Pruning Methods for Rule Induction

Osama M. Othman

Informatics Research Centre
University of Salford, Salford, UK

Submitted in fulfilment of the Requirements of the Degree of
Doctor of Philosophy, 2016

Contents

List of Figures

List of Tables

Acknowledgements

I gratefully acknowledge my PhD supervisor, Dr Chris Bryant, for his continuous encouragement and support during this whole project, from the start of the literature review to the thesis' conclusion. His advice and expertise in the field of machine learning were crucial to the success of this work. His feedback and explanations also helped me understand how to present ideas and demonstrate the results and analyses.

I would like to thank my local advisor, Dr Khalil el Hindi, for his advice and suggestions in the early stages of this project. Next, I would like to express my sincere thanks to Robin Boswell, who implemented the code for CN2 algorithm in 1990, from which Francisco Reinaldo and Marcus Siqueira created the executable file for Windows XP.

Finally, I would like to dedicate this work to my parents and my wife, who taught me things that no school can teach.

List of abbreviations

AI – Artificial Intelligence.

ACO – Ant Colony Optimization.

IBL – Instance Based Learning.

ANN – Artificial Neural Network.

NN – Nearest Neighbour.

VDM – Value Difference Metric.

FSS – Forward Sequential Selection.

BSS – Backward Sequential Selection.

RISE – Rule Induction from Set of Examples.

IREP – Incremental Reduced Error Pruning.

RIPPER – Repeated Pruning to Produce Error Reduction.

CNN – Condensed Nearest Neighbour.

SNN – Selective Nearest Neighbour.

TRKNN – Template Reduction KNN.

RNN – Reduced Nearest Neighbour Rule.

SSRR – Sort then Select Rule Reduction.

SLIPPER – Simple Learner with Iterative Pruning to Produce Error Reduction.

AS – Ant System.

NP hard - Non-Polynomial.

TSP – Traveling Salesman Problem.

ACO-IR – Instance Reduction method using Ant Colony Optimization.

DROP5 - Decremental Reduction Optimization Procedure.

Abstract

Machine learning is a research area within computer science that is mainly concerned with discovering regularities in data. Rule induction is a powerful technique used in machine learning wherein the target concept is represented as a set of rules. The attraction of rule induction is that rules are more transparent and easier to understand compared to other induction methods (e.g., regression methods or neural network). Rule induction has been shown to outperform other learners on many problems. However, it is not suitable to handle exceptions and noisy data in training sets, which can be solved by pruning.

This thesis is concerned with investigating whether preceding rule induction with instance reduction techniques can help in reducing the complexity of rule sets by reducing the number of rules generated without adversely affecting the predictive accuracy.

An empirical study is undertaken to investigate the application of three different rule classifiers to datasets that were previously reduced with promising instance-reduction methods. Furthermore, we propose a new instance reduction method based on Ant Colony Optimization (ACO). We evaluate the effectiveness of this instance reduction method for k nearest neighbour algorithms in term of predictive accuracy and amount of reduction. Then we compared it with other instance reduction methods.

We show that pruning classification rules with instance-reduction methods lead to a statistically significant decrease in the number of generated rules, without adversely affecting performance. On the other hand, applying instance-reduction methods enhances the predictive accuracy on some datasets. Moreover, the results provide evidence that: (1) our proposed instance reduction method based on ACO is competitive with the well-known k-NN algorithm; (2) the reduced sets computed by our method offers better classification accuracy than those obtained by the compared algorithms.

Chapter 1: Introduction

Machine learning is "a mature and well-recognized research area of computer science, mainly concerned with the discovery of models, patterns, and other regularities in data" (Fürnkranz et al., 2012). The field of machine learning has received a great deal of attention recently. The aim is to develop computational methods that implement various forms of learning. Induction is one type of learning that induces a concept description from a set of examples. This is especially important in ill-defined domains that lack algorithmic solution.

In general, machine learning is concerned with the question of how to automatically improve performance for tasks associated with artificial intelligence (AI) (e.g., recognition, diagnosis, planning, robot control, prediction, etc.), based on experience, in order to teach computers to solve problems by merely "showing" them the selected examples.

The importance of machine learning arises from the following (Nilsson, 1996):

1. Some tasks cannot be defined well except by examples, because we can specify the input/output pairs but we cannot define the relation between input and desired output.
2. The amount of knowledge available for a particular task might be too large for explicit manual encoding.
3. Certain characteristics of the working environment might not be completely known at design time; thus, humans may produce machines that do not work as well as desired in the environment in which they are used.
4. Many environments change over time, so machines that can adapt to a changing environment would reduce the constant need for redesign.

5. Machine learning helps us to understand how animals and humans learn.

The machine learning community has expressed a need to improve the performance of learning algorithms with respect to predictive accuracy, and how to produce classifiers that can be understood by humans.

This thesis is concerned with concept descriptions in the form of classification rules that can be easily understood by humans. However, most rule-based systems still tend to induce quite a large number of rules, making the description obtained difficult to understand. A variety of methods have been proposed to prune the produced rule sets. These methods help in reducing the complexity of generated rule sets, but can still suffer from critical problems due to the prevalence of large, noisy datasets in real-world applications and covering hard-to-learn instances.

Furthermore, our work concerns the use of pruning to solve one of the most important problems in the field of machine learning – namely, overfitting, which affects the predictive accuracy. We say that the produced classifier overfits the data if we can find a different classifier with more errors over training examples but smaller errors over test data. Overfitting occurs in two situations: when the training set contains noisy instances and when the training set is not a representative sample from the instance space (Mitchell, 1997). Both of these situations are common in real-world applications.

On other hand, our work concerns with applying Ant Colony optimization (ACO) method in proposing Instance reduction technique. ACO algorithm involve simple ants that cooperate with

each other to achieve a unified behaviour for the system, allowing the design of a robust system able to find a high-quality solution for problems.

1.1 Research Question

The research questions addressed in this thesis are as follows:

Is it possible to reduce the number of generated rules by training rule classifiers on datasets that have previously been reduced with instance-reduction methods? What is the effect of this on the predictive accuracy?

This thesis investigates a reduction in the complexity of rule sets by decreasing the number of generated rules. We investigate new pre-pruning techniques for rule-induction methods by applying the promising instance-reduction methods, specifically instance-reduction methods that eliminate border instances, which tend to be noisy, or difficult to learn and untypical. The aim is to simplify the induced rule set by removing some of the rules without adversely affecting the predictive performance. It also investigates how Ant Colony Optimization (ACO) can be used as an instance-reduction method and using it as a pre -processing technique for rule-induction methods.

1.2 Motivation

El Hindi and Alakhras (2009) showed that filtering out border instances before training an artificial neural network improves the predictive accuracy and speeds up the training process by reducing the training epochs. Previous research on pre-pruning has focused on simplifying the rules during induction. Gamberger et al. (1996) investigated the effect of a new noisy instance

detection method before rule induction on a specific dataset (i.e., early diagnosis of rheumatic diseases) (Gamberger et al., 1996); this method is suitable for datasets with just two classes. In another case, Grudzinski et al. (2010) concentrated on the EkP system (Grudzinski, 2008) as an instance-reduction method before rule induction, and illustrated that it is possible to extract simpler sets of rules from reduced datasets (Grudzinski et al., 2010). However, no study to date has investigated the effect of preceding rule induction with instance reduction, in terms of predictive accuracy and complexity of the rule set produced. Here, we investigate whether there is any advantage to preceding the rule induction with instance-reduction methods in terms of the complexity of a rule set (roughly represented here by the number of generated rules), taking into consideration the effect on predictive accuracy.

On the other hand, we propose a new instance-reduction method using ACO (Dorigo et al., 1996), and how to use it as a pre-pruning technique for rule induction. The main idea of ACO is to use repeated simulations of artificial ants to generate new solutions to the problem at hand. The "ants" use information collected at a previous time to direct their search. They deposit "pheromones" on the ground in order to mark a favourable path that should be followed by other members of the colony.

1.3 Contribution to Knowledge

The contributions to knowledge made by this thesis are in the field of machine learning, specifically in the area of rule induction and pruning. As far as the author is aware, this is the first

work to investigate whether the number of generated rules can be reduced by preceding rule induction with instance-reduction methods.

This thesis considers rule-induction methods that learn a set of propositional rules where the target concept is represented as a set of "if... then..." rules. Each rule consists of an antecedent (or body of rule) and a consequent. The consequent represents the predicted class; the antecedent part is composed of a conjunction of conditions, each involving one attribute. We focus on rule-induction methods that produce an unordered set of rules because we are interested in rule sets where each rule can be understood independently. Moreover, we consider instance-reduction methods that eliminate border instances, which tend to be noisy or difficult to learn and untypical. The results presented in this thesis show that training three rule classifiers on datasets that have previously been reduced with instance-reduction methods leads to a statistically significant decrease in the number of generated rules, without adversely affecting the predictive performance.

This study:

- Investigates whether the number of generated rules can be reduced by preceding rule induction with instance-reduction methods;
- Investigates the effect of preceding rule induction with instance-reduction methods on the predictive performance, compared to using an unpruned training set;
- Proposes a new instance-reduction method based on ACO; and finally
- Compares the achievement of the proposed method with other different instance-reduction methods, in terms of predictive accuracy and number of generated rules.

The work described in this thesis has not been submitted previously as part of requirements for another degree and it is the result of my own independent work, unless otherwise stated. Some of the ideas described in Chapter 5, and most of the work and results presented in Chapter 5, have been proposed and published in the following:

Othman, O. and Bryant, C. (2013), "Preceding rule induction with instance-reduction methods", Perner, Petra (eds.) in Proc. of the 9th International Conference on Machine Learning and Data Mining in Pattern Recognition, Springer-Verlag, Berlin, pp. 209–218.

Othman, O., and Bryant, C. (2015). "Pruning classification rules with instance reduction methods", International Journal of Machine Learning and Computing, Vol. 5 No. 3, pp. 187–191.

1.4 Structure of this Thesis

The remainder of this thesis is structured as follows:

- Chapter 2: Literature Review: Rule induction and Pruning

  Provides an introduction and background to pruning and an overview for learning algorithms related to this thesis – namely, IBL and rule induction method. Some of the different rule-induction methods are compared and discussed. Additionally, the different instance-reduction methods are mentioned. Moreover, we provide an overview of pruning algorithms, including a description of different pruning methods related to our works

- Chapter 3: Literature Review: Ant Colony Optimization

  Provides an introduction and background to Ant Colony Optimization (ACO). The main focus of this section is the concept of ACO and its applications in the AI field.

- Chapter 4: Experimental Framework

  Introduces all materials required to run our experiments; this chapter outlines our work and clarifies the methodology for comparing different algorithms.

  a. Problem statement: Provides a brief description of the problem we are interested in.
  b. Aims: Describes the idea behind our work, and clarifies this using diagrams.
  c. Comparison of methodologies.
  d. Evaluation measure.
  e. Rule-induction characteristics: Specifies the characteristics of the rule-induction methods we are interested in during our experiments.
  f. Experimental setup: Outlines the datasets and programs used in the experiments.
  g. Experiment implementation.

- Chapter 5: Preceding Rule Induction by Instance-Reduction Methods

  Explains the experiments and algorithms used for instance reduction, and outlines the different rule inductions we are testing.

- Chapter 6: Instance-Reduction Method Based on ACO

  Explains the motivation behind the proposed method based on ACO.

  a. Problem representation.
  b. Methodology: Present our algorithm for instance reduction based on ACO.
  c. Comparison of results (with IBL and other instance-reduction methods) in terms of:

     1- Predictive accuracy.

     2- Reductions in number of instances.

d. Comparison of results (with different instance-reduction methods, such as pre-pruning for rule induction) in terms of:

    1- Predictive accuracy.

    2- Reductions in number of rules produced.

- Chapter 7: Discussion and Future Works

  Discusses the conclusions and main findings drawn from the comparison and evaluation and whether the research hypothesis has been proven, and suggests future development, which may be necessary.

- References.

Chapter 2: Literature Review: Rule Induction and Pruning

This chapter starts by providing an overview of the field of machine learning, focusing mainly on its subfields relevant to this work. This is followed by an introduction to rule induction and instance-based learning methods. The concepts of pruning are also explained.

2.1 Introduction

Information accumulated over thousands of years has exceeded the capacity of human brains. Hence, the concern in the science world has always been how to derive useful information from such huge amounts of data. Machine learning has the central purpose of learning from data. Learning refers to any change in a system that causes its performance to improve (Simon, 1983).

The aim of machine learning is to develop computational methods that implement various forms of learning. Most research in machine learning has focused on conceptual learning or classification learning. Induction is a type of learning that induces a concept description from a set of examples. This is especially important in ill-defined domains that lack algorithmic solution.

The study of inductive learning is mainly motivated by the desire to automate the process of knowledge acquisition during the construction of expert systems.

2.2 Definitions

To learn a concept is to infer its general definition from a set of examples (instances) (Domingos, 1997). Learning can be considered a method to generate an approximation to the function, f(x), where the domain is defined by a set of examples, while the range of f(x) is the set of concepts or classes in which the examples are classified.

Inductive methods can be divided into two categories. The first is called supervised concept learning, or classification learning, in which each example appears with its corresponding classification. The other is called unsupervised learning, or clustering, which involves learning from a set of unclassified examples where the goal is to form a new concept description that has certain desired properties (Domingos, 1997).

Important terms must be defined to make the remainder of this review understandable. Instance space refers to the set of all possible examples. Each example can be described in a variety of forms; however, the most common description is as a vector of attributes. An attribute is a variable that can be symbolic (nominal) or continuous. Symbolic attributes can take a finite number of values, which have no ordered relationship. For example, the attribute colour with values {red, white, and blue} is a symbolic attribute. A continuous attribute (e.g., length, weight) is an ordered set of values, such as age and temperature, and it occupies any value over a real number. Each example may contain a combination of the two kinds of attributes, in addition to a categorical attribute (class attribute) that may either be symbolic or continuous.

A training set is a set of examples used to build a classifier – i.e., the function that maps previously unseen examples into predicted classes. These unseen examples are called test examples. They are used to test the accuracy of a generated classifier.

In supervised learning, the concept to be learned is called the target. The examples in the training set that have the same class as the concept are called positive examples and others are called negative examples with respect to that class.

2.3 An overview of learning algorithms

Concept learning can be viewed as having three components: representation, search, and evaluation. Representation is the means of representing the knowledge (e.g., decision trees, sets of rules, instances, graphical models, neural networks, etc.). The search procedure is the process by which the learning algorithm finds the concept description in a space of possible descriptions defined by the representation language. The evaluation component takes a candidate concept description and returns a measure of its quality (Domingos, 1997).

There is a great variety of learning algorithms in terms of knowledge representation. The general definition for the concepts can be represented in different forms, which can be a set of rules (e.g., CN2 [Clark & Niblett, 1989] and AQ algorithms), decision trees (e.g., C4.5 [Quinlan, 1993] and ID3 [Quinlan, 1986]), artificial neural networks (McClelland & Rumelhart, 1986), or the same representation as the training examples (e.g., IBL).

In this section, we will review two well-known learning algorithms that are related to our work in this thesis – namely, IBL and rule-induction methods. In Section 2.3.1 we will discuss the

framework for the instance-reduction method, and in Section 2.3.2 we will outline the different kinds of rule-induction methods.

2.3.1 Instance-Based Learning

IBL (Aha et al., 1991) is based on the idea of letting the examples themselves form the implicit representation of the target concept. The simplest case is the nearest neighbour (NN) (or k-nearest neighbour [k-NN]) algorithm, which simply stores all the examples in a training set. NN classifies a new instance by predicting that it has the same class as its nearest stored instance (or the majority class of its k-nearest stored instances), according to some similarity metric. The best value of k for a given application is difficult to predict, and is typically determined via cross-validation.

The performance of IBL depends critically on the similarity metric used. For numeric attributes (e.g., age, price, and weight), Manhattan distance is a natural candidate; thus, the distances between the two values are, simply, the absolute difference between them. However, different attributes may not have the same range, so two distant values may appear to be near to each other because of a small value range. The obvious solution is to normalize the attribute values as follows:

$$Normalize\ (xi) = \frac{x_i - x_{min}}{x_{max} - x_{min}} \quad (1)$$

Where

x_i: *is the ith value of the attribute x,*

x_{max}: *is the maximum value of the attribute x,*

x_{min}: *is the minimum value of the attribute x.*

If the attributes are nominal (e.g., colour, shape), we can use the value difference metric (Stanfill & Waltz, 1986). Using this metric, two values are considered to be similar if they result in similar classifications. It finds the distance between two values for a specific attribute via:

$$\delta(x,y) = VDM(x,y) = \sum_{h=1}^{C} |\big(p(c_h|x) - p(c_h|y)\big)|^q = \sum_{h=1}^{C} |\left(\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}}\right)|^q \quad (2)$$

Where

C is the number of classes,

$N_{a,x}$ is the number of instances in the training set, T, that has value x for attribute a,

$N_{a,x,c}$ is the number of instances in the training set, T, that has value x for attribute a and class C,

q is a constant, and

$p(c_h|x)$ is the conditional probability that the output class is (c), given that attribute (a) has the value x.

If there are n attributes, E1 = (e_{11}, e_{12}... e_{1n}) is the first instance and E2 = (e_{21}, e_{22}... e_{2n}) is the second instance; then, the distance between the two instances is measured using:

$$\Delta(E_1, E_2) = \sqrt{\sum_{i=1}^{n} \delta^2(e_{1i}, e_{2i})} \quad (3)$$

NN is conceptually simple and "learns" very quickly because it needs only to read the training set without much further processing. However, its output (concept description) is difficult for humans to understand, takes a long execution time (during classification) and is sensitive to irrelevant attributes because these attributes will contribute to computing the distance between two examples, and may "swamp" out the relevant component. It is also sensitive to noisy instances, because when such instances are stored they create a region around themselves, which consists of

all the examples that consider them as one of the k-NN, so we have to choose the value for the k parameter carefully when using the NN algorithm (in Section 4.6.2, we will explain our k value selection). Finally, the NN algorithm may have large memory requirements (after training).

One solution to NN's sensitivity to irrelevant attributes is to remove it before instances are stored. Several methods have been proposed whereby this can be achieved, of which the most widely used are forward sequential selection and backward sequential selection (Domingos, 1997; AlBalas, 2000). On the other hand, there are several methods that focus on reducing the size of the stored set of instances while trying to maintain, or even improve, predictive accuracy.

2.3.2 Rule-Induction Algorithms

Rule induction (Clark & Niblett, 1989; Domingos, 1997) is another paradigm for learning algorithms. Throughout this thesis, we will consider rule-induction methods that learn a set of propositional rules where the target concept is represented as a set of "if... then..." rules. Each rule consists of an antecedent (or body of rule) and a consequent. The consequent represents the predicted class; the antecedent part is composed of a conjunction of conditions, each involving one attribute. If the attribute is nominal, this condition is usually an equality test. Some algorithms use the negation and the disjunction of values. If the attribute is numeric, the condition is an inclusion test in a one-sided interval. A rule is said to cover an example, or the example is said to satisfy it, if all conditions in the body of the rule are true for the example.

There are many rule-induction algorithms. Among them are AQ (Michalski et al., 1986; Cervone et al., 2001; Michalski & Kaufman, 2001), CN2 (Clark & Niblett, 1989; Clark & Boswell, 1991) and Repeated Incremental Pruning to Produce Error Reduction (RIPPER) (Cohen, 1995).

All these algorithms employ the same general method that was used for the first time in the AQ algorithm. AQ21 is the most recent addition to the AQ family (Wojtusiak et al., 2006). The AQ family and some of the algorithms mentioned above have been improved from time to time. They employ a set of covering, or "separate and conquer", algorithms, because they form the class definition by building a rule that covers many positive examples, and then separate out the covered positive examples and start again. However, since they extract rules and then remove the covered examples from a training set of examples, fragmentation has been one of the problems of such algorithms caused by the existence of some rules covering a small number of instances.

In the search for the best rule covering the set of positive examples, we add an antecedent that maximizes certain heuristics. The heuristic is usually a function of the number of positive examples covered by the rule, and the number of negative examples covered by the same rule. We can use the beam search strategy to search for the best rule (Clark & Niblett, 1989), and maintain a list of ***b*** best rule antecedents found so far. In each step, specialization of those antecedents with each possible condition is attempted, and the best ***b*** antecedents are selected to continue the search until no better antecedents can be found with respect to the heuristic used. Finally, the best rule antecedent is selected and all examples covered by the selected rule are removed from the training set.

The choice of evaluation heuristic H for the rule is most important to the performance of the "separate and conquer" algorithm. AQ algorithms use "apparent accuracy":

$$H(e_+, e_-) = \frac{e_+}{e_+ + e_-} \tag{4}$$

Where

e$_+$ *is the number of positive examples covered by the rule .*

e$_-$ *is the number of negative examples covered by the rule.*

The CN2 system (Clark & Niblett, 1989) originally used the entropy of the rule:

$$\text{Entropy} = -\sum_{i=1}^{n} (P_i \, log_2(P_i)) \tag{5}$$

Where

n is the number of classes represented in the training set.

P_i *is the probability distribution of covered examples that have predicted class =* $class_i$ *among all covered examples.*

The lower the entropy, the better the rule. This function prefers rules that cover a large number of examples of a single class and few examples of other classes, and thus score well on the training data when used to predict the majority class covered.

Both the entropy and apparent accuracy favour overly specific rules (those that cover a single positive example with no negative examples), and they achieve their maximum value with a rule covering a single example. This can be overcome by using Laplace accuracy (Clark & Boswell, 1991):

$$H(e_+, e_-) = \frac{1+e_+}{C+ e_+ + e_-} \tag{6}$$

Where

C is the number of classes.

When we classify a new unseen example, it is matched against the set of rules. If there is only one rule covering the example, the class of the new example will be the rule's class. If there is no rule, then we can use a default rule (which usually predicts the class that is the most frequent in the training set). However, if there are many rules covering the example, we have two solutions. The first is to order the rules in a decision list (according to the Laplace or apparent accuracy), and select the first rule that covers the example (Rivest, 1987). The second solution is to let each rule vote and then select the class with the highest number of votes (Clark & Boswell, 1991).

The RULE Extraction System (RULES) is a family of simple inductive learning algorithms inspired by ideas from both AQ and CN2. The RULES family is different from the other algorithms in that it does not induce rules on a class-per-class basis, but instead considers the class of the selected seed example as the target class (Shehzad, 2009). It then attempts to induce rules that cover as many examples of the target class as possible using the rule evaluation function. At present, the RULES family has extended to Rules-7 (Pham, 2012). Among members of the RULES family, Rules-5 is a noteworthy, simple, but efficient algorithm. RULES-5 also employs a more efficient search mechanism, as well as a new post-pruning technique (Pham & Bigot, 2003) in order to handle noisy data.

Other rule-induction methods unify the rule induction with IBL. Rule Induction from Set of Examples (RISE) (Domingos, 1994) is one approach to induction that attempts to tackle some disadvantages of IBL and rule induction. The basic characteristic of RISE is that rules and instances are treated uniformly; thus, an instance is simply a rule, and the rule's extension becomes a set of instances most similar to that rule.

There are rule-induction methods that investigate the application of pruning methods during rule generations. Fürnkranz and Widmer (1994) proposed a novel learning algorithm called Incremental Reduced Error Pruning (IREP). IREP prunes each individual rule right after it has been generated: after learning a rule from the growing set, a condition is deleted in a greedy fashion until any further deletion would decrease the accuracy of this rule in the pruning set. The resulting rule is added to the concept description and all positive and negative instances covered by the generated rule are removed from the training "growing and pruning set". Cohen (1995) also introduced some improvements to IREP that enhance its performance. Three modifications are made to the IREP algorithm:

1. An alternative metric for assessing the value of the rules in the pruning phase of IREP,
2. A new heuristic for determining when to stop adding rules to a rule set, and
3. A post-processing of the generated rules that optimize a rule set in an attempt to more closely approximate IREP.

This algorithm that produces a new optimized rule set is called RIPPER (Cohen, 1995).

Other rule-induction methods try to solve drawbacks via other induction methods. PRISM (Cendrowska, 1987) is a rule-induction method based on ID3 in selecting the attributes for the induced rule set. This algorithm is simple and easy to understand. Cendrowska's original PRISM algorithm selects one class as the target class (TC) at the beginning, and induces all rules for that class. It then selects the next class as TC and resets the whole training data to its original size, and induces all rules for the next TC. This is repeated until all classes have been selected as TC. Figure

1 shows the pseudo code for the PRISM algorithm, where p (c_h | a_x) is the conditional probability that the output class is (c_h), given that attribute (a) has the value a_x.

If the training set contains instances of more than one class (c_h), then for each class c_h

Step 1: Calculate the probability of occurence ,$p(c_h|a_x)$ of class c_h for each attribute - value pairs a_x.

Step 2: Select a_x for which $p(c_h|a_x)$ is a maximum and create a subset of the training set comprising all the instances which contain the selected a_x.

Step 3: Repeat Step 1 and Step 2 for this subset until it contains only instances of class c_h. The induced rule is a conjunction of all the attribute-value pairs used in creating the homogeneous subset.

Step4 : Remove all instances covered by this rule from the training set.

Step 5: Repeat Step 1-4 until all instances of class c_h have been removed

Figure 1. Pseudo-code for PRISM algorithm.

PART (Eibe & Ian, 1998) induces a decision list. This algorithm can be viewed as a combination of C4.5 and RIPPER, and attempts to avoid their respective problems. Unlike both C4.5 and RIPPER, it does not need to perform global optimization to produce accurate rule sets. It adopts the separate and conquer strategy in that it builds a rule, removes the instances it covers, and continues creating rules recursively for the remaining instances until none are left. It differs from the standard approach in the way in which each rule is created. In essence, to make a single rule, a pruned decision tree is built for the current set of instances; the leaf with the largest coverage is made into a rule, and the tree is discarded.

ACO has been applied for rule induction in the Ant-Miner algorithm (Parepinelli et al., 2002). The Ant-Miner algorithm was developed by simulating the foraging of real ants, so it is a good idea to think about the problem as a search for the best path through a graph, where the nodes represent the partial solution and the edges represent the translation between these partial solutions. The edges are associated with measurements that qualify the selected partial solutions. When

applying the Ant-Miner algorithm to classification rule induction, the basic element of a solution is an attribute term. An attribute term, $term_{ij}$ is in the form of $A_i = V_{ij}$, where A_i is the ith attribute and V_{ij} is the jth value of domain A. Therefore, we can consider the classification rule induction problem as a graph, with nodes representing attribute terms and edges modelling the quality of the attribute terms. A complete path is a constructed rule. The quality of the path is assessed by a global fitness function, while the quality of the node is evaluated by a heuristic value and a pheromone level value associated with the node.

Table 1 presents an enumeration of rule induction methods reviewed in this section. The name, reference and key features are provided for each rule induction method.

2.4 An overview of pruning algorithms

This section will overview different kinds of pruning methods related to our work. As our concern is to precede rule induction with instance-reduction methods, we will introduce different methods for instance pruning that aim to obtain representative training sets with lower sizes compared to the original one, and with similar or even higher predictive accuracy for new incoming instances. Moreover, we will overview different ways of pruning rule-induction methods and the motivation for carrying out that pruning.

2.4.1 Instance Pruning

Instance pruning aims to prune the original training set to get a smaller subset of it. Searching for a subset, S, of instances to keep, instead of the original training set, T, can proceed in a variety of directions, including incremental, decremental, and batch (Wilson & Martinez, 1997).

Rule Induction method	Reference	Key feature(s)
PRISM	Endrowska, 1987	Based on ID3 in selecting the attributes for the induced rule set.
CN2	Clark & Niblett, 1991	Incorporates ideas from both Michalski's (1986) AQ and Quinlan's (1983) ID3 algorithm.
IREP	Fürnkranz, 1994	Integrates reduced error pruning with a separate and conquer rule learning algorithm.
RISE	Domingos, 1994	Proceeds by gradually generalizing rules, starting with one rule per example.
RIPPER	Cohen, 1995	Optimized version of IREP. This algorithm is especially more efficient on large noisy datasets. It builds a set of rules that identify the classes while minimizing the amount of error.
PART	Eibe & Ian, 1998	Combination of C4.5 and RIPPER. This algorithm extracts rules faster than decision trees algorithm.
Ant-miner	Parepinelli, Lopez & Freitas, 2002	An Ant Colony Optimization algorithm for rule discovery in database.
RULEs-5	Pham & Bigot, 2003	The first RULES version that handles continuous attributes without discretization.
AQ21	Wojtusiak et al., 2006	It can discover different types of regularities in data, and can generate an optimized collection of alternative models from the same data.
RULEs-7	Pham, 2012	An extension of RULES-6

Table 1: Review some of rule induction methods.

Incremental methods begin with an empty subset, S, and add instances (from training set T) to S if it fulfils some criteria. Thus, if new instances are made available later (after training is completed), they can continue to be added to S according to the same criteria. Incremental methods are sensitive to the order of presentation of the instances. Condensed nearest neighbour (CNN) (Hart, 1968) and selective nearest neighbour (Ritter et al., 1975) are examples of incremental methods. On the other hand, decremental methods begin with all the

instances in the training set (i.e., T = S), and search for instances to remove; they are often computationally more expensive than incremental methods. Reduced nearest neighbour (RNN) (Gates, 1972) and the decremental reduction optimization procedure (DROP 1-5) (Wilson, & Martinez, 2000) represent examples of decremental methods. Finally, batch methods, like decremental methods, begin with all instances in a training set; however, before they remove any, they find all instances that meet the removal criteria and then remove them all at once (Tomek, 1976). Batch methods also suffer from increased time complexity compared with incremental methods. In our experiments, we will use decremental and batch methods because, in comparison to incremental methods, they have been shown to give rise to higher predictive accuracies (Wilson & Martinez, 2000).

Instance-reduction methods can be categorized as retaining either internal or border instances:

- **Border instances (condensation approach)**: The intuition for retaining border instances is that internal instances do not affect the decision boundaries, and can thus be removed with relatively little effect on classification. Several well-known methods belong to the condensation approach and the algorithms that offer the best performance, including:
  - CNN (Hart, 1968): Hart was the first to propose a method for reducing the size of stored data for the NN decision rule.
  - RNN (Gates, 1972) is an extension of the CNN rule. The RNN algorithm uses the CNN resulting set and removes every instance for which deletion does not cause misclassification of another instance in the initial set.
  - The Fast Condensed Nearest Neighbour rule (Angiulli, 2005) is a scalable algorithm on large multidimensional datasets.

- TRKNN (Fayed & Atiya, 2009): This reduces the computational requirement to classify prototypes using the k-NN when the sets of data are large. The aim of this approach is to eliminate instances that cause unnecessary calculations and do not contribute to improving the classification.
- The Class Boundary Preserving Algorithm (Nikolaidis et al., 2011) is a multistep method for pruning the training set.
- DROP 1-5 (Wilson & Martinez, 2000) is a series of six algorithms for set reduction based on the k-NN algorithm, where each algorithm improves the previous one.

- **Internal instances (edition approach):** The intuition for retaining internal instances is that removing border instances should remove noisy instances. The effect obtained is related to the improvement of generalization accuracy in test data, although the reduction rate obtained is lower than the rate achieved by condensation approaches, since there are fewer border instances as compared to internal instances (Gadodiya & Chandak, 2013). Few edition methods have been proposed in comparison to condensation methods. The main reason for this is that the first edition method, edited nearest neighbour (ENN), obtains good results in conjunction with k-NN (Gadodiya & Chandak, 2013) (Grochowski & Jankowski, 2004). An extension of ENN is the RENN (Repeated ENN) method which repeatedly applies ENN until all instances in training set have the same class that the majority of their k Nearest Neighbours. Another variant of ENN is the AllKnn method (Tomek 1976). In Vázquez et al. (2005) a method for instance selection is proposed, which consists in applying ENN but using the probability of belonging to a class instead of the k-NN rule.

In our experiments, we focus on methods that obtain a representative training set with a lower size compared to original one, and with similar or even higher classification accuracy for new data. Thus, we choose three reduction algorithms that perform well in reducing the number of instances (Wilson & Martinez, 1997), and provide good results before applying neural network learning (El Hindi & Al Akhras, 2009) (Sun & Chan, 2014). These algorithms eliminate border instances, which tend to be noisy, or difficult to learn and untypical. Each algorithm is discussed in further detail below.

2.4.1.1 The edited nearest neighbour algorithm

ENN (Wilson, 1972) is a decremental algorithm that removes an instance if it does not agree with the majority of its k nearest neighbours (with k = 3). This removes noisy instances, as well as near border instances, and retains all internal instances. Figure 2 shows the pseudo code for the ENN algorithm.

2.4.1.2 AllKnn

AllKnn (Wilson & Martinez, 1997) is a batch algorithm that makes k iterations. At the ith iteration, it flags as bad any instance that is not classified correctly by its i nearest neighbours. After completing all iterations, the algorithm removes all instances flagged as bad. Figure 3 shows the pseudo code for AllKnn algorithm.

```
For each instance (i)
    If (the class of instance (i) <> the majority class of K nearest neighbors)
        Remove the instance
```

Figure 2. Pseudo-code for ENN algorithm.

```
Oldk = k
For each instance (i)
   For K=1 till oldk
        If (the class of instance (i) <> the majority class of k nearest neighbors)
             Flag the instance for pruning
Remove each flagged instance
```

Figure 3. Pseudo-code for AllKnn algorithm.

```
Let T be the initial set of instances
Measure the distance of each instance in T from its nearest enemy (instance
with different class). Sort the instances in T by their distance, in ascending
order.
Let S = T.
 For each instance P in S:
      Find P.N1..k+1, the k+1 nearest neighbors of P in S.
      Add P to each of its neighbors' lists of associates.
 For each instance P in S:
      Let with= # of associates of P classified correctly with P as a neighbor.
      Let without= # of associates of P classified correctly without P.
      If without >= with
           Remove P from S.
           For each associate A of P
                Remove P from A's list of nearest neighbors.
                Find a new nearest neighbor for A.
                Add A to its new neighbor's list of associates.
 Return S.
```

Figure 4. Pseudo-code for DROP5 algorithm.

2.4.1.3 DROP5

DROP5 (Wilson & Martinez, 2000) is a decremental algorithm that removes an instance, "S", if at least as many of its associates (i.e., instances that have "S" on their NN list) are classified correctly without it. This algorithm removes noisy instances, because a noisy instance, "S", usually has associates that are mostly of a different class, and such associates will be at least as likely to be classified correctly without "S".

First, the algorithm considers removing the instances that are closest to their nearest enemy (i.e., instance from a different class), and proceeds outward. By removing points near the decision boundary first, the decision boundary is smoothed. Figure 4 shows the pseudo code for the DROP5 algorithm.

Another method related to the associate set was proposed by Brighton and Mellish (2002), this method is the Iterative Case Filtering algorithm (ICF), based on the Reachable(S) and Coverage(S) sets which are the neighbour and associate sets respectively. ICF discards instance(S) If |Reachable(S)| > | Coverage(S)| which means that some instances in training set (T) can classify instances similar to (S) without considering it in the training set; as initial step, ICF applies ENN. C-Pruner (Zhao et al, 2003) is another method based on the Reachable (S) and Coverage (S). In this method, the Coverage (S) concept only considers the associates with the same class as instance (S) in order to discard instances in the same class. Before discarding an element, this technique determines whether an instance is noisy, superfluous or critical. In this context, an instance is critical when its deletion affects the classification of other instances; in particular, this method discards either noisy or superfluous (but non-critical) instances. When |Coverage (S)| < |Reachable (S)| then "S" is considered as noisy; "S" is superfluous when it is correctly classified by Reachable (S) (Olvera-Lopez et al., 2010).

2.4.2 Rule Induction Pruning

The main weakness with rule learning systems is that they often scale relatively poorly with the sample size of a training set, particularly in the context of noisy data (Cohen, 1993). This is a critical problem due to the prevalence of large, noisy datasets in real-world applications. A variety of methods has been proposed to prune the produced rule sets, and can be categorized as follows:

- **Pre-pruning** These algorithms either use heuristics (i.e., stopping criteria) to relax the constraint that completely satisfies the training instances, such as CN2 (Clark & Niblett, 1989) and FOSSIL (Fürnkranz, 1994), or reduce the number of training examples before

generating a classifier; the hope is that using fewer training examples will produce fewer rules.

- **Post-pruning** This takes a rule set that is consistent with the training instances and removes rules and conditions that do not reflect true regularities of domain, such as the Reduced Error Pruning (REP) algorithm (Brunk & Pazzani, 1991) and the GROW algorithm (Cohen, 1993).
- **Integration pre-pruning and post-pruning.** Instead of learning the entire rule set and then conducting the pruning, this category prunes a single rule right after the rules have been learned, akin to IREP (Fürnkranz & Widmer, 1994), RIPPER (Cohen, 1995), and Simple Learner with Iterative Pruning to Produce Error Reduction (SLIPPER) (Cohen & Singer, 1999).

2.4.2.1 Pre-pruning

In a rule-induction process, the more conditions we have in the rule, the fewer instances it can cover. Thus, some algorithms employ stopping criteria for noise handling; in addition, to avoid overfitting, there should be a trade-off between covering and accuracy. The pre-pruning for rule inductions can be conducted in two ways:

1. Condition reductions: This can be achieved by pruning each rule independently in the course of learning by using a heuristic to determine when to stop adding conditions to the rule.
2. Rule reductions: These aim to reduce the number of rules produced by either decreasing the instances used to build the rules, or removing the most specific produced rules (which should be those that cover the noisy instances).

2.4.2.2 Post-pruning

While pre-pruning algorithms try to avoid overfitting during rule generation (or before applying the rule-induction method), the post-pruning approach initially ignores the problem of overfitting and learns a complete and consistent rule set. It then estimates the quality of this rule set using some quality measurement (usually apparent or Laplace accuracy). If the accuracy can be improved by simplifying the rule set, then this will be repeatedly done until any further simplification would harm the quality of the rule set.

The post-pruning can be done either by checking the effect of removing the condition from each rule and investigating the effect of this removal, or by considering the effect of removing the whole rule from the rule set and checking its effect on the accuracy.

REP is the most common method used for post-pruning. Pagallo and Haussler (1990), Weiss and Indurkhya (1991), and Brunk and Pazzani (1991) employed straightforward adoption of REP to separate and conquer rule-learning frameworks. Initially, the training set is split into two subsets, a "growing set" and a "pruning set". Then, in the first phase, REP learns the concept that covers all positive and no negative examples from the growing set (no attention is paid to the noise in the data). The resulting rule set is then repeatedly simplified by deleting conditions and rules from the set until any further deletion would result in a decrease of predictive accuracy as measured on the pruning set. A variant of REP can employ a variety simplifications to the rule set, such as deleting each condition of a rule, deleting final sequences of conditions[1] (Cohen, 1993), or finding the best replacement condition (Weiss & Indurkhya, 1991).

[1] For example, the "if w and x and y and z then class = a" might be simplified to either "if w and x and y then class = a" or "if w and x then class = a" or "if w then class = a".

REP for rules usually does improve generalization performance on noisy data (Pagallo & Haussler, 1990), and its search strategy can be regarded as bottom-up as it performs pruning on the resulting rule set. However, it has several shortcomings (Fürnkranz & Widmer, 1994):

- Complexity: REP's time complexity has been shown to be O (n^4) for noisy data, where n is the number of examples (Cohen, 1993).
- Pruning of conditions in a "separate and conquer" rule will affect all subsequent rules. As pruning conditions from a rule can only generalize the concept – i.e., increase the set of covered examples – a post-pruning algorithm has no means for adjusting the subsequent rules to this new situation. Thus, the learner may be deceived, because the set of examples that remain uncovered by the unpruned rules at the beginning of learning may yield a different evaluation of candidate conditions for subsequent rules compared to the set of examples that remain uncovered by the pruned versions of these rules.
- Generated rules are simplified so that the predictive accuracy on the pruning set will be maximized, but in noisy domains REP will have to do a lot of pruning, and therefore has ample opportunity to get caught in the local maximum.

GROW is introduced to solve some of the drawbacks of the REP algorithm, and replace the bottom-up search of REP with a top-down approach. GROW initially finds a rule set (R0) by overfitting the growing set, then each rule, $r_i \in$ (R0), is taken, and repeatedly simplified in such a way that the error on the growing set goes up the least; the result will be a series of generalizations, $r_{i,1}$, …..,$r_{i,k,}$ of original the rule, r_i. All the generalizations in this series are then added to the rule

set (R0). After the initial rule set (R0) has been expanded, we start with an empty rule set and add rule $r_{i,j}$ from (R0), which improves the predictive accuracy the most on the pruning set. Ties are broken by choosing the smaller rule. It has been experimentally confirmed that this results in significant gain efficiency on learning time, along with a slight gain in accuracy (Cohen, 1993).

Another methodology for post-pruning is to use Laplace accuracy as a measurement to decide either to remove or retain the produced rules. Sort then Select Rule Reduction (SSRR) (Othman & El Hindi, 2004) concentrates on retaining rules with the highest Laplace accuracy. For each class, it chooses a rule from the produced rule set with the highest Laplace accuracy. Then, it incrementally augments the pruned rule set with all necessary rules in order to make the same classification derived from the original produced rules on the training set. The rules are tried in order, with the one with best Laplace accuracy first. It has been shown (Othman & El Hindi, 2004) that SSRR slightly improves the accuracy in some datasets while achieving good reduction in produced rules.

Pham et al. (2004) introduced another method for reducing the generated rules by merging them in order to handle expected noise. The main objective of this merging is to create new, more general rules, with a consistency level equal to or higher than a specified value Th (Thresh hold). Th is a user-defined parameter equal to (1 – expected noise level [NL]). This method works by taking one rule at a time from the generated rules (RSet), called the rule to merge (R2M). This rule is merged with each of the other rules for the same class within the RSet. If the consistency measurement of the best resulting rule from these mergers is equal to or higher than Th, then it is added to the RSet. Otherwise, if the consistency of the best rule is lower than Th, the algorithm

stores R2M in a new rule set (NEW_Rset) and removes it from the RSet. If there are still rules within the RSet that are not processed, the algorithm takes one of them as R2M and repeats the procedure. However, within this approach the NL is specified by the user.

2.4.2.3 Integration pre-pruning and post-pruning

While post-pruning first grows a complete concept description and prunes it thereafter, Fürnkranz and Widmer (1994) proposed a novel learning algorithm called IREP. IREP prunes each individual rule right after it has been generated: after learning a rule from the growing set, the condition is deleted in a greedy fashion until any further deletion would decrease the accuracy of this rule in the pruning set. The resulting rule is added to the concept description and all positive and negative instances covered by the generated rule are removed from the training set "growing and pruning set". The remaining training set is then split again to form a new growing and pruning set. When the accuracy of the pruned rule is below predictive accuracy of the empty rule (rule with body fail), the rule will not be added to the concept description and IREP returns the learned rule set. The accuracy of an empty rule is N/(N + P), and the accuracy of the pruned rule is (p + (N - n)) / (P + N), where p (n) is the number of positive (negative) examples covered by the rule from a total of P (N) positive (negative) examples in the current pruning set.

IREP solves some of the drawbacks of the REP method, such as the efficiency on learning time and the effect of pruning on the subsequent rule by completing the pruning on each rule and removing covered examples before the subsequent rules are learned. In addition, IREP uses a top-down, instead of a bottom-up, search. Nevertheless, IREP is flawed since whenever the pruned rule's accuracy is not above the accuracy of the empty rule, no more rules will be learned. In addition, IREP is prone to overgeneralization if the accuracy is not estimated correctly.

Experimentally, it seems that GROW outperforms REP, while IREP is better than REP and GROW whenever a fairly general concept has to be found, whereas REP is appropriate when the underlying concept is specific (Fürnkranz & Widmer, 1994).

Cohen (Cohen, 1995) introduced a modification to IREP that allows it to handle multiple classes by placing them in increasing order of prevalence. IREP is then used to find a rule set that separates certain class Ci from the remaining classes. Next, all instances covered by a learned rule set are removed from the dataset. Cohen also introduced some improvements to IREP that enhance its performance. This included three modifications to the IREP algorithm:

1. An alternative metric for assessing the value of the rules in the pruning phase of IREP;
2. A new heuristic for determining when to stop adding rules to a rule set; and
3. A post-process to generate the rules that optimize a rule set in an attempt to more closely approximate IREP.

This is the RIPPER algorithm. RIPPER significantly improves the generalization performance over IREP (Cohen, 1995).

Cohen and Singer (1999) introduced another algorithm similar to the IREP and RIPPER algorithms, called SLIPPER. However, SLIPPER does not remove examples covered by a new rule; instead, it uses boosting to reduce the weight of these examples.

Using the SLIPPER algorithm, a single rule is generated using one subset of the data (the growing set), and the rules are then pruned using the other subset (the pruning set). The ad hoc

metrics used to guide the growing and pruning of rules are replaced with metrics based on formal analysis of boosting algorithms, specifically Freund and Schapire's (1997) AdaBoot, which employs confidence-rated predictions (Schapire & Singer, 1998).

Other efforts have been applied to generate faster learning. IREP++ (Dain et al., 2004) is one such initiative. It starts by using RIPPER and attempts to develop an algorithm to achieve comparable accuracy by functioning more quickly. The speed improvements are achieved by making several changes to the RIPPER algorithm, including better pruning metrics, a novel data structure, and more efficient stopping criteria. IREP++ (Dain et al., 2004) has been shown to be slightly more accurate than RIPPER, and functioning faster. In addition, IREP++ learns fewer generated rules.

In chapter 5, we compare different rule induction methods based on some important characteristics and choose the methods to be used in our experiments accordingly. Furthermore, we think that the technique of preceding rule induction with instance reduction can achieve a good result with rule induction algorithms which do not use pruning.

2.5 Summary

Rule induction is an attractive learning method, as rules become much more transparent and easier to interpret compared to other induction methods. There are different kinds of rule-induction method algorithms that vary in terms of the type and direction of search. Nevertheless, these methods can suffer when using noisy datasets. Furthermore, most rule-based systems tend to induce quite a large number of rules, making the solution difficult to understand.

Pruning is a common framework to avoid the problem of overfitting noisy data. Rule-induction methods can be entail different types of pruning, including pre-pruning (e.g., CN2; Clark & Niblett, 1989) and FOSSIL (Fürnkranz, 1994), post-pruning (e.g., REP algorithm; Brunk & Pazzani, 1991), and integration pre-pruning and post-pruning (e.g., RIPPER; Cohen, 1995). On other hand, training set can be reduced using different instance reduction methods and retain subset of it. In this thesis, we are investigating different instance-reduction methods to precede rule-induction approaches.

Chapter 3: Literature Review: Ant Colony Optimization

This chapter presents a review of ACO, which is a metaheuristic proposed as a method for solving hard problems, and inspired by the behaviour of real ants.

ACO algorithms are considered to be part of swarm intelligence, which is the study of computational systems inspired by "collective intelligence". Collective intelligence emerges through the cooperation of large numbers of homogeneous agents in the environment.

This chapter is organized as follows. Section 3.1 presents a formal description of the ACO metaheuristic. Section 3.2 overviews the most popular variants of ACO and gives examples of their application. Section 3.2.1 explains the Ant-Miner algorithm and Section 3.2.2 describes how ACO is applied to feature selection.

3.1 Ant Colony Optimization overview

ACO is a branch of the newly developed form of AI called swarm intelligence. Swarm intelligence is a field that designs algorithms inspired by the collective behaviour of social insects and other animal societies (Bonabeau et al., 1999).

The potential benefits of imitating social insects' structural models and behaviour in designing solutions to a problem include:

- Robustness, because a colony as a whole may succeed where an individual may fail.

- Flexibility, in terms of adaptation to changing environments.

In groups of insects that live-in colonies, such as ants and bees, individuals can only accomplish simple tasks on their own, while the colony, working cooperatively, can perform complex tasks. Ants also have the ability to find the shortest path from their nest to a food source. When a food source is first located, several ants may have taken several different paths to reach that food source. When an ant moves, it lays a chemical substance called a pheromone along its path. When foraging for food and taking it back to its nest it follows the path with the greatest amount of a pheromone laid upon it. Pheromone trails evaporate if more ants do not come along to reinforce it, and ants that find the shortest route to the food will arrive back at the nest quicker than others; thus, the greater the number of ants on one path, the greater the amount of pheromone on that path. When new ants seek to travel to the food source they then take the shortest route (since they are guided by the amount of pheromone on the path). It has been observed that all foraging ants eventually converge on the shortest route to the food source (Galea, 2002).

ACO is a technique used with combinatorial optimization problems, which consist of finding an optimal solution from a finite set of solutions. In many such problems, exhaustive search is not feasible. There are, however, some important differences between real and artificial ants (Socha, 2008):

- Artificial ants live in a discrete world – they move sequentially through a finite set of problem states.
- The pheromone update (i.e., pheromone depositing and evaporation) is not accomplished in exactly the same way by artificial ants as by real ones. Sometimes, the pheromone update

is carried out only by some of the artificial ants, and often only after a solution has been constructed.

- Some implementations of artificial ants use additional mechanisms that do not exist in the case of real ants. Examples include look-ahead, local search, backtracking, etc.

The first ACO was developed by Marco Dorigo and published under the name of Ant System (AS) in (Dorigo et al., 1996). The application was the traveling salesman problem (TSP), which is classified as NP-hard combinatorial optimization because the solution cannot be found in polynomial time. The goal of TSP is to find the shortest possible route through a set of connected (N) cities, with each city visited once and only once. The ants find a solution to the TSP by traversing a problem graph from one city to another, depositing pheromone, until they solve the TSP. During an iteration of the AS algorithm, each ant builds a tour comprising N steps:

For each ant, the transition from city i to city j depends on:

1. Whether the city has been visited.
2. The inverse of distance $\alpha = 1/d_{ij}$, which is based on local information and represents the heuristic desirability of choosing city j when in city i.
3. The amount of pheromone trail, μ_{ij}, on the edge connecting city i to city j.

ACO can solve any problem for which the following elements can be defined (Socha, 2004):

1. An appropriate problem representation is required that allows the artificial ants to incrementally build a solution using a probabilistic transition rule. The main idea is to model the problem of searching for the best path through a graph. In the TSP, we

have a fully connected weighted graph, where the set of nodes, N, represent the cities and the set of edges represent the connection between the cities, as shown in Figure 5.

2. A local heuristic provides guidance to the ant in choosing the next node for the path it is building. In the TSP, the local heuristic is the inverse of distance, which represents the heuristic desirability of choosing city j when in city i.
3. The probabilistic transition rule determines which node an ant should visit next. The transition rule is dependent on the heuristic value and pheromone level associated with an edge joining two nodes.
4. A fitness function determines the fitness of the solution built by an ant in the TSP, where the fitness function is the length of the whole path traversed by the ant.
5. A pheromone update rule specifies how to modify the pheromone trail laid along edges of the graph.

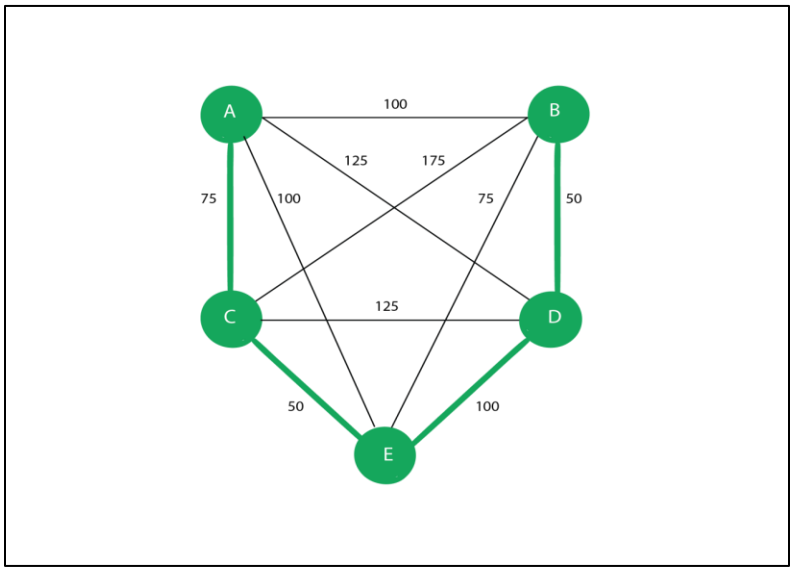


Figure 5. A weighted graph for TSP with five cities.

Figure 5 considers the weighted graph for five cities. An ant at city A has to choose probabilistically one of the four cities to visit. The pheromone, which will be referred to as (μ), is

initially set to be equal to 1. The initial visibility for each city is the inverse of the distance between the cities. Thus, the probability of choosing cities B, C, D and E are:

$P^1_{AB} = \frac{1/100}{\left(\frac{1}{100}\right)+\left(\frac{1}{75}\right)+\left(\frac{1}{100}\right)+\left(\frac{1}{125}\right)} = .242$

$P^1_{AC} = .323$

$P^1_{AD} = .194$

$P^1_{AE} = .242$

Therefore, the ant chooses to visit city C. Continuing the iteration, the ant completes the tour by visiting the cities E, B, and D, in that order, for a tour of length 250. After completing the tour, the ant lays pheromone along the path of the tour. The amount of pheromone added is equal to the inverse of the total length of the tour. Thus,

$\Delta\mu = \frac{1}{75+50+75+50} = .004$

Furthermore, the pheromone is decreased along all edges to simulate pheromone decay according to the pheromone evaporate rate. The pheromone for the edges in the path is therefore updated by (assuming that the pheromone evaporate rate is 0.1):

$\mu = (1- .1) + .1\ (.004) = .9004.$

And the pheromone for edges not in the path will be updated by,

$\mu = (1- .1) + .1\ (0) = .9.$

The new pheromone values along the edges of the graph in Figure 5 are given in Figure 6. The second ant, starting from city B, would complete the tour by visiting cities D, E, C, and A.

Now, the total length of the tour is 275, and hence this tour is taken as the shortest path so far (when the starting point is city B). The pheromone updates are completed as earlier. The algorithm continues to find the shortest path until the terminating condition is met, which is a certain number of solution constructions fixed at the beginning of the algorithm.

The great advantage of ACO over the use of exact methods is that the ACO algorithm provides relatively good results via a comparatively low number of iterations, and is therefore able to find an acceptable solution in a comparatively short time.

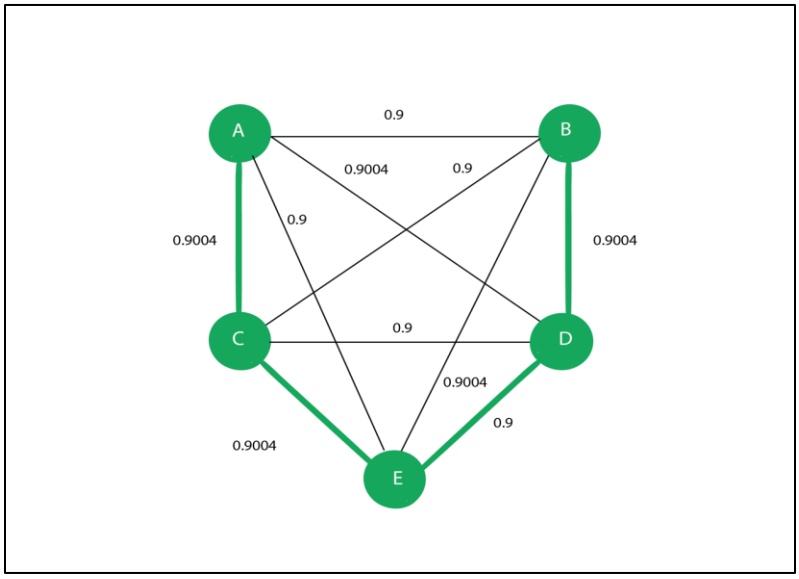


Figure 6. Pheromone values for the structure shown in Figure 5 after the first ant finishes a tour.

The ACO algorithms have also been applied to solve graph colouring (Costa & Hertz, 1997), job-shop scheduling (Colorni et al., 1994), sequential ordering (Gambardella & Dorigo, 1996), and vehicle routing (Bullnheimer, 1999). Results obtained with ant-based algorithms are often as good as those obtained with other algorithms.

3.2 Application of ACO to Classification Rule Induction

Parpinelli et al. (2002) were the first to propose using ACO to discover classification rules with the Ant-Miner system; they argued that an ant-based search is more flexible compared to traditional methods.

Ant algorithms simulate the foraging of real ants, so it is a good idea to think about the problem as a search for a best path through a graph, where the nodes represent the partial solution and edges represent the transition between these partial solutions. The edge labels are measurements that qualify the selected partial solutions.

In this section, we will review two interesting applications for ACO that have achieved good results in their field. We will demonstrate how ACO has been applied to solve these kinds of applications.

3.2.1 Ant-Miner Algorithm

Ant-Miner is an algorithm that incorporates the principles of ACO and rule induction. It starts with the full training dataset and then generates a set of ordered rules through iteratively finding a "best" rule that covers a subset of training data. It then removes the examples covered by the rule until the stop criterion is reached. Figure 7 shows the Ant-Miner algorithm proposed by Parepinelli et al. (2002).

```
TrainingSet = all training examples.
InducedRuleSet = [ ].
while (No. of uncovered examples in the TrainingSet >
MaxUncoveredExamples)
        t = 0;
        Repeat
                t = t+1;
                Ant(t) constructs a classification rule R.
                Prune the Rule
                update the pheromone of attribute terms;
        Until (t >=NoAnt) or (the same rule Rt has been constructed
MaxRulesCovarage times )
        select the best rule among all constructed rules;
        Add the best rule to InducedRuleSet;
        Remove training examples covered by the best rule from the
TrainingSet;
End While.
```

Figure 7. Ant-Miner algorithm.

When applying the Ant-Miner algorithm to classification rule induction, the basic element of the solution is attribute terms. An attribute term, $term_{ij}$, is in the form $A_i = V_{ij}$, where A_i is the ith attribute and V_{ij} is the jth value of domain A. Thus, an appropriate problem for ACO representation regarding the induction of classification rules is a graph whose nodes represent attribute terms. A complete path is a constructed rule, and the quality of the path is assessed by a global[2] fitness function. The quality of node is evaluated by heuristic value and pheromone level value associated with the node.

In Ant-Miner, each ant starts with an empty rule – i.e., with no term in its rule antecedent – and adds one term at a time. The choice of term to be added to the current partial rule antecedent depends on both the heuristic value (based on term entropy) and the pheromone level associated

[2] The scope of the global fitness function extends only to the current constructed rule, and not to full execution of Ant-Miner.

with each term. The entropy in Ant-Miner is computed for a specific attribute value, which is defined by:

$$H(C|A_i = V_{ij}) = -\sum_{c=1}^{k} P(c|A_i = V_{ij}) * log_2 P(c|A_i = V_{ij}) \quad (7)$$

Where:

c is the class attribute and k is the number of class values.

A_i is the ith attribute and V_{ij} is the jth attribute value of the ith attribute.

$P(c \mid A_i = V_{ij})$ is the probability of observing class c, conditional on observing $A_i = V_{ij}$.

The higher the entropy value of a term, the more uniformly distributed the classes are, and, thus, the smaller the probability that the current ant chooses this term to add to its partial rule. In Equation 8, $H(C|A_i = V_{ij})$ is subtracted from 1 because the ant is seeking a term that will distinguish between the class values, since it is building a classification rule. The entropy values are normalized using Equation 8 (Swaminathan, 2006).

$$Ħ_{ij} = (1 - H(C|A_i = V_{ij}))/(\sum_{l=1}^{a} x_l * (1 - \sum_{m=1}^{b_i} H(C|A_l = V_{lm}))) \quad (8)$$

a is the total number of attributes.

x_l is set to 1 if attribute A_l has not yet been selected; otherwise, it is set to 0.

b_i is the number of domain values for ith attribute.

The choice is biased towards terms that have relatively higher heuristic and pheromone values. Ant-Miner uses the transition rule in Equation 9, given an attribute–value pair; the

transition rule gives the probability of adding the attribute–value pair to the rule. The one with highest probability is added to the rule.

$$P_{ij} = (H_{ij} * \mu_{ij}(t))/(\sum_{l=1}^{a}(x_l * \sum_{m=1}^{b_i} H_{lm} * \mu_{lm}(t))) \quad (9)$$

Where:

P_{ij} is the probability that $term_{ij}$ is selected for addition to the current partial antecedent.

H_{ij} is the heuristic value associated with $term_{ij}$.

μ_{ij} (t) is the amount of pheromone associated with termij at iteration t.

a is the total number of attributes.

b_i is the number of domain values of the ith attribute.

x_l is set to 1 if attribute Al has not yet been selected; otherwise, it is set to 0.

Once an ant has stopped building a rule antecedent, a rule consequent is chosen. The rule consequent is assigned the class label of the majority class among the instances covered by the antecedent.

After constructing the rule, the artificial ant performs the rule-pruning procedure. The purpose of rule pruning is to increase the quality and comprehensibility of the built rule by simplifying the rule antecedent. The rule is pruned by removing one term at a time, until the rule cannot be improved further by removing another term. The term that most improves the quality of the rule is chosen. The pruning stops when there is no term whose removal would improve the rule quality. The accuracy of a rule consists of both accuracy among positive examples (called

sensitivity) and accuracy among negative examples (called specificity). Thus, the quality of the rule is defined by the following:

$$Q = \frac{TP}{TP + FN} * \frac{TN}{FP + TN} \quad (10)$$

Where:

TP, true positive, is the number of examples covered by the rule that belong to the class predicted by the rule.

FP, false positive, is the number of examples covered by the rule that belong to a class that is different from the class predicted by the rule.

FN, false negative, is the number of examples that are not covered by the rule, but that belong to the class predicted by the rule.

TN, true negative, is the number of examples that are not covered by the rule and do not belong to the class predicted by the rule.

Other variations from Ant-Miner use Laplace accuracy to estimate the constructed rule. It has been observed that Ant-Miner achieves better prediction accuracy when using Laplace accuracy, compared to using the sensitivity/specificity fitness function (Xuepeng, 2004).

Once rule pruning is complete, the pheromone levels are updated for the terms by increasing the pheromone for the terms that appear in the rule antecedent according to the rule quality given by:

$$\mu_{ij}(t+1) = \mu_{ij}(t) + \mu_{ij}(t) * Q. \quad (11)$$

Where

$\mu_{ij}(t)$ is the pheromone level of $term_{ij}$.

Q is the quality of the rule constructed.

The ant then normalizes the pheromone level of all terms (each pheromone level is divided by the sum of all pheromone levels), which reinforces the pheromone levels of the terms occurring in the rule antecedent and decreases the pheromone levels of other terms that are not selected in the rule.

The process by which an ant creates a rule is repeated for, at most, a predefined number of ants. However, the process may stop if the current ant has just created a rule that is exactly the same as a previous (maxRulesConverge -1) rule. MaxRulesConverge is a user-defined parameter for testing the convergence of ants, which simulates the convergence of real ants to the shortest path between a food source and their nest. The best rule created is added to the InducedRuleSet, the training set is appropriately reduced, and another run generates a best rule to cover more instances from remaining training instances.

Ant-Miner employs an ACO approach that provides a mechanism for conducting a global search that is more effective than those provided by traditional covering algorithms. It copes better with attribute interaction than greedy rule-induction algorithms do. Ant-Miner has been shown to have the best results compared to C4.5 and CN2 in terms of predictive accuracy and simplicity of rule sets (that is, the number of rules in the rule set), using six datasets from the University of California at Irvine (UCI) machine learning repository and a total number of ants equal to 3,000 (Parepinelli et al., 2002).

3.2.2 Feature Subset Selection

Feature subset selection is a method for selecting a subset of relevant features in order to generate good classifiers. The importance of the feature subset selection technique lies in its ability to provide a better understanding of the data and reduce the training time of the learning algorithm, because it helps in reducing the complexity of a given training set. It is computationally expensive and infeasible to implement feature subset selection via exhaustive evaluation of all possible subsets, especially as there may be thousands of features present in real-world datasets.

The feature subset selection algorithms can be categorized into two groups:

1. The filter approach, which is a feature subset selection technique applied independently of the learning algorithm. These methods apply some ranking over features. The ranking denotes how 'useful' each feature is likely to be for classification. a number of performance criteria have been proposed for filter-based feature selection such as fisher score (Duda et al., 2012), methods based on mutual information (Koller & Sahami, 1996) and ReliefF (Kira & Rendell, 1992).
2. The wrapper approach, wherein the evaluation criteria is tied to the learning algorithm. It considers feature subsets by the quality of the performance on a learning algorithm, which is taken as a black box evaluator. (e.g. Naïve Bayes or SVM) (Maldonado et al., 2014).

Shahzad (2010) proposed a hybrid feature subset selection using ACO and a decision tree (ID3) learning algorithm. This is a wrapper feature subset selection approach, in which each ant incrementally constructs a candidate solution that is a subset of the features in the dataset. These features are selected based on pheromone level and the heuristic value of each feature. The main idea of the proposed approach is to provide connected nodes graph (where N is the total number of features present in the dataset) (Shahzad, 2010). In the graph, the nodes represent the features

and links represent the connection between these features. Each ant constructs a solution by traversing a path in the graph. This path represents the selected features. After the ant has completed the feature selection, the fitness of the traversed path is calculated by running the ID3 algorithm using the selected features and estimating the predictive accuracy of the resulting classifier using 10-fold cross-validation (Kohavi, 1995). This estimate is the fitness function that is used to update the pheromone values. After termination of the algorithm, the feature set that has the best accuracy is returned as the solution (Shahzad, 2010).

Like the Ant-Miner algorithm, the ant starts with empty an subset. The ant uses two components to calculate the probability of moving from the present node to the next. The first component is the amount of pheromone present on the edge between $node_i$ and $node_j$, and the second is the heuristic value (e.g., the information gain) that describes the worth of a node. The probability with which the ant chooses node j as the next node, after it has arrived at node i, is shown in equation 12. Node j has to be in the set S of nodes that have not been visited.

$$P_{ij} = ((H_{ij})^{\alpha} * (\mu_{ij})^{\beta}) / (\sum_{k}^{S} (H_{ik})^{\alpha} * (\mu_{ik})^{\beta})) \quad (12)$$

Where

μ_{ij} is the pheromone level between $node_i$ and $node_j$.
H_{ij} is the heuristic value for choosing node j when arriving node i.
α, β are influencing factors of pheromone value and heuristic value, respectively.

Initially, the pheromone values in all edges between nodes are initialized with the same amount. In this way, no attribute is preferred over other attributes by the first ant. Equation 13 represents the initial pheromone for all attributes:

$$\boldsymbol{\mu}\ (t=1) = \frac{1}{N} \tag{13}$$

Where

N is the total number of features (attributes).

The heuristic value used to qualify each node is the information gain for each attribute. The information gain of attribute (A) is the reduction in entropy caused by partitioning the set of examples (S). When an ant selects the next node, it uses Equation 14 to calculate the information gain of a feature (attribute), where values (A) is the set of all possible values for attribute A and S_v is the subset of S for which attribute A has value v . Equation 14 is used to calculate the entropy:

$$Gain(S,A) = Entropy(S) - \left(\sum_{v}^{Values(A)} \frac{|S_v|}{|S|} * Entropy\ (S_v)\right) \tag{14}$$

Where

V is the set of all possible values for feature (attribute) A.
$|S_v|$ is the size of the subset from S, where attribute A takes the value v.
$|S|$ is the number of training instances.

To evaluate the worth of the selected set of features, Shahzad (Shahzad, 2010). used ID3 to build a classifier using the selected features subset and evaluate the generated classifier. He performed this procedure 10 times using 10-fold cross-validation, where the dataset is randomly divided into 10 equally sized subsets. Each of the subsets is used once for testing, and the remaining nine are used as the training set. Further, the fitness function is calculated by:

$$\boldsymbol{\mu}\ (t=1) = \frac{\boldsymbol{N_{corr}}}{\boldsymbol{N}} \quad (15)$$

Where

N is the total number of test instances.

N_{corr} *is the number of test instances correctly classified by the generated classifier.*

This fitness is calculated for each fold, and then averaged.

The pheromone rates are updated after the ant has completed its route. The amount of pheromone on each link occurring in the current feature subset selected by the ant is updated according to:

$$\boldsymbol{\mu}\ (t+1) = \left((\mathbf{1} - Þ) * \ \mu\ (t)\ \right) + \left(\left(1 - \frac{1}{1+fitness}\right) * \ \mu\ (t)\right) \quad (16)$$

Where

$\mu\ (t+1)$ *is the pheromone value between* $node_i$ *and* $node_j$.

Þ *is the pheromone evaporation rate.*

fitness is the quality of the current path constructed by the ant.

The pheromone on the other paths is updated by normalization.

Figure 8 presents the algorithm of the proposed feature subset selection based on ACO. The process continues until the stopping criteria are met. There are two stopping criteria:

1. Completion of a user-specified number of iterations (ants).
2. 10 consecutive ants returning the same set of features.

The experiments have been executed with 1,000 ants, using 32 datasets from the UCI machine learning repository with diverse characteristics. The experimental results reveal that the proposed feature subset selection method selects relevant features from datasets, causing an increase in the predictive accuracy on almost all of the datasets. Shahzad (Shahzad, 2010). compared the proposed approach with the naive Bayes approach for feature selection in terms of predictive accuracy after selecting the features using both approaches. Furthermore, the experimental results indicate that the proposed approach is better at finding features that improve predictive accuracy for the learned classifier.

```
Load the dataset
Calculate information gain (heuristic value) of each attribute
Generate a population of ants
Initialize the paramenters of ACO (No of ants, evaporation rate)
For each Ant, Generate a subset of features (S)
        Evaluate each feature subset S
        IF the fitness or accuracy is better than previous global best
                set the current subset S accuracy as global best accuracy
        END IF
        Update the pheromone value
        IF stopping criteria is met then stop the process
END FOR
```

Figure 8. Feature subset selection based on ACO.

3.3 Summary

ACO is a meta-heuristic algorithm that has been proven to be a successful technique and applied to different combinatorial optimization problems, such as rule induction (Ant-Miner algorithm) and feature subset selection. It is an attractive approach, and requires careful definition of five elements: appropriate problem representation, a local heuristic, the probabilistic transition rule, a fitness function, and the pheromone update rule.

Chapter 4: Experimental Framework

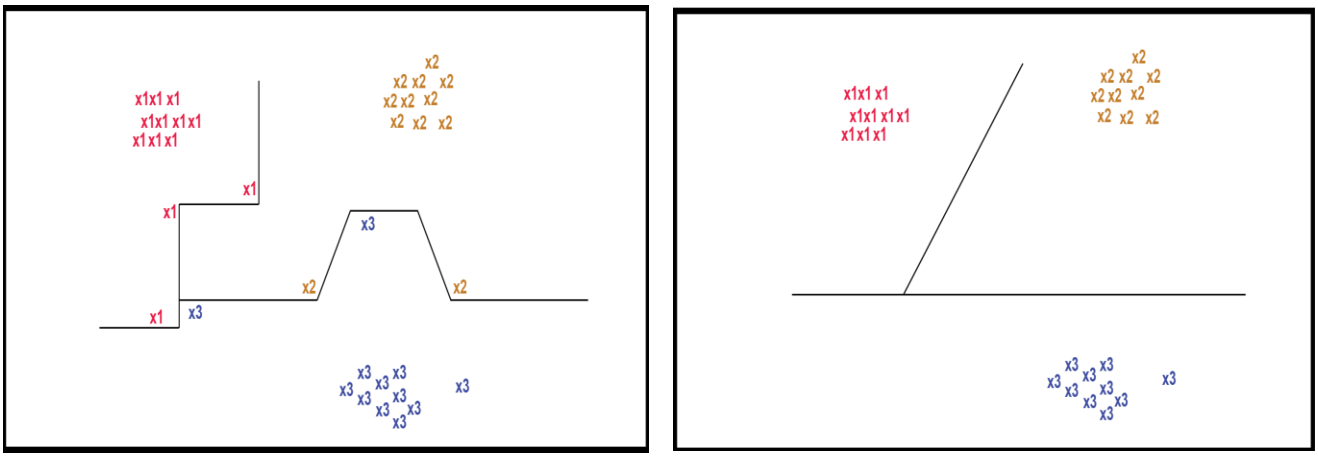
This chapter presents the methodology and experimental framework followed in this thesis to assess the effect of preceding rule induction with instance-reduction methods in terms of the number of generated rules and the predictive accuracy. In this methodology, a new algorithm for instance-reduction method based on ACO is implemented to achieve a good results when preceding rule induction methods.

Section 4.1 explains the problem we are interested in and the pre-processing framework that we are suggesting in our research. In Section 4.2, we explain the datasets used in this research for conducting the experiments, and Section 4.3 compares different rule-induction characteristics and specifies the rule-induction methods that we are interested in in our experiments. Section 4.4 presents how we estimate the prediction accuracy in our experiments. Then, Section 4.5 explains the evaluation measurement and the comparison methodology used in our experiments. Finally, Section 4.6 outlines the experimental setup and methodology we used with different datasets.

4.1 Problem Statement

This thesis is concerned with pruning rule induction by filtering out the border instances by applying instance-reduction methods before rule induction. We will apply three methods for instance reduction: (AllKnn, ENN and DROP5). These instance-reduction methods have been shown to perform well in the context of neural network learning (El Hindi & Alakhras, 2009). Moreover, ENN has been evaluated with ANN and shown that it is the most effective one compared to many other instance reduction methods (Sun & Chan, 2014).

Figure 9 illustrates the idea of eliminating near-border instances and how the decision boundary has been smoothed. Figure 10 explains the framework for the main idea of our work.



Before filtering out border instances **After eliminating border instances**

Figure 9. The line or curve separates instances from different classes.

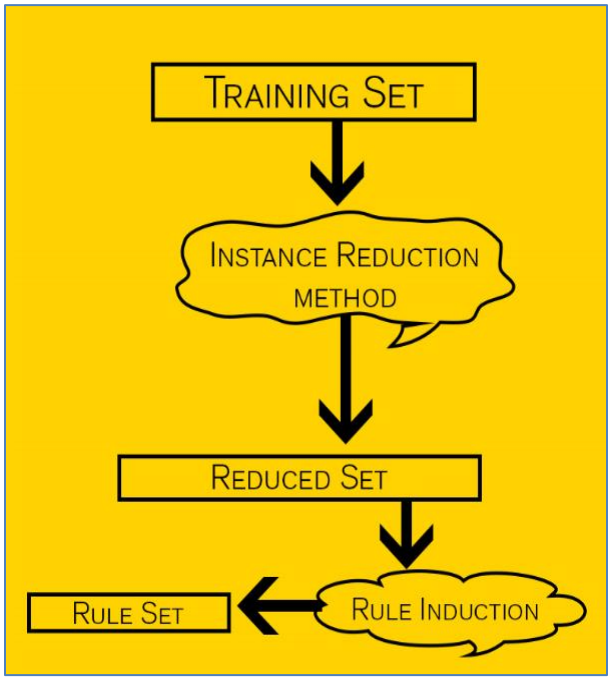


Figure 10. Framework for instance-reduction method preceding rule induction.

We will also apply the DROP5 method in instances flagged by AllKnn to be removed, and will call this the AllKnnDROP5 method. Figure 11 shows the suggested method.

```
Flag instances to be removed by using AllKnn method ( without actually removing)
        oldk = k
        For each instance(i)
                For k = 1 till oldk
                        if ( the class of instance(i) <> the majority class k nearest neighbours)
                                Flag the instance(i) for pruning

Execute DROP5 to validate the flagged instnces for pruning
```

Figure 11. AllKnnDROP5 algorithm.

Instance selection is classified as an NP-hard problem (Babu & Murty, 2001), which means that there is no polynomial algorithm able to find an optimal solution. Moreover, in Chapter 6 we will investigate a new instance-selection method based on ACO principles, and will specify how to set up different elements of ACO (i.e., problem representation, local heuristic, probabilistic transition rule, fitness function and pheromone update rule).

4.2 Benchmark Datasets

Results on a single dataset are typically not very meaningful. Therefore, machine learning techniques are often evaluated on a large set of benchmark datasets. We conduct experiments on a collection of machine learning datasets available from the repository at UCI (Murphy & Aha, 1994). We have chosen datasets with diverse characteristics: some of them have binary classes and others are multi-class; some of them have a lesser number of attributes while others have a relatively higher number; and some have a lesser number of examples while others have more. A summary of the properties of these datasets is given in Table 2.

ID#	Data Sets	No. of examples	No. of classes	Con. Attributes	Disc. attributes
1	Iris	150	3	4	-
2	Voting	435	2	-	16
3	Vowels	528	11	10	-
4	Heart Cleveland	303	2	7	6
5	Glass	214	7	9	–
6	Liver disorders	345	2	6	–
7	Wine	178	3	13	–
8	Pima Indians diabetes	768	2	8	-
9	Promoters	106	2	-	57
10	Hepatitis	155	2	6	13
11	Vehicle	848	4	18	-
12	Pole-and-cart	3481	2	4	-
13	Blood transfusion service	748	2	5	-
14	E-coli	336	8	7	–
15	Soybean	307	9	-	35
16	ZOO	101	7	1	15
17	Yeast	1484	10	8	-
18	Led creator	1000	10	-	7
19	Vertebral column	310	2	6	-
20	Ionosphere	352	2	34	-
21	Wave	5000	3	21	-

Table 2. Description of datasets used in empirical study *Notes: Columns show, in order: serial number, name of dataset, no. of examples in dataset, no. of classes in dataset, no. of continuous attributes, no. of discrete attributes.*

4.3 Rule-Induction Characteristics

We will consider rule-induction methods that learn a set of propositional rules where the target concept is represented as a set of "if... then..." rules. . We focus on rule-induction methods that produce an unordered set of rules, because we are interested in rule sets where each rule can be understood independently.

In this section, we categorize some rule-induction methods according to the following criteria:

1. Type of pruning: This criterion specifies whether the rule induction applies pruning when generating a rule set. The pruning can be:

a. Post-pruning: The pruning procedure is applied after the rule set has been induced.
b. Pre-pruning: A reduction or filtering method is applied before starting to generate the rule set.
c. During rule set generation: The rule is simplified as it is generated and before generating the next rule.

2. Direction of the search: There are three kind of search strategies for rule-induction methods (Pappa & Freitas, 2008):
    a. Specific-to-general (bottom-up strategy): Starts the search with a very specific rule, and iteratively generalizes it.
    b. General-to-specific (top-down strategy): Starts the search with the most general rule and iteratively specializes it.
    c. Hybrid (bi-directional strategy): A bi-directional search is allowed to generalize or specialize the candidate rules
3. Types of search include the following:
    a. Greedy search: Creates an initial rule, specializes or generalizes it, evaluates the extended rules created by the specialization or generalization operation, and keeps only the best extended rule.
    b. Beam search: Tries to eliminate the drawbacks of greedy search by selecting, instead of one, the b best extended rules at each iteration (where b is the width of the beam).

Table 3 uses these criteria to compare the rule-induction methods described in Chapter 2. This will guide us in selecting the algorithm that will be used in our experiments with the pre-pruning process. We think that pre-pruning can achieve good results with rule-induction algorithms that do not use pre-pruning, such as CN2 (modified), RISE, PRISM, the AQ family, the RULEs family, and IREP. In addition, we can choose methods that have different search types and directions. Accordingly, we choose to investigate pruning on CN2 (modified), PRISM, and RISE, as they have different search types and directions.

Rule-induction method	Type of pruning	Direction of search	Type of search
AQ family	Post-pruning	Hybrid	Beam search
CN2 (modified)	During rule generation	General to specific	Beam search
RIPPER	Pre- and post-pruning integration.	General to specific	Greedy search
IREP	During rule generation	General to specific	Greedy search
RULEs family	Post-pruning	General to specific	Beam search
RISE	No	Specific to general	Greedy search
PRISM	No	General to specific	Greedy search

Table 3. Comparison of rule-induction methods.

4.4 Estimating the Predictive Accuracy of Rules

The predictions that really matter to researchers are those for "future" data, whose classes are unknown at the time the classification algorithm is applied. We use predictive accuracy in the test set as an estimate of the predictive accuracy in future data. In this thesis, the classification quality of the rule set is measured by the predictive accuracy, which is defined as the percentage of the total number of correctly classified examples in all classes relative to the total number of tested examples. It has been by far one of the most commonly used metric for assessing performance of classifiers (Witton & Frank, 2005).

$$\textbf{\textit{Accuracy}} = \frac{\boldsymbol{N_{corr}}}{\boldsymbol{N}} \text{ X } 100\% \quad (17)$$

Where

N is the total number of test instances.

N_{corr} is the number of test instances correctly classified by the generated classifier.

4.5 Comparison Evaluation

This thesis reports on experiments that have been conducted to compare the application of different instance-reduction methods prior to rule induction. The comparison is conducted in terms of the predictive accuracy and comprehensibility. For predictive accuracy, the results are compared using statistical paired t-test with confidence at 0.05. A statistically significant improvement in predictive accuracy is referred to as a win, and a statistically significant reduction as a loss. For each pre-pruning method, we count the number of datasets that resulted in a win and the number of datasets that resulted in a loss.

On other hand, when dealing with learning algorithms it is important to be bear in mind that the most desirable property is comprehensibility. Furthermore, in some cases comprehensibility tends to be more important than predictive accuracy because:

1. The discovered knowledge (rule set) will be used for supporting a decision to be made by a human.
2. If the discovered knowledge (rule set) is not comprehensible, nobody will be able to validate it, and a human may not trust it.

In general, the shorter (the fewer number of conditions in) a rule, the more comprehensible it is. The same principle applies to rule sets. In general, the lower the number of generated rules in a rule set, the more comprehensible it is (Shirbhate & Gupta, 2015; Blanco-Vega et al., 2004).

4.6 Experimental Setup

This section explains the different experimental setups used in this thesis. For each evaluation, we conduct testing in all datasets mentioned in Table 2.

4.6.1 Cross-validation

Cross-validation (Kohavi, 1995) is a common method for estimating different learning algorithms. The accuracy of the resulting classifier is estimated by dividing the data into n parts. In each experiment, n -1 parts are combined into a training set and the remaining part is used for testing. A model is then learned on the training set and evaluated on the test set. This is repeated until each part (and thus each training example) has been used once for testing. The final accuracy is then estimated as an average of the accuracy estimates computed in each such experiment. The cross-validation algorithm is shown in Figure 12. This algorithm can be used to estimate any learning algorithm. It is thus shown with generic functions for learning (LearnAlgorithm) and evaluating (Evaluate).

In this thesis, the predictive accuracy is estimated using 10-fold cross-validation. Each of the folds is used once for testing, and the remaining nine are used as a training set.

```
Randomly partition  data set (T) into n disjoint (Ti) of approximately same size
For each j =1 to n do
        Test = Tj
        Training = T excluding Test
        LearnAlgorithm(Training)
        vj = Evaluate( Test )
        v = v + vj
End For
Performance = v / n
```

Figure 12. Estimating the predictive quality of learning algorithms using cross-validation.

4.6.2 Choosing K for K-NN algorithm

The k-NN algorithm is amongst the simplest of all machine learning algorithms. An instance is classified by a majority vote of its neighbours, with the instance being assigned to the class that is most common amongst its k nearest neighbours (k is a positive integer that is typically small). If k = 1, then the instance is simply assigned to the class of its NN.

How should one go about choosing the value of k? In fact, there may not be an obvious best solution. Consider choosing a small value for k. In such a case, it is possible that the classification may be unduly affected by outliers or noise. On the other hand, choosing a value of k that is not too small will tend to smooth out any idiosyncratic behaviour learned from the training set. However, if we take this too far and choose a value of k that is too large, locally interesting behaviour will be overlooked (Larose, 2005). Furthermore, the value of k must set to an odd number to avoid ties.

In this thesis, we avoid using k = 1 in experiments for evaluating the behaviour of the k-NN algorithm, based on the earlier discussion. We set k to 3, the next smallest odd number.

Furthermore, the additional complexity required to use a larger number of neighbours than three is not warranted due to the small decrease in the error rate when more than three are used (Wilson, 1972).

4.6.3 Number of Ants in Ant Colony Optimization

ACO is a promising new approach to solving various problems. Many factors affect the ability of ACO to achieve good solutions to these problems. One of these factors is the number of ants. Finding the exact number of ants required to solve a problem remains an empirical problem based on fine tuning.

In our experiments, we test the effect of changing the number of ants on the predictive accuracy and the number of generated rule sets. We evaluate the ACO with 250, 500, 750, 1,000, and 1,250 ants.

4.6.4 Experiment Implementation

In our experiments, we used the code for the CN2 algorithm implemented by Robin Boswell in 1990, from which Francisco Reinaldo (Univ. Porto, Portugal) and Marcus Siqueira (UnilesteMG, Minas Gerais, and Brazil) created the executable file for Windows XP. We used the version of CN2 that produces an unordered list of rules. We implemented the RISE algorithm using the C programming language. Furthermore, for the PRISM algorithm we used the Inducer rule-induction workbench (Bramer, 2000); this is one of a suite of packages developed to facilitate experiments with different techniques for generating classification rules. Inducer is implemented in Java (version 1.1) in the interests of portability and is available both as a standalone application and as an applet.

We also implemented the proposed ACO using Microsoft visual studio, again using the C programming language.

4.6.5 Summary

This chapter introduced all the items needed to run our experiments, including our test strategy. It also outlined how to compare and evaluate the achieved results in terms of predictive accuracy and comprehensibility (i.e., number of generated rules). We compared and characterized different rule-induction methods, then clarified our chosen methods to be used in our experiments. Moreover, we introduced the parameters for implementing the instance-reduction method based on the ACO concept.

The next chapter describes the details of the experiments conducted, and the results that were obtained.

Chapter 5: Preceding Rule Induction with Instance-Reduction Methods

This chapter presents the empirical results for investigating preceding three different types of rule induction with instance-reduction methods (CN2, PRISM, and RISE). Section 5.1 explains the basic ideas behind the experiments and the setup used to complete them. In Section 5.2, we present our analysis of the results obtained in terms of predictive accuracy and number of generated rules on the 22 datasets described in Section 4.2. Section 5.3 presents our conclusions.

5.1 Experimentation

We focus on instance-reduction methods that have been proven capable of reducing the size of training sets while maintaining as much predictive accuracy as possible (Wilson & Martinez, 1997, 2000). More specifically, we apply algorithms that aim to reduce the border instances before applying the induction method. This can achieve good results as removing border instances should remove instances that are noisy, which may improve the predictive accuracy for the induction method. Furthermore, we investigate the effect of preceding instance-reduction methods on the complexity of rule set (roughly represented here by the number of generated rules). El Hindi and Alakhras (2009) showed that filtering out border instances before training an artificial neural network will improve the predictive accuracy in some cases and speed up the training process by reducing training epochs.

Our experiments concern three reduction algorithms that performed well in reducing the number of instances (Wilson & Martinez, 1997). We applied the three methods for instance

reduction (AllKnn, ENN, and DROP5) that are intended to remove the border and noisy instances before using CN2, PRISM, and RISE. We also applied the DROP5 (Wilson, & Martinez, 2000) method for instances flagged by AllKnn to be removed; we call this method AllKnnDROP5.

.

The CN2 (Clark & Niblett, 1989) algorithm induces an ordered list of classification rules from examples, using entropy as its heuristic. Clark and Boswell improved CN2 by using a Laplacian error estimate as an alternative evaluation function, and producing unordered classification rules (Clark & Boswell, 1991). One of our objectives was to apply some instance-reduction methods before applying the modified CN2 algorithm and compare the results with and without applying the reduction.

5.2 Analysis of Results

Table 4 presents the average number of generated rules by preceding the CN2 algorithm with different instance-reduction methods. Moreover, we compare the amount of reduction with respect to the average number of rules generated by applying CN2 (R_{CN2}) without pre-pruning. From Table 4, it is clear that all of the instance-reduction techniques reduced the number of rules generated by CN2. We can see that DROP5 achieved the largest reduction, as the ratio of the average number of rules between preceding CN2 with DROP5 and applying CN2 without pre-pruning (R_{DROP5}/ R_{CN2}) is 0.34, which means that the reduction was 64% on average. On the other hand, applying ENN, AllKnnDrop5, and AllKnn reduced the generated rules by 51%, 50%, and 55% on average, respectively.

Table 5 reveals the results of the average number of generated rules by applying the instance-reduction techniques prior to the RISE algorithm. We computed the ratio of average number of rules between preceding RISE with different instance-reduction methods and applying RISE without pre-pruning, so we were able to investigate the amount of reduction in the average number of generated rules. It is clear that applying DROP5 still achieved the highest reduction in the number of generated rules followed by applying AllKnn, which achieved 55% on average. Furthermore, AllKnnDrop5 and ENN reduced the generated rules by 51% and 47% on average, respectively.

Finally, Table 6 shows the average number of generated rules by preceding the PRISM algorithm with different instance-reduction techniques. We can see that DROP5 achieved the largest reduction in the number of generated rules, as the ratio of the average number of rules between preceding PRISM with DROP5 and applying PRISM without pre-pruning is 0.28, which means that the reduction was 72% on average. Moreover, AllKnnDrop5, AllKnn, and ENN reduced the generated rules by 46%, 54%, and 47% on average, respectively.

Figure 13 shows that for all rule-induction methods, the number of generated rules reduced after applying different instance-reduction methods. It is clear that applying DROP5 achieved the largest reduction in the number of generated rules of the four rule-induction methods. AllKnn achieved the next best reduction in the number of generated rules, followed by AllKnnDrop5 and ENN.

		ENN		***AllKnn***		***DROP5***		***AllKnnDROP5***	
Datasets	R_{CN2}	R_{ENN}	R_{ENN}/R_{CN2}	R_{AllKnn}	R_{AllKnn}/R_{CN2}	R_{DROP5}	R_{DROP5}/R_{CN2}	$R_{AllKnnDROP5}$	$R_{AllKnnDROP5}/R_{CN2}$
Iris	6.30	3.9	0.62	3.6	0.57	3	0.48	3.6	0.57
Voting	17.3	6.2	0.36	5.7	0.33	3	0.17	6.1	0.35
Vowels	46.2	42.2	0.91	41.5	0.9	31.7	0.69	44.3	0.96
Heart Cleveland	21.3	11.2	0.53	9.4	0.44	7	0.33	10.6	0.5
Glass	22.0	12.8	0.58	12.1	0.55	9.2	0.42	10.3	0.47
Liver disorders	31.3	17.6	0.56	15.2	0.49	12.6	0.4	18.1	0.58
Wine	8.60	7.4	0.86	6.9	0.8	3	0.35	6.9	0.8
Pima Indians diabetes	44.4	20.8	0.47	18.1	0.41	15.6	0.35	21.3	0.48
Promoters	12.4	10.4	0.84	9.6	0.77	2.7	0.22	9.7	0.78
Hepatitis	17.8	1.80	0.1	4.2	0.24	1.7	0.1	4.7	0.26
Vehicle	48.4	29.3	0.61	25.9	0.54	27.2	0.56	29.3	0.61
Pole-and-cart	109.8	56.9	0.52	46.7	0.43	51.7	0.47	50.8	0.46
Blood transfusion service	61.2	13.0	0.21	11.9	0.19	13.2	0.22	16.5	0.27
E-coli	24.7	12.7	0.51	10.5	0.43	7.7	0.31	12.3	0.5
Soybean	32.7	15.9	0.49	24.8	0.76	21.3	0.65	27.2	0.83
ZOO	8.70	6.1	0.7	6.3	0.72	6.2	0.71	6.3	0.72
Yeast	121.2	40.7	0.34	37.0	0.31	40.5	0.33	47.3	0.39
Led creator	79.9	21.8	0.27	19.9	0.25	23.4	0.29	24.3	0.3
Vertebral column	16.7	10.4	0.62	9.1	0.54	6.9	0.41	10.1	0.6
Ionosphere	17.6	6.5	0.37	7.2	0.41	4.9	0.28	9.7	0.55
Wave	204.8	118.0	0.58	102.3	0.5	60.3	0.29	111.6	0.54
Balance scale	150.1	75.4	0.5	63.0	0.42	21.6	0.14	65.2	0.43
Average	50.15	24.59	0.49	22.31	0.45	17.02	0.34	24.83	0.50

Table 4. Empirical results comparing the average number of generated rules for preceding CN2 with ENN (R_{ENN}), AllKnn (R_{AllKnn}), DROP5 (R_{DROP5}), and AllKnnDrop5 ($R_{AllKnnDROP5}$), and comparing the amount of reduction with respect to the average number of rules generated by applying CN2 (R_{CN2}) without pre-processing.

		ENN		AllKnn		DROP5		AllKnnDROP5	
Datasets	R_{RISE}	R_{ENN}	R_{ENN}/R_{RISE}	R_{AllKnn}	R_{AllKnn}/R_{RISE}	R_{DROP5}	R_{DROP5}/R_{RISE}	$R_{AllKnnDROP5}$	$R_{AllKnnDROP5}/R_{RISE}$
Iris	22.50	10.90	0.48	4.70	0.21	4.80	0.21	4.60	0.2
Voting	88.10	56.30	0.64	46.90	0.53	7.60	0.09	48.00	0.41
Vowels	72.10	51.30	0.71	49.40	0.69	77.10	1.07	50.10	0.69
Heart Cleveland	97.30	55.30	0.57	44.40	0.46	20.30	0.21	44.30	0.46
Glass	67.30	42.20	0.63	34.50	0.51	19.60	0.29	35.10	0.52
Liver disorders	183.7	101.6	0.55	74.10	0.4	48.60	0.26	91.40	0.5
Wine	20.50	17.60	0.86	15.40	0.75	5.80	0.28	18.80	0.92
Pima Indians diabetes	379.6	181.3	0.48	146.0	0.38	65.20	0.17	172.90	0.46
Promoters	60.80	55.10	0.91	58.40	0.96	6.00	0.1	59.30	0.98
Hepatitis	71.60	9.00	0.13	8.80	0.12	1.50	0.02	36.30	0.51
Vehicle	267.5	166.8	0.62	127.2	0.48	97.60	0.36	164.60	0.62
Pole-and-cart	3133	368	0.11	370.5	0.12	329.5	0.11	435.10	0.14
Blood transfusion service	212.2	64.00	0.3	43.50	0.2	33.60	0.16	57.80	0.27
E-coli	128.1	63.40	0.49	36.40	0.28	18.10	0.14	43.50	0.34
Soybean	68.00	50.30	0.74	38.60	0.57	45.60	0.67	45.80	0.67
ZOO	8.90	6.70	0.75	6.70	0.75	9.10	1.02	6.90	0.78
Yeast	774.5	366.5	0.47	250.3	0.32	175.5	0.23	330.90	0.43
Led creator	271.7	30.10	0.11	26.00	0.1	42.90	0.16	35.20	0.13
Vertebral column	129.6	87.80	0.68	75.10	0.58	22.30	0.17	83.70	0.65
Ionosphere	147.7	36.10	0.24	33.70	0.23	14.70	0.1	69.20	0.47
Wave	4500	3685	0.82	3213	0.71	515.9	0.11	3348.90	0.74
Balance scale	350.6	303.2	0.85	252.8	0.72	44.10	0.13	257.50	0.73
Average	502.5	264.8	0.53	225.3	0.45	73	0.15	247.3	0.49

Table 5. Empirical results comparing the average number of generated rules for preceding RISE with ENN (R_{ENN}), AllKnn (R_{AllKnn}), DROP5 (R_{DROP5}), and AllKnnDrop5 ($R_{AllKnnDROP5}$), and comparing the amount of reduction with respect to the average number of rules generated by applying RISE (R_{RISE}) without pre-processing.

		ENN		AllKnn		DROP5		AllKnnDROP5	
Datasets	R_{RISE}	R_{ENN}	R_{ENN}/R_{RISE}	R_{AllKnn}	R_{AllKnn}/R_{RISE}	R_{DROP5}	R_{DROP5}/R_{RISE}	$R_{AllKnnDROP5}$	$R_{AllKnnDROP5}/R_{RISE}$
Iris	16.30	7.40	0.45	7.50	0.46	4.00	0.25	7.50	0.46
Voting	31.20	10.50	0.34	7.90	0.25	4.80	0.15	8.50	0.27
Vowels	198.6	186.9	0.94	188.2	0.95	105.9	0.53	189.20	0.95
Heart Cleveland	80.30	35.70	0.44	28.20	0.35	15.10	0.19	33.70	0.42
Glass	84.20	41.50	0.49	39.60	0.47	26.40	0.31	48.40	0.57
Liver disorders	122.9	57.90	0.47	47.10	0.38	39.50	0.32	58.30	0.47
Wine	18.40	18.00	0.98	18.20	0.99	3.40	0.18	18.30	0.99
Pima Indians diabetes	221.2	86.00	0.39	62.60	0.28	49.50	0.22	84.30	0.38
Promoters	15.90	14.30	0.9	14.00	0.88	3.50	0.22	14.40	0.91
Hepatitis	33.70	1.90	0.06	69.30	2.06	1.00	0.03	6.30	0.19
Vehicle	259.6	146.7	0.57	103.0	0.4	91.50	0.35	142.80	0.55
Pole-and-cart	829.9	475.3	0.57	395.2	0.48	408.0	0.49	562.50	0.68
Blood transfusion service	187.9	30.4	0.16	23.40	0.12	27.40	0.15	36.10	0.19
E-coli	80.50	41.80	0.52	37.00	0.46	17.70	0.22	40.50	0.5
Soybean	71.50	51.00	0.71	48.70	0.68	34.60	0.48	52.40	0.73
ZOO	14.10	10.20	0.72	10.30	0.73	6.70	0.48	10.20	0.72
Yeast	698.8	240.2	0.34	197.6	0.28	171.7	0.25	264.30	0.38
Led creator	75.40	27.50	0.36	24.20	0.32	30.90	0.41	30.80	0.41
Vertebral column	67.50	33.10	0.49	27.20	0.4	14.90	0.22	32.20	0.48
Ionosphere	42.00	12.90	0.31	14.50	0.35	10.10	0.24	23.50	0.56
Wave	1416	915.8	0.65	762.8	0.54	280.7	0.2	846.50	0.6
Balance scale	270.1	115.1	0.43	86.20	0.32	29.30	0.11	90.80	0.34
Average	219.8	116.4	0.53	100.6	0.46	62.6	0.28	118.3	0.54

Table 6. Empirical results comparing the average number of generated rules for preceding PRISM with ENN (R_{ENN}), AllKnn (R_{AllKnn}), DROP5 (R_{DROP5}), and AllKnnDrop5 ($R_{AllKnnDROP5}$), and comparing the amount of reduction with respect to the average number of rules generated by applying PRISM (R_{PRISM}) without pre-processing.

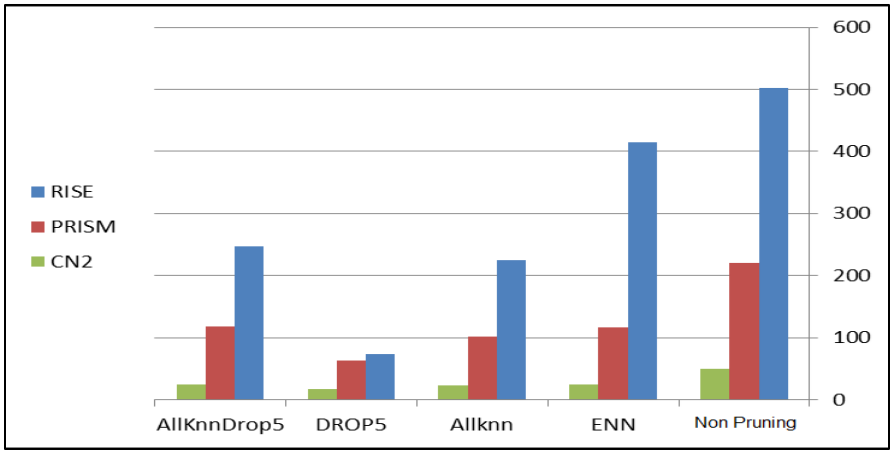


Figure 13. Comparison of the average number of generated rules before and after applying instance-reduction methods for different rule-induction methods.

We are comparing the results using paired t-test with confidence 0.5 to have better interpretation. Table 7 shows the results obtained for CN2 and applying the four pre-pruning methods with respect to the predictive accuracy. The bold results with a superscript of + means that applying pre-pruning resulted in a statistically significant increase in predictive accuracy, while those bold with - showed a statistically significant decrease in predictive accuracy. Our experiments show that there was no statistically significant effect on predictive accuracy after applying ENN, AllKnn, and AllKnnDrop5 on 19, 19, and 20 datasets, respectively. On other hand, there was a statistically significant increase in predictive accuracy for two datasets. We can conclude that preceding CN2 with these instance-reduction methods did not adversely affect the predictive accuracy on most datasets and, for two datasets, it enhanced the predictive accuracy. However, when using DROP5, there was no statistically significant increase in predictive accuracy for any of the datasets. Furthermore, for 15 of the 22 datasets, using DROP5 led to a statistically significant decrease.

Table 8 summarizes the effect of instance selection (pruning training data) on generalization of the RISE algorithm. Our experiments show that the predictive accuracy is not statistically affected after applying ENN, AllKnn, DROP5, and AllKnnDrop5 on 17, 16, 8, and 17 datasets, respectively. Furthermore, applying ENN, AllKnn, and AllKnnDrop5 yielded statistically significant increases in predictive accuracy on 3, 4, and 3 datasets, respectively. Applying DROP5 produced the worst results, and is thus not recommended as a pre-pruning method for RISE rule induction.

Datasets	Without pruning	ENN	AllKnn	DROP5	AllKnnDrop5
Iris	89.98	92.00	92.67	80.67	93.34
Voting	95.34	95.10	95.33	**85.35** $^{-}$	95.57
Vowels	67.11	65.97	66.75	**65.07** $^{-}$	67.31
Heart Cleveland	80.66	76.66	77.33	**71.66** $^{-}$	79.34
Glass	64.76	58.05	61.98	**51.92** $^{-}$	66.22
Liver disorders	66.77	64.11	65.64	**60.30** $^{-}$	66.52
Wine	91.77	94.11	93.52	**70.00** $^{-}$	95.28
Pima Indians diabetes	70.30	73.16	74.70	73.40	72.10
Promoters	85.00	81.00	80.00	**63.00** $^{-}$	80.00
Hepatitis	78.65	80.00	80.00	**52.67** $^{-}$	79.34
Vehicle	57.85	60.10	60.71	54.99	60.10
Pole-and-cart	61.68	63.88	**66.24** $^{+}$	62.56	63.51
Blood transfusion service	75.68	76.61	76.35	73.11	75.96
E-coli	79.10	**83.31** $^{+}$	80.91	**73.34** $^{-}$	80.90
Soybean	86.32	82.67	83.01	**63.00** $^{-}$	83.32
ZOO	92.00	87.00	90.00	**81.00** $^{-}$	89.00
Yeast	48.98	**55.47** $^{+}$	**56.43** $^{+}$	51.82	**56.56** $^{+}$
Led creator	72.30	72.30	71.30	**68.90** $^{-}$	71.90
Vertebral column	80.96	83.21	81.28	81.28	82.24
Ionosphere	89.43	**85.71** $^{-}$	**86.56** $^{-}$	**53.71** $^{-}$	85.71
Wave	69.70	70.38	70.74	**67.96** $^{-}$	**71.38** $^{+}$
Balance scale	75.30	74.70	74.34	**67.10** $^{-}$	74.34
Average	76.35	76.16	76.63	66.95	76.82
Win/tie/loss		2/19/1	2/19/1	0/7/15	2/20/0

Table 7. Empirical results comparing predictive accuracy using ENN, AllKnn, DROP5, and AllKnnDrop5 pre-pruning with CN2.

Datasets	Without pruning	ENN	AllKnn	DROP5	AllKnnDrop5
Iris	95.33	94.00	94.67	94.01	94.67
Voting	95.10	95.32	95.79	93.25	95.32
Vowels	92.68	**88.87** -	**89.25** -	**85.97** -	**89.63** -
Heart Cleveland	77.00	77.01	75.32	**71.01** -	75.01
Glass	67.14	62.85	64.77	**52.37** -	65.70
Liver disorders	65.29	61.18	62.00	**57.05** -	65.23
Wine	97.64	95.28	96.46	**88.83** -	97.64
Pima Indians diabetes	67.63	68.29	68.37	68.56	67.70
Promoters	86.00	92.00	88.00	**67.00** -	87.00
Hepatitis	80.67	80.67	80.66	**52.00** -	80.67
Vehicle	70.35	68.47	**66.55** -	**65.36** -	**67.62** -
Pole-and-cart	61.87	62.18	**65.49** +	58.81	64.24
Blood transfusion service	73.92	**79.19** +	**77.84** +	74.87	**77.34** +
E-coli	84.76	85.75	85.46	83.02	86.35
Soybean	91.00	87.67	87.66	**82.67** -	88.33
ZOO	96.00	**89.00** -	93.00	**89.00** -	93.00
Yeast	52.97	**57.56** +	**58.25** +	**53.99** -	**56.83** +
Led creator	72.60	72.40	72.60	**69.40** -	72.80
Vertebral column	82.91	81.60	81.93	81.30	82.90
Ionosphere	92.56	91.42	91.71	**77.42** -	90.56
Wave	81.84	82.18	**83.26** +	**79.06** -	**82.82** +
Balance scale	78.06	**81.13** +	80.97	77.75	81.62
Average	80.15	79.73	80	73.76	80.14
Win/tie/loss		3/17/2	4/16/2	0/8/14	3/17/2

Table 8. Empirical results comparing predictive accuracy using ENN, AllKnn, DROP5, and AllKnnDrop5 pre-pruning with RISE.

Table 9 clearly shows that applying ENN, AllKnn, DROP5, and AllKnnDrop5 prior to PRISM did not statistically affect the predictive accuracy on 11, 14, 9, and 15 datasets, respectively. On other hand, the results reveal that applying ENN, AllKnn, and AllKnnDrop5 yielded statistically significant increases on 9, 7, and 6 datasets, respectively. Applying DROP5 still produced the worst results, and is not recommended as a pre-pruning method for PRISM rule induction.

Based on the previous results, we observed that applying DROP5 yielded poor results for all investigated rule-induction methods in terms of predictive accuracy. Thus, we focused more on the results achieved by the other instance-reduction methods. Table 10 summarizes the

Datasets	Without pruning	ENN	AllKnn	DROP5	AllKnnDrop5
Iris	91.40	88.20	88.80	**79.20** $^{-}$	88.80
Voting	92.50	**95.50** $^{+}$	**95.70** $^{+}$	93.10	**96.20** $^{+}$
Vowels	52.40	50.70	51.10	**42.40** $^{-}$	51.10
Heart Cleveland	68.00	**74.00** $^{+}$	**73.90** $^{+}$	**62.70** $^{-}$	**72.40** $^{+}$
Glass	43.90	47.20	48.70	**32.90** $^{-}$	48.30
Liver disorders	47.90	**56.90** $^{+}$	53.60	51.20	52.40
Wine	86.30	83.90	83.90	**69.80** $^{-}$	86.30
Pima Indians diabetes	62.80	63.20	64.00	60.40	63.40
Promoters	73.00	77.00	74.00	**52.00** $^{-}$	72.00
Hepatitis	69.30	78.70	77.30	**79.30** $^{+}$	74.60
Vehicle	58.70	57.60	59.30	**50.00** $^{-}$	59.30
Pole-and-cart	52. 50	**56.20** $^{+}$	**56.60** $^{+}$	48.70	55.00
Blood transfusion service	71.70	76.4	72.70	69.20	**73.20** $^{+}$
E-coli	73.30	**79.00** $^{+}$	**78.40** $^{+}$	69.60	**78.40** $^{+}$
Soybean	79.50	**73.90** $^{-}$	**73.40** $^{-}$	**56.30** $^{-}$	**74.20** $^{-}$
ZOO	92.00	**84.00** $^{-}$	88.00	**85.00** $^{-}$	87.00
Yeast	43.80	**49.30** $^{+}$	**46.40** $^{+}$	41.70	46.70
Led creator	71.70	72.40	71.60	67.40	72.10
Vertebral column	73.40	**78.00** $^{+}$	74.20	75.40	75.50
Ionosphere	86.90	87.50	89.30	**53.30** $^{-}$	88.80
Wave	59.30	**63.10** $^{+}$	**63.10** $^{+}$	**54.30** $^{-}$	**63.50** $^{+}$
Balance scale	62.70	**72.10** $^{+}$	**73.00** $^{+}$	**52.30** $^{-}$	**73.00** $^{+}$
Average	69.55	71.13	70.77	61.19	70.55
Win/tie/loss		9/11/2	7/14/1	1/9/12	6/15/1

Table 9: Empirical results comparing predictive accuracy using ENN, AllKnn, DROP5, and AllKnnDrop5 pre-pruning with PRISM.

characteristics of the different datasets used in our experiments. The "total attributes" column specifies the summation of discrete and numerical attributes for a certain dataset, while the "missing attributes?" column specifies whether the dataset had attributes with missing values. We study the application of ENN, AllKnn, and AllKnnDrop5 to different rule-induction methods by summarizing the statistically significant increase or decrease in predictive accuracy for each dataset in the "No. wins/losses" column, which subtracts the number of datasets that had a statistically significant decrease in predictive accuracy from the number of datasets with a statistically significant increase. We then sorted the datasets accordingly. We observed that, for

each dataset, if one or more of the combinations of an instance-reduction method and rule-induction method resulted in a statistically significant increase in predictive accuracy then none of the combinations resulted in a statistically significant decrease, and vice versa. In addition, we noticed that the best results were achieved with datasets with a low number of total attributes with respect to the number of instances. On other hand, we observed that the "Heart Cleveland" and "Voting" datasets had statistically significant increases even though they had a high number of attributes with respect to number of instances. The cause for this may have been the presence of missing values for certain attributes in these datasets.

5.3 Conclusion

In our experiments, we investigated preceding three different types of rule induction with instance-reduction methods. The search strategies used by the three algorithms varied in terms of both type (greedy or beam search) and direction (general-to-specific or specific-to-general). We highlighted several instance-reduction techniques, and applied them as pre-processing. Our experiments show that for most datasets, pruning the training set using AllKnn, ENN, or AllKnnDrop5 significantly reduced the number of rules generated by CN2, RISE, and PRISM, without adversely affecting the predictive performance.

ID#	Datasets	No. of examples	No. of classes	Con. Attributes	Disc. attributes	Total attributes	Missing attributes?	No. wins/ losses	No. of attributes/ No. of examples
3	Vowels	528	11	10	0	10	n	-3	0.0189
15	Soybean	307	9	0	35	35	y	-3	0.114
11	Vehicle	848	4	18	0	18	n	-2	0.0212
16	Zoo	101	7	1	15	16	n	-2	0.1584
20	Ionosphere	352	2	34	0	34	n	-2	0.0966
1	Iris	150	3	4	0	4	n	0	0.0267
5	Glass	214	7	9	0	9	n	0	0.0421
7	Wine	178	3	13	0	13	n	0	0.073
8	Pima Indians diabetes	768	2	8	0	8	y	0	0.0104
9	Promoters	106	2	0	57	57	n	0	0.5377
10	Hepatitis	155	2	6	13	19	y	0	0.1226
18	Led creator	1000	10	0	7	7	n	0	0.007
6	Liver disorders	345	2	6	0	6	n	1	0.0174
19	Vertebral column	310	2	6	0	6	n	1	0.0194
2	Voting	435	2	0	16	16	y	3	0.0368
4	Heart Cleveland	303	4	7	6	13	y	3	0.0429
12	Pole-and-cart	3481	2	4	0	4	n	4	0.0011
13	Blood transfusion service	748	2	5	0	5	n	4	0.0067
14	E-coli	336	8	7	0	7	n	4	0.0208
22	Balance scale	626	3	0	4	4	n	4	0.0064
21	Wave	5000	3	21	0	21	n	6	0.0042
17	Yeast	1484	10	8	0	8	n	8	0.0054

Table 10: Results from application of ENN, AllKnn, and AllKnnDrop5 as pre-pruning techniques with CN2, RISE, and PRISM algorithms.

Chapter 6: Instance-Reduction Method based on Ant Colony Optimization

This chapter investigates a new instance-reduction method based on ACO. Section 6.1 describes the proposed method in detail. In Section 6.2 we investigate the performance of the new method by applying the k-NN classification method, and compare the results of the experiments conducted with those obtained using other instance-reduction methods. In Section 6.3, we present the analysis of the results achieved by preceding the three different types of rule induction with the new instance-reduction method based on ACO, in terms of predictive accuracy and number of generated rules. Section 6.4 presents our conclusions.

The k-NN classification enables classification of unknown instances by using a set of classified training instances. In order to build an efficient k-NN classifier, two principle objectives have to be reached:

1. Achieve high predictive accuracy, and
2. Reduce the set of instances.

Instance-reduction methods are used to find suitable representative instances from data, which can help in reducing the size of the retained instances. This problem is classified as an NP-hard problem (Babu & Murty, 2001), which means that there is currently no polynomial algorithm able to find an optimal solution. In Section 2.4.1, we mentioned different kinds of instance-reduction methods that provide an acceptable solution in reasonable time.

Recently, ACO has been successfully applied in solving different types of combinatorial optimization problems. ACO simulates the natural behaviour of ants, especially their mechanisms of adaptation and cooperation. The basic idea of our proposed algorithm is to retain the internal instances from each class to smooth the decision boundaries by filtering out near-border instances from the training set, as these instances are a major source for overfitting. Furthermore, we concentrate on the most important instances using the predictive accuracy for the original training set as a fitness function. In our proposed approach, we use the ACO principle in instance reduction. An ant will decide whether to select the instance as part of its subset. We consider the training set as a weighted graph with connected nodes where the set of nodes (N) represents the instances and the set of edges represents the distance between pairs of nodes. Moreover, each ant incrementally constructs a solution from an original training set. The selected instances will be training set for the k-NN classifier. Hereinafter, we call our proposed algorithm ACO-IR.

6.1 ACO-IR

This section describes our proposed ACO-IR method. The main idea in our proposed method is that each ant constructs a candidate reduced set from the original training set. After an ant has completed its tour, the fitness of the reduced set is calculated by classifying (using the k-NN algorithm) all instances in the original training set and checking the predictive accuracy. Figure 14 describes the framework for our proposed approach.

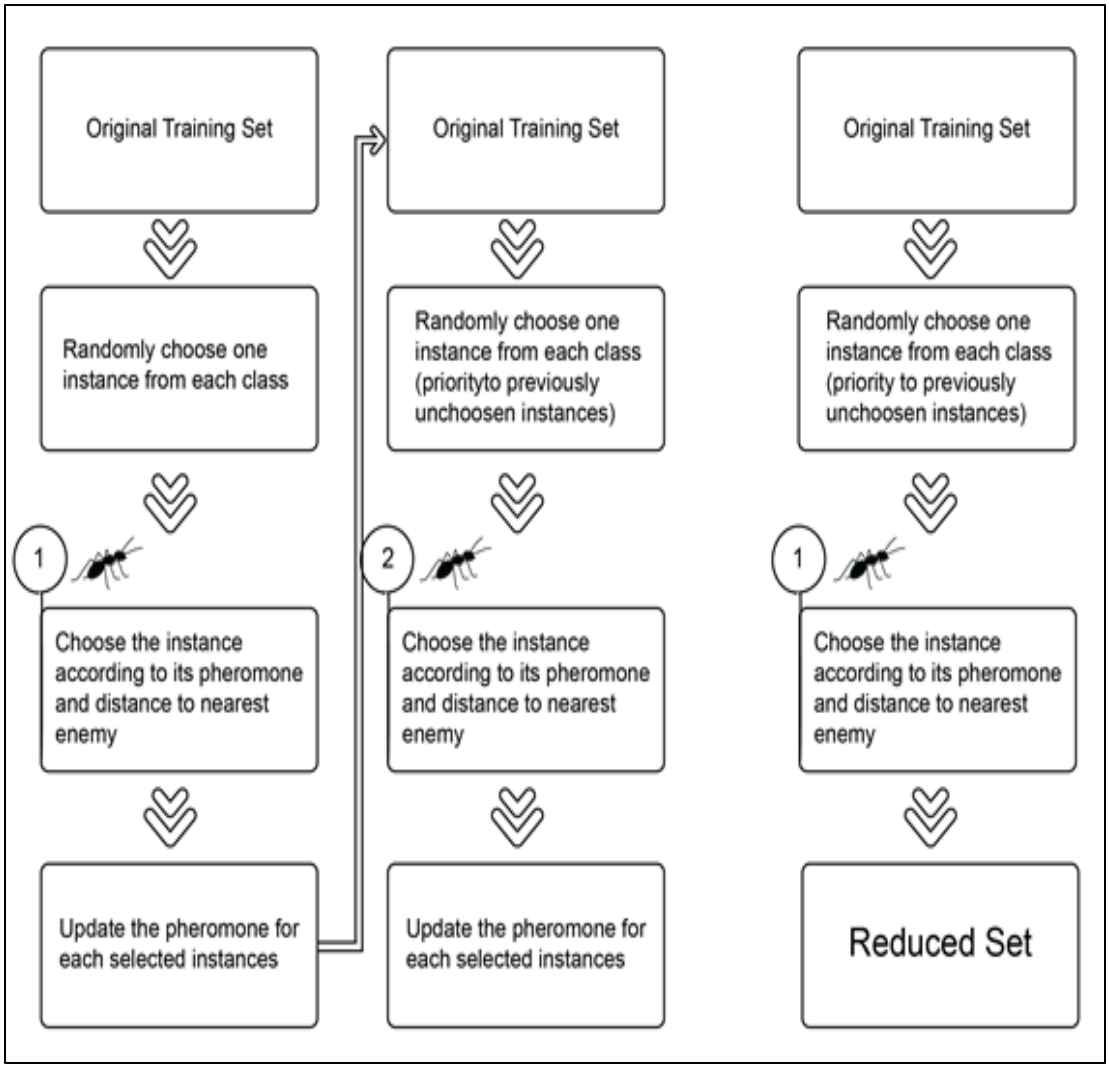


Figure 14. Framework for ACO-IR.

In our method, each ant starts by randomly choosing one instance from each class, and then searching for the instances to be selected. Selection of the instance is based on two parameters:

1. The local heuristic, which is the distance between the candidate instance at time t and the nearest chosen instance at that time with a different class, which represent the heuristic desirability of choosing instance j when we have selected certain instances.

2. The pheromone level associated with an instance.

In our approach, the basic ACO algorithm is used. Figure 15 describes the ACO-IR method.

```
TrainingSet = all training examples.
ReducedSet = [].
t= 0;
Repeat
        t= t + 1;
        For each c in classes
           ReducedSet = ReducedSet + randomly select instance with class =c.
        Ant (t) constructs a ReducedSet Rt
        Update the pheromone of instances;
Until (t>= NoAnt)
```

Figure 15. ACO-IR algorithm.

The main factors involved in our ACO-IR method are setting up initialization of pheromone values, selecting subset of instances (generation of solutions), heuristic function, fitness evaluation of the generated solutions, pheromone evaporation, pheromone update, and number of ants. All these steps for our proposed approach are discussed in the following subsections.

6.1.1 Initialization of Pheromone Values

The presence of pheromone values is the basic component of ACO. It is initialized with some small random values. In our approach, the pheromone is attached to each instance in the training set. The pheromone values on all instances are initialized with same amount of pheromone. In this way, no instance is preferred over other instances by the first ant. The initial pheromone is calculated using Equation 18:

$$\mu_i\,(1) = \frac{\mathbf{1}}{N} \tag{18}$$

Where

N is the total number of instances.

μ_i (1) is initial pheromone for the ith instance.

6.1.2 Selecting Subset of Instances (Generation of Solutions)

In our method, each ant starts by randomly choosing one instance from each class. It then chooses the instances according to their probability. We generate a "bootstrap dataset" by sampling instances from the original training set with a replacement of the same size as our original dataset. As a result, some instances may appear more than once in a given bootstrap dataset, and some not at all.

An ant uses two components to calculate the probability of choosing an instance from set of instances. The first component is the amount of pheromone present in the instance, and second is the heuristic describing the worth of the instance. The probability with which the ant chooses instance i as the next instance is defined by:

$$P_i = \left(H_i * \mu_i(t)\right)/\left(\sum_{j=1}^{a} H_j * \mu_j(t)\right) \tag{19}$$

Where:

P_i is the probability that instance i is selected.

$H_{i(or\ j)}$ is the heuristic value associated with instance i (or j).

$\mu_{i(or\ j)}(t)$ is the amount of pheromone associated with instance i (or j) at iteration t.

a is the total number of instances.

The process by which the ant selects instances is repeated for, at most, a predefined number of ants.

6.1.3 Heuristic Function

The heuristic function indicates the quality of an instance. Its value greatly influences the ant's decision to move and select the next instance to be retained in the reduced set. A good heuristic function is very helpful in solving problems using ACO. In our proposed algorithm, we choose to retain inner instances that are far from enemy instances (instances with a different class). We use the distance between instances and its nearest enemy in the reduced set. This heuristic can be calculated using Equation 20:

$$Ħ_i = \text{MinEnemy}(d_i(t)) \tag{20}$$

Where:

$Ħ_i$ is the heuristic value associated with instance i.
$d_i(t)$ is the distance between instance i and the instances in the reduced set at time t.

6.1.4 Fitness Function

The fitness function helps to identify the worth of selected instances in a reduced set. We choose to classify the instances in the original training set using the reduced set (by applying the k-NN algorithm), and calculate the predictive accuracy accordingly.

6.1.5 Pheromone Updating

The pheromone values are updated after each ant completes it tour, so that future ants can make use of this information in their search. The amount of pheromone in each instance selected in the current reduced set by each ant is updated according to equation 21:

$$\mu_i(t+1) = (1 - \alpha) * \mu_i(t) + \left(1 - \frac{1}{1 + (Q(t) - Q(t-1))}\right) * \mu_i(t) \quad (21)$$

Where:

μ_i (t) is the pheromone level of instance at time t.
Q (t) is the quality of the selected instance to classify the instances in the original set at iteration t.
α is the evaporation rate (we choose 0.1 in our method).

Using Equation 21, the pheromone levels are updated for the instances by increasing the pheromone for the selected instances in the reduced set if their selection enhances the quality compared to the previous ant, and vice versa (the quality of selected instances is computed using the fitness function mentioned in 6.1.4). If these instances are good, they become more attractive for future ants and more likely to be chosen. Furthermore, the pheromone values decrease for unselected instances using Equation 22.

$$\mu_i(t+1) = (1-\alpha) * \mu_i(t) \quad (22)$$

Where:

α is the evaporation rate (we choose 0.1 in our method).

6.1.6 Number of Ants

Selecting the number of ants to be used in ACO is one of the most important factors in this method. We think that the higher this value is, the better the results that can be expected, since the more ants that are used the more likely it is that the most important instances are kept. In our method, we investigate different values for the number of ants, starting from 250 ants and repeating the experiments by increasing by 250 ants each time, until we reach 1,250 ants. Below, we analyse the results obtained to consider the effect of varying the size of the ants.

6.2 Experimental Results for Instance Reduction using the ACO Algorithm

We investigated using the proposed algorithm as an instance-reduction method and compare it to the k-NN algorithm (Cover & Hart, 1967) and other instance-reduction methods. Each test consisted of 10 trials, each of which used one of 10 partitions of the data randomly selected from the datasets – i.e., 10-fold cross-validation (Kohavi, 1995). For each trial, 90% of the training instances were used for the training set, subset S was determined using each reduction technique (except for the k-NN algorithm, which retains all instances), and the remaining 10% of the instances were classified using only the instances remaining in S. The results were compared using a statistical paired t-test with confidence of 0.05. For each instance-reduction method, we counted the number of datasets in which the predictive accuracy was statistically improved (win) or statistically reduced (loss).

Table 11 compares the predictive accuracy using the k-NN algorithm and different instance-reduction methods. Our experiments show that there is no statistically significant effects on predictive accuracy after applying ENN, AllKnn, and AllKnnDrop5 on 13 datasets, and on 15 datasets after applying the ACO-IR method with 250 and 1,000 ants, on 16 datasets with 500 ants,

and on 14 datasets with 750 ants. Moreover, ACO-IR with 750, 1,000, and 1,250 ants achieved the highest number of datasets with a statistically significant increase in predictive accuracy. On the other hand, there was a statistically significant decrease after applying ENN, AllKnn, AllKnnDrop5, and ACO-IR with 250 ants on five datasets, and on two datasets when applying the ACO-IR method with 500 and 750 ants. There was a statistical decrease in predictive accuracy on only one dataset when using the ACO-IR method with 1,000 and 1,250 ants. It is clear that applying the DROP5 method achieved the worst results. When using DROP5, there was a statistically significant increase in predictive accuracy for only one dataset. Furthermore, for nine of the 22 datasets, using DROP5 led to a statistically significant decrease.

We can see that the average predictive accuracy after applying the ACO-IR method with 1,000 and 1,250 ants is the highest among the other instance-reduction methods, and the performance of ACO-IR is improved when increasing the number of ants used. However, from a certain threshold on, a flat-maximum effect is reached; increasing the number of ants only results in more execution time and no significant increase in predictive accuracy.

Another most interesting point pertains to the E-coli dataset, wherein there was a serious negative impact on the prediction accuracy after applying all instance-reduction methods except for the ACO-IR method. This means that applying ACO-IR yielded better results than the other methods did.

Datasets	Without pruning	ENN	AllKnn	DROP5	AllKnnDrop5	ACO-IR(250)	ACO-IR(500)	ACO-IR(750)	ACO-IR(1000)	ACO-IR(1250)
Iris	95.33	96.00	94.67	96.00	95.33	94.00	94.67	94.67	94.67	94.00
Voting	95.35	95.35	95.35	94.00	95.12	96.05	96.05	96.05	96.51	96.55
Vowels	96.79	**92.12** $^{-}$	**93.65** $^{-}$	**90.00** $^{-}$	**93.08** $^{-}$	**93.46** $^{-}$	**94.62** $^{-}$	96.15	95.77	95.77
Heart Cleveland	79.00	79.67	81.00	75.33	80.00	77.33	79.33	77.33	78.33	79.00
Glass	69.52	**60.95** $^{-}$	**61.90** $^{-}$	**60.48** $^{-}$	**62.86** $^{-}$	66.19	69.05	67.62	70.95	70.50
Liver disorders	62.06	57.35	60.29	60.59	60.88	60.29	59.12	60.59	61.18	61.50
Wine	95.88	93.53	94.12	96.47	94.12	95.88	95.29	95.88	96.47	96.00
Pima Indians diabetes	73.29	**71.18** $^{-}$	73.95	**70.26** $^{-}$	72.24	72.63	**70.92** $^{-}$	71.32	**70.92** $^{-}$	**71.00** $^{-}$
Promoters	92	93.00	94.00	**74.00** $^{-}$	94.00	95.00	93.00	94.00	94.00	95.00
Hepatitis	78.00	80.00	80.00	77.00	79.33	80.67	79.33	**82.00** $^{+}$	**82.67** $^{+}$	**82.67** $^{+}$
Vehicle	70.36	**66.55** $^{-}$	**66.19** $^{-}$	**63.93** $^{-}$	**67.26** $^{-}$	70.92	70.63	**71.00** $^{+}$	**71.20** $^{+}$	**71.4** $^{+}$
Pole-and-cart	58.59	60.29	60.20	56.90	57.61	57.00	58.39	59.20	60.20	60.29
Blood transfusion service	72.84	**78.38** $^{+}$	**77.70** $^{+}$	70.95	**76.49** $^{+}$	**70.41** $^{-}$	73.65	72.16	73.24	73.65
E-coli	79.39	**13.94** $^{-}$	**13.94** $^{-}$	**15.76** $^{-}$	**13.94** $^{-}$	**82.73** $^{+}$	**82.73** $^{+}$	**81.52** $^{+}$	**81.21** $^{+}$	**81.52** $^{+}$
Soybean	92.33	89.33	90.00	**76.67** $^{-}$	**90.33** $^{-}$	**87.00** $^{-}$	89.00	**86.33** $^{-}$	88.00	89.33
ZOO	92.00	88.00	**89.00** $^{-}$	91.00	90.00	93.00	94.00	94.00	94.00	94.00
Yeast	50.54	**56.15** $^{+}$	**56.82** $^{+}$	**53.65** $^{+}$	**53.65** $^{+}$	**52.64** $^{-}$	**54.66** $^{+}$	**53.00** $^{+}$	**52.90** $^{+}$	**53.00** $^{+}$
Led creator	66.60	**72.20** $^{+}$	**71.80** $^{+}$	68.10	**72.00** $^{+}$	**71.10** $^{-}$	**70.60** $^{+}$	**71.00** $^{+}$	**71.22** $^{+}$	**71.80** $^{+}$
Vertebral column	79.03	77.42	79.68	81.94	78.06	70.25	78.39	**73.00** $^{-}$	78.71	78.39
Ionosphere	64.00	64.00	64.00	**38.57** $^{-}$	64.00	63.50	64.00	63.90	64.00	63.90
Wave	80.26	**81.84** $^{+}$	**82.08** $^{+}$	80.88	**82.12** $^{+}$	**81.84** $^{+}$	**82.28** $^{+}$	**82.40** $^{+}$	**82.40** $^{+}$	**82.28** $^{+}$
Balance scale	83.23	82.58	82.58	**78.06** $^{-}$	83.06	81.61	83.06	80.65	84.00	84.56
Average	78.47	74.99	75.59	71.39	75.25	77.89	78.76	78.35	79.21	79.37
Win/tie/loss		4/13/5	4/13/5	1/12/9	4/13/5	2/15/5	4/16/2	6/14/2	6/15/1	6/15/1

Table 11. Empirical results comparing prediction accuracy using ENN, AllKnn, DROP5, AllKnnDROP5, and ACO-IR (with 250, 500, 750, 1,000, and 1,250 ants) pre-processing with k-NN.

Datasets	ENN (%)	AllKnn (%)	DROP5 (%)	AllKnnDrop5 (%)	ACO-IR (250) (%)	ACO-IR (500) (%)	ACO-IR (750) (%)	ACO-IR (1000) (%)	ACO-IR (1250) (%)
Iris	85	84	12	84	85	81	80	70	68
Voting	86	85	11	85	87	86	80	78	75
Vowels	86	86	42	87	90	89	86	82	81
Heart Cleveland	75	66	16	69	81	78	71	55	53
Glass	64	59	23	65	91	83	73	54	50
Liver disorders	59	46	26	54	74	73	61	53	51
Wine	85	85	10	85	93	86	78	71	71
Pima Indians diabetes	69	60	19	65	78	77	71	67	65
Promoters	88	86	16	86	99	89	85	78	76
Hepatitis	70	69	11	67	93	86	75	61	60
Vehicle	66	57	24	64	88	82	77	68	64
Pole-and-cart	72	60	29	68	77	68	62	57	55
Blood transfusion service	71	60	10	63	70	68	62	58	55
E-coli	78	70	13	72	78	76	73	67	64
Soybean	82	79	23	81	87	86	79	73	70
ZOO	82	81	15	82	97	95	77	60	58
Yeast	53	42	23	49	71	65	49	39	35
Led creator	67	65	11	66	83	63	58	52	50
Vertebral column	73	67	18	70	68	65	61	56	56
Ionosphere	75	73	9	77	73	74	71	68	67
Wave	83	72	17	76	81	76	70	65	65
Balance scale	77	72	10	73	74	83	74	70	68
Average	75	69	18	72	83	79	72	64	62

Table 12. Empirical results comparing the percentage of instances retained using different instance-reduction methods.

Table 12 shows the reduction in the number of instances after applying different instance-reduction methods. It is clear that applying DROP5 yielded the greatest reduction in the number of instances. Furthermore, when using ACO-IR, the reduction in the number of instances increased as more ants were used. We can see that ACO-IR (1,250) achieved the highest reduction among all methods except for DROP5. Furthermore, we there was no major difference in the amount of instance reduction when we increased the number of ants from 1,000 to 1,250.

From the above results, it is clear that applying the ACO-IR method achieved the best outcome in terms of predictive accuracy and the amount of instance reduction, compared to the other instance-reduction methods. Furthermore, the influence of increasing the number of ants used reached a flat-maximum effect.

Usually, the learning process is carried out just once on the training set, so it seems not to be a very important evaluation method. However, if the learning process takes too long it can become impractical for real applications. Table 13 shows a comparison of the average elapsed time (in minutes) when using ACO-IR for different numbers of ants. The average elapsed time was estimated using 10-fold cross-validation and computing the average total time taken by each fold. The experiments were conducted on an 8 GB machine and the CPU specification was i5 with speed equal to 2.5 GHz.

From Table 13, it is clear that for each dataset the average elapsed time increased as the number of ants increased. Furthermore, the average elapsed time was affected by the number of

instances and number of attributes for each dataset. We can observe that datasets with a large number of instances and attributes takes longer than other datasets.

Datasets	ACO-IR (250)	ACO-IR (500)	ACO-IR (750)	ACO-IR (1000)	ACO-IR (1250)
Iris	0.25	0.40	0.75	0.99	1.17
Voting	21.74	31.61	68.68	92.56	109.41
Vowels	20.96	33.75	62.24	83.88	99.15
Heart Cleveland	5.88	9.47	17.36	22.22	26.26
Glass	1.24	2.00	3.77	4.81	5.69
Liver disorders	3.75	6.05	10.83	14.88	17.59
Wine	1.18	1.90	3.49	4.62	5.42
Pima Indians diabetes	52.15	83.98	157.85	210.96	249.36
Promoters	1.39	2.24	4.13	5.51	6.51
Hepatitis	1.38	2.22	4.25	5.64	6.67
Vehicle	116.97	188.36	352.93	475.65	562.23
Pole-and-cart	781.25	1258.05	2358.99	3179.24	3757.96
Blood transfusion service	22.93	36.92	68.20	91.26	107.87
E-coli	3.81	6.14	11.70	15.37	18.17
Soybean	19.01	30.61	56.89	76.33	90.22
ZOO	0.47	0.76	1.46	1.95	2.29
Yeast	390.63	629.03	1179.50	1589.62	1878.98
Led creator	114.94	185.09	354.44	477.68	564.64
Vertebral column	2.61	4.21	7.38	9.81	11.59
Ionosphere	19.96	32.14	59.41	79.18	93.59
Wave	1718.75	2767.71	5189.79	6994.32	8267.52
Balance scale	20.23	32.58	61.13	82.41	97.41

Table 13: Comparison of elapsed time (in minutes) when using ACO-IR (with 250, 500, 750, 1,000, and 1,250 ants).

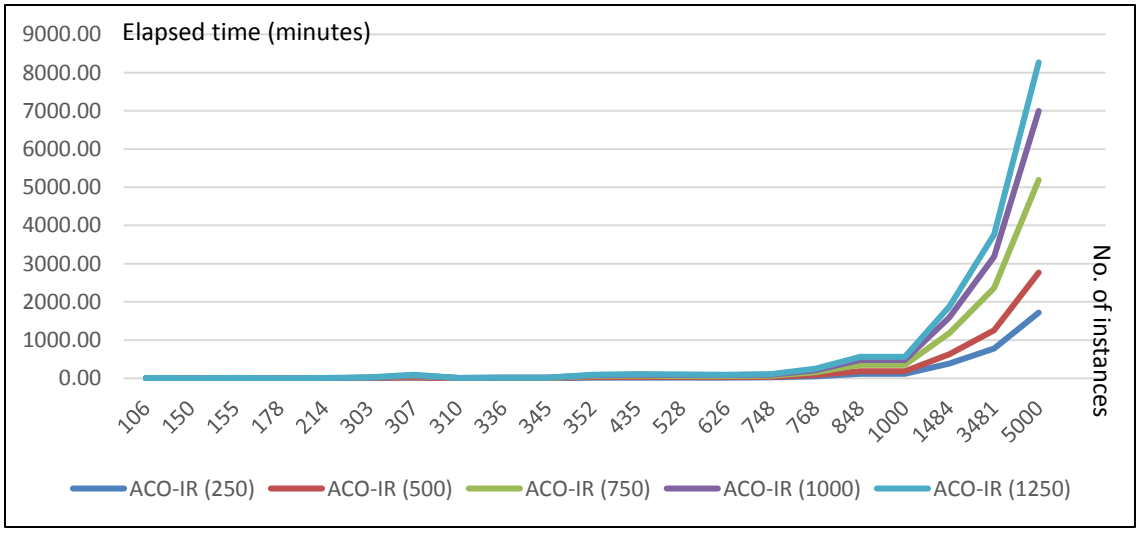


Figure 16. Comparing elapsed time for ACO-IR with size of training set.

Figure 16 shows the comparison between the elapsed time and number of instances in training set. It is clear the elapsed time is increased as the number of instances is increased.

6.3 Pruning Classification Rule using ACO-IR

We will now investigate the new method for instance reduction using ACO-IR as a pre-pruning method before applying different rule-induction methods with different numbers of ants. We also compare the results achieved for preceding rule induction with other instance-reduction methods in terms of the predictive accuracy and number of generated rules.

Table 14 shows the results obtained for CN2 and applying the pre-pruning methods with respect to the predictive accuracy. Our results show that there was no statistically significant decrease on predictive accuracy after applying AllKnnDrop5 and ACO-IR with different numbers of ants. Furthermore, there was a statistically significant increase in predictive accuracy for 4, 5, 7, 11, and 2 datasets when using ACO-IR (250), ACO-IR (500), ACO-IR (750), ACO-IR (1,000 or 1,250), and AllKnnDrop5, respectively, which means that ACO-IR with 1,000 and 1,250 ants achieved the best result of the methods. Moreover, we can conclude that preceding CN2 with instance-reduction methods did not adversely affect the predictive accuracy on most datasets. However, when using DROP5, there was no statistically significant increase in predictive accuracy for any of the datasets. Furthermore, for 15 of the 22 datasets, using DROP5 led to a statistically significant decrease.

Table 15 summarizes the effect of instance selection (pruning training data) on generalization of the RISE algorithm. Our experiments show that the predictive accuracy did not statistically decrease after applying ACO-IR with 750, 1,000, and 1,250 ants. Furthermore, ACO-IR with 1,000 and 1,250 ants yielded statistically significant increases in predictive accuracy on six datasets, which is the highest achievement among the methods. It is clear that the achievement of ACO-IR improved as the number of ants increased.

Table 16 clearly shows that applying AllKnn, AllKnnDrop5, and ACO-IR (1,000) before PRISM yielded a statistically significant decrease in the predictive accuracy for only one dataset, and when applying ACO-IR (1,250), none of the datasets were adversely statistically affected. The results reveal that applying ACO-IR (1,000) and ACO-IR (1,250) gave the best result regarding the number of datasets where the predictive accuracy had a statistically significant increase. Moreover, we observed that the predictive accuracy for ACO-IR was improved as the number of ants used increased.

From the previous results, we can see that applying ACO-IR is the safest method among the other instance-reduction methods in terms of statically decreasing in predictive accuracy. Figure 17 shows the amount of reduction in the number of rules using different instance-reduction methods for each rule-induction approach. It is clear that the most reduction rate was achieved by using DROP5 for all rule-induction methods, followed by ACO-IR (1000) and ACO-IR (1,250). Furthermore, when using ACO-IR we noticed that the reduction in the number of generated rules increased by increasing the number of ants used. However, there was no major difference in the amount of reduction when we increased the number of ants from 1,000 to 1,250

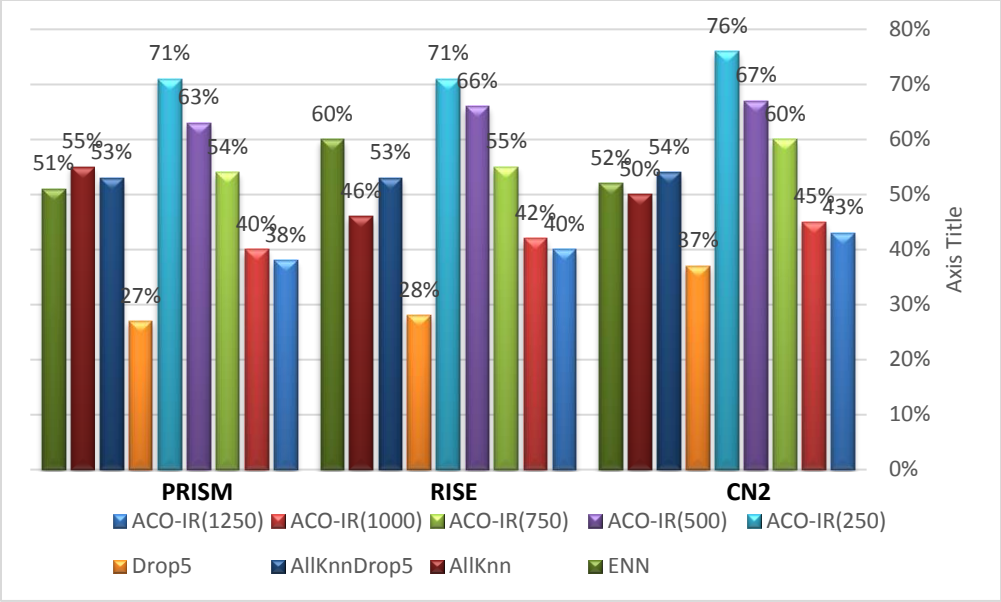


Figure 17. Amount of reduction in number of generated rules using different instance-reduction methods.

6.4 Conclusion

We proposed a new method for instance reduction based on the principles of ACO, and called this ACO-IR. We compared ACO-IR with various instance-reduction methods using k-NN algorithm. Moreover, we investigated the effect of varying the number of ants when using the ACO-IR method. The results of our experiments reveal that the ACO-IR with 1,250 and 1,000 ants achieved the best results in terms of predictive accuracy and the amount of instance reduction. We then investigated preceding three different types of rule induction with ACO-IR. Our experiments show that for most datasets, pruning the training set using ACO-IR significantly reduced the number of rules generated by CN2, RISE, and PRISM, without adversely affecting predictive performance. Furthermore, ACO-IR improved in terms of its predictive accuracy and reduction of generated rules as the number of ants increased. The results show that using ACO-IR with 1,000 and 1,250 ants achieved the best results of the instance-reduction methods in terms of reduction in generated rules, and predictive accuracy, for the three rule-induction methods.

However, there was no major improvement when increasing the number of ants from 1,000 to 1,250, and a flat-maximum effect appeared to be reached.

Datasets	Without pruning	ENN	AllKnn	DROP5	AllKnnDrop5	ACO-IR (250)	ACO-IR (500)	ACO-IR (750)	ACO-IR (1,000)	ACO-IR (1,250)
Iris	89.98	92.00	92.67	80.67	93.34	**92.67** +	**94.00** +	**93.33** +	**93.32** +	**94.00** +
Voting	95.34	95.10	95.33	85.35	95.57	94.19	95.11	94.64	95.34	95.11
Vowels	67.11	65.97	66.75	85.07	67.31	70.00	67.90	**69.61** +	**71.17** +	**71.17** +
Heart Cleveland	80.66	76.66	77.33	71.66	79.34	75.00	81.68	76.33	81.34	81.68
Glass	64.76	58.05	61.98	51.92	66.22	66.66	65.23	63.82	**68.58** +	**68.58** +
Liver disorders	66.77	64.11	65.64	60.3	66.52	62.11	63.21	63.54	63.82	63.21
Wine	91.77	94.11	93.52	70.00	95.28	91.76	**95.28** +	**94.10** +	**95.28** +	**95.28** +
Pima Indians diabetes	70.30	73.16	74.70	73.40	72.10	72.76	72.64	73.03	73.95	73.03
Promoters	85.00	81.00	80.00	63.00	80.00	81.00	86.00	81.00	85.00	86.00
Hepatitis	78.65	80.00	80.00	52.67	79.34	**82.65** +	79.32	77.33	**84.00** +	**82.65** +
Vehicle	57.85	60.10	60.71	54.99	60.10	**60.37** +	58.22	**60.40** +	**61.00** +	**61.00** +
Pole-and-cart	61.68	63.88	66.24	62.56	63.51	62.20	**64.80** +	**64.90** +	**64.70** +	**64.90** +
Blood transfusion service	75.68	76.61	76.35	73.11	75.96	73.12	76.08	74.70	76.60	76.61
E-coli	79.10	83.31	80.91	73.34	80.90	79.99	79.99	80.56	80.60	80.90
Soybean	86.32	82.67	83.01	63.00	83.32	81.33	84.65	85.67	85.00	85.67
ZOO	92.00	87.00	90.00	81.00	89.00	93.00	**94.00** +	93.00	**94.00** +	**94.00** +
Yeast	48.98	55.47	56.43	51.82	56.56	**51.09** +	49.66	**51.41** +	**51.09** +	**51.09** +
Led creator	72.30	72.30	71.30	68.90	71.90	72.70	71.30	72.60	72.40	72.70
Vertebral column	80.96	83.21	81.28	81.28	82.24	78.39	81.62	81.93	**82.57** +	**82.24**+
Ionosphere	89.43	85.71	86.56	53.71	85.71	89.52	92.00	91.13	91.42	91.13
Wave	69.70	70.38	70.74	67.96	71.38	70.38	**71.64** +	**72.00** +	**72.10** +	**72.00** +
Balance scale	75.30	74.70	74.34	67.10	74.34	74.19	76.30	76.19	76.51	76.30
Average	76.35	76.16	76.63	67.86	76.82	76.87	77.30	76.87	78.17	78.15
Win/tie/loss		2/19/1	2/19/1	0/7/15	2/20/0	4/18/0	5/17/0	7/15/0	11/11/0	11/11/0

Table 14: Empirical results for predictive accuracy using ENN, AllKnn, DROP5, AllKnnDROP5, and ACO-IR (with 250, 500, 750, 1,000, and 1,250 ants) pre-pruning with CN2.

Datasets	Without pruning	ENN	AllKnn	DROP5	AllKnnDrop5	ACO-IR (250)	ACO-IR (500)	ACO-IR (750)	ACO-IR (1,000)	ACO-IR (1,250)
Iris	95.33	94.00	94.67	94.01	94.67	95.33	94.00	95.33	94.66	95.33
Voting	95.10	95.32	95.79	93.25	95.32	94.87	95.10	95.32	95.56	95.10
Vowels	92.68	88.87	89.25	85.97	89.63	90.01	90.77	92.32	92.69	92.69
Heart Cleveland	77.00	77.01	75.32	71.01	75.01	76.33	77.65	74.35	74.34	75.32
Glass	67.14	62.85	64.77	52.37	65.70	65.72	69.05	64.75	68.10	67.05
Liver disorders	65.29	61.18	62.00	57.05	65.23	59.70	62.65	61.47	63.52	63.52
Wine	97.64	95.28	96.46	88.83	97.64	95.28	97.64	96.46	96.46	95.28
Pima Indians diabetes	67.63	68.29	68.37	68.56	67.70	**71.71** +	**71.32** +	**72.25** +	**72.10**+	**72.25** +
Promoters	86.00	92.00	88.00	67.00	87.00	**92.00** +	**94.00** +	**92.00** +	**94.00**+	**94.00+**
Hepatitis	80.67	80.67	80.66	52.00	80.67	80.00	**78.01** +	78.67	79.35	79.35
Vehicle	70.35	68.47	66.55	65.36	67.62	69.88	70.71	70.00	70.50	70.71
Pole-and-cart	61.87	62.18	65.49	58.81	64.24	60.50	62.46	62.10	62.40	62.46
Blood transfusion service	73.92	79.19	77.84	74.87	77.34	70.41	71.74	72.03	72.57	73.92
E-coli	84.76	85.75	85.46	83.02	86.35	85.76	86.35	**86.37** +	**86.97** +	**86.36+**
Soybean	91.00	87.67	87.66	82.67	88.33	84.67	85.65	89.66	88.33	89.66
ZOO	96.00	89.00	93.00	89.00	93.00	94.00	94.00	94.00	94.00	94.00
Yeast	52.97	57.56	58.25	53.99	56.83	**55.14** +	**57.24** +	52.03	**57.03** +	**57.24** +
Led creator	72.60	72.40	72.60	69.40	72.80	72.80	71.90	**73.00** +	**73.10** +	**73.00 +**
Vertebral column	82.91	81.60	81.93	81.30	82.90	**76.43** -	**79.04** -	81.60	82.25	82.90
Ionosphere	92.56	91.42	91.71	77.42	90.56	90.27	90.57	90.27	90.01	90.01
Wave	81.84	82.18	83.26	79.06	82.82	80.40	83.00	82.80	**83.50** +	**83.00+**
Balance scale	78.06	81.13	80.97	77.75	81.62	77.60	78.86	78.38	78.38	78.86
Average	80.15	79.73	80.00	73.76	80.14	79.04	80.08	79.78	80.45	80.55
Win/tie/loss		3/17/2	4/16/2	0/8/14	3/17/2	3/18/1	4/17/1	4/18/0	6/16/0	6/16/0

Table 15: Empirical results for predictive accuracy using ENN, AllKnn, DROP5, AllKnnDROP5, and ACO-IR (with 250, 500, 750, 1,000, and 1,250 ants) pre-pruning with RISE.

Datasets	Without pruning	ENN	AllKnn	DROP5	AllKnnDrop5	ACO-IR (250)	ACO-IR (500)	ACO-IR (750)	ACO-IR (1000)	ACO-IR (1250)
Iris	91.40	88.20	88.80	79.20	88.80	**100.0** $^{+}$	89.30	**100.0** $^{+}$	**100.0** $^{+}$	**100.0 +**
Voting	92.50	95.50	95.70	93.10	**96.20**	92.70	**93.80** $^{+}$	**93.50** $^{+}$	**94.50** $^{+}$	**93.80 +**
Vowels	52.40	50.70	51.10	42.40	51.10	53.00	52.60	53.10	52.60	52.60
Heart Cleveland	68.00	74.00	73.90	62.70	72.40	66.60	**71.30** $^{+}$	68.90	**72.20 +**	**73.90 +**
Glass	43.90	47.20	48.70	32.90	48.30	**51.50** $^{+}$	**49.00** $^{+}$	**49.00** $^{+}$	**51.10 +**	**51.50 +**
Liver disorders	47.90	56.90	53.60	51.20	52.40	**49.10** $^{+}$	**53.00** $^{+}$	**52.70** $^{+}$	**52.90 +**	**53.00 +**
Wine	86.30	83.90	83.90	69.80	86.30	84.89	85.20	85.20	87.50	86.30
Pima Indians diabetes	62.80	63.20	64.00	60.40	63.40	**68.90** $^{+}$	**64.00** $^{+}$	**70.30** $^{+}$	**70.30** $^{+}$	**70.30 +**
Promoters	73.00	77.00	74.00	52.00	72.00	68.00	70.00	70.00	70.50	72.00
Hepatitis	69.30	78.70	77.30	79.30	74.60	**79.90** $^{+}$	69.90	**80.20** $^{+}$	**81.20** $^{+}$	**81.20 +**
Vehicle	58.70	57.60	59.30	50.00	59.30	56.43	58.00	59.1.0	59.30	59.30
Pole-and-cart	52. 50	56.20	56.60	48.70	55.00	**54.00** $^{+}$	**56.30** $^{+}$	**56.80** $^{+}$	**56.60** $^{+}$	**56.80 +**
Blood transfusion service	71.70	76.4	72.70	69.20	73.20	67.50	**67.30** $^{-}$	**67.00** $^{-}$	**67.00** $^{-}$	69.00
E-coli	73.30	79.00	78.40	69.60	78.40	**75.40** $^{+}$	**76.60** $^{+}$	**77.20** $^{+}$	**78.10** $^{+}$	**78.40 +**
Soybean	79.50	73.90	73.40	56.30	74.20	**72.90** $^{-}$	**74.70** $^{-}$	76.30	76.00	76.60
ZOO	92.00	84.00	88.00	85.00	87.00	90.00	86.20	90.00	90.00	90.00
Yeast	43.80	49.30	46.40	41.70	46.70	**38.00** $^{-}$	51.20	**39.00** $^{-}$	41.00	46.40
Led creator	71.70	72.40	71.60	67.40	72.10	71.70	71.70	71.10	71.30	71.70
Vertebral column	73.40	78.00	74.20	75.40	75.50	**71.70** $^{-}$	**77.60** $^{+}$	**76.00** $^{+}$	**77.60** $^{+}$	**76.00 +**
Ionosphere	86.90	87.50	89.30	53.30	88.80	87.00	84.30	87.10	87.60	87.00
Wave	59.30	63.10	63.10	54.30	63.50	**65.00** $^{+}$	**76.80** $^{+}$	**76.40** $^{+}$	**76.80 +**	**76.40 +**
Balance scale	62.70	72.10	73.00	52.30	73.00	**55.00** $^{-}$	**66.30** $^{+}$	**65.30** $^{+}$	**66.50 +**	**66.50 +**
Average	69.55	71.13	70.77	61.19	70.55	69.06	70.23	71.1	71.83	72.21
Win/tie/loss		9/11/2	7/14/1	1/9/12	6/15/1	8/10/4	10/10/2	11/9/2	12/9/1	12/10/0

Table 16: Empirical results for predictive accuracy using ENN, AllKnn, DROP5, AllKnnDROP5, and ACO-IR (with 250, 500, 750, 1,000, and 1,250 ants) pre-pruning with PRISM.

Chapter 7: Discussion and Future Works

7.1 Thesis Summary

In Chapter 2, we introduced the field of machine learning, and provided insights into some of the learning algorithms that can be used. Furthermore, we reviewed the different kinds of pruning techniques and explained the advantages of using these to reduce the complexity of learned classifiers. Our work was concerned with investigating whether new pre-pruning techniques for rule-induction methods can help in reducing the complexity of rule sets by reducing the number of generated rules, without adversely affecting the predictive accuracy. ACO is a relatively new metaheuristic, which means that there is certainly still significant potential for improvement and development. Chapter 3 discussed the principles of ACO. We described how ACO mimics the behaviour of real ant colonies, and differentiated between real and artificial ants. Moreover, we defined the elements related to the ACO method for solving combinatorial optimization problems. Finally, we reviewed different applications in which ACO achieved impressive results. In Chapter 4, we formalized our proposed method for reducing the complexity of the produced rules set from rule-induction methods, taking into consideration the effect on predictive accuracy. We introduced all the materials required to undertake a series of experiments to address this proposal.

In Chapter 5, we began by formalizing our proposed technique to precede the rule-induction method with instance-reduction methods that try to remove border instances, which can smooth the decision boundaries between different instances. We went on to undertake an empirical

study to determine the effect on the complexity of rule sets (roughly represented here by the number of generated rules) and predictive accuracy. The remainder of Chapter 5 presented the experiments that were conducted in this study, and analysed their results.

Chapter 6 explained in detail how to apply the concept of ACO as an instance-reduction method, and presented the experiments used to investigate the performance of the new algorithm. We then investigated the results obtained from preceding the different types of rule induction with the new instance-reduction method based on ACO, in terms of predictive accuracy and number of generated rules.

7.2 Main Findings

The results presented in Chapters 5 and 6 indicate that preceding rule-induction methods with instance-reduction methods is indeed a promising technique for reducing the generated rule set without adversely affecting the predictive accuracy. Throughout our experiments, we ensured that the predictive performance was measured on unseen test data. We did this by applying a 10-fold stratified cross-validation testing strategy.

The main contributions and findings of this thesis may be summarized as follows:

- Preceding rule-induction methods with instance-reduction methods was found to significantly reduce the number of generated rules without adversely affecting the predictive accuracy, and may even improve the accuracy in some cases.

- The best results achieved by preceding rule induction with instance-reduction methods was with datasets that had a low number of total attributes with respect to the number of instances.
- For each dataset, if one or more of the combinations of instance-reduction and rule-induction method resulted in statistically significant increases in predictive accuracy, then none of the combinations resulted in a statistically significant decrease, and vice versa.
- ACO can be used to solve combinatorial optimization problems, and we succeeded in designing a novel instance-reduction method based on ACO principles (ACO-IR).
- When applying ACO-IR with different numbers of ants, we observed that better results are achieved when increasing the number of ants before reaching the flat-maximum effect.
- When applying ACO-IR, a flat-maximum effect was reached when increasing the number of ants, at which point there was no major reduction in the number of generated rules, or improvement in the predictive accuracy. Moreover, we expect that the predictive accuracy may be adversely affected as the number of ants is increased after reaching the flat-maximum effect. This may be the cause of overfitting and exaggerated focus on certain instances in the training set.

Summing up these results, we come to the final conclusion that applying instance reduction techniques as a pre-pruning process for rule induction reduces the number of rules generated, and may improve the predictive accuracy in some cases.

7.3 Future Work

This section presents several suggestions on how the work presented in this thesis might be extended. Some of the ideas presented here could not be incorporated in this thesis because the author did not have the required access to resources or data; however, most of the ideas were left unaddressed simply due to a lack of time available for the project.

- There are several design decisions and several possible parameters when applying ACO-IR, which can be used to fine-tune the performance of the algorithm. More research is needed to better understand the interactions between these, and how each of them influences the algorithm performance (i.e., the evaporation rate, using another heuristic function, using pheromone update function, etc.).
- More research is needed to understand the best situation and dataset characteristics for applying ACO-IR as a pre-pruning process.
- Investigations are needed to understand the effect of preceding different learning algorithms with the ACO-IR algorithm.
- There is a need to investigate other instance-reduction methods that conduct instance pruning more carefully, such as c-pruner (Zhao et al., 2003).
- Investigation of the effect of preceding instance-reduction methods with rule induction on noisy datasets is also highly recommend.

References

Aha, D. W., Kibler, D. and Albert, M. K. (1991). “Instance-based learning algorithm”, *Machine Learning*, Vol. 6, pp. 37–66.

AlBalas, F. (2000). Developing new feature selection methods for discrete and continuous class prediction, Unpublished MSc Thesis, Computer science department, Jordan University.

Angiulli F. (2005). Fast condensed nearest neighbor rule, Technical report, *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, Germany.

Babu, T. R. and Murty, M. N. (2001). "Comparison of genetic algorithm based prototype selection schemes*", Pattern Recognition*, Vol. 34, pp. 523–525.

Blanco-Vega, R., Hernández-Orallo, J. and Ramírez-Quintana, M. (2004). “Analysing the trade-off between comprehensibility and accuracy in mimetic models”, *Lecture Notes in Computer Scienc*e. Einoshin Suzuki and Setsuo Arikawa (eds.). *7th International Conference, Padova, Italy*, Vol. 3245, Springer-Verlag, Berlin. pp. 338–346.

Bonabeau, E., Dorigo, M. and Theraulez, G. (1999). *Swarm Intelligence: From natural to artificial intelligence*, Oxford University Press, New York.

Bramer, M. A. (2000). “Inducer: A rule induction workbench for data mining”, in Z. Shi, B. Faltings and M. Musen (eds), *Proceedings of the 16th IFIP World Computer*

Congress Conf. Intelligent Information Processing, Publishing House of Electronics Industry, Beijing, pp. 499–506.

Brighton H. and Mellish C. (2002) *Advances in instance selection for instance-based learning algorithms*. Data mining and knowledge discovery, vol. 6, no. 2, pp. 153–172.

Brunk, C. A., and Pazzani, M. J. (1991). An investigation of noise-tolerant relational concept learning algorithms. *Lawrence A. Birnbaum and Gregg C. Collins (eds). Proceedings of the 8th International Workshop on Machine Learning (ML-91)*, Morgan Kaufmann, Evanston, IL, pp. 389–393.

Bullnheimer, B. (1999). *Ant Colony Optimization in vehicle routing*. Doctoral thesis, University of Vienna, Jan.

Cendrowska, J. (1987). "PRISM: An Algorithm for Inducing Modular Rules", *International Journal of Man-Machine Studies* Vol. 27 No. 4, pp. 349–370.

Cervone, G., Panait, L. A. and Michalski, R. S. (2001). "The development of the AQ20 learning system and initial experiments", Mieczysław A. Kłopotek, Maciej Michalewicz and Sławomir T. Wierzchon (eds). *Proc. of the 10th Int. Symposium on Intelligent Information Systems*, Poland, pp. 13-29.

Clark, P. and Boswell, R. (1991). "Rule induction with CN2: Some recent improvement", in Kodratoff, Y. (ed.), *Machine Learning – Proceedings of the Fifth European Conference (EWSL-91)*, Springer-Verlag, Berlin, pp. 151–163.

Clark, P. and Niblett, T. (1989). "The CN2 induction algorithm", *Machine Learning*, Vol. 3, pp. 261–283.

Cohen, W. (1993). “Efficient pruning methods for separate-and-conquer rule learning systems”, in Bajcsy, R. (ed.), *Proceedings of the 13th international joint conference on Artificial Intelligence*, Morgan Kaufmann, Chambery, France, pp. 988–994.

Cohen, W. (1995). "Fast effective rule induction". *Armand Prieditis and Stuart J. Russell*. (ed.), Proc. *of the 12th Int. Conf. on Machine Learning*, Tahoe City, CA, pp. 115–123.

Cohen, W. and Singer Y. (1999). "A simple, fast and effective rule learner", in Hendler, J. and Subramanian, D. (eds.), *Proceedings of the Sixteenth National Conference on Artificial Intelligence*. AAAI/MIT Press, Menlo Park, CA, pp.335–342.

Colorni, A., Dorigo, M., Maniezzo V. and Trubian M. (1994). "Ant system for job shopping scheduling", *Belgian Journal of Operations Research, Statistics and Computer Science*, Vol. 34 No. 1, pp. 39–53.

Costa, D. and Hertz, A. (1997). "Ants can color graphs", *Journal of the Operational Research Society*, Vol. 48, pp. 295–305.

Cover, T. and Hart, P. (1967). "Nearest neighbor pattern classification", in IEEE Transaction. Volume (13) Issue 1, pp. 21-27.

Dain, O., Cunningham R. and Boyer S. (2004). "IREP++ a faster rule learning algorithm", in Michael w, Dayal, U., Kamath, C. and Davis, B. (eds.), *Proceeding Fourth SIAM Int. Conf. Data Mining*, Lake Buena Vista, FL, pp. 138–146.

Domingos, P. (1994). "The RISE system: Conquering without separating", *Proceedings of the Sixth IEEE International Conference on Tools with Artificial Intelligence*, IEEE Computer Society Press, New Orleans, LA, pp. 704-707.

Domingos, P. (1997). *A Unifying Approach to Concept Learning*, Unpublished PhD thesis, University of California, Irvine.

Dorigo, M., Maziezzo, V. and Colorni A., (1996). "The Ant System: Optimization by Colony of cooperating Ants", *IEEE Transaction on SML 26(1)*, pp. 29–41.

Dorigo, M., Birattari, M., Blum, C., Gambardella, L. M., Mondada, F. and Stützle, T. (eds), *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop*, Vol. 3172 of LNCS, Springer-Verlag, Berlin, Germany, pp. 25–36.

Duda R.O., Hart P.E. and D.G. Stork. *Pattern classification*. Wiley interscience, 2012.

Eibe F. and Ian H. W. (1998). "Generating accurate rule sets without global optimization", in Shavlik, J. (ed.), *Fifteenth International Conference on Machine Learning*, Morgan Kaufmann, pp. 144–151.

El Hindi, K. and Alakhras, M. (2009). "Eliminating border instance to avoid overfitting", in Alakhras, A. P. and dos Reis, A. (eds.), *Proceeding of Intelligent Systems and Agents*, IADIS Press, Algarve, Portugal, pp. 93–99.

Fayed, H. A. and Atiya A. F. (2009). *A novel template reduction approach for the K-nearest neighbor method*. IEEE Transactions On Neural Networks , vol. 20, N. 5, pp. 890–896.

Freund, Y. and Schapire, R. E. (1997). "A decision-theoretic generalization of on-line learning and an application to boosting", *Journal of Computer and System Sciences*, Vol. 55 No. 1, pp. 119–139.

Fürnkranz, J. (1994). *FOSSIL: a robust relational learner.* In the Proceedings of the European conference on Machine Learning. pp. 122 – 137. Catania, Italy: Springer-Verlag.

Fürnkranz, J., Gamberger, D. and Lavrac, N. (2012). *Foundations of Rule Learning*, Springer-Verlag, Berlin.

Fürnkranz, J. and Widmer, G. (1994). "Incremental reduced error pruning", in Cohen. W. and Hirsh H. (eds.), *Proceedings of the 11th International Conference on Machine Learning (ML-94)*, Morgan Kaufmann, New Brunswick, NJ, pp. 70–77.

Gadodiya, S.V. and Chandak, M.B. (2013). *Prototype Selection Algorithms for kNN Classifier: A Survey* in International Journal of Advanced Research in Computer and Communication Engineering Vol.2, Issue 12, December 2013.

Galea, M. (2002), *Applying Swarm Intelligence to Rule Induction*, MS thesis, University of Edinburgh, Edinburgh, UK.

Gambardella L. M. and Dorigo (1996). "Has SOP: A hybrid ant system for sequential ordering problem", *Technical report 11-97*, AAAI Press, CA, pp. 114–119.

Gamberger, D., Lavrac, N. and Dzerski, S (1996). "Noise elimination in inductive concept learning: A case study in medical diagnosis", *Lecture Notes in Computer Science, 7th International Workshop, ALT '96 Sydney*, Vol. 1160, Springer-Verlag, Berlin, pp. 199–212.

Gates, G. W. (1972). "The reduced nearest neighbor rule", *Institute of Electrical and Electronics Engineers Transactions on Information theory*, Vol. 18 No. 3, pp. 431–433.

Grochowski M. and Jankowski N. (2004), *Comparison of instance selection algorithms. ii. results and comments*, LNCS 3070, pp. 580–585

Grudzinski, K. (2008), "EkP: A fast minimization-based prototype selection algorithm", *Intelligent Information System XVI*, Academic Publishing House EXIT, Warsaw, pp. 45-53

Grudzinski, K., Grochowski, M. and Duch, W (2010). "Pruning classification rules with reference vector selection methods", *Artificial Intelligence and Soft Computing* Vol. 6113, Springer LNCS, pp. 347–354.

Hart, P. E. (1968). "The condensed nearest neighbor rules", *Institute of Electrical and Electronics Engineers Transactions on Information Theory*, Vol. 14 No. 3, pp. 515–516.

Kira K. and Rendell L. (1992). *A practical approach to feature selection*. In Proceedings of the ninth international workshop on Machine learning, pp. 249–256. Morgan Kaufmann Publishers Inc., 1992.

Kohavi, R. (1995). "A study of cross-validation and bootstrap for accuracy estimation and model selection", in Mellish, C. (ed.), *Proceedings of 14th International Joint Conference on Artificial Intelligence*, Morgan Kaufmann. San Francisco, CA, pp. 1137–1143.

Koller D. and Sahami M. (1996). *Toward optimal feature selection*. International Conference on Machine Learning.

Larose, D. (2005). *Discovering Knowledge in Data, an Introduction to Data Mining*, 1st ed., Wiley-Blackwell.

Maldonado S., Weber R., and Famili F. (2014). "Feature selection for high-dimensional class-imbalanced data sets using Support Vector Machines," Information Sciences, vol. 286, pp. 228–246.

Maloof M. A. and Michalski R. S. (2000). *Selecting examples for partial memory learning. Machine Learning*. 41. pp. 27-52.

McClelland, J. L., and Rumelhart, D. E. (1986). *Parallel Distributed Processing. Explorations in the Microstructure of Cognition*, Vol. 2: Psychological and Biological Models, MIT Press, Cambridge, MA.

Michalski, R. S. and Kaufman, K.A. (2001). "The AQ19 system for machine learning and pattern discovery: A general description and user guide", *Reports of the Machine Learning and Inference Laboratory*, *MLI 01-2*, George Mason University, Fairfax, VA.

Michalski, R. S., Mozetic, I., Hong, J. and Lavrac, N. (1986). *The Multi-Purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains*, American Association of Artificial Intelligence, Los Altos, CA, Morgan Kaufmann, pp. 1041–1045.

Mitchell T. M. (1997). *Machine Learning*, McGraw-Hill, New York, NY.

Murphy P. M., and Aha D. W. (1994). *UCI Repository of Machine Learning Databases*, available via anonymous ftp at ics.uci.edu in the pub/machine-learning-databases directory.

Nikolaidis K., Wu Q.H. and Goulermas J.Y. (2011), *A class boundary preserving algorithm for data condensation*, Pattern Recognition, vol. 44, pp. 704-715.

Nilsson, N. J. (1996). *Introduction to Machine Learning*, available at http://robotics.stanford.edu/people/nilsson/MLBOOK.pdf.

Olvera-Lopez J.A., Kittler J. and Carrasco-Ochoa J.A. (2010), "*A review of instance selection methods*", Journal of Artificial Intelligence Review, Springer Netherlands, vol.34, no.2, pp. 133-143.

Othman, O. and Bryant, C. (2013), "Preceding rule induction with instance-reduction methods", *Perner, Petra (eds.)* in *Proc. of the 9th International Conference on Machine Learning and Data Mining in Pattern Recognition*, Springer-Verlag, Berlin, pp. 209–218.

Othman, O., and Bryant, C. (2015). "Pruning classification rules with instance reduction methods", *International Journal of Machine Learning and Computing*, Vol. 5 No. 3, pp. 187–191.

Othman, O. and El Hindi, K. (2004). "Rule reduction technique for RISE algorithm", *Advances in Modeling, Series B: Signal Processing and Pattern Recognition*, Vol. 47, pp. 2.

Pagallo, G. and Haussler, D. (1990). "Boolean feature discovery in empirical learning", *Machine Learning*, Vol. 5, Kluwer Academic Publishers, Boston, pp. 71-99.

Pappa, G. L. and Freitas, A. (2008). "Discovering new rule-induction algorithms with grammar-based genetic programming", in *Soft Computing for Knowledge Discovery and Data Mining. Part II*, Springer, pp. 133–152.

Parepinelli, R. S., Lopez, H. S. and Freitas, A. (2002). "Ant Colony Algorithm for classification rule discovery", in Newton, H. A. a. S. a. C. (ed.), *Data Mining: Heuristic Approach*, pp. 191-208. Idea Group Publishing, London.

Pham, D. (2012). *A novel rule-induction algorithm with improved handling of continuous valued attributes*. PhD Dissertation, Cardiff Univ., Cardiff.

Pham, D. T. and Bigot, S. (2003). "Rules-5: A rule-induction algorithm for classification problems involving continuous attributes", *Proc. Inst. Mechanical Engineers*, Part C, Vol. 217, pp. 1273–1285, SAGE.

Pham, D. T., Bigot, S. and Dimov, S. (2004). "A rule merging technique for handling noise in inductive learning", *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, Vol. 218 (C), pp. 1255–1268, SAGE.

Quinlan, J. R. (1986). "Induction of decision trees", *Machine Learning*, Vol. 1, pp. 81–106.

Quinlan, J. R. (1993). *C4.5: A program for machine learning*, Morgan Kauffmann, San Mateo, CA.

Ritter, G. L., Woodruff, H. B., Lowry, S. R. and Isenhour, T. L. (1975). "An algorithm for a selective nearest neighbor decision rule", *IEEE Transactions on Information Theory*, Vol. 21 No. 6, pp. 665–669.

Rivest, R. (1987). "Learning decision lists", *Machine Learning*, Vol. 2, pp. 229–246.

Schapire, R. and Singer, Y. (1998). "Improved boosting algorithms using confidence-rated predictions", in Bartlett, P. L. and Mansour, Y. (eds), *Proceeding*

COLT' 98 Proceedings of the Eleventh Annual Conference on Computational Learning Theory, ACM Press, New York, NY, pp. 80–91.

Shahzad, W. (2010). *Classification and Associative Classification Rule Discovery using Ant Colony Optimization*, National University, Islamabad, Pakistan.

Shehzad, K. (2009). *New Rule Induction Algorithms with Improved Noise Tolerance and Scalability*, PhD thesis, Systems Engineering Division, University of Wales Cardiff, Cardiff.

Shirbhate, D. and Gupta, R. (2015). *Data mining and knowledge discovery*, *International Journal of Research in Science & Engineering*, Vol. 1 Special issue 1, pp. 384–389.

Simon, H. A. (1983). "Why should machines learn?" in Michalski, R. S., Garbonell J. G. and Mitchell, T. M. (eds.), *Machine Learning: An Artificial Intelligence Approach*, Tiogo, Palo Alto, CA, pp. 25–37.

Socha, K. (2008). *Ant Colony Optimization for Continuous and Mixed-Variable Domains*. Unpublished doctoral dissertation, University Libre de Bruxelles, Belgium.

Socha, K., (2004). *ACO for continuous and mixed-variable optimization*, in M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada and T. St¨utzle (eds), Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004, Vol. 3172 of LNCS, Springer-Verlag, Berlin, Germany, pp. 25–36.

Stanfill, C. and Waltz, D. (1986). "Toward memory-based reasoning", *Communications of the ACM*, Vol. 29 No. 12, pp. 1213–1228.

Sun X. and Chan P. K. (2014), *An Analysis of Instance Selection for Neural Networks to Improve Training Speed*.13th International Conference on Machine Learning and Applications , Detroit, MI, USA

Swaminathan, S. (2006). *Rule Induction Using Ant Colony Optimization for mixed variable attributes*, Master's thesis, University of Texas Tech.

Tomek, I. (1976). "An experiment with the edited nearest-neighbor rule", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 6 No. 6, pp. 448–452.

Vázquez F., Sánchez S. and Pla F. (2005) *A stochastic approach to Wilson's editing algorithm*. In: Marques JS et al (eds) IbPRIA 2005, LNCS 3523. Estoril, Portugal, pp 35–42.

Weiss, S. and Indurkhya, N. (1991). "Reduced complexity rule induction", in Mylopouslos, J. and Reiter, R. (eds.), *Proceedings of 12th International joint conference on Artificial Intelligence*, Morgan Kauffmann, Sydney, Australia, pp. 678–684.

Wilson, D. L. (1972). "Asymptotic properties of nearest neighbor rules Using Edited Data", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 2 No. 3, pp. 408–421.

Wilson, D. L. and Martinez, T. (1997). "Instance pruning technique", Fisher, D. M. (ed.), *Machine Learning: Proceedings of the Fourteenth International Conference (ICML'97)*, Morgan Kauffmann, San Francisco, CA, pp. 403–411.

Wilson, D. L. and Martinez, T. (2000). "Reduction techniques for instance based learning algorithms", *Machine Learning*, Vol. 38 No. 3, pp. 257–286.

Witten I.H. and Frank E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edition. Morgan Kaufmann, San Francisco.

Wojtusiak, J., Michalski, R. S., Kaufman, K. A., and Pietrzykowski, J. (2006). "*The AQ21 natural induction program for pattern discovery: Initial version and its novel features*", *In: Proceedings of the 18th IEEE International conference on tools with Artificial Intelligence. Washington DC*.

Xuepeng X. (2004). *Classification rule induction with an ant colony optimization algorithm,* Master's thesis, University of Texas Tech, USA.

Zhao, K., Zhou, S., Guan, J. and Zhou, A. (2003). "C-Pruner: An improved instance pruning algorithm", *Proceedings of the 2th International Conference on Machine Learning and Cybernetics*, Vol. 1, Sheraton Hotel, Xi'an, China: Piscataway, pp. 94–99, NJ: IEEE, piscataway.