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

It is entitled:
Study of Effect of Coverage and Purity on Quality of Learned Rules

Student's name: Kumar Gandharva

This work and its defense approved by:

Committee chair: Raj Bhatnagar, Ph.D.

Committee member: Yizong Cheng, Ph.D.

Committee member: Carla Purdy, Ph.D.
Study of Effect of Coverage and Purity on Quality of Learned Rules

A thesis submitted to the

Graduate School

Of the University of Cincinnati

In partial fulfilment of the

Requirements for the degree of

Master of Science

In the Department of Electrical Engineering and Computing Systems

Of the College of Engineering and Applied Sciences

by

Kumar Gandharva

B.Tech. CMR Institute of Technology, India, 2012

March 2015

Thesis Advisor and Committee Chair: Dr. Raj Bhatnagar
Abstract

Rule based algorithms have emerged as a highly effective classification technique with a wide range of applications in the field of medicine, financial data analysis and business marketing to name a few. These classifiers work with real world data and are used to make predictions based on high purity rules developed using pattern mining algorithms.

There are a number of aspects that differentiate rule learning algorithms from each other. One way to differentiate them is based on re-use of training instances in rule induction. Existing rule based techniques either do not allow sharing of training instances, discovering too few patterns or allow unlimited sharing of training instances, generating an explosive number of patterns. Recent rule induction algorithms which focus on controlling instance re-use, fail to draw a relation between performance of the classifier and extent of instance re-use.

In this work, we propose a novel approach to generate high purity rules by restricting how many times an instance can be utilized while mining frequent patterns. In order to avoid generating an explosive number of rules, we introduce a parameter known as Coverage Limit, to allow control over contribution of each instance in the data towards rule generation. We study the effect of varying the Coverage Limit and Rule Purity in order to achieve best classification accuracy. In addition to this we also propose a Weighted Voting technique which allows multiple rules to collectively predict the label of an unseen instance. A detailed analysis of results on several datasets confirms that the proposed method performs better at classification than many existing techniques.
Acknowledgement

I take this opportunity to thank my advisor, Dr. Raj Bhatnagar, for his valuable guidance and feedback that has helped me complete this work successfully. I would also like to thank Dr. Carla Purdy and Dr. Yizong Cheng for taking time out of their busy schedules to be a part of my thesis committee. I extend my gratitude towards my fellow mates Sambhavi Jayaraman and Nikhil Samuel for making this experience a memorable one. Finally, I would like to thank my family for giving me endless love and support and for always having faith in me.
Table of Contents

Chapter 1: Introduction 1
  1.1 Motivation 2
    1.1.1 Control Over Instance Coverage 2
    1.1.2 Effect of Varying Rule Purity 3
    1.1.3 Classification Based on a Rule Set 3
    1.1.4 Efficient Pruning of Rule Search Space 4
    1.1.5 Handling Noise 4
    1.1.6 Handling Unclassifiable Instances 5
  1.2 Our Approach and its Benefits 5
  1.3 Overview of Other Chapters 7

Chapter 2: Related Work 9
  2.1 Related Rule Generation Methods 9
    2.1.1 C 4.5 9
    2.1.2 CBA 11
    2.1.3 CN2 12
    2.1.4 CMAR 13
    2.1.5 CPAR 15
  2.2 Novelty of Our Approach 17
Chapter 3: Rule Induction Algorithm

3.1 Important Terms

3.1.1 Attribute-Value Pair

3.1.2 Candidates and Rules

3.1.3 Entropy

3.1.4 Support and Confidence

3.1.5 COVERAGE_LIMIT

3.2 Data Discretization

3.3 Rule Generation Process

3.4 Approach

3.4.1 Rule Generation

3.4.2 Classification Using Weighted Voting

3.5 Example

3.5.1 Rule Generation

3.5.2 Applying Rule Set to Classify Instances

Chapter 4: Experimental Setup and Results

4.1 Datasets

4.1.1 Data Preprocessing

4.1.2 Random Sampling

4.1.3 Quality Metrics

4.1.4 Benchmarks for Classification
List of Figures

1.1 Rules using Coverage Limit 1 (Left) and 2 (Right) 6

3.1 Example of Equal Frequency discretization using 3 bins 21

3.2 Flowchart for Step 1 of Rule Generation 23

3.3 Flowchart for Step 2 of Rule Generation 25

3.4 Flowchart for Step 3 of Rule Generation 25

3.5 Flowchart for Step 4 of Rule Generation 26

4.1 Text Cleaning 35

4.2 Example of Equal Frequency discretization on seeds dataset 36

4.3 Effect of Coverage Limit and Acceptable Entropy on News articles dataset 42

4.4 Effect of Coverage Limit and Acceptable Entropy on Car evaluation dataset 44

4.5 Example of low support rules generated using Decision Tree 45

4.6 Effect of Coverage Limit and Acceptable Entropy on Breast cancer dataset 46

4.7 Effect of Coverage Limit and Acceptable Entropy on Mushroom dataset 47

4.8 Effect of Coverage Limit and Acceptable Entropy on LED display dataset 47

4.9 Effect of Coverage Limit and Acceptable Entropy on Seeds dataset 48
4.10 Effect of Coverage Limit and Acceptable Entropy on Voting records dataset 48

4.11 Overall Effect of Coverage Limit 49

4.12 Coverage Limit vs Database coverage 49
### List of Tables

3.1 Training Data  
3.2 Test Data  
4.1 Dataset information  
4.2 Performance benchmarks  
4.3 Comparison of performance against benchmarks  
4.4 Comparison of rules generated  
4.5 Class distribution in News articles and Car evaluation datasets  
4.6 Performance on News articles dataset  
4.7 Performance on Car evaluation dataset  
4.8 Rules generated using CN2 – low support  
4.9 News articles dataset  
4.10 Mushroom dataset  
4.11 Breast cancer dataset  
4.12 Car evaluation dataset  
4.13 LED display dataset
4.14 Voting records dataset

4.15 Seeds dataset

4.16 Comparison with CMAR and CPAR
List of Algorithms

1. Create Unit Set 26
2. Rule Generation 27
3. Classification of a new instance 29
Chapter 1

Introduction

Rule based classification is a popular methodology and a number of algorithms for inducing rules from training data have been examined and presented in literature. This field of study has been a topic of research for a number of decades and a rich set of insights into various aspects of rule learning and performance already exists. Recent contributions by researchers are helping to improve association rule mining techniques by further improvements in their performance [1]. To build a classification model, a set of training instances are provided to the algorithm. Each training instance contains feature values and a class label. Algorithms learn rules from the training data and these rules’ performance is tested against a set of test instances. Good quality rules are then used to assign class labels to unseen instances[2].

There are a number of aspects that differentiate rule learning algorithms form each other. One way to differentiate them is to group them based on the re-use of training instances for generating distinct rules during the rule generation process. Algorithms which do not support re-
use of training instances for supporting generation of multiple rules restrict each training instance to help validate only one generated rule or only one branch of a tree structured classifier. Most of the Decision-Tree induction algorithms use this type of approach. They are fast at building classifiers but can potentially miss out on important patterns in the data [3]. On the other hand, there is also a large group of algorithms which allows unlimited sharing of training instances to construct frequent itemsets. Classification based on association rules learned from frequent itemsets falls in this category. Though this group of algorithms is able to find all patterns in the data, they discover too many redundant patterns in the process [4]. In this thesis, we investigate the various options of multiple use of instance between these two extremes and seek to find out ways in which this insight can help in generating more effective rules.

1.1 Motivation

1.1.1 Control over Instance Coverage

The number of times a training instance can be used to justify creation of a rule is referred to as the “coverage” of that training instance. If an algorithm allows an instance to be used only once, we call it to be a single coverage algorithm. One of the major advantages of using association rule mining is that it lifts the single coverage constraint on training instances, and does not place any restriction on the coverage. However, since there is no constraint on instance re-use, each instance is repeatedly covered to generate an exhaustive set of frequent itemsets, and also generate an exhaustive set of rules, causing a high amount of redundancy [5]. To improve the computational complexity and reduce the size of rule set generated by this method, there is a
need to allow control over the extent of coverage of instances during the rule induction. In this way, algorithms can utilize the benefits of multiple instance coverage, and can also maintain a reasonable computational complexity. Some recently presented algorithms that allow such multiple coverage of instances, do not provide an insight into relation between different database coverage levels and accuracy of the classifier [6] [7].

1.1.2 Effect of Varying Rule Purity

The performance of a classifier is based on a number of factors relating to the training dataset. If the data is very consistent, it may be beneficial to accept only high purity rules. Purity refers to the percentage of instances covered by the rule that belong to the majority class for the rule. However, as the noise in the dataset increases, it becomes difficult to find pure rules [3]. In order to accommodate noise, the rule generation algorithm should relax the purity threshold for accepting rules to build a classifier. Considering this, it is important for an algorithm to allow its users control over the purity threshold. We not only aim to accommodate this, but also study the effect of varying rule purity on the constructed rules.

1.1.3 Classification Based on a Rule Set

When redundancy is permitted in a rule learning process, we may generate a large number of rules, some of which may be duplicates, and in presence of noise they may also be contradictory. That is, it is possible that for a given instance, two rules may announce different classes for the instance. Many classification methodologies rely on selecting the best applicable rule to perform the task [1]. In an application where rule redundancy exists and classification accuracy is important, relying on a single rule for classification should be avoided.
Ensemble methods such as Random forests and Bagging use multiple learning algorithms to build a classifier and assign a class label to a new instance by selecting the most frequently predicted label as the final prediction [8][9]. This approach has been proven to achieve better classification accuracy than any of the constituent learning algorithms. Inspired by this, in our work, we aim to improve performance of the classifier by introducing an effective way of selecting a group of rules to collectively classify a new instance.

1.1.4 Efficient Pruning of Rule Search Space

Frequent pattern mining techniques generate a large number of frequent itemsets. Once an itemset is assigned a class label and a rule is generated from it, further specialization of the same itemset is continued in search of better quality rules. This leads to redundant rules being added to the rule set. To avoid redundancy, some popular rule generation algorithms first discover all rules and then select the best ones among the overlapping rules to build the classifier [1]. However, to further improve the efficiency of the rule generation process, discovery of redundant rules should be prevented by eliminating further specializations of those itemsets which have already been used to generate good quality rules.

1.1.5 Handling Noise

Decision trees cover every single training instance to build a classifier. However, fitting the training data completely leads to too specific rules which may often have very low correlation between attributes present in the rule. In order to handle real world datasets, which have high probability of noise, classification algorithms should lift the constraint of covering the training
data completely. Allowing some training instances to be left uncovered will ensure that the classifier is built based only on highly consistent set of instances present in the data.

1.1.6 Handling Unclassifiable Instances

Most classification algorithms assign a class label to every unseen instance even when no good quality rule is applicable [10]. Often, this is done by using a default rule that assigns majority class label to a test instance if no existing rule is able to classify it. However, this approach may lead to high misclassification rate and is unreliable for critical applications such as in medical domains. Hence, there is a need for an alternative to handle instances which cannot be classified with confidence, without incurring the cost of misclassification associated with such predictions.

1.2 Our Approach and its Benefits

In order to accomplish the goals mentioned earlier, we adopt the following approach,

- Instead of completely lifting the single coverage constraint, we propose using a coverage limit to control the number of times a training instance can be re-used for rule generation. Increasing the coverage limit will allow a larger number of rules to be generated. This enables the algorithm to re-use the consistent instances in the training data multiple times to extract more rules rather than discarding these instances immediately after they contribute towards generating one rule. The idea behind using a fixed coverage limit is to restrict the number of rules generated, and thereby, put a check
on the computational complexity of the approach. Studying the effect of varying coverage limit, also reveals its impact on the performance metrics of the classifier. Figure 1.1 compares rules generated using coverage limit 1 with those generated using a coverage limit of 2. The columns in each diagram represent the attributes in the dataset, and the rows represent instances. The colored rectangles denote rules. As seen in the figure, Rule R1 contains the attributes Att1, Att2 and Att3 and is justified by the instances 1, 2 and 3. Also, when coverage limit is increased (diagram on the right in Figure 1.1), instances are shared to justify multiple rules.

Figure 1.1 Rules using Coverage Limit 1 (Left) and 2 (Right)

- To control the purity of accepted rules, we use a parameter known as Acceptable Entropy. A very low entropy threshold allows our algorithm to mine only high purity rules. This value can be increased based on the nature of a dataset. We study the effect of varying this threshold to understand how it impacts the rule generation process.
- Class labels are predicted based on all rules applicable to a new instance rather than on the decision announced by the selected best rule. The rules are assigned weights based
on their quality and instances are classified using a weighted voting method to ensure that the prediction is not biased.

- In order to eliminate redundancy, we impose an effective pruning strategy where we remove frequent itemsets from the candidate pool once they are used to generate a rule of acceptable quality. This ensures that itemsets are not specialized any further once they have been sufficiently utilized in rule induction.

- In our approach, we ease the database coverage constraint seen in decision trees. The entire database need not be covered to complete the rule generation process, thereby allowing our model to accommodate noise. We stop the rule generation process, when maximum possible database coverage is achieved i.e. no further good quality rules can be generated without increasing the coverage limit for instance.

- Most existing algorithms predict a class label for every new instance, even if the classification is based on poor quality rules. We propose that class instances should be left undecided when no high quality rule is applicable to them. Due to this, our model will always have very high precision and can be held accountable for its predictions. This may be a desired quality in applications such as medical diagnosis. Further, it allows the users to depend on other options such as ensemble classifiers to predict labels for undecided instances.

1.3 Overview of Other Chapters

After testing our approach on a number of datasets, we confirmed that our model’s performance metrics are higher than existing rule based classification algorithms, at least for
some of the datasets. Our algorithm also does well to meet the objectives mentioned earlier. Importantly, we introduce coverage limit as a measure to control the database coverage. Using this parameter we allow restricted re-use of consistent training instances to generate multiple high quality rules instead of overfitting rules to outlier instances using single coverage. In chapter 2, we discuss the traditional methods used in rule based classification, followed by a study of related work in the direction of our research. In chapter 3 and 4 we present our algorithm, justify our design choices and analyze the effect of coverage limit and rule purity on the classifier’s accuracy on a number of datasets. We conclude our work in chapter 5 with a summary of our observations and discuss the possible future work in the area.
Chapter 2

Related Work

Over the past two decades, there has been a lot of work in the area of rule based classifiers. Recent contributions by researchers are helping to improve association rule mining techniques by further enhancements in their performance [1]. In this section we review some of the improvements in these algorithms towards eliminating the drawbacks discussed in Chapter 1.

2.1 Related Rule Generation Methods

2.1.1 C4.5

Developed by Ross Quinlan, the C4.5 algorithm [3] is used to generate decision trees for classification. It was proposed as an improvement of Quinlan’s earlier ID3 algorithm [11]. Both C4.5 and ID3 algorithms take training data as input and build a decision tree by successively choosing the best available attribute to split the dataset based on Information gain. Each node in
the tree represents an attribute from the test instance and each branch represents a possible value or set of values of the attribute. The leaf nodes represent target classes. In order to classify an unseen instance, its attribute-value pairs are compared to those in the decision tree, starting at the root node. When a leaf node is reached, the class value is assigned to the instance [3] [12].

Similar to our approach, C4.5 is capable of handling noisy data. A technique known as pre-pruning is used to cut the growth of a branch when its information is unreliable. In addition to information gain, the chi-squared test [3] is used to identify if a branch has significant associations and should be pursued. C4.5 also employs post-pruning. Here, a fully grown decision tree is trimmed to remove branches that fit data too specifically. Quinlan’s algorithm is similar to our method in this aspect as well because it lifts the constraint of fitting the training data perfectly.

Some other improvements seen in the C4.5 include handling continuous data, training data with missing attributes and attributes with differing costs [3]. C5, a later version of the algorithm [13], also facilitates boosting, which is an example of a technique that combines weak learners with poor classification accuracy to form stronger ones with very high accuracy.

Despite its significant contribution towards improving decision tree algorithms, C 4.5 continues to face the following limitations,

- It adopts the single coverage approach as each instance in the training data is used at most once. Useful instances present in the data are not utilized enough in the rule generation process. We solve this by using a multiple coverage strategy which is further explained in Chapter 3.
• Classification is performed using only one branch of the tree, therefore the risk of misclassification is high. On the other hand, the proposed method performs classification based on multiple rules.

2.1.2 CBA

Classification based on association rules is an approach introduced in the paper presented by Bing Liu [5]. The main focus of this paper is to make association rule mining techniques available to data classification. Introduced by Agarwal et al [14], an Association rule is formally defined as follows.

Let \( I = \{i_1, i_2, \ldots, i_n\} \) be a set of \( n \) binary attributes called items. Let \( D = \{t_1, t_2, \ldots, t_m\} \) be a set of transactions called the database. Each transaction in \( D \) has a unique transaction ID and contains a subset of the items in \( I \). A rule is defined as an implication of the form \( X \Rightarrow Y \) where \( X, Y \subseteq I \) and \( X \cap Y = \emptyset \). The sets of items \( X \) and \( Y \) are called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule respectively [14].

The datasets used in classification have a large number of associations between attributes and target classes. To utilize these associations, the CBA generates Class Association Rules (CAR) in order to classify unseen instances. CAR’s are rules with target class attribute as consequent. In order to generate CAR’s, the algorithm adopts the Apriori approach. Similar to frequent itemsets discovered in the Apriori approach, this algorithm mines rule items satisfying a support threshold, \( \text{minSupp} \). A rule item is defined as a combination of a condition set and a class label.
Unlike our approach, CBA lifts the coverage constraint by allowing unlimited re-use of instance to mine rules.

Once rules are discovered, they are subjected to pruning using pessimistic error rate [5]. CBA selects the best set of rules to build a classifier. The best set of rules is found by sorting all rules by decreasing confidence. If rules have same confidence value, they are sorted by decreasing support. In cases where support and confidence are the same, rules are placed in the order in which they are generated. An unseen instance is classified using the first rule from the sorted set of rules that is applicable to it [5]. Once again this is in contrast with our method which is based on using all applicable rules.

CBA shows a better classification accuracy then many algorithms [5]. An improvement of CBA, msCBA was introduced by Bing Liu et al [5]. The new method uses different support threshold values for each class based on the class distribution. Though this addition improves the accuracy of the classifier, both methods are computationally intensive as they generate a very large number of rules.

2.1.3 CN2

The CN2 algorithm was introduced by Clark, P. and Niblett, T in 1989 [15]. This algorithm is based on Quinlan’s ID3 [11] and Michalski’s AQ algorithm [16] and was designed to overcome their drawbacks namely handling noisy data and increasing the space of rules searched. The CN2 proposes improvements in two aspects, rule generation and classification [15].

The rule generation process of CN2 algorithm is similar to our algorithm in more than one way. Like our method, the search space is expanded by also searching for complexes (complex is a
term used to represent a conjunction of attribute-value pairs from dataset) that do not perform perfectly on the training data. The algorithm also adopts a top down approach similar to decision tree to search for rules, halting specialization when no further associations can be derived. However, the difference is that CN2 allows single coverage of training instances. This is because, once a complex is accepted as a rule, instances that agree with the complex are removed from the data.

The classifier is a group of if-then rules arranged in decreasing order of rule confidence. The last rule is known as a default rule, which assigns the most frequently occurring class label to a new instance. To classify an instance all rules are inspected in the order they are stored and the first rule that is applicable is used to assign class label. If no rules apply to the new instance, the default rule is selected [17]. This is in contrast to our objective, because we leave an instance undecided if we are unable to assign a class label to them with high confidence.

Despite the pitfalls, CN2 performs better than many other rule generation algorithms [15]. It is capable of finding a minimalistic set of rules from a given dataset which classify unseen instances with high accuracy.

2.1.4 CMAR

All the algorithms discussed until this point use a single rule to classify a test instance. Also, an explosive number of frequent patterns are considered in order to generate rules. The CMAR (Classification based on multiple class-association rules) is an algorithm proposed by Li et al (2001) [6]. It is the closest in spirit to our algorithm. Similar to our approach, instead of depending on a single rule for classification, a set of rules is used. The algorithm choses a group
of closely related high confidence rules and measures the correlation between them. A novel
technique, weighted chi-squared measure [6] is used to determine the goodness of a rule,
considering both conditional support and class distribution. To classify an instance, all rules
applicable to the instance are extracted. These rules are grouped by target category and the
group with the highest chi-square metric is selected to assign a class to the new instance.

FP-growth method [18] is used as an alternative to Apriori based approaches in the construction
of rule set. A dedicated candidate generation process to generate rules is avoided using FP-tree
resulting in much faster computation time. Further, pruning is performed to improve the rule
generation process. CMAR proposes three types of pruning. Firstly, it prunes all rules which are
subsets of already accepted rules. No rule which is a further specification of an existing rule is
accepted if it has lower confidence than that already achieved. Secondly, only rules that have a
positive correlation between their attributes and target labels are accepted, thereby reducing
the effect of noisy data. The last step is to allow every instance of the training data to be covered
\( k \) number of times before it is removed from the database, where \( k \) is known as the coverage
threshold. This allows multiple coverage of training instances unlike the algorithms discussed
earlier, where each tuple of the database is immediately discarded after being covered by a rule
[6].

CMAR also proposes improvements in storage and retrieval of rules using a CR-Tree. This prefix-
tree structure stores each rule as a branch in the tree. Long rules share a branch until they have
common attributes and split into new branches when their attributes vary towards the bottom
of the tree. The leaf nodes contain the class label, support and confidence of the rule. This
storage technique allows CMAR to store all rules generated using a small number of nodes.
This algorithm does well to provide solutions for problems commonly faced by rule generation algorithms. It addresses scalability, computational time, storage and retrieval of rules and coverage. It also uses multiple rules to predict class labels resulting in better accuracy. CMAR is a novel and effective approach, however like other algorithms has some drawbacks. It introduces coverage threshold as a way to control the database coverage, however does not provide clear insight into its effect on accuracy. There is no fixed method to obtain highest accuracy as the peak of accuracy for different datasets is attained at different coverage and confidence thresholds [6].

2.1.5 CPAR

Xiaoxin et al. [7] propose Classification Based on Predictive Association Rules (CPAR). This algorithm is based on FOIL (First order inductive learner), developed by Ross Quinlan in 1993 [19]. CPAR is a combination of greedy approach and exhaustive search and has the advantages of both.

FOIL is a greedy algorithm that repeatedly searches the database to find the current best rule. Once a rule is found it deletes all instances to which the rule applies. CPAR adopts this approach of FOIL, however once the current best rule is found, it decreases the weight of the instances to which it applies. Due to this reason, it overcomes one of the major drawbacks of FOIL, which is too few rules. Similar to FOIL, for multi class datasets, rules are generated for each class and then combined to form the final rule set.

CPAR uses multiple rules to classify an instance. All rules applicable to an instance are extracted and divided into groups based on the class label assigned. Within each group, rules are ordered
by decreasing expected accuracy. Expected accuracy of a rule is calculated using Laplace error estimate [7], calculated as follows,

\[
\text{Laplace Accuracy} = \frac{(n_c+1)}{(n_{tot} + k)},
\]

Where \(n_c\) = total number of instances that satisfy rule’s body and belong to class c.

\(n_{tot}\) = total number of instances that satisfy rule’s body

\(k\) = Number of classes

Next, the algorithm selects the top \(N\) rules of each class (\(N\) is a user defined parameter) and calculates the average expected accuracy. The group with best accuracy is chosen as the class label to be assigned [7].

Performance of CPAR is comparable to that of CMAR, and is significantly higher than CBA and decision trees [7]. The number of rules generated is much smaller than most association rule based algorithms. There is negligible redundancy in rules generated, making it more efficient than its competitors. Despite an efficient coverage of database, CPAR’s classification accuracy may be less. Calculating average expected accuracy to select class label may be biased, as it favors minority classes. In contrast, our approach addresses this limitation by introducing a weighted voting method which prevents biased predictions.
2.2 Novelty of Our Approach

Rule based algorithms have drawn constant attention since their inception and significant work has been done towards their improvement. Though some of the algorithms discussed in this section were successful in enhancing the performance of existing methods, there is no significant work towards controlling instance coverage. Also, none of the methods lay emphasis on studying the effect of rule purity on the final set of rules. Recent work introduces novel ways to classify instances using multiple rules [6] [7]. However, when no rule is applicable to an instance, these methods fail to leave instances undecided. In our research we lay emphasis on these limitations.

In the approach presented ahead, we introduce a new rule based algorithm which allows users to control the extent of instance re-use and the purity threshold of the accepted rules. We study the effect of varying these parameters to draw an inference of their relation with the model’s accuracy. In addition to this, we also accommodate design changes in the rule generation and classification process to overcome the drawbacks listed in Chapter 1.
Chapter 3

Rule Induction Algorithm

In this chapter, we introduce our rule induction algorithm which allows control over instance coverage and rule purity. We provide a detailed explanation of our rule generation approach and discuss the weighted voting technique used to assign a class label to an instance. We also justify our algorithm's design and provide an example to illustrate how it works.

3.1 Important Terms

In order to explain the rule generation process, it is important to first understand a few key terms. In this section we provide some basic definitions to explain our approach.

3.1.1 Attribute-Value Pair

In a given dataset, the columns are known as attributes or features. An attribute-value pair is a unique combination of an attribute and its value. Each attribute may have a different number of
unique values. For example if an attribute \( A1 \) has unique values \( v_1, v_2, v_3 \) and \( v_4 \), then the possible attribute value pairs for attribute \( A1 \) are \( \{A1, v_1\}, \{A1, v_2\}, \{A1, v_3\} \) and \( \{A1, v_4\} \).

3.1.2 Candidates and Rules:

**Candidate:** One or more attribute value pairs joined with a logical AND is known as a candidate. A candidate may consist of a single attribute or multiple attributes.

If \( A1, A2, \) and \( A3 \) are attributes and \( v_1, v_2 \) and \( v_3 \) are their corresponding values then \( \{A1, v_1\} AND \{A2, v_2\} AND \{A3, v_3\} \) is an example of a candidate. A single pair \( \{A1, v_1\} \) is also a valid example of a candidate.

**Rule:** An association rule is a relation between a candidate and a target category. A rule has two parts, an antecedent (if) and a consequent (then). In this case, the antecedent is a candidate and the consequent is a class label or target category that is found in combination with the antecedent. An association rule can be represented as,

\[
\{\text{Candidate}\} \rightarrow \text{Class Label}
\]

For example, if the dataset contains two class values, 0 and 1, then the rule \( \{A1, v_1\} AND \{A3, v_3\} \rightarrow 0 \) implies that in a new test instance, if value of attribute \( A1 \) is \( v_1 \) and value of attribute \( A3 \) is \( v_3 \) then the label Class 0 can be assigned to that instance.

3.1.3 Entropy:

Entropy is a measure of the purity of a candidate. It is calculated using the formula,

\[
\text{Entropy} = -\sum p_i \log_2 (p_i)
\]
Where, \( p_i \) is the probability that a candidate corresponds to class \( i \). \( P_i \) is calculated using the formula, \( N_i / N \). Here, \( N \) is the total number of instances where the candidate holds true and \( N_i \) is the number of instances in \( N \) which belong to class \( i \).

For a two class dataset, entropy ranges from 0 to 1, 0 being the purest and 1 being the worst purity. If a candidate has entropy 0, it means that all instances that the candidate is applicable to, belong to the same class. For datasets with more than two class labels, the entropy is normalized to a scale of 0 to 1 in order to maintain consistency.

**ACCEPTABLE_ENTROPY**: It is a parameter used to specify the rule purity threshold. It is defined as the maximum acceptable value of entropy for which a candidate can be selected in the final set of rules.

**ENTROPY_UPPER_THRESHOLD**: It is a constant specifying the maximum value of entropy that is allowed for a candidate to be re-considered at later stages of the algorithm for creating its more specific versions.

### 3.1.4 Support and Confidence:

Support of a candidate is the measure of the number of instances that satisfy the candidate. An instance is said to satisfy a candidate if, for all attributes present in the candidate, the values of the attributes are same as corresponding values seen in the instance. The minimum value of support in order to accept a candidate into the final set of rules is given by \( \text{MIN_SUPPORT} \).

Confidence of a rule \( X \rightarrow Y \) is said to be \( c \), if \( c\% \) of the transactions in the dataset that contain \( X \), also contain \( Y \). Confidence is given by the formula,
Confidence($X \rightarrow Y$) = Support($X \cup Y$) / Support($X$)

In our approach, we use confidence as a measure of rule purity.

3.1.5 \textit{COVERAGE\_LIMIT}

\textit{COVERAGE\_LIMIT} is a parameter used to specify the number of times each instance can be reused in the rule generation process. If \textit{COVERAGE\_LIMIT} is set to a constant $k$, an instance is discarded from the dataset once it has been covered $k$ times.

3.2 Data Discretization

Besides categorical values, our algorithm is capable of handling continuous data. However, in order to do this, some data preprocessing is required. Continuous attributes are subjected to Equal-Frequency discretization [20] with 5 bins. Using this method of discretization divides attributes into specified number partitions such that each partition has the same frequency.

Figure 3.1 illustrates Equal frequency discretization. Here, the \textit{Attr1} is divided into three bins with frequency of three each.

\textit{Figure 3.1 Example of Equal frequency discretization using 3 bins}
Dividing continuous attributes into too few bins may result in loss of important patterns present in the data. For this reason, we use five bins in our algorithm. Using five bins ensures that we consider a larger number of attribute value combinations when searching for frequent patterns.

The reason for selecting this method of discretization instead of Entropy based MDL method as seen in Fayyad and Irani’s paper [21] is that the latter does not give control over the number of bins in each attribute.

3.3 Rule Generation Process

In order to generate association rules, a key step is to identify the frequent itemsets. The Apriori algorithm is an efficient algorithm to find optimal frequent itemsets. It uses a "bottom up" approach in a breadth-first manner, identifying frequent subsets one at a time. This process is known as candidate generation. Candidate itemsets of length \( k \) are generated from itemsets of length \( k-1 \). A minimum support threshold is defined at the start of the process and candidates are pruned if they have support below the threshold. The algorithm terminates when no further candidates are found [14].

Our algorithm uses a method similar to Apriori to discover frequent item sets. In addition to minimum support, we use an entropy threshold in order to accept a candidate. The entropy allows us to control the purity of the rules. To avoid generating redundant itemsets, we prune a candidate if it is accepted into the final set of rules. This ensures that no subsets of existing rules are accepted. To generate candidates of size \( k \), we consider all combinations of candidates of
size $k-1$ which had sufficient support but did not make the entropy cut off. The candidates that satisfy entropy and support thresholds are accepted as valid rules and same process is repeated for candidates of next higher size.

Apart from entropy and support, coverage also plays an important factor in accepting a candidate. A predefined coverage limit indicates how many times a training instance is allowed coverage. An instance is removed from the training data once it reaches this coverage threshold.

3.4 Approach

3.4.1 Rule Generation

The following steps are involved in the rule generation process,

**Step 1:** The rule generation process starts by first creating a *Unit Set* containing all candidates of size 1. The *Unit Set* is sorted by increasing entropy values. Candidates having same entropy values are sorted internally by decreasing support values. Sorting enables us to inspect high quality candidates before considering poor ones. A copy of these candidates is stored in a *Candidate Bag*.

*Figure 3.2 Flowchart for Step 1 of Rule Generation*
Step 2: The Candidate Bag is inspected one candidate at a time and all candidates with either too high entropy (Entropy greater than ENTROPY_UPPER_THRESHOLD) or too low support (Support lesser than MIN_SUPPORT) are removed from the Candidate Bag. A Coverage table is maintained to keep track of how many times each instance has been covered. A candidate is accepted if it has entropy less than the ACCEPTABLE_ENTROPY. Such a candidate is then assigned a label to make it a rule. To assign a label to a candidate, the majority class label of supporting instances is chosen. The selected candidate is removed from the Candidate Bag and the generated rule is accepted into the final set of rules. Deleting a candidate from the Candidate Bag once it is accepted ensures that no further specializations of this candidate are generated, thereby avoiding redundant rules.

Once a candidate is accepted and a rule is created from it, the coverage table is updated. This is done by inspecting the instances where the rule’s LHS (candidate) holds true and then incrementing the number of times those instances have been covered in the coverage table mentioned earlier.

A constant called COVERAGE_LIMIT puts a restriction on the number of times an instance is allowed to contribute toward rule generation. This is done by discarding an instance from the database if it has been covered ‘k’ number of times. Here ‘k’ is given by the COVERAGE_LIMIT. When k value is one, the algorithm behaves as a single coverage model similar to Decision Trees.
Step 3: Once the Candidate Bag is exhausted, it only contains those candidates that have a support greater than \textit{MIN\_SUPPORT} and entropy between \textit{ACCEPTABLE\_ENTROPY} and \textit{ENTROPY\_UPPER\_THRESHOLD}. The Unit Set is pruned so that it only contains attributes that appear in the Candidate Bag. In this manner, candidates containing infrequent attributes are not explored. From the Candidate Bag we generate new candidates of the next higher size. To do this, all unique combinations of candidates in the Candidate Bag and Unit Set are generated. The contents of the Candidate Bag are replaced with these new candidates. The Candidate Bag is sorted in the same manner as the Unit Set above.

Figure 3.4 Flowchart for Step 3 of Rule Generation
**Step 4:** All above steps are repeated with the new *Candidate Bag*. This process is repeated until either the *Candidate Bag* is empty or all instances in the training data have been covered $k$ number of times ($k$ is *COVERAGE_LIMIT*). At the end of the process, the final *Rules* are extracted.

Running the algorithm for different *COVERAGE_LIMIT* and *ACCEPTABLE_ENTROPY* has an effect on the rules generated.

![Flowchart for Step 4 of Rule Generation](image)

*Figure 3.5 Flowchart for Step 4 of Rule Generation*

Algorithm 1 shows pseudo code for creating a *Unit Set* and Algorithm 2 shows pseudo code for Rule generation process.

```plaintext
For each Feature f in Data
    For each distinct Value v in f
        UnitSet ← UnitSet U {f, v}
    End
End
```

*Algorithm 1 Create Unit Set*
1) Create UnitSet
   Sort (UnitSet)
   CandidateBag <- UnitSet

2) For each candidate cand in CandidateBag
   If Support (cand) < MIN_SUPPORT
     CandidateBag. Remove (cand)
     Continue
   End if

   If Entropy (cand) > ENTROPY_UPPER_THRESHOLD
     CandidateBag. Remove (cand)
     Continue
   End if

   If Entropy (cand) <= ACCEPTABLE_ENTROPY
     rule <- AssignLabel (cand)
     FinalRules.Add (rule)
     CandidateBag. Remove (cand)
     UpdateCoverage (cand)
   End if

End

3) Prune(UnitSet)
   CandidateBag <- GenerateCandidates (CandidateBag, UnitSet)
   Sort (CandidateBag)

4) Repeat step 2 to 3 until CandidateBag is Empty

Algorithm 2 Rule generation
3.4.2 Classification Using Weighted Voting

Once the final set of rules is obtained, the performance of the classifier is tested by apply these rules to a test set. If only one rule is applicable to a test instance, we assign the class label suggested by the rule. However, if multiple rules apply to the same test instance and the rules have different consequents then a conflict arises. A common approach is to assign the majority label to the test instance. However this is not always accurate as some of those rules may have very high support and confidence, whereas the others may be just enough to be accepted as a rule. Treating all rules equally may thus give inaccurate results. In order to resolve this, we introduce the concept of weighted voting.

We assign each rule a weight, which is equal to its lift, a measure of importance of a rule. The lift of a rule $X \rightarrow Y$ is calculated using the formula,

$$\text{Lift} (X \rightarrow Y) = \frac{\text{Confidence} (X \rightarrow Y)}{\text{Probability} (Y)}$$

If the lift of a rule is 1, it means that the probability of occurrence of the antecedent and that of the consequent are independent of each other. Since the two events are independent of each other, they should not be used to draw any rules.

When the lift is > 1, it denotes the degree to which those two occurrences are dependent on one another. Higher the value of lift, greater is the importance of the rule in accurately predicting labels of test instances.

When the number of rules is large and all applicable rules are collectively used to predict the label of a test instance, a conflict of rules is imminent. To resolve this, we group the rules by their
consequents and calculate the sum of lifts of all rules for each group. The group of rules with the highest sum is selected in order to assign a label to the instance in question. Algorithm 3.1 shows a pseudo code for the classification method we use.

```
ClassLabel Classify (TestInstance testInst)
    ApplicableRules ← φ
    For each Rule r in finalRules
        If r applies to testInst
            ApplicableRules ← ApplicableRules ∪ r
    End if
    End
    For each Rule r in ApplicableRules
        C ← ClassLabel(r)
        WeightC ← WeightC ∪ Lift(r)
    End
    maxWeight ← 0
    classLabel ← φ
    For each distinct ClassLabel C in ApplicableRules
        If WeightC > maxWeight
            classLabel ← C
            maxWeight ← WeightC
        End if
    End
    Return classLabel
```

Algorithm 3 Classification of a new instance
3.5 Example

In order to explain rule generation and classification, consider the following example,

Table 3.1 below consists of attributes A to F and Tuples T1 to T10. There are two class values X and Y. The Support threshold is set to 40%. Acceptable Entropy is set to 0.1 and Entropy Upper Threshold is set to 0.95. The Coverage Limit is 3.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>T2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>T3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>T4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>T5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>T6</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Y</td>
</tr>
<tr>
<td>T7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>T8</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Y</td>
</tr>
<tr>
<td>T9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>T10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Y</td>
</tr>
</tbody>
</table>

*Table 3.1 Training Data*

Using the given data, we run through the steps to classify the following 2 instances,

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>T11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>?</td>
</tr>
<tr>
<td>T12</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>?</td>
</tr>
</tbody>
</table>

*Table 3.2 Test Data*
Note: Since the data above is binary, candidates need not be stored as attribute-value pairs. The name of the attribute is sufficient.

### 3.5.1 Rule Generation

**Iteration 1**

Step 1: Unit Set is \{A, B, C, D, E, F\}

After sorting by entropy and support, Unit Set = \{F, A, E, D, B, C\}

Candidate Bag = \{F, A, E, D, B, C\}

Step 2  F is accepted to generate a rule. (Satisfies support threshold (4), Entropy threshold (0.1))

Final Rules = \{F-> Y\} (Assigned Majority class Label)

Candidate Bag = \{A, E, D, B, C\}

Coverage Table = \{T1:0, T2:1, T3:0, T4:1, T5:0, T6:1, T7:0, T8:1, T9:0, T10:1\}

A is accepted to generate a rule

Final Rules = \{F-> Y, A-> X\} (Assigned Majority class Label)

Candidate Bag = \{E, D, B, C\}

Coverage Table = \{T1:1, T2:1, T3:0, T4:1, T5:1, T6:1, T7:1, T8:1, T9:1, T10:1\}

E is accepted to generate a rule

Final Rules = \{F-> Y, A-> X, E->X\} (Assigned Majority class Label)

Candidate Bag = \{D, B, C\}
Coverage Table = \{T1:1, T2:1, T3:0, T4:1, T5:2, T6:1, T7:2, T8:1, T9:2, T10:1\}

D, B and C remain in the Candidate Bag as they have enough support, have entropy below the Entropy upper threshold and cover a new tuple T3.

Step 3  Candidate Bag = \{DB, DC, BC\}

After sorting by entropy and support, Candidate Bag = \{DC, DB, BC\}

Step 4  Go to Step 2

Iteration 2

Step 2  DC is accepted for rule generation.

Final Rules = \{F \rightarrow Y, A \rightarrow X, E \rightarrow X, DC \rightarrow X\}

Coverage Table = \{T1:1, T2:1, T3:1, T4:1, T5:3, T6:1, T7:3, T8:1, T9:3, T10:1\}

Since T5, T7 and T9 have reached coverage limit (3), they are removed from data.

DB and BC are discarded due to too low support.

Step 3  Candidate Bag = \emptyset

Step 4  END

Resultant rules are \{F \rightarrow Y, A \rightarrow X, E \rightarrow X, DC \rightarrow X\}
The result of the algorithm is a set of high purity rules. Number of rules may vary for different values of Coverage Limit.

3.5.2 Applying Rule Set to Classify Instances

Classify T11

The following rules are applicable to T11: F -> Y, A -> X, E->X

We divide the rules into two groups,


Lift of F-> Y = 2, Lift of A->X = 1.6, Lift of E->X = 1.2

Sum of weights in Group 2 > Sum of weights in Group 1

Assigned label = X

Classifying T12

One rule is applicable, F->Y.

Assigned label = Y
Chapter 4

Experimental Setup and Results

4.1 Datasets

Six datasets were selected from UCI’s machine learning data repository [22] to test our algorithm’s performance. An additional text news articles dataset was created from websites of NYTimes, Chicago Tribune, Washington Post, BBC, Reuters and Guardian. Information about all the datasets is in Table 4.1.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># of instances</th>
<th># of attributes</th>
<th>Attribute types</th>
<th># of class values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>699</td>
<td>9</td>
<td>Integers</td>
<td>2</td>
</tr>
<tr>
<td>Car Evaluation</td>
<td>1728</td>
<td>6</td>
<td>Categorical</td>
<td>4</td>
</tr>
<tr>
<td>LED Display</td>
<td>5000</td>
<td>24</td>
<td>Binary</td>
<td>10</td>
</tr>
<tr>
<td>Mushroom</td>
<td>8123</td>
<td>22</td>
<td>Categorical</td>
<td>2</td>
</tr>
<tr>
<td>News Articles</td>
<td>500</td>
<td>1300</td>
<td>Real</td>
<td>5</td>
</tr>
<tr>
<td>Seeds</td>
<td>210</td>
<td>7</td>
<td>Real</td>
<td>3</td>
</tr>
<tr>
<td>Voting records</td>
<td>435</td>
<td>16</td>
<td>Categorical</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 4.1 Dataset information
4.1.1 Data Preprocessing

In order to classify data with continuous attributes, some data pre-processing is required. The seeds and breast cancer dataset are directly discretized using Equal frequency discretization [20] using five bins. The news articles dataset consists of 500 text files. These text files were cleaned using the Text mining package in R version 3.1.1. The steps involve converting to lower case, removal of numbers, stop words [23] and punctuation, stripping white spaces and stemming the documents using Porter stemmer [24]. The files are then converted into a term document matrix with words as attributes. Each row corresponds to a document. Each row of the Term-Document matrix is normalized to 1000. This ensures that the values corresponding to each document have the same scale despite the difference in size of the articles. Finally, Equal frequency discretization is applied on the normalized term document matrix. Figure 4.1 shows the data pre-processing steps used for the news articles dataset.

![Diagram: Text cleaning process]

*Figure 4.1 Text cleaning*
4.1.2 Random Sampling

The training set is obtained by randomly selecting 70% tuples from the dataset. The remaining 30% are treated as test set. This process is repeated five times. The performance of a classifier is measured over all five training–test combinations and averaged to represent the final result.

4.1.3 Quality Metrics

We use the following metrics to measure the goodness of a classifier,

**Precision:** It is defined as the fraction of retrieved instances that are relevant. In our scenario it is calculated as the ratio of the number of instances correctly predicted as Class $k$ to the total number of instances predicted as Class $k$.

$$\text{precision}_k = \frac{\# \text{ of instances correctly predicted as Class } k}{\# \text{ of instances predicted as Class } k}$$

**Recall:** It is defined as the fraction of relevant instances that are retrieved. It is calculated as the ratio of the number of instances correctly predicted as Class $k$ to the actual number of instances belonging to Class $k$. 

![Figure 4. 2 Example of Equal frequency discretization on Seeds Dataset with 5 bins](image-url)
Recall_k = (# of instances correctly predicted as Class k) / (# of instances of Class k)

**F-measure:** It is the harmonic mean of precision and recall. Since F-measure combines both these metrics, it can be used as a representative measure of goodness of a classifier.

$$F_{measure_k} = \frac{(2 \times precision_k \times recall_k)}{(precision_k + recall_k)}$$

In a multi class problem, precision, recall and F-measure are calculated for each class and averaged to represent final result.

### 4.1.4 Benchmarks for Classification

To explain the performance of existing classification techniques on the selected datasets, Table 4.2 shows the precision, recall and F-measure of C4.5 Decision tree and CN2 algorithms. The algorithms were implemented using Orange Canvas tool kit version 2.6.1. Orange canvas is an open source software for explorative data analysis and visualization.

<table>
<thead>
<tr>
<th>Dataset name</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>Precision</th>
<th>Recall</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>0.98</td>
<td>1</td>
<td>0.98</td>
<td>0.96</td>
<td>0.98</td>
<td>0.97</td>
</tr>
<tr>
<td>Car Evaluation</td>
<td>0.85</td>
<td>0.77</td>
<td>0.81</td>
<td>0.83</td>
<td>0.85</td>
<td>0.84</td>
</tr>
<tr>
<td>LED Display</td>
<td>0.78</td>
<td>0.91</td>
<td>0.84</td>
<td>0.81</td>
<td>0.91</td>
<td>0.86</td>
</tr>
<tr>
<td>Mushroom</td>
<td>1</td>
<td>0.42</td>
<td>0.59</td>
<td>0.99</td>
<td>1</td>
<td>0.99</td>
</tr>
<tr>
<td>News Articles</td>
<td>0.63</td>
<td>0.95</td>
<td>0.76</td>
<td>0.73</td>
<td>0.63</td>
<td>0.68</td>
</tr>
<tr>
<td>Seeds</td>
<td>0.78</td>
<td>0.91</td>
<td>0.84</td>
<td>0.81</td>
<td>0.91</td>
<td>0.86</td>
</tr>
<tr>
<td>Voting records</td>
<td>0.93</td>
<td>0.93</td>
<td>0.93</td>
<td>0.98</td>
<td>0.89</td>
<td>0.93</td>
</tr>
</tbody>
</table>

*Table 4.2 Performance benchmarks*
4.2 Results

4.2.1 Parameters

The proposed algorithm has four parameters. They are: Minimum Support, Acceptable Entropy, Entropy Upper Threshold and Coverage Limit. To generate rules, the Minimum Support is set to five. This means that rules generated by the algorithm will apply to at least five tuples in the database. This value of support was chosen as default based on intuition and can be increased or decreased for different datasets. The Entropy Upper Threshold is set to 0.95 and Acceptable Entropy ranges between 0.1 and 0.3. Coverage Limit is varied from 1 to 5 to test its effect on classification accuracy and rules generated.

4.2.2 Performance

Table 4.3 compares the performance of our algorithm to CN2 [15] and C4.5 decision tree [4], both available in the Orange classification toolbox version 2.6.1. Since F-measure combines precision and recall, we use it as a representative of goodness of a classifier.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>CN2 F-measure</th>
<th>C4.5 F-measure</th>
<th>Our Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acceptable Entropy</td>
<td>Coverage limit</td>
<td>F-measure</td>
</tr>
<tr>
<td>Breast Cancer</td>
<td>0.98</td>
<td>0.97</td>
<td>0.1</td>
</tr>
<tr>
<td>Car Evaluation</td>
<td>0.81</td>
<td>0.84</td>
<td>0.3</td>
</tr>
<tr>
<td>LED Display</td>
<td>0.84</td>
<td>0.86</td>
<td>0.3</td>
</tr>
<tr>
<td>Mushroom</td>
<td>0.59</td>
<td>0.99</td>
<td>0.1</td>
</tr>
<tr>
<td>News Articles</td>
<td>0.76</td>
<td>0.68</td>
<td>0.1</td>
</tr>
<tr>
<td>Seeds</td>
<td>0.84</td>
<td>0.86</td>
<td>0.15</td>
</tr>
<tr>
<td>Voting records</td>
<td>0.93</td>
<td>0.93</td>
<td>0.2</td>
</tr>
</tbody>
</table>

*Table 4.3 Comparison of performance against benchmarks*
Of the seven datasets used, our algorithm has better performance on five of them. For the LED Display and Car Evaluation datasets, Decision tree has best F-measure. Detailed results on each of the datasets are explained in further sections.

4.2.3 Comparison of Rules Generated

Table 4.4 shows a comparison of rules generated using our algorithm to those using CN2. In most cases, the number of rules generated by our algorithm and average rule length is much smaller than CN2.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># of rules</th>
<th>Average rule length</th>
<th># of rules</th>
<th>Average rule length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast Cancer</td>
<td>23</td>
<td>3.08</td>
<td>10</td>
<td>1.46</td>
</tr>
<tr>
<td>Car Evaluation</td>
<td>109</td>
<td>4.41</td>
<td>28</td>
<td>3.42</td>
</tr>
<tr>
<td>LED Display</td>
<td>420</td>
<td>7.46</td>
<td>22</td>
<td>4.72</td>
</tr>
<tr>
<td>Mushroom</td>
<td>10</td>
<td>1.6</td>
<td>26</td>
<td>1.34</td>
</tr>
<tr>
<td>News Articles</td>
<td>37</td>
<td>1.35</td>
<td>97</td>
<td>1.02</td>
</tr>
<tr>
<td>Seeds</td>
<td>8</td>
<td>2.37</td>
<td>8</td>
<td>1.375</td>
</tr>
<tr>
<td>Voting records</td>
<td>14</td>
<td>2.64</td>
<td>9</td>
<td>2.77</td>
</tr>
</tbody>
</table>

*Table 4.4 Comparison of rules generated*

4.3 Effect of Coverage Limit

In this section we explain the effect of varying the coverage limit and acceptable entropy and measure its impact on the precision, recall and F-measure. Also we report the number of wrongly classified instances and database coverage of the prediction rules.
In order to study the results obtained, we analyze the performance on two representative datasets, the news articles dataset and the car evaluation dataset. There are 500 instances and 5 class labels in the news articles dataset whereas the car evaluation dataset has 1728 instances and 4 class labels. Table 4.5 shows how these two datasets differ with respect to support of class labels.

<table>
<thead>
<tr>
<th>Class Label</th>
<th>Support in dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business</td>
<td>27.75%</td>
</tr>
<tr>
<td>Politics</td>
<td>25.5%</td>
</tr>
<tr>
<td>Science</td>
<td>21.75%</td>
</tr>
<tr>
<td>Sports</td>
<td>26.75%</td>
</tr>
<tr>
<td>Technology</td>
<td>23.25%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class Label</th>
<th>Support in Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unacc</td>
<td>70.02%</td>
</tr>
<tr>
<td>Acc</td>
<td>22.22%</td>
</tr>
<tr>
<td>Good</td>
<td>3.99%</td>
</tr>
<tr>
<td>V-good</td>
<td>3.76%</td>
</tr>
</tbody>
</table>

*Table 4. 5 Class distribution in News articles dataset and Car evaluation dataset*

### 4.3.1 Set (I)

**News Articles Dataset**

The performance of our algorithm on the news articles dataset is shown in figure 4.3. As shown, the precision, recall and F-measure increase as the *Coverage Limit* and *Acceptable Entropy* increases. When the *Coverage Limit* is 1, each tuple can participate in rule generation at most once. On the other hand, with a high *Coverage Limit*, tuples can be covered more than once to learn rules about the dataset. This differentiates our technique from Decision Trees, which allow only single coverage.
We test our results with different values of Acceptable Entropy Threshold. A high entropy threshold relaxes the constraints on quality of rules. The entropy of a rule is inversely proportional to its purity. When entropy threshold is low, it means that we accept only high purity rules. Increasing the threshold implies that in addition to high purity rules, we also accept rules with slightly lower purity, thereby increasing the size of the rule set. Since all applicable rules are used to collectively predict the label of a test instance, the overall accuracy of the model changes. In order to achieve the best accuracy, we vary the entropy threshold, starting with a low entropy and gradually increasing it. Effect of change in entropy for different datasets does not show a fixed trend as some of the datasets are classified better with less stringent rules, whereas others show dip in performance when purity of rules decreases.

Table 4.6 shows that there is an increase in database coverage for rule generation as Coverage Limit increases. Unlike Decision Trees, at Coverage Limit 1, our algorithm does not cover all tuples as the Minimum Support for a rule to be accepted is set to 5. Decision Trees and CN2 cover all tuples in the database to learn rules with no restriction on the support of a rule, thereby increasing the risk of overfitting. As seen in Table 4.6, the algorithm we use covers 383 out of 396 training instances for coverage limit 5, clearly not accounting for some tuples if no rules with adequate support are applicable to them. Also, we can see information about how many test instances were wrongly classified, and how many test instances could not be assigned a class label.

Figure 4.3 shows that Database coverage and accuracy are related. Since Coverage Limit affects Database coverage, it also affects accuracy. The best performance is achieved at the Coverage Limit value where Database coverage is maximum. From our results we observe that Database
coverage remains constant on attaining its maximum value. The least amount of computation required to generate the best rule set is at the smallest Coverage Limit value where Database coverage has reached its maximum. This value can be identified as the point at which increasing the Coverage Limit no longer increases the Database coverage.

![Figure 4.3 Effect of Coverage limit and Acceptable entropy on News Articles Dataset](image)

*Figure 4.3 Effect of Coverage limit and Acceptable entropy on News Articles Dataset*

<table>
<thead>
<tr>
<th>Coverage Limit</th>
<th>No. of rules</th>
<th>Coverage</th>
<th>Could not classify</th>
<th>Wrongly Classified</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>24</td>
<td>245/396</td>
<td>42/104</td>
<td>5/62</td>
</tr>
<tr>
<td>2</td>
<td>52</td>
<td>325/396</td>
<td>24/104</td>
<td>29/80</td>
</tr>
<tr>
<td>3</td>
<td>76</td>
<td>361/396</td>
<td>9/104</td>
<td>20/95</td>
</tr>
<tr>
<td>4</td>
<td>94</td>
<td>381/396</td>
<td>6/104</td>
<td>16/98</td>
</tr>
<tr>
<td>5</td>
<td>97</td>
<td>383/396</td>
<td>5/104</td>
<td>13/99</td>
</tr>
</tbody>
</table>

*Table 4.6 Performance on News articles dataset*
4.3.2 Set (II)

Car Evaluation Dataset

Figure 4.4 shows the performance of our algorithm on the car evaluation dataset. We see a similar trend as Set (I), where precision, recall and f-measure increase as the coverage limit increases. Table 4.7 shows that the number of rules increases until a certain limit then either stays constant or decreases. Even though the coverage of the model increases it remains lower than that of Set (I) and is unable to assign a label to a large number of test instances. On taking a look at the rules generated, we observed that the algorithm was unable to generate any high purity rules that assigned label “Good”, “V-good” to unseen instances. Table 4.7 shows that only 3.99% and 3.76% of instances in the dataset belong to class “Good” and “V-good” respectively. With a minimum support of 5, our algorithm does not find any high purity pattern that can be used to classify instances into these classes. Lowering the support to 3 also did not help improve the classification, as it decreases the quality of rules generated for other classes. Due to this reason though high precision is achieved, the recall is poor.

Decision tree and CN2 are able to handle low support of class distribution as there is no restriction on support of rules generated. Table 4.8 shows an example of rules generated by CN2 on the Car Evaluation dataset. The rules generated for classes “Good” and “V-good” have very low support. The same can be observed in Figure 4.5, which shows a section of the decision tree generated on the same data. Though these algorithms perform better than our method on this dataset, we oppose generating rules with such low support. Using 1 or 2 instances to validate a
frequent pattern should be avoided as it may increase the risk of misclassification due to overfitting.

![Figure 4.4 Effect of Coverage Limit and Acceptable Entropy on Car Evaluation Dataset](image)

**Table 4.7 Performance on Car evaluation dataset**

<table>
<thead>
<tr>
<th>Coverage Limit</th>
<th>No. of rules</th>
<th>Coverage</th>
<th>No Label</th>
<th>Wrongly Classified</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>21</td>
<td>675/1210</td>
<td>269/518</td>
<td>4/249</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>922/1210</td>
<td>162/518</td>
<td>11/356</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
<td>929/1210</td>
<td>155/518</td>
<td>11/363</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>933/1210</td>
<td>156/518</td>
<td>8/362</td>
</tr>
<tr>
<td>5</td>
<td>24</td>
<td>933/1210</td>
<td>160/518</td>
<td>8/358</td>
</tr>
</tbody>
</table>
Table 4. 8 Rules generated using CN2 – low support

<table>
<thead>
<tr>
<th>Rule</th>
<th>Quality</th>
<th>Support</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF lug_boot=med AND safety=high AND buying=low AND doors=4 AND</td>
<td>0.667</td>
<td>1.000</td>
<td>vgood</td>
</tr>
<tr>
<td>maint=high THEN targetcategory=vgood</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF safety=high AND maint=med AND persons=more AND buying=med</td>
<td>0.667</td>
<td>1.000</td>
<td>vgood</td>
</tr>
<tr>
<td>AND doors=3 THEN targetcategory=vgood</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF lug_boot=med AND safety=high AND maint=med AND buying=med</td>
<td>0.667</td>
<td>1.000</td>
<td>vgood</td>
</tr>
<tr>
<td>AND doors=5 more AND persons=4 THEN targetcategory=vgood</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF buying=low AND persons=more AND lug_boot=med AND maint=low</td>
<td>0.750</td>
<td>2.000</td>
<td>good</td>
</tr>
<tr>
<td>AND safety=med THEN targetcategory=good</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF maint=low AND safety=high AND lug_boot=small AND doors=3 AND</td>
<td>0.750</td>
<td>2.000</td>
<td>good</td>
</tr>
<tr>
<td>persons=more THEN targetcategory=good</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IF buying=low AND safety=high AND maint=med AND persons=4 AND</td>
<td>0.750</td>
<td>2.000</td>
<td>good</td>
</tr>
<tr>
<td>lug_boot=small THEN targetcategory=good</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4. 5 Example of low support rules generated using Decision tree
4.3.3 Other datasets

Set (I) and (II) summarize the behavior of all seven datasets. The results for the Breast Cancer, Mushroom, Seeds and voting records dataset resemble Set (I). LED Display dataset is similar in nature to the Car Evaluation Dataset and Set (II) explains behavior of our algorithm on this dataset. Precision, recall and F-measure for each dataset is shown in figures 4.6 to 4.10. Figure 4.11 shows the overall effect of Coverage Limit on rule induction and classification.

Figure 4. 6 Effect of Coverage Limit and Acceptable Entropy on Breast Cancer Dataset
Figure 4.7 Effect of Coverage Limit and Acceptable Entropy on Mushrooms Dataset

Figure 4.8 Effect of Coverage Limit and Acceptable Entropy on LED Display Dataset
Figure 4. 9 Effect of Coverage Limit and Acceptable Entropy on Seeds Dataset

Figure 4. 10 Effect of Coverage Limit and Acceptable Entropy on Voting Records Dataset
Figure 4.11 Overall Effect of Coverage Limit

Figure 4.12 shows average database coverage in rule induction for the two dataset groups. Due to low coverage in Car Evaluation and LED Display dataset, performance of our algorithm is lower using these datasets when compared to others.

Figure 4.12 Coverage Limit vs Database coverage
4.3.4 Additional Result Details

In this section we provide detailed performance results on each of the seven datasets. Tables 4.9 to 4.15 contain information about number of rules generated, fraction of database covered in rule induction, number of instances to which no label was assigned, number of misclassified instances and performance measures along with entropy threshold and Coverage Limit used to obtain these results.

<table>
<thead>
<tr>
<th>Acceptable entropy</th>
<th>Coverage Limit</th>
<th>No. of rules</th>
<th>Database Coverage</th>
<th>No Label</th>
<th>Wrongly Classified</th>
<th>Precision</th>
<th>Recall</th>
<th>Fmeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1</td>
<td>24</td>
<td>0.62</td>
<td>42/104</td>
<td>5 of 62</td>
<td>0.91</td>
<td>0.54</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>52</td>
<td>0.82</td>
<td>24/104</td>
<td>29/80</td>
<td>0.93</td>
<td>0.71</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>76</td>
<td>0.91</td>
<td>9/104</td>
<td>20/95</td>
<td>0.87</td>
<td>0.79</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>94</td>
<td>0.96</td>
<td>6/104</td>
<td>16/98</td>
<td>0.88</td>
<td>0.83</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>97</td>
<td>0.97</td>
<td>5/104</td>
<td>13/99</td>
<td>0.9</td>
<td>0.86</td>
<td>0.88</td>
</tr>
<tr>
<td>0.2</td>
<td>1</td>
<td>24</td>
<td>0.62</td>
<td>42</td>
<td>47/104</td>
<td>0.91</td>
<td>0.54</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>49</td>
<td>0.81</td>
<td>24</td>
<td>29/104</td>
<td>0.93</td>
<td>0.71</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>75</td>
<td>0.91</td>
<td>9</td>
<td>20/104</td>
<td>0.87</td>
<td>0.79</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>93</td>
<td>0.96</td>
<td>6</td>
<td>16/104</td>
<td>0.88</td>
<td>0.83</td>
<td>0.85</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>97</td>
<td>0.97</td>
<td>5</td>
<td>13/104</td>
<td>0.9</td>
<td>0.86</td>
<td>0.88</td>
</tr>
</tbody>
</table>

Table 4.9 Performance on News articles dataset

<table>
<thead>
<tr>
<th>Acceptable entropy</th>
<th>Coverage Limit</th>
<th>No. of rules</th>
<th>Database Coverage</th>
<th>No Label</th>
<th>Wrongly Classified</th>
<th>Precision</th>
<th>Recall</th>
<th>Fmeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1</td>
<td>21</td>
<td>0.90</td>
<td>243</td>
<td>0</td>
<td>1</td>
<td>0.9</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>27</td>
<td>0.97</td>
<td>57</td>
<td>0</td>
<td>1</td>
<td>0.97</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>26</td>
<td>1.00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>27</td>
<td>1.00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>27</td>
<td>1.00</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.10 Performance on Mushroom dataset
<table>
<thead>
<tr>
<th>Acceptable entropy</th>
<th>Coverage Limit</th>
<th>No. of rules</th>
<th>Database coverage</th>
<th>No. Label</th>
<th>Misclassified</th>
<th>Precision</th>
<th>Recall</th>
<th>Fmeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1</td>
<td>6</td>
<td>0.77</td>
<td>33</td>
<td>34/140</td>
<td>0.99</td>
<td>0.69</td>
<td>0.81</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>12</td>
<td>0.89</td>
<td>10</td>
<td>11/140</td>
<td>0.99</td>
<td>0.87</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>18</td>
<td>0.92</td>
<td>8</td>
<td>9/140</td>
<td>0.99</td>
<td>0.89</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>23</td>
<td>0.93</td>
<td>7</td>
<td>8/140</td>
<td>0.99</td>
<td>0.9</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>26</td>
<td>0.95</td>
<td>7</td>
<td>7/140</td>
<td>1</td>
<td>0.91</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>1</td>
<td>6</td>
<td>0.72</td>
<td>32</td>
<td>32/140</td>
<td>1</td>
<td>0.71</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>0.92</td>
<td>5</td>
<td>7/140</td>
<td>0.98</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>16</td>
<td>0.96</td>
<td>6</td>
<td>8/140</td>
<td>0.98</td>
<td>0.91</td>
<td>0.94</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>15</td>
<td>0.97</td>
<td>5</td>
<td>7/140</td>
<td>0.98</td>
<td>0.92</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>17</td>
<td>0.97</td>
<td>5</td>
<td>7/140</td>
<td>0.98</td>
<td>0.92</td>
<td>0.95</td>
</tr>
<tr>
<td>0.3</td>
<td>1</td>
<td>5</td>
<td>0.73</td>
<td>36</td>
<td>37/140</td>
<td>0.99</td>
<td>0.65</td>
<td>0.78</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>8</td>
<td>0.89</td>
<td>8</td>
<td>10/140</td>
<td>0.98</td>
<td>0.89</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>12</td>
<td>0.96</td>
<td>4</td>
<td>6/140</td>
<td>0.98</td>
<td>0.94</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>10</td>
<td>0.99</td>
<td>0</td>
<td>1/140</td>
<td>0.98</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>13</td>
<td>0.99</td>
<td>0</td>
<td>1/140</td>
<td>0.98</td>
<td>0.99</td>
<td>0.99</td>
</tr>
</tbody>
</table>

*Table 4.11 Performance on Breast Cancer dataset*

<table>
<thead>
<tr>
<th>Acceptable entropy</th>
<th>Coverage Limit</th>
<th>No. of rules</th>
<th>Database coverage</th>
<th>No. Label</th>
<th>Wrongly Classified</th>
<th>Precision</th>
<th>Recall</th>
<th>Fmeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1</td>
<td>21</td>
<td>0.55</td>
<td>269/518</td>
<td>4/249</td>
<td>0.96</td>
<td>0.39</td>
<td>0.56</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>32</td>
<td>0.76</td>
<td>162/518</td>
<td>11/356</td>
<td>0.9</td>
<td>0.56</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>28</td>
<td>0.76</td>
<td>155/518</td>
<td>11/363</td>
<td>0.9</td>
<td>0.57</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>24</td>
<td>0.77</td>
<td>156/518</td>
<td>8/362</td>
<td>0.9</td>
<td>0.57</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>24</td>
<td>0.77</td>
<td>160/518</td>
<td>8/358</td>
<td>0.9</td>
<td>0.57</td>
<td>0.7</td>
</tr>
<tr>
<td>0.2</td>
<td>1</td>
<td>21</td>
<td>0.55</td>
<td>269</td>
<td>273/518</td>
<td>0.96</td>
<td>0.39</td>
<td>0.56</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>32</td>
<td>0.76</td>
<td>162</td>
<td>173/518</td>
<td>0.9</td>
<td>0.56</td>
<td>0.69</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>28</td>
<td>0.76</td>
<td>155</td>
<td>166/518</td>
<td>0.9</td>
<td>0.57</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>24</td>
<td>0.77</td>
<td>156</td>
<td>164/518</td>
<td>0.9</td>
<td>0.57</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>27</td>
<td>0.77</td>
<td>147</td>
<td>168/518</td>
<td>0.89</td>
<td>0.57</td>
<td>0.69</td>
</tr>
<tr>
<td>0.3</td>
<td>1</td>
<td>29</td>
<td>0.61</td>
<td>240</td>
<td>250/518</td>
<td>0.74</td>
<td>0.29</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>44</td>
<td>0.81</td>
<td>124</td>
<td>151/518</td>
<td>0.68</td>
<td>0.38</td>
<td>0.49</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>47</td>
<td>0.84</td>
<td>98</td>
<td>166/518</td>
<td>0.66</td>
<td>0.39</td>
<td>0.49</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>44</td>
<td>0.85</td>
<td>91</td>
<td>164/518</td>
<td>0.65</td>
<td>0.39</td>
<td>0.49</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>48</td>
<td>0.86</td>
<td>82</td>
<td>144/518</td>
<td>0.65</td>
<td>0.39</td>
<td>0.49</td>
</tr>
</tbody>
</table>

*Table 4.12 Performance on Car Evaluation Dataset*
### Table 4.13 Performance on LED Display dataset

<table>
<thead>
<tr>
<th>Acceptable entropy</th>
<th>Coverage Limit</th>
<th>No. of rules</th>
<th>Database coverage</th>
<th>No. Label</th>
<th>Wrongly Classified</th>
<th>Precision</th>
<th>Recall</th>
<th>Fmeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1</td>
<td>6</td>
<td>0.13</td>
<td>1304</td>
<td>1320/1500</td>
<td>0.86</td>
<td>0.31</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>9</td>
<td>0.15</td>
<td>1275</td>
<td>1294/1500</td>
<td>0.83</td>
<td>0.29</td>
<td>0.43</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
<td>0.15</td>
<td>1274</td>
<td>1294/1500</td>
<td>0.82</td>
<td>0.29</td>
<td>0.43</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>12</td>
<td>0.15</td>
<td>1270</td>
<td>1291/1500</td>
<td>0.82</td>
<td>0.29</td>
<td>0.43</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>12</td>
<td>0.15</td>
<td>1270</td>
<td>1291/1500</td>
<td>0.82</td>
<td>0.29</td>
<td>0.43</td>
</tr>
<tr>
<td>0.2</td>
<td>1</td>
<td>9</td>
<td>0.25</td>
<td>1112</td>
<td>1164/1500</td>
<td>0.79</td>
<td>0.32</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>15</td>
<td>0.32</td>
<td>1013</td>
<td>1089/1500</td>
<td>0.79</td>
<td>0.4</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>18</td>
<td>0.33</td>
<td>989</td>
<td>1083/1500</td>
<td>0.77</td>
<td>0.4</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>21</td>
<td>0.35</td>
<td>974</td>
<td>1076/1500</td>
<td>0.77</td>
<td>0.41</td>
<td>0.54</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>21</td>
<td>0.35</td>
<td>974</td>
<td>1076/1500</td>
<td>0.77</td>
<td>0.41</td>
<td>0.54</td>
</tr>
<tr>
<td>0.3</td>
<td>1</td>
<td>14</td>
<td>0.57</td>
<td>667</td>
<td>807/1500</td>
<td>0.82</td>
<td>0.58</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>22</td>
<td>0.68</td>
<td>500</td>
<td>694/1500</td>
<td>0.8</td>
<td>0.67</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>26</td>
<td>0.71</td>
<td>454</td>
<td>678/1500</td>
<td>0.78</td>
<td>0.69</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>26</td>
<td>0.72</td>
<td>420</td>
<td>669/1500</td>
<td>0.77</td>
<td>0.7</td>
<td>0.73</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>26</td>
<td>0.72</td>
<td>420</td>
<td>669/1500</td>
<td>0.77</td>
<td>0.7</td>
<td>0.73</td>
</tr>
</tbody>
</table>

### Table 4.14 Performance on Voting records dataset

<table>
<thead>
<tr>
<th>Acceptable entropy</th>
<th>Coverage Limit</th>
<th>No. of rules</th>
<th>Database coverage</th>
<th>No. Label</th>
<th>Wrongly Classified</th>
<th>Precision</th>
<th>Recall</th>
<th>Fmeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1</td>
<td>3</td>
<td>0.60</td>
<td>45</td>
<td>46/130</td>
<td>0.93</td>
<td>0.53</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>9</td>
<td>0.88</td>
<td>10</td>
<td>12/130</td>
<td>0.97</td>
<td>0.89</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>15</td>
<td>0.91</td>
<td>8</td>
<td>11/130</td>
<td>0.96</td>
<td>0.9</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>17</td>
<td>0.95</td>
<td>7</td>
<td>9/130</td>
<td>0.98</td>
<td>0.92</td>
<td>0.95</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>20</td>
<td>0.96</td>
<td>6</td>
<td>8/130</td>
<td>0.98</td>
<td>0.93</td>
<td>0.95</td>
</tr>
<tr>
<td>0.2</td>
<td>1</td>
<td>3</td>
<td>0.60</td>
<td>45</td>
<td>46/130</td>
<td>0.93</td>
<td>0.53</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>9</td>
<td>0.94</td>
<td>4</td>
<td>6/130</td>
<td>0.97</td>
<td>0.95</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>14</td>
<td>0.97</td>
<td>3</td>
<td>6/130</td>
<td>0.96</td>
<td>0.95</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>17</td>
<td>0.98</td>
<td>3</td>
<td>5/130</td>
<td>0.97</td>
<td>0.96</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>17</td>
<td>0.99</td>
<td>2</td>
<td>5/130</td>
<td>0.96</td>
<td>0.96</td>
<td>0.96</td>
</tr>
</tbody>
</table>
Table 4.15 Performance on Seeds dataset

<table>
<thead>
<tr>
<th>Acceptable entropy</th>
<th>Coverage Limit</th>
<th>No. of rules</th>
<th>Database Coverage</th>
<th>No. Label</th>
<th>Wrongly Classified</th>
<th>Precision</th>
<th>Recall</th>
<th>Fmeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>1</td>
<td>3</td>
<td>0.76</td>
<td>15</td>
<td>20/63</td>
<td>0.91</td>
<td>0.69</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>0.79</td>
<td>12</td>
<td>18/63</td>
<td>0.9</td>
<td>0.72</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>5</td>
<td>0.80</td>
<td>11</td>
<td>17/63</td>
<td>0.91</td>
<td>0.73</td>
<td>0.81</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>6</td>
<td>0.81</td>
<td>11</td>
<td>17/63</td>
<td>0.91</td>
<td>0.73</td>
<td>0.81</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>8</td>
<td>0.89</td>
<td>7</td>
<td>14/63</td>
<td>0.88</td>
<td>0.78</td>
<td>0.83</td>
</tr>
<tr>
<td>0.15</td>
<td>1</td>
<td>3</td>
<td>0.76</td>
<td>15</td>
<td>20/63</td>
<td>0.91</td>
<td>0.69</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
<td>0.84</td>
<td>9</td>
<td>15/63</td>
<td>0.9</td>
<td>0.77</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7</td>
<td>0.86</td>
<td>6</td>
<td>12 of 63</td>
<td>0.91</td>
<td>0.81</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>8</td>
<td>0.88</td>
<td>5</td>
<td>11 of 63</td>
<td>0.91</td>
<td>0.82</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>10</td>
<td>0.94</td>
<td>2</td>
<td>9 of 63</td>
<td>0.89</td>
<td>0.85</td>
<td>0.87</td>
</tr>
<tr>
<td>0.2</td>
<td>1</td>
<td>3</td>
<td>0.76</td>
<td>15</td>
<td>20/63</td>
<td>0.91</td>
<td>0.69</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>7</td>
<td>0.84</td>
<td>9</td>
<td>15/63</td>
<td>0.9</td>
<td>0.77</td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>7</td>
<td>0.86</td>
<td>6</td>
<td>12 of 63</td>
<td>0.91</td>
<td>0.81</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>8</td>
<td>0.88</td>
<td>5</td>
<td>11 of 63</td>
<td>0.91</td>
<td>0.82</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>10</td>
<td>0.94</td>
<td>2</td>
<td>9 of 63</td>
<td>0.89</td>
<td>0.85</td>
<td>0.87</td>
</tr>
</tbody>
</table>

4.4 Comparison with CMAR and CPAR

The direction of our research aligns with the CMAR algorithm to an extent. Similar to our strategy, Li et al. [6] experiment with instance sharing using a coverage threshold. However, they draw an inference that classification accuracy is not sensitive to this threshold. Due to this reason, they are unable to propose a fixed method to determine the optimal coverage threshold to achieve best accuracy. Results from [6] show that when coverage threshold is increased beyond a certain limit, there is a decline in accuracy. Unlike CMAR, the results obtained using our approach do not display such a trend. The higher quality of rules developed by our model ensure that increasing the number of rules does not lower our classification accuracy. We infer
that the optimal Coverage Limit is the point where database coverage stabilizes. At this threshold, we may conclude that our rule set covers maximum patterns in the data.

In addition to the benchmarks used for measuring the goodness of our classifier, we also compare the results of the algorithm to those of the CMAR and CPAR algorithms. To do this, we selected the Pima Indian diabetes dataset, Iris Dataset and the Wisconsin breast cancer dataset, all available at the UCI’s Machine learning repository [25]. The accuracy of CMAR and CPAR on these datasets are extracted from [6] and [7]. Table 4.8 compares accuracy of our classifier to these algorithms. It also shows the performance of Decision Trees and Rule based classifier on the selected datasets.

Accuracy is measured as the percentage of testing set examples correctly classified by the algorithm. As shown in Table 4.8, our algorithm performs better than its competitors on the three datasets selected.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># attributes</th>
<th># classes</th>
<th># instances</th>
<th>C4.5</th>
<th>CBA</th>
<th>CMAR</th>
<th>CPAR</th>
<th>Our Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breast</td>
<td>10</td>
<td>2</td>
<td>699</td>
<td>95</td>
<td>96.3</td>
<td>96.4</td>
<td>96</td>
<td>99</td>
</tr>
<tr>
<td>Iris</td>
<td>4</td>
<td>3</td>
<td>150</td>
<td>95.3</td>
<td>94.7</td>
<td>94</td>
<td>94.7</td>
<td>97.5</td>
</tr>
<tr>
<td>Pima</td>
<td>8</td>
<td>2</td>
<td>768</td>
<td>75.5</td>
<td>72.9</td>
<td>75.1</td>
<td>73.8</td>
<td>78.08</td>
</tr>
</tbody>
</table>

*Table 4.9 Comparison with CMAR and CPAR*
Chapter 5

Conclusion and Future Work

Rule based algorithms have been studied extensively over the past two decades. They are widely used in areas such as medical diagnosis, finance and marketing. These algorithms are popular as they are fast, accurate and are capable of finding an optimal number of rules without missing many patterns in the data. Over the years, several approaches have been considered to mine frequent patterns. The most commonly adopted approaches are variants of Apriori algorithm and Decision Trees.

Most existing algorithms and their variants are capable of developing high accuracy classifiers, however they are still associated with some drawbacks. Association rule mining based algorithms perform explosive number of computations and use too few rules to classify an instance. On the other hand Decision Trees adopt a greedy approach to split the dataset due to which good patterns may be divided into different branches. They also do not allow re-use of training instances for rule generation, resulting in a small set of sub-optimal rules. Further, algorithms with stringent coverage requirements tend to over fit the data and produce too specific rules.
Newer rule based algorithms are centered on resolving limitations of earlier methods but have only managed to do so, partially.

The algorithm proposed by us uses a fast Apriori based rule generation process and follows it by classifying instances using all applicable rules. We focus on re-use of training instances by using a multiple coverage approach with a fixed threshold to control the database coverage and measure its impact on the accuracy of the classifier. Chapter 4 gives a deep insight into the results of our approach on several datasets, from which we confirm that coverage is an important parameter that can be used to control the rule induction process. We observed that highest classification accuracy is achieved when the database coverage is at its peak. We also measured the impact of rule purity on our accuracy and learnt that the effect of rule purity threshold on accuracy is highly dependent on the nature of the dataset. Depending on how much noise is present, few datasets require very stringent rules, whereas others show higher accuracy with relaxed rule purity threshold.

Despite its high accuracy compared to other algorithms, our approach has a few drawbacks. Firstly, after using different datasets, we inferred that the performance of our algorithm is higher than its competitors when each class in the data has a good support. When the distribution of one of the classes is extremely low when compared to the others, our algorithm does not generate many rules for that class, resulting in lower recall despite high precision. The reason for this is that we use a fixed support threshold for all classes. In the future, to improve the algorithm, a different support threshold can be used for each class by making the threshold a function of the class distribution. Secondly, unlike decision trees, our algorithm requires the user to input support and entropy thresholds as a parameter. This may be inconvenient to a few, as
they may have no insight into the data. For this reason, there should be some work towards analyzing the dataset and setting appropriate thresholds based on number of training instances present in each class. However, a default support threshold of 5 and entropy upper threshold of 0.95 confirms that our algorithm in general performs better than many others.
References


[18] J. Han, J. Pei, and Y. Yin., “Mining frequent patterns without candidate generation,” *SIGMOD’00*, Dallas, TX, May: pp. 1-12, 2000.


