INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
INSTABILITY OF DECISION TREE CLASSIFICATION ALGORITHMS

BY

RUEY-HSIA LI

B.S., Chung Yang Christian University, 1990
M.S., University of Illinois at Urbana-Champaign, 1992

THESIS

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2001

Urbana, Illinois
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
GRADUATE COLLEGE

JULY 2001

date

WE HEREBY RECOMMEND THAT THE THESIS BY

RUEY-HSIA LI

ENTITLED INSTABILITY OF DECISION TREE CLASSIFICATION ALGORITHMS

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY

Committee on Final Examination†

†Required for doctor’s degree but not for master’s.

O-517
Abstract

Fundamental theorems are derived for the instability problem of decision tree classification algorithms. The instability problem of decision tree classification algorithms is that small changes in input training samples may cause dramatically large changes in the output tree classifiers. The past research emphasized the instability of the prediction but not the tree structure change, which is more important to provide consistent, stable, and insightful information to facilitate the process of decision making.

We present theorems to prove the relationship between a data change and the resulting tree structure change (i.e., split change). The relative sensitivity between two splits is defined based on the theorems as the smallest change that may cause the superior split to become inferior. A split is defined to be almost as good as another split if the relative sensitivity of the two splits is small. The Instability Theorem provides the cause of the instability problem. Algorithms are presented to lessen the instability problem.

Empirical results illustrate that the trees constructed by the proposed algorithm are more stable, noise-tolerant, informative, expressive, and concise. The proposed sensitivity measure can be used as a metric to evaluate the stability of splitting predicates. The tree sensitivity is an indicator of the confidence level in rules and the effective lifetime of rules.
To my parents, my husband, and my son.
Acknowledgments

I would like to express my great thanks to my advisor Geneva Belford for providing invaluable guidance and encouragement during this research work. I also thank my committee members George Friedman, Mehdi Harandi, Faisal Saied, and Marianne Winslett for their time and suggestions.

I thank my parents for their support. My office mate and husband Chao-Ming Wang for proofreading the thesis and for working at home to help take care of our son during the last phase of my graduate study. My son James for being cooperative and for being the joy of my life. Finally I would like to thank Mike Folk for providing me the opportunity to work as a research assistant at HDF in NCSA.
# Table of Contents

Chapter 1  Introduction .................................................. 1  
   1.1  An Overview of Tree Classification Algorithms .................. 2  
   1.2  Problems with Tree Classifiers: Instability of Rules and Information Loss .... 4  

Chapter 2  Related Work .................................................. 9  
   2.1  Previous Work ....................................................... 9  
      2.1.1  Bagging Classification Trees .................................. 9  
      2.1.2  CART and CART 4.0 Pro ...................................... 10  
      2.1.3  ID3, C4.5, and C5.0 ........................................... 10  
      2.1.4  IC ............................................................. 11  
      2.1.5  SLIQ .......................................................... 11  
      2.1.6  SPRINT ....................................................... 12  
      2.1.7  RainForest .................................................... 12  
      2.1.8  BOAT ........................................................ 13  
      2.1.9  Meta Learning ................................................ 14  
      2.1.10  Classifier in SONAR (System for Optimized Numeric Association Rules) .... 15  
      2.1.11  Decision Graphs .............................................. 15  
   2.2  Summary ......................................................... 16  

Chapter 3  Instability Theorem .......................................... 18  
   3.1  Split Selection Methods .......................................... 18  
   3.2  The Instability Problem with Decision Tree Classifiers ............... 20  
      3.2.1  The Instability Problem ..................................... 20  
      3.2.2  Assumptions and Parameters .................................. 21  
      3.2.3  Three Lemmas ................................................ 22  
      3.2.4  Theorem 1 – A function $\varepsilon$ that measures the closeness of $t_0$ and $t$ 29  
      3.2.5  Almost Equally Good Splits .................................. 32  
      3.2.6  Theorem 2 – Instability Theorem .............................. 33  
      3.2.7  Theorem 3 – A general function $\varepsilon$ to measure the closeness of $t_0$ to $t$ 34  
   3.3  Summary .......................................................... 39  

Chapter 4  A Stable Decision Tree Classifier .......................... 41  
   4.1  The Sensitivity of a Decision Tree Classifier ...................... 42  
      4.1.1  The Sensitivity of the Split of a Node ........................ 43  
      4.1.2  An Example ................................................... 46  
   4.2  Building a Stable Decision Tree Classifier ....................... 49  
      4.2.1  Almost Equally Good Splits .................................. 50  

vi
# List of Tables

3.1 Descriptions of the parameters .................................................. 24

4.1 The computed sensitivities of the best split “\# of Years ≤ 3” to splits “\# of Years ≤ 1.5”, “\# of Years ≤ 6”, and “sports car” ............................................. 46

5.1 The synthetic data sets and the evaluation classification rules for the first experiment ................................................................. 61

5.2 The information about the goodness measures of the split candidates, and stability of the splitting predicates at the root node before and after a change made to 1.5% of the sample .................................................. 72
List of Figures

1.1 An example of a training sample and the corresponding tree classifier .... 2
1.2 Standard tree growth algorithm ........................................... 4
1.3 The changed sample and the corresponding tree classifier ................ 5
1.4 An example of different rules resulting from pruning two logically equivalent trees . 7

3.1 Trees grown on $S$ and $S'$ differ in the subtree $T_2$ and $T'_2$ ........... 21
3.2 Parameters to evaluate $t_0$ and $t$ at the root nodes of $T_2$ and $T'_2$ ..... 23
3.3 The class distributions before and after change ............................ 26
3.4 The effect of the change $c$ to the class distributions in $D_{t_0,L}$, $D_{t_0,R}$, $D_{t,L}$, and $D_{t,R}$ . 28
3.5 Possible changes for the general model .................................... 34

4.1 Algorithm to compute the sensitivity of a tree ............................. 42
4.2 Algorithm to compute the sensitivity of $t_0$ ............................... 43
4.3 Algorithm to compute the sensitivity of $t_0$ to $t$ ......................... 45
4.4 An example of the computed change that will make the split "# of Years $\leq 1.5$" superior to the split "# of Years $\leq 3$" in the changed sample. ................. 48
4.5 Algorithm to identify splits that are almost as good as $t_0$ with respect to the change fraction $c$ ................................................. 50
4.6 Revised algorithm to identify splits that are almost as good as $t_0$ with respect to the change fraction $c$ ................................................. 51
4.7 An example of the possible splitting predicates formed by two almost equally good splits .................................................. 54
4.8 Algorithm of forming the splitting predicate at a node ..................... 55
4.9 A Stable Tree Growth Algorithm ............................................ 56
4.10 The goodness of the basis predicates and the grown tree .................. 58

5.1 The classifiers grown on data set 1 with rule $A_1 \land A_2$; $A_1$ and $A_2$ are identified as almost equally good splits ....................................... 64
5.2 The classifiers grown on data set 2 with rule $A_1 \land A_2 \land A_3 \land A_4$; $A_1$, $A_2$, $A_3$, and $A_4$ are identified as almost equally good splits ....................................... 65
5.3 The classifiers grown on data set 3 with rule $A_1 \lor A_2$; $A_1$ and $A_2$ are identified as almost equally good splits ....................................... 66
5.4 The classifiers grown on data set 4 with rule $A_1 \lor A_2 \lor A_3 \lor A_4$; $A_1$, $A_2$, $A_3$ and $A_4$ are identified as almost equally good splits ....................................... 67
5.5 The classifiers grown on data set 5 with rule $A_1 \lor (A_2 \land A_3)$; no splits are almost as good as $A_1$ ................................................. 68
5.6 The classifiers grown on data set 6 with rule $(A_1 \land A_2) \lor (A_3 \land A_4)$; $A_3$ and $A_4$ are identified as almost equally good splits ....................................... 69
5.7 Comparison of tree classifiers for the rule from [AIS93], \((A_1 \land (A_2 \lor A_3)) \lor (A_4 \land (A_3 \lor A_5)) \lor (A_6 \land (A_2 \lor A_7))\) ................................................................. 70
5.8 Tree construction times for data sets of various sizes for Rule 2 (Fig. 5.2) ............... 71
5.9 Tree construction times for data sets of various sizes for Rule 7 (Fig. 5.7) ............... 71
5.10 Stability comparison of the two algorithms ................................................................. 73
5.11 The accuracy comparison of the two algorithms for Experiment II ............................ 75
5.12 The split “\(A_2 = 1\)”, which is almost as good as “\(A_{10} = 1\)”, is not discovered by CART, but is discovered by the proposed algorithm .................................................... 76
Chapter 1

Introduction

Data mining is a process of discovering interesting and unexpected patterns/relationships in large amounts of data. One important application of data mining is to classify the objects in a database [AIS93, SAM96]. In classification, we are given a training sample. Each object in the training sample belongs to a class and is described by a set of attributes. The objective of classification is to construct classification rules to classify objects based on the descriptive attributes. The rules can then be used to assist in decision making or make predictions on the class to which future unclassified objects belong. Figure 1.1(a) shows an example of a training sample. In this example, each customer of an auto insurance company is described by two attributes, the number of years of driving experience and whether the insured vehicle is a sports car or not. The customers are categorized into either high risk or low risk. Based on the training sample, classification rules are built. The rules are then used to determine the risk of future applicants. The applications of classification have been used in fraud detection, chemistry, finance, medical diagnosis, etc.[DF95].

Several classification models have been proposed over the years, e.g. neural networks, statistical models like linear discriminants, and decisions trees. Among these models, decision trees are particularly suited for data mining [AIS93, MAR96, SAM96] for the following reasons. Tree classifiers can be constructed relatively fast [MAR96, SAM96, GRG98, GGR+99] and obtain similar or better accuracy [SAM96, GRG98, GGR+99] compared to other methods. The intuitive representation of classification rules is simple and easy to understand [BFOS84]. Moreover, the tree-structured rules graphically display the relationships found in data. The factors in the rules are prioritized; that is, the most significant factors appear at the top levels of the tree. In addition, the rules can be
<table>
<thead>
<tr>
<th># of Years</th>
<th>Sports Car</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>yes</td>
<td>high</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>high</td>
</tr>
<tr>
<td>2</td>
<td>yes</td>
<td>high</td>
</tr>
<tr>
<td>4</td>
<td>no</td>
<td>low</td>
</tr>
<tr>
<td>8</td>
<td>no</td>
<td>low</td>
</tr>
</tbody>
</table>

(a) Training sample

(b) Classification rule

Figure 1.1: An example of a training sample and the corresponding tree classifier converted to SQL statements [AGI+92] that can be used to access a database efficiently.

In the following, we provide an overview of the standard tree classification algorithms, and then describe the problems with tree-structured classifiers.

1.1 An Overview of Tree Classification Algorithms

A tree classification algorithm takes an input training sample and constructs classification rules in a tree structure. The input training sample contains a set of objects, and each object is represented by a set of object-description attributes and associated with a class label. A standard tree classification algorithm consists of two phases: a tree growth phase followed by a pruning phase. A large tree is first grown to fit the training sample closely and is then pruned by removing parts that are predicted to have a relatively high error rate.

In the tree growth phase, a tree is constructed by recursively dividing the data associated with a node until all partitions are homogeneous (i.e., the objects in each partition are in the same class)
or relatively small. Initially, the root node contains all the training objects. To determine which attribute to use to partition the data, the algorithm analyzes the data and generates a set of split candidates with respect to all the descriptive attributes. Each split candidate represents a way to partition the data. Then, a greedy split selection method chooses the split candidate that has the most tendency to divide the objects into homogeneous subsets. We will discuss the split selection method in more detail in Chapter 3. The process repeats for each partition until all the partitions are homogeneous or relatively small. The standard tree growth algorithm is shown in Figure 1.2.

A split is a test to partition the data associated with a node. An attribute is numerical if its domain is continuous, and categorical if its domain is discrete. For numeric attributes, a split is of the form “attribute ≤ value”, where value ∈ domain(attribute). For categorical attributes, a split is of the form “attribute.value ∈ subset”, where subset ⊆ domain(attribute).

To illustrate a tree classifier, consider again the auto insurance sample in Figure 1.1(a). Assume that four split candidates are generated at the root, “# of years ≤ 1.5 ”, “# of years ≤ 3”, “# of years ≤ 6”, and “sports car”. Since only the split “# of years ≤ 3” divides the training sample into homogeneous subsets, the resultant rule is “if the number of years of driving experience is greater than three years, then the risk is low; otherwise, the risk is high”. The tree classifier is shown in Figure 1.1(b).

With the presence of noise in data and to prevent overfitting the training sample, the large tree grown in the first phase needs to be pruned to result in a more accurate tree. Smaller trees also make classification more efficient. The pruning process can be done by generating a set of subtrees of the tree grown in the first phase and picking the subtree with the lowest misclassification rate [BFOS84, Quin87, MRA95].

Once the final tree is generated, the classification rules are derived by tracing the path from the root to each leaf node.
GrowTree (Data $S$) {
    if ($S$ is homogeneous)
        return;
    $split\_candidates = \{\};$
    for (each attribute $A$)
        $split\_candidates = split\_candidates \cup$
        Splits ($A$);
    $test = $ BestSplit ($split\_candidates$);
    $subsets = $ Partition ($S$, $test$);
    for (each subset $S_i \in subsets$)
        GrowTree ($S_i$);
}

Figure 1.2: Standard tree growth algorithm

The performance of tree classifiers is evaluated by the accuracy of the discovered rules, classifier construction time, and tree size. The accuracy of classification rules is the percentage of the test objects that are correctly classified. Thus, the design goal for tree classifiers is to develop a fast algorithm that generates an accurate and compact tree.

In the next section, we will describe the problems with tree classifiers.

1.2 Problems with Tree Classifiers: Instability of Rules and Information Loss

One of the problems with tree classifiers is that they may be sensitive to small changes in training samples. Tree classification algorithms may construct rules significantly different from the original ones if a small change is made to a training sample. In other words, the rules generated from two slightly different (or almost the same) samples may be very different. A change can be an update, deletion, or addition of objects in the training sample.

To demonstrate the concept of the unstable behavior of tree-structured classifiers, consider the training sample in Figure 1.1(a). If the second record in the training sample is replaced by $<4,$
### Table 1.1

<table>
<thead>
<tr>
<th># of Years</th>
<th>Sports Car</th>
<th>Risk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>yes</td>
<td>high</td>
</tr>
<tr>
<td>4</td>
<td>yes</td>
<td>high</td>
</tr>
<tr>
<td>2</td>
<td>yes</td>
<td>high</td>
</tr>
<tr>
<td>4</td>
<td>no</td>
<td>low</td>
</tr>
<tr>
<td>8</td>
<td>no</td>
<td>low</td>
</tr>
</tbody>
</table>

(a) Training sample

![Diagram of a classification rule]

(b) Classification rule

Figure 1.3: The changed sample and the corresponding tree classifier

yes, high>, then the rule for the high risk class changes from “(# of years ≤ 3) → high risk” to “sports car → high risk”. The changed sample and the corresponding tree are shown in Figure 1.3. In the changed sample, the split “(# of years ≤ 3)” which is the best split in the original sample is no longer the best choice because one of the partitions is not homogeneous. On the other hand, the split “sports car”, which is inferior to “(# of years ≤ 3)” in the original sample, partitions the data into two homogeneous subsets. Since none of the split candidates associated with attribute # of years are superior to the split “sports car”, the split “sports car” is used to partition the sample. Hence, different rules are constructed.

We have shown that one record change to the sample results in different rules. However, one record change in this example is not a small change – it is equivalent to 20% of the sample change. We use it mainly for demonstrating the instability problem.

In this example, the factor driving experience is discovered in the original sample and the factor sports car is discovered in the changed sample. Note that the two factors are mutually exclusive; that is, both factors are not discovered together. It is dangerous that both classifiers consider only
one factor, and not both, to determine the risk of a driver. We know from our common sense that both the driving experience and whether the insured car is a sports car or not are factors that should be both taken into account when determining the risk of a driver. If the prediction is based on only one factor, loss to the company is inevitable.

The instability of tree classifiers has implied the possibility of information loss. In the original sample, the factor *sports car* is undiscovered, and the information for the factor *driving experience* is undiscovered in the changed sample.

The instability problem of tree classifiers has complicated the process of making the right predictions and decisions. The danger of the instability problem is that it may be possible that some important factors remain undiscovered. Predictions/decisions based on partial factors may result in tremendous loss for the business. If a decision maker is unaware of the instability problem and makes predictions/decisions based on the factors discovered by the algorithm, it is possible that some of the important factors are not considered. Therefore, the decisions based on partial information may result in low profit. If a decision maker is fully aware of the danger of the instability problem, given the rules generated by one sample he/she may not be able to trust the rules enough to make any prediction or decision. If several sets of rules are produced by several different samples, the decision maker may be too overwhelmed or confused to make a good prediction/decision. Therefore, from the decision-makers’ point of view, the generated rules should be stable, accurate, and comprehensible.

The tree pruning process can make the instability problem even worse. It is obvious that pruning on different trees results in different rules. Consider the logically equivalent trees grown on slightly different samples in the tree growth phase. If the pruned subtrees are logically inequivalent, then different trees are generated. Figure 1.4 shows an example. Assume that $T_1$ and $T_2$ are two logically equivalent trees grown on two slightly different samples. Note that the splits $B$, $C$, and $D$ occur at different levels in the two trees. If the pruning process prunes the last node of both trees; that is, $D$ is pruned from $T_1$, and $B$ is pruned from $T_2$, then the final rules are different.
Figure 1.4: An example of different rules resulting from pruning two logically equivalent trees

Intuitively, the unstable behavior of tree classification algorithms has something to do with the split selection method at each node. The split selection method in most tree classifiers evaluates split candidates according to some split evaluation function and selects the best one to partition the data. If at some node $n$ there is no best split (i.e., a tie or almost equally good split candidates), one split candidate is selected as the best split. The resultant tree structure may be sensitive to small changes because a minor change in the training sample may result in a different split being selected at node $n$. Once a different split is selected, the subtree evolving from node $n$ may be very different from the original one. Thus different rules are generated.

Understanding the danger and the possible cause of the instability problem, is it possible to build a more stable classifier? How can we be sure that all the important factors are discovered by the tree classification algorithms? This motivates us to work on the instability problem of tree classifiers. Our goal is to help users gain confidence in the classification rules. We want to find out the reasons and causes of the instability problem and further provide useful information to assist decision making.
As we mentioned earlier, the unstable behavior of tree classifiers is caused by the instability of split selection. Therefore in this thesis we focus only on the tree growth phase.

The rest of this thesis is organized as follows: Chapter 2 reviews the representative classification algorithms. Chapter 3 presents and proves basic theorems for the instability problem, and formally defines the concept of “almost equally good splits”. Chapter 4 proposes algorithms to improve the stability of tree classifiers. The performance evaluation of the proposed algorithms is given in Chapter 5. We conclude with a summary of our contributions and suggestions for future work in Chapter 6.
Chapter 2

Related Work

Most research on tree classifiers done in the past were focused on the accuracy of the classification rules and the design of efficient classifiers to scale up to large training samples. The accuracy is measured by the number of the objects correctly classified, and efficiency is usually achieved by reducing the number of disk accesses. In addition, past research indicates that decision trees are inefficient to represent disjunctive rules and strongly correlated numeric attributes. Recently research has begun to deal with the instability problem. [Brei94, Brei96] present aggregation methods that generate a set of classifiers and then predict the class of an object based on a purity vote of the classifiers. This scheme has been used in recent commercial products [Salf00, Rule00].

In the following, the survey of research on tree classifiers includes the techniques that deal with the instability of the tree classifiers [Brei94, Brei96, Rule00, Salf00], well-known traditional classifiers [BFOS84, Quin86, Quin87, Rule00, Salf00], recently proposed scalable classifiers [MAR96, SAM96, GRG98, GGR99], and classifiers for disjunctions and strongly correlated attributes [FMM96, Oliv93].

2.1 Previous Work

2.1.1 Bagging Classification Trees

Bagging [Brei94, Brei96] is a technique that works well for unstable procedures like tree classification algorithms. Bagging (bootstrap aggregating) is applied to classification trees by generating
multiple versions of tree classifiers and using these to get an aggregated classifier. The aggregation does a plurality vote when predicting a class. The multiple classifiers are formed by making bootstrap replicas of the training sample. Tests on real and simulated data sets show that bagging can give substantial gains in accuracy if perturbing the training sample can cause significant changes in the classifier constructed.

2.1.2 CART and CART 4.0 Pro

The binary tree-structured classifiers in CART (Classification and Regression Trees) [BFOS84] are constructed in two phases: tree growth phase and pruning phase. In the tree growth phase, a split with the biggest decrease in impurity is selected to partition the records in a node. The process is repeated until all the records in a node are homogeneous. In the pruning phase, the tree is pruned upward to prevent overfitting. A sequence of trees \( T_0, T_1, \ldots, T_k \) is generated. \( T_0 \) is the original decision tree and each \( T_{i+1} \) is obtained by replacing one or more subtrees of \( T_i \) with leaves until the final tree \( T_k \) is just a leaf. Then \( T_0, \ldots, T_k \) are evaluated by a cost-complexity function and the best tree is selected as the final pruned tree.

CART 4.0 Pro [Sal00] is a commercial successor to CART by salfords systems. CART 4.0 Pro handles committees of trees (bagging) to gain more accurate prediction and provides users the details of individual trees used in the committee. In some cases these details can allow users to draw robust conclusions that are not otherwise apparent.

2.1.3 ID3, C4.5, and C5.0

ID3, and C4.5 [Quin86, Quin87] are classification algorithms introduced by Quinlan. ID3 is a limited classifier in the sense that it handles only discrete attributes and has no pruning phase. Unlike CART which grows a binary tree, ID3 generates a branch for every possible discrete attribute value. In the tree building phase, an entropy function is used to measure how informative a node is. The attribute with the most information gain is selected to partition the records.
C4.5 is an extension of ID3 that deals with a continuous attribute value range and also does pruning to get a more accurate tree. The pruning is done by examining the change in misclassifications over the test data set that would occur if a non-leaf subtree were replaced by the best possible leaf. This process continues until any further replacements would increase the number of errors over the test set.

C5.0 [Rule00] released by RuleQuest Research is a commercial successor to C4.5. C5.0 implements “boosting”, a technique that combines multiple decision trees into a single classifier. The idea is to generate several classifiers. When a new case is to be classified, each classifier votes for its predicted class and the votes are counted to determine the final class. To generate several classifiers, as the first step, a single decision tree is constructed from the training data. This classifier will usually make mistakes on some cases in the data. As a consequence, the second classifier is constructed with more attention to the cases that were predicted wrong by the first classifier. The process continues for a predetermined number of iterations.

2.1.4 IC

The Interval Classifier (IC) developed by Agrawal et al. [AGI+92] is essentially an ID3 algorithm. IC creates a branch for every value of a non-numeric attribute, and divides the range of a numeric attribute into \( k \) intervals. Instead of growing a full tree and then pruning it, IC dynamically prunes the tree as it is expanded. By limiting tests at nodes to point and range predicates, IC can utilize SQL query optimizers and database indices to retrieve data efficiently.

2.1.5 SLIQ

Most of the classification algorithms have the restriction that the training data should fit in memory. Mehta et al. [MAR96] have proposed a decision tree classifier, called SLIQ (Supervised Learning in Quest), that removes the memory constraint for large training data. SLIQ uses a one-time presorting technique to eliminate the task of resorting data at each node. Hence the cost of evaluating numeric attributes is reduced. This sorting procedure is integrated with a breadth-first tree growing
strategy to enable classification of disk-resident data sets. In the pruning phase, it uses a pruning strategy based on the Minimum Description Length (MDL) principle. The experiments show that SLIQ not only exhibits the same accuracy as other classifiers but also executes much faster and produces smaller trees.

### 2.1.6 SPRINT

SLIQ has removed the restriction that training data be memory-resident. However, it requires some information (*class list*) to stay memory-resident. The class list has to stay in memory because it is randomly accessed and frequently updated. Otherwise SLIQ will suffer severe performance degradation. The size of this memory-resident information grows linearly with the number of input records. Thus the size of the training set that SLIQ can handle is limited.

SPRINT (Scalable PaRallelizable INduction of decision Trees) presented by [SAM96] is a classifier that removes all of the memory restrictions. SPRINT bypasses the *class list* by creating an attribute list for each attribute which consists of attribute value, class label, and record identifier. During the tree growth phase, the splitting criterion is computed through a scan of all attribute lists. The distribution of an attribute list is performed through a hash-join of the attribute list with the attribute list of the splitting attribute.

While SPRINT removes all the memory constraint and scales to large training databases, it adds the hash-join overhead to distribute attribute lists.

### 2.1.7 RainForest

RainForest developed by [GRG98] is a unifying framework that can be applied to most decision tree algorithms, and results in a scalable version of the algorithm without modifying the result.

The observation by Gehrke et al. is that any classifier can be scaled up if there is an efficient way of finding splitting criteria (i.e. finding the AVC-group of a node). An AVC-group is a set
of all AVC-sets (AVC stands for Attribute Value and Class label), which provide the necessary information to decide on the best split at a node. The advantage of the AVC-group is that the size of an AVC-set at a node is relatively small. It depends on the number of distinct attribute values and the number of class labels at a node. The AVC-group of a node can be constructed by scanning the training database. Once the AVC-group is constructed, the best split can be determined. Then records in the node are distributed by scanning the database. Depending on the initial AVC-group size, several algorithms are proposed.

In contrast to Sprint where attribute lists are created and sorted only once, RainForest uses AVC-groups and repeatedly sorts the AVC-groups in memory. The results show that the RainForest family of algorithms outperforms SPRINT. Although SPRINT sorts attribute lists only once, it suffers the overhead of hash-join to partition records.

2.1.8 BOAT

BOAT (Bootstrapped Optimistic Algorithm Decision Tree) presented by [GGR+99] is a scalable algorithm for decision tree construction. BOAT uses a main-memory tree construction algorithm and a bootstrapping technique to grow a sample tree with coarse splitting criteria. Then the sample tree is used to guide the construction of the final tree.

The idea is that the sample tree grown on a sample should be very similar to the real tree grown on the training database. BOAT applies a bootstrapping technique to construct several trees and compares the splitting criterion of each node in these trees to derive the coarse splitting criteria. The correctness of the coarse splitting criteria is then checked by a lower bound technique. When the records in a node can fit in main memory, BOAT uses a traditional main-memory tree construction algorithm to grow the node.

The coarse splitting criteria have narrowed down the search space for possible splits at each node and made it possible to optimize the construction of the first few levels of the final tree. With the information of the coarse splitting criteria, it takes two scans to find the final splitting criteria.
for the nodes at the first few levels of the final tree compared to one scan of the training records per level by other algorithms. Hence the performance is improved.

Their results have shown that BOAT not only constructs the same tree as if a traditional main-memory algorithm were run on the complete training data set, but also runs faster. However, the instability of split selection will deteriorate the performance of BOAT, because two bootstrap samples will disagree on the splitting criterion.

2.1.9 Meta Learning

To solve the scaling problem with large training data sets, Chan and Stolfo [CS97] partition the data into subsets that fit in memory and then develop a classifier on each subset in parallel. The output of each classifier is then integrated to produce the final prediction. The process of integrating classifiers is called meta-learning.

The two integration strategies proposed are arbitration and combining. For each pair of classifiers, an arbitration classifier is built from a training set containing data disagreed upon by the two classifiers. The classification outcome is decided based on the predictions of the arbiter and the two classifiers. The process is repeated until the root arbiter is created.

Instead of using disagreed-upon data as the training set, a meta-level training set generated from the predictions and training examples is used to build a classifier called a combiner. The process of combining classifiers is repeated until the root combiner is created.

The approach of meta-learning reduced running time significantly but the integration of multiple classifiers did not achieve the accuracy of a single classifier using all the data.
2.1.10 Classifier in SONAR (System for Optimized Numeric Association Rules)

One problem with tree classifiers is that tree classifiers are inefficient if numeric attributes are strongly correlated. Fukuda et al. propose a scheme to solve this problem. Their scheme works as follows: Each numeric attribute is divided into $N$ ordered buckets such that each bucket contains roughly the same number of tuples. The next step is to find all pairs of strongly correlated numeric attributes. For each pair of strongly correlated attributes, create an $N \times N$ pixel grid generated as a Cartesian product of buckets. A split associated with a grid plan is a region in the plan that splits data into two classes: data inside the region and data outside the region. The optimal region that minimizes the entropy function (i.e., the best split associated with the strongly correlated attributes) is computed and added to the set of candidate tests in Quinlan’s entropy-based heuristic.

Their approach can produce small-sized decision trees. However, the decision rule is sometimes hard to describe.

2.1.11 Decision Graphs

Decision graphs are generalizations of decision trees. The feature that distinguishes decision graphs from decision trees is that decision graphs allow two nodes to have a common child. Hence decision graphs are more expressive than decision trees. For some decision functions like disjunctions which are inefficiently represented by decision trees due to replicated subtrees, decision graphs can describe them more simply.

An algorithm proposed by Oliver [Oliv93] applies a Minimum Message Length Principle to construct decision graphs. A message consists of a theory and the data being explained by the theory. To extend a graph, the algorithm determines whether to split a node or perform a Join. At each step, the algorithm finds the split and records the saving in message length for each leaf. For each pair of leaves, the algorithm performs a tentative Join and records the saving in message length. Then the alternation with the greatest saving in message length is selected. The process is repeated
until the graph is perfect or cannot be grown any further.

Oliver’s experiments showed that, compared to decision trees like those generated by ID3, decision graphs not only have comparable accuracy, but also describe the disjunctive concept efficiently.

2.2 Summary

The instability problem with tree-structured classifiers is that they may be sensitive to small changes in the training sample. That is, tree classification algorithms may produce dramatically different rules if the training sample is slightly changed. The instability problem complicates the process of the knowledge discovery because the users are disturbed by the different decision trees generated from almost the same input training samples. All the classifiers we surveyed may suffer the instability problem.

Aggregating methods, such as bagging and boosting [Brei96, Safl00, Rule00], are incorporated in the tree classification algorithms to mitigate the instability problem of the class predictions. Aggregating methods generate several versions of classifiers and then combine these classifiers to make predications. Although aggregating methods obtain higher accuracy, the downfall is that rules are difficult to comprehend because there is no one complete classifier and predictions are based on the verdicts of a committee which usually contains 25 or more [Brei94, Domi98] tree classifiers.

The problem that aggregating methods deal with is the instability of the class predictions, not the instability of the rules constructed. None of the classifiers we surveyed have dealt with the instability of rules, which is more important to data mining than just to give a stable prediction. The user would like a single stable rule, not just to provide a prediction, but to provide insight into the data.

Intuitively, the cause of the instability of tree classifiers is the instability of the split selection methods. At each node, the most promising split is selected to partition the data set associated
with the node. If the split candidates at a node are almost equally good, then the selection of one split to partition the data may result in rules that are sensitive to small changes.

Knowing that the instability problem is caused by split selection among almost equally good splits, the next questions that come to mind are "What are *almost equally good* splits?", "How to formalize and identify *almost equally good* splits?", and "How to lessen the instability problem?".

Answers to these questions are given in the following chapters. The next chapter analyzes the unstable behavior of decision tree classifiers, and proves theorems regarding the instability of tree classifiers. Based on the theorems, *almost equally good* splits can be defined.
Chapter 3

Instability Theorem

In this chapter, we prove the correctness of the conjecture that the instability of decision tree classifiers is caused by the split selection among a set of almost equally good split candidates at a node. A brief overview of split selection methods is given in 3.1. Then, we present the first theorem in 3.2.4, which shows the relationship between a change in the data and the resulting split change. Based on Theorem 1, the fuzzy term \textit{almost equally good splits} can be defined. Then, we present the second theorem in 3.2.6 – an instability theorem. The instability theorem confirms the correctness of the conjecture that the split selection among almost equally good split candidates causes the unstable behavior of decision tree classifiers. Since Theorem 1 assumes changes are made to the class labels only, the third theorem, in 3.2.7, shows the relationship between a data change and the resulting split change in the general case.

3.1 Split Selection Methods

The split selection method studied in this thesis is adopted from [BFOS84], because it has been widely used and produces trees with high predictive accuracy [BFOS84, Quin86, MRA95, MAR96, GRG98, GGR+99, Rule00, Salf00].

Classification trees are built by recursively partitioning the training sample. To find the best split to partition the data associated with a node, a set of split candidates are generated and evaluated. The goodness of a split candidate is defined as the decrease in node impurity between
the parent node and its two child nodes. The split candidate with the largest decrease in impurity (largest goodness measure) is selected as the split of the node. Once a good split is found, the data is partitioned by the split, and the process repeats for each of the two child nodes. The impurity of a node is defined as follows:

Assume that a node \( n \) has a set of objects \( D \) associated with it and let \( f_i \) be the proportion of the objects in \( D \) belonging to class \( i \), where \( i = 1, \ldots, k \). A measure \( I(n) \) of the impurity of a node \( n \) is defined as a nonnegative function \( \phi \) of class frequencies \( f_1, \ldots, f_k \), such that

\[
I(n) = \phi(f_1, \ldots, f_k),
\]

\[
\phi(1/k, \ldots, 1/k) = \text{maximum}, \text{ and}
\]

\[
\phi(1,0,\ldots) = \phi(0,1,\ldots) = \ldots = \phi(0,\ldots,1) = 0.
\]

That is, the node impurity is largest when all classes are equally mixed together in it, and smallest when the node contains only one class.

Suppose that a split candidate \( s \) divides the data \( D \) at a node \( n \) into two subsets such that a proportion \( p_L \) of the objects in \( D \) go to the left child node \( n_L \) and the proportion \( p_R \) go to the right child node \( n_R \). The goodness of the split is defined as the decrease in impurity

\[
good_{n,D}(s) = I(n) - p_L \cdot I(n_L) - p_R \cdot I(n_R).
\]

Many criteria can be used to measure the impurity of a node, such as entropy function and gini index of diversity [BFOS84]. We use the gini index to measure the impurity of a node because it is one of the most popular splitting indices [BFOS84, MRA95, MAR96, GRG98, Salf00], is simple, can be quickly computed, and is often the best splitting rule [BFOS84, Salf00]. The impurity of a node \( n \) measured by the gini index is defined as

\[
I(n) = \phi(f_1, \ldots, f_k) = \sum_{i \neq j} f_i \cdot f_j.
\]

In the case of decision tree classifiers where the objects in a training sample are categorized into two classes, the node impurity measured by the gini index is \( I(n) = 2 \cdot f_1 \cdot f_2 \).
3.2 The Instability Problem with Decision Tree Classifiers

3.2.1 The Instability Problem

Assume that $S$ and $S'$ are two slightly different training samples (i.e., $S'$ can be thought of as the sample obtained by making a small change to $S$) and the trees grown on the two samples differ in the subtrees $T_2$ and $T'_2$ as shown in Figure 3.1. The difference between the two training samples (or the change to $S$) results in a split change from $t_0$ at the root node of $T_2$ to $t$ at the root node of $T'_2$.

In classification trees, each node in a tree represents the subset of the training sample that satisfies the condition formed by the splits of the nodes from the root to the parent node of the node. Since the two subtrees $T_1$ and $T'_1$ are the same (that is, the same splits at the same positions), the data associated with the root nodes of $T_2$ and $T'_2$ satisfy the same condition. Because the training sample $S'$ is slightly different from the training sample $S$, the data $D'$ associated with the root node of $T'_2$ will in general be slightly different from the data $D$ associated with the root node of $T_2$.

At the root node of $T_2$, $t_0$ is the best split to partition the data $D$. After a small change is made to $S$, $t$ is the best split to partition $D'$ which is the data associated with the root node of $T'_2$ and almost the same as the data $D$. Why was a different split $t$ selected to partition $D'$ that is almost the same as $D$? It is easy to conjecture that $t$ should be also a good split candidate at the root node of $T_2$ (in $D$). Otherwise, a small change won't make it become the best split in $D'$. This implies that the goodness measures of the two split candidates $t_0$ and $t$ at the root node of $T_2$ (in $D$) should be relatively close. But how to measure the closeness? In the following, we will show the relationship between the closeness of the two splits $t$ and $t_0$, and the change $c$, where $c$ is the difference between $D$ and $D'$. We will prove in the first theorem that if a change $c$ in $D$ results in a split change from $t_0$ to $t$, then the difference between the goodness measures of $t_0$ and $t$ is less than or equal to the value of a function $\varepsilon(c)$, and conversely, if the difference between the goodness measures of $t_0$ and $t$ is less than or equal to $\varepsilon(c)$, then $t$ will be superior to $t_0$ in the changed data.
3.2.2 Assumptions and Parameters

We assume that the objects in a training sample belong to either the class ‘+’ or the class ‘−’, each decision tree node has only two child nodes, and changes are made to the objects’ class labels only.

Assume that a change in a training sample $S$ results in a split change from $t_0$ at the root of the subtree $T_2$ to $t$ at the root of the subtree $T'_2$ as shown in Figure 3.1. Denote the data associated with the root nodes of $T_2$ and $T'_2$ by $D$ and $D'$, respectively. The difference between $D$ and $D'$ is represented by the change parameter $c$, i.e., $D + c = D'$. Define the size of $c$, $|c|$, as the fraction of the objects in $D$ that are changed.

To study the relationship between the change $c$ and the split change from $t_0$ to $t$, we analyze the goodness measures of the two splits before and after the change. The measurement of the goodness
of a split at a node requires the knowledge of the class frequencies in the data associated with the node and the two child nodes. Therefore, four sets of parameters are necessary to evaluate \( t_0 \) and \( t \) at the root nodes of \( T_2 \) and \( T'_2 \).

To measure the goodness of the split \( t_0 \) of the root node of \( T_2 \), assume that \( t_0 \) partitions the data \( D \) into two subsets \( D_{t_0,L} \) and \( D_{t_0,R} \) such that a proportion \( p \) of the objects in \( D \) go to \( D_{t_0,L} \) and the proportion \((1 - p)\) go to \( D_{t_0,R} \). Let \( f, g, \) and \( h \) be the fractions of the objects in \( D, D_{t_0,L}, \) and \( D_{t_0,R} \), respectively, that belong to the class '+'.

To measure the goodness of the split candidate \( t \) at the root node of \( T_2 \), assume that \( t \) partitions the data \( D \) into two subsets \( D_{t,L} \) and \( D_{t,R} \) such that a proportion \( q \) of the objects in \( D \) go to \( D_{t,L} \) and the proportion \((1 - q)\) go to \( D_{t,R} \). Let \( u \) and \( v \) be the fractions of the objects in \( D_{t,L} \) and \( D_{t,R} \), respectively, that belong to the class '+'. Figure 3.2(a) shows all these parameters pictorially.

Another two sets of parameters are needed to measure the goodness of \( t_0 \) and \( t \) at the root node of \( T'_2 \). Since the two sets of parameters are similar to those at the root node of \( T_2 \), denote the parameters by attaching an apostrophe (prime) to the parameters used to evaluate the two splits at the root of \( T_2 \). Figure 3.2(b) shows these parameters pictorially. Table 3.1 summaries all the parameters.

### 3.2.3 Three Lemmas

The goodness of a split candidate at a node is determined by the proportion of the objects that go to each child node, and the class frequencies in the data associated with the node and the child nodes. Therefore, to measure the goodness of \( t_0 \) and \( t \) at the root node of \( T'_2 \), we have to know the parameters \( f', p', g', h', q', u', \) and \( v' \). Note that there is no need to evaluate split candidates that result in empty child nodes (i.e., \( p, q = 0, \) or \( 1 \)), because there is no impurity improvement for the same data sets. Therefore, \( p \) and \( q \) are assumed not equal to 0 or 1.
(a) Parameters to measure the goodness of $t_0$ and $t$ at the root of $T_2$

(b) Parameters to measure the goodness of $t_0$ and $t$ at the root of $T_2'$

Figure 3.2: Parameters to evaluate $t_0$ and $t$ at the root nodes of $T_2$ and $T_2'$

In the following, we present three lemmas for the proof of Theorem 1. The first lemma proves that the proportions of the objects that go to the left child node are the same before and after the change, i.e., $p' = p$ and $q' = q$. Then, we study the effect of the change on the class frequencies and show the new class frequencies in the second lemma. The third lemma shows the relationship between the change parameters $c_0$, $c_1$, $c_2$, and $c_3$.

**Lemma 1** With the assumption that changes are made to class labels only, $D$ and $D'$, which are the data associated with the root nodes of $T_2$ and $T_2'$ respectively, contain equal number of objects. If $D$ and $D'$ are divided by the same split, then the left child nodes contain equal number of objects. Therefore, $p' = p$ and $q' = q$. 

23
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>a sample</td>
</tr>
<tr>
<td>$D$</td>
<td>a subset of $S$</td>
</tr>
<tr>
<td>$t_0, t$</td>
<td>split candidates</td>
</tr>
<tr>
<td>$D_{x, L}, D_{x, R}$</td>
<td>the two subsets of $D$ divided by the split $x$</td>
</tr>
<tr>
<td>$p (q)$</td>
<td>the proportion of the objects in $D$ that go to $D_{x, L}$ ($D_{x, R}$)</td>
</tr>
<tr>
<td>$\phi$</td>
<td>the proportion of the objects in $D$ that belong to the class '+$'</td>
</tr>
<tr>
<td>$g (u)$</td>
<td>the proportion of the objects in $D_{x, L}$ ($D_{x, R}$) that belong to the class '+$'</td>
</tr>
<tr>
<td>$h (v)$</td>
<td>the proportion of the objects in $D_{x, R}$ ($D_{x, L}$) that belong to the class '+$'</td>
</tr>
<tr>
<td>$S'$</td>
<td>the sample obtained by applying a change to $S$</td>
</tr>
<tr>
<td>$D'$</td>
<td>a subset of $S'$ ($D + c = D'$)</td>
</tr>
<tr>
<td>$e$</td>
<td>the difference between $D$ and $D'$</td>
</tr>
<tr>
<td>$D'<em>{x, L}, D'</em>{x, R}$</td>
<td>the two subsets of $D'$ divided by the split $x$</td>
</tr>
<tr>
<td>$p' (q')$</td>
<td>the proportion of the objects in $D'$ that go to $D'<em>{x, L}$ ($D'</em>{x, R}$)</td>
</tr>
<tr>
<td>$\phi'$</td>
<td>the proportion of the objects in $D'$ that belong to the class '+$'</td>
</tr>
<tr>
<td>$g' (u')$</td>
<td>the proportion of the objects in $D'<em>{x, L}$ ($D'</em>{x, R}$) that belong to the class '+$'</td>
</tr>
<tr>
<td>$h' (v')$</td>
<td>the proportion of the objects in $D'<em>{x, R}$ ($D'</em>{x, L}$) that belong to the class '+$'</td>
</tr>
</tbody>
</table>

Table 3.1: Descriptions of the parameters

Proof

In classification trees, splits are based on non-class-label attributes, and each node represents the subset of the training sample that meets the condition formed by the splits of the nodes from the root to the parent node of the node. Since $T_1$ and $T'_1$ are the same (i.e., the same splits at the same positions), $D$ and $D'$ are the subsets of $S$ and $S'$ respectively that satisfy the same condition. Because $S$ and $S'$ differ only in the class labels of some objects, $D$ and $D'$ contain the same number of the objects. In fact, $D$ and $D'$ are the same except for the class labels of some objects.

It is obvious that if $D$ and $D'$ are divided by the same split, then the left child nodes contain the same number of objects. The proportion of the objects that go to the left child nodes for $t_0$ and $t$ are $p' = \frac{|D'_{x, L}|}{|D'|} = \frac{|D_{t_0, L}|}{|D|} = p$ and $q' = \frac{|D'_{x, R}|}{|D'|} = \frac{|D_{t, R}|}{|D|} = q$, respectively.

The second lemma gives the proportions of the objects in $D'$ and the child subsets of $D'$ divided

24
by \( t_0, D'_{t_0,L} \) and \( D'_{t_0,R} \), that belong to the class ' \(+\) '. From Lemma 1, the data \( D_{t_0,L} \) and \( D'_{t_0,L} \) are the same except for the class labels of some objects, and so are the data \( D_{t_0,R} \) and \( D'_{t_0,R} \). Thus, a change \( c \) to \( D \) can be described as the change to the objects' class labels in the child subsets as follows:

\[
x_0 : \text{ the fraction of the objects that belong to } D_{t_0,L} \text{ and change class from } '+' \text{ to } '-' .
\]

\[
x_1 : \text{ the fraction of the objects that belong to } D_{t_0,L} \text{ and change class from } '-' \text{ to } '+' .
\]

\[
x_2 : \text{ the fraction of the objects that belong to } D_{t_0,R} \text{ and change class from } '+' \text{ to } '-' .
\]

\[
x_3 : \text{ the fraction of the objects that belong to } D_{t_0,R} \text{ and change class from } '-' \text{ to } '+' .
\]

The \( x_i \) are all fractions of the number of objects in \( D, |D| \). Figure 3.3 shows the impact of the change \( c \) on the class distributions. The ellipse represents the data \( D \) and the split \( t_0 \) divides it into two subsets \( D_{t_0,L} \) and \( D_{t_0,R} \). Within each subset, the objects are grouped by the classes to which they belong. The original class distribution is shown in Figure 3.3(a). Figure 3.3(b) shows the change \( c \) (i.e., \(|c| = x_0 + x_1 + x_2 + x_3\)) made to \( D \) as described above. Figure 3.3(c) shows the new class distribution in the subsets \( D'_{t_0,L} \) and \( D'_{t_0,R} \). The new class frequencies in \( D' \), \( D'_{t_0,L} \) and \( D'_{t_0,R} \) are given in the following lemma.

**Lemma 2** The class frequencies in \( D' \), \( D'_{t_0,L} \), and \( D'_{t_0,R} \) are

(a) \( f' = f - c_0 - c_1 \),

(b) \( g' = g - \frac{c_0}{p} \), and

(c) \( h' = h - \frac{c_1}{1-p} \), where \( c_0 = x_0 - x_1 \), and \( c_1 = x_2 - x_3 \).

**Proof** (a) The proportion \( x_0 + x_2 \) of the objects in \( D \) change class from ' \(+\) ' to ' \(-\) ' and the proportion \( x_1 + x_3 \) of the objects in \( D \) change class from ' \(-\) ' to ' \(+\) '. Therefore, the proportion of the objects in \( D' \) that belong to the class ' \(+\) ' is
Figure 3.3: The class distributions before and after change

\[ f' = f - (x_0 + x_2) + (x_1 + x_3) = f - c_0 - c_1, \text{ where } c_0 = x_0 - x_1, \text{ and } c_1 = x_2 - x_3. \]

(b) The proportion \( p \) of the objects in \( D \) go to \( D_{t_0,L} \), and the proportion \( g \) of them belong to the class '＋'. Therefore, the number of the objects in \( D_{t_0,L} \) that belong to the class '＋' is \( g \cdot p \cdot |D| \). The change has made \( x_0 \cdot |D| \) objects in \( D_{t_0,L} \) change class from '＋' to '－', and \( x_1 \cdot |D| \) objects change classes from '－' to '＋'. Therefore, the number of the objects that belong to the class '＋' in \( D'_{t_0,L} \) is \( g \cdot p \cdot |D| - (x_0 \cdot |D| - x_1 \cdot |D|) \).
From lemma 1, \(|D'| = |D|\) and \(p' = p\). Hence, the frequency of the class '+\(' in \(D'_{t_0,L}\) is
\[
g' = \frac{\# \text{ of the objects in } D'_{t_0,L} \text{ that belong to class } '+'}{\# \text{ of objects in } D'_{t_0,L}}
= \frac{g \cdot p \cdot |D| - (x_0 - x_1) \cdot |D|}{p |D|}
= g - \frac{(x_0 - x_1)}{p}
= g - \frac{c_0}{p}.
\]

(c) Proof of (c) is similar to the proof of (b).

\[
\square
\]

To analyze how the goodness of a split \(t_0\) is effected by \(c\), the change \(c\) is described in terms of the class frequency changes in the child subsets. The impact of the change \(c\) on the class distributions in \(D_{t_0,L}\) and \(D_{t_0,R}\) are described by the parameters \(c_0\) and \(c_1\). The signs of \(c_0\) and \(c_1\) indicate the class change from '+' to `'-' or from `'-' to '+''. If \(c_0\) is positive, then at least \(c_0 \cdot |D|\) objects in \(D_{t_0,L}\) change class from '+' to `'-' . If \(c_0\) is negative, then at least \((-c_0) \cdot |D|\) objects in \(D_{t_0,L}\) change class from `'-' to '+' . Similar to \(c_0\) in \(D_{t_0,L}\), the parameter \(c_1\) represents the smallest fraction of the objects in \(D_{t_0,R}\) that change classes. Therefore, \(|c_0| + |c_1| \leq |c|\).

Let the parameters \(c_2\) and \(c_3\) denote the impact of \(c\) on the class distributions in \(D_{t,L}\) and \(D_{t,R}\), i.e., \(u' = u - \frac{c_2}{q}\), and \(v' = v - \frac{c_3}{1-q}\).

Figure 3.4 shows an example of the impact of a change on the class frequencies in \(D_{t_0,L}\), \(D_{t_0,R}\), \(D_{t,L}\), and \(D_{t,R}\). Consider a data set consisting of eight objects, \(o_1, \ldots, o_8\). Figure 3.4 (a) and (b) show the class distributions in the child subsets. If the first and the seventh objects change classes, then the effect of the change to the class distributions in \(D_{t_0,L}\) and \(D_{t_0,R}\) are described by \(c_0 = \frac{1}{8}\) and \(c_1 = -\frac{1}{8}\), respectively. That is, one object \((o_1)\) in \(D_{t_0,L}\) changes class from '+' to `'-' , and one object \((o_7)\) in \(D_{t_0,R}\) changes class from `'-' to '+' . Because the objects that change classes are both in \(D_{t,L}\), the change has no effect on the class distribution in \(D_{t,R}\), i.e., \(c_3 = 0\). Since the number
Figure 3.4: The effect of the change $c$ to the class distributions in $D_{t_0,L}$, $D_{t_0,R}$, $D_{t,L}$, and $D_{t,R}$ of the objects changing classes from $'+'$ to $'-'$ is the same as the number of the objects changing classes from $'-'$ to $'+'$ in $D_{t,L}$, $c_2 = 0$. The class distributions in the changed data are shown in Figure 3.4 (c) and (d).

The relationship between the parameters $c_0$, $c_1$, $c_2$, and $c_3$ are shown in the next lemma.

Lemma 3 $c_0 + c_1 = c_2 + c_3$.  

Proof $c_0 + c_1$

$= c_2 + c_3$

$= \text{the number of the objects in } D \text{ changing class from } '+' \text{ to } '-'$
– the number of the objects in $D$ changing class from $'-'$ to $'+'$.

\[ \square \]

In summary, we have studied the effect of the change $c$ to the class frequencies in the subsets of $D$ divided by the splits $t_0$ and $t$. The class frequency changes in $D_{t_0,L}$ and $D_{t_0,R}$ can be described by the parameters $c_0$ and $c_1$, and the class frequency changes in $D_{t,L}$ and $D_{t,R}$ can be described by the parameters $c_2$ and $c_3$. Since the parameters $c_0$ and $c_1$ represent the same change as the change represented by the parameters $c_2$ and $c_3$, the relationship between $c_0$, $c_1$, $c_2$, and $c_3$ are given in Lemma 3.

Having established all the necessary properties, we are now ready to prove the first theorem, which provides a measurement of the closeness of the splits $t_0$ and $t$ at the root of $T_2$.

### 3.2.4 Theorem 1 – A function $\varepsilon$ that measures the closeness of $t_0$ and $t$

**Theorem 1** Assume that $D'$ is obtained by changing the classes of a fraction $|c|$ of the objects in $D$. The effect of the change $c$ to the class frequencies in $D_{t_0,L}$ and $D_{t_0,R}$ are described by $c_0$ and $c_1$, and the effect of the change $c$ to the class frequencies in $D_{t,L}$ and $D_{t,R}$ are described by $c_2$ and $c_3$.

Then the split $t_0$, which is the split selected to partition the data $D$, will be inferior to a split $t$ in $D'$ if and only if

\[ 0 \leq \text{good}_{n,D}(t_0) - \text{good}_{n,D}(t) \leq \varepsilon(c), \quad (3.1) \]
\[ \varepsilon(c) = \varepsilon(c_0, c_1, c_2) = 2 \left[ 2gc_0 + 2hc_1 - 2uc_2 - 2v(c_0 + c_1 - c_2) - \frac{c_0^2}{p} - \frac{c_1^2}{1-p} \right. \\
+ \left. \frac{c_2^2}{q} + \frac{(c_0 + c_1 - c_2)^2}{1-q} \right]. \]

**Proof (⇒)** From the definition, the goodness measures of the split candidates \( t_0 \) and \( t \) in \( D \) and \( D' \) using the gini index are

\[ \text{good}_{n,D}(t_0) = 2f(1 - f) - 2pg(1 - g) - 2(1 - p)h(1 - h), \]

\[ \text{good}_{n,D}(t) = 2f(1 - f) - 2qu(1 - u) - 2(1 - q)v(1 - v), \]

\[ \text{good}_{n',D'}(t_0) = 2f'(1 - f') - 2p'g'(1 - g') - 2(1 - p')h'(1 - h'), \]

\[ \text{good}_{n',D'}(t) = 2f'(1 - f') - 2q'u'(1 - u') - 2(1 - q')v'(1 - v'). \]

The difference of the goodness measures of \( t_0 \) and \( t \) in \( D' \) can be expressed as

\[ \text{good}_{n',D'}(t_0) - \text{good}_{n',D'}(t) \]

\[ = \text{good}_{n,D}(t_0) - \text{good}_{n,D}(t) - 2(K_0 - K_1), \]

where \( K_0 = pg^2 + (1 - p)h^2 - [p'g'^2 + (1 - p')h'^2] \), and

\[ K_1 = qu^2 + (1 - q)v^2 - [q'u'^2 + (1 - q')v'^2]. \]

Since \( t \) is a better split than \( t_0 \) in \( D' \), \( \text{good}_{n',D'}(t_0) \leq \text{good}_{n',D'}(t) \).

Therefore,

\[ \text{good}_{n,D}(t_0) - \text{good}_{n,D}(t) \leq 2(K_0 - K_1). \]

From Lemma 2, the new class frequencies for the class \('+' in \( D'_{t_0,L}, D'_{t_0,R}, D'_t, L, D'_{t,R} \) are

\[ g' = g - \frac{c_0}{p}, \]

\[ h' = h - \frac{c_1}{1-p}, \]

\[ u' = u - \frac{c_2}{q}, \]

and

\[ v' = v - \frac{c_3}{1-q}, \]

30
where \( c_0, c_1, c_2, \) and \( c_3 \) are defined the same as in Lemma 2.

From Lemma 3, \( c_0 + c_1 = c_2 + c_3 \). Therefore,

\[
v' = v - \frac{c_0 + c_1 - c_2}{1 - q},
\]

Thus,

\[
\varepsilon(c)
\]

\[
= \varepsilon(c_0, c_1, c_2)
\]

\[
= 2(K_0 - K_1)
\]

\[
= 2[pg^2 + (1 - p)h^2 - p'g'^2 - (1 - p')h'^2 - qu^2 - (1 - q)v^2 + q'u'^2 + (1 - q')v'^2]
\]

\[
= 2[pg^2 + (1 - p)h^2 - p(g - \frac{c_0}{p})^2 - (1 - p)(h - \frac{c_1}{1 - p})^2
\]

\[-qu^2 - (1 - q)v^2 + q(u - \frac{c_2}{q})^2 + (1 - q)(v - \frac{c_0 + c_1 - c_2}{1 - q})^2]
\]

\[
= 2[2gc_0 + 2hc_1 - 2uc_2 - 2v(c_0 + c_1 - c_2) - \frac{c_0^2}{p} - \frac{c_1^2}{1 - p} + \frac{c_2^2}{q} + \frac{(c_0 + c_1 - c_2)^2}{1 - q}].
\]

(\(\Leftarrow\)) Conversely, suppose that \( good_{n,D}(t_0) - good_{n,D}(t) \leq \varepsilon(c) \).

Therefore,

\[
good_{n,D}(t_0) - good_{n,D}(t) - \varepsilon(c) = good_{n',D'}(t_0) - good_{n',D'}(t) \leq 0.
\]

Thus, \( good_{n',D'}(t_0) \leq good_{n',D'}(t) \). This proves that the split \( t_0 \) is inferior to the split \( t \) in the changed data \( D' \).

\[\square\]

From Theorem 1, if a change \( c \) has made the fraction \(|c|\) of the objects in \( D \) change classes and the change \( c \) results in a different split \( t \) selected to partition the changed data \( D' \), then the difference of the goodness measures between \( t_0 \), which is the split to partition \( D \), and \( t \) is less than or equal to \( \varepsilon(c) \) in the original data \( D \). Conversely, if there exists a change \( c \) and a split \( t \) such that the inequality 3.1 holds, then \( t \) will be superior to \( t_0 \) in the changed data. Notice that the function \( \varepsilon \) is different for every pair of splits \( t_0 \) and \( t \), where \( t \) is a split candidate in \( D \). Therefore, the change \( c \) required to make the inequality 3.1 hold is different for every pair of splits. The size of the change \(|c|\) is proportional to the size of the data \( D \). Therefore, the change \( c \) in \( D \), which is
a subset of the training sample $S$, is equivalent to the proportion $\frac{c|x|_D}{|S|}$ of change in the training sample.

### 3.2.5 Almost Equally Good Splits

In the following, we define the fuzzy term *almost equally good splits*. Let $t_0, t_1, \ldots, t_k$ be the split candidates in $D$, and let $t_0$ be the best split, i.e., $\text{good}_{n,D}(t_0) \geq \text{good}_{n,D}(t_i), \ i = 1, \ldots, k$. Let $c^{i*}$ be the smallest change that satisfies the inequality $\text{good}_{n,D}(t_0) - \text{good}_{n,D}(t_i) \leq \varepsilon(c^{i*})$ for every pair of the splits $t_0$ and $t_i, i = 1, \ldots, k$. We say a change $x$ is smaller than another change $y$ if the number of the objects changed by $x$ is less than or equal to the number of the objects changed by $y$, i.e., $|x| \leq |y|$. Define the “sensitivity of $t_0$” and splits that are “almost as good as $t_0$” as follows:

**Definitions** Define the *sensitivity of $t_0$ to $t_i$* as the size of the smallest change that satisfies the inequality 3.1, $|c^{i*}|$. That is, if the fraction $|c^{i*}|$ of the objects are changed, then $t_0$ may become inferior to $t_i$ in the changed data.

The *sensitivity of $t_0$* is the fraction $c^*$ such that $c^* = \min\{|c^{1*}|, \ldots, |c^{k*}|\}$. That is, the sensitivity of $t_0$ is the smallest fraction change in $D$ that may make $t_0$ no longer the best split in the changed data.

A split $t_i$ is defined to be *almost as good as $t_0$* with respect to a predefined fraction $c$ if $|c^{i*}| \leq c$. Denote the *set of splits that are almost as good as $t_0$* with respect to a predefined fraction $c$ by $\text{AlmostEquallyGoodSplitSet}(t_0, c) = \{t_i \mid |c^{i*}| \leq c\}$.

As an example, consider four split candidates $t_0, t_1, t_2,$ and $t_3$. The smallest changes that satisfy the inequality 3.1 for every pair of the splits $t_0$ and $t_i$ are given below.

- $c^{1*} =$ sensitivity of $t_0$ to $t_1 = 1\%$,
- $c^{2*} =$ sensitivity of $t_0$ to $t_2 = 2\%$, 

32
\( c^* = \text{sensitivity of } t_0 \text{ to } t_3 = 5\%. \)

The sensitivity of \( t_0 \) is 1% which is the smallest change that may make \( t_0 \) no longer the best split in the changed data. The splits that are almost as good as \( t_0 \) with respect to a fraction 3% change are \( t_0, t_1, \text{ and } t_2 \); that is, \( \text{AlmostEquallyGoodSplits}(t_0, 3\%) = \{t_0, t_1, t_2\} \). If 3% of the data are changed, then any split in \( \text{AlmostEquallyGoodSplits} \) may have chance to become the best split in the changed data.

Having defined almost equally good splits, next we present an instability theorem, which confirms that almost equally good splits are the cause of the instability of classification algorithms.

### 3.2.6 Theorem 2 – Instability Theorem

**Theorem 2** (Instability Theorem) Assume that \( t_0 \) is the split of a node and \( t \) is a split that is almost as good as \( t_0 \) with respect to a predefined fraction \( c \). Then \( t \) may be the split selected to partition the changed data, if a change is made to a fraction \( \frac{c|D|}{|S|} \) of the objects in the training sample \( S \), where \( D \) is the data associated with the node.

**Proof** Since \( t \) is a split that is almost as good as \( t_0 \) with respect to a fraction \( c \), there exists a change \( c' \) that satisfies the inequality 3.1 and \( |c'| \leq c \). From Theorem 1, if change \( c' \) is applied to \( D \), \( t \) will be superior to \( t_0 \). Therefore, if the fraction \( c \) of \( D \) is changed, \( t \) may be superior to \( t_0 \) and may be the best split to partition the changed data.

A fraction \( c \) change in \( D \) is equivalent to the fraction \( \frac{c|D|}{|S|} \) change in \( S \). Therefore, \( t \) may be the split selected to partition the changed data, if the fraction \( \frac{c|D|}{|S|} \) of \( S \) is changed.

\[ \square \]
3.2.7 Theorem 3 – A general function $\varepsilon$ to measure the closeness of $t_0$ to $t$

So far we have assumed that changes are made to objects' class labels only. In this section, we relax this restriction by allowing changes to be deletion or addition of objects, or changes in the values of attributes.

The analysis for the general model is similar to the analysis for the simple model that changes are made to the class labels only. The evaluation of the goodness of a split $t_0$ in the changed data $D'$ requires the knowledge of the class frequencies in $D'$ and its child subsets $D'_{t_0,L}$ and $D'_{t_0,R}$. In the following, we study the effect of a change to the class frequencies in $D$, $D_{t_0,L}$, and $D_{t_0,R}$. The effect of a change in the training sample can be deletion or addition of objects in $D$, or the change can make objects in $D$ change classes or move from one child subset to the other one. The effect of a change to the class frequencies in the subsets $D_{t_0,L}$ and $D_{t_0,R}$ are described below. All the parameters are proportional to the size of the data $D$.

![Diagram](image_url)

Figure 3.5: Possible changes for the general model
\( c_0 \): the fraction of the objects that leave or join the class \('+'\) in \( D_{t_0,L} \). If \( c_0 \) is positive, then the change causes \( c_0 \cdot |D| \) of the objects, which belong to the class \('+'\) in \( D_{t_0,L} \), to leave \( D \). If \( c_0 \) is negative, the change results in \((-c_0) \cdot |D|\) more objects from other subsets to join the class \('+'\) in \( D_{t_0,L} \).

\( c_1 \): the fraction of the objects that leave or join the class \('-'\) in \( D_{t_0,L} \).

\( c_2 \): the fraction of the objects that leave or join the class \('+'\) in \( D_{t_0,R} \).

\( c_3 \): the fraction of the objects that leave or join the class \('-'\) in \( D_{t_0,R} \).

\( c_4 \): the fraction of the objects that move between the class \('+'\) and the class \('-'\) in \( D_{t_0,L} \). If \( c_4 \) is positive, then \( c_4 \cdot |D| \) objects in \( D_{t_0,L} \) change classes from \('+'\) to \('-'\). If \( c_4 \) is negative, then \((-c_4) \cdot |D|\) objects in \( D_{t_0,L} \) change classes from \('-'\) to \('+'\).

\( c_5 \): the fraction of the objects that move between the class \('+'\) in \( D_{t_0,L} \) and the class \('+'\) in \( D_{t_0,R} \).

\( c_6 \): the fraction of the objects that move between the class \('+'\) in \( D_{t_0,L} \) and the class \('-'\) in \( D_{t_0,R} \).

\( c_7 \): the fraction of the objects that move between the class \('-'\) in \( D_{t_0,L} \) and the class \('+'\) in \( D_{t_0,R} \).

\( c_8 \): the fraction of the objects that move between the class \('-'\) in \( D_{t_0,L} \) and the class \('-'\) in \( D_{t_0,R} \).

\( c_9 \): the fraction of the objects that move between the class \('+'\) and the class \('-'\) in \( D_{t_0,R} \).

**Lemma 4** The size of \( D' \), the proportion \( p' \) of the objects in \( D' \) that go to the subset \( D'_{t_0,L} \), and the class frequencies in \( D'_{t_0,L} \) and \( D'_{t_0,R} \) are

(a) \( |D'| = (1 - c_0 - c_1 - c_2 - c_3) \cdot |D|, \)

(b) \( |D'_{t_0,L}| = (p - c_0 - c_1 - c_5 - c_6 - c_7 - c_8) \cdot |D|, \)

\( |D'_{t_0,R}| = (1 - p - c_2 - c_3 + c_5 + c_6 + c_7 + c_8) \cdot |D|, \)

(c) \( p' = \frac{p - c_0 - c_1 - c_5 - c_7 - c_8}{1 - c_0 - c_1 - c_2 - c_3}, \)

(d) \( g' = \frac{g - p - c_0 - c_4 - c_5 - c_6}{p - c_0 - c_1 - c_5 - c_6 - c_7 - c_8} \).
(e) $h' = \frac{h(1-p)-c_2+c_5+c_7-c_9}{1-p-c_2-c_3+c_5+c_6+c_7+c_9}.$

Proof (a) The parameters $c_0$, $c_1$, $c_2$, and $c_3$ represent the fractions of the objects that leave $D$ if they are positive, and join $D$ if they are negative. Therefore, the size of the changed data $D'$ is

$$|D'| = |D| - (c_0 + c_1 + c_2 + c_3) \cdot |D| = (1 - c_0 - c_1 - c_2 - c_3) \cdot |D|.$$  

(b) The parameters $c_0$ and $c_1$ represent the fractions of the objects that leave or join the subset $D_{t_0,L}$ after the change, and the fractions $c_5$, $c_6$, $c_7$, and $c_8$ represent the fractions of the objects that move between $D_{t_0,L}$ and $D_{t_0,R}$. Therefore, the number of objects in $D_{t_0,L}'$ is

$$|D_{t_0,L}'| = |D_{t_0,L}| - (c_0 + c_1 + c_5 + c_6 + c_7 + c_8) |D|$$

$$= p \cdot |D| - (c_0 + c_1 + c_5 + c_6 + c_7 + c_8) |D|$$

$$= (p - c_0 - c_1 - c_3 - c_6 - c_7 - c_8) \cdot |D|.$$  

The number of objects in $D_{t_0,R}'$ is

$$|D_{t_0,R}'| = |D'| - |D_{t_0,L}'| = (1 - p - c_2 - c_3 + c_5 + c_6 + c_7 + c_8) \cdot |D|.$$  

(c) $p' = \frac{|D_{t_0,L}'|}{|D'|} = \frac{p - c_0 - c_1 - c_3 - c_6 - c_7 - c_8}{1 - c_0 - c_1 - c_2 - c_3}.$

(d) The number of the objects that belong to the class '+' in $D_{t_0,L}$ is $g \cdot p \cdot |D|$. The parameters $c_0$, $c_4$, $c_5$, and $c_6$ represent the fractions of the objects that leave or join the class '+' in $D_{t_0,L}$. Hence,

$$g' = \frac{g \cdot p \cdot |D| - (c_0 + c_4 + c_5 + c_6) |D|}{|D_{t_0,L}'|} = \frac{g \cdot p - c_0 - c_4 - c_5 - c_6}{p - c_0 - c_1 - c_3 - c_6 - c_7 - c_8}.$$  

(e) The proof of (e) is similar to the proof of (d).
Theorem 3 is similar to Theorem 1 except that changes are not restricted to the class labels; therefore, a different function $\varepsilon$ is derived. The change can be addition of new objects, deletion of existing objects, or changing the attribute values of the objects in the training sample.

**Theorem 3** Let $S'$ be the training sample obtained by making a change $c$ in a training sample $S$. Then the split $t_0$, which is the split selected to partition the data $D$, will be inferior to a split $t$ in $D'$, if and only if

$$good_{n,D}(t_0) - good_{n,D}(t) \leq \varepsilon(c), \quad (3.2)$$

where

$$\varepsilon(c) = \varepsilon(c_0, \ldots, c_9, d_0, \ldots, d_8)$$

$$= 2[p^2 + (1 - p)h^2 - qu^2 - \frac{(pg + (1 - p)h - qu)^2}{1 - q} - \frac{(gp - c_0 - c_4 - c_5 - c_6)^2}{(1 - c_0 - c_1 - c_2 - c_3)(p - c_0 - c_1 - c_2 - c_6 - c_7 - c_8)}$$

$$- \frac{(h(1 - p) - c_2 + c_5 + c_7 - c_9)^2}{(1 - c_0 - c_1 - c_2 - c_3)((1 - p) - c_2 - c_3 + c_5 + c_7 + c_8)}$$

$$+ \frac{(qu - d_0 - d_4 - d_5 - d_6)^2}{(1 - d_0 - d_1 - d_2 - d_3)(q - d_0 - d_1 - d_5 - d_6 - d_7 - d_8)}$$

$$+ \frac{(pg + (1 - p)h - qu - c_0 - c_2 - c_4 - c_5 - c_6 + d_0 + d_4 + d_5 + d_6)^2}{(1 - c_0 - c_1 - c_2 - c_3)(1 - q - c_0 - c_1 - c_2 - c_3 + d_0 + d_1 + d_5 + d_6 + d_7 + d_8)}.\]$$

**Proof ($\Rightarrow$)** From the definition, the goodness measures of the split candidates $t_0$ and $t$ at the nodes $n$ and $n'$ using the gini index are

$$good_{n,D}(t_0) = 2f(1 - f) - 2pg(1 - g) - 2(1 - p)h(1 - h),$$

$$good_{n,D}(t) = 2f(1 - f) - 2qu(1 - u) - 2(1 - q)v(1 - v),$$

$$good_{n',D'}(t_0) = 2f'(1 - f') - 2p'g'(1 - g') - 2(1 - p')h'(1 - h'),$$

$$good_{n',D'}(t) = 2f'(1 - f') - 2q'u'(1 - u') - 2(1 - q')v'(1 - v').$$

37
The difference of the goodness measures of \( t_0 \) and \( t \) at the node \( n' \) can be expressed as

\[
good_{n',D'}(t_0) - good_{n',D'}(t) = good_{n,D}(t_0) - good_{n,D}(t) - 2(K_0 - K_1),
\]

where

\[
K_0 = pg^2 + (1 - p)h^2 - [p'g'^2 + (1 - p')h'^2], \quad \text{and} \quad K_1 = qu^2 + (1 - q)v^2 - [q'u'^2 + (1 - q')v'^2].
\]

At the node \( n' \), \( t \) is a better split than \( t_0 \). Thus, \( good_{n',D'}(t_0) \leq good_{n',D'}(t) \).

Therefore,

\[
good_{n,D}(t_0) - good_{n,D}(t) \leq 2(K_0 - K_1).
\]

From Lemma 4, the new class \( '+' \) frequencies in \( D'_{t_0,L}, D'_{t_0,R}, D'_{t,L}, D'_{t,R} \) are

\[
g' = \frac{gp|D|-(c_0+c_4+c_5+c_6)|D|}{|D'_{t_0,L}|},
\]

\[
h' = \frac{h(1-p)|D|-(c_2-c_5-c_7+c_9)|D|}{|D'_{t_0,R}|}, \quad \text{and}
\]

\[
u' = \frac{qu|D|-(d_0+d_4+d_5+d_6)|D|}{|D'_{t,R}|},
\]

where \( c_i \) and \( d_i \) , \( i = 0, \ldots, 9 \), represent the effect of the change \( c \) to the class distribution in the child subsets of \( D \) divided by \( t_0 \) and \( t \), respectively.

Since \( g'\cdot p' + h' \cdot (1 - p') = u' \cdot q' + v' \cdot (1 - q') = \text{the fraction of the objects in } D' \text{ belonging to class } '+' \), the new class frequency for the class \( '+' \) in \( D'_{t,R} \) is \( v' = \frac{g'\cdot p' + h' \cdot (1 - p') - u' \cdot q'}{1 - q'} \).

Similarly, \( v = \frac{g\cdot p + h \cdot (1 - p) - u \cdot q}{1 - q} \).

Thus,

\[
\varepsilon(c) = \varepsilon(c_0, \ldots, c_9, d_0, \ldots, d_9) = 2(K_0 - K_1)
\]

\[
= 2[pg^2 + (1 - p)h^2 - p'g'^2 - (1 - p')h'^2 - qu^2 - (1 - q)v^2 + q'u'^2 + (1 - q')v'^2]
\]

\[
= 2[pg^2 + (1 - p)h^2 - qu^2 - (1 - q)(\frac{pg+(1-p)h-qu}{1-q})^2
\]

\[
- p^c_0-c_1-c_2-c_5-c_7-c_9 \cdot \left( \frac{gp-c_0-c_4-c_5-c_6}{p-c_0-c_1-c_2-c_5-c_7-c_9} \right)^2
\]

\[
- (1 - p^c_0-c_1-c_2-c_5-c_7-c_9) \cdot \left( \frac{h(1-p)-c_2+c_5+c_7-c_9}{(1-p)-c_2-c_5+c_6+c_7+c_9} \right)^2
\]

38
\[ + \frac{q-d_0-d_1-d_2-d_3-d_7-d_8}{1-d_0-d_1-d_2-d_3} \cdot \left( \frac{qu-d_0-d_4-d_5-d_6}{q-d_0-d_1-d_2-d_3-d_7-d_8} \right)^2 \\
+(1-q') \cdot \left( p'q' + \frac{(1-p')h' - q'u'}{1-q'} \right)^2 \]
\[ = 2[pq^2 + (1-p)h^2 - qu^2 - \frac{(pg+(1-p)h-gu)^2}{1-q} \]
\[ - \frac{(gp-c_0-c_4-c_5-c_6)^2}{(1-c_0-c_1-c_2-c_3)(p-c_0-c_1-c_5-c_6-c_7-c_8)} - \frac{(h(1-p)-c_2+c_5+c_7-c_9)^2}{(1-c_0-c_1-c_2-c_3)((1-p)-c_2-c_3+c_5+c_6+c_7+c_8)} \]
\[ + \frac{(qu-d_0-d_4-d_5-d_6)^2}{(1-d_0-d_1-d_2-d_3)(q-d_0-d_1-d_2-d_3-d_7-d_8)} + \frac{(p'q' + (1-p')h' - q'u')^2}{1-q'} \]
\[ = 2[pq^2 + (1-p)h^2 - qu^2 - \frac{(pg+(1-p)h-gu)^2}{1-q} \]
\[ - \frac{(gp-c_0-c_1-c_2-c_3)(p-c_0-c_1-c_5-c_6-c_7-c_8)}{(1-c_0-c_1-c_2-c_3)} - \frac{(h(1-p)-c_2+c_5+c_7-c_9)^2}{(1-c_0-c_1-c_2-c_3)((1-p)-c_2-c_3+c_5+c_6+c_7+c_8)} \]
\[ + \frac{(qu-d_0-d_4-d_5-d_6)^2}{(1-d_0-d_1-d_2-d_3)(q-d_0-d_1-d_2-d_3-d_7-d_8)} + \frac{(p'q' + (1-p')h - q'o)^2}{(1-c_0-c_1-c_2-c_3)(1-q-c_0-c_1-c_2-c_3+d_0+d_1+d_5+d_6+d_7+d_8)} \]

\[ \Leftrightarrow \quad \text{The proof is similar to the proof in theorem 1.} \]

The function $\varepsilon$ in Theorem 1, which assumes that changes are made to the class labels only, is a special case of the general model. By setting the parameters $c_i$ and $d_i$ to zero, except for $c_4$, $c_9$, $d_4$, and $d_9$, the function $\varepsilon$ in Theorem 3 is the same as the function $\varepsilon$ in Theorem 1. Theorem 2 remains the same for the general case except the inequality 3.1 is changed to the inequality 3.2.

### 3.3 Summary

In summary, we have investigated the relationship between a change in a data set and the resulting split change. To study the impact of a change on the goodness of splits, we analyzed the effect of the change to the class frequencies in the data and the child subsets. Theorem 1 proves that if a change $c$ in $D$ results in a different split $t$ being selected, then the closeness of the best split in $D$, $t_0$, and $t$ is less than or equal to $\varepsilon(c)$. Conversely, if the inequality 3.1 holds for a change $c$ and a
split \( t \), then \( t \) will be superior to \( t_0 \) if \( c \) is applied in \( D \).

Based on Theorem 1, we defined the sensitivity of \( t_0 \) to \( t \) as the size of the smallest change that satisfies the inequality 3.1. In other words, the sensitivity of \( t_0 \) to \( t \) is the smallest fraction change in \( D \) that may make \( t \) superior to \( t_0 \). We defined splits that are almost as good as \( t_0 \) with respect to a fraction \( c \) as those that may be superior to \( t_0 \) if a fraction \( c \) of \( D \) is changed.

Theorem 2, the instability theorem, proves that the split selection among almost equally good splits is the cause of the instability of decision tree classifiers. Theorem 3 shows the relationship of a change and the resulting split change for the general case, where changes can be deletion, addition, or update of objects.

Knowing the cause of the instability problem of decision tree classifiers, improvements can be made to lessen the problem. In the next chapter, we present a method to identify almost equally good splits and propose a split selection method that can be adapted to the existing decision tree classifiers to lessen the instability problem. The idea of the proposed method is to consider almost equally good split candidates as having the same degree of importance in determining the splitting predicate at a node.
Chapter 4

A Stable Decision Tree Classifier

The Instability Theorem in Chapter 3 shows that if there are a set of almost equally good splits at a node, then the selection of one split to partition the data results in a sensitive node. That is, the split of the sensitive node is likely to be replaced as a result of a small change in the training sample. In this chapter, we present the methods to compute the sensitivity of a node, and to build a stable tree.

The sensitivity of a tree is determined by the split of the node with the smallest sensitivity. In section 4.1.1, we show how to find the sensitivity of the split of a node. Once the sensitivities of all nodes are computed, the sensitivity of the tree is decided.

The solution we propose to lessen the instability problem is to consider the splits of the same degree of importance when choosing the splitting predicate of a node. Theorem 1 in Chapter 3 defines a measurement for splits that are almost equally good, which is used in section 4.2.1 to identify almost equally good splits. Then, a split selection method that forms the splitting predicate of a node based on almost equally good splits is presented in section 4.2.2. The proposed split selection method can be adapted to current tree classification algorithms to lessen the instability problem. A stable tree growth algorithm is presented in section 4.2.3.
4.1 The Sensitivity of a Decision Tree Classifier

In this section, we present a method to compute the sensitivity of decision tree classifiers. The sensitivity of a tree is the smallest fraction change in the sample that will alter the tree structure. In other words, the sensitivity of a tree is determined by the most sensitive node. To find the sensitivity of a tree, the first step is to compute the sensitivities of the splits of all the nodes in the tree. After the sensitivities of all the nodes are computed, the smallest sensitivity is the sensitivity of the tree. The algorithm to measure tree sensitivity is shown in Figure 4.1.

```
TreeSensitivity (Tree T) {
    tree_sensitivity = 1;

    // Compute the sensitivity of the split of each node
    for (each node n ∈ T) {
        l₀ = Split (n);
        split_sensitivity = SplitSensitivity (l₀);
        if (split_sensitivity < tree_sensitivity)
            tree_sensitivity = split_sensitivity;
    }

    return tree_sensitivity;
}
```

Figure 4.1: Algorithm to compute the sensitivity of a tree

The tree sensitivity is an indicator of the level of confidence in the classification rules. If a tree is highly sensitive, then the decisions made based on the rules may not be reliable. The sensitivity of a tree is also a good indicator of the effective time of validity of the classification rules. When changes to the sample are greater than the sensitivity of the tree, the rules may need to be reconstructed to reflect the changes.

In the next section, we show how to compute the sensitivity of the split of a node.
4.1.1 The Sensitivity of the Split of a Node

Recall that the sensitivity of the split of a node is the size of the smallest change (the fraction $c$ of the data changed) that will cause the split to be replaced. Therefore, the split is insensitive to changes less than $c$. For example, if the smallest change to cause the split $t_0$ of a node to be replaced is a change in 5% of the data, then $t_0$ remains unchanged for any change in less than 5% of the data. If the fraction of the data changed is 5% or more, it is possible that $t_0$ will be replaced.

Assume that the split candidates at a node are $t_0$, $t_1$, $t_k$, and $t_0$ is the best split. To measure the sensitivity of $t_0$, the first step is to compute the sensitivities of $t_0$ to $t_i$, $i = 1, \ldots, k$. That is, compute the smallest change that will have $t_0$ replaced by $t_i$. The sensitivity of $t_0$ is then the smallest one of the sensitivities of $t_0$ to $t_i$, $i = 1, \ldots, k$. The algorithm to measure the sensitivity of the split of a node is shown in Figure 4.2. Next, we will show how to compute the sensitivity of $t_0$ to $t_i$.

```plaintext
SplitSensitivity (Split $t_0$) {
    // the sensitivity of $t_0$ to $t_i$
    for $i = 1$ to $k$
        $\sigma_i =$ RelativeSensitivity ($t_0$, $t_i$);
    sensitivity = min ($\sigma_1$, ..., $\sigma_k$);
    return sensitivity;
}
```

Figure 4.2: Algorithm to compute the sensitivity of $t_0$

The Sensitivity of $t_0$ to $t$

In Chapter 3, the function $\varepsilon$ defines the relationship between the split change and the change in the sample. For the purpose of simplicity and illustration, we use the function $\varepsilon$ in Theorem 1.

From Theorem 1, if there is a change $c$ such that the inequality $good_{n,D}(t_0) - good_{n,D}(t) \leq \varepsilon(c) = \varepsilon(c_0, c_1, c_2)$ holds, then $t$ is superior to $t_0$ in the data obtained by applying the change.
specified by the parameters $c_0$, $c_1$, and $c_2$ to the data set $D$. The problem to find the smallest change that satisfies the inequality; i.e. sensitivity of $t_0$ to a split $t$, can be formalized as follows:

\[\text{Minimize} \quad \text{the change } c \text{ represented by } c_0, c_1, \text{ and } c_2\]

\[\text{Subject to} \quad \text{good}_{n,D}(t_0) - \text{good}_{n,D}(t) \leq \varepsilon(c) = \varepsilon(c_0, c_1, c_2),\]

where
\[
0 \leq |c_0| + |c_1| \leq 1,
0 \leq |c_2| + |c_0 + c_1 - c_2| \leq 1,
(1 - g) \cdot p \leq c_0 \leq g \cdot p,
(1 - h) \cdot (1 - p) \leq c_1 \leq h \cdot (1 - p),
(1 - u) \cdot q \leq c_2 \leq u \cdot q.
\]

The constraints on $c_0$, $c_1$, and $c_2$ come from the following facts:

- The effect of a change $c$ to the class distributions in the subsets of $D$ divided by $t_0$ are that at least $|c_0| \cdot |D|$ of the objects in $D_{t_0,L}$ and at least $|c_1| \cdot |D|$ of the objects in $D_{t_0,R}$ change classes. The signs of $c_0$ and $c_1$ indicate the class change from $'+'$ to $'-'$ or from $'-'$ to $'+'$. Hence, the change $c$ is made to at least the fraction $|c_0| + |c_1|$ of the data, i.e., $|c_0| + |c_1| \leq |c|$. Since the size of a change (i.e., the fraction of the objects changed) is at most 1, $|c_0| + |c_1| \leq |c| \leq 1$.

- Similarly, the change $c$ has at least $|c_2| \cdot |D|$ of the objects in $D_{t,L}$ and at least $|c_3| \cdot |D|$ of the objects in $D_{t,R}$ change classes. Hence, the change $c$ is made to at least the fraction $|c_2| + |c_3|$ of the data, i.e., $|c_2| + |c_3| \leq |c|$. From Lemma 3, $c_3 = c_0 + c_1 - c_2$. Therefore, $|c_2| + |c_0 + c_1 - c_2| \leq |c| \leq 1$. Thus, the size of the smallest change $c$ specified by $c_0$, $c_1$, and $c_2$ is $\max(|c_0| + |c_1|, |c_2| + |c_0 + c_1 - c_2|)$.

- The parameter $c_0$ represents the fraction of the objects that change classes in $D_{t_0,L}$. The number of the objects that can change classes from $'+'$ to $'-'$ in $D_{t_0,L}$ is at most $g \cdot p \cdot |D|$, which is the number of the objects belonging to the class $'+'$ in $D_{t_0,L}$. Therefore, $c_0 \leq g \cdot p$. Since the maximum number of the objects that can change classes from $'-'$ to $'+'$ in $D_{t_0,L}$ is the number
of the objects that belong to the class \( -1 \), \( c_0 \geq -(1 - g) \cdot p \). Thus, \(-(1 - g) \cdot p \leq c_0 \leq g \cdot p\).

The constraints for \( c_1 \), and \( c_2 \) are obtained in the similar way.

The process to approximate the smallest change \( c \) that satisfies the inequality 3.1 is to check the possible combinations of the parameters \( c_0 \), \( c_1 \), and \( c_2 \), and choose the combination that corresponds to the smallest change as the estimate of the smallest change. The worst case is that the smallest change won't be found until all the combinations are checked; that is, 100% change in the training sample. Since the constraints for \( c_0 \), \( c_1 \), and \( c_2 \) are \(-(1 - g) \cdot p \leq c_0 \leq g \cdot p\), \(-(1 - p) \cdot (1 - h) \leq c_1 \leq (1 - p) \cdot h\), and \(-(1 - u) \cdot q \leq c_2 \leq u \cdot q\), the number of the possible values for \( c_0 \), \( c_1 \), and \( c_2 \) are \((1 - g) \cdot p \cdot |D| + g \cdot p \cdot |D|\), \( (1 - p) \cdot (1 - h) \cdot |D| + (1 - p) \cdot h \cdot |D|\), and \((1 - u) \cdot q \cdot |D| + u \cdot q \cdot |D|\), respectively. Therefore, the number of possible combinations are \( p \cdot (1 - p) \cdot q \cdot |D|^3\). Thus, the sensitivity of \( t_0 \) to \( t \) can be found in \( O(|D|^3) \) time.

```
RelativeSensitivity (Split \( t_0 \), \( t \)) {
    Compute the function \( \varepsilon(c) = \varepsilon(c_0, c_1, c_2) \) for \( t_0 \) and \( t \)

    // Find a combination that corresponds to the smallest
    // change and satisfies the inequality (3.1).
    for ( \( i = 1 \) to \( |D| \) )
        for (each combination \( (c_0, c_1, c_2) \) that corresponds to changes in \( i \) objects)
            If (the combination \( (c_0, c_1, c_2) \) satisfies the inequality 3.1)
                return \( i/|D| \);
}
```

Figure 4.3: Algorithm to compute the sensitivity of \( t_0 \) to \( t \)

The search for the smallest change is to generate the possible combinations of \( c_0 \), \( c_1 \), and \( c_2 \) that correspond to changes in one object, and check those combinations to see if there exists one satisfying the inequality 3.1. If none of the combinations satisfy the inequality, the possible combinations of \( c_0 \), \( c_1 \), and \( c_2 \) corresponding to changes in two objects are generated and checked to see if there exists one satisfying the inequality. The process continues until a change \( (c_0, c_1, c_2) \) is
found. The size of the change (i.e., $\max(|c_0| + |c_1|, |c_2| + |c_0 + c_1 - c_2|)$) is the sensitivity of $t_0$ to $t$. The algorithm to compute the sensitivity of $t_0$ to $t$ is shown in Figure 4.3.

### 4.1.2 An Example

| split candidate | goodness measure | the computed smallest change $c = (c_0, c_0, c_0)$ | the computed sensitivity $|c|$ |
|-----------------|------------------|-----------------------------------------------|------------------|
| "# of years ≤ 1.5" | 0.08             | (0.2, -0.2, 0.2)                                | 40%              |
| "# of years ≤ 3"  | 0.48             |                                                |                  |
| "# of years ≤ 6"  | 0.18             | (0, -0.2, -0.2)                                | 20%              |
| "sports car"      | 0.21             | (0.2, 0, 0)                                    | 20%              |

Table 4.1: The computed sensitivities of the best split "# of Years ≤ 3" to splits "# of Years ≤ 1.5", "# of Years ≤ 6", and "sports car"

To demonstrate how to estimate the sensitivity of a split and the sensitivity of a tree, consider again the five-object training sample in Figure 1.1(a) and the tree grown on the sample shown in Figure 1.1(b).

Recall that the four split candidates at the root are "# of Years ≤ 1.5", "# of Years ≤ 3", "# of Years ≤ 6", and "sports car". The goodness of each split candidate is shown in the second column of the Table 4.1. The split "# of Years ≤ 3" is the best split because it has the largest goodness measure.

To find the smallest change that will make the split "# of Years ≤ 3" no longer the best split in the changed sample, compute the function $\varepsilon$ for every pair of the best split and other split candidates. Use the two splits "# of Years ≤ 1.5" and "# of Years ≤ 3" as an example. One
of the partitions of the sample divided by "\# of Years ≤ 3" contains three high risk objects, and
the other one contains two low risk (no high risk) objects. Therefore, \( p = \frac{3}{5}, \ g = 1, \) and \( h = 0. \)
The split "\# of Years ≤ 1.5" partitions the sample into one homogeneous subset with one high
risk object and one inhomogeneous subset with two objects in each class. Therefore, \( q = \frac{1}{5}, \ u = 1, \) and \( v = \frac{2}{4}. \) The function \( \varepsilon \) "\# of Years ≤ 1.5" for the two splits is

\[
\varepsilon \) "\# of Years ≤ 1.5" (c) = \varepsilon (c_0, c_1, c_2) = 2[c_0 - \frac{5}{2}c_0^2 - c_1 - \frac{5}{2}c_1^2 + \frac{5}{4}(c_0 + c_1 - c_2)^2 - c_2 + 5c_2^2].
\]

Similarly, the function \( \varepsilon \) for the other pairs of splits are

\[
\varepsilon \) "\# of Years ≤ 6" (c) = \varepsilon (c_0, c_1, c_2) = 2[2c_0 - \frac{5}{2}c_0^2 - \frac{5}{2}c_1^2 + 5(c_0 + c_1 - c_2)^2 - \frac{3}{2}c_2 + \frac{5}{4}c_2^2], \text{ and}
\]

\[
\varepsilon \) "sports car" (c) = \varepsilon (c_0, c_1, c_2) = 2[2c_0 - \frac{5}{2}c_0^2 - \frac{5}{2}c_1^2 - \frac{2}{3}(c_0 + c_1 - c_2) + \frac{5}{3}(c_0 + c_1 - c_2)^2 - 2c_2 + \frac{5}{3}c_2^2].
\]

If a change that satisfies the inequality \( \varepsilon \) "\# of Years ≤ 1.5" (c) − (0.48 − 0.08) ≤ 0, is made to
the sample, then "\# of Years ≤ 1.5" is superior to "\# of Years ≤ 3" in the changed sample.
The search space for the smallest change that satisfies the inequality is bounded by \( 0 ≤ c_0 ≤ \frac{3}{5}, \)
\( -\frac{2}{5} ≤ c_1 ≤ 0, \) and \( 0 ≤ c_2 ≤ \frac{1}{5}. \)

To find the smallest change, the algorithm checks if there is any combination which corresponds
to changes in one object and satisfies the inequality. If no such change exists, the combinations
corresponding to change in two objects are checked. The process continues until a change that
satisfies the inequality is found.

As the result, the computed smallest change to have "\# of Years ≤ 3" replaced by "\# of Years ≤ 1.5" is described as \( (c_0, c_1, c_2) = (0.2, -0.2, 0.2) \) which corresponds to changes in 40% of the sample.
Therefore, the computed sensitivity of "\# of Years ≤ 3" to "\# of Years ≤ 1.5" is 40%. The
computed change is that if the class of an object with driving experience less than or equal to 1.5
years is changed from high risk to low risk, and the class of an object with driving experience more
Figure 4.4: An example of the computed change that will make the split "# of Years ≤ 1.5" superior to the split "# of Years ≤ 3" in the changed sample.

than three years is changed from low risk to high risk, then the split "# of Years ≤ 1.5" will be as good as or better than the split "# of Years ≤ 3".

To verify if the computed change will make the split "# of Years ≤ 1.5" at least as good as the split "# of Years ≤ 3", consider a change corresponding to the combination (0.2, −0.2, 0.2) that the class labels of the first and the last objects in the sample are changed. The changed sample is shown in Figure 4.4(a). Figure 4.4(b) and (c) show the class distributions in the subsets of the original sample and the changed sample divided by "# of Years ≤ 3" and "# of Years ≤ 1.5", re-
spectively. The dotted lines show the effect of the change described by \((c_0, c_1, c_2) = (0.2, -0.2, 0.2)\) to the class distributions in the child subsets of the original sample.

The goodness measures of the two splits in the changed sample are

\[
good \left( \text{"# of Years } \leq 1.5\text{"} \right) = 0.18, \text{ and} \nonumber \]
\[
good \left( \text{"# of Years } \leq 3\text{"} \right) = 0.013. \nonumber \]

This confirms that \("# of Years \leq 1.5"\) is a split better than \("# of Years \leq 3"\) in the changed sample. It is easy to verify that a change in two objects is the smallest change that makes \("# of Years \leq 1.5"\) better than \("# of Years \leq 3"\) in the changed sample.

The computed smallest changes and sensitivities of \("# of Years \leq 3"\) to splits \("# of Years \leq 6"\) and \("sports car"\) can be obtained in the similar way and are shown in the third and fourth columns of Table 4.1.

Since the smallest change that causes the best split \("# of Years \leq 3"\) to be replaced is a change in one record, the sensitivity of the root in Figure 1.1(b) is 20%. Thus, the tree sensitivity is 20%.

\[ \square \]

4.2 Building a Stable Decision Tree Classifier

The Instability Theorem proves that the split selection among a set of almost equally good splits is the cause of the instability of decision tree classifiers. To lessen the instability problem, we propose a split selection algorithm that forms the splitting predicate of a node based on the splits of the same importance (almost equally good splits).

In this section, we first present the algorithm to identify the splits that are almost as good as the best split \(t_0\). Then, a split selection method that has taken the almost equally good splits into
account when constructing the splitting predicate of a node is proposed. A stable tree classification algorithm is given last.

4.2.1 Almost Equally Good Splits

As defined in section 3.2.5, a split $t_i$ is almost as good as the best split $t_0$ with respect to a fraction $c$ if the sensitivity of $t_0$ to $t_i$ is smaller than or equal to $c$. In other words, if there exists a change smaller than $c$ that will make $t_i$ superior to $t_0$ in the changed sample, then $t_i$ is almost as good as $t_0$. From the previous section, we already know how to find the sensitivity of $t_0$ to $t_i$. Therefore, if the sensitivity of $t_0$ to $t_i$ is no larger than $c$, then $t_i$ is a split almost as good as $t_0$. The algorithm to identify the set of almost equally good splits is shown in Figure 4.5.

```
AlmostEquallyGoodSplits (Split $t_0$, Fraction $c$) {
    // Initialize the set of almost equally good splits
    GoodSplits ← {};
    for (each split candidate $t_i$)
        $\sigma_i = \text{relative\_sensitivity} (t_0, t_i)$;
        if ($\sigma_i \leq c$)
            GoodSplits ← GoodSplits ∪ \{ $t_i$ \};
    return GoodSplits;
}
```

Figure 4.5: Algorithm to identify splits that are almost as good as $t_0$ with respect to the change fraction $c$

From section 4.1.1, relative_sensitivity has the computational complexity $O(|D|^3)$, where D is the data associated with the node. If there are $k$ split candidates, the complexity of AlmostEquallyGoodSplits is $O(k \cdot |D|^3)$.

The problem with this approach is that it is computationally expensive. First, there is no need
to check all the split candidates to find splits that are almost as good as \( t_0 \). That is, the size of the for loop can be cut down. It is intuitive that splits with close goodness measures should be very sensitive to each other. Therefore, the loop size is cut down by checking only some of the split candidates. For example, users can specify the number of the split candidates to be checked, or define a threshold to check only the split candidates with goodness measures greater than the predefined threshold. Since we only expect a small number of almost equally good splits at a node, the number of split candidates that need to be checked are small. Thus, the performance can be improved. Second, there is no need to compute the sensitivity of \( t_0 \) to \( t_i \), we only need to check if there exists a change less than or equal to \( c \) that will make \( t_i \) superior to \( t_0 \). If no such change exists for \( t_i \), then \( t_i \) is not a split almost as good as \( t_0 \). The revised algorithm is shown in Figure 4.6.

```cpp
AlmostEquallyGoodSplits (Split \( t_0 \), Fraction \( c \)) {

   // Initialize the set of almost equally good splits
   good_splits = \{
   \};

   // Filter split candidates
   good_split_candidates = the splits that are close to \( t_0 \);

   for (each split \( t_i \) in good_split_candidates) {
      if (there exists a change smaller than \( c \) that satisfies the inequality 3.1)
         good_splits = good_splits \cup \{ t_i \};
   }

   return good_splits;
}
```

Figure 4.6: Revised algorithm to identify splits that are almost as good as \( t_0 \) with respect to the change fraction \( c \)

The running time of the revised algorithm depends on the number of split candidates that need to be checked and how long it takes to check whether there exists a change smaller than \( c \) that causes \( t_0 \) to be replaced. The worst case is that there are no splits almost as good as \( t_0 \). Assume that there are \( k_0 \) splits in good_split_candidates. Since \(|c_0| + |c_1| \leq c \) and \(|c_2| \leq c \), the number of the possible combinations of \( c_0, c_1, \) and \( c_2 \) are \( O(c^3 \cdot |D|^3) \). Therefore, the revised algorithm runs in \( O(k_0 \cdot c^3 \cdot |D|^3) \).
Having identified the set of almost equally good splits at a node, in next section we discuss how to form a splitting predicate based on the almost equally good splits.

4.2.2 The Splitting Predicate of a Node

The problem with the selection of one split as the splitting predicate at a node is that it may result in not only the instability of the generated rules but also loss of information about other equally good decision factors.

Instead of using the split with the greatest goodness measure to partition the data, we use the best splitting predicate formed by almost equally good splits. Assume that there are $k$ almost equally good splits at a node. To form the splitting predicate, it is straightforward to generate and evaluate all the possible predicates formed by the $k$ almost equally good splits, and then pick the best one as the splitting predicate by which to partition the data.

In general, if there are $k$ independent almost equally good splits, then there are $2^k$ different combinations formed by the $k$ splits, because each split is either true or false. Define the basis predicates as the possible combinations of the almost equally good splits. The number of possible splitting predicates formed by the $2^k$ basis predicates is $2^{2^k}$. Since half of them are the negation of the other half, the number of splitting predicates formed by the $k$ almost equally good splits is $2^{2^k} - 1$.

Figure 4.7 shows an example of the possible splitting predicates formed by two independent splits. The basis predicates are shown in the first column of the table in Figure 4.7(a). The values (0/1) in each column indicate whether the corresponding basis predicate is in the splitting predicate or not. For example, splitting predicate number 3 is the disjunction of the first and the second basis predicates. Note that the splitting predicates 0 and 15, 1 and 14, 2 and 13, ... are negations of each other. Therefore, the total number of splitting predicates formed by the two independent almost equally good splits is 8. The possible splitting predicates, which are disjunction of the basis

52
predicates, are shown in Figure 4.7(b).

Two splits are independent if they are based on different splits; otherwise, they are dependent. For instance, the split candidates "# of years ≤ 3" and "# of years ≤ 6" in the previous example are dependent splits, and "# of years ≤ 3" and "sports car" are independent splits. For a set of almost equally good splits which consist of both independent and dependent splits, we generate the possible combinations for the dependent splits and the independent splits, and combine them together as the basis predicates. The combinations of the dependent splits depend on the semantic meaning of the splits. Using the above example, the possible combinations of the splits with the number of years driving experience less than 3 and 6 years are the number of years driving experience no greater than 3 years, greater than 3 years, between 3 and 6 years, no greater than 6 years, and greater than 6 years.

The advantage of the approach by exhaustively search for the best splitting predicate is completeness. That is, it guarantees to find the best splitting predicate if one exists. The drawback with this approach is that it is computationally expensive for a large number of almost equally good splits. However, in general, we only expect a very small number of the almost equally good splits at a node. Therefore, it is feasible to do exhaustively search for a very small number of almost equally good splits. There are various ways to deal with a moderate number of almost equally good splits. For instance, if there are five almost equally good splits at a node, to compromise the computational cost, we can use the best splitting predicate formed by two almost equally good splits to partition the data. For those almost equally good splits that are not used to partition the data, they may be discovered at the next level. In the following, we present a method to form splitting predicates based on the basis predicates.

The proposed split selection method is based on the observation that any splitting predicate is formed by the disjunction of the basis predicates. The algorithm to find the splitting predicate of a node is described as follows:

1. Generate the split candidates and find the best split \( t_0 \).
(a) The $2^4$ possible splitting predicates formed by $t_1$ and $t_2$

<table>
<thead>
<tr>
<th>Basis Predicates</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1 \land t_2$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$t_1 \land \neg t_2$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\neg t_1 \land t_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\neg t_1 \land \neg t_2$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

(b) Each splitting predicate is formed by the disjunction of the basis predicates

<table>
<thead>
<tr>
<th>Splitting Predicate</th>
<th>The Simplified Logically Equivalent Splitting Predicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 False</td>
<td>False</td>
</tr>
<tr>
<td>1 $(t_1 \land t_2)$</td>
<td>$(t_1 \land t_2)$</td>
</tr>
<tr>
<td>2 $(t_1 \land \neg t_2)$</td>
<td>$(t_1 \land \neg t_2)$</td>
</tr>
<tr>
<td>3 $(t_1 \land t_2) \lor (t_1 \land \neg t_2)$</td>
<td>$t_1$</td>
</tr>
<tr>
<td>4 $(\neg t_1 \land t_2)$</td>
<td>$(\neg t_1 \land t_2)$</td>
</tr>
<tr>
<td>5 $(t_1 \land t_2) \lor (\neg t_1 \land t_2)$</td>
<td>$t_2$</td>
</tr>
<tr>
<td>6 $(t_1 \land \neg t_2) \lor (\neg t_1 \land t_2)$</td>
<td>$(t_1 \land \neg t_2) \lor (\neg t_1 \land t_2)$</td>
</tr>
<tr>
<td>7 $(t_1 \land t_2) \lor (t_1 \land \neg t_2) \lor (\neg t_1 \land t_2)$</td>
<td>$t_1 \lor t_2$</td>
</tr>
<tr>
<td>8 $(\neg t_1 \land \neg t_2)$</td>
<td>$(\neg t_1 \land \neg t_2)$</td>
</tr>
<tr>
<td>9 $(t_1 \land t_2) \lor (\neg t_1 \land \neg t_2)$</td>
<td>$(\neg t_1 \lor t_2) \land (t_1 \lor \neg t_2)$</td>
</tr>
<tr>
<td>10 $(t_1 \land \neg t_2) \lor (\neg t_1 \land \neg t_2)$</td>
<td>$\neg t_2$</td>
</tr>
<tr>
<td>11 $(t_1 \land t_2) \lor (t_1 \land \neg t_2) \lor (\neg t_1 \land \neg t_2)$</td>
<td>$(t_1 \lor \neg t_2)$</td>
</tr>
<tr>
<td>12 $(\neg t_1 \land t_2) \lor (\neg t_1 \land \neg t_2)$</td>
<td>$\neg t_1$</td>
</tr>
<tr>
<td>13 $(t_1 \land t_2) \lor (\neg t_1 \land t_2) \lor (\neg t_1 \land \neg t_2)$</td>
<td>$(\neg t_1 \lor t_2)$</td>
</tr>
<tr>
<td>14 $(t_1 \land \neg t_2) \lor (\neg t_1 \land t_2) \lor (\neg t_1 \land \neg t_2)$</td>
<td>$(\neg t_1 \lor \neg t_2)$</td>
</tr>
<tr>
<td>15 $(t_1 \land t_2) \lor (t_1 \land \neg t_2) \lor (\neg t_1 \land t_2) \lor (\neg t_1 \land \neg t_2)$</td>
<td>True</td>
</tr>
</tbody>
</table>

Figure 4.7: An example of the possible splitting predicates formed by two almost equally good splits
2. Identify splits that are almost as good as $t_0$ with respect to a predefined fractional data change $c$.

3. Form the basis predicates from the set of almost equally good splits.

4. Discard basis predicates with goodness measures less than that of $t_0$.

5. Form the predicate that is the disjunction of the remaining "good" basis predicates.

6. Use the predicate with the largest goodness measure from $t_0$, the good basis predicates from step 4, and the predicate from step 5, as the final splitting predicate by which to partition the data.

```cpp
SplittingPredicate(Data D, Fraction c) {
    split_candidates = {}; //step 1
    for (each attribute A)
        split_candidates = split_candidates ∪ Splits(A);
    $t_0 = \text{BestSplit}(split\_candidates)$;

    good_splits = \text{AlmostEquallyGoodSplits}($t_0$, c); //step 2

    basis_predicates = \text{BasisPredicates}(good_splits); //step 3

    good_basis_predicates = {};
    for (each $b \in basis\_predicates$)
        if (goodness($b$) ≥ goodness($t_0$))
            good_basis_predicates = good_basis_predicates ∪ {$b$};

    disj predicate = \text{Disjunction}(good_basis_predicates); //step 5

    final_splitting_predicate = \text{BestSplit}($t_0$, good_basis_predicates, disj predicate);
    return final_splitting_predicate;
}
```

Figure 4.8: Algorithm of forming the splitting predicate at a node

Figure 4.8 shows the algorithm to form the splitting predicate at a node. The computational complexity is dominated by steps 1, 2, 4, and 6. In Step 1, the split selection algorithm scans the data once to find the best split $t_0$; thus, the complexity is $O(|D|)$. Step 2, which identifies almost equally good splits at a node, runs in $O(|D|^3)$. Step 4, the evaluation of the basis predicates,
requires a scan of the data. The evaluation of the disjunction of basis predicates in step 6 requires another scan of the data. Thus, the running time of SplittingPredicate is $O(\left| D \right|^3 + 3 \cdot Time_{scan \ D})$.

### 4.2.3 A Stable Decision Tree Classifier

To build a tree insensitive to a fractional change $c$ in the sample, we use the splitting predicate formed by one or more almost equally good splits to partition the data at a node. The process repeats for each partition until all the objects within the partition belong to the same class. Figure 4.9 shows the proposed tree growth algorithm. In the subset $S_i$, a change in $c \cdot |S|$ objects is the fraction $\frac{c \cdot |S|}{|S|}$ of the objects in $S_i$ changed.

```plaintext
GrowStableTree (Data $S$, Fraction $c$) {
    if ($S$ is homogeneous)
        return;
    splitting_predicate = SplittingPredicate ($S$, $c$);
    subsets = Partition ($S$, splitting_predicate);
    for (each subset $S_i \in$ subsets)
        GrowStableTree ($S_i$, $c \cdot |S_i \setminus S|$);
}
```

Figure 4.9: A Stable Tree Growth Algorithm

### An Example of Growing a Tree

To demonstrate the algorithm, consider again the sample in Figure 1.1(a). The steps to build a tree to tolerate a 20% sample change is in the following:

1. **Identify almost equally good splits.** From the example in section 4.1.2, the sensitivities of "$\# of years \leq 3$" to "$\# of years \leq 1.5$", "$\# of years \leq 6$", and "sports car" are 40%, 20%, and 20%, respectively. If we consider almost equally good splits as those that will replace the best split within one record change, then splits "$\# years \leq 6$" and "sports car" are the splits
that are as good as the best split 

\[ \# \text{years} \leq 3 \], i.e., \( \text{AlmostEquallyGoodSplits}(\# \text{years} \leq 3, 20\%) = \{ \# \text{of years} \leq 3, \# \text{years} \leq 6, \text{sports car} \}\).

2. **Form basis predicates.** 
   
   "\# of years \leq 3" and "\# of years \leq 6" are dependent splits. The possible combinations of the two dependent splits are "\# of years \leq 3", "\# of years > 3", "\# of years \leq 6", and "\# of years > 6", and "3 \leq \# of years \leq 6". The ten basis predicates and the goodness measures are shown in Figure 4.10(a).

3. **Form the final splitting predicates.** The splitting predicate formed by "\# of years > 3" and "\text{not sports car}" has the same goodness measure as "\# of years \leq 3". The split with more information is selected to partition the data. The resultant tree is shown in Figure 4.10(b).

The proposed algorithm performs better than the standard algorithm. First, the rules generated by the proposed algorithm contain information about \textit{sports car} and \textit{driving experience} compared to only one factor, \textit{driving experience}, discovered by the standard algorithm. Information loss can be reduced by the proposed algorithm due to the consideration of almost equally good splits when forming a splitting predicate for a node.

Second, the constructed tree is more stable. Recall that the sample in Figure 1.3(a) is obtained by updating the second record in Figure 1.1(a). For the two samples that differ in one record, the proposed algorithm generates the same tree, compared to the two different trees constructed by the standard algorithm.

Third and the last, the tree built is compact. A tree with almost equally good splits at a node tends to be more intuitive and concise than a tree with one split at different levels.

\[ \square \]
<table>
<thead>
<tr>
<th>Basis Predicate</th>
<th>Goodness Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| “# of years ≤ 3” \(\land\) “sports car”               | 0.213            |
| “# of years > 3” \(\land\) “sports car”               | 0                |
| “# of years ≤ 6” \(\land\) “sports car”               | 0.213            |
| “# of years > 6” \(\land\) “sports car”               | 0                |
| “3 ≤ # of years ≤ 6” \(\land\) “sports car”           | 0                |
| “# of years ≤ 3” \(\land\) “not sports car”           | 0.08             |
| “# of years > 3” \(\land\) “not sports car”           | 0.48             |
| “# of years ≤ 6” \(\land\) “not sports car”           | 0.013            |
| “# of years > 6” \(\land\) “not sports car”           | 0.18             |
| “3 ≤ # of years ≤ 6” \(\land\) “not sports car”       | 0.18             |

(a) The goodness of the basis predicates

(b) The tree grown by the proposed algorithm

Figure 4.10: The goodness of the basis predicates and the grown tree

The above example has shown that the proposed algorithm can produce more stable trees and discover more important factors at a time. However, compared to the standard tree growth algorithm, the proposed algorithm requires extra computation to identify almost equally good splits and to form the splitting predicate to partition the data. The overhead may not be as severe as it looks because the instability problem only occurs at very few nodes. When it happens, the number of almost equally good splits are usually small.
We have proposed a split forming algorithm to construct the splitting predicate to partition the data. It is still helpful to have a domain expert to resolve the ambiguity in split selection. As in the previous example, given the three almost equally good splits, a domain expert can easily see that both driving experience and sports car are important factors. The two dependent almost equally good splits based on #of years driving experience shows the ambiguity in defining experienced drivers.

4.3 Summary

Based on Theorem 1, an algorithm is proposed in section 4.1 to compute the sensitivity of a tree. The sensitivity of a tree is an indicator of the possibility of having an instability problem. It is also a good indicator of the duration of the tree's effective lifetime. When changes to a sample are greater than the sensitivity of the tree, it would be better to reconstruct the rules to reflect the changes.

Based on the Instability Theorem, section 4.2 gives a way to lessen the instability problem of decision tree classifiers by considering all the split candidates of the same degree of importance when deciding on the splitting predicate of a node. We also present a method to identify almost equally good splits based on Theorem 1. After the almost equally good splits are identified, the most promising splitting predicate formed by one or more almost equally good splits is used to partition the data. The proposed split selection method can be adapted to the existing decision tree classification algorithms to construct more stable splitting predicates, and hence lessen the instability problem. By considering all important decision factors, information loss may also be alleviated.

In the next chapter, we perform experiments to study the splitting predicate forming process, test the stability of the constructed tree classifiers, and examine information loss about almost equally good splits.
Chapter 5

Experimental Results

Since our goal is to construct stable, comprehensible, and accurate classifiers, this chapter presents an evaluation of the properties of the proposed classification tree growth algorithm. We first discuss, in section 5.1, the metrics used in the evaluation and then describe, in section 5.2, the experimental setup. To study the performance of the proposed stable tree growth algorithm, classifiers built by the proposed algorithm are compared with those built by the well-known classification algorithm CART [BFOS84]. Section 5.3.1 evaluates the splitting predicate forming process of the proposed algorithm. An instability experiment is given in section 5.3.2, and an experiment examining information loss is in section 5.3.3. Section 5.4 concludes the experiments.

5.1 Metrics

The classical performance measurements for classification algorithms are classification accuracy (i.e., the percentage of the test objects correctly classified), tree size, and the tree construction time. To evaluate the stability of tree classifiers, we compare the tree structures grown on slightly different samples. The sensitivities of tree classifiers, which are computed by the algorithm TreeSensitivity in section 4.1, can be used as a metric to evaluate the stability of tree classifiers. Since the instability problem implies the possibility of information loss, we compare the factors discovered by the two algorithms.
<table>
<thead>
<tr>
<th>Data Set</th>
<th>#Objects</th>
<th>#Attributes</th>
<th>Classification Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000</td>
<td>5</td>
<td>$A_1 \land A_2$</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>20</td>
<td>$A_1 \land A_2 \land A_3 \land A_4$</td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>5</td>
<td>$A_1 \lor A_2$</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>30</td>
<td>$A_1 \lor A_2 \lor A_3 \lor A_4$</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>10</td>
<td>$A_1 \lor (A_2 \land A_3)$</td>
</tr>
<tr>
<td>6</td>
<td>1000</td>
<td>20</td>
<td>$(A_1 \land A_2) \lor (A_3 \land A_4)$</td>
</tr>
<tr>
<td>7</td>
<td>1000</td>
<td>20</td>
<td>$[A_1 \land (A_2 \lor A_3)] \lor [A_4 \land (A_3 \lor A_5)] \lor [A_6 \land (A_2 \lor A_7)]$</td>
</tr>
</tbody>
</table>

Table 5.1: The synthetic data sets and the evaluation classification rules for the first experiment

5.2 Experiment Setup

We applied our split selection algorithm to the well known classification algorithm CART, and the tree classifiers built by the enhanced algorithm (the proposed algorithm) were compared empirically to those built by CART.

Synthetic data was used to assess the effectiveness of the proposed algorithm. Our synthetic data generator is similar to the data generator introduced by Agrawal et al. [AIS93]. A data set is formed by first randomly generating the objects, and then assigning the class labels based on the classification rules. Since decision trees are recursive, to evaluate the proposed split selection method, we need only to study the split-forming process for the representative classification rules at the root node. Without loss of the generality, the attributes in our synthetic data are boolean.

The experiment consists of three parts. The first experiment evaluates the proposed split selection algorithm and compares the tree construction time. The evaluation classification rules and the description of the data sets used in this experiment are shown in Table 5.1. To evaluate the stability of the proposed algorithm, the second experiment examines the sensitivity of classifiers to small changes. Since instability implies the possibility of information loss regarding almost equally good splits, the third experiment compares the factors discovered by CART and the proposed algorithm.

All the experiments were performed on a Dell Inspiron 5000e 700/850 MHz with 512MB main memory running Linux 2.2.16.
5.3 Performance Comparison with CART

5.3.1 Experiment I: Evaluation of the Proposed Split Selection Method and Tree Construction Time

Evaluation of the Splitting Predicate Forming Process

The splitting predicate forming process of the proposed algorithm is evaluated by comparing the sensitivities of the splitting predicates of the root nodes and tree sizes with those built by CART. We didn't compare the accuracy because the tree classifiers generated by the two algorithms in these experiments are logically equivalent. Since the data sets and the number of almost equally good splits at each node are small, the execution times of the two algorithms are nearly the same and therefore not shown.

Assume that a split is considered to be almost as good as another split if the relative sensitivity of the two splits is less than 1%. Figure 5.1 to Figure 5.6 show the tree classifiers built by the two algorithms for data sets 1–6. Each figure contains a table that shows the goodness measures of the split candidates at the root nodes, and the sensitivities of the splitting predicates of the root nodes.

Figure 5.1 shows the tree classifiers grown on data set 1, and the sensitivities of the splitting predicates of the root nodes. The computed sensitivities show that \( A_1 = 1 \) is the only split that may replace the best split \( A_2 = 1 \) in less than a 1% change to the sample, and is therefore almost as good as \( A_2 = 1 \). The conjunction of the two almost equally good splits (with sensitivity 11.1%) is more stable than \( A_2 = 1 \) (with sensitivity 0.5%).

Figure 5.2 is similar to Figure 5.1 except that there are more factors in the rules. Instead of building a four-node tree, the proposed algorithm generates a one-node tree. Figures 5.3 and 5.4 show the trees built for disjunctive rules with different numbers of factors. Figure 5.5 shows the tree grown on a training sample that has no ambiguity in split selection. In this case, both algorithms generate identical trees.
Figure 5.6 compares the trees generated for a disjunction of conjunction rules. The tree built by the proposed algorithm is more concise, intuitive, and expressive compared to the tree generated by CART. Recall that one of the problems with decision tree classifiers is the inefficient representation of disjunctive rules due to the duplicated subtrees in the constructed tree [Oliv93]. Note the two duplicated subtrees for “$A_2 = 1 \land A_1 = 1$” in the tree constructed by CART. Because “$A_3 = 1$” is almost as good as “$A_4 = 1$”, by combining the two splits into one node, the duplicate subtree is eliminated. Therefore, the proposed algorithm can mitigate the inefficient representation of disjunctive rules. In this experiment, the change fraction $c$ is 1%. If $c$ is 2%, then $A_2$, $A_3$, and $A_4$ are almost equally good. The evaluation of the combination of the almost equally good splits will pick “$A_2 \lor (A_3 \land A_4)$” to partition the data. The rest of the rules “$A_1 \lor (A_3 \land A_4)$” will be constructed at the lower levels. If $c$ is 3%, then four splits are almost equally good. By exhaustively searching all the possibilities, the best splitting predicate for the node is “$(A_1 \land A_2) \lor (A_3 \land A_4)$”.

Figure 5.7 compares the trees grown on data set 7 with the rule that is logically equivalent to the evaluation function 4 from [AIS93].

The results confirm the effectiveness of the function $e$. In the experiments where data are clean, the computed sensitivities are able to differentiate the important factors from the irrelevant ones; that is, the identified almost equally good splits are the factors in the classification rules. The computed sensitivities show that the splitting predicate formed by the almost equally good splits is more stable than the best split candidate. By putting factors of the same degree of importance at the same node, the resulting tree tends to be more expressive, intuitive, and compact.

Note that the goodness measures of almost equally good splits are very close. Although it is possible to guess the almost equally good splits based on the goodness measure, the relative sensitivity is more informative and intuitive than the goodness measure, because it shows the relationship between splits, and provides information about the possibility of making a split change.

In these experiments, the change fraction $c$ is set to 1% to identify almost equally good splits.
<table>
<thead>
<tr>
<th>Split Candidate</th>
<th>Goodness Measure</th>
<th>Sensitivity of $A_2$ to $A_i$</th>
<th>Sensitivity of $A_1 \land A_2$ to $A_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 = 1$</td>
<td>0.117527</td>
<td>0.5%</td>
<td>11.8%</td>
</tr>
<tr>
<td>$A_2 = 1$</td>
<td>0.125301</td>
<td>-</td>
<td>11.1%</td>
</tr>
<tr>
<td>$A_3 = 1$</td>
<td>0.000406</td>
<td>11.5%</td>
<td>17.8%</td>
</tr>
<tr>
<td>$A_4 = 1$</td>
<td>0.000309</td>
<td>11.8%</td>
<td>17.4%</td>
</tr>
<tr>
<td>$A_5 = 1$</td>
<td>0.000392</td>
<td>11.6%</td>
<td>17.7%</td>
</tr>
</tbody>
</table>

(a) CART  

(b) Proposed algorithm

Figure 5.1: The classifiers grown on data set 1 with rule $A_1 \land A_2$; $A_1$ and $A_2$ are identified as almost equally good splits

In general, almost equally good splits are identified by taking account of the predefined change fraction $c$, goodness measures, relative split sensitivity and computation cost. To minimize the computation cost of forming the splitting predicate of a node, the number of the almost equally good splits should be kept small. If there are no splits with goodness measures close to the best split, there are probably no splits almost as good as the best split. Otherwise, the change fraction $c$ is used to define the almost equally good splits.

Comparison of Tree Construction Time

We compared tree construction times by increasing the sizes of data sets 1 and 7 from 1,000 to 128,000 objects. Recall that the proposed split selection method can be run in $O(D^3 + 3 \cdot Time_{scan\ D})$. The results are shown in Figure 5.8 and Figure 5.9. The trees built by the two algorithms are shown in Figure 5.2 and Figure 5.7.
<table>
<thead>
<tr>
<th>Split Candidate</th>
<th>Goodness Measure</th>
<th>Sensitivity of $A_2$ to $A_i$</th>
<th>Sensitivity of $A_1 \land A_2 \land A_3 \land A_4$ to $A_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 = 1$</td>
<td>0.010552</td>
<td>0.2%</td>
<td>7.6%</td>
</tr>
<tr>
<td>$A_2 = 1$</td>
<td>0.011477</td>
<td>-</td>
<td>7.5%</td>
</tr>
<tr>
<td>$A_3 = 1$</td>
<td>0.010766</td>
<td>0.2%</td>
<td>7.5%</td>
</tr>
<tr>
<td>$A_4 = 1$</td>
<td>0.010595</td>
<td>0.2%</td>
<td>7.6%</td>
</tr>
<tr>
<td>$A_5 = 1$</td>
<td>0.000066</td>
<td>3.5%</td>
<td>7.5%</td>
</tr>
<tr>
<td>$A_6 = 1$</td>
<td>0.000030</td>
<td>3.6%</td>
<td>7.6%</td>
</tr>
<tr>
<td>$A_7 = 1$</td>
<td>0.000707</td>
<td>2.9%</td>
<td>7.6%</td>
</tr>
<tr>
<td>$A_8 = 1$</td>
<td>0.000012</td>
<td>3.7%</td>
<td>7.6%</td>
</tr>
<tr>
<td>$A_9 = 1$</td>
<td>0.000017</td>
<td>3.7%</td>
<td>7.6%</td>
</tr>
<tr>
<td>$A_{10} = 1$</td>
<td>0.000212</td>
<td>3.3%</td>
<td>7.6%</td>
</tr>
<tr>
<td>$A_{11} = 1$</td>
<td>0.000054</td>
<td>3.5%</td>
<td>7.5%</td>
</tr>
<tr>
<td>$A_{12} = 1$</td>
<td>0.000079</td>
<td>3.4%</td>
<td>7.6%</td>
</tr>
<tr>
<td>$A_{13} = 1$</td>
<td>0.000051</td>
<td>3.5%</td>
<td>7.6%</td>
</tr>
<tr>
<td>$A_{14} = 1$</td>
<td>0.000161</td>
<td>3.3%</td>
<td>7.5%</td>
</tr>
<tr>
<td>$A_{15} = 1$</td>
<td>0.000063</td>
<td>3.6%</td>
<td>7.5%</td>
</tr>
<tr>
<td>$A_{16} = 1$</td>
<td>0.000114</td>
<td>3.4%</td>
<td>7.6%</td>
</tr>
<tr>
<td>$A_{17} = 1$</td>
<td>0.000056</td>
<td>3.6%</td>
<td>7.6%</td>
</tr>
<tr>
<td>$A_{18} = 1$</td>
<td>0.000003</td>
<td>3.7%</td>
<td>7.5%</td>
</tr>
<tr>
<td>$A_{19} = 1$</td>
<td>0.000119</td>
<td>3.4%</td>
<td>7.5%</td>
</tr>
<tr>
<td>$A_{20} = 1$</td>
<td>0.000066</td>
<td>3.5%</td>
<td>7.6%</td>
</tr>
</tbody>
</table>

(a) CART

(b) Proposed algorithm

Figure 5.2: The classifiers grown on data set 2 with rule $A_1 \land A_2 \land A_3 \land A_4$; $A_1$, $A_2$, $A_3$, and $A_4$ are identified as almost equally good splits.
<table>
<thead>
<tr>
<th>Split Candidate</th>
<th>Goodness Measure</th>
<th>Sensitivity of $A_1$ to $A_i$</th>
<th>Sensitivity of $A_1 \lor A_2$ to $A_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 = 1$</td>
<td>0.138710</td>
<td>-</td>
<td>11.3%</td>
</tr>
<tr>
<td>$A_2 = 1$</td>
<td>0.130104</td>
<td>0.5%</td>
<td>12%</td>
</tr>
<tr>
<td>$A_3 = 1$</td>
<td>0.000011</td>
<td>12.9%</td>
<td>18.7%</td>
</tr>
<tr>
<td>$A_4 = 1$</td>
<td>0.000391</td>
<td>12.1%</td>
<td>18.3%</td>
</tr>
<tr>
<td>$A_5 = 1$</td>
<td>0.000032</td>
<td>13%</td>
<td>18.7%</td>
</tr>
</tbody>
</table>

(a) CART  

(b) Proposed algorithm

Figure 5.3: The classifiers grown on data set 3 with rule $A_1 \lor A_2$; $A_1$ and $A_2$ are identified as almost equally good splits
<table>
<thead>
<tr>
<th>Split Candidate</th>
<th>Goodness Measure</th>
<th>Sensitivity of $A_2$ to $A_i$</th>
<th>Sensitivity of $A_1 \lor A_2 \lor A_3 \lor A_4$ to $A_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 = 1$</td>
<td>0.004499</td>
<td>0.2%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_2 = 1$</td>
<td>0.005052</td>
<td>-</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_3 = 1$</td>
<td>0.004571</td>
<td>0.2%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_4 = 1$</td>
<td>0.004854</td>
<td>0.1%</td>
<td>4.9%</td>
</tr>
<tr>
<td>$A_5 = 1$</td>
<td>0.000002</td>
<td>2.4%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_6 = 1$</td>
<td>0.000131</td>
<td>2.1%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_7 = 1$</td>
<td>0.000164</td>
<td>2.1%</td>
<td>4.9%</td>
</tr>
<tr>
<td>$A_8 = 1$</td>
<td>0.000007</td>
<td>2.4%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_9 = 1$</td>
<td>0.000196</td>
<td>2.0%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{10} = 1$</td>
<td>0.000038</td>
<td>2.3%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{11} = 1$</td>
<td>0.000153</td>
<td>2.1%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{12} = 1$</td>
<td>0.000004</td>
<td>2.5%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{13} = 1$</td>
<td>0.000248</td>
<td>2.0%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{14} = 1$</td>
<td>0.000029</td>
<td>2.4%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{15} = 1$</td>
<td>0.000024</td>
<td>2.4%</td>
<td>4.9%</td>
</tr>
<tr>
<td>$A_{16} = 1$</td>
<td>0.000074</td>
<td>2.3%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{17} = 1$</td>
<td>0.000107</td>
<td>2.1%</td>
<td>4.9%</td>
</tr>
<tr>
<td>$A_{18} = 1$</td>
<td>0.000204</td>
<td>2.1%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{19} = 1$</td>
<td>0.000072</td>
<td>2.3%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{20} = 1$</td>
<td>0.000346</td>
<td>1.9%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{21} = 1$</td>
<td>0.000002</td>
<td>2.5%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{22} = 1$</td>
<td>0.000004</td>
<td>2.5%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{23} = 1$</td>
<td>0.000483</td>
<td>1.8%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{24} = 1$</td>
<td>0.000022</td>
<td>2.3%</td>
<td>4.9%</td>
</tr>
<tr>
<td>$A_{25} = 1$</td>
<td>0.000010</td>
<td>2.5%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{26} = 1$</td>
<td>0.000009</td>
<td>2.4%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{27} = 1$</td>
<td>0.000212</td>
<td>2.0%</td>
<td>4.9%</td>
</tr>
<tr>
<td>$A_{28} = 1$</td>
<td>0.000079</td>
<td>2.2%</td>
<td>4.9%</td>
</tr>
<tr>
<td>$A_{29} = 1$</td>
<td>0.000001</td>
<td>2.4%</td>
<td>4.8%</td>
</tr>
<tr>
<td>$A_{30} = 1$</td>
<td>0.000099</td>
<td>2.2%</td>
<td>4.8%</td>
</tr>
</tbody>
</table>

Figure 5.4: The classifiers grown on data set 4 with rule $A_1 \lor A_2 \lor A_3 \lor A_4$; $A_1$, $A_2$, $A_3$ and $A_4$ are identified as almost equally good splits.
<table>
<thead>
<tr>
<th>Split Candidate</th>
<th>Goodness Measure</th>
<th>Sensitivity of $A_1$ to $A_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 = 1$</td>
<td>0.261691</td>
<td>-</td>
</tr>
<tr>
<td>$A_2 = 1$</td>
<td>0.024756</td>
<td>12.4%</td>
</tr>
<tr>
<td>$A_3 = 1$</td>
<td>0.043947</td>
<td>10.5%</td>
</tr>
<tr>
<td>$A_4 = 1$</td>
<td>0.000483</td>
<td>17.2%</td>
</tr>
<tr>
<td>$A_5 = 1$</td>
<td>0.000000</td>
<td>17.7%</td>
</tr>
<tr>
<td>$A_6 = 1$</td>
<td>0.000786</td>
<td>17.1%</td>
</tr>
<tr>
<td>$A_7 = 1$</td>
<td>0.001190</td>
<td>16.3%</td>
</tr>
<tr>
<td>$A_8 = 1$</td>
<td>0.000150</td>
<td>17.2%</td>
</tr>
<tr>
<td>$A_9 = 1$</td>
<td>0.005560</td>
<td>15.3%</td>
</tr>
<tr>
<td>$A_{10} = 1$</td>
<td>0.000775</td>
<td>16.7%</td>
</tr>
</tbody>
</table>

Figure 5.5: The classifiers grown on data set 5 with rule $A_1 \lor (A_2 \land A_3)$; no splits are almost as good as $A_1$.
<table>
<thead>
<tr>
<th>Split Candidate</th>
<th>Goodness Measure</th>
<th>Sensitivity of $A_4$ to $A_i$</th>
<th>Sensitivity of $A_3 \land A_4$ to $A_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 = 1$</td>
<td>0.053753</td>
<td>2.5%</td>
<td>6.4%</td>
</tr>
<tr>
<td>$A_2 = 1$</td>
<td>0.075606</td>
<td>1.0%</td>
<td>5.3%</td>
</tr>
<tr>
<td>$A_3 = 1$</td>
<td>0.080260</td>
<td>0.7%</td>
<td>5.1%</td>
</tr>
<tr>
<td>$A_4 = 1$</td>
<td>0.090598</td>
<td>-</td>
<td>4.6%</td>
</tr>
<tr>
<td>$A_5 = 1$</td>
<td>0.000005</td>
<td>10.5%</td>
<td>12.5%</td>
</tr>
<tr>
<td>$A_6 = 1$</td>
<td>0.000178</td>
<td>10.2%</td>
<td>12.1%</td>
</tr>
<tr>
<td>$A_7 = 1$</td>
<td>0.000693</td>
<td>9.5%</td>
<td>12%</td>
</tr>
<tr>
<td>$A_8 = 1$</td>
<td>0.000040</td>
<td>10.4%</td>
<td>12.4%</td>
</tr>
<tr>
<td>$A_9 = 1$</td>
<td>0.001105</td>
<td>9.3%</td>
<td>11.9%</td>
</tr>
<tr>
<td>$A_{10} = 1$</td>
<td>0.000031</td>
<td>10.2%</td>
<td>12.6%</td>
</tr>
<tr>
<td>$A_{11} = 1$</td>
<td>0.000177</td>
<td>10.2%</td>
<td>12.2%</td>
</tr>
<tr>
<td>$A_{12} = 1$</td>
<td>0.000655</td>
<td>9.8%</td>
<td>11.8%</td>
</tr>
<tr>
<td>$A_{13} = 1$</td>
<td>0.000085</td>
<td>10.3%</td>
<td>12.3%</td>
</tr>
<tr>
<td>$A_{14} = 1$</td>
<td>0.000505</td>
<td>9.5%</td>
<td>12.3%</td>
</tr>
<tr>
<td>$A_{15} = 1$</td>
<td>0.000114</td>
<td>10.2%</td>
<td>12.4%</td>
</tr>
<tr>
<td>$A_{16} = 1$</td>
<td>0.000412</td>
<td>9.7%</td>
<td>12.3%</td>
</tr>
<tr>
<td>$A_{17} = 1$</td>
<td>0.000819</td>
<td>9.6%</td>
<td>11.8%</td>
</tr>
<tr>
<td>$A_{18} = 1$</td>
<td>0.000007</td>
<td>10.5%</td>
<td>12.3%</td>
</tr>
<tr>
<td>$A_{19} = 1$</td>
<td>0.000001</td>
<td>10.3%</td>
<td>12.4%</td>
</tr>
<tr>
<td>$A_{20} = 1$</td>
<td>0.000026</td>
<td>10.4%</td>
<td>12.4%</td>
</tr>
</tbody>
</table>

(a) CART

(b) Proposed algorithm

Figure 5.6: The classifiers grown on data set 6 with rule $(A_1 \land A_2) \lor (A_3 \land A_4)$; $A_3$ and $A_4$ are identified as almost equally good splits
(a) The tree classifier built by CART

(b) The tree classifier built by the proposed algorithm

Figure 5.7: Comparison of tree classifiers for the rule from [AIS93], \((A_1 \land (A_2 \lor A_3)) \lor (A_4 \land (A_3 \lor A_5)) \lor (A_6 \land (A_2 \lor A_7))\)
Figure 5.8: Tree construction times for data sets of various sizes for Rule 2 (Fig. 5.2)

Figure 5.9: Tree construction times for data sets of various sizes for Rule 7 (Fig. 5.7)

Figure 5.8 shows that the proposed algorithm ran slightly longer than CART. This is because the tree built by the proposed algorithm has only one node which is much smaller than the four nodes built by CART. Although it takes more time for the proposed algorithm to form a splitting predicate, the tree size compensates for the cost. Figure 5.9 confirms that the execution time of the proposed algorithm is proportional to $O(|D|^3)$. Even though the proposed algorithm has a
relatively long execution time, it took only about 20 seconds to construct a tree from a sample of 128,000 objects.

5.3.2 Experiment II: Stability Comparison

The next experiment evaluates the sensitivity of the two algorithms to small changes. A data set with noise level 20% was used to simulate real life data; that is, 20% of the sample objects are misclassified. The training sample contains 1000 objects, and each object is described by 20 attributes. The classification rule used in this experiment is “if \((A_1 = 1) \land (A_2 = 1) \land (A_3 = 1) \land (A_4 = 1)\), then class = '+', otherwise class = '−'”. We compared the tree classifiers grown on slightly different samples and the accuracy of the two tree classifiers.

<table>
<thead>
<tr>
<th>Split Candidate</th>
<th>Before Change</th>
<th>After Change</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Goodness Measure</td>
<td>Sensitivity of (A_1) to (A_i)</td>
</tr>
<tr>
<td>(A_1 = 1)</td>
<td>0.006063</td>
<td>-</td>
</tr>
<tr>
<td>(A_2 = 1)</td>
<td>0.003443</td>
<td>0.7%</td>
</tr>
<tr>
<td>(A_3 = 1)</td>
<td>0.001516</td>
<td>1.4%</td>
</tr>
<tr>
<td>(A_4 = 1)</td>
<td>0.002250</td>
<td>1.1%</td>
</tr>
<tr>
<td>(A_5 = 1)</td>
<td>0.000001</td>
<td>2.7%</td>
</tr>
<tr>
<td>(A_6 = 1)</td>
<td>0.000263</td>
<td>2.2%</td>
</tr>
<tr>
<td>(A_7 = 1)</td>
<td>0.001920</td>
<td>1.2%</td>
</tr>
<tr>
<td>(A_8 = 1)</td>
<td>0.000162</td>
<td>2.3%</td>
</tr>
<tr>
<td>(A_9 = 1)</td>
<td>0.000264</td>
<td>2.2%</td>
</tr>
<tr>
<td>(A_{10} = 1)</td>
<td>0.000202</td>
<td>2.3%</td>
</tr>
<tr>
<td>(A_{11} = 1)</td>
<td>0.000880</td>
<td>1.7%</td>
</tr>
<tr>
<td>(A_{12} = 1)</td>
<td>0.000515</td>
<td>2.0%</td>
</tr>
<tr>
<td>(A_{13} = 1)</td>
<td>0.000314</td>
<td>2.2%</td>
</tr>
<tr>
<td>(A_{14} = 1)</td>
<td>0.000000</td>
<td>2.7%</td>
</tr>
<tr>
<td>(A_{15} = 1)</td>
<td>0.000242</td>
<td>2.3%</td>
</tr>
<tr>
<td>(A_{16} = 1)</td>
<td>0.000219</td>
<td>2.3%</td>
</tr>
<tr>
<td>(A_{17} = 1)</td>
<td>0.000000</td>
<td>2.8%</td>
</tr>
<tr>
<td>(A_{18} = 1)</td>
<td>0.000013</td>
<td>2.6%</td>
</tr>
<tr>
<td>(A_{19} = 1)</td>
<td>0.000073</td>
<td>2.5%</td>
</tr>
<tr>
<td>(A_{20} = 1)</td>
<td>0.000145</td>
<td>2.4%</td>
</tr>
</tbody>
</table>

Table 5.2: The information about the goodness measures of the split candidates, and stability of the splitting predicates at the root node before and after a change made to 1.5% of the sample
Table 5.2 shows the goodness measures of the split candidates at the root node before and after a change is made to 1.5% of the training sample. Since the change has caused the split “$A_7 = 1$” to become the best split in the changed sample, “$A_7 = 1$” is selected by CART as the split of the root node. The tree classifiers constructed by CART before and after the change are shown in Figure 5.10 (a) and (b). The proposed algorithm, on the other hand, is able to generate the same rule after the change because of its consideration of almost equally good splits. If “$A_1 = 1$” and “$A_2 = 1$” are identified as splits almost as good as “$A_7 = 1$”, the evaluation of the combinations of the three splits filters out “$A_7 = 1$”, which is either an insignificant pattern change or a noise, and selects “$A_1 = 1 \land A_2 = 1$” as the split of the root node. The tree classifiers built by the proposed algorithm before and after change are shown in Figure 5.10 (a’) and (b’).

We continued to test the stability of the two algorithms by adding another 2.3% change to the
changed training sample. The new tree classifier built by CART is shown in Figure 5.10(c). Note that even though the rule still holds for almost 78% of objects in the sample, the classifier built by CART fails to discover it and only two of the factors in the rule are discovered. On the other hand, the proposed algorithm is insensitive to noise because the splitting predicate of a node is formed by almost equally good splits. At the root node, “$A_2 = 1$”, “$A_4 = 1$”, and “$A_7 = 1$” are almost equally good. The combination of “$A_2 = 1$” and “$A_4 = 1$” filters out “$A_7 = 1$”. Because the right splitting predicate is used, the remaining part of the rule is discovered at the lower level. The tree classifier built by the proposed algorithm is shown in Figure 5.10(c').

Adding another 2.5% change to the sample, the tree classifiers built by the two algorithms are shown in Figure 5.10 (d) and (d'). Note that there is no information about $A_1$, $A_2$, $A_3$, and $A_4$ in the classifier built by CART even though there is still about 76% of the sample classified by the rule based on those factors. The identified almost equally good splits at the root node are “$A_2 = 1$”, “$A_7 = 1$”, and “$A_{19} = 1$”. The evaluation of the three splits chose “$A_2 = 1 \land A_7 = 1$” as the splitting predicate of the root. Because “$A_2 = 1$” is part of the splitting predicate of the root node, the splits based on $A_1$, $A_3$, and $A_4$ are discovered at lower levels. Note that “$A_7 = 1$” is included in the splitting predicate of the root node because the accumulated changes which were in favor of $A_7$ become significant enough to indicate a pattern change. The accuracy comparison of the two algorithms is shown in Figure 5.11.

The results show that the proposed algorithm is not only more stable, but also more accurate and more tolerant to noise. Unlike the series of abrupt changes in the tree structures built by CART, the proposed algorithm holds the tree structure until the changes are significant enough to become a new pattern. At the beginning, the trees built by the two algorithms are able to discover the rules even when the data is noisy. As the noise level increases, CART starts to pick the noise to partition data. Once a different split is selected, the subtree grown on that node may become very different. On the contrary, the proposed algorithm is able to eliminate the noise and uses the right splitting predicate to partition the data. Therefore, the proposed algorithm is more stable and accurate.
5.3.3 Experiment III: Information Loss in Case of Almost Equally Good Splits

In this experiment we examine the possible information loss in case of almost equally good splits. The training sample used in this experiment contains 1000 objects and each record has 10 attributes. Assume that the splits “\(A_2 = 1\)” and “\(A_{10} = 1\)” are strongly correlated. An example of strongly correlated splits is “student” and “age under 22”, because most students are under 22.

The goodness measures and the sensitivity of the split candidates at the root and its left child node, and the tree classifiers generated by the two algorithms are shown in Figure 5.12. Although “\(A_2 = 1\)” and “\(A_{10} = 1\)” are strongly correlated, CART only selects “\(A_{10} = 1\)” to partition the data. Hence, the information about “\(A_2 = 1\)” is lost. If both “\(A_2 = 1\)” and “\(A_{10} = 1\)” are important factors, then CART misses an important factor. If only one of the splits is important and it
<table>
<thead>
<tr>
<th>Split Candidate</th>
<th>Goodness Measure</th>
<th>Sensitivity of $A_1$ to $A_i$</th>
<th>Goodness Measure</th>
<th>Sensitivity of $A_{10}$ to $A_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 = 1$</td>
<td>0.117564</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$A_2 = 1$</td>
<td>0.094238</td>
<td>1.3%</td>
<td>0.472125</td>
<td>0.4%</td>
</tr>
<tr>
<td>$A_3 = 1$</td>
<td>0.002337</td>
<td>10.3%</td>
<td>0.003288</td>
<td>21.3%</td>
</tr>
<tr>
<td>$A_4 = 1$</td>
<td>0.000021</td>
<td>11.8%</td>
<td>0.000587</td>
<td>23.3%</td>
</tr>
<tr>
<td>$A_5 = 1$</td>
<td>0.000124</td>
<td>11.4%</td>
<td>0.000517</td>
<td>23.5%</td>
</tr>
<tr>
<td>$A_6 = 1$</td>
<td>0.000170</td>
<td>11.5%</td>
<td>0.000125</td>
<td>23.7%</td>
</tr>
<tr>
<td>$A_7 = 1$</td>
<td>0.000827</td>
<td>10.9%</td>
<td>0.002872</td>
<td>21.9%</td>
</tr>
<tr>
<td>$A_8 = 1$</td>
<td>0.000001</td>
<td>12.1%</td>
<td>0.002921</td>
<td>22.3%</td>
</tr>
<tr>
<td>$A_9 = 1$</td>
<td>0.000050</td>
<td>11.9%</td>
<td>0.000571</td>
<td>23.5%</td>
</tr>
<tr>
<td>$A_{10} = 1$</td>
<td>0.094277</td>
<td>1.3%</td>
<td>0.483696</td>
<td>-</td>
</tr>
</tbody>
</table>

(a) CART

(b) Proposed algorithm

Figure 5.12: The split "$A_2 = 1$", which is almost as good as "$A_{10} = 1$", is not discovered by CART, but is discovered by the proposed algorithm.

happens to be that "$A_2 = 1$" is the important one, CART misses the important factor again. The proposed algorithm, on the other hand, uses both splits to partition the data, and is therefore able to discover factors that are not discovered by CART.

5.4 Summary

In this chapter, we compared the proposed algorithm with the well known decision tree classification algorithm CART. The empirical results illustrate that the proposed split selection method, which forms the splitting predicate of a node based on almost equally good splits, tends to be
more expressive, comprehensible, stable, and results in more concise trees. Moreover, the proposed algorithm is more insensitive to noise because the combination of good splits filters out the noise. Because the combination of almost equally good splits is more expressive, it alleviates some of the inefficient representation problem for disjunctive rules. The proposed split selection algorithm also reduces the possible information loss in case of strongly correlated splits.

The relative sensitivity between the best split candidate and another split candidate shows the relationship between the two splits, and can be used as a metric to evaluate the stability of splitting predicates.

In this experiment, we use the change fraction $c$ to identify almost equally good splits. In general, almost equally good splits can be identified for use in our algorithm by taking into account the goodness measure, the relative sensitivity, the change fraction $c$, and the computation cost. Although the proposed algorithm requires more CPU time, it produces tree classifiers of better quality.
Chapter 6

Conclusions and Future Work

6.1 Conclusions

The instability problem of decision tree classifiers is that small changes in the input training samples may produce dramatically large changes in the output tree classifiers. Intuitively, the instability of decision tree classifiers is caused by split selection among a set of almost equally good split candidates at a node. The reason for the selection of a split sensitive to small changes is that a small change may cause a split candidate that was slightly inferior to the best split candidate to become superior to it. Once a new split is selected, the subtree evolved from that node may be very different.

To study the cause of the instability problem, we investigated the impact of changes in training samples on tree classifiers. Since the split candidates are evaluated by the goodness measures, we analyzed the effects of a change in the data associated with a node to the goodness measures of the split candidates at the node. Theorem 1 proves the relationship between a change in a data set and the resulting split change. Based on Theorem 1, the sensitivity of a split and the fuzzy term "almost equally good splits" can be defined.

The sensitivity of the best split candidate \( t_0 \) to another split candidate \( t \) is defined as the smallest change that causes \( t_0 \) to be replaced by \( t \). The sensitivity of the best split candidate is defined as the smallest change that causes it to be replaced. A split candidate is defined to be almost as good as the best split candidate if the relative sensitivity is less than or equal to a predefined change fraction \( c \).
The relative sensitivity of $t_0$ to $t$ exhibits the relationship between the splits. A split candidate to which the best split candidate is highly sensitive is considered almost as good as the best split. Unlike the goodness measures which evaluate split candidates by real numbers, the relative sensitivity, which evaluates splits by the smallest change fractions that may cause split changes, is more meaningful and intuitive for users to understand the relationship between splits and therefore the possible changes in tree structures. The sensitivity of a tree is not only an indicator of the confidence level of the classification rules, but also a good indicator of the effective lifetime of the rules.

Theorem 2, the instability theorem, provides the cause of the instability problem. Theorem 2 shows that the selection of the best split candidate from almost equally good splits results in a node that is sensitive to small changes. Theorem 1 and Theorem 2 assume changes are made to class labels. Theorem 3 relaxes this restriction by allowing changes to be insertion, deletion, or update of the objects in training samples. The instability theorem for the general case is the same as Theorem 2 except that the function $\varepsilon$ is different.

Based on the theorems, improvements are proposed to lessen the instability problem of decision tree classifiers. We presented algorithms to compute the sensitivity of splits, identify almost equally good splits, and form splitting predicates. The proposed split selection algorithm constructs the splitting predicate of a node by evaluating the combinations of the almost equally good split candidates at the node, and then using the best splitting predicate to partition the data. The proposed split selection algorithm can be adapted to decision classifiers using the gini splitting criterion to lessen the instability problem.

The empirical results of Chapter 5 illustrate that in the presence of almost equally good splits at a node, the splitting predicate formed by the almost equally good splits is more stable than the best split candidate. In addition, the splitting predicate formed by almost equally good splits makes the splits with the same degree of importance appear at the same node, and is more expressive. Hence, the tree built by the proposed algorithm tends to be more intuitive and more concise. The splitting
predicates based on almost equally good splits also relieve some of the inefficient representation problems of disjunctive rules. Moreover, the proposed split selection algorithm is more tolerant to noise, because the evaluation of the combinations of almost equally good splits can filter out noises. Therefore, the tree classifiers built by our algorithm can obtain higher accuracy. Besides, the consideration of splits of the same degree of importance in forming the splitting predicate may prevent loss of information about other equally good splits. Although the proposed classification tree algorithm requires more computation time to construct stable splits, the resulting classification trees tend to be more expressive, comprehensible, stable, concise, informative, and noise-tolerant.

6.2 Summary of Contributions

The instability of tree classification has recently drawn the attention of researchers [Brei94, Brei96, Salf00, Rule00]. However, none of the work emphasizes the tree structure change, which is important to provide decision makers stable, comprehensible, and accurate tree classifiers.

In this thesis, the quality of tree classifiers is investigated from the point of view of sensitivity to small data changes. Theorem 1 proves the relationship between a change in a data set and the resulting split change. Based on this theorem, the relative sensitivity of two splits can be computed. The relative sensitivity not only shows the relationship between the split candidates of a node, but can also be used as a metric to evaluate the stability of tree classifiers. The sensitivity of tree classifiers are indicators of both the confidence level in the rules and the effective lifetime of the rules.

The instability theorem (Theorem 2) provides the cause of the instability of decision tree classification algorithms. An algorithm to form splitting predicates is derived based on the theorems to lessen the instability problem. The experimental results indicate that our enhanced decision tree classification algorithm performs better than CART.
6.3 Future Work

Many different splitting criteria are used to select the best split at each node. The function $\varepsilon$ was derived based on the gini index. For a different splitting criterion, Theorem 1 still holds except that a different function $\varepsilon$ needs to be defined. Therefore, it would be interesting to analyze the relationship between a change in a data set and the resulting split change for a different splitting criterion. By following the steps in the proof of Theorem 1, the function $\varepsilon$ for a different criterion can be derived.

In this thesis, almost equally good splits are combined either conjunctively or disjunctively. This approach is appropriate only for categorical attributes. Therefore, algorithms need to be developed to combine numeric splits. [FMM96] presented algorithms that dealt with the inefficient representation problem of decision tree classifiers if numeric attributes are strongly correlated. Since strongly correlated numeric splits must be identified together as almost equally good splits, it would be worthwhile to test whether the splitting predicates formed by almost equally good splits can improve the inefficient representation problems for strongly correlated numeric attributes.

The sensitivity of a tree is a good indicator of the effective lifetime of classification trees. In a dynamic environment where data changes, effective classification rules are crucial to make correct decisions. To ensure the effectiveness of the rules, changes to the database need to be monitored. Since not any change greater than the computed sensitivity will cause a split change, the algorithm should be able to detect a structure change only when it actually happens. When a split change occurs at a subtree, instead of growing a new tree we can utilize the information about changes to reconstruct the subtree.
References


82
[CHY96] Ming-Syan Chen, Jiawei Han, and Philip Yu. Data mining: an overview from database perspective. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 6, 1996.


Vita

Ruey-Hsia Li was born in Hsinchu, Taiwan. She received her B.S. degree in Applied Mathematics in 1990 from Chung Yang Christian University, Taiwan, and her M.S. degree in Applied Mathematics in 1992 from University of Illinois at Urbana-Champaign. After that, she continued her Ph.D. study in Computer Science.

During the period of her graduate study at University of Illinois at Urbana-Champaign, she had been working at NCSA as a research assistant where she was responsible for the project HDFNOW sponsored by NASA. In addition, she wrote parallel programs, and maintained and developed HDF software. Her Ph.D. research is in data mining, especially the stability of decision tree classification algorithms.