I, Arjun Bakshi, hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Science.

It is entitled:
Methodology For Generating High-Confidence Cost-Sensitive Rules For Classification

Student’s name: Arjun Bakshi

This work and its defense approved by:

Committee chair: Raj Bhatnagar, Ph.D.

Committee member: Anil Jegga, D.V.M., M.Res.

Committee member: Ali Minai, Ph.D.
Methodology For Generating High-Confidence Cost-Sensitive Rules For Classification

A thesis submitted to the faculty of
University of Cincinnati
in partial fulfillment of the requirements for the degree of
Master of Science
Department of Computer Science
University of Cincinnati
by
Arjun Bakshi
June 28, 2013
Bachelor of Engineering Information Technology
University of Mumbai
May 2010
Committee Chair: Dr. Raj Bhatnagar
Abstract

Rule based classifiers are often used to make crucial decision in domains like medicine and business intelligence, where there is a need to build insightful models that are quick to train, perform accurate classification, and take the costs of mistakes into account while making or helping with predictions. The existing techniques that address these requirements suffer from some disadvantages that cause them to generate overly complicated rule sets that sometimes do not perform well on new data, or do not take differing misclassification costs into account. The work proposed here aims to build a rule based classifier that extracts rules that have higher support and confidence than existing techniques as well as a classification model that minimizes the cost incurred from misclassifications by making cost sensitive decisions and flagging instances that are likely to be misclassified.
Acknowledgement

I would like to thank my advisor, Dr. Bhatnagar, for his guidance and support which has helped improve the quality of my work and helped shape and enhance my ability to conduct quality research. I would also like to thank my committee members, Dr. Ali Minai and Dr. Anil Jegga for their guidance. Finally, I am thankful to my family for their support, motivation and love.
Contents

1 Introduction ............................................................... 1
   1.1 Motivation ............................................................ 2
       1.1.1 Greedy selection and single coverage in Decision Trees .......... 2
       1.1.2 Computational cost of optimal rule set ............................ 3
       1.1.3 Cost sensitivity .................................................. 3
   1.2 Our Approach ........................................................ 4

2 Background ............................................................... 5
   2.1 Classification ......................................................... 5
   2.2 Decision Trees, Multiple Coverage and Guided Breadth First Search ... 6
   2.3 Cost Sensitive Classification ........................................ 9
       2.3.1 Cost sensitive datasets ......................................... 9
       2.3.2 Issues with using Cost-blind classifiers ....................... 11

3 Related Work ............................................................ 13
   3.1 Single Decision Tree based approaches ............................... 13
   3.2 Ensemble classifiers using Decision trees .......................... 14
   3.3 Association rule based approaches .................................. 15
       3.3.1 CBA ............................................................... 16
       3.3.2 GARC ............................................................. 17
3.3.3 HARMONY ......................................................... 17
3.4 Cost sensitive Rule based classifiers ............................................ 18
3.5 Novelty of our approach and results ............................................. 20

4 Approach Adopted for Rule Induction ............................................ 22
  4.1 Preliminaries ................................................................. 22
    4.1.1 Defining a rule ......................................................... 23
    4.1.2 Support and coverage of a rule ......................................... 23
    4.1.3 Quality of a rule ....................................................... 23
    4.1.4 The apriori algorithm ................................................ 25
  4.2 Approach ................................................................. 26
  4.3 Classifying a new instance ................................................... 32
  4.4 Choosing entropy cutoffs and a \( \text{minSupp} \) .................................. 32

5 Experimental Setup and Results ................................................... 34
  5.1 Experimental Setup ............................................................ 34
  5.2 Experiments and Results ...................................................... 35
    5.2.1 Analysis on small datasets ........................................... 35
    5.2.2 Analysis on large datasets ........................................... 37
    5.2.3 Analysis with non-uniform cost matrices ............................. 39
    5.2.4 Reduction in misclassification costs by leaving instances undecided 43
    5.2.5 Effect on Average rule length, precision, and support .............. 46
  5.3 Comparision with KNN and SVM .............................................. 47

6 Conclusion ................................................................. 49

7 Appendix ................................................................. 51
  7.1 Results for remaining small datasets, with \( \text{minSupp} \) set to 10 .................. 52
  7.2 Results for remaining small datasets, with \( \text{minSupp} \) set to 20 .................. 53
7.3 Results for remaining large datasets, with \textit{minSupp} set to 50
List of Algorithms

1  Making base rules. ................................................................. 28
2  Rule Generation ................................................................. 30
# List of Figures

2.1 A sample, 2 dimensional dataset and a decision boundary. ......................... 6  
2.2 Overlapping rules. ................................................................. 8  
2.3 Non-overlapping rules. ............................................................. 8  
2.4 Sample 2 class dataset with 4 distinct patterns. .................................... 12  
3.1 MetaCost original dataset .......................................................... 20  
3.2 MetaCost relabeled dataset ......................................................... 21  
4.1 Entropy vs Support across classes. ............................................... 25  
4.2 Prefix tree search for test dataset. ............................................... 28  
5.1 Cost-sensitive performance: Pima ............................................... 41  
5.2 Cost-sensitive performance: Hepatitis ........................................... 42  
5.3 Cost-sensitive performance: Breast Cancer ....................................... 42  
5.4 Cost-sensitive performance: Iris .................................................. 43  
5.5 Cost-sensitive performance: Wine ............................................... 43  
5.6 Performance with undecided instances: Pima .................................... 44  
5.7 Performance with undecided instances: Glass .................................... 45
List of Tables

2.1 Non-symmetric cost matrix. ........................................... 10
2.2 Non-symmetric and non-uniform cost matrix ......................... 10
2.3 Sample misclassification costs. .................................. 11
3.1 Sample market basket database. ................................... 15
3.2 A sample non-uniform, non-symmetric cost matrix. ............... 19
4.1 Original sample real valued dataset ................................. 27
4.2 Discretized sample dataset ......................................... 27
5.1 Details of 12 UCI datasets ......................................... 35
5.2 Accuracy on small datasets - 1 .................................... 37
5.3 Accuracy on small datasets - 2 .................................... 38
5.4 Performance on large datasets .................................... 39
5.5 Effects on leaving instances undecided. ............................ 45
5.6 Comparison of rule length, support and precision. ............... 46
5.7 Performance on large data sets - continued. ....................... 48
7.1 Accuracy on small datasets - 1 continued. ........................... 52
7.2 Accuracy on small datasets - 2 continued. ........................... 53
7.3 Performance on large data sets - continued. ....................... 54
Chapter 1

Introduction

Classification of data is one of the fundamental concepts in data mining. Its objective is to correctly model the underlying class structure of the system, and accurately predict classes of future instances based on patterns learnt by training on past data. Classifiers are used in medical diagnostics, business intelligence, computer vision and many other fields. Over the past few decades a lot of work has been done to make more robust classification models. Support Vector Machines, Neural Networks, decision tree algorithms, and many other approaches have been investigated and tuned to get very accurate models.

Classifiers can be broadly categorized in a few different ways. One classification is based on whether the models generated are rule based or not. A rule based model is one that extracts conditions across attributes from the dataset that are unique to a class, and therefore predictive of it. The rules that are extracted are in human-readable form and are easy interpret. Example of non-rule based classifiers, like Support Vector Machines, Neural Networks and Nearest Neighbor Classifiers.

Another way of categorizing classifiers is based on whether they take various costs associated with a dataset into account while building classification models. These costs may be incurred either for conducting at attribute-value test or from the loss suffered due to a wrong classification.

Decision trees are popular rule based classifiers that are frequently used as base algorithms for
ensemble classifiers, meta algorithms, as well as stand alone classifiers. The rules they generate are in the form of conjunctions of conditions across attributes and their values. Despite their popularity, speed of operation and high accuracy, decision tree algorithms have some drawbacks that are a result of design choices made during the decision tree induction. In particular, they work in a greedy manner, choosing split points based on immediate gain, and the algorithm irreversibly partitions datasets as it progresses.

This work seeks to improve the classification rule induction process in such product form rule based “white-box” classifiers by making it more efficient and also by reducing the expected cost of misclassifications by the generated rule set.

1.1 Motivation

1.1.1 Greedy selection and single coverage in Decision Trees

Using a greedy approach to select the split point for an iteration and partitioning the dataset into exclusive subsets are reasonable choices. Selecting test attributes from among all the attributes in a greedy manner makes the algorithm run fast. However, it increases the chances of ending up with a sub-optimal model, especially when the number of attributes is large, and there are a significant number of almost equally good candidates for split points. The results are sub-optimal from the perspective of rule length and overall classification accuracy.

Repeatedly partitioning the dataset helps enforce a single coverage constraint on the rows. That is, once a row has been covered by one rule, it is removed from the set of rows participating in the generation of other, newer rules in future iterations. This constraint prevents the algorithm from generating redundant rules. However, it can also cause the algorithm to overlook significant predictive patterns split across different partitions performed in preceding iterations, and instead generate rules that may not be statistically significant. Both of these issues become more prominent as the size of the dataset increases in terms of features and/or instances.
1.1.2 Computational cost of optimal rule set

One way of overcoming the problems caused by greedy selection and single coverage is to perform an exhaustive search in the feature space of the data. However, doing so would take a prohibitively long time. This is because the number of possible split points and their combinations grows exponentially with the number of features. It becomes necessary to use heuristics to prune the search space effectively to maintain or improve the run-time and accuracy of the classifier.

1.1.3 Cost sensitivity

Until recently, most of the work done assumed that the costs of misclassifying instances are symmetric and uniform for all classes. That is to say that all mistakes carry the same performance penalty. For example, given a three-class dataset, the penalty of misclassifying an instance of class A as that of class B is the same as the penalty of misclassifying an instance of class B as that of class A, or that of class C as class A. This was initially assumed to be the case in general, and the case of non-symmetric, non-uniform misclassification cost distribution was treated as a special case. However, such datasets, called “Cost Sensitive Datasets”, are quite common in medical and anomaly detection applications.

Researchers have worked on the problem of cost sensitive discriminant analysis, especially in the context of banking and economic decision [16] [7]. However, the idea for building rule based classifiers that take costs into account recently started gathering interest.

When classifiers that do not take such asymmetric costs into account are used on cost-sensitive data, the result is a model that maximizes accuracy but does not minimize the cost incurred by misclassifications. The work presented here aims to address this issue also by incorporating the cost matrix into the process that guides the search for classification rules and their evaluation. The model built this way ensures that the decisions made minimize cost incurred from misclassification errors.
1.2 Our Approach

The work presented here addresses these issues in the following manner:

• Instead of greedily selecting database split points, all splits are explored in a breadth first traversal manner. Heuristics are used to prune unpromising sections of the search space.

• The single coverage constraint is dropped, which allows rows to be reused for generating new, more significant rules.

• The approach includes the misclassification costs during rule generation to ensure that the model minimizes the cost incurred from misclassification.

Additionally, in some situations it is better to leave a test instance undecided rather than risking misclassifying it. However, most classifiers always assign a class label to a test instance. By using a classifier that leaves instances that are likely to be misclassified as undecided, the cost of misclassification can be further reduced. In some application domains when instances are left unclassified it may be an indication that more information is needed to help decide the class. More tests may be ordered, old tests may be repeated, or other sources of information may be tapped for help.
Chapter 2

Background

This chapter reviews concepts like classification, decision trees, cost sensitive decision situations and data sets, and problems encountered in building classifiers for such datasets.

2.1 Classification

Classification is a frequent data mining activity. It is the task of predicting the category or class to which a new data point belongs. Predictions are made based on rules, patterns or models built from training data where class memberships of data points are known. A data point is an n-dimensional vector, where the value at each index in the vector corresponds to the value of a feature of the data point. By learning how these different feature values vary across different classes, it is possible to predict the class of new data points.

Classifiers are used across many domains. They are used in medical diagnostics, speech recognition, computer vision, and anomaly detection to name a few. Depending on the application and dataset, different types of classifiers may be preferred. If the task is to simply predict the label of a new instance, “black box” classifiers like Support Vector Machines or Neural Networks may be used. However, if the aim of the classification is to understand why an instance is being classified as belonging to a class, or to generate human readable rules to get insight into the structure of the data set, “white box” or rule based classifiers like decision trees should be used.
2.2 Decision Trees, Multiple Coverage and Guided Breadth First Search

Decision Tree algorithms are a popular category of algorithms since they are easy to implement, can deal with missing values, produce intuitive classification models and have good accuracy on most datasets. However, most decision tree algorithms have the following drawbacks:

- **Greedy selection of split points**: The decision tree algorithms based on ID3 and C4.5 and similar methodologies iteratively split a dataset into subsets that predominantly contain examples of only one class \[18\] \[17\]. Greedy splits of datasets with large number of dimensions and instances can easily lead to sub-optimal results.

- **Single coverage**: A training instance is used to generate only one classification rule. A lot more information can be extracted if instances would be shared and reused for making multiple rules.

- **Iteratively splitting the dataset**: All decision tree algorithms work by iteratively splitting the dataset, and subsequent subsets on a split point that is chosen in a greedy way. This can
cause prominent patterns to get split across different subsets and therefore prevent them from becoming rules.

The arguments for and against these restrictions can be made. Finding the globally optimal split point is a time consuming problem and so an approximation using a greedy approach may be well worth the gain in running time. However, this approach can also lead to globally sub-optimal and unnecessarily long rules that could have been avoided by simply choosing more split points at each iteration.

The single coverage constraint and exclusive splits of subset ensure that the algorithm does not spend time generating redundant rules for instances that are already covered by existing rules that have high confidence. By successively splitting the dataset on a split point, the algorithm aims to end up with leaf nodes that have a more homogeneous class distribution. By marking an instance as covered by a rule, the algorithm can ignore it from then onwards, and work towards finding rules that cover the remaining instances. It is a way of ensuring that the maximum number of instances are covered by the minimum number of rules.

However, this design choice can lead to 2 unwanted effects. Firstly, certain rules that are more generalizable to an unseen test set may be overlooked as support for those rules may be split across different subsets. Secondly and consequently, the rules that are generated from successively smaller sets tend to be over-fitted to the training set. This can cause the performance of the decision tree on test sets to be lower than expected.

Figure 2.2 and 2.3 show the kind of rules a decision tree algorithm would make, and the kind of rules a classifier that allows overlapping rule generation may generate respectively. The rules in figure 2.2 are applicable to rows exclusively and their support is generally lower than the set of rules in figure 2.3. Rules that have lower support or that are too specialized may lose their statistical significance and may not be widely applicable to new instances.
Figure 2.2: Overlapping rules.

Figure 2.3: Non-overlapping rules.
2.3 Cost Sensitive Classification

Traditionally, classifiers are built with the objective of maximizing the classification accuracy across all classes. However, in many applications it is more important to correctly identify instances of one class than the others. Additionally, in multi-class classification problems it may be more acceptable to mis-classify an instance of class $A$ as that of $B$, than to mis-classify it as an instance of class $C$. Simply said, different types of misclassification errors have different penalties for the performance of the model or application. An application may accept low classification accuracy for some classes, but needs very high accuracy for others. The objective then becomes to minimize the cost performance of the classifier rather than to maximize the overall classification accuracy. For example, when trying to identify legitimate or fraudulent transactions in a credit card transaction database, it is more important that more instances belonging to the “fraud” class are correctly classified compared to that of the “legitimate” class. This is because the cost incurred from classifying a fraudulent transaction as a legitimate one and not investigating it is higher than investigating a legitimate transaction.

Classifiers that aim to maximize classification accuracy across all classes may indirectly reduce costs incurred by misclassifying instances by simply making sure that the predicted class labels are correct. However such models tend to be sub-optimal, and there is a need to address this issue directly.

Classifiers that do not take the different misclassification costs into account are sometimes called “Cost-blind classifiers”, whereas those that do are called “Cost sensitive classifiers”. This terminology shall be used here onwards. The following subsection reviews the issues in applying cost-blind classifiers to cost sensitive datasets.

2.3.1 Cost sensitive datasets

A dataset is called a cost sensitive dataset if there are non-uniform costs associated with misclassifying instances of each class. The terms “Cost”, and “Weight” are frequently used to describe the importance of identifying an instance of that class correctly. The higher the importance of having
high sensitivity/ specificity/ accuracy for a class, higher the cost of misclassifying instances of that class. The cost of misclassifying an instance of a class may or may not depend on which other class it was mistakenly classified as. Therefore, the cost of misclassifying an instance of class $i$ as that of class $j$ may be $C_i$ if the misclassification cost is independent of what it gets misclassified as, or it may be $C_{ij}$ if it is dependent on what $j$ is.

The preference ordering or relative costs of misclassification of classes are usually represented using a cost matrix. The following are examples of cost matrices:

<table>
<thead>
<tr>
<th>Actual-Predicted</th>
<th>Class A</th>
<th>Class B</th>
<th>Class C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Class B</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Class C</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.1: A cost matrix where the cost of misclassifying an instance of a class does not depend on what it gets misclassified as.

In table 2.1, we can see that there is no cost incurred when an instance is classified correctly. The cost of misclassifying an instance of class A is 5 regardless of which class it is misclassified as. The same goes for classes B and C.

<table>
<thead>
<tr>
<th>Actual-Predicted</th>
<th>Class A</th>
<th>Class B</th>
<th>Class C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>0</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>Class B</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Class C</td>
<td>6</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.2: A cost matrix where the cost of misclassifying an instance of a class depends on what it gets misclassified as.

In table 2.2, the cost of misclassifying an instance of class A depends on what class the instance is misclassified as. If an instance of class A is misclassified as that of B the cost incurred is 5, but if it is misclassified as C, the cost incurred is 2. The costs of misclassifying instances of class B and C vary similarly.
2.3.2 Issues with using Cost-blind classifiers

Classifiers that aim to maximize the overall classification accuracy, and are not able to target higher accuracy specially for the important class(es). For example, a doctor could be more interested in a classifier that has higher sensitivity and specificity for the diseased class than for the healthy class of samples.

The following example helps illustrate the problems encountered when applying cost-blind classifiers to cost sensitive data.

Let \( D \) be a dataset with two classes, \( A \) and \( B \), with misclassification costs of 0.5 and 5 respectively. Classifying an instance of type \( A \) as that of \( B \) has a penalty of 0.5, where as the cost of classifying an instance of class \( B \) as that of \( A \) has penalty of 5.

<table>
<thead>
<tr>
<th>Actual-Predicted</th>
<th>Class A</th>
<th>Class B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Class B</td>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.3: Misclassification costs for classes in dataset \( D \)

In figure 2.4 we can see four patterns that exist across all columns. P1 is unique to class \( A \), whereas P3 and P4 are unique in class \( B \). P2 however, covers one instance in class \( A \) and nine in class \( B \).

If a cost-blind classifier were to choose a class label for a test instance with pattern P2, it would choose \( B \) and would be correct 90\% of the time. This may be acceptable in terms of classification accuracy, but not when differing costs of misclassification are taken into account. The cost of this rule can be defined as:

\[
\text{misclassification cost} = \text{Number of instances misclassified} \times \text{cost of misclassifications}
\]

Say such a rule were to be used to classify 100 new test instances. If it were to classify them all as Class \( B \) and be correct for 90\% of the time, then it would incur a cost of 50 units for the 10
instances misclassified. Similarly, if it were to classify them as class A, it would then incur a cost of 45 units from misclassification errors.

This is an example of how a rule with relatively high accuracy becomes detrimental for the quality of the classification model when non-uniform costs are taken into account. In the given scenario the classification algorithm should reject this rule and try to find rules which have lower expected cost of misclassification. The calculation and assessment of the quality or expected cost of a rule becomes more complex when more than two classes are involved. Additionally, it may be the case that an algorithm cannot find any rule to classify an instance without expecting high misclassification cost. In such cases, it may be better to withhold any prediction of the class for that instance.
Chapter 3

Related Work

Decision tree algorithms, and classifiers based on association rules are two examples of rule based classifiers. The following subsections review related work done in cost-blind and cost-sensitive rule-based classifiers.

3.1 Single Decision Tree based approaches

Breiman et. al. were one of the first to work on decision tree learning algorithms[2]. Classification and Regression Trees, or CART, uses a binary tree like approach to split the dataset into smaller sets that have more homogeneous class distributions. The term CART can be used to refer to all algorithms that are modeled around the decision tree learning algorithm. Ross Quinlan proposed the Iterative Dicotomiser-3 algorithm, or ID3, which used entropy to choose the split points while making decision trees [17]. The C4.5 algorithm, also designed by Ross Quinlan, uses information gain to choose the best split point for an iteration. C4.5 improves on ID3 by being able to work with continuous as well as discrete valued data, and being able to handle missing values in data. In order to prevent over-fitting of the classifier to training data, C4.5 carries out post-hoc pruning of the decision trees by merging sibling nodes that do not improve the performance of the classifier by much. A newer version of this algorithm, C5, allows boosting and weighing different training instances differently. These weights represent how important an instance is to the classification
model, and how much the instance can influence rule generation.

The algorithms mentioned above are algorithms that generate single decision trees, and choose only one split point per iteration in a greedy manner. This causes the algorithms to have the disadvantages discussed in the previous chapter. Namely, overlooking significant rules, and building sub-optimal trees that do not generalize well to new instances. Ensemble techniques, discussed in the next section, have been proposed to overcome these limitations to some extent.

3.2 Ensemble classifiers using Decision trees

Random Forests, Bagging and Boosting are frequently used ensemble methods or meta-algorithms that use decision trees to build strong classifiers with high classification accuracy [3] [1] [11].

The bagging meta-algorithm builds multiple decision trees for the same dataset by re-sampling the training set. Given a training dataset $D$ of size $N$, the bagging algorithm will make $m$ smaller datasets of size $n$ each, where $n < N$. Once these $m$ smaller datasets have been generated, a different decision tree is built for each one. This increases the total number of rules extracted and allows different decision trees to focus on different classes and patterns in the training data. When a new test instance needs to be classified, each of these decision trees is probed for a prediction and the most frequently predicted class label is chosen as the final prediction of the ensemble classifier. The disadvantage of this method is that it may end up with redundant or too many rules.

Boosting is a meta algorithm that aims to build strong classification models using a set of weak classifiers. The term “weak classifier” is used for classification models that perform slightly better than random guesses for test instances, whereas a “strong classifier” is one that has very high prediction accuracy. Boosting is carried out by repeatedly learning weak classifiers on data points that were previously misclassified by the classifier. Instances start out being weighted equally, but as the algorithm progresses, instance weights are increased if they were misclassified and decreased if they were correctly classified. This is done to make sure that the next classification model gives more importance to those instances that were misclassified in the previous iteration. Once multiple models for correctly classifying all or most instances have been build, classification is carried out
by performing a weighted voting on class labels predicted by different models.

Random Forests, similar to Bagging, re-sample the original dataset into smaller datasets and build a decision tree on it [3]. Additionally, each tree in the random forest uses only a subset of the attributes in the original dataset. The number of instances \( n \) and attributes \( m \) to be used for each decision tree are given as input to the algorithm. When an unknown instance needs to be classified the ensemble classification model pushes the instance to all the trees and returns the most frequently predicted class label as the final prediction of the new instance.

Many variations of these ensemble methods have been investigated [21][20][8]. The main difference between these varieties is how the weights for different instances during training, and weights for different trees or rules during testing are learned or evaluated.

### 3.3 Association rule based approaches

Association rule mining is best explained in the context of market basket analysis. Given a dataset that contains a large number of transactions, where each transaction being a list of products purchased by a customer, the database can be represented in a tabular form where each column is a product, each row is a transaction. Each cell contains either a one or a zero depending on whether the product was bought as a part of the transaction or not.

\[
<table>
<thead>
<tr>
<th>T_{ID}</th>
<th>Bread</th>
<th>Eggs</th>
<th>Meat</th>
<th>Soda</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_4 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( t_5 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( t_6 )</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( t_7 )</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( t_8 )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( t_9 )</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( t_{10} )</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( t_{11} )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
\]

Table 3.1: Sample market basket database.
From table 3.1 we can deduce the following:

- Soda is not a frequently bought item, whereas bread and eggs are.
- If a customer bought bread they also bought eggs, since the conditional probability of $P(\text{bread} \mid \text{eggs})$ is 1.

More combinations of items can be examined and used to draw more inferences about the buying habits of consumers in general. Statements that have a high probability are accepted as association rules, while the rest are discarded. Association rule mining algorithms are built to find such rules that are embedded in the data. However, association rule mining is mostly limited to binary datasets.

### 3.3.1 CBA

Bing Liu’s paper introduces CBA, Classification Based on Associations, a rule based classifier that makes predictions based on association rules extracted from the dataset [12] [13]. An association rule is a logical expression that presents a set of preconditions on the left hand side, and indicates a consequence of the preconditions on the right hand side.

Bing et al. propose building classifiers that are based on association rules where the predicate/consequent is always a class label and the antecedent is a combination of feature-value pairs from the table. In order to be able to use association rule mining algorithms for building classifiers on continuous valued data they propose that each continuous valued attribute be discretized. This may be done using entropy based, or frequency based, or MLD discretization techniques. In their paper Bing Liu et. al use an entropy based discretization technique proposed by Fayyad and Irani[9].

Once the feature values have been discretized they can be used to make association rules. The basic idea of association rule mining remains the same. The antecedent of the rule consists of a set of discrete feature-value pairs from different columns and the predicate contains a class label. The CBA algorithm then enumerates all combinations of discrete feature-value pairs and ranks them in order of confidence and then support. Once ordered, each rule is examined for whether it should be
accepted into the final list of classification rules. If a rule correctly predicts at least one previously uncovered row, it is accepted into the final set of rules.

CBA performs at par with C4.5 on most datasets tested [12]. However, like most association rule based approaches, the bottleneck of the algorithm is the rule generation step where it enumerates all possible rules. CBA has also been found to generate an excessive number of rules compared to other rule based classifiers [4].

3.3.2 GARC

One of the issues with association rule based classification is that as the number of columns, and distinct values within each column increases, the set of possible column-value combinations and therefore the number of candidate rules grows exponentially. GARC uses a heuristic to reduced the number of rules generated [4]. GARC first uses information gain to find the best attribute to make the first split of the dataset on. It then generates only those rules that contain the attribute chosen in the first step. In this way GARC prunes off a large part of the rule space. In tests, it produced only 5% of the number of rules generated by CBA while maintaining almost the same percentage of classification accuracy.

CBA and GARC allow the sharing of training instances for rule generation. However, both the methods generate all frequent item-sets first, and then choose the best subset of rules from them. This becomes a bottleneck when there are many frequent item-sets in the dataset and only a small subset is used in the final classifier. The approach described in this thesis combines these two stages by simultaneously updating the coverage constraints, thereby speeding up the process of building the classifier.

3.3.3 HARMONY

Wang et. al. introduced HARMONY, an instance centric association rule-based classifier. The algorithm explores the feature space using a prefix tree, and traverses the prefix tree in a depth first manner to find the best rule to cover each training instance [23]. It uses a number of heuristics
to order prefixes while exploring them, and to prune off unpromising areas of the search space. The authors propose that prefixes ordered by confidence, entropy or correlation are more helpful in enumerating useful rules quickly compared to an ordering based on support. HARMONY was shown to perform better [23] in terms of accuracy and speed compared to other rule based algorithms. However, because of its instance-centric approach the algorithm generates too many rules compared to existing approaches. Additionally, the algorithm does not take the different costs of misclassification into account while building classifiers.

Thabtah F. has conducted an extensive review of the existing approaches and algorithms that are based on association rule mining [19]

3.4 Cost sensitive Rule based classifiers

A significant amount of work has been done in converting existing cost-blind rule based classifiers into cost-sensitive ones. One of the major approaches used earlier was to post-process the model built by a cost-blind classifier. In this approach, first a cost-blind decision tree is built, then each leaf node is examined and is reassigned a class label that minimizes the expected cost of a prediction. This approach is useful if the misclassification costs are not known during the construction phase, or when these costs are likely to change frequently.

Another approach has been centered around the function that is used to choose split points and guide the decision tree algorithm. The aim is to modify this function so that it can take costs into account. It was first suggested by Breiman et al. that costs can be incorporated into the calculation of prior probabilities[2]. The instances of class $i$ can be weighted by the relative cost of misclassifying it. That is, the prior probability or weight of an instance of class $i$ becomes:

$$p_i = \frac{\text{cost}(i)(N_i/N)}{\sum \text{cost}(j)N_j/N}$$

A similar approach was used by Quinlan in C4.5 which could initialize instance weights according
to misclassification cost of that instance’s class. C4.5-CS is an improvement to GINIAlteredPriors that was proposed by Ting et. al. [22]. It improved on GINIAlteredPriors by using costs in the post-hoc pruning stages as well.

However, this approach does not extend well to datasets with more than two classes. It is because the function assumes that the cost of misclassifying an instance is independent of which class it gets misclassified as. It would not be possible to use the type of cost matrix seen in table 3.2 in the above function:

<table>
<thead>
<tr>
<th></th>
<th>Pred. A</th>
<th>Pred. B</th>
<th>Pred. C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actually A</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Actually B</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Actually C</td>
<td>7</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3.2: A sample non-uniform, non-symmetric cost matrix.

Margineantu and Dietterich studied the performance of functions that incorporated such cost matrices into the split function [15]. They propose using weighted averaging, or maximum cost as ways of condensing the non-uniform cost matrix into a cost vector, and then using it in the split function[15]. However, their results showed that such operations do not always produce better results compared to other techniques. They also propose a method called EvalCount. In this method, first a cost insensitive tree is built on a subset of the training data. The rest of the training set is used to validate this tree and identify those instances that were misclassified and had a high misclassification cost. The weights of these instances are then adjusted according to these costs which can be used for building the tree again.

There has also been work done to build cost sensitive classifiers using the concept of boosting. Most of these classifiers are based on AdaBoost, an adaptive boosting algorithm that incorporates instance weights[20]. UBoost and AdaCost are two examples of such classifiers[21] [8]. The difference between these algorithms is mostly the way in which instance weights are reevaluated, and how weights for different trees are calculated.

Lomax and Vadera present a comprehensive survey of existing decision tree based approaches
which have been modified to take misclassification costs and test costs into account[14].

There have been some efforts in building wrapper algorithms that are able to convert existing cost blind classification algorithms into cost sensitive algorithms by wrapping them with meta algorithms[6]. MetaCost is a popular wrapper method that is run on the dataset before the actual classifier. It works by relabeling instances from the dataset that are at the boundary of two or more classes. An instance in such a position is assigned the label of the most important class in its vicinity. This relabeling takes into account the relative costs of misclassification and the probabilities of that point belonging to different classes. Instances are relabeled in a way that when the base classifier is run on the dataset, the rules at such boundaries predict classes that minimize the misclassification cost.

Figure 3.1: Original dataset with indistinct boundary.

3.5 Novelty of our approach and results

The approach presented here aims to overcome the disadvantages of existing techniques listed above. By combining a prefix based search for rules with effective pruning heuristics and a rule quality function which is cost-sensitive, this work aims to build rule sets that have higher support and precision, and lower expected misclassification costs compared to rule sets generated by existing techniques. The results shown in Chapter 5 support our methodology for generating high-confidence, low-cost
Figure 3.2: Relabeled dataset with boundary points relabeled to more important class “A”.

rules.
Chapter 4

Approach Adopted for Rule Induction

The approach described in this work overcomes some of the drawbacks of rule-based classifiers in the following manner:

- It allows instances to be used multiple times for rule generation.

- It proposes, implements, and investigates various optimizations for reducing the time required for training.

- It proposes a cost-sensitive function to incorporate misclassification costs associated with different classes in the rule induction process.

The algorithm uses a prefix based search to explore the feature space while mining for frequent patterns that can be used for classification. Unlike previous approaches, this algorithm traverses the prefix tree in a breadth first manner. This causes the algorithm to find shorter rules that have higher support and significance, and are therefore better for classifying new instances.

4.1 Preliminaries

Some basic definitions and nomenclature will be reviewed in this subsection.
4.1.1 Defining a rule

A pattern or a rule can be thought of as a frequently occurring condition across one or more features in the database. A rule, $R = \{(f_i, v_i)\}$ is a set of feature-value pairs, where feature $f_i$ has value $v_i$. A rule, $R = \{(f_i, v_i)\}$, is applicable to an instance in the dataset if, for each feature $f_i$ in the rule, the value of the feature $f_i$ in the instance is $v_i$. The length of a rule is the number of feature-value pairs that the rule contains. A pattern that is unique to a class can be used to correctly predict unknown instances of that class.

4.1.2 Support and coverage of a rule

The support of a rule $R$ in a class $\text{Class}_i$, denoted by $\text{Supp}_i(R)$, is the number or fraction of instances in $\text{Class}_i$ to which $R$ is applicable. The support of a rule, denoted by $\text{Supp}(R)$, is the largest support that the rule has across all classes.

$$\text{Supp}(R) = \max\{\text{Supp}_i(R)\}$$

If the support of a rule is lower than a predefined minimum threshold it can be discarded. No rule whose condition set is a superset of the discarded rule is generated since the support of such a rule can only be less than or equal to the support of the discarded rule. This is a result of the downward closure property of set intersection operations. The support of a rule is used as measure for pruning the search space by the algorithm.

The coverage of a rule $\text{coverage}(R)$ is a set of row IDs across all classes in the training set that satisfy the conditions in the rule.

4.1.3 Quality of a rule

A rule defines a pattern across a subset of rows and columns, where each column in the condition set has a fixed value across all the supporting rows. The quality of a rule defines how unique a pattern is to any one class.
The support of a rule across different classes, along with misclassification costs of instances, are used to measure the quality of a rule.

We define the quality of a rule as the cost-weighted entropy of it’s support across different classes, or as the entropy of different misclassification costs. That is,

\[ \text{quality}(R) \text{ or } \text{entropy}(R) = - \sum p_i \log m(p_i), \]

where

\[ p_i = c_i / \sum c_i \]
\[ c_i = \text{Supp}_i(R) \text{ cost}(i), \]
\[ m = \text{number of classes in the database} \]
\[ \text{cost}(i) = \text{cost of misclassifying an instance of class } i \]

For example, given a rule has support of 10 and 45 respectively in two classes A and B, where A has a misclassification cost of 1 unit and B has a misclassification cost of 2, the quality of this rule will be calculated in the following manner:

\[ c_A = 10 \times 1 = 10, \quad c_B = 45 \times 2 = 90 \]
\[ p_A = 10/100 = 0.1, \quad p_B = 90/100 = 0.9 \]
\[ \text{quality}(R) = -\{0.1\log(0.1) + 0.9\log(0.9)\} = 0.46 \]

Given a rule, a lower entropy value means that a large portion of the cost comes from misclassifying one class, say class i. Class i then becomes the predicted class of this rule, as making that prediction avoids the largest misclassification cost.

When all the misclassification costs are equal (say, to 1), the above formula reduces to a cost-blind function based on support across different classes. The function assumes that the cost of misclassifying an instance is fixed, irrespective of which other class it gets misclassified as.

However, for multi-class problems where the cost of misclassifying an instance may depend on which class it gets misclassified as. In order to accommodate such a cost matrix, the misclassification costs for a particular can be the sum of all possible misclassifications for that class. This condenses
the cost matrix and makes it easy to apply the above function to. This is similar to one of the methods proposed by Dietterich et al[15].

The quality of the rule is also used to prune the search space. Two thresholds are chosen to define the ranges of acceptable, workable and unfit rules. The threshold \( t_{good} \) defines the upper limit for the entropy of a good rule. If the entropy of a rule is below a threshold, \( t_{good} \), we can accept it into the final set of classification rules and do not need to make the rule more specific by including more features. The reason being that rules generated by further specialization may have better quality (lower entropy) but will not cover any extra instances due to the downward closure property. The support for any such rules, after specialization, can only go down.

The threshold \( t_{bad} \) defined the upper limit for the entropy of a workable rule. Rules with entropy values between \( t_{good} \) and \( t_{bad} \) are considered workable, meaning that they can be kept around for further specialization in the next generation. Those rules that have an entropy higher than \( t_{bad} \) can be discarded as unpromising rules. However, this threshold must be kept reasonably high to allow some rules to develop by further specializations for a few iterations before they are discarded.

![Figure 4.1: Entropy vs Support across classes.](image)

### 4.1.4 The apriori algorithm

The apriori algorithm is a classic algorithm for finding frequent patterns in a binary dataset. It starts by finding all basic patterns of size one. Each patterns of length \( n \) is then specialized by merging it with these basic patterns. The support of the new pattern of length \( n+1 \) is the number
of rows or instances common to both the parent patterns. If the support is lower than some minimum support threshold, the pattern is discarded, and no patterns that are a superset of this are generated.

4.2 Approach

The approach is divided into two stages. In the first stage, the dataset is discretized using an entropy based discretization method. In the second stage, the feature-value space is explored in a breadth first manner in search of good rules for classification.

Discretization is done using Fayyad and Irani’s MDL discretization technique [9]. This method is chosen because it was used by Bing Liu et al. in their original paper on association rule based classification[12]. The method used there was the entropy based method proposed by Fayyad and Irani[9]. The alternatives to using MDL based discretization are equal-frequency or equal-width discretization. As a test, all datasets were discretized using these methods also and run though different classifiers in the OrangeCanvas toolkit. In addition to the reason stated earlier, these two discretization techniques were not used for the following two reasons:

- Classifiers achieved lower classification accuracy on datasets discretized by these two methods compared to MDL.
- Both Equal Frequency and Equal Width discretization techniques require the user to set the number of intervals for discretization. It is difficult to ascertain what the optimal number of intervals for each dataset is and so adds complexity to the testing methodology for no obvious gains. No such parameter is needed for the MDL based technique.

The algorithm works with categorical data only, and therefore needs real valued attributes to be discretized before running. However, this also works to it’s advantage as the algorithm does not need to exhaustively search for good split points in the continuous valued attribute.

In order to find good rules the algorithm examine features, their values, and interactions among different feature-value pairs in an iterative and systematic manner using the prefix based search. For
<table>
<thead>
<tr>
<th>Attr1</th>
<th>Attr2</th>
<th>Attr3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.9</td>
<td>2.4</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>11.1</td>
<td>2.5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>2.9</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>3.9</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>4.5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>1.12</td>
<td>4.5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>2.9</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2.2</td>
<td>1.1</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2.1</td>
<td>9.1</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>2.4</td>
<td>12.1</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>7.7</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>12.1</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>12.5</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>21.8</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>16.5</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>11.2</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>12.5</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

Table 4.1: Original sample real valued dataset

<table>
<thead>
<tr>
<th>Attr1</th>
<th>Attr2</th>
<th>Attr3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;= 9</td>
<td>&lt;= 5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>&gt;= 9</td>
<td>&lt;= 5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>&lt;= 5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>&lt;= 5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>&lt;= 5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>&lt; 2.5</td>
<td>&lt;= 5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>&lt; 2.5</td>
<td>&lt;= 5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>&lt; 2.5</td>
<td>&lt;= 5</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>&lt; 2.5</td>
<td>&gt; 7</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>&lt; 2.5</td>
<td>&gt; 7</td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td>7</td>
<td>&gt; 7</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>&gt; 7</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>&gt; 7</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>&gt; 7</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>&gt; 7</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>&gt; 7</td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td>7</td>
<td>&gt; 7</td>
<td>1</td>
<td>B</td>
</tr>
</tbody>
</table>

Table 4.2: Discretized sample dataset
each candidate rule, the set of satisfying instances are found out, and used compute the quality(R) for the rule R. The algorithm accepts rules that have low entropy. The remaining rules are set aside for further specializations in future iterations.

![Set of all distinct attribute-value pairs](image)

Figure 4.2: Prefix tree search for test dataset.

Figure 4.2 shows the way a prefix tree enumerates feature-value combinations to find good classification rules for the test dataset with \( \text{minSupp} \) set to 3.

First, a set of all rules of length one are generated. The function \( \text{makeBasicRule} \) iterates over each feature or column, and makes a rule for each distinct value of that feature. The function returns the set of all candidate rules formed in this manner.

\[
\text{ruleSet} \leftarrow \phi; \\
\text{for } f_i \text{ in } F \text{ do} \\
\quad \text{for each distinct } v_j \text{ in } f_i \text{ do} \\
\quad\quad \text{ruleSet} \leftarrow \text{ruleSet} \cup \{(f_i, v_j)\}; \\
\quad \text{end} \\
\text{end} \\
\text{return ruleSet}
\]

**Algorithm 1:** makeBasicRules
Rules that have a support lower than a pre-specified threshold $\min_{\text{supp}}$ are removed from this set. “Attr1 is $\geq 9$” gets removed for this reason. The rules that have entropy lower than $t_{\text{good}}$ are also removed from the set $\text{baseSet}$ and are added to the set of rules accepted for the classifier, called $\text{acceptedRules}$. “Attr1 is $< 2.5$” and “Attr2 $\leq 5$” are accepted into $\text{acceptedRules}$ for this reason. It should be noted that if the dataset was split exclusively, the candidate “Attr1 is $< 2.5$” would have been discarded for having low support as part of its support is shared with the other candidate.

The remaining set of rules of length one represents a set of basic rules and is stored in $\text{baseSet}$. This set consists of ”Attr 2 is $> 7$“ and ”Attr1 is 7“. The candidates in this set are ordered. The order is based on the quality or entropy of the rule, but can be ordered on other metrics like f-score, precision or support too.

Next, compound rules are generated. For the first iteration in this stage, all rules from $\text{baseSet}$ are copied into a temporary set called $\text{workingSet}$. In each iteration, a rule, $R_a = \{(f_i, v_i)\}$, from $\text{workingSet}$ is specialized by adding a condition defined by a rule from the set $\text{baseSet}$. This operation is performed by the function $\text{combineRules}$.

For example, given a rule from $\text{workingSet}$ $R_1 = \{(f_1, v_3)\}$, (read as “feature $f_1$ has value $v_3$”), and a rule from $\text{baseSet}$, $R_{\text{base1}} = \{(f_4, v_1)\}$, the following new rule can be formed by the set conjunction of their conditions:

$$R_b = \{(f_1, v_3), (f_4, v_1)\}$$

Which reads “feature $f_1$ has value $v_3$ AND feature $f_4$ has value $v_1$”. The support of class $i$ in $R_b$, $\text{Supp}_i(R_b)$, will contain those rows which were common to $R_1$ and $R_b$, and belonged to class $i$. A temporary label is assigned to the new rule representing the class that it predicts. This is the class that has highest cost weighted support in the rule. In the example for the test dataset, candidate ”Attr2 is $> 7$“ is specialized by adding the condition “Attr1 is 7”.

Once a new rule, $R_b$, is formed, the following conditions are checked:
\[
\begin{align*}
\text{baseSet} & \leftarrow \text{makeBasicRules} \\
\text{baseSet} & \leftarrow \text{prune}(\text{baseSet}) \\
\text{workingSet} & \leftarrow \text{baseSet} \\
\text{while } & \text{workingSet } \neq \phi \text{ do} \\
\text{nextWorkingSet} & \leftarrow \phi \\
& \text{for } r \text{ in workingSet do} \\
& \text{for } b \text{ in baseSet do} \\
& \quad r_k \leftarrow \text{combineRules}(r, b) \\
& \quad \text{if } \text{entropy}(r_k) > t_{\text{bad}} \text{ then} \\
& \quad \quad \text{continue} \\
& \quad \text{else if } \text{supp}(r_k) \leq \text{minSupp}(k) \text{ then} \\
& \quad \quad \text{continue} \\
& \quad \text{else if } \text{entropy}(r_k) \leq t_{\text{good}} \text{ then} \\
& \quad \quad \text{if } \text{increasesCoverage}(r_k) \text{ then} \\
& \quad \quad \quad \text{acceptedRules} \leftarrow \text{acceptedRules} \cup r_k \\
& \quad \quad \quad \text{updateCoverage}(r_k) \\
& \quad \quad \text{else} \\
& \quad \quad \quad \text{nextWorkingSet} \leftarrow \text{nextWorkingSet} \cup r_k \\
& \quad \text{end} \\
& \text{end} \\
& \text{workingSet} \leftarrow \text{nextWorkingSet} \\
& \text{prune}(\text{workingSet}) \\
& \text{prune}(\text{baseSet}) \\
\text{end}
\end{align*}
\]

**Algorithm 2:** Rule Generation

1. Is the entropy of the rule higher than \( t_{\text{bad}} \).

2. Is the support \( \text{Supp}_i(R_b) \), for the class it predicts, less than the minimum support threshold for that class.

If either condition is true for \( R_b \), the algorithm ignores this rule and continues to the next rule without adding \( R_b \) to the set of final rules, \( \text{acceptedRules} \), or to the set of rules to be processed in the next iteration \( \text{nextWorkingSet} \).

If neither of those two conditions are true but the entropy of the rule is greater than \( t_{\text{good}} \) it is added to \( \text{nextWorkingSet} \). If the entropy is less than \( t_{\text{good}} \) the rule is added to \( \text{acceptedRules} \) as long as it covers at least one instance previously not covered by any accepted rule for that predicted class.
class. Once a rule is added to acceptedRules the set of uncovered rows belonging its predicted class is updated. The rows of the predicted class of the rule are marked as covered but are not removed from further participation in the rule generation process.

In the example for the test dataset, the new specialized rule has zero entropy, and covers 7 rows that were previously uncovered by any accepted rule.

As the rule becomes more specialized, its support across different classes changes. The objective is to find rules that have much higher support in one class than in any other. Such a rule will have lower entropy and is suitable for use as a classification rule.

At the end of each iteration, the sets baseSet and nextWorkingSet are updated. This is done to remove rules that have become redundant since they were added to these respective sets. A rule becomes redundant when all of its supporting rows have been covered by one or more accepted rules. These operations are performed by the function prune(set). Additionally, nextWorkingSet may be pruned down to a smaller size to reduce the number of rules to be processed in the next iteration. This may be done by keeping only the top $K$ rules in the nextWorkingSet based on their $fScore$ or some other metric.

At the end of an iteration, the rules in workingSet are replaced by the rules in the nextWorkingSet, the process is repeated. The algorithm terminates when one of the following conditions is met:

- Rules of length $F$ have been generated, where $F$ is the number of features in the dataset. In other words, it terminates after $F$ iterations.

- nextWorkingSet is empty because all the generated candidate rules were pruned off.

- The rules in acceptedRules set cover all the instances across all classes.

When training ends, acceptedRules contains a set of rules with their associated quality metric values and class labels. It is possible that some instances were not covered during the training process. Consequently, it is possible that a test instance may not get a predicted class. This is a common issue with all association rule based classification approaches, and is usually resolved by using the majority class in the training dataset as the default predicted class for such instances. In
the case of this algorithm, another set of candidate rules called secondarySet may be maintained during rule generation. Rules that have entropy close to the threshold we desire, but not lower than it, can be added to this set. At the end of training, rules from the secondarySet can be selected to cover the instances that do not have low entropy rules covering them.

4.3 Classifying a new instance

When a new instance needs to be classified, it is assigned the label of the rule that applies to this instance and has the lowest entropy. The entropy of the rule is used to resolve conflicts when two or more rules with different class labels apply to an instance. If no rule in acceptedRules applies to a new instance, its class may be left undecided, or predicted based on the majority or rules taken from the secondarySet. In many applications however, it is preferable to leave instances undecided if the confidence of the rule’s prediction is low.

4.4 Choosing entropy cutoffs and a minSupp

The minSupp value should be chosen depending on the size of the dataset. It is usually set to a percentage of the total number of instances in the dataset, or different minSupp values can be used for different classes in the dataset. A minSupp value of 1% was used by Bing Liu et al. in their work[12]. Choosing a high value for this parameter will result in too few rules being accepted and a lower accuracy of the classifier overall, whereas a very low value will result in over-fitting the data and generation of too many rules.

As for choosing the cutoffs, this almost entirely on how well separated are the classes in the dataset. If the classes are easily separable, strict cutoffs like 0.1 can be used, where as higher cutoffs may be preferable if the classes overlap. Alternately, the acceptable percentage of undecided instances be used to guide the process of choosing a cutoff. A strict cutoff can result in high accuracy on predictions but also a high rate of undecided instances, conversely, a lenient cutoff will result in less instances being left undecided but lower overall classification accuracy.
The upper boundary of the entropy cutoff should be kept reasonably high, so as to let the poor quality rules have the opportunity to specialize and improve over a few iterations before they get discarded. If this cutoff is kept low, too many rules will get discarded, some of which could have possibly been used for classification.
Chapter 5

Experimental Setup and Results

5.1 Experimental Setup

11 datasets from UCI’s machine learning data repository were used for training and validation using ten-fold cross validation[10]. This means that the original dataset is divided into 10 subsets of equal size. Then, the classifier is trained on nine subsets and tested on the remaining one. This training and testing cycle is carried out 10 times with each one of the ten subsets being used for testing in one iteration. The performance for each iteration is recorded and averaged to represent the final result.

Classification Tree, Random Forests and CN2 were used to compare the performance of the algorithm on six small datasets. Performance is measured in terms of classification accuracy, average precision of the generated rule set, length of rules generated, and average support of each rule. The \textit{minSupport} for each of these classifiers was varied to study its effect on the accuracy of the classifier. Orange Canvas, a free and open source data mining toolkit was used to run implementations of these three classifiers[5].

The performance of this algorithm was also compared to an existing association rule based classifier, HARMONY [23], on five large databases.

The cost sensitivity of the quality function for generating optimal rule sets is measured empir-
ically by varying the cost matrix for a dataset. The changes in our classifier’s recall for different classes is observed as the cost matrices are varied.

We test our hypothesis about reduction in rule length, and increase in the average precision and support of the generated ruled compared to the rule set generated by existing techniques in section 5.2.5

Additionally, OrangeCanvas was used to record the classification accuracy of Support Vector Machine and K Nearest Neighbor classifiers. A table comparing the results of our approach with theses classifiers is also included in this chapter.

The details of the discretized datasets are show in table 5.1

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Attributes</th>
<th>Instances</th>
<th>Classes</th>
<th>Distinct feature-value pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adult salary</td>
<td>13</td>
<td>48841</td>
<td>2</td>
<td>168</td>
</tr>
<tr>
<td>Mushrooms</td>
<td>22</td>
<td>8124</td>
<td>2</td>
<td>119</td>
</tr>
<tr>
<td>Nursery</td>
<td>8</td>
<td>12959</td>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>Page Blocks</td>
<td>10</td>
<td>5473</td>
<td>5</td>
<td>76</td>
</tr>
<tr>
<td>Waveform</td>
<td>19</td>
<td>5000</td>
<td>3</td>
<td>107</td>
</tr>
<tr>
<td>Breast cancer</td>
<td>10</td>
<td>699</td>
<td>2</td>
<td>33</td>
</tr>
<tr>
<td>Glass</td>
<td>7</td>
<td>214</td>
<td>6</td>
<td>26</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>16</td>
<td>155</td>
<td>2</td>
<td>48</td>
</tr>
<tr>
<td>Iris</td>
<td>4</td>
<td>150</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>Pima</td>
<td>6</td>
<td>768</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>Wine</td>
<td>13</td>
<td>178</td>
<td>3</td>
<td>40</td>
</tr>
</tbody>
</table>

5.2 Experiments and Results

5.2.1 Analysis on small datasets

The algorithms were run on the 6 small datasets with \textit{minSupport} of 10 and 20. The results of our algorithm are compared with those of Classification Tree, Random Forest and CN2 in tables 5.2, 7.1, 5.3 and 7.2. All runs were done with ten-fold cross-validation.
The results in tables 5.2 and 7.1 show that our proposed classifier performs better than the other three in four out of the six datasets. The classification accuracy in the Wine dataset is above 98% which is much higher than CN2 which has a classification accuracy of about 93% and is the highest among the other classifiers. Only slight improvement is seen in the Pima, Breast Cancer and Glass datasets.

When the minimum support is increased, the classification accuracy of our approach remains relatively same and better in four out of the six datasets. The is a marked decrease in accuracy for the other three classifiers for the Glass and Wine datasets, whereas the classification accuracy of our approach remains the same as before. This hints to the effectiveness of reusing training instances as this can be particularly helpful when there aren’t many instances to train on.

In section 5.2.5, the average rule length, rule precision and support is compared for the rule-set generated for each dataset by Classification Tree, CN2 and our approach. The results show that our approach performs better in most cases when compared using those criteria.
Table 5.2: Accuracy Compared in % to Classification Tree, CN2 and Random Forest on small datasets with min Support = 10

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classification Tree</th>
<th>CN2 Rules</th>
<th>Random Forest</th>
<th>$t_{good}$ cutoff</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>95.57</td>
<td>94.42</td>
<td>94.42</td>
<td>0.1</td>
<td>95.85</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>94.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>94.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>95.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>96.28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>96.14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td>94.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>94.28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>94.42</td>
</tr>
<tr>
<td>Glass</td>
<td>63.03</td>
<td>68.66</td>
<td>62.64</td>
<td>0.1</td>
<td>49.07</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>58.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>60.28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>61.68</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>69.16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>64.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td>64.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>50.47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>48.13</td>
</tr>
</tbody>
</table>

5.2.2 Analysis on large datasets

Performance results for HARMONY were taken from the paper [23] and compared to the results of our classifier on the six large datasets from UCI. The algorithms were compared in terms of classification accuracy, rule-set size, and run time when validated using ten-fold cross validation with a minimum support of 50 and the average of the ten runs is reported. The results are shown in tables 5.4 and 7.3. 

The results show that the algorithm has higher classification accuracy in most cases compared to HARMONY, while at the same time, the number of rules needed to achieve this level of accuracy is much lesser than that needed by HARMONY. For example, for the “Adult” dataset, our approach can be run at a $t_{good}$ cut off of 0.9 and still attain a classification accuracy of 82.76% with only 52 rules and a run-time of less than 1.5 seconds, compared to HARMONY’s classification accuracy of 81.9% attained by generating over 6000 rules and a run-time of about 20 minutes. From the
Table 5.3: Accuracy Compared in % to Classification Tree, CN2 and Random Forest on small datasets with min Support = 20.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classification Tree</th>
<th>CN2 Rules</th>
<th>Random Forest</th>
<th>$t_{good}$ cutoff</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td><strong>95.13</strong></td>
<td>95.13</td>
<td>92.13</td>
<td>0.1</td>
<td>95.14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>94.42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>94.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>95.28</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td><strong>96.42</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td><strong>96.28</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td>94.71</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>94.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>94.42</td>
</tr>
<tr>
<td>Glass</td>
<td>58.83</td>
<td><strong>64.39</strong></td>
<td>59.74</td>
<td>0.1</td>
<td>50.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>55.14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>59.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>58.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td><strong>68.22</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td><strong>64.49</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td><strong>65.42</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>54.21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>47.20</td>
</tr>
</tbody>
</table>

table 5.4 and 7.3 it can be seen that the runtime of this algorithms is also much lower than that of HARMONY, usually running an order of magnitude faster.
Table 5.4: Accuracy Compared in % to Harmony (HMNY) on large datasets with min Support = 50

<table>
<thead>
<tr>
<th>Data set</th>
<th>$l_{\text{good cut-off}}$</th>
<th>Accuracy</th>
<th># Rules</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Our algorithm</td>
<td>HMNY</td>
<td>Our algorithm</td>
<td>HMNY</td>
</tr>
<tr>
<td>Adult</td>
<td>0.1</td>
<td>82.37</td>
<td>81.9</td>
<td>6431.3</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>82.55</td>
<td>282.3</td>
<td>1462.12</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>82.89</td>
<td>179.9</td>
<td>14.624</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>83.23</td>
<td>148.6</td>
<td>10.687</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>83.90</td>
<td>146</td>
<td>9.194</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td><strong>84.38</strong></td>
<td><strong>126.2</strong></td>
<td>9.887</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>84.30</td>
<td>121.2</td>
<td><strong>7.201</strong></td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>82.80</td>
<td>55.6</td>
<td>5.246</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>82.76</td>
<td>51.4</td>
<td>1.276</td>
</tr>
<tr>
<td>Mushrooms</td>
<td>0.1</td>
<td><strong>100</strong></td>
<td>35.5</td>
<td>95.9</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>99.80</td>
<td>24.3</td>
<td>0.729</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>99.40</td>
<td>11.3</td>
<td>0.216</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>99.40</td>
<td>11</td>
<td>0.197</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>99.47</td>
<td>12.6</td>
<td>0.186</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>99.40</td>
<td>12</td>
<td>0.119</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>98.22</td>
<td>11</td>
<td>0.134</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>98.22</td>
<td>10.9</td>
<td>0.119</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>98.22</td>
<td>10.9</td>
<td>0.134</td>
</tr>
<tr>
<td>Nursery</td>
<td>0.1</td>
<td>79.26</td>
<td>60.6</td>
<td>391.64</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>85.82</td>
<td>87.5</td>
<td>0.729</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td><strong>91.31</strong></td>
<td><strong>98.3</strong></td>
<td><strong>0.735</strong></td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>90.16</td>
<td>84.2</td>
<td>0.683</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>79.66</td>
<td>39.5</td>
<td>0.483</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>78.37</td>
<td>47.8</td>
<td>0.446</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>74.07</td>
<td>19.9</td>
<td>0.186</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>71.27</td>
<td>23.3</td>
<td>0.083</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>70.97</td>
<td>23.6</td>
<td>0.083</td>
</tr>
</tbody>
</table>

5.2.3 Analysis with non-uniform cost matrices

In order to measure the sensitivity of the rule quality function, the cost matrix was varied and the recall for different classes in the dataset was measured. The cost ratio was varied from 5:1 to 1:5 for two 2-class datasets. This means that first one class has a misclassification cost of 5 while the
other has a misclassification cost of 1, then the cost of the first class is reduced gradually by 1 each
time till both classes have a misclassification cost of 1, then gradually the misclassification cost of
the second class is increased by 1 till it reached 5. The changes in recall for each class as their
relative cost of misclassification is varied is recorded. The results are shown in figure 5.1, 5.2 and
5.3. The minSupp was set to 10 and entropy cutoff to 0.5.

The plots in figures 5.1 and 5.2 show that as the cost of misclassifying an instance of a class
increases, the recall for that class also increases. For example, at the extreme left when the cost
ratio is 5:1 in favor of Class-1, the recall for class one is much higher than that of Class-2. As the
costs before more balanced, this difference reduces, and then the recall grows for Class-2 as the cost
ratio favors Class-2. When the cost ratio is 1:5 in favor of Class-2, it is seen that the recall of Class-2
is 100% while that for Class-1 is 0%. This is because the algorithm assigns more importance to
instances of the class that is expensive to misclassify and readily accepts rules that predict in favor
of this class. This reduces costs incurred from misclassifying instances at the decision boundaries
by making predictions in favor of the class that is expensive to misclassify.

It should also be noted that the Pima and Hepatitis datasets have imbalanced class distributions.
Pima has twice as many instances in the “Positive” or “1” class than in the “negative” or “0” class.
Hepatitis dataset has four times as many instances in the “Live” or “2” class than in the “Die” or
“1” class. As the cost ratios are varied to give more importance to the minority class, the recall
for that class increases. This property of the cost matrix and the learned classifier can be used to
correctly handle class imbalances in the dataset. An imbalance of roughly 2:1 also exists in the
Breast cancer dataset, however, the classes in this dataset are easily separable and therefore the
pattern seen in Pima and Hepatitis datasets is not seen here.

Similar tests were performed for the two 3-class datasets, Wine and Iris. The cost of misclassifying
an instance of a class was constant regardless of which wrong class label was predicted for it.
Table 5.2.3 shows the kind of cost function used in this experiment.

In order to show that the trend seen in the Pima and Hepatitis datasets can also be seen in 3-class
datasets, the cost of misclassifying one class was kept constant while the ratio of misclassification
Figure 5.1: Pima Indian Diabetes Dataset from UCI Data Repository. As the weights are reduced or increased for a class, the recall for that class also increases.

<table>
<thead>
<tr>
<th>Actual/Predicted</th>
<th>Class A</th>
<th>Class B</th>
<th>Class C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Class B</td>
<td>Y</td>
<td>0</td>
<td>Y</td>
</tr>
<tr>
<td>Class C</td>
<td>Z</td>
<td>Z</td>
<td>0</td>
</tr>
</tbody>
</table>

costs for the other two classes was varied from 5:1 to 1:5. The results are shown in figure 5.5 and 5.4.

In the Iris dataset, the Setosa class is linearly separable from the other two, which is why it was decided to keep its cost fixed and vary the cost for the other two classes. The instances of Versicolor and Virginica overlap in multiple feature spaces. As a result, when their relative costs are varied, the recalls of these classes reflect the same. At 1,5,1 the recall of Versicolor is high and Virginica is low, and at 1,1,5 the recall of Virginica is high and Versicolor is low. The recall of Setosa class is constant since it is linearly separable.

The classes in Wine, much like the Breast Cancer dataset, show much less variation compared to other datasets. The dataset has easily separable classes, but there is still a noticeable increase in the recall of Class-1 when its misclassification cost is increased. These results provide evidence that the rule quality function is working as intended for cost-sensitive datasets.
Figure 5.2: Hepatitis Dataset from UCI Data Repository. This dataset has an imbalance of 4:1, and an effect similar to the one seen for Pima is seen here. As the cost ratio grows in favor of the “1” class, the difference in their recall reduces.

Figure 5.3: Breast Cancer Dataset from UCI Data Repository. Despite having an imbalance of 2:1, since the classes are well separated, the pattern observed in the Hepatitis and Pima dataset is not observed here.
Figure 5.4: Iris dataset from UCI Data repository. The recall of Setosa remains constant while the recall for the others varies in the same fashion as seen before for Pima and Hepatitis.

Figure 5.5: Wine dataset from UCI Data repository. Similar to the Breast Cancer dataset, the classes here are well separated, so the recall for all classes does not vary much.

5.2.4 Reduction in misclassification costs by leaving instances undecided

The last set of experiments are aimed at showing the effects of leaving some instances undecided. If a strict cutoff for acceptable rules is selected, the algorithm generates only high confidence rules and as a result, some instances tend to be left unclassified during testing. No high confidence rules were found for these instances since these instances are likely to be misclassified. By choosing to
<table>
<thead>
<tr>
<th>Actual/Predicted</th>
<th>Class A</th>
<th>Class B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class A</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Class B</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

withhold prediction on these instances, the classifier is able to reduce its running cost.

Figures 5.6 and 5.7 show how the cost incurred increases as the threshold for acceptable rules is raised. For this experiment, the non-diagonal costs were set to 1 as shown in table 5.2.4.

The tables 5.6 and 5.7 show that as the threshold for \( t_{\text{good}} \) is raised, since more lower confidence rules are accepted by the classifier, the percentage of instances left undecided goes down, but the accuracy of the classifier also goes down. The accuracy in this experiment is measured as the percentage of correctly predicted instances to all predicted instances. Instances for which no prediction was made, the ones left undecided, were not used to compute this accuracy.

![Plot for percentage undecided instances, prediction accuracy, and incurred cost for Pima dataset. As the entropy cutoff \( t_{\text{good}} \) is raised, the percentage of instances left undecided decreases, and the cost of misclassification increases.](image)

Figure 5.6: Plot for percentage undecided instances, prediction accuracy, and incurred cost for Pima dataset. As the entropy cutoff \( t_{\text{good}} \) is raised, the percentage of instances left undecided decreases, and the cost of misclassification increases.
Figure 5.7: Plot for percentage undecided instances, prediction accuracy, and incurred cost as $t_{\text{good}}$ cutoff is raised for Glass Identification dataset

Table 5.5: Effects of varying entropy cutoff on classification accuracy, and undecided percentage and cost incurred from misclassifications

<table>
<thead>
<tr>
<th>Dataset name</th>
<th>$t_{\text{good}}$ cutoff</th>
<th>Classification Accuracy</th>
<th>Undecided</th>
<th>Cost incurred</th>
</tr>
</thead>
<tbody>
<tr>
<td>Glass types</td>
<td>0.1</td>
<td>81.82</td>
<td>84.58</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>79.41</td>
<td>68.22</td>
<td>1.40</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>82.91</td>
<td>45.33</td>
<td>2.00</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>76.98</td>
<td>35.05</td>
<td>3.20</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>65.90</td>
<td>19.16</td>
<td>5.90</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>68.42</td>
<td>2.34</td>
<td>6.60</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>65.42</td>
<td>0.00</td>
<td>7.40</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>48.60</td>
<td>0.00</td>
<td>11.00</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>48.60</td>
<td>0.00</td>
<td>11.00</td>
</tr>
<tr>
<td>Pima</td>
<td>0.1</td>
<td>83.33</td>
<td>96.88</td>
<td>0.40</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>90.16</td>
<td>66.93</td>
<td>2.50</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>90.38</td>
<td>55.34</td>
<td>3.30</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>87.17</td>
<td>40.10</td>
<td>5.90</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>84.99</td>
<td>33.20</td>
<td>7.70</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>83.77</td>
<td>24.61</td>
<td>9.40</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>80.85</td>
<td>14.32</td>
<td>12.60</td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>76.25</td>
<td>6.25</td>
<td>17.10</td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>76.43</td>
<td>0.00</td>
<td>18.10</td>
</tr>
</tbody>
</table>
5.2.5 Effect on Average rule length, precision, and support

The algorithm’s performance was also compared in terms of average length, precision and support of rules generated. Table 5.6 shows the values for average rule length, precision and support for Classification Tree, CN2 and this algorithm. All classifiers were run with a minimum support of 5.

The table shows that the algorithm always forms rules that are shorter in length. This is because the algorithm traverses the search space in a breadth first manner using the prefix tree. Shorter
rules are preferable since they are easier to understand and can also mean that lesser tests need to be run to find out the class of a new instance. The algorithm almost always generates rules that have higher support than the other two classifiers, which is also a consequence of breadth first traversal. The deeper the decision tree goes, the lower the support gets for a rule. With shorter rules, this algorithm does not go very deep and thus has rules with higher support. The OrangeCanvas canvas implementation is able to merge leaf nodes in the decision tree and is therefore able to get higher support in some cases.

Depending on the $t_{good}$ entropy cutoff and minSupp used, our algorithm may or may not over all instances in the dataset. This means that some rows in the training set could not be correctly classified by the generated rule-set. This is reflected in the “Algo-Coverage-%” row. Although in most cases this value is reasonably high, it is significantly low for the “Glass” and “Pima” dataset. For these datasets, the algorithm was able to find rule that cover only 66% and 50% of the training samples, however the rules it does generate have much higher precision compared to those generated by other classifiers. In general, our algorithm is able to generate much better rule-sets in terms of rule length, coverage and precision. For example, our approach is able to extract rules of average length 1.85 with average precision of 0.97 and average support of over 250 while Classification Tree generated a rule-set with an average rule length of 4.56, precision of 0.88 and support of only 77. The algorithm performs in a similar way for the “Hepatitis” and “Wine” dataset too. This shows that our approach generates more significant and higher quality rules for at least three of these datasets.

5.3 Comparision with KNN and SVM

Although this work aims to improve the performance of rule based classifiers, the performance of our approach was also compared to the KNN and SVM classifiers. OrangeCanvas was used for running KNN and SVM. For KNN, k was set to 5 and neighbors were weighted by distance. The best setting for the SVM is found using the “Automatic parameteric search” option with different types of kernels. Our algorithm was run with a minSupp of 5. 10 fold cross validation was carried
out for all classifiers.

Table 5.7: Accuracy Compared in % to KNN and SVM on 6 small datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>KNN</th>
<th>SVM-Best</th>
<th>Best kernel</th>
<th>Our Algorithm</th>
<th>$t_{good}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>95.71</td>
<td>96.71</td>
<td>RBF</td>
<td>96.28</td>
<td>0.6</td>
</tr>
<tr>
<td>Hepatitis</td>
<td>80.67</td>
<td>88.38</td>
<td>Sigmoid</td>
<td>86.45</td>
<td>0.5</td>
</tr>
<tr>
<td>Glass</td>
<td>77.55</td>
<td>78.48</td>
<td>RBF</td>
<td>68.7</td>
<td>0.6</td>
</tr>
<tr>
<td>Iris</td>
<td>93.33</td>
<td>96.13</td>
<td>RBF</td>
<td>95.33</td>
<td>0.7</td>
</tr>
<tr>
<td>Pima</td>
<td>75.91</td>
<td>78</td>
<td>Linear</td>
<td>76.17</td>
<td>0.6</td>
</tr>
<tr>
<td>Wine</td>
<td>97.71</td>
<td>99.41</td>
<td>RBF</td>
<td>98.31</td>
<td>0.5</td>
</tr>
</tbody>
</table>

The results in table 5.3 show that our approach is able to attain a higher classification accuracy compared to KNN in all but one case. At the same time, its performance is close to that of the best possible SVM configuration.
Chapter 6

Conclusion

Rule based classifiers have been a topic of interest for long because of their speed of operation, good accuracy, and most importantly the ability to extract intuitive rules from the data. The decision tree based algorithms provide all these benefits but suffer from some disadvantages like not reusing training instances, overlooking significant rules because of exclusive partitioning of subsets, and generation of some low support and low confidence rules when all attributes have been examined in the greedy order. The work presented here overcomes all of these issues and also presents a function to measure rule quality which is cost sensitive for datasets with non uniform cost matrices.

The results show that this approach works better than existing single and ensemble rule based techniques on small and large datasets, especially when the minimum support for a rule is kept significantly high. Although most association rule based techniques tend to be slow as they perform a guided or semi-exhaustive search in the feature space, this work presents optimizations in the form of pruning and ordering on the working set of rules during rule generation.

Finally, it can be seen that the cost of running the classifier can be further reduced if a threshold for quality of rules is applied during rule generation. This ensures that the classifier will overlook low quality rules, focus on high confidence rules and withhold prediction on instances that would likely be misclassified by low confidence rules. By identifying instances that lie around the decision boundary of classes, the algorithm allows users to closely examine these instances as they themselves
can provide insight into what goes on in the fuzzy areas between class boundaries.

The approach described here does have some disadvantages though. Unlike decision trees, where the user generally does not specify a threshold for rule quality, this algorithm needs the user to specify some threshold. Although using this threshold has it’s advantages, finding the optimal threshold will require searching through the range or 0 to 1 with some interval size. Also, the algorithm will only find rules that are close to and above this threshold. If the entropy cut off is chosen to be 0.5, all the generated rules will have an entropy of around 0.4 and rarely anything close to 0.1, even if such rules exist. This is because once a rule passes the entropy cutoff, it is not refined anymore.

One work around to both these problems would be to iteratively run the classifier for all entropy cutoff values, starting from 0 or 0.1, to 0.9. The number of rows covered, and rules generated by each previous iteration must be carried forward so that the algorithm does not generate redundant rules. The disadvantage of doing this is that the run time of the classifier will increase significantly.

Despite this limitation, the results show that the approach described here has many advantages like a more significant rule-set and cost-sensitive rules that are necessary for datasets of all sizes and cost-sensitive datasets.
Chapter 7

Appendix

The experiments were conducted on a set of 11 datasets, however not all results were included in Chapter 5. The rest of the tables containing results from all the experiments done are placed here.
### 7.1 Results for remaining small datasets, with $\text{minSupp}$ set to 10

Table 7.1: Accuracy Compared in % to Classification Tree, CN2 and Random Forest on small datasets with min Support = 10. Continued.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classification Tree</th>
<th>CN2 Rules</th>
<th>Random Forest</th>
<th>$\ell_{\text{good}}$ cutoff</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hepatitis</td>
<td>84.54</td>
<td>80.79</td>
<td>80</td>
<td>0.1</td>
<td><strong>83.23</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>80.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>81.94</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>81.29</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>78.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>79.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td>79.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>78.06</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>78.71</td>
</tr>
<tr>
<td>Iris</td>
<td>94.67</td>
<td>94.67</td>
<td><strong>95.33</strong></td>
<td>0.1</td>
<td>92.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>93.33</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>94.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>94.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>94.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td><strong>94.67</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td>94.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>92.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>94.00</td>
</tr>
<tr>
<td>Pima</td>
<td>75.78</td>
<td><strong>76.44</strong></td>
<td>76.44</td>
<td>0.1</td>
<td>65.76</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>69.40</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>70.96</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>70.96</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>75.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>74.74</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td><strong>77.21</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>75.65</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>75.65</td>
</tr>
<tr>
<td>Wine</td>
<td>91.05</td>
<td><strong>93.86</strong></td>
<td>89.28</td>
<td>0.1</td>
<td>97.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td><strong>98.88</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td><strong>98.88</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td><strong>97.75</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td><strong>96.63</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td><strong>96.07</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td><strong>96.07</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>88.20</td>
</tr>
</tbody>
</table>
### 7.2 Results for remaining small datasets, with \( \text{min} \text{Supp} \) set to 20

Table 7.2: Accuracy Compared in % to Classification Tree, CN2 and Random Forest on small datasets with \( \text{min} \) Support = 20. Continued.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Classification Tree</th>
<th>CN2 Rules</th>
<th>Random Forest</th>
<th>( \ell_{\text{good}} ) cutoff</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hepatitis</td>
<td>79.42</td>
<td>81.25</td>
<td>79.42</td>
<td>0.1</td>
<td>79.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>79.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>79.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>79.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>79.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>79.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td>79.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>79.35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>79.35</td>
</tr>
<tr>
<td>Iris</td>
<td>94.67</td>
<td>90</td>
<td>94</td>
<td>0.1</td>
<td>94.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>94.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>94.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>94.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>94.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>94.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td>94.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>94.67</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>94.67</td>
</tr>
<tr>
<td>Pima</td>
<td>76.57</td>
<td>75.65</td>
<td>75.79</td>
<td>0.1</td>
<td>76.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>76.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>76.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>76.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>76.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>76.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td>76.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>76.43</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>76.43</td>
</tr>
<tr>
<td>Wine</td>
<td>89.31</td>
<td>88.24</td>
<td>80.39</td>
<td>0.1</td>
<td>98.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.2</td>
<td>98.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>98.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td>98.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.5</td>
<td>98.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.6</td>
<td>98.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.7</td>
<td>98.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td>98.31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
<td>98.31</td>
</tr>
</tbody>
</table>
7.3 Results for remaining large datasets, with $minSupp$ set to 50

Table 7.3: Accuracy Compared in % to Harmony (HMNY) on large datasets with min Support = 50. Continued

<table>
<thead>
<tr>
<th>Data set</th>
<th>$t_{good cut-off}$</th>
<th>Algorithm</th>
<th>HMNY</th>
<th>Algorithm</th>
<th>HMNY</th>
<th># Rules</th>
<th>Rules</th>
<th>Runtime (s)</th>
<th>Runtime (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page blocks</td>
<td>0.1</td>
<td>93.98867</td>
<td>91.6</td>
<td>33.7</td>
<td>78.1</td>
<td>0.218</td>
<td>0.092</td>
<td>0.083</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>93.97</td>
<td></td>
<td>24.7</td>
<td></td>
<td></td>
<td></td>
<td>0.083</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>93.23</td>
<td></td>
<td>15.5</td>
<td></td>
<td></td>
<td></td>
<td>0.082</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>91.02</td>
<td></td>
<td>10.9</td>
<td></td>
<td></td>
<td></td>
<td>0.083</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>91.10</td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>0.082</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>91.06</td>
<td></td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td>0.083</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>91.10</td>
<td></td>
<td>14.2</td>
<td></td>
<td></td>
<td></td>
<td>0.083</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>91.12</td>
<td></td>
<td>15.7</td>
<td></td>
<td></td>
<td></td>
<td>0.044</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>91.12</td>
<td></td>
<td>15.6</td>
<td></td>
<td></td>
<td></td>
<td>0.043</td>
<td></td>
</tr>
<tr>
<td>Waveforms</td>
<td>0.1</td>
<td>57.48</td>
<td>80.46</td>
<td>59.7</td>
<td>958.6</td>
<td>1.677</td>
<td>1.626</td>
<td></td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>69.84</td>
<td></td>
<td>177.8</td>
<td></td>
<td></td>
<td></td>
<td>1.621</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>78.72</td>
<td></td>
<td>319.1</td>
<td></td>
<td></td>
<td></td>
<td>1.786</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>81.46</td>
<td></td>
<td>427.8</td>
<td></td>
<td></td>
<td></td>
<td>1.742</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>81.7</td>
<td></td>
<td>436.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>79.72</td>
<td></td>
<td>287.3</td>
<td></td>
<td></td>
<td></td>
<td>0.729</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>72.8</td>
<td></td>
<td>114</td>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.8</td>
<td>62.44</td>
<td></td>
<td>72.1</td>
<td></td>
<td></td>
<td></td>
<td>0.215</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.9</td>
<td>63.76</td>
<td></td>
<td>47.8</td>
<td></td>
<td></td>
<td></td>
<td>0.138</td>
<td></td>
</tr>
</tbody>
</table>
References


