effect of restricting the minimum size of decision regions is that the probability of a creating a pure decision region – one that contains examples belonging to a single class – is reduced. That is, with a coarser partitioning of the space, a given region is more likely to classify a mixture of positive and negative examples, since with larger decision regions, the learner's ability to isolate small pockets of similarly labelled examples is hampered. Whereas pure regions are assigned low scores by the selective sampling method, impure regions are more likely to cause disagreement among the committee members, and are therefore more likely to be sampled. It follows that if more positive examples reside in regions where the classification is uncertain, the probability of labelling a positive example increases.

Dataset	BootM	QBag	BootM-rb-min	QBag-rb-min
anneal	.724 (1)	.680 (3)	.697 (2)	.661 (4)
australian	.491 (4)	.558 (1)	.507 (3)	.519 (2)
car	.339 (4)	.403 (3)	.480 (1)	.428 (2)
corner-kick	.040 (4)	.066 (3)	.073 (2)	.076 (1)
german	.348 (3)	.379 (2)	.248 (4)	.383 (1)
give-and-go	.278 (4)	.311 (2)	.282 (3)	.320 (1)
kr-vs-kp	.484 (4)	.493 (3)	.559 (1)	.504 (2)
letter	.052 (4)	.066 (3)	.071 (2)	.074 (1)
nursery	.357 (3)	.408 (2)	.149 (4)	.444 (1)
pendigits	.116 (4)	.162 (3)	.184 (1)	.182 (2)
pima-indians	.381 (4)	.408 (3)	.416 (2)	.434 (1)
segmentation	.161 (4)	.195 (3)	.208 (2)	.249 (1)
tic-tac-toe	.404 (3)	.426 (2)	.370 (4)	.441 (1)
vehicle	.319 (4)	.338 (2)	.322 (3)	.342 (1)
vowel	.131 (4)	.171 (2)	.170 (3)	.173 (1)
wdbc	.390 (3)	.388 (4)	.428 (2)	.433 (1)
yeast	.332 (4)	.369 (2)	.359 (3)	.372 (1)
Avg. rank	(3.588)	(2.529)	(2.471)	(1.412)

Table 4.3: Average fraction of examples in the training set that belong to the positive class (i.e. the minority class). These are unweighted averages, computed over all iterations of selective sampling.

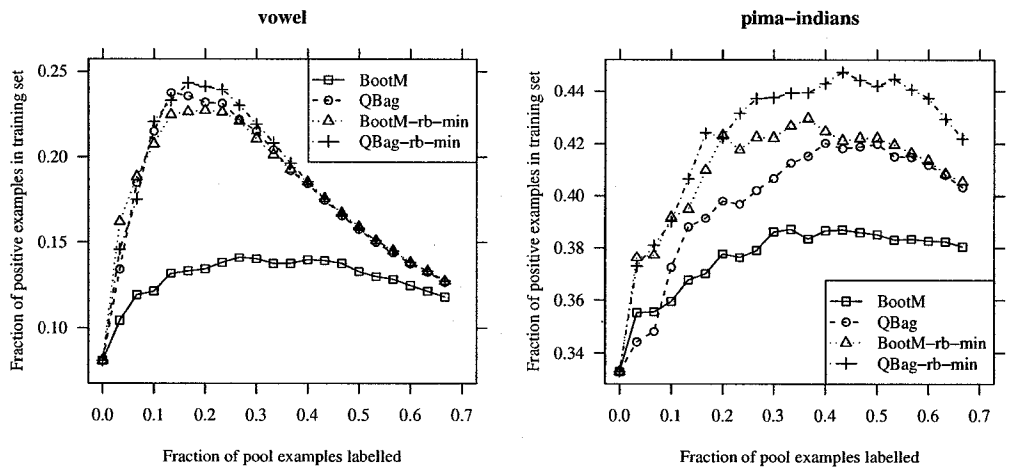

Figure 4.6: Average fraction of examples in the training set belonging to the positive class (i.e the minority class), on the vowel (left) and pima-indians (right) datasets. Note that the scale of the y-axis differs between the two plots.

4.5.2 Execution Time and Hypothesis Complexity

In addition to monitoring performance criteria such as error rate and structural stability, the execution times and committee sizes were recorded for QBag and QBag-rb-min.[8] The weight sampling methods, BootM and BootM-rb-min, are not examined here, since the differences between these methods and the bagging methods are trivial in this regard. Two aspects of the selective sampling procedure were timed for each method: the committee building procedure and the example selection procedure. Also, the number of decision regions in the committee and in the merged tree (in the case of QBag-rb-min) were recorded. For QBag, the committee building procedure consists of creating 10 bootstrap samples from the available training data and inducing a decision tree from each of these samples. The example selection procedure involves having the committee assign a score to each unlabelled example (in doing so, each example is classified by each committee member), and then sorting the examples by their scores. As was described in Section 4.4, QBag-rb-min builds a committee using the same procedure as QBag, except that λ-Alignment is performed by the former; in addition, the committee building time for QBag-rb-min includes the time taken to construct the merged tree. The example selection procedure for QBag-rb-min consists of having the merged tree classify the unlabelled examples, and then sorting the decision regions defined by this tree according to their scores.

To gain additional insight into the reduction in the complexity of the committee that is caused by the λ-Alignment procedure, these same statistics were also recorded for a method called QBag-rb, which is identical to QBag-rb-min except that the former does not perform pre-pruning or threshold averaging. In other words, QBag-rb is equal to QBag-rb-min with $\lambda_{pct} = 0$. The QBag-rb method is of little practical value, since it constructs a merged tree that represents the same hypothesis as that of a bagging committee induced from a given data sample, and therefore the examples selected by QBag-rb and QBag are guaranteed to have the same scores. Thus, the additional computation required to build the merged tree is unjustifiable. Nevertheless, it is useful to study the behaviour of QBag-rb in order to better understand the workings of QBag-rb-min.

Table 4.4 lists the two components of the execution times for the three methods, while Table 4.5 displays the average number of decision regions in the committees and merged trees. There is no column indicating the number of decision regions for the QBag-rb committees in Table 4.5, since the figures were virtually identical to those of QBag.[9] The merged trees constructed by QBag-rb tended to dwarf those created by QBag-rb-min; in fact, on 7 of the 17 datasets, QBag-rb could not complete 10 runs of selective sampling, as the merged trees it formed could not always fit into the available memory. Nevertheless, the potential efficiency gained by ranking decision regions rather

[8] These experiments were performed on a machine that possessed a 2400 megahertz dual core processor and 8 gigabytes of memory.

[9] Recall that, given a particular training set, QBag and QBag-rb build identical committees. However, it is not guaranteed that the two methods will selected exactly the same examples, due to ordering effects when ranking examples versus ranking regions. Note also that QBag-rb and QBag-rb-min construct identical committees from datasets that contain no continuous attributes.

Dataset	Time to build committee (ms)			Time to select examples (ms)		
	QBag	QBag-rb	QBag-rb-min	QBag	QBag-rb	QBag-rb-min
anneal	11	27	8	0	3	0
australian	9	270	9	0	82	0
car	8	12	12	1	0	0
corner-kick	533	692	423	59	10	6
german	17	3,343	2,129	1	1,417	836
give-and-go	3,233	–	1,265	139	–	30
kr-vs-kp	38	51	51	4	1	1
letter	628	–	578	19	–	37
nursery	87	122	123	12	2	2
pendigits	289	–	1,026	9	–	199
pima-indians	19	–	128	0	–	30
segmentation	89	–	131	2	–	8
tic-tac-toe	6	41	41	1	7	7
vehicle	40	–	253	1	–	65
vowel	18	2,298	201	0	929	50
wdbc	26	–	61	0	–	6
yeast	33	2,801	68	1	1,119	10

Table 4.4: Average execution times (in milliseconds) for committee building and example selection. A blank entry indicates that QBag-rb did not complete all 10 runs for the particular dataset, while a zero entry indicates an average execution time of less than 1 millisecond.

Dataset	No. of committee regions		No. of merged tree regions	
	QBag	QBag-rb-min	QBag-rb	QBag-rb-min
anneal	251	120	4,712	358
australian	262	95	90,288	487
car	473	466	355	355
corner-kick	607	52	4,878	35
german	639	617	1,153,948	741,023
give-and-go	3,696	465	–	18,591
kr-vs-kp	239	248	1,124	1,124
letter	672	189	–	43,727
nursery	2,816	2,824	2,165	2,165
pendigits	385	195	–	210,945
pima-indians	201	108	–	38,703
segmentation	115	84	–	11,594
tic-tac-toe	452	458	11,531	11,531
vehicle	195	62	–	75,646
vowel	105	99	874,091	63,008
wdbc	63	51	–	9,255
yeast	357	153	1,012,417	15,618

Table 4.5: Average number of decision regions in the committee and in the merged tree. A blank entry indicates that QBag-rb did not complete all 10 runs for the particular dataset.

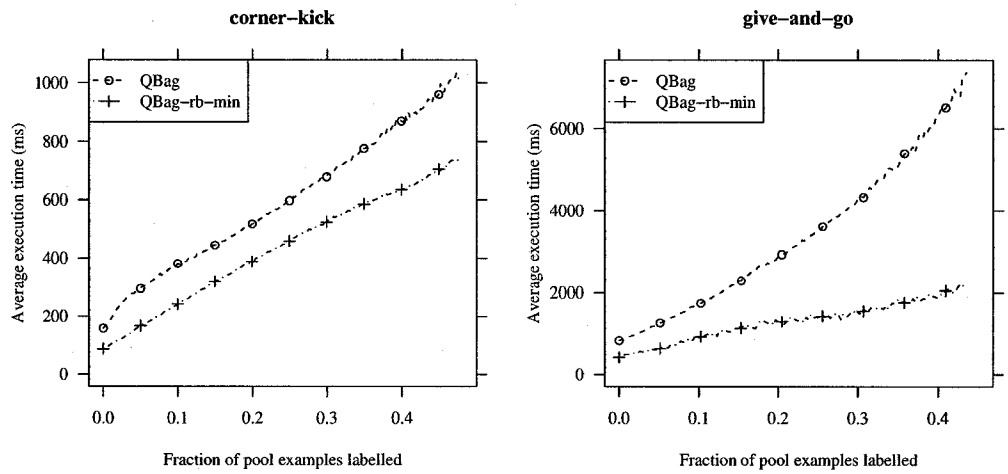

Figure 4.7: Average execution times on the corner-kick (left) and give-and-go (right) datasets. Note that the scale of the y-axis differs between the two plots.

than unlabelled examples was exhibited on the car, corner-kick, kr-vs-kp, and nursery datasets, where the example-selection times for QBag-rb were lower than those of QBag. In all these cases, however, the cost of constructing the merged tree offset these reductions, resulting in higher total execution times for QBag-rb. On datasets where QBag-rb created particularly large merged trees (e.g. german, vowel, yeast), both the committee-building and example-selection times were much higher than those of QBag; the latter increase was due to the fact that there were many more decision regions in the merged tree than there were unlabelled examples in the pool.

Next, the impact of performing λ-Alignment using the λ_{pct} values from Table 4.1 is assessed. First of all, for the 13 datasets that possessed at least one continuous attribute, enabling the pre-pruning rule always led to a reduction in the average number of decision regions contained in the committee. This meant that on each of these datasets, C4.5 attempted to produce decision regions that were smaller than λ_a along some dimension a, and these attempts were thwarted by the pre-pruning rule that is enabled when using λ-Alignment. The simplifications made to the committee did not necessarily translate into lower committee-building execution times for QBag-rb-min, however, as more time was usually required to construct the merged tree than was saved by pre-pruning. Exceptions to this trend were the anneal, corner-kick, give-and-go, and letter datasets, where the time to build the committee was greater when using QBag then with QBag-rb-min. As for the datasets that contained discrete attributes only, QBag-rb-min actually built larger committees than QBag on some of these, which seems counterintuitive, since the λ_{pct} parameter has no effect here. This is simply a consequence of different examples being labelled by the example-based and region-based methods – given identical training sets, equivalent committees would be produced by the two methods from these datasets.

Regarding the datasets in Table 4.5 for which QBag-rb completed all 10 runs of selective sampling, it is clear that QBag-rb-min reduced the size of the merged tree, often dramatically, in cases where at least one continuous attribute was present. The chief concern, however, is in the overall execution times for QBag relative to those of QBag-rb-min, as these are the two viable sampling methods. On 12 of the 17 datasets, the average execution time for QBag was below that of QBag-rb-min; only on the anneal, australian, corner-kick, give-and-go, and letter datasets was QBag-rb-min more computationally efficient. Still, for the large corner-kick and give-and-go datasets, where the computational savings were the greatest, the total times for a complete run of active learning were reduced by an average of approximately 33 and 355 seconds, respectively. The execution times for these two datasets are plotted in Figure 4.7. In the case of corner-kick (left), QBag-rb-min is faster than QBag by a relatively constant amount; on the give-and-go dataset (right), however, the gap between the execution times for the two methods widens as the training set gets larger. In Table 4.2, one can see that QBag and QBag-rb-min obtained the same weighted average error rates on both of these datasets. These examples demonstrate the potential of the region-based approach to reduce the amount of computation required for selective sampling, while maintaining a competitive level of accuracy. Of course, there were also datasets on which the average execution time for QBag-rb-min was much greater than that of QBag; hence, the efficiency of the former is highly dependent on the particular dataset.

It should be noted that because QBag-rb-min employs direct selection, it does not actually need to compute the merged tree. Since the m highest-scoring examples are always selected during a given iteration, it makes no difference whether scores are assigned to the decision regions contained in the merged tree or assigned directly to the unlabelled examples. Thus, the computational efficiency of QBag-rb-min can be greatly improved by performing λ-Alignment to simplify the hypothesis formed by the committee, but skipping the construction of the merged tree and assigning scores to the unlabelled examples instead. The merged tree is required for BootM-rb-min, as it selects examples according to the distribution of scores defined over the decision regions in the merged tree. When these methods are converted into query learners in Section 4.6, the merged tree must be computed by both methods, since no pool of unlabelled examples exists in this setting.

Finally, an interesting observation was made when comparing the average amount of time taken by QBag and QBag-rb to classify the unlabelled examples. For QBag, this consists of having each of the 10 committee members classify the examples, whereas for QBag-rb it involves classifying each example using the single merged tree. As can be seen in Table 4.6, of the 10 datasets on which the merged tree could be constructed and stored in memory, it classified the unlabelled examples more efficiently than did the actual committee in 5 of these cases. This is interesting because both entities represent the same hypothesis, and only on the car and nursery datasets was the number of decision regions in the merged tree actually less than sum of the decision region counts in the committee members (see Table 4.5).

Dataset	QBag	QBag-rb	Ratio
anneal	0.7	0.3	**2.33**
australian	0.4	2.9	0.14
car	1.0	0.2	**5.00**
corner-kick	36.7	7.1	**5.17**
german	0.9	35.9	0.03
kr-vs-kp	3.5	0.5	**7.00**
nursery	9.2	1.2	**7.67**
tic-tac-toe	0.8	0.8	1.00
vowel	0.4	25.2	0.02
yeast	1.1	31.0	0.04

Table 4.6: Average time (in milliseconds) taken to classify the unlabelled pool examples. The rightmost column displays the ratio of QBag's average time to that of QBag-rb, and bold entries indicate that QBag-rb classified examples more efficiently than QBag on the particular dataset.

Thus, if a bagged committee of decision trees is learned from a given dataset, for example, and will subsequently be used to classify a large amount data, it may be worthwhile to make a one-time investment of CPU-time to convert the committee into a single tree classifier, using the TreeMerge procedure described in Section 4.2 (provided that sufficient memory is available to store the tree). Systems that require real-time performance, such as a network intrusion detection system that processes thousands of packets every second, can utilize this approach to reduce the cost of classifying examples. These are plausible applications for the TreeMerge algorithm, since a tree committee created using bagging is typically more accurate than any single tree that is induced from the same training data [5]; yet, a merged tree representing the committee may take less time to classify unseen examples. Of course, committees formed using other ensemble methods, such as boosting, could also be merged for the purpose of creating more computationally efficient classifiers. It must be emphasized, however, that the merged tree was far less efficient than the original committee in some cases (e.g. german and vowel datasets). Only on certain datasets, where the merged tree is not extremely large, is classification cheaper when using the merged tree.

4.5.3 Increasing the Minimum Allowable Decision Region Size

In order to investigate the effect of increasing the value of the λ_{pct} parameter for the methods that used λ-Alignment, the experimental procedure described in Section 4.4 was repeated, this time using λ_{pct} values that were .05 higher than the ones listed in Table 4.1. The datasets that contained discrete attributes only were not included in this set of experiments, due to the fact that λ_{pct} does not affect the decision trees grown from such datasets. The weighted average error rates for the remaining 13 datasets are displayed in Table 4.7. Only the results for the region-based selective sampling methods are shown; the versions using the higher λ_{pct} values are named BootM-rb-min+ and QBag-rb-min+. In general, increasing λ_{pct} resulted in higher error rates, although the differences in performance were not statistically significant.

Dataset	BootM-rb-min+	QBag-rb-min+	BootM-rb-min	QBag-rb-min
anneal	.121 (3)	.120 (2)	.123 (4)	.117 (1)
australian	.122 (1)	.129 (3)	.125 (2)	.131 (4)
corner-kick	.011 (4)	.009 (2)	.009 (2)	.009 (2)
german	.298 (3)	.289 (1)	.309 (4)	.292 (2)
give-and-go	.196 (4)	.194 (3)	.193 (1.5)	.193 (1.5)
letter	.015 (3.5)	.015 (3.5)	.012 (1.5)	.012 (1.5)
pendigits	.011 (2.5)	.011 (2.5)	.011 (2.5)	.011 (2.5)
pima-indians	.285 (4)	.283 (3)	.282 (2)	.277 (1)
segmentation	.014 (2.5)	.016 (4)	.012 (1)	.014 (2.5)
vehicle	.224 (2.5)	.230 (4)	.222 (1)	.224 (2.5)
vowel	.037 (2)	.037 (2)	.038 (4)	.037 (2)
wdbc	.063 (4)	.058 (3)	.056 (2)	.055 (1)
yeast	.254 (4)	.251 (1)	.252 (2)	.253 (3)
Avg. rank	(3.077)	(2.615)	(2.269)	(2.038)

Table 4.7: Weighted error rates.

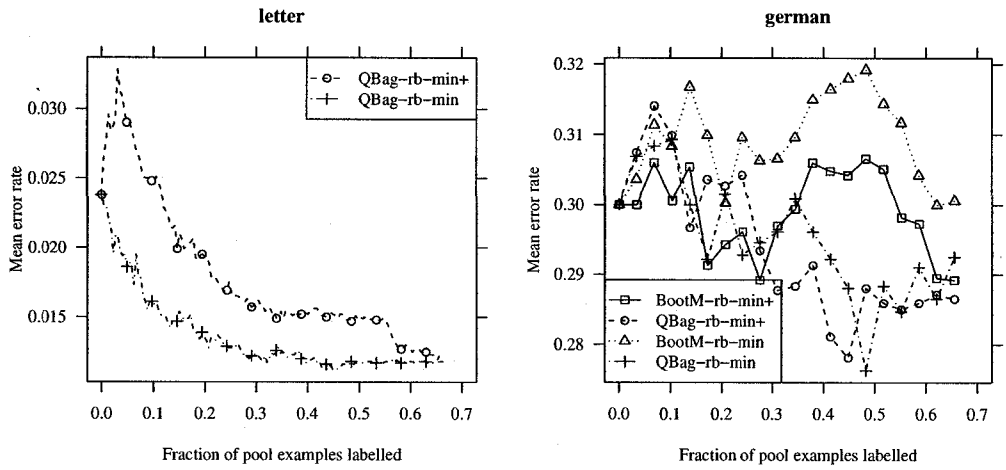

Figure 4.8: Mean error rates on the letter (left) and german (right) datasets.

Figure 4.8 (left) displays the learning curves for QBag-rb-min+ and QBag-rb-min on the letter dataset. The plots for the two BootM methods were omitted from this graph to improve clarity – their relative performances were similar to those of the bagging methods. QBag-rb-min achieved a lower mean error rate than QBag-rb-min+ at every iteration on this dataset, which indicates that the value of $\lambda_{pct} = 15\%$ used by the latter method oversimplified the hypotheses represented by the committee, and led to poorer example selection. Still, there were some datasets for which using a higher value of λ_{pct} improved the predictive performance of the induced classifiers. For example, on the german dataset, whose error rate plots are shown in Figure 4.8 (right), QBag-rb-min+ and BootM-rb-min+ both outperformed their respective versions that used smaller λ_{pct} values.

These results suggest that it may be beneficial to tune the λ_{pct} parameter for particular datasets when using λ-Alignment. The approach taken in the experiments conducted in this chapter was simply to set λ_{pct} to the lowest multiple of 5% for which the merged trees that were constructed could always fit into memory. Thus, a lower limit may exist for λ_{pct} on a given dataset due to the computational demands of merging the committee members. With regard to increasing λ_{pct}, there will come a point where the hypothesis formed by the committee becomes oversimplified, and ceases to choose examples intelligently. Although the λ_{pct} values used in these experiments did not go to such extremes, in preliminary trials, for example, setting λ_{pct} at or above 20% on the corner-kick dataset caused each committee member to grow a tree consisting of a single leaf that predicted the majority class. In this situation, selective sampling deteriorated into random selection.

4.6 Region-based Query Learning

Another active learning paradigm, which was briefly mentioned in Chapter 3, is learning with membership queries [3]. Under this learning scheme, unlabelled examples are generated and are then labelled by an expert or oracle. Two advantages of learning with membership queries instead of performing selective sampling are: 1) no pool of unlabelled examples is needed, meaning that the accuracy with which the target concept can be learned is not limited by the choice of pool examples, and 2) it is not necessary to assign scores to and rank a potentially large number of unlabelled examples during each iteration of learning, which may reduce the computational cost of active learning if queries can be constructed efficiently. Query learning may not be appropriate in some domains, however, since it may not always be possible to construct meaningful queries. For example, Baum and Lang [4] applied membership query learning to the problem of recognizing hand-written digits; however, it turned out that many of the queries were difficult for the human experts to label, as the images that were generated were frequently unnatural combinations of other characters.

4.6.1 Query Learning Versions of Region-based Algorithms

The region-based approach to selective sampling can be formulated as a query learning method. Recall that region-based selective sampling is a two-stage process, which consists of choosing an

informative region, and then selecting an unlabelled example from that region. A region-based query learning method performs the first stage in the same manner; then, once a potentially informative region is identified, an unlabelled example (or query) is *generated* from that region and then labelled by the expert or oracle.

To instantiate this method, QBag-rb-min and BootM-rb-min were converted into into query learning methods. The only difference between these selective sampling methods and their respective membership query versions, which were called QBag-query and BootM-query, is that the query learners generate unlabelled examples instead of choosing them from a pool. The committee formation and selection of decision regions do not change. Given a decision region from which a labelled example is desired, both QBag-query and BootM-query generate a random example that satisfies the constraints specified by the region, according to a uniform distribution.

Like QBag-rb-min, QBag-query also uses direct selection. Since the number of unique examples that can be generated from a given decision region is likely to be larger than the batch size m (and may be virtually unlimited if many continuous attributes are present), the following sampling schedule was followed. For a batch size of m, QBag-query generated $\max(m \cdot 2^{-i}, 1)$ examples from the i^{th} highest-scoring region, then proceeded to the $(i+1)^{st}$ highest-scoring region, and so on, until the quota was filled. As for BootM-query, which uses weight sampling rather than direct selection, no such schedule is needed. Also, for both methods, steps were taken to ensure that the same query was not generated multiple times during a single iteration.

4.6.2 Experiments and Discussion

Experiments were conducted in order to compare the performance of the two query learning methods, QBag-query and BootM-query, to that of their region-based and example-based selective sampling equivalents. Since no oracle was available for the UCI datasets that were used in previous sections, a query learning environment was simulated. For a given dataset, one third of the examples were randomly selected to form the evaluation set E, as described in Section 3.3. However, in this case, the examples in E were used to train a 1-nearest neighbour (1-NN) classifier,[10] which subsequently served as the oracle. For trials in which the query learning methods were used, the remaining data was discarded, and the method could request the label of any point in the space. Maximum and minimum values were imposed on continuous attributes for examples that were generated, according to the extreme values that occurred in the original data. For trials involving the selective sampling methods, the instances not contained in E had their labels removed, and subsequently formed the initial training set and example pool. Whenever the class label of one of these examples was requested, the oracle assigned a label that may have been different than the label of the example in the original dataset, since the 1-NN classifier was merely an approximation of the true target concept. For each dataset, the same batch size and maximum number of iterations that were

[10]The distance metric used by the 1-NN classifier was the Heterogeneous Euclidean-Overlap Metric [55], which is applicable to datasets that consist of continuous and/or discrete attributes.

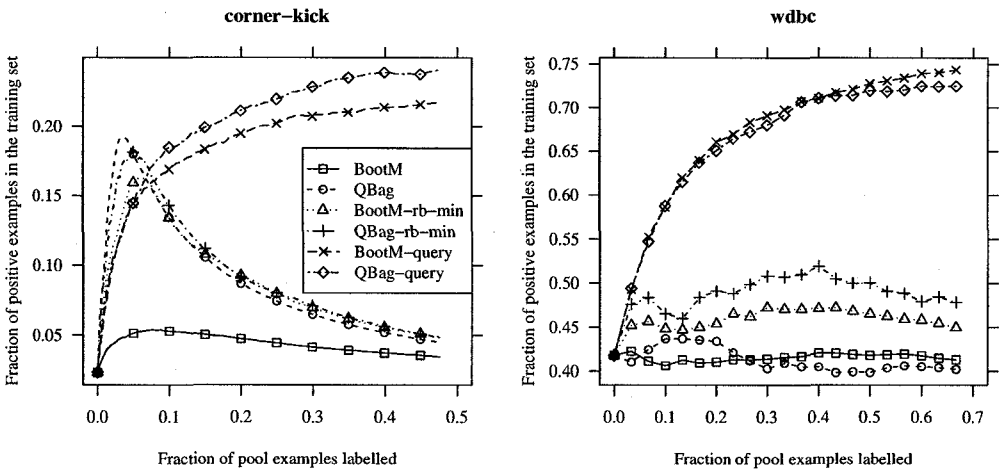

Figure 4.9: Average fraction of examples in the training set belonging to the positive class (i.e. the minority class), on the corner-kick (left) and wdbc (right) datasets. Note that the scale of the y-axis differs between the two plots, and the legend for corner-kick also applies to wdbc.

used in Section 4.4 were also used in these experiments. Error rates were calculated with respect to the examples in E.

The results of these experiments illuminated a fundamental problem with the query learning approach that had been formulated. With respect to error rates, the query learning methods were, in fact, inferior to the selective sampling versions on most of the datasets. For brevity, the table of results is deferred to Appendix C.2. A glaring symptom of the poor performance was that both QBag-query and QBoost-query tended to drastically inflate the number of positive (i.e. minority class) examples in the training set – in some cases, the fraction of examples belonging to the positive class increased at every iteration (see Figure 4.9). This behaviour typically yielded extremely complex models that overfit the data and performed poorly on the unseen examples in the evaluation set.

The reason that these query learning methods fail is that there is no limit to the number of labelled examples that can be requested from a given area of the instance space. Recall that the scoring function employed by both QBag and BootM (as well as by QBag-query and BootM-query) calculates the margin, which is a measure of the disagreement among the committee members. Disagreement is to be expected near a true decision boundary for a continuous attribute, for instance, since the particular data sample on which a given committee member is trained is likely to contain some examples near the boundary that are not included in another committee member's data sample, and vice versa. As a consequence, each member places a threshold in the vicinity of this boundary, but at slightly different locations. When performing selective sampling, the disagreement among the committee members regarding such a boundary leads to examples in that area being labelled. This

information may or may not cause the members to make the same predictions near the boundary at subsequent iterations; however, if the disagreement continues, there will (presumably) come a point where no unlabelled examples remain in the area, and labels will thereafter be requested from other locations in the space.

When using QBag-query or BootM-query, the disagreement regarding a given decision region also may never be resolved. However, a crucial difference in this situation is that the method will never be prevented from sampling in such a region. Consequently, the query learner may concentrate much or all of its sampling effort on relatively small areas of the space, which can lead to poor generalization. Due to the fact that fundamental changes to the selective sampling methods are needed in order to adapt them for query learning, further examination of the circumstances under which these problems are manifested, as well as possible corrections, are left as future work.

4.7 Conclusions

In this chapter, a novel approach to active learning with decision trees was proposed, in which decision regions, rather than individual examples, are ranked by the active learner. An algorithm, called TreeMerge, was devised for the purpose of combining several decision trees, such as the set of trees created by a committee-based selective sampling method, into a single tree. Using this algorithm, the hypothesis represented by a committee of trees was converted into a set of mutually exclusive decision regions; these regions were then processed by the particular sampling method in order to select informative examples from the unlabelled pool.

The merged trees constructed from some datasets were extremely complex – in certain cases, they were too large to be stored in memory. It was also argued that tiny decisions could be detrimental to selective sampling, since they can encourage the selection of instances that are very similar, such that examples in the batch are redundant. To address both of these concerns, a procedure called λ-Alignment was developed that simplified the joint committee hypothesis by eliminating decision regions that were smaller than some minimum allowable size. This was accomplished by instituting a pre-pruning rule when growing each committee member, as well as "aligning" thresholds on a given continuous attribute that were sufficiently close to one another. In accordance with this enhancement, an additional parameter to the active learning procedure was required, namely, λ_{pct}, which determined the minimum allowable size of decision regions for datasets that contained one or more continuous attributes.

Experimental results showed that the region-based approach, when combined with λ-Alignment, was competitive with existing example-based sampling methods. In particular, a region-based version of the BootM method, called BootM-rb-min, outperformed the original example-based version by a statistically significant margin. The lower error rates achieved by BootM-rb-min were attributed to the fact that it tended to define a distribution over the decision regions that was less biased toward the negative (majority) class, compared to the distribution that BootM defined over the unlabelled

examples. As a consequence, the former method added more positive examples (i.e. examples from the minority class) to the training set, which usually led to lower error rates. A region-based version of query-by-bagging performed as well as the original method, on average. This result is interesting because it suggests that when training a single decision tree by selective sampling, a simplified committee is as effective as a more complex one in most cases, in terms of choosing informative examples. Increasing the λ_{pct} by an additional 5% for each dataset resulted in poorer performance, although improvements were observed in some cases.

Last of all, a method for converting the region-based selective sampling methods into query learning methods was proposed. The experiments that were conducted, however, revealed a fundamental flaw in the approach, which had to do with the fact that there was no limit to the number of queries that could be made in a given decision region. Although the basic region-based query learning approach holds promise, a number of implementation details must be addressed before it can be competitive with existing selective sampling procedures. This is left as future work.

Chapter 5

Related Work

5.1 Improving the Stability of Decision Trees

The motivation for replacing the entropy splitting criterion in C4.5 [42] with the DKM criterion [14, 28] in this thesis was to increase the stability of the decision tree algorithm while maintaining a single, accurate, interpretable model. A number of different techniques have been proposed in the literature that aim to achieve this. These techniques are not considered to be in the same category as certain ensemble methods, such as bagging [5], which produce tree-based models that tend to be more semantically stable, but that are no longer interpretable. The stability-improving techniques reviewed here are only those that yield a single tree model.

With the goal of building a classifier that is accurate, stable, and reasonably interpretable (i.e. not too large), Breiman and Shang [8] devised an algorithm for representing a bagged ensemble of trees by a single tree classifier. First, a bagging ensemble was created from the available training data. Next, additional attribute vectors were *manufactured* and labelled by the ensemble. Finally, a single decision tree, called a *representer tree* was trained on the manufactured data.[1] The authors observed that, on average, the representer trees were more accurate than the trees grown from the original training data, but less accurate than the bagging ensemble. They attributed the accuracy gains of the representer tree over the individual tree to the decrease in variance that results from representing an ensemble, which is inherently more stable.

A method that is closely related to the idea of representer trees is "Combined Multiple Models" (CMM), which produced results that were similar to those of the above study with respect to classification accuracy [17]. The rule-learning algorithm C4.5Rules [42] was used as the base learner in this case,[2] and semantic stability was empirically measured as the expected agreement between pairs of hypotheses (see Section 2.4). A comparison between bagging and CMM revealed that CMM retained, on average, a significant fraction of bagging's stability gains, while adequately maintaining the desired property of intelligibility.

[1] Data is actually manufactured during the tree-growing phase, and not after, as steps are taken to ensure that each node that is created receives the same number of examples. Additional details may be found in [8].

[2] CMM is a meta-learner that can be applied to any base learning algorithm, including a decision tree inducer.

In order to increase the stability of a single decision tree, Dannegger [12] suggests a modification to the procedure that is used to decide on the best split at a given node when growing the tree. Inspired by the stability and accuracy gains achieved by bagging, Dannegger created 100 bootstrap replicates of the training examples at a node. The best split was determined for each of these data samples, and the attribute a that was nominated most frequently was the attribute tested at this node. All features were continuous-valued, and the threshold for the test was chosen as the median of the thresholds for a that were derived from the bootstrap replicates. This "node-level stabilization" method was tested on a single dataset, where it was found that the resulting decision tree was inferior to, but competitive with bagging in terms of error rate. However, stability was not actually quantified in this study.

Essentially the same technique employed by Dannegger was also proposed by Perez et. al [40], under the name "Consolidated Trees." Structural stability was measured using the *Common* metric (see Section 2.4) in this case, and the authors concluded that the CT learner produced trees that were more stable and more accurate than those created by C4.5. They also found that the structural stability of a CT tended to increase with the number of subsamples that were generated at a given node.

Li and Belford [32] suggest another node-level modification to the tree-growing procedure; however, a decision tree produced by this method is fundamentally different than a C4.5 tree, as the former allows a conjunction of tests to be performed at a single node. Nevertheless, the idea here was to identify the "almost equally good" splits at a given node – a set of splits which, if small changes were made to the data sample, would overtake the current best split. Once the set of almost equally good tests was determined, the possible predicates in conjunctive normal form involving these tests were generated, and the conjunction of tests that maximized the splitting criterion was performed at the node. An experiment conducted using a synthetic data demonstrated that the proposed algorithm was less sensitive to noise than the well-known decision tree inducer CART [7]. It should be noted, however, that the proposed method significantly increased the amount of computation required to grow a tree, particularly in cases where there were a large number of almost equally good splits.

Finally, a technique proposed by Shannon and Banks [51] combines a set of decision trees into a single tree by first defining a probability distribution over the tree structures (ignoring thresholds on continuous attributes). Then, given a collection of observed trees, the maximum likelihood estimate (MLE) of a central tree is computed, and serves as the estimate of the true tree structure. The thresholds for tests on continuous attributes can then be specified by fixing the attribute at a given node to that which appears in the MLE tree, and using a splitting criterion to find the best cut-point. Experiments conducted on biomedical data showed that the MLE tree was more accurate than a single decision tree grown from the entire dataset, but exhibited a higher error rate than bagging.

5.2 Active Learning

In the active learning experiments conducted in Chapters 3 and 4, the batch size (i.e. the number of examples labelled per iteration) was fixed for each dataset. This is typical of many active learning studies (e.g. [1, 25, 30, 47, 49, 56]). Provost et al. [41] examine methods for *progressive sampling*, in which successively larger samples are drawn. This research does not involve active learning, but is instead concerned with learning from a very large (labelled) dataset, where it is computationally expensive to induce a classifier from all the available examples. Whereas active learning aims to reduce the number of labelled examples that are needed to train an accurate classifier, progressive sampling strives to minimize the total computational cost of inducing a classifier, without compromising accuracy. Another difference between the two methods is that progressive sampling attempts to detect convergence – the minimum training set size beyond which accuracy does not improve on a given dataset. No further sampling is performed once convergence has been detected. By contrast, active learning typically ceases once the available resources (e.g. labelling effort, computational time) have been exhausted. One exception is an active learning approach based on support vector machines that uses a heuristic stopping criterion [49]. I am not aware of any learning technique that incorporates ideas from both active learning and progressive sampling.

Although not designed specifically for active learning, *incremental* decision tree learners exist that incorporate new examples into an existing tree, and update the tree structure accordingly. The Incremental Tree Inducer (ITI) developed by Utgoff et al. [53], for instance, offers a number of different training "modes," some of which have interesting implications for active learning. The first of these is *incremental mode*, which adds a new example to an existing tree and then restructures the tree, such that each decision node performs the optimal test given the training examples reaching that node. This training mode guarantees that the resulting tree is identical to the one that is induced from scratch, given the same set of examples – there is a unique tree for each training set. Thus, the only reason that one would use this mode during active learning would be to improve computational efficiency.[3] By contrast, in *error-correction mode*, an incoming example is added to the existing tree only if the tree would misclassify it. This seems counterproductive when considered in the active learning setting, since discarding a correctly classified example $\mathbf{x}$ implies that the effort taken to label $\mathbf{x}$ is wasted. In any case, incremental decision tree learners are interesting from this perspective, and potential applications to active learning may be worthy of further investigation.

[3]The average cost of the procedure that is used to update the existing tree is shown to be lower than the average cost of rebuilding a new tree from scratch.

Chapter 6

Conclusions

6.1 Summary

This thesis has examined the stability of the C4.5 decision tree learning algorithm [42] in two different contexts. In Chapter 2, pairs of decision trees were induced from training sets that were obtained by dividing the available data into two approximately equal halves. A novel measure of structural stability, called the region stability metric, was proposed and then used to quantify the structural similarity between each pair of trees. Given two decision trees T_1 and T_2, the region stability score represents the probability that a randomly chosen example is classified in "equivalent" decision regions (or leaf nodes) by T_1 and T_2. The definition of "equivalent" decision regions can be relaxed by increasing the value of the ε_{pct} parameter when computing stability scores; in doing so, thresholds defined on a particular continuous attribute by two decision regions (in two different trees) are considered to be equal if they are within some range of one another. Semantic stability, or the degree to which two trees make the same predictions on unseen examples, was also measured from the same divisions of the training data. A 2-fold cross-validation procedure, proposed by Turney [52], was used to estimate both types of learner stability.

In Chapter 3, the stability of C4.5 was measured in the context of active learning, whereby the learner plays a role in choosing the examples that are used to train a classifier. The focus here was on a particular active learning paradigm known as selective sampling, in which a pool of unlabelled examples is provided; the learner attempts to choose the most informative of these examples, in order to build an accurate classifier while minimizing the labelling effort required. Including random sampling, 7 sampling methods were compared empirically in terms of their predictive accuracy and stability, when using C4.5 as a base learner. Once again, the region stability metric was used to measure structural stability; however, in this case, three different aspects of active learning stability were quantified, by applying the metric to specific pairs of hypotheses. These applications of the region stability metric were called FinalStab, PrevStab, and RunStab. The random sampling and uncertainty sampling methods were found to be generally inferior to the committee-based selective sampling methods with respect to stability and error rate. Furthermore, the three committee-based

methods that employed direct selection outperformed the two methods that used weight sampling. In particular, the query-by-bagging method produced the most accurate classifiers, and achieved competitive scores on each of the stability measures.

Another focal point of this research has been the comparison of two different splitting criteria that are used by C4.5 when growing a decision tree. The default criterion employed by C4.5 is based on the entropy impurity measure; an alternative splitting criterion, named DKM [14, 28], was also tested in this thesis. The experiments conducted in Chapters 2 and 3 were each repeated by first growing trees using the entropy criterion, and then using DKM. In this manner, the stability and predictive accuracy achieved by entropy and DKM were compared on a representative collection of benchmark datasets. The experimental results from Chapter 2 provided evidence suggesting that DKM improves the stability of C4.5 relative to the entropy criterion; the error rates were virtually indistinguishable in the passive learning setting. However, the active learning results overwhelmingly favoured DKM, which grew trees that were consistently more accurate and more stable than those produced using the default entropy criterion.

In Chapter 4, a novel approach to active learning with decision trees was proposed, in which decision regions, rather than individual examples, were ranked by the active learner. The focus here was on committee-based methods, and an algorithm was devised for merging several decision trees into a single tree. By this procedure, a hypothesis represented by a committee of trees was converted into a set of mutually exclusive decision regions that were then processed by the particular sampling method in order to select informative examples. Because the merged trees constructed from some datasets were extremely complex, a mechanism was devised for simplifying the hypothesis formed by the committee prior to the merge operation; this introduced an additional parameter to the active learning procedure that determined the minimum allowable size of a decision region with respect to continuous dimensions. Experiments were conducted that demonstrated the merits of the region-based approach to selective sampling, when compared to the traditional example-based approach. A region-based version of the bootstrap-M method significantly outperformed the original example-based version, and a region-based version of query-by-bagging performed as well as the original method, on average, while using a simplified committee to choose examples. An attempt to formulate the region-based approach as a query learning method was unsuccessful; the query learners tended to concentrate their efforts in relatively small areas of the space, as there was no limit on the number of queries that could be made in a given decision region.

6.2 Limitations and Future Work

Although moderate improvements in the structural stability of C4.5 were attributed to the DKM splitting criterion based on the results of the 2-fold cross-validation experiments conducted in Chapter 2, in the active learning setting of Chapter 3, DKM produced trees that were significantly more stable than those grown using entropy. As was noted in Section 2.7, the region stability scores for

both splitting criteria were very low on many of the datasets, when ε_{pct} was set to 5%, for example. In hindsight, it may have been an overly demanding task for C4.5 to grow structurally similar trees from two (presumably) non-overlapping training sets, such as the ones presented to the learner during the 2-fold cross-validation procedure. This is especially challenging if a given dataset contains a relatively small number of examples even before the available data is halved. Unless there is a fair amount of redundancy in the training data, C4.5 is unlikely to perform the same sets of tests when classifying the examples in the two folds. On the other hand, the pairs of trees that were evaluated by the region stability metric during active learning were usually induced from data samples that contained at least some, and often many of the same training examples. The only situation in which the data samples used to grow a pair of trees could have been completely independent was when the RunStab measure was calculated; moreover, this could only have been the case if fewer than half of the examples in $L_1 + U_1$ had been labelled. The structural stability scores achieved by DKM were significantly better than those obtained by entropy when the pairs of training sets were likely to have examples in common; this suggests that, in a passive learning situation, clearer differences between the structural stability of the two criteria may be visible if the training sets presented to the decision tree learner contain some degree of overlap. This is a possible avenue for future research. Of course, from a practical perspective, if the particular learning task naturally involves forming different data samples and growing trees from these (e.g. active learning), then the stability of the learner should be evaluated with respect to these trees, without concern for the properties of the data samples.

Although not suggested by Utgoff et al. [53], one can imagine another training mode for an incremental decision tree learner (see Section 5.2) in which each new example $\mathbf{x}$ is incorporated into the tree, but the tree is only updated in the event that it misclassifies $\mathbf{x}$. That is, if $\mathbf{x}$'s class label matches the predicted label of the leaf in which it is classified, no changes are made to the tree structure. This can be regarded as maintaining a tree that is "good enough," in the sense that it is consistent with the examples in the training set,[1] but may not perform the best test at each decision node. Nevertheless, it is plausible that such a scheme could improve the stability of active learning, particularly on the PrevStab measure. The procedure adopted throughout this thesis was to grow a tree from scratch whenever new examples were added to the training set, simply because this is the approach that has typically been taken when performing active learning with tree classifiers (e.g. [30, 47]). However, an obvious direction for future research on improving the stability (and efficiency) of active learning with decision trees is to explore incremental induction algorithms and related training strategies.

One drawback with the region-based active learning approach proposed in Chapter 4 is that the λ-Alignment method introduces an additional parameter into the active learning procedure. This is the λ_{pct} parameter that controls the minimum decision region size permitted in the joint committee

[1] This may not be the case if the tree has been pruned. The ITI algorithm performs "virtual pruning," which sets a flag for a subtree that has been pruned, but does not actually remove the subtree, since new examples may reverse the original decision that was made to prune the subtree.

hypothesis. Although there is a clear need to reduce the complexity of the joint committee hypothesis, due to the massive merged trees that were often produced in the absence of λ-Alignment, the λ_{pct} parameter may not be the most straightforward way of specifying the degree of simplification. The main problem with this particular parameter is that its value may often have to be determined by trial-and-error, since the number of decision regions in the joint committee hypothesis cannot always be estimated accurately. Future work on the region-based approach might involve re-formulating the λ-Alignment procedure, so that it instead accepts a parameter that dictates the maximum *number* of decision regions that may be created; a minimum decision region size could then be determined from this value. It was also assumed in Chapter 4 that the minimum widths along each attribute were proportional to one another. However, if the number of unique values appearing in the data for a continuous attribute a_1 is much greater than the number of different values for another continuous attribute a_2, the minimum decision region width along a_2 could conceivably be smaller than that of a_1, as fewer thresholds are likely to be set by the trees in the committee along dimension a_2.

Another important direction for future research that was noted in Chapter 4 is the instantiation of the region-based active learning approach as a viable query learning method. An attempt was made in Section 4.6 to do just this; however, the query learners performed poorly on most datasets, frequently yielding higher error rates than the selective sampling methods. It appears that some mechanism is needed to prevent the query learner from concentrating most or all of its sampling effort in a small portion of the instance space. For example, the density with which a given decision region has been sampled (i.e. the number of labelled examples in the region relative to its volume) could be tracked, and additional queries in already densely-sampled regions could be discouraged or even prevented entirely. Different strategies are also possible for choosing the query point within a region that is believed to be informative. In Section 4.6, a simplistic approach was taken, in which an unlabelled example was randomly generated so as to satisfy the constraints of the selected region. One alternate approach might be to refine decision boundaries by concentrating queries near the borders of adjacent decision regions that predict differently [56]. Finally, for sampling methods that employ direct selection, it is necessary to establish a schedule or policy for dividing the m queries in the batch among the regions that have the best scores. In selective sampling, this is simply a matter of requesting that the top m scoring examples be labelled; yet, in a query learning environment, there is no inherent restriction on the number of queries that can be made in a given region. For example, all m queries could be made in the highest-scoring region, although this might lead to an undue amount of redundancy among the examples in the batch.

Lastly, the C4.5 decision tree learner contains certain features, such as the heuristic penalty that it applies to candidate splits on continuous attributes, which do not readily accommodate different splitting criteria. In pairing the DKM criterion with C4.5 in this thesis, steps were taken to reduce the number of confounding factors that might influence a comparison between it and entropy (see Appendix B). Although DKM performed favourably under these conditions, there may be a more

principled way to estimate, or even derive an appropriate penalty term for DKM, in order to further enhance its performance. In addition, an extension of DKM to learning problems that contain more than two classes has yet to be proposed [14].

6.3 Final Word

The main contributions of this thesis include a detailed study of decision tree instability, involving a comparison between two different splitting criteria, as well as an examination of the instability of several active learning methods that employ the C4.5 decision tree inducer as a base learner. A novel structural stability metric was proposed, which was also used to measure three different aspects of learner stability in the context of active learning. This is the first known study of decision tree stability in an active learning setting, and this work also marks the first case where a committee of decision tree learners selected examples for training a single tree. Finally, a region-based approach to active learning was formulated that yielded positive results when applied to two existing selective sampling methods. This idea creates a number of directions for future research involving active learning with decision trees; in particular, committee-based query learning methods may be developed within this framework.

The most notable findings based on the experiments conducted in this thesis were that the DKM splitting criterion outperformed entropy, particularly on the active learning tasks. Secondly, it was determined that certain selective sampling methods were significantly more accurate and more stable than others, in terms of the decision trees that they produced. Last of all, the performances of the region-based selective sampling methods demonstrated that simplifying the hypothesis formed by a committee of trees could sometimes lead to improved example selection. The region-based approach was especially effective for a method that employed weight sampling.

These results are important because decision tree algorithms, such as C4.5, are widely used in practice, not only as passive learners, but as active learners as well. The use of DKM as a splitting criterion for C4.5 is recommended, as this combination yields trees that are at least as accurate and typically more stable than those grown using entropy. When training a single decision tree via selective sampling, DKM should be used in combination with a competitive committee-based method, such as query-by-bagging, for best results. Finally, reducing the complexity of the committee by enforcing a minimum decision region size can lead to better example selection, and can also improve computational efficiency under certain conditions. The conclusions presented here constitute a significant step forward in understanding and improving the decision tree classifiers that are created, in both passive and active learning settings.

Bibliography

[1] Naoki Abe and Hiroshi Mamitsuka. Query learning strategies using boosting and bagging. In *Proceedings of the 15th International Conference on Machine Learning*, pages 1–9, 1998.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, Cambridge, 2004.

[3] Dana Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[4] Eric B. Baum and Kevin J. Lang. Query learning can work poorly when a human oracle is used. In *In Proceedings of the International Joint Conference on Neural Networks*, pages 335–380, 1992.

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

[6] Leo Breiman. Heuristics of instability and stabilization in model selection. *Annals of Statistics*, 24(6):2350–2383, 1996.

[7] Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and regression trees*. Chapman and Hill, 1984.

[8] Leo Breiman and N. Shang. Born again trees. Technical Report, Dept. of Statistics, University of California, Berkeley, Available at ftp://ftp.stat.berkeley.edu/pub/users/breiman/BAtrees.ps, 1996.

[9] Wray Buntine and Tim Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, 8(1):75–85, 1992.

[10] David A. Cohn, Les Atlas, and Richard E. Ladner. Training connectionist networks with queries and selective sampling. In *Advances in Neural Information Processing Systems*, pages 566–573, 1990.

[11] David A. Cohn, Les Atlas, and Richard E. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1992.

[12] Felix Dannegger. Tree stability diagnostics and some remedies for instability. *Statistics In Medicine*, 19(4):475–491, 2000.

[13] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[14] Thomas G. Dietterich, Michael Kearns, and Yishay Mansour. Applying the weak learning framework to understand and improve C4.5. In *Proceedings of the 13th International Conference on Machine Learning*, pages 96–104. Morgan Kaufmann, 1996.

[15] Thomas G. Dietterich and Eun Bae Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical Report, Dept. of Computer Science, Oregon State University, Available at http://web.engr.oregonstate.edu/~tgd/publications/tr-bias.ps.gz, 1995.

[16] Cristian Dima and Martial Hebert. Active learning for outdoor obstacle detection. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, 2005.

[17] Pedro Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 2(3):187–202, 1998.

[18] Harris Drucker. Z splitting criterion for growing trees and boosting. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1255–1258, 1999.

[19] Chris Drummond and Robert C. Holte. Exploiting the cost (in)sensitivity of decision tree splitting criteria. In *Proceedings of the 17th International Conference on Machine Learning*, pages 239–246. Morgan Kaufmann, 2000.

[20] Bradley Efron and Robert J. Tibshirani. *An introduction to the bootstrap*. Chapman and Hill, 1993.

[21] Charles Elkan. The foundations of cost-sensitive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 973–978. Morgan Kaufmann, 2001.

[22] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning*, pages 148–156, 1996.

[23] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[24] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32:675–701, 1937.

[25] Steven C. H. Hoi, Rong Jin, Jianke Zhu, and Michael R. Lyu. Batch mode active learning and its application to medical image classification. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 417–424, 2006.

[26] Ronald L. Iman and James M. Davenport. Approximations of the critical region of the Friedman statistic. *Communications in Statistics*, A9(6):571–595, 1980.

[27] Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 2:241–254, 1967.

[28] Michael Kearns and Yishay Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of the 28th ACM Symposium on the Theory of Computing*, pages 459–468. ACM Press, 1996.

[29] Mark Last, Oded Maimon, and Einat Minkov. Improving stability of decision trees. *International Journal of Pattern Recognition And Artificial Intelligence*, 16(2):145–159, 2002.

[30] David D. Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proceedings of the 11th International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1994.

[31] David D. Lewis and William A. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th ACM International Conference on Research and Development in Information Retrieval*, pages 3–12. Springer Verlag, 1994.

[32] Ruey-Hsia Li and Geneva G. Belford. Instability of decision tree classification algorithms. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 570–575. ACM Press, 2002.

[33] Prem Melville and Raymond J. Mooney. Diverse ensembles for active learning. In *Proceedings of the 21st International Conference on Machine Learning*, pages 584–591, New York, NY, USA, 2004. ACM Press.

[34] Prem Melville, Stewart M. Yang, Maytal Saar-Tsechansky, and Raymond J. Mooney. Active learning for probability estimation using Jensen-Shannon divergence. In *Proceedings of The 16th European Conference on Machine Learning*, pages 268–279, 2005.

[35] John Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3(4):319–342, 1989.

[36] Trevor J. Monk, R. Scott Mitchell, Lloyd A. Smith, and Geoffrey Holmes. Geometric comparison of classifications and rule sets. In *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases*, pages 395–406, 1994.

[37] Sreerama K. Murthy and Steven Salzberg. Lookahead and pathology in decision tree induction. In *Proceedings of the 14th International Joint Conference on Artifical Intelligence*, pages 1025–1033, 1995.

[38] Peter B. Nemenyi. *Distribution-free multiple comparisons*. Ph.D. thesis, Princeton University, 1963.

[39] David J. Newman, S. Hettich, C. L. Blake, and C. J. Merz. UCI repository of machine learning databases, 1998.

[40] Jesus M. Perez, Javier Muguerza, Olatz Arbelaitz, Ibai Gurrutxaga, and Jose I. Martín. Consolidated trees: an analysis of structural convergence. In *Data Mining*, volume 3755 of *Lecture Notes In Computer Science*, pages 39–52. Springer-Verlag, 2006.

[41] Foster Provost, David Jensen, and Tim Oates. Efficient progressive sampling. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 23–32. ACM Press, 1999.

[42] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[43] J. Ross Quinlan. Bagging, boosting, and C4.5. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 725–730, 1996.

[44] J. Ross Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[45] J. Ross Quinlan. Miniboosting decision trees. Available at http://www.boosting.org/papers/Qui98.ps.gz, 1998.

[46] Nicholas Roy and Andrew McCallum. Toward optimal active learning through sampling estimation of error reduction. In *Proceedings of the 18th International Conference on Machine Learning*, pages 441–448. Morgan Kaufmann, 2001.

[47] Maytal Saar-Tsechansky and Foster Provost. Active sampling for class probability estimation and ranking. *Machine Learning*, 54(2):153–178, 2004.

[48] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[49] Greg Schohn and David A. Cohn. Less is more: Active learning with support vector machines. In *Proceedings of the 17th International Conference on Machine Learning*, pages 839–846. Morgan Kaufmann, 2000.

[50] H. Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the 5th Annual ACM Conference on Computational Learning Theory*, pages 287–294. ACM Press, 1992.

[51] William D. Shannon and David Banks. Combining classification trees using MLE. *Statistics in Medicine*, 18(6):727–740, 1999.

[52] Peter Turney. Bias and the quantification of stability. *Machine Learning*, 20(1-2):23–33, 1995.

[53] Paul E. Utgoff, Neil C. Berkman, and Jeffery A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.

[54] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.

[55] D. Randall Wilson and Tony R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[56] Gang Xiao, Finnegan Southey, Robert C. Holte, and Dana Wilkinson. Software testing by active learning for commercial games. In *Proceedings of the 20th International Conference on Artificial Intelligence*, pages 898–903, 2005.

Appendix A

Dataset Information

A.1 UCI Datasets

The datasets from the UCI machine learning repository [39] that were used in this thesis are listed below. Note that many of these datasets were modified for use in this work, as is explained in Section 2.5.1. The short name appearing in the thesis is displayed in bold typeface for each dataset. Also listed for each dataset is its full name, and a URL from which the data (and in many cases a detailed description) can be obtained.

abalone Abalone database.
ftp://ftp.ics.uci.edu/pub/machine-learning-databases/abalone/

anneal Annealing database.
ftp://ftp.ics.uci.edu/pub/machine-learning-databases/annealing/

australian Australian credit approval.
ftp://ftp.ics.uci.edu/pub/machine-learning-databases/statlog/australian/

breast-cancer Breast cancer (Ljubljana).
ftp://ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer/

car Car evaluation database.
ftp://ftp.ics.uci.edu/pub/machine-learning-databases/car/

contraceptive Contraceptive method choice.
ftp://ftp.ics.uci.edu/pub/machine-learning-databases/cmc/

dermatology Dermatology database.
ftp://ftp.ics.uci.edu/pub/machine-learning-databases/dermatology/

german German credit approval.
ftp://ftp.ics.uci.edu/pub/machine-learning-databases/statlog/german/

glass Glass identification database.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/glass/

heart-c Heart disease (Cleveland).

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/heart-disease/

heart-h Heart disease (Hungary).

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/heart-disease/

heart-statlog Heart disease (Statlog database).

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/statlog/heart/

hepatitis Hepatitis database.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/hepatitis/

ionosphere Ionosphere database.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/ionosphere/

kr-vs-kp Chess: King-Rook vs. King-Pawn.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/chess/king-rook-vs-king-pawn/

letter Letter recognition database.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/letter-recognition/

lymphography Lymphography (Ljubljana).

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/lymphography/

nursery Nursery database.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/nursery/

pendigits Pen-based recognition of handwritten digits.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/pendigits/

pima-indians Pima Indians diabetes database.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/pima-indians-diabetes/

segmentation Image segmentation database.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/image/

sonar Sonar: mines vs. rocks.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/undocumented/connectionist-bench/sonar/

tic-tac-toe Tic-tac-toe endgame database.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/tic-tac-toe/

vehicle Vehicle silhouettes.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/statlog/vehicle/

vote Congressional voting records database

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/voting-records/

vowel Vowel recognition

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/undocumented/connectionist-bench/vowel/

wdbc Wisconsin diagnostic breast cancer.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer-wisconsin/

wpbc Wisconsin prognostic breast cancer.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/breast-cancer-wisconsin/

yeast Protein localization sites.

ftp://ftp.ics.uci.edu/pub/machine-learning-databases/yeast/

A.2 FIFA Soccer Datasets

Two original datasets were generated using the FIFA 2004 soccer game by Electronic Arts, and were used during the experiments conducted in Chapter 4. In this appendix, these datasets, as well as the procedure that was used to create them, are described.

Each soccer dataset is derived from a different game scenario, in which a small subset of the gameplay is parameterized and then simulated repeatedly using different values for these parameters (i.e. attributes). One of two possible outcomes (i.e. classes) is recorded for each trial (i.e. instance/example), which in these scenarios is either "goal" or "miss." The *corner-kick* dataset is two-dimensional, and consists of the attributes angle and distance. During a corner kick in soccer, the ball is kicked into the field of play from a position near one of the corners of the field in the defending team's half; in a FIFA 2004 corner kick situation, the values for these two attributes specify the position on the field where the ball is aimed. After the ball is kicked, the attacking team attempts to deflect or shoot the ball into the opposing team's net, while the defenders try to clear the ball away from the net. The *give-and-go* scenario involves just two attackers, three defenders, and one goalie (on the defending team). The objective for the attacking team is to score a goal, possibly by evading the defenders through some combination of dribbling and passing, while the defending team simply tries to gain possession of the ball. Of the five attributes in the give-and-go dataset, two specify the initial configuration, namely, the positions of the players and of the ball, while the remaining three influence the attacking players' policy. Finally, the attributes in these scenarios each have predetermined minimum and maximum values. Any combination of values for the various attributes constitutes a legal example that can be labelled by the game engine. The properties of the two soccer datasets are summarized in Table A.1.

Name	Attributes	Instances	Min. Class	% Min.
corner-kick	2	66,045	goal	2.4
give-and-go	5	66,000	goal	20.6

Table A.1: Datasets generated using the FIFA 2004 soccer game. All attributes are continuous.

An example was labelled by simulating it n times in order to account for randomness within the game; this produced several binary labels per point. Before running an active learning experiment, the n outcomes for each point were reduced to a single binary label. This was achieved by applying a threshold t to each point, such that points having less than t positive outcomes (i.e. goals scored) were assigned a negative label, while those with at least t positives received a positive label. The threshold was applied to the entire dataset, including the data used for evaluation. The value of t was chosen to distinguish points associated with a high probability of scoring from those having a low probability of scoring. Although this is a subjective notion, the distribution of the raw scoring frequencies served as a guide. For the corner-kick scenario, high-scoring points were extremely rare, and so a threshold of 4 goals (out of 40 attempts) was used. On the other hand, goals were more frequent in the give-and-go scenario; for this reason, a threshold of 5 goals (out of 20 attempts) was established.

Appendix B

Estimating a C4.5 Penalty Term for the DKM Criterion

The approach taken when estimating the DKM penalty was to assume that, given a set of candidate splits S for a continuous attribute a, the DKM penalty should be calculated so as to eliminate the same percentage of splits in S from consideration (based on the DKM scores for S) that the entropy penalty removes from contention (based on the entropy scores for S). During the tree-growing procedure, let n be the number of examples at the current node, and assume that the data contains k classes. First, all the distinct labellings that could be created from the n items were generated – there are k^n such labellings. For each labelling, the procedure used by C4.5 was employed to find the threshold that maximized the information gain, and its corresponding score was recorded; this was subsequently repeated for DKM. Next, the entropy penalty $pen_{ent}(n, u) = \frac{1}{n}\log_2(u)$ was calculated for each possible number of candidate thresholds u, given n examples. The number of candidates is related to the number of unique values for a among the n examples; if there are q unique values for a, then $u = q - 1$. However, C4.5 upper bounds the value of u by a quantity t_{min} that equals the number of thresholds for which both resulting subsets contain at least *minSubsets* examples. The *minSubsets* term in C4.5 is calculated according to the following heuristic:

$$minSubsets(k, m, n) = \begin{cases} m & \text{if } \frac{n}{10k} \leq m, \\ 25 & \text{if } \frac{n}{10k} > 25, \\ \frac{n}{10k} & \text{otherwise.} \end{cases}$$

Here, m is the minimum number of examples that may constitute a subset in C4.5, as was introduced in Section 2.1; as a reminder, the default setting of $m = 2$ was used throughout this thesis. Thus, for each distinct labelling, $t_{min} = minSubsets(k, m, n)$ was calculated, and u was varied from 1 to t_{min}, yielding t_{min} different values for $pen_{ent}(n, u)$. Since no splits were possible under this scheme with $n < 4$, the lowest value of n considered was 4. Next, for a given value of u, $pen_{ent}(n, u)$ was subtracted from each of the entropy scores that were calculated from the various labellings. In doing so, the percentage of "profitable" splits $pct_{ent}(n, u)$ – those splits with a penalized gain score greater than zero – was determined. Finally, for each combination of n and u, the

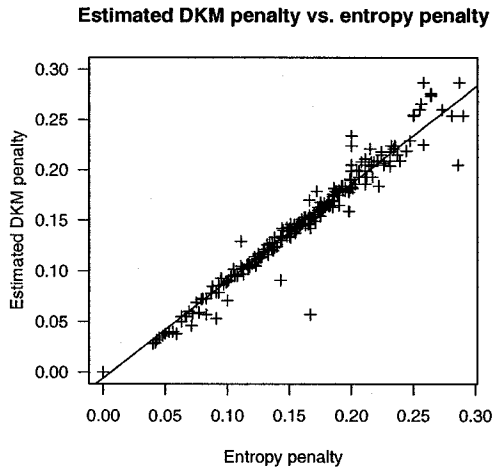

Figure B.1: Estimated DKM penalty as a function of the entropy penalty for $4 \leq n \leq 25$. The line fitted to this data by the method of least squares is also displayed.

DKM penalty $pen_{dkm}(n, u)$ was calculated so as to minimize the difference between $pct_{ent}(n, u)$ and $pct_{dkm}(n, u)$.

As n increases, generating the k^n labellings and computing their scores quickly becomes intractable. For this reason, the DKM penalty was computed for $4 \leq n \leq 25$, and the above calculation was approximated for values of n greater than 25. For $4 \leq n \leq 25$, there were 253 (n, u) combinations, corresponding to 253 paired observations of $pen_{ent}(n, u)$ and $pen_{dkm}(n, u)$ (hereafter abbreviated as pen_{ent} and pen_{dkm}, respectively). Based on this data, there was a strong indication of a linear relationship between pen_{ent} and pen_{dkm} (see Figure B.1); the method of least squares was used to fit a line to the data, according to the equation: $pen_{dkm} = x_1 \cdot pen_{ent} + x_0$. This yielded the coefficients $x_0 = 0.005887$ and $x_1 = 0.961252$, and these were subsequently used to compute the DKM penalty for $n > 25$.

To evaluate the utility of this new DKM penalty term (hereafter referred to as the estimated penalty), C4.5 was executed using the datasets listed in Tables 2.2 and 2.3, according to the repeated two-fold cross-validation procedure described in Section 2.5.2. The datasets that contained only discrete attributes (i.e. those listed in Table 2.1) were omitted; C4.5 does not apply any penalty to the score of a discrete attribute test, and so the induced decision trees would be identical regardless of the penalty term that is used. Three different penalty strategies were tested in combination with the DKM splitting criterion: 1) apply the estimated penalty, 2) apply no penalty, and 3) apply the entropy penalty. For each scheme, the error rates of the resulting pruned trees, as well as the number of leaf nodes contained in each, were recorded. The results were averaged over 10,000 random partitions of the given dataset into two equal halves.

It is important to note that increasing the magnitude of the penalty term (i.e. from no penalty,

to the estimated penalty, to the entropy penalty) does not necessarily mean that smaller trees will be produced. A larger penalty term increases the probability that the "no gain" stopping criterion is invoked, thereby pre-pruning the tree. However, it can also cause the set of final candidate splits, which consists of the best proposed split for each attribute, to be re-ranked. This latter effect can indirectly lead to an increase in tree size. For example, in the case of datasets that contained both discrete and continuous attributes, one effect of penalizing a candidate split s_c on a continuous attribute a_c is that a potential split s_d on a discrete attribute a_d may consequently score higher than s_c. If s_d becomes the best split, however, it is possible for the subtree formed below s_d to be larger than the subtree that would have been created had the split s_c been performed. This illustrates how the application of a penalty could potentially increase the size of the tree.

The implications for using a penalty on datasets containing only continuous attributes are slightly more subtle. In this case, the penalty is applied to each candidate split. Recall that the value deducted from the score of a split on a continuous attribute a_c is a function of the number of examples at the current tree node, as well as the number of unique values for a_c among these examples. If the number of unique values for two continuous attributes are not equal, then one will incur a larger penalty than the other, possibly causing the best splits for each of these attributes to be re-ranked. This can lead to the situation described in the previous example, in which a larger subtree is created because the splits are re-ranked as a result of applying the penalty. To reinforce the point made at the beginning of this paragraph, increasing the severity of the penalty can, in addition to affecting pre-pruning, produce different rankings for candidate splits and possibly lead to larger trees – there is no guarantee that tree size will decrease as a result.

Comparing the trees grown when using the estimated penalty to those induced with the penalty disabled, only minor differences in accuracy were observed (see Table B.1). The estimated penalty yielded more accurate trees on 9 of the 18 datasets, and there were 3 draws. On the remaining datasets, the difference in the error rates was at most .017. With respect to tree size, the use of the estimated penalty consistently led to less complex models. There were only 2 datasets for which the version of C4.5 with no penalty term produced smaller trees, and the average size reduction was never greater than 2.25 leaves. On the other hand, there were cases where the trees grown without any penalty were significantly larger than those induced when using the estimated penalty. The former scheme produced at least 50% more leaf nodes than the latter on 2 datasets. On the pima-indians and yeast datasets, for example, the number of extra leaf nodes in the unpenalized trees was greater than 25 and 47, respectively. Furthermore, the trees built using the estimated penalty were more accurate on both of these datasets. Based on these findings, it can be concluded that using the estimated penalty in conjunction with the DKM splitting criterion produces trees that are approximately as accurate as those grown without any penalty term, and that the estimated penalty mitigates excess tree structure, sometimes drastically.

The remaining question is whether the DKM splitting criterion performs better with the esti-

Dataset	Error rate						Number of leaf nodes					
	No pen.			Estimated pen.			No pen.			Estimated pen.		
anneal	.124	±	.017	.126	±	.017	28.4	±	5.2	25.3	±	4.8
australian	.150	±	.017	.146	±	.017	13.3	±	12.7	9.4	±	12.0
german	.288	±	.019	.291	±	.019	52.1	±	15.9	49.3	±	17.5
glass	.254	±	.053	.256	±	.055	9.1	±	2.1	8.3	±	2.3
heart-c	.255	±	.035	.238	±	.033	16.8	±	3.8	16.4	±	4.3
heart-h	.253	±	.033	.243	±	.030	13.1	±	4.9	10.9	±	5.0
heart-statlog	.257	±	.038	.240	±	.037	15.8	±	3.5	15.3	±	3.8
ionosphere	.124	±	.027	.124	±	.026	10.1	±	1.7	10.1	±	1.8
letter	.013	±	.002	.013	±	.001	73.3	±	8.5	75.5	±	8.1
lymphography	.218	±	.050	.214	±	.050	7.2	±	3.0	7.2	±	2.9
pendigits	.013	±	.002	.013	±	.002	39.4	±	4.4	39.5	±	4.3
pima-indians	.284	±	.023	.272	±	.023	38.4	±	6.0	12.8	±	6.1
segmentation	.017	±	.005	.018	±	.005	14.1	±	2.2	14.1	±	2.3
sonar	.285	±	.052	.286	±	.052	9.7	±	1.2	9.7	±	1.2
vehicle	.243	±	.021	.241	±	.022	42.3	±	4.5	29.8	±	8.8
vowel	.044	±	.012	.045	±	.012	13.4	±	1.9	13.0	±	2.1
wdbc	.069	±	.016	.068	±	.015	8.3	±	1.7	8.3	±	1.7
yeast	.271	±	.017	.254	±	.015	64.7	±	11.1	16.7	±	7.3

Table B.1: Comparison between the use of no penalty and the use of the estimated DKM penalty. The value to the right of the ± sign represents one standard deviation.

Dataset	Error rate						Number of leaf nodes					
	Entropy pen.			Estimated pen.			Entropy pen.			Estimated pen.		
anneal	.128	±	.017	.126	±	.017	24.6	±	4.8	25.3	±	4.8
australian	.145	±	.016	.146	±	.017	8.8	±	11.6	9.4	±	12.0
german	.291	±	.019	.291	±	.019	49.2	±	17.5	49.3	±	17.5
glass	.257	±	.056	.256	±	.055	8.0	±	2.3	8.3	±	2.3
heart-c	.238	±	.033	.238	±	.033	16.4	±	4.3	16.4	±	4.3
heart-h	.243	±	.030	.243	±	.030	10.7	±	5.0	10.9	±	5.0
heart-statlog	.240	±	.037	.240	±	.037	15.3	±	3.8	15.3	±	3.8
ionosphere	.124	±	.026	.124	±	.026	10.0	±	1.8	10.1	±	1.8
letter	.013	±	.001	.013	±	.001	75.6	±	8.2	75.5	±	8.1
lymphography	.213	±	.050	.214	±	.050	7.1	±	2.8	7.2	±	2.9
pendigits	.013	±	.002	.013	±	.002	39.5	±	4.3	39.5	±	4.3
pima-indians	.271	±	.023	.272	±	.023	9.3	±	5.0	12.8	±	6.1
segmentation	.018	±	.005	.018	±	.005	14.1	±	2.3	14.1	±	2.3
sonar	.286	±	.052	.286	±	.052	9.7	±	1.2	9.7	±	1.2
vehicle	.241	±	.022	.241	±	.022	23.6	±	8.6	29.8	±	8.8
vowel	.045	±	.012	.045	±	.012	12.7	±	2.3	13.0	±	2.1
wdbc	.068	±	.016	.068	±	.015	8.3	±	1.7	8.3	±	1.7
yeast	.255	±	.015	.254	±	.015	11.3	±	5.3	16.7	±	7.3

Table B.2: Comparison between the use of the entropy penalty and the use of the estimated DKM penalty. The value to the right of the ± sign represents one standard deviation.

mated penalty than with the default penalty that is traditionally applied when using the entropy criterion. This comparison is displayed in Table B.2. Again, the differences in accuracy between the two strategies were minor, with draws occurring on 13 of the datasets. Moreover, the largest difference between error rates was .002. On the issue of tree size, the entropy penalty tended to grow trees that contained fewer leaf nodes; however, the reductions in tree size were, in most cases, minimal. Of the 11 datasets for which the entropy penalty yielded less complex models, the trees differed by less than 1 leaf node on 8 of them. In contrast to the comparisons made between trees grown with the estimated penalty and those grown using no penalty, in which dramatic differences in tree size were observed, the entropy penalty trees and the estimated penalty trees never differed by more than 6.2 leaf nodes (vehicle dataset). Taking the ratios of the tree sizes into account, the largest difference occurred on the yeast dataset. Here, the trees built using the estimated penalty were approximately 1.5 times than those grown with the entropy penalty, although this constituted an increase of only 5.4 leaf nodes.

When used in conjunction with the DKM splitting criterion, the entropy penalty performed marginally better than the estimated penalty on this collection of datasets. Evidently, deducting a larger penalty term from the scores assigned to candidate continuous splits by DKM can be beneficial, as it often leads to slight reductions in tree size without degrading accuracy. However, this trend was not unanimous, as the estimated penalty sometimes produced better trees, indicating that the entropy penalty may be too severe in some cases. Although it is easily apparent from the experiments in which the penalty was disabled that failing to reduce the scores of continuous splits can be detrimental, it is not entirely clear whether the entropy penalty or the estimated penalty is more appropriate when using the DKM criterion in C4.5. Use of the estimated penalty has some justification, due to the fact that it was calibrated for use with DKM in such a way as to remove the same number of candidate splits from contention as the entropy penalty does when the computing the information gain. On the other hand, the entropy penalty seems rather arbitrary when applied to scores produced by the DKM criterion.

Appendix C

Additional Experimental Data

C.1 Data from Selective Sampling Experiments

The tables in this section contain the weighted averages of the structural stability scores that were obtained during the experiments conducted in Section 3.3, when using positive values for ϵ_{pct}. Although the scores for the car, kr-vs-kp, nursery, and tic-tac-toe datasets are unaffected by the value of ϵ_{pct} (as they contain discrete attributes only), they are included in each table for the purpose of computing average ranks for the sampling methods.

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.134 (7)	.302 (3)	.381 (1)	.375 (2)	.157 (6)	.161 (5)	.301 (4)
australian	.793 (3)	.750 (5)	.451 (7)	.534 (6)	.848 (2)	.782 (4)	.959 (1)
car	.782 (7)	.836 (4)	.821 (5)	.837 (3)	.844 (2)	.862 (1)	.790 (6)
german	.163 (7)	.289 (5)	.357 (2)	.291 (4)	.224 (6)	.344 (3)	.589 (1)
kr-vs-kp	.825 (5)	.842 (4)	.853 (3)	.854 (2)	.555 (7)	.880 (1)	.650 (6)
letter	.477 (6)	.730 (3)	.833 (1)	.778 (2)	.714 (4)	.471 (7)	.606 (5)
nursery	.841 (7)	.942 (1)	.899 (5)	.894 (6)	.935 (2)	.906 (4)	.908 (3)
pendigits	.738 (6)	.837 (1)	.799 (3)	.794 (4)	.818 (2)	.745 (5)	.661 (7)
pima-indians	.100 (5)	.234 (1)	.209 (2)	.095 (6)	.092 (7)	.168 (3)	.126 (4)
segmentation	.328 (5)	.583 (1)	.542 (3)	.189 (7)	.551 (2)	.348 (4)	.256 (6)
tic-tac-toe	.455 (1)	.254 (7)	.264 (6)	.307 (4)	.276 (5)	.333 (3)	.416 (2)
vehicle	.117 (1)	.015 (5)	.010 (6)	.067 (2)	.008 (7)	.043 (3)	.033 (4)
vowel	.114 (7)	.689 (4)	.709 (3)	.772 (1)	.739 (2)	.584 (6)	.600 (5)
wdbc	.160 (6)	.280 (4)	.348 (3)	.384 (1)	.126 (7)	.170 (5)	.356 (2)
yeast	.206 (2)	.035 (7)	.175 (4)	.064 (6)	.117 (5)	.221 (1)	.185 (3)
Avg. rank	(5.000)	(3.667)	(3.600)	(3.733)	(4.400)	(3.667)	(3.933)

Table C.1: Weighted structural FinalStab scores (entropy, $\varepsilon_{pct} = 5\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.320 (3)	.235 (5)	.153 (7)	.174 (6)	.364 (1)	.301 (4)	.342 (2)
australian	.970 (1)	.676 (7)	.722 (4)	.735 (3)	.708 (5)	.687 (6)	.816 (2)
car	.798 (5)	.819 (4)	.827 (3)	.891 (1)	.684 (7)	.731 (6)	.841 (2)
german	.467 (7)	.595 (2)	.541 (5)	.553 (3)	.513 (6)	.545 (4)	.783 (1)
kr-vs-kp	.821 (5)	.962 (2)	.967 (1)	.960 (3)	.627 (6)	.946 (4)	.616 (7)
letter	.554 (6)	.793 (4)	.880 (3)	.888 (1)	.881 (2)	.565 (5)	.457 (7)
nursery	.822 (7)	.942 (1)	.926 (4)	.872 (6)	.935 (2)	.903 (5)	.929 (3)
pendigits	.480 (5)	.786 (1)	.641 (4)	.698 (3)	.742 (2)	.338 (6)	.233 (7)
pima-indians	.114 (4)	.090 (6)	.143 (2)	.135 (3)	.104 (5)	.089 (7)	.160 (1)
segmentation	.783 (2)	.721 (3)	.411 (4)	.357 (5)	.282 (7)	.865 (1)	.283 (6)
tic-tac-toe	.449 (1)	.253 (5)	.307 (3)	.304 (4)	.245 (7)	.252 (6)	.376 (2)
vehicle	.162 (1)	.000 (7)	.022 (6)	.038 (4)	.085 (2)	.028 (5)	.056 (3)
vowel	.765 (4)	.756 (5)	.750 (6.5)	.750 (6.5)	.899 (1)	.769 (3)	.807 (2)
wdbc	.300 (6)	.362 (4)	.400 (3)	.441 (2)	.467 (1)	.352 (5)	.155 (7)
yeast	.163 (5)	.155 (6)	.255 (3)	.354 (2)	.136 (7)	.397 (1)	.221 (4)
Avg. rank	(4.133)	(4.133)	(3.900)	(3.500)	(4.067)	(4.533)	(3.733)

Table C.2: Weighted structural FinalStab scores (DKM, $\varepsilon_{pct} = 5\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.530 (5)	.435 (7)	.512 (6)	.535 (4)	.548 (2)	.541 (3)	.662 (1)
australian	.898 (3)	.902 (2)	.834 (7)	.846 (6)	.875 (5)	.892 (4)	.981 (1)
car	.933 (3)	.928 (5)	.821 (7)	.929 (4)	.941 (2)	.927 (6)	.945 (1)
german	.540 (5)	.586 (2)	.486 (7)	.541 (4)	.506 (6)	.551 (3)	.808 (1)
kr-vs-kp	.964 (2)	.965 (1)	.955 (5)	.958 (3)	.928 (6)	.957 (4)	.915 (7)
letter	.960 (6)	.981 (1)	.979 (2)	.978 (3)	.968 (5)	.969 (4)	.943 (7)
nursery	.978 (7)	.989 (2)	.988 (3.5)	.988 (3.5)	.987 (5)	.985 (6)	.993 (1)
pendigits	.913 (7)	.965 (2)	.963 (3)	.971 (1)	.957 (4)	.940 (6)	.941 (5)
pima-indians	.393 (3)	.378 (4)	.409 (2)	.315 (6)	.307 (7)	.365 (5)	.457 (1)
segmentation	.850 (3)	.924 (1)	.820 (5)	.770 (6)	.917 (2)	.849 (4)	.705 (7)
tic-tac-toe	.718 (2)	.598 (5)	.528 (6)	.523 (7)	.622 (3)	.619 (4)	.768 (1)
vehicle	.311 (1)	.112 (7)	.157 (4)	.140 (6)	.147 (5)	.274 (2)	.203 (3)
vowel	.783 (7)	.844 (3)	.850 (2)	.834 (6)	.836 (5)	.838 (4)	.858 (1)
wdbc	.592 (6)	.776 (1)	.732 (3)	.690 (4)	.631 (5)	.546 (7)	.746 (2)
yeast	.421 (1)	.255 (6)	.368 (3)	.335 (5)	.245 (7)	.410 (2)	.350 (4)
Avg. rank	(4.067)	(3.267)	(4.367)	(4.567)	(4.600)	(4.267)	(2.867)

Table C.3: Weighted structural PrevStab scores (entropy, $\varepsilon_{pct} = 5\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.743 (2)	.526 (6)	.537 (5)	.470 (7)	.684 (3)	.654 (4)	.762 (1)
australian	.988 (1)	.934 (6)	.921 (7)	.954 (2)	.943 (5)	.948 (3)	.945 (4)
car	.932 (3)	.923 (5)	.905 (7)	.948 (1)	.919 (6)	.945 (2)	.929 (4)
german	.674 (7)	.739 (2)	.689 (6)	.731 (3)	.711 (5)	.718 (4)	.897 (1)
kr-vs-kp	.960 (5)	.982 (3)	.981 (4)	.984 (2)	.868 (7)	.989 (1)	.894 (6)
letter	.963 (6)	.989 (2.5)	.991 (1)	.989 (2.5)	.983 (4)	.976 (5)	.918 (7)
nursery	.981 (7)	.989 (3)	.989 (3)	.986 (5.5)	.989 (3)	.986 (5.5)	.992 (1)
pendigits	.930 (6)	.930 (6)	.946 (2.5)	.947 (1)	.940 (4)	.946 (2.5)	.930 (6)
pima-indians	.454 (2)	.377 (3)	.361 (5)	.278 (6)	.259 (7)	.362 (4)	.498 (1)
segmentation	.910 (2)	.901 (3)	.697 (7)	.726 (6)	.764 (5)	.955 (1)	.794 (4)
tic-tac-toe	.673 (2)	.519 (7)	.547 (6)	.566 (5)	.652 (3)	.581 (4)	.720 (1)
vehicle	.409 (1)	.176 (4)	.096 (7)	.169 (6)	.174 (5)	.206 (3)	.328 (2)
vowel	.892 (6)	.926 (2)	.920 (4)	.938 (1)	.922 (3)	.853 (7)	.896 (5)
wdbc	.739 (2)	.680 (7)	.698 (5)	.718 (3)	.748 (1)	.693 (6)	.710 (4)
yeast	.417 (6)	.509 (4)	.530 (3)	.543 (1)	.282 (7)	.541 (2)	.459 (5)
Avg. rank	(3.867)	(4.233)	(4.833)	(3.467)	(4.533)	(3.600)	(3.467)

Table C.4: Weighted structural PrevStab scores (DKM, $\varepsilon_{pct} = 5\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.169 (4)	.187 (3)	.273 (2)	.314 (1)	.131 (5)	.116 (6)	.076 (7)
australian	.568 (4)	.620 (2)	.547 (6)	.500 (7)	.565 (5)	.611 (3)	.783 (1)
car	.533 (7)	.598 (5)	.617 (3)	.739 (1)	.710 (2)	.577 (6)	.601 (4)
german	.105 (7)	.203 (2)	.209 (1)	.174 (4)	.198 (3)	.160 (5)	.139 (6)
kr-vs-kp	.662 (5)	.843 (2)	.837 (3)	.870 (1)	.470 (6)	.803 (4)	.343 (7)
letter	.341 (6)	.716 (3)	.798 (1)	.726 (2)	.709 (4)	.421 (5)	.321 (7)
nursery	.767 (7)	.915 (3.5)	.927 (2)	.915 (3.5)	.933 (1)	.850 (5)	.827 (6)
pendigits	.554 (6)	.771 (3)	.775 (2)	.776 (1)	.735 (4)	.642 (5)	.381 (7)
pima-indians	.085 (4)	.139 (2)	.144 (1)	.105 (3)	.039 (7)	.079 (5)	.067 (6)
segmentation	.175 (6)	.488 (3)	.512 (2)	.264 (5)	.562 (1)	.325 (4)	.116 (7)
tic-tac-toe	.189 (5)	.203 (2)	.207 (1)	.194 (4)	.153 (7)	.198 (3)	.166 (6)
vehicle	.020 (2)	.014 (5)	.018 (3)	.024 (1)	.010 (6)	.015 (4)	.009 (7)
vowel	.263 (5)	.558 (4)	.654 (3)	.733 (1)	.663 (2)	.154 (7)	.158 (6)
wdbc	.027 (7)	.283 (1)	.265 (3)	.276 (2)	.189 (4)	.079 (6)	.083 (5)
yeast	.088 (5)	.127 (2)	.115 (4)	.130 (1)	.072 (6)	.116 (3)	.057 (7)
Avg. rank	(5.333)	(2.833)	(2.467)	(2.500)	(4.200)	(4.733)	(5.933)

Table C.5: Weighted structural RunStab scores (entropy, $\varepsilon_{pct} = 5\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.167 (4)	.171 (3)	.136 (7)	.158 (5)	.282 (1)	.198 (2)	.155 (6)
australian	.800 (1)	.686 (4)	.664 (5)	.699 (3)	.740 (2)	.661 (6)	.603 (7)
car	.542 (7)	.631 (3)	.598 (5)	.700 (1)	.649 (2)	.614 (4)	.548 (6)
german	.330 (7)	.500 (1)	.397 (4)	.461 (2)	.395 (5)	.366 (6)	.430 (3)
kr-vs-kp	.737 (5)	.920 (2)	.921 (1)	.894 (3)	.449 (7)	.865 (4)	.460 (6)
letter	.372 (6)	.714 (3)	.900 (2)	.918 (1)	.677 (4)	.438 (5)	.315 (7)
nursery	.768 (7)	.921 (3)	.926 (2)	.911 (4)	.944 (1)	.858 (5)	.841 (6)
pendigits	.258 (6)	.721 (1)	.528 (4)	.626 (3)	.700 (2)	.500 (5)	.147 (7)
pima-indians	.073 (4)	.089 (3)	.109 (2)	.110 (1)	.068 (5)	.042 (7)	.067 (6)
segmentation	.622 (3)	.689 (2)	.286 (5)	.302 (4)	.244 (6)	.729 (1)	.158 (7)
tic-tac-toe	.176 (5)	.201 (3)	.221 (2)	.223 (1)	.171 (6)	.183 (4)	.153 (7)
vehicle	.046 (1)	.026 (2)	.012 (4)	.011 (5)	.017 (3)	.009 (6)	.007 (7)
vowel	.677 (4)	.746 (2)	.736 (3)	.773 (1)	.608 (6)	.635 (5)	.568 (7)
wdbc	.083 (7)	.309 (3)	.219 (4)	.345 (2)	.458 (1)	.145 (5)	.093 (6)
yeast	.134 (5)	.214 (4)	.226 (3)	.285 (1)	.072 (7)	.236 (2)	.117 (6)
Avg. rank	(4.800)	(2.600)	(3.533)	(2.467)	(3.867)	(4.467)	(6.267)

Table C.6: Weighted structural RunStab scores (DKM, $\varepsilon_{pct} = 5\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.146 (7)	.359 (4)	.412 (2)	.460 (1)	.201 (5)	.173 (6)	.377 (3)
australian	.793 (3)	.750 (5)	.451 (7)	.534 (6)	.848 (2)	.783 (4)	.959 (1)
car	.782 (7)	.836 (4)	.821 (5)	.837 (3)	.844 (2)	.862 (1)	.790 (6)
german	.201 (7)	.402 (3)	.363 (4)	.354 (5)	.352 (6)	.420 (2)	.611 (1)
kr-vs-kp	.825 (5)	.842 (4)	.853 (3)	.854 (2)	.555 (7)	.880 (1)	.650 (6)
letter	.515 (6)	.789 (3)	.848 (1)	.804 (2)	.731 (4)	.500 (7)	.658 (5)
nursery	.841 (7)	.942 (1)	.899 (5)	.894 (6)	.935 (2)	.906 (4)	.908 (3)
pendigits	.757 (6)	.853 (1)	.809 (3)	.801 (4)	.848 (2)	.775 (5)	.725 (7)
pima-indians	.126 (5)	.253 (1)	.216 (2)	.123 (6)	.115 (7)	.189 (3)	.157 (4)
segmentation	.348 (5)	.607 (1)	.544 (3)	.200 (7)	.551 (2)	.351 (4)	.268 (6)
tic-tac-toe	.455 (1)	.254 (7)	.264 (6)	.307 (4)	.276 (5)	.333 (3)	.416 (2)
vehicle	.122 (1)	.015 (5)	.013 (6)	.067 (2)	.008 (7)	.052 (3)	.033 (4)
vowel	.185 (7)	.719 (4)	.794 (1)	.788 (2)	.780 (3)	.651 (6)	.665 (5)
wdbc	.179 (6)	.280 (4)	.351 (3)	.385 (1)	.126 (7)	.193 (5)	.360 (2)
yeast	.221 (2)	.045 (7)	.192 (4)	.084 (6)	.134 (5)	.256 (1)	.202 (3)
Avg. rank	(5.000)	(3.600)	(3.667)	(3.800)	(4.400)	(3.667)	(3.867)

Table C.7: Weighted structural FinalStab scores (entropy, $\varepsilon_{pct} = 10\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.384 (4)	.269 (5)	.185 (7)	.200 (6)	.443 (1)	.412 (3)	.427 (2)
australian	.970 (1)	.683 (7)	.752 (3)	.742 (4)	.727 (5)	.688 (6)	.816 (2)
car	.798 (5)	.819 (4)	.827 (3)	.891 (1)	.684 (7)	.731 (6)	.841 (2)
german	.482 (7)	.599 (2)	.541 (5)	.553 (4)	.513 (6)	.554 (3)	.784 (1)
kr-vs-kp	.821 (5)	.962 (2)	.967 (1)	.960 (3)	.627 (6)	.946 (4)	.616 (7)
letter	.616 (6)	.910 (3)	.911 (2)	.909 (4)	.918 (1)	.747 (5)	.571 (7)
nursery	.822 (7)	.942 (1)	.926 (4)	.872 (6)	.935 (2)	.903 (5)	.929 (3)
pendigits	.595 (5)	.796 (1)	.656 (4)	.708 (3)	.749 (2)	.353 (7)	.374 (6)
pima-indians	.116 (6)	.147 (5)	.174 (2)	.162 (3)	.148 (4)	.108 (7)	.185 (1)
segmentation	.787 (2)	.721 (3)	.411 (4)	.368 (5)	.282 (7)	.877 (1)	.284 (6)
tic-tac-toe	.449 (1)	.253 (5)	.307 (3)	.304 (4)	.245 (7)	.252 (6)	.376 (2)
vehicle	.215 (1)	.000 (7)	.024 (6)	.038 (4)	.101 (2)	.030 (5)	.057 (3)
vowel	.769 (4.5)	.769 (4.5)	.760 (7)	.762 (6)	.912 (1)	.782 (3)	.834 (2)
wdbc	.300 (6)	.362 (4)	.400 (3)	.441 (2)	.467 (1)	.352 (5)	.155 (7)
yeast	.179 (6)	.180 (5)	.386 (3)	.417 (2)	.159 (7)	.428 (1)	.271 (4)
Avg. rank	(4.433)	(3.900)	(3.800)	(3.800)	(3.933)	(4.467)	(3.667)

Table C.8: Weighted structural FinalStab scores (DKM, $\varepsilon_{pct} = 10\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.544 (6)	.487 (7)	.545 (5)	.583 (3)	.587 (2)	.577 (4)	.686 (1)
australian	.898 (3)	.902 (2)	.837 (7)	.846 (6)	.876 (5)	.893 (4)	.981 (1)
car	.933 (3)	.928 (5)	.821 (7)	.929 (4)	.941 (2)	.927 (6)	.945 (1)
german	.577 (3)	.589 (2)	.512 (7)	.563 (5)	.543 (6)	.572 (4)	.811 (1)
kr-vs-kp	.964 (2)	.965 (1)	.955 (5)	.958 (3)	.928 (6)	.957 (4)	.915 (7)
letter	.964 (6)	.984 (1)	.983 (2)	.981 (3)	.973 (4.5)	.973 (4.5)	.947 (7)
nursery	.978 (7)	.989 (2)	.988 (3.5)	.988 (3.5)	.987 (5)	.985 (6)	.993 (1)
pendigits	.928 (7)	.971 (2)	.970 (3)	.972 (1)	.961 (4.5)	.948 (6)	.961 (4.5)
pima-indians	.438 (2)	.409 (4)	.432 (3)	.349 (6.5)	.349 (6.5)	.406 (5)	.486 (1)
segmentation	.860 (3)	.932 (1)	.833 (5)	.786 (6)	.917 (2)	.850 (4)	.717 (7)
tic-tac-toe	.718 (2)	.598 (5)	.528 (6)	.523 (7)	.622 (3)	.619 (4)	.768 (1)
vehicle	.320 (1)	.131 (7)	.165 (4)	.155 (6)	.158 (5)	.303 (2)	.203 (3)
vowel	.787 (7)	.874 (4)	.884 (3)	.857 (6)	.886 (2)	.862 (5)	.902 (1)
wdbc	.604 (6)	.781 (1)	.746 (3)	.693 (4)	.631 (5)	.557 (7)	.751 (2)
yeast	.453 (1)	.270 (6)	.392 (4)	.409 (3)	.269 (7)	.418 (2)	.367 (5)
Avg. rank	(3.933)	(3.333)	(4.500)	(4.467)	(4.367)	(4.500)	(2.900)

Table C.9: Weighted structural PrevStab scores (entropy, $\varepsilon_{pct} = 10\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.773 (2)	.580 (5)	.559 (6)	.542 (7)	.736 (3)	.718 (4)	.778 (1)
australian	.988 (1)	.934 (6)	.927 (7)	.954 (2)	.943 (5)	.950 (3)	.946 (4)
car	.932 (3)	.923 (5)	.905 (7)	.948 (1)	.919 (6)	.945 (2)	.929 (4)
german	.675 (7)	.745 (2)	.691 (6)	.732 (3)	.712 (5)	.721 (4)	.897 (1)
kr-vs-kp	.960 (5)	.982 (3)	.981 (4)	.984 (2)	.868 (7)	.989 (1)	.894 (6)
letter	.967 (6)	.990 (3)	.992 (1)	.991 (2)	.986 (4)	.978 (5)	.929 (7)
nursery	.981 (7)	.989 (3)	.989 (3)	.986 (5.5)	.989 (3)	.986 (5.5)	.992 (1)
pendigits	.937 (6)	.933 (7)	.949 (2.5)	.949 (2.5)	.943 (4)	.951 (1)	.938 (5)
pima-indians	.474 (2)	.433 (3)	.421 (4)	.360 (6)	.299 (7)	.383 (5)	.519 (1)
segmentation	.911 (2)	.903 (3)	.721 (7)	.746 (6)	.764 (5)	.957 (1)	.796 (4)
tic-tac-toe	.673 (2)	.519 (7)	.547 (6)	.566 (5)	.652 (3)	.581 (4)	.720 (1)
vehicle	.420 (1)	.187 (6)	.109 (7)	.190 (5)	.191 (4)	.217 (3)	.345 (2)
vowel	.896 (6)	.932 (2)	.921 (4)	.940 (1)	.927 (3)	.858 (7)	.903 (5)
wdbc	.746 (2)	.682 (7)	.710 (5)	.719 (3)	.751 (1)	.703 (6)	.712 (4)
yeast	.470 (6)	.523 (4)	.539 (3)	.607 (1)	.308 (7)	.575 (2)	.486 (5)
Avg. rank	(3.867)	(4.400)	(4.833)	(3.467)	(4.467)	(3.567)	(3.400)

Table C.10: Weighted structural PrevStab scores (DKM, $\varepsilon_{pct} = 10\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.177 (5)	.225 (3)	.323 (2)	.385 (1)	.180 (4)	.146 (6)	.107 (7)
australian	.569 (4)	.620 (2)	.549 (6)	.500 (7)	.565 (5)	.612 (3)	.783 (1)
car	.533 (7)	.598 (5)	.617 (3)	.739 (1)	.710 (2)	.577 (6)	.601 (4)
german	.121 (7)	.221 (4)	.231 (3)	.253 (2)	.258 (1)	.213 (5)	.191 (6)
kr-vs-kp	.662 (5)	.843 (2)	.837 (3)	.870 (1)	.470 (6)	.803 (4)	.343 (7)
letter	.412 (6)	.787 (2)	.828 (1)	.755 (4)	.785 (3)	.439 (5)	.372 (7)
nursery	.767 (7)	.915 (3.5)	.927 (2)	.915 (3.5)	.933 (1)	.850 (5)	.827 (6)
pendigits	.698 (6)	.817 (1)	.796 (4)	.801 (2)	.797 (3)	.707 (5)	.447 (7)
pima-indians	.107 (4)	.157 (2)	.163 (1)	.125 (3)	.080 (7)	.104 (5)	.098 (6)
segmentation	.188 (6)	.523 (3)	.525 (2)	.274 (5)	.565 (1)	.327 (4)	.127 (7)
tic-tac-toe	.189 (5)	.203 (2)	.207 (1)	.194 (4)	.153 (7)	.198 (3)	.166 (6)
vehicle	.022 (2)	.019 (5)	.021 (3)	.026 (1)	.011 (6)	.020 (4)	.010 (7)
vowel	.440 (5)	.673 (4)	.699 (2)	.745 (1)	.694 (3)	.237 (7)	.287 (6)
wdbc	.030 (7)	.287 (1)	.272 (3)	.285 (2)	.193 (4)	.089 (6)	.097 (5)
yeast	.127 (5)	.151 (2)	.150 (3)	.173 (1)	.108 (6)	.133 (4)	.083 (7)
Avg. rank	(5.400)	(2.767)	(2.600)	(2.567)	(3.933)	(4.800)	(5.933)

Table C.11: Weighted structural RunStab scores (entropy, $\varepsilon_{pct} = 10\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.186 (6)	.208 (4)	.166 (7)	.190 (5)	.313 (1)	.231 (2)	.224 (3)
australian	.804 (1)	.697 (4)	.694 (5)	.734 (3)	.745 (2)	.661 (6)	.604 (7)
car	.542 (7)	.631 (3)	.598 (5)	.700 (1)	.649 (2)	.614 (4)	.548 (6)
german	.340 (7)	.513 (1)	.397 (4)	.461 (2)	.396 (5)	.375 (6)	.432 (3)
kr-vs-kp	.737 (5)	.920 (2)	.921 (1)	.894 (3)	.449 (7)	.865 (4)	.460 (6)
letter	.417 (6)	.856 (4)	.922 (2)	.930 (1)	.893 (3)	.598 (5)	.352 (7)
nursery	.768 (7)	.921 (3)	.926 (2)	.911 (4)	.944 (1)	.858 (5)	.841 (6)
pendigits	.323 (6)	.733 (1)	.543 (4)	.639 (3)	.713 (2)	.521 (5)	.193 (7)
pima-indians	.095 (6)	.116 (4)	.129 (2)	.132 (1)	.128 (3)	.073 (7)	.099 (5)
segmentation	.626 (3)	.689 (2)	.297 (5)	.306 (4)	.246 (6)	.736 (1)	.160 (7)
tic-tac-toe	.176 (5)	.201 (3)	.221 (2)	.223 (1)	.171 (6)	.183 (4)	.153 (7)
vehicle	.054 (1)	.031 (2)	.016 (5)	.017 (4)	.024 (3)	.011 (6)	.010 (7)
vowel	.730 (6)	.777 (4)	.790 (1)	.786 (2)	.783 (3)	.733 (5)	.688 (7)
wdbc	.109 (6)	.310 (3)	.220 (4)	.347 (2)	.460 (1)	.153 (5)	.103 (7)
yeast	.193 (5)	.236 (4)	.275 (2)	.327 (1)	.095 (7)	.264 (3)	.170 (6)
Avg. rank	(5.133)	(2.933)	(3.400)	(2.467)	(3.467)	(4.533)	(6.067)

Table C.12: Weighted structural RunStab scores (DKM, $\varepsilon_{pct} = 10\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.191 (7)	.384 (4)	.459 (2)	.557 (1)	.227 (6)	.241 (5)	.427 (3)
australian	.793 (3)	.750 (5)	.451 (7)	.534 (6)	.848 (2)	.783 (4)	.959 (1)
car	.782 (7)	.836 (4)	.821 (5)	.837 (3)	.844 (2)	.862 (1)	.790 (6)
german	.205 (7)	.402 (3)	.370 (4)	.366 (5)	.360 (6)	.427 (2)	.637 (1)
kr-vs-kp	.825 (5)	.842 (4)	.853 (3)	.854 (2)	.555 (7)	.880 (1)	.650 (6)
letter	.529 (6)	.800 (3)	.862 (1)	.817 (2)	.743 (4)	.503 (7)	.685 (5)
nursery	.841 (7)	.942 (1)	.899 (5)	.894 (6)	.935 (2)	.906 (4)	.908 (3)
pendigits	.766 (6)	.858 (1)	.811 (3)	.804 (4)	.850 (2)	.784 (5)	.734 (7)
pima-indians	.167 (5)	.281 (1)	.234 (2)	.144 (6)	.141 (7)	.204 (4)	.212 (3)
segmentation	.381 (4)	.621 (1)	.544 (3)	.200 (7)	.551 (2)	.372 (5)	.271 (6)
tic-tac-toe	.455 (1)	.254 (7)	.264 (6)	.307 (4)	.276 (5)	.333 (3)	.416 (2)
vehicle	.149 (1)	.015 (5)	.013 (6)	.070 (3)	.008 (7)	.071 (2)	.033 (4)
vowel	.186 (7)	.753 (4)	.847 (3)	.852 (2)	.865 (1)	.653 (6)	.679 (5)
wdbc	.179 (6)	.280 (4)	.352 (3)	.385 (1)	.126 (7)	.193 (5)	.366 (2)
yeast	.255 (2)	.066 (7)	.199 (4)	.096 (6)	.144 (5)	.269 (1)	.219 (3)
Avg. rank	(4.933)	(3.600)	(3.800)	(3.867)	(4.333)	(3.667)	(3.800)

Table C.13: Weighted structural FinalStab scores (entropy, $\varepsilon_{pct} = 15\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.465 (4)	.308 (5)	.226 (7)	.234 (6)	.555 (1)	.503 (3)	.535 (2)
australian	.971 (1)	.683 (7)	.752 (3)	.742 (4)	.727 (5)	.688 (6)	.816 (2)
car	.798 (5)	.819 (4)	.827 (3)	.891 (1)	.684 (7)	.731 (6)	.841 (2)
german	.482 (7)	.599 (2)	.542 (5)	.553 (4)	.515 (6)	.555 (3)	.784 (1)
kr-vs-kp	.821 (5)	.962 (2)	.967 (1)	.960 (3)	.627 (6)	.946 (4)	.616 (7)
letter	.631 (6.5)	.918 (2)	.914 (3)	.911 (4)	.924 (1)	.755 (5)	.631 (6.5)
nursery	.822 (7)	.942 (1)	.926 (4)	.872 (6)	.935 (2)	.903 (5)	.929 (3)
pendigits	.674 (4)	.801 (1)	.670 (5)	.718 (3)	.763 (2)	.363 (7)	.395 (6)
pima-indians	.144 (6)	.182 (3)	.203 (1)	.179 (4)	.172 (5)	.135 (7)	.202 (2)
segmentation	.791 (2)	.722 (3)	.483 (4)	.372 (5)	.307 (6)	.881 (1)	.286 (7)
tic-tac-toe	.449 (1)	.253 (5)	.307 (3)	.304 (4)	.245 (7)	.252 (6)	.376 (2)
vehicle	.236 (1)	.000 (7)	.028 (6)	.038 (4)	.108 (2)	.030 (5)	.057 (3)
vowel	.773 (5)	.780 (4)	.762 (7)	.763 (6)	.915 (1)	.791 (3)	.841 (2)
wdbc	.300 (6)	.367 (5)	.406 (3)	.449 (2)	.467 (1)	.374 (4)	.185 (7)
yeast	.199 (5.5)	.199 (5.5)	.390 (3)	.433 (1)	.164 (7)	.431 (2)	.275 (4)
Avg. rank	(4.400)	(3.767)	(3.867)	(3.800)	(3.933)	(4.467)	(3.767)

Table C.14: Weighted structural FinalStab scores (DKM, $\varepsilon_{pct} = 15\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.603 (4)	.506 (7)	.593 (6)	.634 (2)	.625 (3)	.598 (5)	.711 (1)
australian	.898 (3)	.902 (2)	.837 (7)	.846 (6)	.876 (5)	.894 (4)	.981 (1)
car	.933 (3)	.928 (5)	.821 (7)	.929 (4)	.941 (2)	.927 (6)	.945 (1)
german	.585 (3)	.590 (2)	.516 (7)	.567 (5)	.548 (6)	.573 (4)	.814 (1)
kr-vs-kp	.964 (2)	.965 (1)	.955 (5)	.958 (3)	.928 (6)	.957 (4)	.915 (7)
letter	.967 (6)	.985 (1)	.984 (2)	.982 (3)	.974 (4.5)	.974 (4.5)	.952 (7)
nursery	.978 (7)	.989 (2)	.988 (3.5)	.988 (3.5)	.987 (5)	.985 (6)	.993 (1)
pendigits	.935 (7)	.971 (2)	.970 (3)	.975 (1)	.963 (5)	.950 (6)	.965 (4)
pima-indians	.450 (3)	.429 (5)	.453 (2)	.375 (6)	.374 (7)	.432 (4)	.502 (1)
segmentation	.869 (3.5)	.937 (1)	.847 (5)	.791 (6)	.931 (2)	.869 (3.5)	.720 (7)
tic-tac-toe	.718 (2)	.598 (5)	.528 (6)	.523 (7)	.622 (3)	.619 (4)	.768 (1)
vehicle	.331 (1)	.138 (7)	.165 (4)	.161 (5.5)	.161 (5.5)	.325 (2)	.215 (3)
vowel	.803 (7)	.875 (4)	.885 (3)	.869 (5)	.896 (2)	.865 (6)	.904 (1)
wdbc	.606 (6)	.795 (1)	.766 (2)	.729 (4)	.651 (5)	.558 (7)	.751 (3)
yeast	.470 (1)	.292 (6)	.404 (4)	.419 (3)	.280 (7)	.435 (2)	.371 (5)
Avg. rank	(3.900)	(3.400)	(4.433)	(4.267)	(4.533)	(4.533)	(2.933)

Table C.15: Weighted structural PrevStab scores (entropy, $\varepsilon_{pct} = 15\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.793 (2)	.630 (5)	.611 (6)	.567 (7)	.756 (3)	.743 (4)	.818 (1)
australian	.988 (1)	.934 (6)	.927 (7)	.954 (2)	.943 (5)	.950 (3)	.946 (4)
car	.932 (3)	.923 (5)	.905 (7)	.948 (1)	.919 (6)	.945 (2)	.929 (4)
german	.676 (7)	.745 (2)	.692 (6)	.732 (3)	.714 (5)	.723 (4)	.897 (1)
kr-vs-kp	.960 (5)	.982 (3)	.981 (4)	.984 (2)	.868 (7)	.989 (1)	.894 (6)
letter	.969 (6)	.991 (3)	.992 (1.5)	.992 (1.5)	.987 (4)	.980 (5)	.933 (7)
nursery	.981 (7)	.989 (3)	.989 (3)	.986 (5.5)	.989 (3)	.986 (5.5)	.992 (1)
pendigits	.940 (6)	.934 (7)	.949 (3)	.952 (2)	.946 (4)	.953 (1)	.941 (5)
pima-indians	.484 (2)	.452 (3)	.432 (4)	.381 (6)	.317 (7)	.411 (5)	.543 (1)
segmentation	.921 (2)	.905 (3)	.750 (7)	.769 (6)	.809 (4)	.962 (1)	.800 (5)
tic-tac-toe	.673 (2)	.519 (7)	.547 (6)	.566 (5)	.652 (3)	.581 (4)	.720 (1)
vehicle	.438 (1)	.194 (6)	.134 (7)	.195 (5)	.198 (4)	.225 (3)	.349 (2)
vowel	.900 (6)	.933 (2)	.922 (4)	.941 (1)	.928 (3)	.861 (7)	.906 (5)
wdbc	.746 (2)	.684 (7)	.715 (4)	.720 (3)	.753 (1)	.708 (6)	.712 (5)
yeast	.477 (6)	.534 (4)	.549 (3)	.621 (1)	.314 (7)	.584 (2)	.495 (5)
Avg. rank	(3.867)	(4.400)	(4.833)	(3.400)	(4.400)	(3.567)	(3.533)

Table C.16: Weighted structural PrevStab scores (DKM, $\varepsilon_{pct} = 15\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.237 (4)	.273 (3)	.371 (2)	.449 (1)	.224 (5)	.206 (6)	.124 (7)
australian	.570 (4)	.620 (2)	.549 (6)	.500 (7)	.565 (5)	.613 (3)	.783 (1)
car	.533 (7)	.598 (5)	.617 (3)	.739 (1)	.710 (2)	.577 (6)	.601 (4)
german	.122 (7)	.251 (3)	.233 (5)	.267 (2)	.269 (1)	.241 (4)	.219 (6)
kr-vs-kp	.662 (5)	.843 (2)	.837 (3)	.870 (1)	.470 (6)	.803 (4)	.343 (7)
letter	.451 (5)	.810 (2)	.834 (1)	.765 (4)	.796 (3)	.447 (6)	.408 (7)
nursery	.767 (7)	.915 (3.5)	.927 (2)	.915 (3.5)	.933 (1)	.850 (5)	.827 (6)
pendigits	.707 (6)	.819 (1)	.801 (4)	.805 (2)	.802 (3)	.716 (5)	.474 (7)
pima-indians	.115 (5)	.198 (1)	.179 (2)	.158 (3)	.099 (7)	.126 (4)	.113 (6)
segmentation	.214 (6)	.567 (2)	.537 (3)	.278 (5)	.569 (1)	.345 (4)	.131 (7)
tic-tac-toe	.189 (5)	.203 (2)	.207 (1)	.194 (4)	.153 (7)	.198 (3)	.166 (6)
vehicle	.028 (1)	.019 (5)	.025 (3)	.027 (2)	.012 (7)	.023 (4)	.013 (6)
vowel	.464 (5)	.683 (4)	.717 (3)	.760 (1)	.730 (2)	.249 (7)	.301 (6)
wdbc	.032 (7)	.293 (1)	.274 (3)	.288 (2)	.199 (4)	.093 (6)	.100 (5)
yeast	.134 (5)	.173 (2)	.159 (3)	.182 (1)	.113 (6)	.140 (4)	.099 (7)
Avg. rank	(5.267)	(2.567)	(2.933)	(2.633)	(4.000)	(4.733)	(5.867)

Table C.17: Weighted structural RunStab scores (entropy, $\varepsilon_{pct} = 15\%$)

Dataset	Random	QBag	QBoost	QBoostC	BootLV	BootM	Uncert.
anneal	.287 (4)	.239 (6)	.209 (7)	.269 (5)	.397 (1)	.303 (2)	.295 (3)
australian	.804 (1)	.697 (4)	.695 (5)	.734 (3)	.745 (2)	.663 (6)	.604 (7)
car	.542 (7)	.631 (3)	.598 (5)	.700 (1)	.649 (2)	.614 (4)	.548 (6)
german	.340 (7)	.513 (1)	.397 (5)	.461 (2)	.400 (4)	.376 (6)	.432 (3)
kr-vs-kp	.737 (5)	.920 (2)	.921 (1)	.894 (3)	.449 (7)	.865 (4)	.460 (6)
letter	.439 (6)	.870 (4)	.925 (2)	.931 (1)	.897 (3)	.607 (5)	.412 (7)
nursery	.768 (7)	.921 (3)	.926 (2)	.911 (4)	.944 (1)	.858 (5)	.841 (6)
pendigits	.343 (6)	.744 (1)	.557 (4)	.652 (3)	.724 (2)	.531 (5)	.201 (7)
pima-indians	.105 (6)	.136 (4)	.157 (2)	.158 (1)	.142 (3)	.089 (7)	.123 (5)
segmentation	.641 (3)	.690 (2)	.338 (4)	.318 (5)	.275 (6)	.746 (1)	.206 (7)
tic-tac-toe	.176 (5)	.201 (3)	.221 (2)	.223 (1)	.171 (6)	.183 (4)	.153 (7)
vehicle	.072 (1)	.032 (2)	.019 (5)	.021 (4)	.031 (3)	.012 (6.5)	.012 (6.5)
vowel	.754 (6)	.801 (3)	.800 (4)	.802 (2)	.838 (1)	.758 (5)	.714 (7)
wdbc	.110 (7)	.312 (3)	.227 (4)	.350 (2)	.461 (1)	.164 (5)	.127 (6)
yeast	.199 (5)	.249 (4)	.281 (2)	.336 (1)	.101 (7)	.272 (3)	.174 (6)
Avg. rank	(5.067)	(3.000)	(3.600)	(2.533)	(3.267)	(4.567)	(5.967)

Table C.18: Weighted structural RunStab scores (DKM, $\varepsilon_{pct} = 15\%$)

C.2 Data from Query Learning Experiments

Table C.19 displays the weighted average error rates of the six active learning methods that were compared in Section 4.6.2. These included two example-based selective samplers (BootM, QBag), two region-based selective samplers (BootM-rb-min, QBag-rb-min), and two region-based query learners (BootM-query, QBag-query).

Dataset	BootM	QBag	BootM-rb-min	QBag-rb-min	BootM-query	QBag-query
anneal	.119 (3)	.112 (1)	.118 (2)	.126 (4)	.194 (6)	.185 (5)
australian	.129 (2)	.128 (1)	.133 (3)	.138 (5)	.166 (6)	.137 (4)
car	.234 (5)	.217 (4)	.244 (6)	.211 (3)	.195 (2)	.159 (1)
corner-kick	.008 (5.5)	.008 (5.5)	.007 (3.5)	.007 (3.5)	.006 (1.5)	.006 (1.5)
german	.278 (1.5)	.282 (3)	.290 (4)	.278 (1.5)	.300 (5)	.302 (6)
give-and-go	.196 (4)	.194 (2.5)	.194 (2.5)	.193 (1)	.199 (5)	.205 (6)
letter	.015 (2)	.013 (1)	.020 (4)	.017 (3)	.024 (5)	.026 (6)
nursery	.321 (5)	.314 (3.5)	.323 (6)	.314 (3.5)	.309 (2)	.224 (1)
pendigits	.012 (3)	.012 (3)	.012 (3)	.011 (1)	.030 (5.5)	.030 (5.5)
pima-indians	.297 (4)	.282 (1)	.287 (2)	.291 (3)	.328 (6)	.315 (5)
segmentation	.023 (4)	.018 (1)	.022 (2.5)	.022 (2.5)	.058 (6)	.055 (5)
tic-tac-toe	.499 (5)	.485 (3)	.483 (2)	.488 (4)	.500 (6)	.424 (1)
vehicle	.226 (2)	.220 (1)	.227 (3)	.233 (4)	.277 (5)	.278 (6)
vowel	.046 (4)	.037 (2)	.038 (3)	.036 (1)	.084 (6)	.080 (5)
wdbc	.062 (3)	.056 (1)	.061 (2)	.063 (4)	.121 (5)	.131 (6)
yeast	.249 (4)	.245 (3)	.243 (2)	.242 (1)	.273 (5)	.276 (6)
Avg. rank	(3.562)	(2.281)	(3.156)	(2.812)	(4.812)	(4.375)

Table C.19: Weighted error rates.