DISCOVERING REGRESSION AND CLASSIFICATION RULES WITH MONOTONIC CONSTRAINTS USING ANT COLONY OPTIMIZATION

A THESIS SUBMITTED TO

THE UNIVERSITY OF KENT

IN THE SUBJECT OF COMPUTER SCIENCE

FOR THE DEGREE

OF PHD.

By

James Brookhouse

December 2018

Copyright

Abstract

Data mining is a broad area that encompasses many different tasks from the supervised classification and regression tasks to unsupervised association rule mining and clustering. A first research thread in this thesis is the introduction of new Ant Colony Optimization (ACO)-based algorithms that tackle the regression task in data mining, exploring three different learning strategies: Iterative Rule Learning, Pittsburgh and Michigan strategies.

The Iterative Rule Learning strategy constructs one rule at a time, where the best rule created by the ant colony is added to the rule list at each iteration, until a complete rule list is created. In the Michigan strategy, each ant constructs a single rule and from this collection of rules a niching algorithm combines the rules to create the final rule list. Finally, in the Pittsburgh strategy each ant constructs an entire rule list at each iteration, with the best list constructed by an ant in any iteration representing the final model. The most successful Pittsburgh-based Ant-Miner-Reg_{PB} algorithm, among the three variants, has been shown to be competitive against a well-known regression rule induction algorithm from the literature.

The second research thread pursued involved incorporating existing domain knowledge to guide the construction of models as it is rare to find new domains that nothing is known about. One type of domain knowledge that occurs frequently in real world data-sets is monotonic constraints which capture increasing or decreasing trends within the data. In this thesis, monotonic constraints have

been introduced into ACO-based rule induction algorithms for both classification and regression tasks. The enforcement of monotonic constraints has been implemented as a two step process. The first is a soft constraint preference in the model construction phase. This is followed by a hard constraint post-processing pruning suite to ensure the production of monotonic models. The new algorithms presented here have been shown to maintain and improve their predictive power when compared to non-monotonic rule induction algorithms.

Acknowledgements

So many people have helped and encouraged me during my PhD and the writing of this thesis and I would like to thank them all. I would like to individually thank the following people. First, my fiancée Ayah for her support and encouragement over the last few years since she showed up in the office and started to come to lunch with us.

I would also like to thank my supervisors Fernando Otero and Alex Freitas who have guided me through the last four years of research that I have undertaken during my PhD and the writing of this thesis.

I must also thank my good friends at the University of Kent Medway campus: Sam, Anna, Janine, Shannon and Mick (the friendly retired police officer) who have listened to my rants, provided support, nuggets of wisdom and of course constantly asking how the writing up is going. Finally, my parents who have always supported me and pushed me to stretch myself and tolerated the lack of a“proper” job.

Research is what I'm doing when I don't know what I'm doing.

WERNHER VON BRAUN

Contents

List of Tables

List of Figures

List of Algorithms

Nomenclature

β — Coefficients for linear models

A — An attribute from the data-set

AE — Absolute Error

$Coverage$ — Number of instances satisfied by a model

MAE — Mean Absolute Error

n — Number of instances in a data-set

NMI — Non Monotonicity Index

p — Predicted value of a target variable

Q — Quality

$RMSE$ — Root Mean Square Error

$RRMSE$ — Relative Root Mean Square Error

S — The entropy gain of an attribute

T — The set of training instances

t — True value of a target value

x — Predictor attributes for the regression task

y — Target attribute for the regression task

Chapter 1

Introduction

As computational resources have improved and increased, the amount of data collected and stored about the world we live in has greatly increased. As Frawley, Piatetsky-Shapiro and Matheus (1992) state "*Computers have promised us a fountain of wisdom but delivered a flood of data*". Since this vast quantity of new data cannot be manually processed, we have to turn towards automated methods to condense and abstract knowledge from this raw data.

Data mining is the task of finding and extracting interesting relationships and connections present in data using (semi-)automated techniques (Fayyad, Piatetsky-Shapiro and Smith 1996). It comprises of many tasks, including the unsupervised tasks of clustering and association rule mining, and the supervised tasks of classification and regression. This thesis will concentrate on the latter two tasks. Both classification and regression tasks aim to build a model capable of making predictions from a set of predictor attributes. The two tasks are differentiated by the form their predictions take. In classification, the prediction is chosen from a predetermined set of class labels. Classification problems include e.g. spam email detection (Yu and Xu 2008) and the detection of breast cancer (Mangasarian, Street and Wolberg 1995). In regression, the prediction is continuous and can

take the value of any real number. Example regression problems include predicting the size of a forest fire based on meteorological data (Cortez and Morais 2007) or the prediction of future stock prices (Liu and Yeh 2017).

There are a number of different approaches to create classification and regression models —these include traditional greedy algorithms. In their attempt to find the global optimum, greedy algorithms make the best choice at every decision point. While this decision may be locally optimal, it may not lead to the global optimum (Cormen et al. 2009). Meta-heuristic algorithms (Bonabeau et al. 1999) take a different approach, as they optimise many decisions at the same time, allowing the decisions to interact with each other as they pursue their goal of finding the global optimum.

When constructing classification and regression models, the structure of the model has to be decided upon. There are two broad categories of models: *white-box* and *black-box* models. Black-box models have inner workings that are to complex to be directly interpreted by users i.e., the decision making process is opaque. This is in contrast to white-box models, where the decision process is transparent and interpretable. Users can therefore interrogate the model to discover how and why a particular output was generated given the inputs into the model. Well known black box models include SVMs (Boser, Guyon and Vapnik 1992) and Neural Networks (Schmidhuber 2015).

This thesis will focus on one particular type of white box model, which are classification and regression rules. Rules have a very simple structure built from two parts: a consequent, which defines the prediction made by a rule; and an antecedent, which encapsulates the reasons why a rule makes a prediction through a set of tests on attribute values. As each rule only makes a prediction when its antecedent is satisfied, a number of rules can be grouped together to create a list of rules, resulting in a complete model capable of making different predictions based on the inputs provided. Rule lists are in general comprehensible and interpretable

models as a user can intuitively inspect the reasons why a rule list made a specific prediction (Witten et al. 2016).

Ant Colony Optimization (ACO) (Dorigo 1992) is a nature inspired approach from the wider field of swarm intelligence. The ACO meta-heuristic mimics the behaviour of how ants (agents with a simple behaviour) work together to solve complex tasks, which can be applied to solve optimisation problems. In the field of data mining, and more specifically the classification task, ACO-based algorithms have been developed to create a number of different classification models, including lists of classification rules (Martens et al. 2007), and decision trees (Otero, Freitas and Johnson 2012). Ant-Miner (Parpinelli, Lopes and Freitas 2002) is the best known family of ACO-based algorithms. Classification has received the majority of investigation and, to the best of my knowledge, no ACO-based algorithms have been developed to tackle the regression problem. The success of Ant-Miner and its derivatives (Parpinelli, Lopes and Freitas 2002; Martens et al. 2006b; Otero, Freitas and Johnson 2008, 2009, 2013; Otero and Freitas 2013; Otero 2017) when applied to the classification suggests that the ACO paradigm can also be successful when applied to the regression task. The first major contribution of this thesis is the proposal of new ACO algorithms for the regression task, specifically the construction of lists of regression rules.

It is rare in data mining to find a problem where nothing is known before automated techniques are applied. This existing knowledge may be the product of previous data mining techniques or knowledge provided by domain experts (Frawley, Piatetsky-Shapiro and Matheus 1992; Kubat, Holte and Matwin 1998). If existing knowledge is ignored when finding new relationships, the risk is the production of models that violate known domain relationships, resulting in models that are confusing and nonsensical, which may ultimately lead to model rejection by the user (Witten et al. 2016). The second major contribution of this thesis is the incorporation of existing domain knowledge when algorithms (focusing on

ACO-based algorithms for the regression and ordinal classification tasks) create models to extract new knowledge from the data.

One type of domain knowledge is monotonic constraints, which can be either monotonically increasing or decreasing. In this thesis, monotonic constraints have been implemented in two different ways. The first is soft monotonic constraints, where a preference is shown for a model to be monotonic but, as this is only a preference, they can be ignored if a model has a sufficiently high quality. The second method implements hard monotonic constraints, where the enforcement of monotonic relationships is rigid and no violations are tolerated in the models. These two approaches have been combined into Ant-Miner-derived algorithms (including the regression rule induction algorithms from the first contribution) for both the classification and regression tasks. The soft constraints are enforced initially, this ensures that the search is not overly constrained, allowing a greater exploration of the search space. The hard constraints are enforced later, once the search space has been explored, to ensure the models produced are completely monotonic. ACO algorithms are a good candidate for implementing soft constraints, as monotonicity is a global property of a model (a list of rules) and not the property of a single rule, and the ability of ACO algorithms to allow interactions between the different rules in a model allows the successful optimisation of monotonic constraints.

1.1 Thesis Structure

The rest of this thesis will be structured as follows. First, background on the relevant literature will be introduced in three chapters. In Chapter 2, the regression and classification tasks in data mining will be introduced along with a number of model representations, including rules and learning strategies to create lists of rules. Chapter 3 will introduce the concept of prior domain knowledge and how

this can be encapsulated as monotonic constraints. The final background chapter will introduce Ant Colony Optimization (ACO) before concentrating on existing ACO-based algorithms that tackle classification problems.

Chapter 5 contains the first original contribution of this thesis, proposing new ACO-based algorithms for the regression task, using different learning strategies to produce lists of regression rules. The computational results of these algorithms are presented in Chapter 6.

Next, monotonic rule lists are explored with the proposal of ACO-based algorithms for both the regression and classification tasks that create monotonic lists of rules, along with post-processing procedures that will rigidly enforce the given constraints. This post-processing procedure can be applied to any existing data mining algorithm that produces a list of rules. Chapter 8 contains the experimental analysis of the monotonic ACO algorithms introduced in Chapter 7.

Chapter 10 summarises the contributions presented earlier before drawing conclusions and suggestions for future directions that can be explored to build on the work presented.

The last contribution from this thesis is in Chapter 9, which proposes a new ACO algorithm that replaces the traditional combinatorial ACO construction methodology with an ACO model construction technique aimed at optimising both combinatorial and continuous problems. This allows the optimisation of rule conditions involving continuous attributes by the pheromone model and dispenses with the extra discretisation procedures required by combinatorial approaches.

1.2 Publications

A number of chapters in this thesis are based on my previous publications. These publications are listed below along with a brief introduction and the chapters they influenced.

- Brookhouse and Otero (2015), Discovering Regression Rules with Ant Colony Optimization, In: Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO'15 Companion) pp. 1005-1012 — This paper introduced the original Ant-Miner-Reg algorithm with two discretisation strategies, comparing it to the original greedy Iterative Rule Learning based algorithm SeCoReg. This paper is incorporated into Chapters 5 and 6.

- Brookhouse and Otero (2016), Monotonicity in Ant Colony Classification Algorithms, In: Proceedings of the 10th International Conference on Swarm Intelligence (ANTS 2016) pp. 137-148 — Based on the implementation of monotonic constraints to regression problems, the focus was expanded to include the classification task and incorporating constraints to the Pittsburgh approach-based algorithm *c*Ant-Miner$_{\mathrm{PB}}$. This paper was incorporated into Chapters 7 and 8.

- Brookhouse and Otero (2016), Using an Ant Colony Optimization Algorithm for Monotonic Regression Rule Discovery, In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2016) pp. 437-444 — Ant-Miner-Reg was given the capability to discover monotonic rule lists with the addition of soft constraints during the learning phase and a naive hard constraint pruner. The paper was incorporated into Chapters 7 and 8.

- Brookhouse and Otero (2018), Post-Processing Methods to Enforce Monotonic Constraints in Ant Colony Classification Algorithms, In: Proceedings of the 2018 International Joint Conference on Neural Networks pp. 1-8 — The original *c*Ant-Miner$_{\mathrm{PB+MC}}$ algorithm (Brookhouse and Otero 2016a) used a simple backtrack pruner to enforce monotonic constraints. A pruning suite was developed with three hard pruners to improve predictive accuracy,

reducing the negative impact on accuracy caused by hard constraint pruning. Chapters 7 and 8 incorporate the algorithms and results published in this paper.

- Helal, Brookhouse and Otero (2018), Archive-Based Pheromone Model for Discovering Regression Rules with Ant Colony Optimization, In: Proceedings of the 2018 IEEE Congress on Evolutionary Computation pp. 1-7—Previously, we concentrated on traditional graph-based ACO algorithms, which work well on combinatorial optimisation problems. However, extra discretisation steps are required for continuous attributes. Ant-Miner-Reg_{MA} implements an archive pheromone model that attempts to address continuous attributes and optimise rule conditions involving this type of attribute within the pheromone model. The initial results are presented in Appendix 9.

Chapter 2

Data Mining

Data mining is a research area focused on automating the search for useful patterns in data (Fayyad, Piatetsky-Shapiro and Smith 1996). There are a number of different tasks including: classification, regression, association rule mining, and clustering. These task can be divided into two learning types, supervised learning (e.g. classification and regression) and unsupervised learning (e.g. clustering and association rule mining).

In a data-set, we often refer to instances and attributes. The instances represent entities (e.g. cars, patients, customers, etc), whilst attributes (also called features) describe properties of the entities, e.g., a data-set of cars may have attributes such as engine size, number of doors, or manufacturer.

In unsupervised learning the problems that are tackled involve unlabelled data-sets and the algorithm is required to learn patterns with no explicit feedback on the quality of the patterns learnt. One example of unsupervised learning is clustering, where the aim is to create groups of similar of instances within the total data-set based on a cluster cohesion metric such as the distance from the cluster centre (Russell and Norvig 2016; Witten et al. 2016).

In the supervised learning task, the data-set is labelled, i.e., it has a known target attribute. This is the attribute that we are trying to predict based on

the other (predictor) attributes present in the data-set. It is also common to partition data-sets into training and testing data. The training data is a subset of the complete data-set, where the value of the target attribute is known. This set is used to learn relationships present in the data. Once a set of patterns has been learnt, it can be tested on the second set. On the testing set we predict the value of the target attribute and compare our predictions to the true value, which is useful to assess the accuracy of our patterns.

Since in supervised learning the target attribute is known for a number of training instances, the aim is to create a model from our labelled training data, where the algorithm can use knowledge of its prediction and the true value of the target to measure the quality of solutions it creates. The models produced can then be used to make a prediction of the target attribute on unlabelled test data (Russell and Norvig 2016). In this thesis, I will be concentrating on supervised learning for both the regression and classification tasks.

In the rest of this chapter I will outline and describe the two main tasks that will be tackled in my thesis. First, I will discuss the regression task, where the aim is to predict a continuous value and the different model representations that are commonly used. Then I will move to the classification task, where the aim is to predict a categorical (nominal) value from a predetermined set of class labels. Finally, I will discuss two different types of model representations and corresponding algorithms that use these representations, namely rule and decision tree induction algorithms.

2.1 Regression Task

The regression task involves the construction of models that produce real value predictions for each instance they are given. The prediction of a desired target value is made based on the values of predictor attributes (Fahrmeir et al. 2013).

Table 2.1: Simple house rental data set

Target Attribute	Predictor Attributes		
Rental Value	Floor Area	Location	Garage
£300	45	3	No
£600	80	1	Yes
£250	33	3	No
£400	65	2	Yes
£350	54	2	Yes

The predictor attributes can be a combination of real, ordinal, and nominal values. Real-valued attributes are continuous in nature and can take any numeric value, e.g., the length of an object. Nominal attributes have a set of allowed values, an example of a binary nominal attribute could be gender which can take the value of male or female. Finally, we have ordinal attributes, which have qualities of both real and nominal attributes. They still have a set of allowed categorical values like nominal attributes. However, this set has a natural order, for example the set `(Small, Medium, Large)` is an ordinal set of values.

Table 2.1 shows a simple data-set for house rentals. This data-set consists of a continuous target (response) attribute, which in this case is the rental value of the property, and three predictor (regressor) attributes. The first predictor attribute is the floor area, a continuous (real) attribute that can take any numerical value. The second attribute takes an ordinal value referring to the quality of the location, where 1 is a good location, 2 average and 3 is poor. In this case, location quality values have a natural order, which is the differential property between ordinal and nominal attributes. For this reason, regression algorithms may treat the two attribute types differently. Finally, the third attribute takes a nominal value representing the presence/absence of a garage.

Considering the data set in Table 2.1, a regression model may predict that

a house with a floor area less than 50 in location 3 will have a rental value of £275. Models have to be evaluated to assess how good they are, that is to say the model's quality. The better a model is, the closer it will get to predicting the actual values found in the data set —this is referred to as the model's quality. In regression, the value predicted is rarely going to be exact, therefore a notion of error is required. Common measures of prediction error in regression problems are the mean absolute error and the root mean square error (Fahrmeir et al. 2013).

The mean absolute error is given by:

$$MAE = \frac{\sum_{i=1}^{n} |p_i - t_i|}{n} \tag{1}$$

where n is the number of instances and p_i and t_i are the predicted and true values of the target variable respectively. The root mean square error is given by:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (p_i - t_i)^2}{n}} \tag{2}$$

While the MAE is more intuitive due to being a simpler metric, the RMSE has the advantage of penalising large errors. That is, RMSE uses the average of the square of an error, so increasingly large errors in a prediction are heavily penalised while a number of small errors in a prediction can receive a smaller penalty. On the other hand, this makes the RMSE metric more sensitive to outliers or noisy data than the MAE metric.

Typical regression problems include predicting house prices based on the size of the living area, house location, number of bedrooms and bathrooms; predicting the height of a child based upon the heights of their parents (Fahrmeir et al. 2013); boat aerodynamic performance based on hull shape and design; or CPU performance based on architecture, clock speed and manufacturing techniques.

A number of different algorithms have been developed to construct regression models. Table 2.2 lists a number of algorithms that will be discussed in this thesis

along with the model representation used by the algorithm. It should be noted that Table 2.2 is not an exhaustive list of algorithms developed for the regression task, but rather a flavour of the techniques available. For a discussion of other regression algorithms please refer to Fahrmeir et al. (2013).

From the perspective of interpretability and comprehensibility, models are often categorised as either black-box or white-box models (Bibal and Frénay 2016). A black-box model is one where the inputs and outputs of the model are known however the internal workings of the model — that is to say how the model reaches its prediction — are hidden from the user. White-box models, on the other hand, allow users to see the inputs and outputs like black-box models but also the decision process that has led to the output observed. Common white-box models include regression rules and regression trees. However, even within the broad group of white-box models, different models have varying degrees of comprehensibility –that is how easy a model is to understand. One advantage of regression rules is that it is simple to understand the decisions behind the model's predictions. They also only predict values from within the range of any training data and will therefore remain well behaved — i.e., any value predicted must be within the limits present in the training data set. Whereas linear and non-linear models from classical multivariate statistics produce numeric equations, which may be difficult to understand and can also predict values extrapolated beyond the known training data (Montgomery, Peck and Vining 2012). Therefore they are not guaranteed to be well behaved — i.e., values predicted outside the range observed in the training data may quickly diverge from the true values, as no notion of correctness was known for these areas when the model was constructed. In addition, as the non-linearity of a model increases the risk of being less well behaved outside the observed range of values.

Table 2.2: A selection of existing regression algorithms classified by the type of model representation that they use. A comprehensive review of regression models can be found in Fahrmeir et al. (2013)

Model Representation	Algorithm	Reference
Linear Models	Simple Linear Regression	Fahrmeir et al. (2013)
Non-Linear Models	Support Vector Regression	Wu et al. (2008)
	Genetic Programming	Augusto and Barbosa (2000); Uy et al. (2011)
	Neural Networks	Specht (1991)
Regression and Model Trees	CART	Breiman et al. (1984)
	M5	Quinlan (1992)
Regression Rules	SeCoReg	Janssen and Fürnkranz (2010a)
	M5' Rules	Holmes, Hall and Frank (1999)
	PSOMiner	Minnaert and Martens (2012)
	GIBRG	Liu and Cocea (2018)

It should be noted that it is possible to extract white box models like rules from black box models like neural networks and support vector machines, as shown in (Bologna and Hayashi 2018; Martens, Baesens and Van Gestel 2009). However, this rule extraction process requires the a priori learning of a black box model followed by the extraction procedure, which is usually a very time consuming process. This approach is out of the scope of this work, instead this thesis focuses on learning a white box model directly from the data.

2.1.1 Linear Regression

Linear regression attempts to find a function so that $f(\vec{x_i}) \rightarrow y_i$ for each i-th instance, where $\vec{x_i}$ are the predictor attribute values and y_i is the target attribute value for the i-th instance. A linear regression model is illustrated in Equation 3:

$$y_i = \beta_0 + \beta_1 x_{i1} + ... + \beta_j x_{ij} + \varepsilon \tag{3}$$

where β_0 is the intercept, β_j are the linear coefficients for the predictor attributes, j is the attribute number, $x_{i1} \ldots\ x_{ij}$ are the predictor attribute values for the i-th instance, y_i is the target attribute; and finally, ε is the random error associated with each measurement with an expected value of 0 and normal variance. While this model looks restrictive, the predictor attributes themselves can be functions chosen by the researcher to better fit the problem, e.g., $x' = \frac{1}{x}$. This enables a linear equation to fit non-linear data (Fahrmeir et al. 2013).

Simple linear regression computes the values of the intercept and the $\beta_1 \ldots$ β_j coefficients by using a least squares regression method, where the aim is to minimise the squared errors between the true target attribute values and the values predicted by the linear model (Fahrmeir et al. 2013). Another linear regression approximation algorithm is the Least Absolute Deviations (LAD). LAD attempts

to find the linear model that minimises the absolute error residuals, calculated as:

$$AE = \sum_{i=1}^{n} |y_i - f(\vec{x_i})| \tag{4}$$

where $\vec{x_i}$ is the vector of input attribute values for the i-th instance, $f(\vec{x_i})$ is the proposed linear model and y_i is the actual target value for the i-th instance (Bloomfield and Steiger 1980).

2.1.2 Non-Linear Regression

The Support Vector Regression (SVR) algorithm is an extension of the original Support Vector Machine (SVM) algorithm for classification (Wu et al. 2008; Vapnik 2013). SVMs model the training data-set in a high dimensional space. A kernel function is used to map the data to a typically much higher dimensional space, where finding a hyper-plane[1] that separates the classes is typically easier than in the original feature space. Then, the SVM algorithm finds the maximum margin hyper-plane separating instances of different classes, i.e., a hyper-plane that bisects the original space such that instances of the positive and negative class are located on the opposite sides of that hyper-plane, and the plane has the greatest margin between the classes (Wu et al. 2008; Vapnik 2013).

To extend this into regression problems the hyper-plane generated is required to approximate the function represented in the training data. This is accomplished by choosing an error measure that approaches zero as the difference between the predicted value and actual value also approaches some ε specified by the user. The problem then is simply to minimise the Lagrangian that contains information on the residuals (Wu et al. 2008).

Genetic programming (GP) has also been used to generate symbolic regression models (Augusto and Barbosa 2000; Uy et al. 2011). GP is an evolutionary

[1] A hyper-plane is a plane with one less dimension than the space it occupies.

technique that modifies a current solution population based on procedures inspired from natural evolution (Koza 1994). In nature, evolution is driven by natural selection and survival of the fittest, that is to say the best (fittest) individuals in a population are the ones most likely to successfully reproduce, allowing their genetic improvements to influence future generations. GP attempts to mimic this process in the creation and evolution of solutions to computationally hard problems. Hence, solutions are represented by individuals of the GP population and the fitness function is given by the objective function to be optimised.

For symbolic regression, the models are internally represented as trees before being flattened to create non-linear models. The two main procedures that allow GP to create new solutions from current solutions are crossover and mutation. Crossover requires the selection of two "parent" solutions from the population, where the selection is based on fitness. Random positions in the solutions trees are selected as crossover points, at which point the two sub-trees are swapped creating two new solutions for the next generation. Mutation is the second commonly used evolutionary procedure, where a current solution (again, selected based on fitness) has a randomly selected sub-tree replaced by a new randomly generated sub-tree (Augusto and Barbosa 2000; Uy et al. 2011). When some pre-specified stopping criteria are met, the best (fittest) population member is chosen as the solution to be returned.

Neural networks can also be used to create non-linear regression models. A network is composed of a large number of simple processors (neurons) that are interconnected with each other. Each neuron contains a function that modifies its inputs to produce an output. These functions can be linear or non-linear, allowing the construction of both types of models. Each neuron connection has an associated weight to represent the strength of the connection. The weights are adapted through the learning phase as the network attempts to approximate the non-linear function that will map the predictor attributes to the target attribute

(Specht 1991).

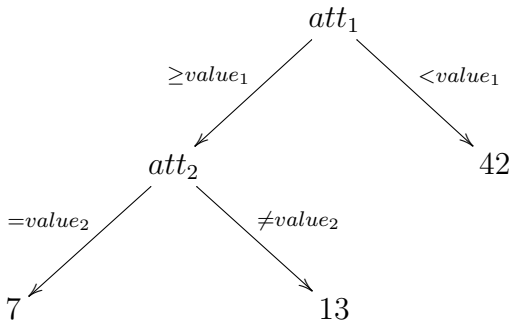
2.1.3 Regression and Model Trees

Regression and Model Trees are tree-like structures that consist of connected nodes (Barros et al. 2012). There are two main types of nodes. Internal nodes are nodes that represent a decision point in a tree with a single input and at least two outputs, and contain a logical test using the predictor attributes in a data-set. The second node type are leaf nodes, these nodes are terminal nodes in a tree as they have a single input and no outputs. These leaf nodes are also the nodes that contain the predictions a tree can make.

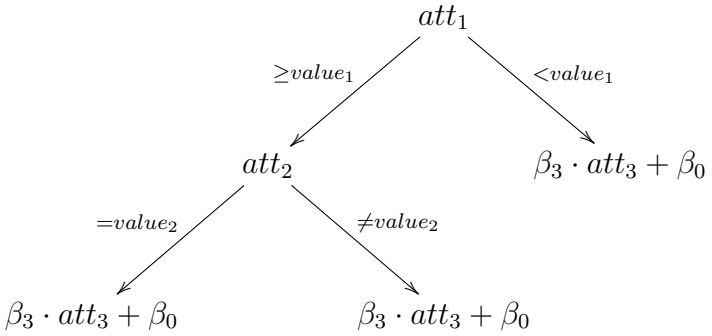
Regression trees start at a root node, where a logical test is located. The test is based on one of the predictor attributes. For each test outcome, there is a branch that can be followed. The branches will lead to another internal node, where another logical test using another predictor attribute is performed or to a leaf node, representing a prediction. In order to classify an instance, that instance is processed in a top down fashion starting with the root node until a leaf node is encountered and a prediction made. The path followed by the instance from the root node to a leaf node is determined by the outcome of the tests at each of the internal nodes.

There are two types of trees for the regression task. Figure 2.1 contains examples of both types: Figure 2.1a shows a regression tree with internal nodes representing different logical tests and leaf nodes representing single real-valued predictions; the second tree variant is a model tree, shown in Figure 2.1b. Model trees have a different leaf structure, where each leaf contains a linear model to make different predictions depending on the values of the predictor attributes for each data instance.

Regression trees produce comprehensible models (as long as a tree is not too large), as trees provide an easy way to understand the decisions that were made to



(a) Regression tree, with continuous predicted values on the leaf nodes



(b) Regression model tree with linear models on the leaf nodes.

Figure 2.1: Regression and Model Trees. Both trees use the same decision nodes and branches, but they have different types of leaf nodes.

arrive at a prediction. This comprehensibility is diluted slightly when introducing linear models into the leaf nodes of a tree, however they still maintain some comprehensibility as the models are restricted to simple linear approximations. The benefit of having linear models is the expansion of the variability of the predictions a tree can make, as the linear models allow the tree to make small adjustments to the final prediction.

$$IF\ att_1 \geq value_1\ AND\ att_2 = value_2\ THEN\ 3.5$$

Figure 2.2: A Regression rule, containing logical tests in the antecedent and a real valued prediction.

2.1.4 Regression Rules

A regression model can also be represented as a list of rules. Each regression rule is composed by an antecedent, which corresponds to a list of logical statements to be satisfied, and a consequent representing the value predicted. To classify an instance, we can simply start from the first rule in a list and check if the antecedent is satisfied; if so, return the prediction, otherwise move to the next rule in the list. Figure 2.2 shows one possible format for regression rules, where the antecedent takes the form of a sequence of (attribute, operator, value) tuples connected by a logical AND statements.

In Figure 2.2 att_1 and att_2 are two predictor attributes. The value after the $THEN$ corresponds to the rule's prediction (consequent of the rule). Regression rules can create comprehensible models as users can quickly understand why a prediction is made due to their simple logical nature, as long as the rules do not have too many conditions and the list of rules is not too long.

2.2 Classification Task

The classification task is similar to the regression task, in the sense it uses a set of predictor attributes. However, now the goal is to correctly classify instances from a data-set into a list of classes that have been defined before hand.

Table 2.3 shows the data-set that was introduced in the previous section about regression (Table 2.1), modified to a classification problem. While in Table 2.1 the rental value (target attribute) was a continuous attribute containing the price in pounds, in Table 2.3 these values have been grouped and discretised into three classes "Low" (Rental Value < 300), "Medium" ($300 \leq$ Rental Value < 400) and

Table 2.3: Modified house rental data-set for the classification task. The rental price attribute has been discretised into low, medium and high.

Target Attribute	Predictor Attributes		
Rental Value	Floor Area	Location	Garage
Medium	45	2	No
High	80	1	Yes
Low	33	3	No
High	65	2	Yes
Medium	54	2	Yes

"High" ($400 \leq$ Rental Value). The difference between the two tasks focuses on the target attribute, in the classification task this attribute has a pre-defined set of allowed values, while in the regression task it can be any continuous value.

We will concentrate on two specific types of classification models in this section: classification trees and classification rules. These models share many features, as is often the case that tree-based models are converted into rule-based ones (Quinlan 1993). However, this is usually a one way operation. In rules the antecedent of each rule is specific to that rule, as while each logical test has two outcomes a single rule is only concerned with one outcome and we cannot guarantee another rule will cover the disregarded outcome. Rules also only predict based on the positive satisfaction of the antecedent, while each internal node of a tree is associated with two paths for the positive and negative outcomes of the test. When converting trees to rules you can take the path from the root node to each leaf node and turn the path into a rule.

2.2.1 Classification Trees

Classification (or decision) trees are tree-like structures, where each decision (internal) node represents a single logical condition on a particular attribute, each

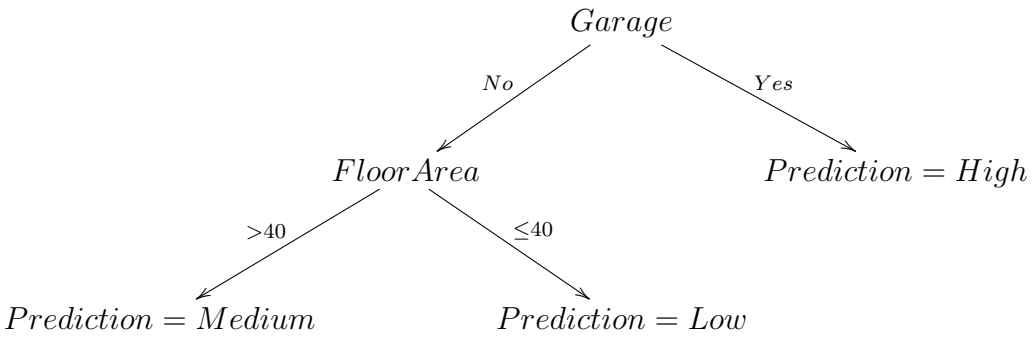


Figure 2.3: A simple classification tree that can be used to make predictions on the house rental data set shown in Table 2.3

branch represents a possible outcome of the test, and each leaf node makes a single prediction from the set of allowed classes. Note that each instance will be classified by exactly one leaf node (i.e., each instance follows exactly one path from the root to a lead node). Figure 2.3 shows a decision tree that could be used to make predictions on the data set shown in Table 2.3.

Table 2.4 shows how the tree in Figure 2.3 would classify the data-set. In this case, the tree correctly made four correct predictions, however the right hand branch of the first decision incorrectly predicted the fifth instance as "`High`" when in fact the true value was "`Medium`", giving this tree an accuracy of 80% (four correct predictions from a total of five).

2.2.2 Classification Rules

Classification rules are similar to regression rules, following the same structure. The antecedent consists of logical tests, which, if satisfied, will lead the rule to make the prediction specified by the rule consequent. One possible rule representation can be seen in Figure 2.4, which is similar to the one for regression rules shown in Figure 2.2.

If the rule shown in Figure 2.4 was applied to the sample classification data-set

Table 2.4: Prediction made by the classification tree found in Figure 2.3 on the data-set from Table 2.3. The first column contains the predicted value while the second contains the true value.

Target Attribute		Predictor Attributes		
Tree Prediction	Rental Value	Floor Area	Location	Garage
Medium	Medium	45	2	No
High	High	80	1	Yes
Low	Low	33	3	No
High	High	65	2	Yes
High	Medium	54	2	Yes

$$IF\ FloorArea \geq 40\ AND\ Location = 2\ THEN\ Medium$$

Figure 2.4: Example of a classification rule, where the predicted value is sampled from a set of allowed values.

in Table 2.3, we find that the antecedent — the logical conditions — of the rule is satisfied by instances 1, 4 and 5. It would then make the prediction that the correct rental value would be "`Medium`". When compared to the actual value found in the data-set, we find that in two cases the rule was correct and in one case it misclassified the instance, resulting in an accuracy of 66% on this data-set.

Multiple rules can then be combined together to create a rule list, allowing different predictions to be made depending on which rule of the list has its antecedent satisfied first by the input instance.

2.3 Rule Learning Paradigms

In this section we will discuss three rule learning paradigms: Iterative Rule Learning (IRL), Michigan, and Pittsburgh. Both IRL (Mitchell 1997) and Michigan

Algorithm 2.1: Iterative Rule Learning high-level pseudo-code

```
   Data: Instances
   Result: RuleList
 1 RuleList ⟵ ∅
 2 while Stoping Criterion < Threshold do
 3 |   Rule ⟵ LearnOneRule(Instances)
 4 |   LocalSearch(Rule, Instances) // Optional
 5 |   // Adds rule to list
 6 |   RuleList ⟵ RuleList ∪ Rule
 7 |   // Removes covered instances
 8 |   Instances ⟵ Instances − Covered(Rule)
 9 end
10 // Adds the default rule
11 RuleList ⟵ List ∪ Rule_Default
12 return RuleList
```

(Booker, Goldberg and Holland 1989) rule learning paradigms concentrate on optimising the construction of individual rules, reducing the task of creating a list of rules into smaller problems of creating a single rule. The Pittsburgh paradigm, on the other hand, considers the task of creating a complete list of rules as a single problem, allowing the optimisation of rule interactions at the cost of a more complex optimisation problem (Smith 1983).

2.3.1 Iterative Rule Learning

Iterative Rule Learning (IRL)—also known as sequential covering—is a rule learning paradigm that allows the construction of a list of rules. It creates a single rule at a time, where each rule covers a subset of the training instances in a data-set. Algorithm 2.1 presents the generic pseudo-code for an IRL algorithm.

The algorithm starts with an empty list of rules (line 1). Line 3 of Algorithm 2.1 contains the function `LearnOneRule(Instances)`, which is responsible for creating a single rule on the current set of instances. There are different strategies for creating a single rule and this is usually the main point where IRL algorithms

differentiate. IRL algorithms can also differ in the evaluation function used to guide the rule learning process (Minnaert et al. 2015). Regardless of the strategy or rule evaluation function employed, the goal of this procedure is to create a good (best) rule given the current set of training instances.

Once a rule is created, an optional local search step is usually implemented to further refine the created rule. This involves making small changes to a rule's antecedent as long as the resulting antecedent improves the rule's quality. The rule is then added to the partial list of rules constructed so far and any instances covered by the new rule are removed from the current set of instances (line 8). The removal of covered instances encourages the algorithm to create rules that cover different instances within the training data.

This process is repeated until the stopping criterion is met (line 3). This criterion can take a number of different forms. It may be based on the quality of the last rule constructed dropping below a set threshold. Another common stopping criterion is the number of uncovered instances remaining in the training set dropping below a set value.

Once the loop (lines 2-9) has ended, the default rule can be added to the rule list. The default rule is normally a rule with an empty antecedent, such that it will always make a prediction for every instance that is not covered by any of the other rules in the list. This ensures that the rule list will always make a prediction, as the default rule will always evaluate true for any instance. The default rule's prediction is set differently for classification and regression tasks. In the former it is set as the most frequent class among the instances not covered by any other rule, while in the regression task it is often set as the mean value of these uncovered instances.

Algorithm 2.2: Michigan-based rule induction high-level pseudo-code

```
   Data: Instances
   Result: RuleList
 1 RuleList ⟵ ∅
 2 while Size(RuleList) < Max Rules do
 3 |   Rule ⟵ LearnOneRule(Instances)
 4 |   LocalSearch(Rule, Instances) // Optional
 5 |   RuleList ⟵ RuleList ∪ Rule
 6 end
 7 // Perform Niching Operation to select rules for final list
 8 RuleList ⟵ Niche(RuleList, Instances)
 9 // Adds the default rule
10 RuleList ⟵ List ∪ Rule_Default
11 return RuleList
```

2.3.2 Michigan Rule Learning

In the Michigan paradigm, single rules are constructed in the same manner that was previously discussed for IRL (Section 2.3.1). However, instead of removing covered instances, multiple rules are created to compete for a place in the final list of rules (Booker, Goldberg and Holland 1989; Booker 1982).

Algorithm 2.2 presents the pseudo-code for a Michigan-based rule learning algorithm. The algorithm starts with an empty set of rules, and in each iteration of the while loop (lines 2-6), it generates a single rule to add to this set. On line 3, the same `LearnOneRule(Instances)` procedure from the earlier IRL pseudo-code (Algorithm 2.1) is used to generate a single rule based on the current training instances. The newly created rules are then added to the current set of rules.

At this point in the algorithm, there is not a list of rules representing a single model, but in fact a collection of rules all learned from the same training set. As their is no order to the rules, this is called a rule set. Once all the rules have been generated based on a predetermined stopping criterion (line 2), a niching step takes place.

The goal of the niching procedure (line 8) is to select the best rules from the

set it is given to construct a list of rules. This is usually achieved by forcing the rules to compete for instances in the training set. First, the rule set is sorted by rule quality to create a list of rules. Then the best rule can claim any uncovered instances it covers, and at this point the instances covered by this rule are marked as covered. This is repeated for all rules in the rule list. Once all rules have claimed the instances that they can cover, rules that have covered at least one instances are added to the final rule list, respecting the order that they appear in the original (sorted by quality) list of rules. This new ordered list then has the default rule added (line 10) (Booker, Goldberg and Holland 1989; Booker 1982; Olmo, Romero and Ventura 2010).

2.3.3 Pittsburgh Rule Learning

The previous two rule induction paradigms concentrate on reducing the problem of creating a list of rules into smaller problems of creating a single rule, relying on a separate procedure to create a list. Pittsburgh-based rule induction algorithms differ as they aim to create an entire rule list in one construction step. Additionally, they concentrate on the quality of an entire rule list, often ignoring the quality of individual rules (Smith 1983, 1980). This allows Pittsburgh-based algorithms to optimise rule interactions during rule construction, which IRL and Michigan-based algorithms cannot (Mitchell 1997; Freitas 2002). Rule interaction occurs when the outcome of a rule affects the subsequent rules that can be created.

Algorithm 2.3 presents the pseudo-code for a generic Pittsburgh-based rule induction algorithm. The algorithm starts with an empty list of rules and generates an entire rule list in each iteration (lines 2-7). The `LearnOneRule (Instances)` function present in the previous Algorithms 2.1 and 2.2 has been replaced by a `LearnRuleList(Instances)` procedure (line 3 of Algorithm 2.3). Pittsburgh-based algorithms are not interested in the quality of a single rule, therefore, quality measures that evaluate an entire list are required. These new measures can then

Algorithm 2.3: High-level pseudo-code for a Pittsburgh-based rule induction algorithm.

```
   Data: Instances
   Result: RuleList
1  RuleList_best ⟵ ∅
2  while Iterations < Max Iterations do
3  |  RuleList ⟵ LearnRuleList(Instances)
4  |  LocalSearch(RuleList, Instances) // Optional
5  |  if Quality(RuleList ) > Quality(RuleList_best) then
6  |  |  RuleList_best ⟵ RuleList
7  |  end
8  end
9  return RuleList_bst
```

be used to compare list qualities to find the best list (line 5).

2.3.4 Regression Rule Induction Algorithms

Regression rules are induced by many different algorithms. In this section I will introduce a number of these algorithms. First, I will discuss the algorithm Separate-and-conquer Regression (SeCoReg) (Janssen and Fürnkranz 2010a,b). SeCoReg employs the commonly used sequential covering (IRL) strategy to construct a list of rules, its high-level pseudo-code has been presented earlier in Algorithm 2.1.

In SeCoReg, the `LearnOneRule(Instances)` procedure is implemented by a greedy search strategy to create regression rules. The strategy involves generating a list of possible modifications to the current rule, where each modification is a potential (attribute, operator, value) tuple. It then tentatively adds one modification at a time to the current rule, searching for the modification that gives the best quality of all the modifications. If the new rule generated by the addition of a modification is better than the best-so-far rule, the new rule replaces the current best rule. This process is repeated until there are no more modifications that can be added. The consequent of a rule is obtained by calculating the mean value of the target attributes among the training instances covered by that rule.

The rule quality is defined as the product of two measures (Janssen and Fürnkranz 2010a,b), as follows. The first one is the Relative Root Mean Square Error (RRMSE), given by:

$$L_{RRMSE} = \frac{L_{RMSE}}{\sqrt{\frac{1}{n}L_{default}}} \tag{5}$$

where n is the total number of instances in the training set, L_{RMSE} is the root mean square error and $L_{Default}$ is a normalising factor that will approximately bound the RRMSE between 0 and 1. Both are defined as:

$$L_{RMSE} = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (y_i - \bar{y_i})^2} \tag{6}$$

$$L_{default} = \sum_{i=1}^{n} (y_i - y')^2 \tag{7}$$

where y_i is the value of the target attribute of the current instance i, $\bar{y_i}$ is the predicted target value of the current instance i, and finally, y' is the mean target value over all instances. RRSME attempts to normalise the RMSE between 0 and 1, however it is still possible to achieve values above 1 when the predicted values are worse than predicting the mean value.

The second error measure is the relative coverage of a rule, which normalises the absolute coverage of the rule between 0 and 1—a rule with a value of 1 covers all the instances in the current training set. The relative coverage is given by:

$$relCov = \frac{1}{n} \cdot coverage(Rule) \tag{8}$$

where the coverage is the sum of all instances that satisfy a rule's antecedent. These two measures are combined to produce a single quality value Q for each

rule, given by:

$$Q = \alpha \cdot (1 - L_{RRMSE}) + (1 - \alpha) \cdot relCov \tag{9}$$

where α sets the weighting between RRSME and relative coverage. An α value of 1 will only take into account the RRSME and a value of 0 will consider just the coverage.

A number of different algorithms also use the sequential covering strategy, where each algorithm provides a different implementation for the `LearnOneRule (Instances)` procedure. In this section I will be discussing two more sequential covering algorithms that discover regression rules, namely M5'Rules (Holmes, Hall and Frank 1999) and PSOminer (Minnaert and Martens 2012).

M5'Rules is a wrapper for the tree-based algorithm M5 (Quinlan 1992). M5' Rules uses the sequential covering strategy to create a list of rules. It starts with an empty rule list, and in each iteration, an entire regression tree is generated. This tree is then flattened to produce a set of rules, where each path from the tree's root node to a leaf will become an individual rule. The best rule from the set is added to the list of rules. The covered instances are then removed from the training set and this process is repeated. As in M5, the prediction of the rules created by M5' Rules is made by a linear model, which compromise the comprehensibility of the rules that are produced, as discussed previously in Section 2.1.1.

PSOMiner is a Particle Swarm Optimisation (PSO)-based regression rule induction algorithm (Minnaert and Martens 2012). PSOMiner uses a sequential covering strategy to build a list of rules that covers the training instances, using a PSO procedure to find high quality rules. In each covering iteration, the PSO procedure generates a swarm of particles that move through the solution space, where each position is a potential rule. Each particle repeatedly tests solutions (candidate rules), where the direction followed by a particle in the solution space

is governed by the short term memory of the particle's best solution and the best solution generated by nearby particles. These effects cause particles to converge to high quality regions in the solution space.

PSO algorithms traditionally operate on numeric attributes, however, PSOMiner has the ability to deal with both numeric and categorical attributes by encoding the attribute's values to each particle in the following manner. All attributes are encoded over a number of dimensions with the allowed value range of [0,1]. Numeric attributes do not require split points like in M5' Rules and SeCoReg, instead they are mapped to two dimensions: the first dimension specifies the attribute's lower bound while the second encodes the upper bound of each attribute. To allow unbounded attributes, for both lower and upper bounds independently, the high and low values are reserved to signify the attribute is unbound. Categorical attributes are encoded via dummy encoding, where a dimension is used for each allowed value of the attribute in question. When determining which categorical value to use, the categorical value associated with the dimension that has a value closest to the maximum of 1. Null is set through an additional dimension, which enables an attribute to be unused. PSOMiner is limited to generating a list of rules which is constrained in size — in the experiments reported in (Minnaert and Martens 2012), the rule lists were arbitrarily limited to 5 or 10 rules.

2.3.5 Classification Rule Algorithms

There are a number of algorithms that represent their learned model as classification rules. In this section, I will discuss classical algorithms that are non-ACO-based. ACO-based algorithms will be described in detail in section 4.5.

A classical classification rule induction algorithm is Quinlan's C5.0 Rules (Quinlan 1993, 1996)[2]. This algorithm first constructs an un-pruned classification tree in the same manner as C5.0. The generated tree is then flattened into a

[2]C5.0 can be downloaded from https://www.rulequest.com/see5-info.html

rule list. A rule pruner is then applied to the rule list to generate the final model.

The delayed pruning (i.e., pruning the rules instead of the tree) allows for different outcomes than pruning the original tree and then flattening the pruned tree. When a tree is pruned, the decision to remove a node affects all subtrees (branches) below it. As every branch has a partially shared logical test with the other branches in the tree originating from the same node, any pruning has to be considered with respect to all other affected sub-trees, reducing the number of possible prunes and increasing the computational complexity of calculating the effect of a prune.

However, with rule lists, each rule has its own set of logical conditions (antecedent) and a modification of this set during pruning will not modify another rule's antecedent. This gives rule-based pruning strategies more options and increased flexibility. As a result a rule set often cannot be converted back to a classification tree, since branches may have been removed.

Another algorithm that generates a decision tree before extracting classification rules is PART (Frank and Witten 1998). PART uses an IRL strategy to produce a list of rules. First PART uses the C4.5 algorithm (Quinlan 1993) —a previous version of C5.0— to generate a decision tree. Then a rule is extracted from the tree based on the best leaf node and added to the partial list of rules. Any covered examples in the training set are removed and the procedure is repeated until a complete list of rules is generated.

Another classical rule induction algorithm is Cohen's RIPPER (Cohen 1995). RIPPER is based on an earlier algorithm IREP (Incremental Reduced Error Pruning) (Fürnkranz and Widmer 1994). Unlike the previously discussed C5.0, RIPPER and by extension IREP do not generate an intermediate model but directly create a list of rules.

RIPPER and IREP use the previously discussed IRL procedure to construct a set of rules, where in each iteration a rule is grown to cover a subset of the

uncovered instances in the training data. This rule is then pruned and added to the rule set. This process continues until the error on the last rule to be discovered is above the algorithm's stopping criterion, at which point the rule list is returned.

Rules are grown by repeatedly adding the (attribute,operator,value) tuple that maximises FOIL's information gain criterion (Quinlan 1990) until the rule covers a set of instances that belong to a single class. This rule is then immediately pruned, where the pruner maximises the rule quality function:

$$v(Rule, PrunePos, PruneNeg) = \frac{p + (N - n)}{P + N} \tag{10}$$

where *PrunePos* and *PruneNeg* are the sets of instances coverd by the pruned rule representing the positive and negative classes; P and N are the number of instances in *PrunePos* and *PruneNeg* respectively. The pruner continues until no improvement is found.

RIPPER contains a more complex pruning phase than IREP, allowing multiple conditions to be removed from rules and modified stopping criteria, support for numerical attributes, and also coping with missing attributes and multiple classes, expanding the number of problems that could be tackled by the improved algorithm.

Equation 10 will successfully prune a rule for a binary class problem. RIPPER adds multi-class capabilities by grouping instances by class as $C_1, C_2 \ldots C_k$ (where k is the number of classes in the data-set), and then ordering these groups by class based on the frequency that each class appears in the data-set; C_1 being the class with the fewest number of instances and class C_k the one with the greatest number of instances. RIPPER then uses the IREP learning processes to create a rule where the positive class is set to C_1 and all other classes become the negative class. This is repeated until a single class remains, which is then used as the default rule (Cohen 1995).

2.4 Decision Tree Induction

Decision trees use a divide-and-conquer approach to their construction. Starting from the root node, child internal nodes are recursively created to partition the training data into different sets. This divide-and-conquer strategy is analogous to the IRL strategy commonly used in rule induction, in the sense that it reduces the problem of creating a tree to smaller problems of selecting an attribute to create an internal node to partition the data. Algorithms that generate decision trees have been developed for both the regression and classification tasks and are discussed in the following sections.

2.4.1 Regression Tree Induction Algorithms

A classical regression tree algorithm is Classification and Regression Trees (CART) (Breiman et al. 1984). CART creates binary partition trees, where each node contains a logical condition that evaluates to either a true or false value. If the condition holds, the left branch is taken; otherwise, the right branch is taken upon failure. CART is also able to use nominal and continuous attributes as targets during tree construction, allowing the creation of classification or regression models, respectively. The trees are grown until no further improvement can be made. When constructing regression trees, CART selects attributes for internal nodes using either Least Squares or Least Absolute Deviation (LAD). LAD is the minimisation of the sum of absolute errors (Equation 4) in each of the branches. Once a complete tree is produced it undergoes pruning based upon the cost complexity measure:

$$R_a(T) = R(T) + a|T| \tag{11}$$

where $R(T)$ is the cost on the training sample, $|T|$ is the total number of leaf nodes, and a is the complexity bias (if $a = 0$ then larger trees take preference).

As a is increased, the trees will become smaller as the least helpful branches will be pruned; a can be increased between 0 and the maximum value required to prune all splits, and can be tuned by a user to limit or increase the size of the produced pruned tree. Pruning produces a number of variations of the original un-pruned tree. The tree that has the lowest cost is chosen as the optimal tree (Wu et al. 2008).

The second regression tree algorithm, called M5, solves regression problems by creating model trees. Model trees are regression trees, where the leaf nodes contain a linear model that can be used to produce a prediction, instead of a predicting a single value (Frank et al. 1998). Figure 2.1b shows an example of a simple model tree.

Model trees are grown from the root node with attributes chosen using the standard deviation split point generation method. This method attempts to maximise the expected reduction in the error of the target value in a subset of instances. The expected error reduction is given by:

$$\Delta error = \sigma(T) - \sum_i \frac{|T_i|}{|T|} \cdot \sigma(T_i) \tag{12}$$

where T is the entire set of training instances, T_i is the i-th subset of instances associated with the i-th branch coming out from a tree node and $\sigma(T)$ is the standard deviation of the target value of a (sub-)set. To find the optimal split point p for a continuous attribute att_c, the current covered instances are scanned from beginning to end (the entire instance set is scanned if a continuous attribute is the attribute selected to be added as the root of a tree). Each split point generates two candidate instance sub-sets T: one subset containing instances that satisfy the condition $att_c < p$ and another containing instances that satisfy the condition $att_c \geq p$.

For nominal attributes, this methodology is modified so that the algorithm evaluates all combinations of values that produce two non-empty sets of training

instances. The binary partition that gives the best predicted reduction in error is chosen.

Once a tree is grown and each branch is terminated by a leaf node containing a prediction, a pruning step is undertaken. The pruning step replaces internal nodes with leaf nodes (linear models). Starting from a leaf node, the algorithm moves up the tree to the next internal node. A linear model is then generated and placed at the internal node. If the quality of the model tree is improved the sub-tree is pruned, i.e., the linear model replaces the internal node and the sub-tree. This process continues until the accuracy of a branch is not improved by the linear model. The linear model produced by M5 is not a model that uses all the available attributes, but is simplified to only use the attributes present in the sub-tree that will potentially be pruned. In the example tree in Figure 2.1b, the linear models on each of the branch nodes will have replaced a split on attribute 3.

The authors of M5 note that while typical regression trees cannot give predictions outside of the current data-set, model trees can be extrapolated with possible undesired consequences if the model created is using training data that does not represent the test data or a model is used to make predictions on an area outside of the training data (Quinlan 1992). The addition of linear models does reduce the comprehensibility of the entire model as users must interpret the decisions made by the linear models in order to understand the predictions.

2.4.2 Classification Tree Induction Algorithms

C5.0 (see5), like its predecessors (C4.5 and ID3), is a well-known classification algorithm that produces tree models (Quinlan 1993, 1996; Wu et al. 2008) [3]. C5.0 uses the entropy of the class distribution (a concept from information theory) to identify good splits for internal nodes by calculating the information gain ratio

[3]C5.0 can be downloaded from https://www.rulequest.com/see5-info.html

(Equation 16) of attributes in the training data. A leaf node is then created if a branch only contains training instances belonging to a single class, or if all possible splits result in the same distribution of the classes and therefore the information gain is 0. When creating a leaf node, the most frequently observed class (majority class) is chosen as the class predicted by that node.

The information gain and gain ratio are calculated as follows. First, the concept of information, based on the entropy of the class distribution, is defined as:

$$Info(T) = -\sum_{i=1}^{c} \frac{freq(C_i, T)}{|T|} \times log_2(\frac{freq(C_i, T)}{|T|}) \tag{13}$$

where T is the set of training instances at the current tree node, C_i is one of the c target classes and $freq(C_i, S)$ is the frequency of class i in the instance set T. We can then define the entropy gain for a particular attribute S as:

$$Gain(T, A) = Info(T) - \sum_{j=1}^{n} \frac{|T_j|}{|T|} \times Info(T_j) \tag{14}$$

where A is the attribute being tested, n is the number of outcomes (values) of the logical test using attribute A and T_j is the subset of training instances that has value j for attribute A. Simply, this calculates the information gain that would be achieved if this attribute was used as an internal node (with the corresponding logical test) based on the known class frequencies for each of the allowed values for attribute A. The gain can then be combined with the attribute split information, which is defined as:

$$SplitInfo(A) = -\sum_{j=1}^{n} \frac{|T_j|}{|T|} \times log_2(\frac{|T_j|}{|T|}) \tag{15}$$

Split information is required as attributes with many values will naturally produce higher gains than those with fewer, and to mitigate this tendency, the split information acts as a normalising factor across attributes. The gain and split

information are then combined to calculate the gain ratio as:

$$GainRatio(T, A) = \frac{Gain(T, A)}{SplitInfo(A)} \tag{16}$$

The gain ratio can then be used to measure and rank all the attributes in the data-set and the attribute with the highest gain ratio[4] will be used to generate a new internal node and split the set of instances at the current node. The divide-and-conquer process is repeated for each generated instance sub-set until the stopping criteria are met and leaf nodes are generated.

2.5 Discussion of Model Representation for Regression and Classification Tasks

SVRs and symbolic GPs produce models that are much less comprehensive than other algorithms that produce Decision Trees and Regression and Classification Rules. The non-linear models produced by SVRs and GPs are algebraic expressions, which contain terms for each predictor attribute. The user is required to understand how these expressions will interact with each other as different test input instances are presented to the algorithms for prediction.

On the other hand, both regression and classification rules and decision trees have simple logical tests directly based on the original predictor attributes (without applying non-linear transformations to those attributes), and when a conjunction of such tests are satisfied, this directly leads to a prediction. For trees, the path from the root node can be easily followed, and when examining rule lists, the antecedent of the satisfied rule can be inspected as a series of simple logical tests.

[4]The attribute information gain must also be equal or higher than the average information gain over all attributes, in order to prevent the choice of an attribute which has a high gain ratio just because its split info is very small.

The four types of regression models discussed here have a number of advantages and disadvantages. The non-linear models produced by SVRs produce very good models with low prediction error rates, however their complexity decreases their comprehensibility. Comprehensibility is important when models are presented to experts as they can easily verify the models against the existing domain knowledge, creating trust in models and improving acceptance. M5 combines the comprehensibility of decision trees with the expressiveness of linear models. The linear models produced are simpler than conventional linear models due to the restriction in available attributes, as the linear model is only allowed to use attributes that have been previously pruned from a sub-tree.

Regression rules provide such a simple and comprehensible structure, combined with the ability for rules to be pruned at the individual level. This is in contrast to regression trees, where pruning affects multiple leaf nodes at the same time reducing the number of predictions that a model can make. Given the advantages and flexibility of regression rules, this is the model chosen for conducting the research.

Chapter 3

Domain Knowledge and Monotonic Constraints

Domain knowledge represents prior knowledge about the data at hand. It is usually provided by "human" experts (domain experts) based on their knowledge of the field. However, it could also be provided by previous machine learning algorithms that have extracted knowledge from the data or an inspection of the data-set. It is rare to find a domain that is untouched, where nothing is known a priori.

Many have identified the importance of domain knowledge, including Witten et al. (2016), who state: "*Knowledge of the domain is absolutely essential for success.*". Witten et al. (2016) classify domain knowledge into three kinds of relationships: semantic, causal and functional. Semantic relationships are those that encapsulate relationships, where if one attribute is included then another must be as well and to include one without the other is nonsensical. Causal relationships occur when one attribute causes a second, creating chains of connected attributes. Finally, functional relationships are seen as the values of one attribute determine the values of the second. Without knowledge of functional relationships the construction of tautologies is possible, when these relationships

are re-discovered during an automatic learning phase.

Frawley, Piatetsky-Shapiro and Matheus (1992) state that discovering new knowledge is computationally expensive. Domain context and additional knowledge about the data is useful to guide a focused search, potentially reducing the complexity of the task. However, they also acknowledge that adding domain knowledge into a search can restrict the search by telling it what to look for and where to look for it, leading to a restriction of the search space and the missing of valuable discoveries. These concerns show that any domain knowledge given to an algorithm has to be carefully constructed and considered, so that a search is not overly constrained and can find valuable additional knowledge. The correctness of domain knowledge is also important, as incorrect patterns will disrupt the ability for machine learning algorithms to find good models.

As an example of the difficulty of using domain knowledge, when using machine learning techniques to detect oil spills in satellite data, Kubat, Holte and Matwin (1998) used domain experts to identify and manually construct features using their domain knowledge. This was only partially successful as they acknowledged that the features that the domain experts expected to be useful were not. They hypothesised that if they had explicitly encapsulated the domain knowledge and used this to guide the machine learning algorithms, this additional knowledge would allow more useful features to be identified.

Semantic constraints provide a mechanism to incorporate existing domain knowledge into the construction of new models. In this chapter, I will present and discuss existing work in the literature relating to domain knowledge and, more specifically, monotonicity.

One example of existing knowledge that could be incorporated into the construction of models can be illustrated in house rental prices data. When you consider house rent, the price can depend on many features, such as the location

Table 3.1: House rental data set with monotonic relationships. As the Floor Area value increases, so does the Rental Value.

Target Attribute	Predictor Attributes	
Rental Value	Floor Area	Location
£300	45	2
£600	80	1
£250	33	3
£400	65	2
£450	70	1
£350	54	2

and floor area. Table 3.1 shows a simple hypothetical data-set. An obvious relationship in this data-set is that, as the floor area increases so does the rental price for all possible pairs.

Another example of existing knowledge could be the decreasing fuel efficiency of automotive engines as the number of cylinders increases, due to additional friction losses attributed to the extra pistons. A model that does not conserve these patterns would seem counter intuitive and may lead to model rejection by domain experts. For example, Hoover and Perez (2000) state that the economic field distrusts data mining as a technique to search for models due to the discovery of accidental correlations. They say "*Data mining is considered reprehensible largely because the world is full of accidental correlations, so that what a search turns up is thought to be more a reflection of what we want to find than what is true about the world.*" (Hoover and Perez 2000). Semantic constraints provide a method for guiding searches by providing information on real correlations present within the data.

There are many different possible monotonic constraints —Martens and Baesens (2010) presented a taxonomy of such constraints. This taxonomy can be seen

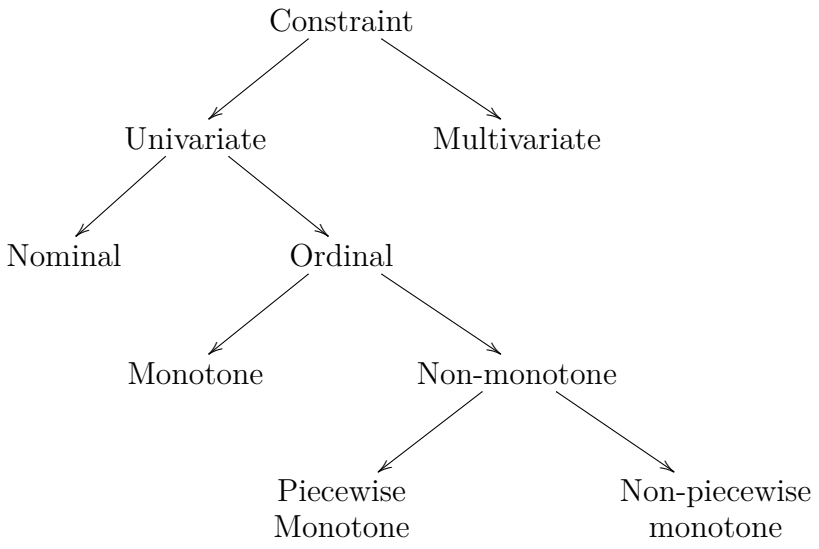


Figure 3.1: Taxonomy of constraints. Constraints can exist in soft and hard variants. Uni-variate Ordinal Monotone constraints are the most common found in the literature (Martens and Baesens 2010) .

in Figure 3.1, where the constraints featuring in the taxonomy can be implemented in either *hard* or *soft* variants. Hard constraints are enforced rigidly, guaranteeing compliance in the final model, while soft constraints bias a preference towards compliance but will not enforce them if model quality would be badly affected. So far, the literature has focused on implementing monotonic constraints when tackling the classification task, since many real-world problems contain monotonic properties such as house prices, customer credit ratings (Ben-David, Sterling and Tran 2009). Monotonicity is found in many different fields, including house prices, medicine, finance and law.

3.1 Monotonicity

Taking the first example of house prices, it is expected that as the total floor area of a property increases the value of the property will also increase. This

is illustrated in the example data shown in Table 3.1, where the rental value is always monotonically increasing with respect to the floor area. From the rental price data set shown in Table 3.1, we could extract the rules:

`IF floor area` $\leq$ `65 THEN rental value = 325 ELSE`

`IF floor area` $\geq$ `65 AND location = 1 THEN rental value = 525`

where we can see that the rules have a monotonic relationship between floor area and rental price with respect to each another, as no prediction can be made where the floor area decreases and the price would increase. Incidentally, the model is also monotonic w.r.t. location, as the second rule does not constrain this attribute. However, this may allow the prediction of values that are non-monotonic w.r.t. each other when considering all the values in the instances being compared. While this example is of a regression problem where the prediction is a real value, monotonic features can also exist in ordinal classification problems where there is a natural order present to the classes for example the classes `Small, Medium, Large` have an order.

Many data mining algorithms do not enforce monotonic constraints when constructing models and still produce good models. However, if models violate these constraints they may not be accepted by experts as valid —conforming to monotonicity constraints improves model acceptance (Feelders and Pardoel 2003; Duivesteijn and Feelders 2008).

Monotonicity can be defined formally in the following manner. Let $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2 \times \cdots \times \mathcal{X}_i$ be the instance space of i attributes, $\mathcal{Y}$ be the target space, and $f : \mathcal{X} \to \mathcal{Y}$ is the function that maps the attributes to the targets. It is also assumed that both the instance space and the target space have an ordering. A function can then be considered monotonic if one of the following two equations hold:

$$\forall \vec{x}, \vec{x'} \in \mathcal{X} : \vec{x} \leq \vec{x'} \implies f(\vec{x}) \leq f(\vec{x'}) \tag{17}$$

$$\forall \vec{x}, \vec{x'} \in \mathcal{X} : \vec{x} \leq \vec{x'} \implies f(\vec{x}) \geq f(\vec{x'}) \tag{18}$$

where $\vec{x}$ and $\vec{x'}$ are two attribute vectors (instances) in instance space, $\vec{x} = (x_1, x_2, \cdots, x_i)$ where i is the number of attributes in the training set (Potharst, Ben-David and van Wezel 2009). In other words, $f(\vec{x})$ is monotonic if and only if all the pairs of instances $\vec{x}$, $\vec{x'}$ are monotonic with respect to each other. Two equations are required to encapsulate monotonic relationships as a relationship can be either monotonically increasing, where as $\vec{x}$ increases so must $f(\vec{x})$, or monotonically decreasing relationships, where as $\vec{x}$ increases $f(\vec{x})$ decreases.

Monotonicity constraints can be enforced in a number of different stages in the data mining process. Firstly, in the pre-processing stage where the training data is manipulated so that it becomes monotonic in nature. Secondly, in the model construction stage, where models are constructed in a monotonic fashion. Finally, constraints could be enforced in a post-processing stage, which modifies constructed models so that they are monotonic. Table 3.2 presents previous work, identified in the literature review, categorized into these categories. To the best of our knowledge in the literature algorithms that incorporate monotonic constraints only enforce these constraints in a single phase.

Constraints can be implemented as *hard* or *soft* variants. Hard constraints are enforced rigidly, so that the algorithm rejects any model or change to a model that would cause a violation to occur. This method can cause the rejection of good models due to small violations in their monotonicity. The second method, enforcing soft constraints, aim to balance the monotonicity of a model against other model quality measures —mainly predictive accuracy.

Table 3.2: Previous works enforcing monotonicity constraints, categorised by their enforcement stage.

Enforcement Stage	Algorithm Names	Reference
Pre-Processing	Relabelling data to ensure monotonicity	Daniels and Velikova (2006)
Model Construction	MID (ID3 with Monotonic Constraints)	Ben-David (1995)
	Ordinal Learning Model	Ben-David, Sterling and Tran (2009)
	Mk-NN	Duivesteijn and Feelders (2008)
	Fused Monotonic Decision Trees	Qian et al. (2015)
	Ordinal Stochastic Dominance Learner	Lievens, De Baets and Cao-Van (2008)
	MID-RF	González, Herrera and García (2015)
	MC-SVM	Chen and Li (2014)
	ORNN(ELM)	Fernández-Navarro, Riccardi and Carloni (2014)
	AntMiner+ with Constraints	Martens et al. (2006a)
Post-Processing	Decision Trees with Monotonic pruning	Feelders and Pardoel (2003)
	Isotonic Classification Trees (ICT)	Van De Kamp, Feelders and Barile (2009)
	RULEM	Verbeke, Martens and Baesens (2017)

3.1.1 Non-Monotonicity Index

Some models and domains have varying degrees of monotonic features, with some domains being more monotonically noisy than others. One measure of the amount of non-monotonicity is the Non-Monotonicity Index (NMI) (Ben-David 1995), given by:

$$NMI = \frac{\sum_{i=1}^{k} \sum_{j=1}^{k} m_{ij}}{k^2 - k} \quad , \tag{19}$$

where m_{ij} is 1 if the pair of i-th and j-th objects violate the monotonic constraint and 0 otherwise and k is the number of objects. Ben-David (1995) used the NMI to calculate the degree of violations in a decision tree, where each of the objects were comprised of both a leaf node and the decisions made when traversing the tree from the root note to the leaf node. However, this can be applied to other models and even entire data-sets.

When applying it to a data-set, each instance becomes an object and each pair of instances can be compared to look for a violation between an independent (predictor) attribute and the dependent (target) attribute. A systematic search of all the independent attributes can be performed to identify good constraint candidates if domain experts cannot be relied upon or to verify the decisions made by domain experts.

3.1.2 Enforcing Constraints in the Pre-Processing Stage

Implementing constraints as a pre-processing step involves manipulating the training data to remove any violation of monotonic constraints found within the training data. Duivesteijn and Feelders (2008) approached the problem with the objective to re-label the training data to ensure it was monotonic with as few changes as possible. This is achieved by creating a Monotonic Violations Graph (MVG), where they can compute the minimum number of relabels required to create a

monotonic training set. This can be achieved in time complexity $O(k^3)$ where k is the number of instances. While this time is cubic, the authors note that in practice it is generally quick as most points will not be connected in the MVG as they will not be involved in any of the constraint violations. This will be true if the violations are caused by random noise and not a systematic trend.

One disadvantage of using the pre-processing stage to enforce monotonic constraints is that there are no guarantees the resulting model produced by the learning algorithm during the construction stage will be monotonic. Enforcing constraints at later stages, including construction and post-processing, will guarantee that a model is monotonic. The advantage of using the pre-processing stage to apply constraints is that any subsequent learning algorithm can be used to create the final model with the aim that more monotonic models will be generated.

3.1.3 Enforcing Constraints in the Model Construction Stage

Enforcement of constraints during model construction involves modifying the learning process to be aware of monotonic constraints. Depending on the type of constraint, this might mean suggesting a preference to more monotonic solutions, in the case of soft constraints. On the other hand, we have to prevent violations being introduced by the learning algorithm, if hard constraints are being enforced. Enforcing constraints during model construction allows both monotonicity and model quality to be optimised together.

Soft constraints have been implemented in the model construction stage of decision trees by Ben-David (1995), with the introduction of the decision tree algorithm MID. The approach attempts to minimise the value of the non-monotonic index of each decision tree produced. The index is the ratio between the number of non-monotonic leaf node pairs and the maximum number of pairs that could have been non-monotonic. First a non-monotonicity matrix m is constructed,

which has dimensions k (the number of leaf nodes in the tree). This matrix is used to find the number of violations in the current tree, given by:

$$\begin{aligned} W &= \sum_{i=1}^{k}\sum_{j=1}^{k} m_{ij} \\ m_{ij} &= \begin{cases} 1 & \text{if } i,j \text{ is non-monotonic} \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{20}$$

where i and j denote the current cell being referenced in the matrix m. W can then be used to find a tree's non-monotonicity index, given by:

$$I_{a_1 \ldots a_v} = \frac{W_{a_1 \ldots a_v}}{k^2_{a_1 \ldots a_v} - k_{a_1 \ldots a_v}} \tag{21}$$

where $a_1 \ldots a_v$ are the attributes being constrained and $k_{a_1 \ldots a_v}$ are the number of leaf nodes that have an ancestor node that uses one of the constrained attributes as a internal decision node. The $I_{a_1 \ldots a_v}$ index can be converted to an ambiguity score A and then incorporated with a tree accuracy score T, given by:

$$A_{a_1 \ldots a_v} = \begin{cases} 0 & \text{if } I_{a_1 .. a_v} = 0 \\ -(log_2(I_{a_1 \ldots a_v}))^{-1} & \text{otherwise} \end{cases} \tag{22}$$

$$T_{a_1 \ldots a_v} = E_{a_1 \ldots a_v} + RA_{a_1 \ldots a_v} \tag{23}$$

where $E_{a_1 \ldots a_v}$ is the accuracy associated with the constrained tree, R is the importance (weight) given to the monotonicity of trees produced. It should be noted that an entropy-based accuracy method was used in this example, which is a logarithmic function, hence the ambiguity value is also made logarithmic. This modification is performed to ensure that both measures used to calculate the total T scale at the same rate without one dominating unduly. If the error measure E

is altered then care should be taken to modify A based on the new scale to ensure the algorithm remains well-behaved. It was found that this method of using a combined measure produced fewer models that breached monotonicity constraints while not significantly degrading the accuracy of the trees generated (Ben-David 1995).

Ben-David, Sterling and Tran (2009) also investigated the effects of monotonicity constraints on ordinal classifiers, with the conjecture that adding monotonicity constraints to learning algorithms will impair their predictive accuracy against those that do not. Ordinal classifiers are classifiers that are aware that there is an order to discrete categories, e.g., credit rating may have the categories "poor", "acceptable" and "good" which have an obvious order.

The results presented contain two unexpected results. First, the authors found that ordinal classifiers did not significantly improve over non-ordinal classifiers. Secondly, the monotonicity algorithms Ordinal Learning Model (OLM) (Ben-David, Sterling and Pao 1989) and Ordinal Stochastic Dominance Learner (OSDL) (Cao-Van 2003; Lievens, De Baets and Cao-Van 2008) were not able to significantly outperform a majority-based classifier, which trivially predicts the most frequent class in the training set for all new instances in the test set. It is theorised that these results were due to noisy data sets: the monotonic classifiers enforced hard constraints, in the presence of noisy data a softer approach may lead to better results (Ben-David, Sterling and Tran 2009).

Qian et al. (2015) have explored the possibility of fusing monotonic decision trees to improve the predictive accuracy of the final model. This is achieved by reducing the original data set to create data sub-sets that will naturally maintain the original data-set's monotonicity, as a monotonicity is preserved in sub-sets of monotonic data. From these new reduced data sub-sets, monotonic trees can be constructed. Each leaf node of a decision tree then contain probabilities of the correctness of the prediction based on the reduced training set. When a

prediction is required, the probabilities at each leaf nodes are averaged with the highest average being the class predicted by the model. The authors compared the predictive accuracy of the fused monotonic trees (each constructed on a data sub-set) to a single tree which had access to the entire data-set and found an improved accuracy across all 10 of the tested data-sets.

González, Herrera and García (2015) introduced MID-RF, a random forest ensemble algorithm that implements soft monotonic constraints. These constraints aim to influence the monotonic nature of the ensemble produced, and at the same time, to not require all trees produced to be completely monotonic. MID-RF uses the previously discussed MID algorithm to create each tree in the ensemble, where each MID tree is constructed using a random sample of the training data-set and a random selection of attributes. The constructed trees are then ordered by NMI score, with the top p trees —those with the lowest NMI score— retained as part of the ensemble. The parameter p is used to increase or decrease the ensemble size.

Chen and Li (2014) have implemented a monotonically-aware SVM for credit checking, where the goal of the algorithm is to classify individuals as good or bad credit risks. This is achieved by constructing constraints from the training data-set and adding the constraints into the SVM model so that the hyper-planes constructed will obey the given constraints. When testing against the original SVM, MC-SVM produced models with statistically significant increases in predictive performance on the two credit check data-sets being tested, along with an increase in the monotonicity of the results. However, as the number of constraints identified and provided to the algorithm grew, so did the algorithm execution time.

3.1.4 Enforcing Constraints in the Post-Processing Stage

Feelders and Pardoel (2003) have suggested that using non-monotonic criteria

in tree construction is not beneficial as splits later in the construction process can transform a tree from a state of non-monotonicity to one that is. Therefore pruning methods have been developed to perform the minimal number of changes to make a tree monotonic in a post-processing phase (Feelders and Pardoel 2003).

The first method proposed is the Most Non-monotone Parent (MNP) method, which aims to prune the node whose removal will remove the most number of existing non-monotonic pairs of leaf nodes. This method has the disadvantage of possibly creating new non-monotonic pairs. The second method proposed is the best fix method, which prunes the node that gives the biggest reduction in non-monotonicity —based on the NMI calculation found in Equation 19. The authors have also combined these pruning methods with existing complexity-based pruning methods and found that the monotonic trees produced no significant difference in performance compared to trees produced without monotonic pruning. However, it was observed that the trees produced by the algorithms that considered the tree's monotonicity were smaller, which aids the comprehensibility of the models.

Van De Kamp, Feelders and Barile (2009) introduced a new post-processing technique, that, given any non-monotonic classification tree, can re-label the leaf nodes of the tree and create a monotonic one. The Isotonic Classification Tree (ICT) technique can therefore be applied as a post-processing step to any algorithm that creates decision trees for the classification task. ICT evaluates each leaf in the non-monotonic tree and identifies the non-monotonic pairs. At this point, isotonic regression is used to find the re-labelling of leaf nodes that enforces monotonicity constraints and produces the smallest increase in error on the training data-set. Once all pairs have been re-labelled, the tree is pruned by merging any branch (sub-tree) that evaluates to a leaf node which will make the same prediction. This procedure is repeated until the tree is found to be monotonic.

The ICT technique was tested on both noisy monotonic data-sets and data-sets that had been re-labelled to ensure the monotonicity of the data. On the noisy

data-sets, it was shown that ICT trees performed better. On re-labelled data-sets, there was no difference in predictive performance. However, a reduction in the size of trees produced by ICT was found compared to standard decision trees, providing simplified models.

Verbeke, Martens and Baesens (2017) introduced a new algorithm, called RULEM, that tackles the monotonic problem in a different way. While still a post-processing technique, RULEM adds additional rules to a rule list to force monotonic behaviour. One advantage of RULEM is that any learning algorithm that produces a model, which can be transformed into a list of rules, can be fixed and made monotonic by the algorithm.

RULEM fixes rule lists by adding new rules to fix any non-monotonic areas created by the rule list. RULEM first creates an n-dimensional matrix, where n is the number of attributes in the solution space. Rules from the original rule list are then added to this solution space, claiming the regions that they cover and declaring the prediction they would make. Any non-monotonic regions can then be identified by a decreasing or increasing prediction along the dimension of any constrained attributes. Rules are then iteratively generated to fix the identified non-monotonic regions with respect to the existing rules. Finally, these rules are compacted to reduce the number of rules added. Compaction is needed as RULEM creates a rule for each cell in the matrix. After all rules are created, these rules can be replaced with ones that cover whole areas of the matrix. The new compacted rules are then added to the top of the rule list to ensure they create a monotonic rule list. The top is the only place that ensures a monotonic list, as in an ordered list the first rule that covers an instance makes a prediction. Therefore, being at the top of the list ensures the added rules will make a prediction before the rule that would result in a violation.

One disadvantage of an additive post-processing technique such as RULEM is the risk that the additional rules will over-fit the data and bloat the model.

Also, these rules are not created and added based on their predictive power or any notion of correctness other than fixing a non-monotonic area in the current model. These new rules may make poor predictions when used, which reduces the overall predictive power of the model.

Chapter 4

Ant Colony Optimization

Ant Colony Optimization (ACO) is a stochastic meta-heuristic that has been used to approximate solutions to many NP-hard optimisation problems. The ACO meta-heuristic was first proposed by Marco Dorigo (1992) based on the foraging behaviour used by ants to search for food sources and then communicate the location of food to the whole colony. In nature, this is achieved via stigmergy and the deposition and sensing of pheromone by the ants in the colony.

In this chapter, I will introduce the ACO meta-heuristic by first summarising the original observations of ants in nature, followed by the generic ACO meta-heuristic and a classical problem before finally introducing existing algorithms that tackle the data mining classification task. As far as I am aware there are no existing ACO-based algorithms that tackle the regression task in data mining.

The current chapter is structured as follows. First, I will present the initial observations recorded using natural ant colonies, specifically work involving Argentinian ants. Next, in Section 4.2 the Ant Colony Optimization meta-heuristic will be introduced followed by a discussion of combinatorial and continuous ACO variants, in Section 4.3 and Section 4.4, respectively. Finally, in Section 4.5 existing ACO-based algorithms for data mining tasks will be introduced.

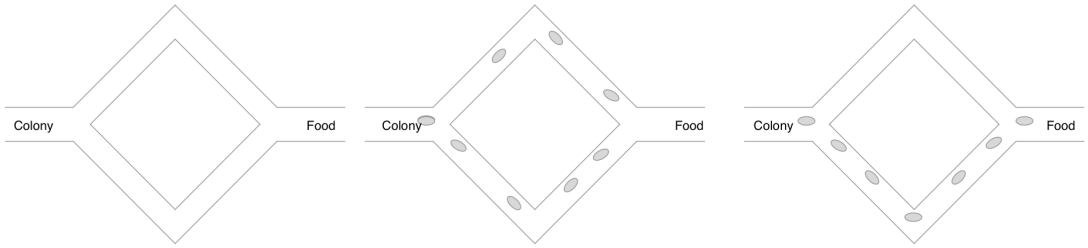
4.1 From Nature to Artificial Ants

In nature, ant colonies are able to complete complex tasks even though each individual has limited capabilities —in essence an ant colony acts as a distributed system. A number of experiments have been conducted that show that ant colonies can accomplish their tasks and coordinate responses through the use of stigmergy, that is, using the environment as an indirect communication medium (Deneubourg, Pasteels and Verhaeghe 1983; Manderick and Moyson 1988; Dorigo and Stutzle 2004). In ant colonies, this stigmergy is achieved with the deposition of pheromone along the paths that ants walk on and subsequently the detection of the deposited pheromone by other ants who are drawn towards it. Therefore, areas with higher pheromone deposition are more attractive to ants than those with lower amounts of pheromone (Dorigo and Stutzle 2004).

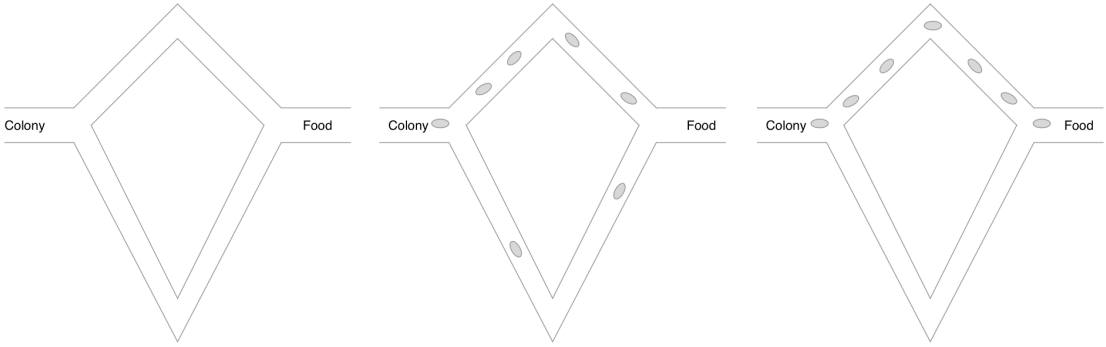
Goss et al. (1989) and Deneubourg et al. (1990) conducted several interesting experiments on Argentinian ants, including a double bridge experiment —the experimental setup can be seen in Figure 4.1. When given the option of two branches with identical lengths, as seen in Figure 4.1a, ants will select a single branch with equal probability to form a single trail between the nest and the food.

In Figure 4.1b, ants are given a different choice, the top branch is shorter than the bottom branch. Initially ants will explore both branches, looking for the best route from the nest to the food. Eventually a single dominant path will emerge, with the shorter path more likely to be chosen. In fact, if the ratio between the long branch to the short branch is greater than 2, that is to say if the long branch is twice as long as the short one, the shorter branch is always chosen (Goss et al. 1989).

Goss et al. (1989) remarked that this occurs because it takes less time for the ants to traverse back and forth along the shorter branch. This branch will rapidly contain pheromone from the initial ants that traversed the path and the ants returning from the food source. For the longer branch this does not occur



(a) Double branch experiments with two branches of equal length, repeated experiments show the ants will randomly select either branch with an equal probability.



(b) Double branch experiment with branches of un-equal length, after initial exploration the ants will repeatedly form a trail along the shortest path, ignoring the longer branch.

Figure 4.1: Double branch experiment with ants. Initially the branches are empty. Ants then explore the space using both branches, before converging and forming pheromone trails along the shortest paths.

until much later, as it takes the ants longer to complete a round trip from the nest to the food. This allows the amount of pheromone on the shorter path to build faster than the longer one, leading to subsequent ants preferring the shorter route.

Deneubourg et al. (1990) proposed a stochastic model to describe the ant colony behaviour that calculated the probability that an ant would chose either the longer branch $P(l)$ or the shorter branch $P(s)$, if i ants have already crossed the bridge depositing i amounts of pheromone. The probability that an ant will chose the shorter path is given as:

$$P(s) = \frac{(k + s_i)^n}{(k + s_i)^n + (k + l_i)^n} \quad (P(s) + P(l) = 1) \tag{24}$$

Algorithm 4.1: High-level ACO pseudo-code

```
1 while Not Terminated do
2 |   ConstructSolutions()
3 |   LocalSearch() // Optional step
4 |   PheromoneUpdate()
5 end
6 return BestSolution
```

$$s_{i+1} = s_i + \delta, \quad l_{i+1} = l_i + (1 - \delta) \quad (s_i + l_i = i) \tag{25}$$

where k is a parameter that represents the attraction of an unmarked branch, where a higher k requires more pheromone to be deposited before the pheromone becomes the dominant factor in the decision made by an ant; δ is the stochastic variable that takes a value of either 1 if ant $i+1$ takes the shorter path s or 0 if the longer path l is taken —this has the effect of increasing the amount of pheromone present on one of the branches; finally, n determines the degree of non-linearity in the choices the ant makes, higher values of n will mean a branch with a modest increase in pheromone compared to another will lead to a large increase in the probability the branch will be taken.

4.2 Artificial Ants

Dorigo (1992) introduced the ACO meta-heuristic in his PhD thesis. The ACO meta-heuristic models the foraging behaviour of ant colonies discussed in the previous section to solve other complex problems using simple agents. Algorithm 4.1 shows a high-level pseudo-code of the ACO meta-heuristic (Dorigo 1992; Dorigo, Maniezzo and Colorni 1996; Dorigo and Stutzle 2004).

The algorithm works as follows. First each ant constructs a solution (line 2) influenced by the pheromone deposited in the search space by previous ant iterations. Then a local search operator is optionally applied (line 3). This often

takes the form of a simple hill climbing strategy, where small incremental changes are made in an effort to increase the quality of the generated solutions. Next, a pheromone update procedure is performed (line 4). There are normally two phases during pheromone update. The first increases the amount of pheromone belonging to good solutions, increasing the likelihood they will be selected in subsequent iterations. The second phase is pheromone evaporation, where the amount of pheromone on unused areas of the search space is reduced, mimicking the evaporation of the chemical pheromones deposited by ants in nature. Pheromone evaporation is important, as it allows the colony over time to forget poor decisions made by previous iterations. These steps (lines 1-5,) are repeated until stopping criteria are reached, this could be a maximum number of iterations or the convergence of the pheromone trails to a single solution causing the colony to stagnate.

A classical ACO problem is the travelling salesman problem (TSP). The TSP is a problem where a salesman has a number of cities to be visited, which are all interconnected. The salesman wishes to choose the shortest route that allows him to visit all the cities only once. A very simple example TSP can be seen in Figure 4.2. Optimising the path taken by the salesman to visit each city has parallels to the foraging behaviour of ants in nature, where they are trying to find the shortest path from their nest to a food source.

The original ACO, Ant-System (Dorigo, Maniezzo and Colorni 1996), implemented a pheromone graph and belongs to the combinatorial ACO approach, which will be discussed in the next section. A second pheromone model is the solution archive which belongs to the continuous ACO approach and will be discussed in Section 4.4.

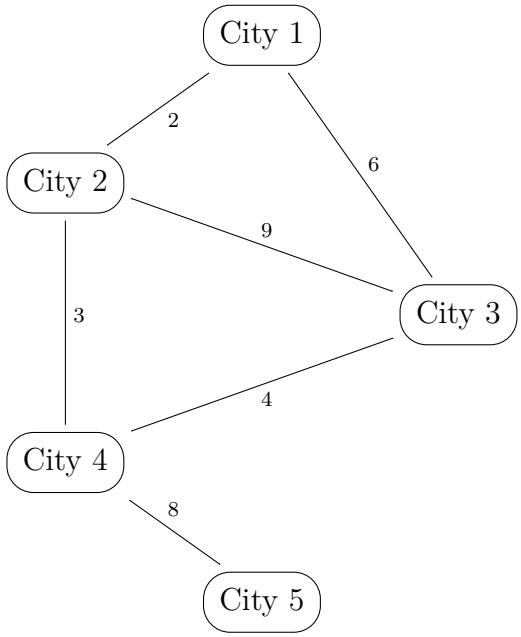


Figure 4.2: Example graph representing a Travelling Salesman Problem, where each node in the graph is a city the salesman is required to visit. Each edge has a cost associated which is the distance between cities. The objective is to create a tour that minimises the distance travelled by the salesman.

4.3 Combinatorial ACO

In combinatorial ACO-based algorithms the search space is represented by a construction graph with each node representing a component of the solution connected by edges that ants can lay pheromone along. Figure 4.2 could be used to represent the construction graph. A solution is therefore a combination of these nodes and the path taken between them. If we consider the TSP, each city can be represented as a node in the pheromone graph and each connection as an edge that the artificial ants can follow.

Recall that an ACO algorithm is an iterative procedure, during the 'Construction' phase (line 2 in Algorithm 4.1) each ant in the colony generates a candidate solution by traversing the construction graph. At each node, a decision is made by an ant to choose the node that should be visited next. This choice is probabilistic,

where the probability of selection is proportional to the current pheromone level on the edge or node and a problem specific heuristic information. Equation 26 shows the probability that an edge E_{ij} is selected by an ant —this equation was used by Dorigo (1992) in the original Ant System (Dorigo, Maniezzo and Colorni 1996; Dorigo and Stutzle 2004):

$$P(E_{ij}) = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in \text{ allowed j}} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}} \tag{26}$$

where τ_{ij} is the pheromone level of the edge connecting nodes i and j; η_{ij} is its heuristic value[1]; α and β are constants that alter the importance of the pheromone and heuristic; l is taken from the allowed set of nodes, where an allowed node is simply one that has a valid connecting edge between itself and the node the ant is currently at. In summary, the probability of an individual edge being selected is the ratio of the product of the heuristic information and pheromone level of the edge in question over the sum of all those products on all the branches an ant is allowed to take.

Once solutions have been generated, an optional step can be performed where a local search can be used to optimise the values chosen by the ants. For example, for the TSP the 2-opt local search operator could be performed. The 2-opt procedure identifies edges in a solution that cross each other it then rearranges the edges so they do not cross. (Mavrovouniotis, Müller and Yang 2017; Stützle and Hoos 1999; Croes 1958).

The quality of the generated solutions is used to modify the pheromone values of the construction graph's edges or nodes. First, edges (or nodes) used in good solutions have their pheromone level increased to increase the probability they are selected in the next iteration —the increase in pheromone value is proportional to the solution quality. All edges then undergo a process of evaporation, where

[1]Heuristic information is used to provide additional information about the quality of the edge an ant could take e.g. for the TSP this could be information regarding the distance between cities.

the amount of pheromone deposited on each edge is decreased. This means that edges that did not have any additional pheromone deposited have their likelihood of selection decreased in subsequent iterations. Pheromone evaporation allows colonies to forget poor decisions they made in the past and explore new areas of the problem domain. The pheromone gives the ant colony an implicit memory, allowing future ants to improve their solutions based on the successes of those before them (Dorigo and Stutzle 2004).

4.4 Continuous ACO

Liao et al. (2014) introduced an archive based ACO algorithm, called $\text{ACO}_{\mathbf{MV}}$. This new algorithm can be applied to optimisation problems that contain mixed variables, e.g., categorical, continuous and ordinal variables. Each variable represents a column of the solution archive, therefore a row represents the different solutions in the archive. Archive-based ACO algorithms dispense with the classical construction graph and the combinatorial pheromone model found in the majority of ACO algorithms and replace them with a solution archive. This can be seen as a shift from sampling a discrete probability function in the form of the construction graph to a continuous probability function in the form of a solution archive which the ants now sample from (Socha and Dorigo 2008).

The solution archive stores a sorted list of the best solutions discovered by the ants so far. An example archive is shown in Figure 4.3, which contains separate sections for the different variable types along with the solution's quality, f(S), and a weighting, ω, based on the solution's rank in the archive. The high-level pseudo-code for $\text{ACO}_{\mathbf{MV}}$ is shown in Algorithm 4.2.

The algorithm maintains a solution archive SA, which contains k solutions.

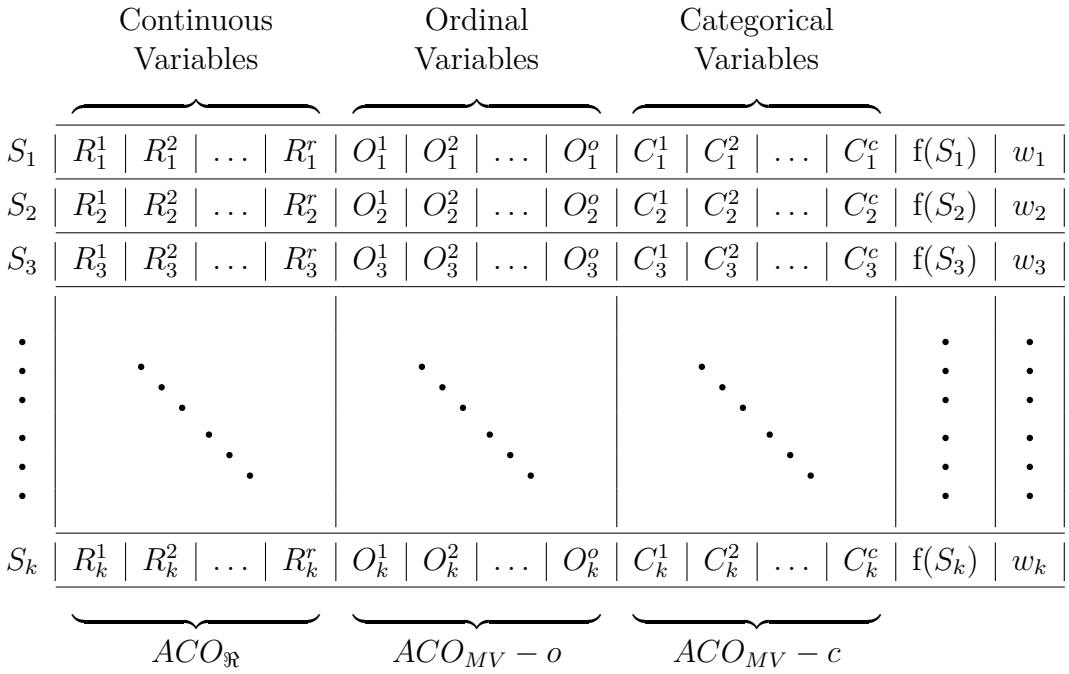


Figure 4.3: Solution archive present in ACO_{MV}. The archive is made up of three sections, one for each of the variable types; continuous, ordinal, and categorical. Finally each solution is ranked by its quality f(S) and is given a weight ω.

Algorithm 4.2: High-level pseudo-code for $\text{ACO}_{\mathbf{MV}}$.

```
1  Initialise Decision Variables
2  // Create k random solutions for the archive
3  SA ⟵ InitialiseArchive()
4  while Termination Criteria Not Satisfied do
5  |  for 1 to M do
6  |  |  ConstructSolutionACO_ℜ
7  |  |  ConstructSolutionACO_{MV−o}
8  |  |  ConstructSolutionACO_{MV−c}
9  |  |  SA ⟵ SA ∪ Solution
10 |  end
11 |  // Sort and select the best k solutions for the archive
12 |  SA ⟵ Best(Sort(SA),k)
13 end
```

The solutions are sorted by quality from best to worst. Each solution S_j is associated with a weight ω_j, which is calculated using a Guassian function as follows:

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} e^{\frac{-(rank(j)-1)^2}{2q^2k^2}} \tag{27}$$

where q is a user defined parameter and $rank(j)$ is a function that returns the rank of solution j. The weight is used to bias the probability of a solution being selected for sampling by subsequent solutions. Therefore, better solutions have a higher probability of being selected during solution creation in subsequent iterations. Solution construction involves three procedures (lines 6-8 of Algorithm 4.2), one for each of the attribute types (continuous, ordinal and categorical).

Continuous attributes are processed by $ACO_{\Re}$ (Socha and Dorigo 2008) as follows. First an ant probabilistically chooses a solution from the archive with the probability of a solution j being chosen given by:

$$P_j = \frac{\omega_j}{\sum_{l=1}^{k} \omega_l} \tag{28}$$

where ω_j is calculated using equation 27. When a solution has been selected, each continuous attribute has a value assigned to it by sampling around the value found in the selected solution using a normal probability density function. Where the mean is set as the value of the selected solution and the standard deviation is calculated as:

$$\sigma = \xi \sum_{l=1}^{k} \frac{|R_l^i - R_j^i|}{k-1} \tag{29}$$

where R_j^i is the value found in the selected solution j; R_l^i are the values found for attribute i in the other solutions in the archive; k is the number of solutions in the archive and ξ is a scaling factor that modifies the convergence rate of the colony, with higher values of ξ slowing convergence.

Ordinal attributes are selected in the same way as continuous attributes. A

mapping between ordinal values and indices is then created. This is because ordinal attributes may not be numeric but could take another form that implies order, e.g., "small", "medium" and "large". Once a value has been generated, it is rounded to the nearest valid index.

For categorical variables, $\text{ACO}_{\mathbf{MV}}$ operates differently to both ordinal and continuous variables. For each categorical variable, an ant will select one of the available values for that variable. The probability of choosing the l-th value from a set of t allowed values is given as:

$$P_l = \frac{w_l}{\sum_{j=1}^{t} w_j} \tag{30}$$

where w_l is the weight associated with the l-th value, calculated as:

$$w_l = \begin{cases} \frac{\omega_{j_l}}{u_l^i} + \frac{q}{\eta}, & \text{If}(\eta > 0, u_l^i > 0) \\ \frac{\omega_{j_l}}{u_l^i}, & \text{If}(\eta = 0, u_l^i > 0) \\ \frac{q}{\eta}, & \text{If}(\eta > 0, u_l^i = 0) \end{cases} \tag{31}$$

ω_{j_l} is calculated according to Equation 27, where j_l is the index of the solution with the highest quality that uses the value v_l^i for the categorical variable i; u_l^i is the number of solutions that use the value v_l^i for the categorical variable i in the current solution archive; finally, η corresponds to the number of values from the t_i available values that are not used by any of the solutions in the archive.

4.5 ACO in Data Mining

Ant-based algorithms have been used in the unsupervised learning task of clustering, including concepts introduced by Deneubourg et al. (1992). These algorithms reproduce the behaviour of some ant species that form heaps out of corpses to create clusters. ACO has also been used in the semi-supervised learning task

—an example is the ANT-LABELER algorithm (Albinati et al. 2015), which used *c*Ant-Miner$_{\text{PB}}$ (Otero, Freitas and Johnson 2013) to create rule lists and an additional self-training procedure to fully label a partially labelled data-set. However, I will be concentrating on supervised learning in this section and the algorithms introduced for this task.

There have been a number of implementations that use ACO-based algorithms to address the classification task in data mining. To the best of my knowledge, no ACO-based algorithms have been developed that attempt to create solutions for the regression task. Therefore, the algorithms discussed here are restricted to classification problems. These classification algorithms can be broken down into two broad groups, the first being graph-based approaches starting with Ant-Miner (Parpinelli, Lopes and Freitas 2002) in Section 4.5.1 and its many extensions in Section 4.5.2.

The second approach to the classification task dispenses with the classical graph-based model and instead uses a new archive approach, these archive algorithms will be discussed in Section 4.5.3.

4.5.1 Ant-Miner

The first ACO classification algorithm, called Ant-Miner, was proposed by Parpinelli, Lopes and Freitas (2002). Ant-Miner follows a sequential covering strategy, where individual rules are created by an ACO procedure. The ants in the Ant-Miner colony search for the best classification rule given the current training data at each iteration of the sequential covering.

Ants traverse a construction graph selecting terms to create a rule in the form `IF` $\texttt{term}_1$ `AND ... AND` $\texttt{term}_n$ `THEN class`, where the `IF`-part represents the antecedent and the `THEN`-part is the class prediction. Each ant starts with an empty rule and iteratively selects terms to add to its partial rule based on their values of the amount of pheromone τ and a problem-dependent heuristic

information η, similarly to Ant System (AS) (Dorigo, Maniezzo and Colorni 1996).

After a rule is created by the ACO procedure, the training instances covered by the rule are removed. A new rule is then created using the ACO procedure. This is repeated until the number of training instances is below a user defined threshold. The high-level pseudo-code for Ant-Miner can be found in Algorithm 4.3. The sequential covering procedure starts on line 2 checking if there are still too many uncovered instances. If this is the case, the construction graph and pheromone matrix are initialised (line 3) and a rule created (line 7). A local search procedure is then applied before the pheromone matrix is updated (line 9). If the rule created in this iteration is the best created so far then the global best rule is updated. This procedure is repeated until the colony size is reached or the colony stagnates (line 6). Stagnation occurs when ants repeatedly create the same rule as they walk the construction graph, this is likely to occur when the pheromone has converged to a single path.

4.5.1.1 Construction Graph and Pheromone Matrix

Ant-Miner creates a construction graph that represents the attributes present in the data-set, each (attribute,operator,value) tuple becomes a node on the construction graph and a potential destination for an ant. Each attribute is fully connected such that a valid edge connects every attribute to all the others.

Once the construction graph has been created, pheromone is required to be deposited onto the edges. The amount of pheromone deposited onto each edge is equal to:

$$InitialPheromone = \frac{1}{Numberof Edges} \tag{32}$$

each edge can be represented by a cell in a matrix where the two connected by the edge are represented by a column and a row, so that the element m_{ij} contains the pheromone value associated with the edge connecting nodes i and j.

Algorithm 4.3: High-level pseudo-code for Ant-Miner

```
   Data: Instances
   Result: RuleList
 1 RuleList ⟵ ∅
 2 while | Instances | > maximum_uncovered do
 3 |   PheromoneInitialization()
 4 |   Rule_gb ⟵ null
 5 |   n = 0
 6 |   while n < colony_size AND !NotStagnation() do
 7 |   |   Rule ⟵ CreateRule()
 8 |   |   Prune(Rule)
 9 |   |   UpdatePheromone(Rule)
10 |   |   if Quality(Rule) > Quality(Rule_gb) then
11 |   |   |   Rule_gb ⟵ Rule
12 |   |   end
13 |   end
14 |   Instances ⟵ Instances - Covered(Rule_gb)
15 |   List ⟵ List ∪ Rule_gb
16 end
17 RuleList ⟵ List ∪ Rule_Default
18 return RuleList
```

4.5.1.2 Rule Construction

In each iteration of the ACO procedure an ant creates a single classification rule. The decision to add a term to the current rule is made stochastically, with those terms with a larger amount of pheromone and heuristic information having a higher probability of being chosen. Terms are repeatedly added to the rule until one of two stopping criteria has been reached: either all valid terms have already been added to the rule, or all remaining terms that could be added to the rule would cause the rule to cover less than the required number of instances in the training set. This number is a user defined parameter that can be optimised before the algorithm is executed.

Once a rule has been created and the ant has stopped adding terms to the rule, a local search operator is applied (line 3 of Algorithm 4.1). In Ant-Miner,

the pruner attempts to remove all irrelevant terms that were added to the rule due to the stochastic nature of the ACO procedure. This is achieved by iteratively removing each term from a rule and calculating the new pruned rule's quality. The change that gives the largest increase in quality is retained. This is repeated until no term can be removed without decreasing the quality of the pruned rule.

4.5.1.3 Pheromone Deposition and Evaporation

Ant-Miner deposits pheromone like all other graph-based ACO algorithms. The best rule in the current iteration is selected for pheromone update. Each term in the best rule then has pheromone deposited onto the node that represents the term. The amount of pheromone deposited is proportional to the quality of the rule that the term is associated with.

Once the additional pheromone has been deposited, the pheromone matrix is normalised by summing all the pheromone present in the matrix and dividing every pheromone value by this sum. This normalisation process provides the pheromone evaporation functionality, as additional pheromone has been added to the matrix in the previous step. Therefore, the terms that were not updated as part of the best rule will undergo a reduction in their normalised value, and those present in the best rule will have their normalised pheromone level increased (Parpinelli, Lopes and Freitas 2002).

4.5.2 Ant-Miner Extensions

Following on Ant-Miner's success, many extensions have been proposed in the literature (Martens, Baesens and Fawcett 2011): they involve different rule pruning mechanisms, pheromone update procedures, new rule quality measures and heuristic information calculations. There are a number of Ant-Miner extensions relevant to this work: AntMiner+ (Martens et al. 2007), *c*Ant-Miner (Otero, Freitas and Johnson 2008), and *c*Ant-Miner$_{\mathrm{PB}}$ (Otero, Freitas and Johnson 2013).

4.5.2.1 Ant-Miner+

AntMiner+ extends Ant-Miner in several aspects: (1) the complexity of the construction graph is reduced, in terms of the number of edges connecting vertices, by defining it as a direct acyclic graph (DAG); (2) it makes a distinction between nominal attributes with categorical and ordered values, where the latter are handled as ordinal attributes allowing the algorithm to create interval conditions; (3) the class value to be predicted and weight parameters used to control the influence of the pheromone and heuristic information are incorporated in the construction graph as vertices.

AntMiner+ also uses the $\mathcal{MAX} - \mathcal{MIN}$ pheromone model first used in $\mathcal{MM}$-AS by Stützle and Hoos (2000) as a modification of the original Ant System. In $\mathcal{MAX} - \mathcal{MIN}$, limits are assigned to the pheromone values found in the matrix, so that each node or edge has a minimum and maximum amount of pheromone. This still allows the algorithm to prefer components associated with higher pheromone values and forget bad choices, while preventing extreme pheromone values to either dominate or prevent the selection of components.

4.5.2.2 *c*Ant-Miner

The original Ant-Miner requires the pre-processing of continuous attributes. Each continuous attribute requires the generation of split points. Split points are used to transform continuous attributes into categorical attributes, which can then be represented as nodes on the construction graph. Discretisation methods attempt to find good values that enable the continuous values of an attribute to be split, or binned, into different groups. In classification, these groups are often formed from the training data with knowledge of the class distribution, in a pre-processing step. One drawback of this pre-processing approach is that we cannot know what the good splits will be for the rules ants create. Split points that are good for the whole training set may not be good for the subset of instances that a partial rule

has covered.

*c*Ant-Miner (where the initial "*c*" refers to continuous attributes) was introduced to remove the requirement of a pre-processing step for continuous attributes in Ant-Miner. Each continuous attribute is given a single node in the construction graph. When a continuous node is selected by an ant, a dynamic discretisation step is performed to generate split-points that are appropriate for the subset of instances being operated on by the current partial rule. This allows the generation of better splits for continuous attributes and showed a significant increase in the predictive accuracy when tested against the original Ant-Miner (Otero, Freitas and Johnson 2008).

4.5.2.3 *c*Ant-Miner$_{\text{PB}}$

One potential drawback of using a sequential covering to create a list of rules is that there is no guarantee that the best list of rules is created. Ant-Miner (and the majority of its extensions) perform a greedy search for the list of best rules, using an ACO procedure to search for the best rule given a set of examples, and it is highly dependent on the order that rules are created. Therefore, they are limited to creating the *list of best rules*, which does not necessarily corresponds to the *best list of rules*.

*c*Ant-Miner$_{\text{PB}}$ is an ACO classification algorithm that employs, what the authors refer to as, an improved sequential covering strategy to search for the best list of classification rules (Otero, Freitas and Johnson 2013). While Ant-Miner uses an ACO procedure to create individual rules in a one-at-a-time (sequential covering) fashion, *c*Ant-Miner$_{\text{PB}}$ employs an ACO procedure that creates a complete list of rules for each ant. Therefore, it can search and optimise the quality of a complete list of rules instead of individual rules—i.e., it is not concerned by the quality of the individual rules as long as the quality of the entire list of rules is improving. This is analogous to the Pittsburgh approach for rule learning

discussed in Section 5.4, and in the name of the algorithm (cAnt-Miner$_{\text{PB}}$) PB stands for "Pittsburgh approach".

cAnt-Miner$_{\text{PB}}$ works by modifying the construction graph with the addition of depth, each depth has a corresponding pheromone value. In this case, edges of the graph have multiple depths and the depth corresponds to the position of a rule in a rule list where the edge is used. Each ant constructs a complete rule list by traversing each depth of the graph sequentially, constructing a single rule before incrementing the depth. Critically, the quality of each rule is not used during the pheromone update, it is the quality of the entire list that is used. Therefore, every term in every rule used in the best rule list receives an increase to its pheromone value. These rules may not be the best individual rule at each depth, but belong to the best overall list of rules.

4.5.3 Ant-Miner$_{\text{MA}}$

Ant-Miner$_{\text{MA}}$ (where MA stands for "Mixed Attribute") is an archive-based ACO implementation introduced by Helal and Otero (2016). The motivation for introducing an archive into the ACO procedure, was to remove the use of dynamic-discretisation procedures when using continuous attributes, by incorporating their selection into the learning process. The high-level pseudo-code for Ant-Miner$_{\text{MA}}$ is shown in Algorithm 4.4. Ant-Miner$_{\text{MA}}$ is based on the original ACO$_{\mathbf{MV}}$ algorithm (Liao et al. 2014) and implements an archive in place of the traditional construction graph and pheromone matrix that existed in Ant-Miner and all the previous extensions discussed earlier. The choice of which attributes to select for a rule is performed in the same way as ACO$_{\mathbf{MV}}$ (line 9 of Algorithm 4.4), a description of attribute selection can be found in Section 4.4. Ant-Miner$_{\text{MA}}$ implements the iterative rule learning strategy outlined in Section 2.3.1 as the archive stores individual rules and the best rule learnt in each iteration (lines 6-20) is added to the partial `RuleList` that is under construction (line 21).

Algorithm 4.4: High-level pseudo-code for the archive-based ACO algorithm Ant-Miner$_{\text{MA}}$.

```
   Data: Instances
   Result: RuleList
 1 RuleList ⟵ {}
 2 Restarted ⟵ 0
 3 Iteration ⟵ 0
 4 while |Instances| > MaxUncovered do
 5 |   A ⟵ GenerateRandomRules()
 6 |   while Iteration < MaxIterations AND Restarted ≠ 1 do
 7 |   |   RuleSet ⟵ {}
 8 |   |   while i < ColonySize do
 9 |   |   |   R_i ⟵ Create New Rule
10 |   |   |   R_i ⟵ Prune(R_i)
11 |   |   |   RuleSet ⟵ RuleSet ∪ R_i
12 |   |   |   i ⟵ i + 1
13 |   |   end
14 |   |   A ⟵ UpdateArchive(RuleSet)
15 |   |   Iteration ⟵ Iteration + 1
16 |   |   if Stagnation() then
17 |   |   |   Restart(A)
18 |   |   |   Restarted ⟵ 1
19 |   |   end
20 |   end
21 |   RuleList ⟵ RuleList ∪ TopRule(A)
22 |   Instances ⟵ Instances - Covered(TopRule(A))
23 end
24 return RuleList
```

During the creation of each rule (lines 4-20), Ant-Miner-Reg$_{\text{MA}}$ starts by creating a solution archive containing random rules (line 5). Then each ant in the colony creates a rule using the sampling procedures discussed in Section 4.4 (line 9), this new rule is then pruned and added to the set of rules created during this iteration (lines 10-12). Once each ant has created a rule, the archive is updated by adding the set of rules produced during the iteration, sorting them by quality and retaining the top m rules (line 14). This process is repeated until the maximum number of iteration is performed or the algorithm restarts more than once (lines

6-20). The best rule created (the top rule in the archive) is then added to the list of rules under construction (line 21) and any instances covered by this rule removed from the training set (line 22) and the process is repeated until less than the maximum number of instances allowed remain.

Figure 4.4 illustrates a solution archive for Ant-Miner$_{\text{MA}}$ containing a number of rules, each rule represents an example of each allowed attribute type (categorical, ordinal and continuous). Each attribute has a flag to represent weather that attribute is activated in a rule or not, where an active attribute is one that is used by the rule when deciding if it covers an instance or not.

Active attributes have an operator followed by one or more values. For continuous attributes, the allowed operators are `Greater Than` ($>$), `Less Than or Equal` ($\leq$) and `In Range` (InR), the later operator requires two values to create an allowed range of values, e.g., $V_1 \leq att < V_2$. Ordinal attributes also have two operators namely `Less Than Or Equal` ($\leq$) and `Greater Than Or Equal` ($\geq$). Finally, categorical attributes have a single allowed operator which is `Equal To` ($=$) which matches a single value. In Figure 4.4, the column titled "$f(S)$ contains the quality of the rule and and "w" is the weight associated with the rule.

The substitution of the pheromone graph with a solution archive allows Ant-Miner-Reg$_{\text{MA}}$ to dispense with both a discretisation step that was required during pre-processing for Ant-Miner and a dynamic-discretisation stage within cAnt-Miner. In other words the removal of a separate discretisation step allows Ant-Miner-Reg$_{\text{MA}}$ to optimise continuous values during rule creation, unlike Ant-Miner when values were determined before run time; and unlike cAnt-Miner where, while values were generated at the point of use, the values were not stored in the pheromone mode; and therefore could not be optimised by the ACO algorithm.

	Continuous attribute (A_r)						Categorical attribute (A_c)					Ordinal attribute (A_o)							
	Flag	Op	Value1	Value2	...	...	Flag	Op	Value	...	...	Flag	Op	Value	...	...	f(S)	w
S_1	T	$>$	v_1	-	...	...	T	$=$	v_1	...	...	F	-	-	...	...	f(S_1)	w_1
S_2	T	InR	v_3	v_2	...	...	F	-	-	...	...	T	$\leq$	v_1	...	...	f(S_2)	w_2
S_3	F	-	-	-	...	...	T	$=$	v_4	...	...	T	$\geq$	v_3	...	...	f(S_3)	w_3
$\vdots$			$\ddots$						$\ddots$					$\ddots$			$\vdots$	$\vdots$
S_k	T	$\leq$	v_2	-	...	...	F	-	-	...	...	T	$\leq$	v_2	...	...	f(S_k)	w_k
	Continuous attributes						Categorical attributes					Ordinal attributes							

Figure 4.4: Structure of the archive found in Ant-Miner$_{\text{MA}}$. The three solutions (rules) in the archive show examples of the three different attribute types: continuous, categorical, and ordinal attributes.

Helal and Otero (2016) compared Ant-Miner$_{\text{MA}}$ to the existing traditional graph-based ACO algorithm *c*Ant-Miner. Both algorithms implement the iterative rule learning strategy discussed in Section 2.3.1. In computational experiments with thirty data-sets, Ant-Miner$_{\text{MA}}$ showed improved accuracy on seventeen data-sets. However, when comparing computational time, it was found that Ant-Miner$_{\text{MA}}$ outperformed *c*Ant-Miner in twenty seven of the thirty data-sets, showing a statistically significant reduction in computational time. This was attributed to the removal of the dynamic discretisation step, which was computational expensive on large data-sets as the procedure needed to scan the entire training set while generating the split points required for continuous attributes.

Chapter 5

Ant Colony Optimization for Regression

This chapter will introduce the first contributions in this thesis, a collection of Ant Colony Optimization-based algorithms for the regression task. These algorithms are based on several approaches. The first algorithm follows an Iterative Rule Learning (IRL) approach, where rules are sequentially learned and added to a list of rules to create a complete model. Subsequently, other learning paradigms were investigated: a Pittsburgh-inspired approach, which aims to construct a complete list of rules in a single iteration of the ACO procedure, and a Michigan-based algorithm, where a set of rules are constructed and then compete for inclusion into the current iterations rule list.

The first algorithm, Ant-Miner-Reg is an IRL algorithm first published at the 2015 Genetic and Evolutionary Computation Conference (GECCO 2015) (Brookhouse and Otero 2015).

5.1 Discovering Regression Rules

We focus on creating ordered lists of regression rules. Regression rules are similar to classification rules; in fact, the antecedent used in regression rules has the same structure that is found in Ant-Miner, an algorithm for discovering classification rules. The differences are found in the prediction: where classification rules predict a value from a predefined set of memorised values, regression rules are free to predict any real value.

Regression rules contain a list of logical conditions that, if satisfied, predict a dependent value in their consequent. The predicted value can be directly a numeric value or, in a more complex case, a value computed by a linear model (Holmes, Hall and Frank 1999). Similarly to classification rules, regression rules can be represented in an *IF-THEN* form, where the *IF* is regarded as the antecedent of the rule and contains logical conditions involving the predictor attributes, while the *THEN* is regarded as the consequent of the rule and contains the prediction. When combined as a list, regression rules provide a comprehensible prediction model.

The prediction that each rule makes is created by calculating the mean value of any instances that are covered by the rule in the training data-set during rule creation. When rules are combined into an ordered list, the first rule in the list that is satisfied by the test instance makes the prediction for the list.

5.2 Ant-Miner-Reg: An Iterative Rule Learning ACO Regression Algorithm

ACO-based algorithms using the Iterative Rule Learning (IRL) paradigm have been previously introduced for the classification task in the Ant-Miner algorithm

(Parpinelli, Lopes and Freitas 2002) and many of its extensions, as reviewed in Sections 4.5.1 and 4.5.2. In this section we will introduce Ant-Miner-Reg, a new IRL ACO-based algorithm for the regression task. Ant-Miner-Reguses the same rule construction methodology found in *c*Ant-Miner however, Ant-Miner-Regreplaces the rule quality metric and dynamic discretisation procedures with ones that are more appropriate for the regression task. Another difference is the absence of heuristic information during rule construction, the motivation for this is discussed later in Section 5.2.5.1.

Ant-Miner-Reg uses the iterative IRL approach to building a complete solution, which in this case is a list of rules. The high-level pseudo-code for a generic IRL procedure can be seen in Algorithm 2.1 (Section 2.3.1). Rule creation is done in the function $LearnOneRule(Instances)$ (line 3 of Algorithm 2.1). In Ant-Miner-Reg, this procedure is replaced with an ACO-based rule learner. In each iteration of the IRL procedure, $LearnOneRule(Instances)$ tries to create the best rule to cover a sub-set of the uncovered instances. It can therefore be considered as an optimisation problem, which, in Ant-Miner-Reg is addressed by an ACO procedure. By reducing the task from the creation of a complete rule list into the creation of a single rule at each iteration, the complexity of the problem is reduced, as the optimisation function in $LearnOneRule(Instances)$ only has to find the best single rule for the current uncovered instance set. As rules are created, the uncovered instance-set is reduced, simplifying the optimisation problem with each iteration.

Algorithm 5.1 shows the high-level pseudo-code for Ant-Miner-Reg, where the IRL's $LearnOneRule(Instances)$ function is replaced with an ACO-based rule learner (lines 3 - 18). In each iteration of the ACO procedure, each ant in the colony creates a single rule by traversing the construction graph. A pruning function is then used to generalise the newly constructed rule (line 9). The pruner keeps removing the final term of a rule (considering the order in which the terms

were added) until the quality of the rule does not increase, at which point it stops and returns the pruned rule. If the rule created by an ant is the best-so-far in the iteration, it is retained; otherwise it is discarded (lines 7-13).

After all the ants have each constructed a single rule, the pheromone matrix is updated. Terms in the best rule in the iteration have their pheromone levels increased and all other terms have their pheromone levels decreased via normalisation using the process introduced in Ant-Miner (Parpinelli, Lopes and Freitas 2002). After all iterations of the ACO algorithm have been performed, any instances covered by the best rule produced are removed from the training set and the new rule added to the partial rule list (lines 19 and 20). This continues until the number of instances remaining uncovered is less than the allowed number. Finally, a default rule is added and the list returned. The default rule is simply a rule which has no antecedent, therefore, it will always be satisfied and always make a prediction. When placed as the final rule in a list, it ensures that the rule list will always make a prediction for any instance it is given. The prediction made by the default rule is calculated as the mean of the values of the target variable among the remaining uncovered instances in the training set.

5.2.1 Rule Quality

The rule quality in Ant-Miner-Reg is measured by the combination of two factors: Relative Root Mean Squared Error (RRMSE) and relative coverage, as suggested by Janssen et al. for use in their SeCoReg algorithm (Janssen and Fürnkranz 2010a). These two measures are combined to give a rule quality Q, given by:

$$Q = \alpha \cdot (1 - RRMSE) + (1 - \alpha) \cdot relCov \tag{33}$$

where α sets the relative importance of RRMSE and relative coverage. An α value of 1 will only take into account the prediction error and a value of 0 will

Algorithm 5.1: High-level pseudo-code for Ant-Miner-Reg.

```
Data: Instances
Result: RuleList
RuleList ⟵ ∅
while | Instances | > maximum_uncovered do
    PheromoneInitialization()
    Rule_gb ⟵ null
    for n = 0 to ant_iterations do
        Rule_ib ⟵ null
        for j = 0 to colony_size do
            Rule ⟵ CreateRule()
            Prune(Rule)
            if Quality(Rule) > Quality(Rule_ib) then
                Rule_ib ⟵ Rule
            end
        end
        UpdatePheromone(Rule_ib)
        if Quality(Rule_ib) > Quality(Rule_gb) then
            Rule_gb ⟵ Rule_ib
        end
    end
    Instances ⟵ Instances - Covered(Rule_gb)
    List ⟵ List ∪ Rule_gb
end
RuleList ⟵ List ∪ Rule_Default
return RuleList
```

only consider rule coverage.

RRMSE is defined as:

$$RRMSE = \frac{RMSE}{\sqrt{\frac{1}{m}Default}} \tag{34}$$

where $RMSE$ is the root mean square error and $Default$ is a normalising factor that will approximately bound the RRMSE between 0 and 1. Both are defined

as:

$$RMSE = \sqrt{\frac{1}{m} \cdot \sum_{i=1}^{m} (y_i - \bar{y_i})^2} \tag{35}$$

$$default = \sum_{i=1}^{m} (y_i - y')^2 \tag{36}$$

where m is the total number of instances in the data-set, y is the value of the current instance, $\bar{y}$ is the predicted value of the current instance and, finally, y' is the mean over all instances. RRSME attempts to normalise the RMSE between 0 and 1, however it is still possible to achieve values above 1 when the predicted values are worse than predicting the mean.

The second rule quality measure is relative coverage, given by;

$$relCov = \frac{1}{m} \cdot coverage(Rule) \tag{37}$$

where m is the number of instances in the data-set and the function $coverage(Rule)$ evaluates the number of instances covered by the rule. Relative coverage is a normalisation of a rule's coverage, where 1 means a rule covers all instances in a data-set and 0 indicated it covers no instances.

At this point it is also worth mentioning how the quality of a rule list is measured. List quality ignores relative coverage and instead uses RRMSE as its only measure. A list makes predictions using all the rules that it currently contains and the RRMSE between the prediction and the true value of each instance is calculated.

5.2.2 Construction Graph

In order to create a rule, the search space is represented as a graph, where each node corresponds to a tuple containing an attribute, operator and value. An example of a construction graph is shown in Figure 5.1.

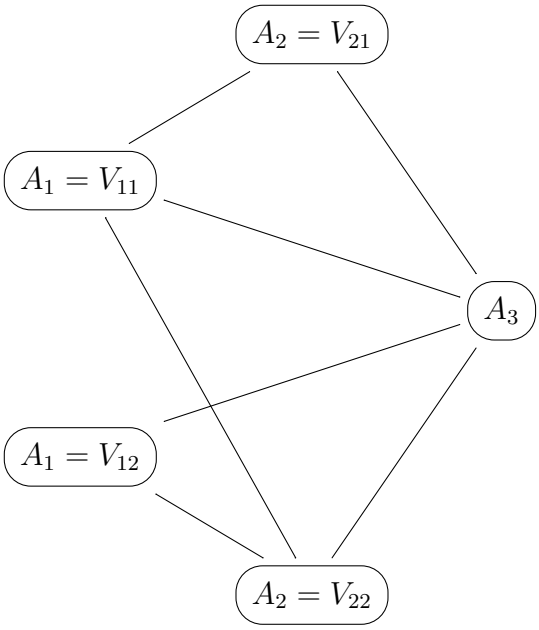


Figure 5.1: Example construction graph that can be used to construct rules, including both categorical attribute (A_1 and A_2) and a continuous attribute (A_3).

For nominal attributes, a node is created for each attribute value pair to represent the condition where the attribute has a specific value. In our construction graph example, we can see that attributes A_1 and A_2 are nominal and each have two allowed values (V_{11}, V_{12}) and (V_{21}, V_{22}), respectively.

The third attribute A_3 in Figure 5.1 has no associated value and is represented by a single node. This attribute is a continuous attribute. The values associated with this attribute are calculated at run-time using a dynamic discretisation method —as discussed in Section 5.2.4. It is important to note that the values generated and used in any created regression rules are not stored in the construction graph and are recalculated each time a continuous attribute node is selected. When nodes are selected in the same order, the same values will be generated since the discretisation procedure is deterministic.

Each node is then connected with every other node associated with a different attribute and an initial amount of pheromone is deposited on each edge during

initialisation. In Ant-Miner-Reg this initial amount is simply $\frac{1}{n}$ where n is the number of edges in the construction graph. Therefore, all edges have the same initial pheromone value.

5.2.3 Pheromone Update

At the end of each iteration of the colony, the pheromone associated with each edge of the graph is updated. This is a two-stage process, where the pheromone on the edges connecting terms (each term in a rule is represented by a node in the construction graph) used in good rules is increased, while unused terms see their edge's pheromone value decreased (line 14 of Algorithm 5.1). Pheromone update is performed in a similar way to *c*Ant-Miner (Otero, Freitas and Johnson 2008).

Pheromone increase is performed by taking the best rule that has been created in an iteration and increasing the pheromone for each edge leading to a term present in the best rule. The amount that the pheromone is increased by is dependent on the quality of the rule. The pheromone increase for each edge is given by:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \tau_{ij}(t) \cdot Q(t) \tag{38}$$

where $\tau_{ij}(t)$ is the current pheromone value for $edge_{ij}$ at iteration t and $\tau_{ij}(t+1)$ is the corresponding pheromone in the next iteration; $Q(t)$ is the quality of the best rule in iteration t. It can be seen that the increase in pheromone is proportional to the current pheromone value and the quality of the rule which the edge's term belongs to.

After all edges have been updated, pheromone evaporation is performed. This is achieved by normalising the amount of pheromone on the graph. While this has the effect of decreasing the pheromone values of all edges, those that have just received a pheromone increase will retain an increased amount of pheromone

compared to the previous iteration. Normalisation is performed by summing all the pheromone values on the edges originating from the same node and then dividing each pheromone value by this sum. This is repeated for every node on the construction graph.

5.2.4 Continuous Attribute Processing

Two different dynamic discretisation methods have been used for generating the split points for continuous attributes. The first method is the same one found in the SeCoReg algorithm and the second one is an adaptation of M5's standard deviation to the context of regression rules.

5.2.4.1 SeCoReg Split Point Generation

The SeCoReg method performs "supervised" clustering[1] of (attribute, target) pairs and produces n split points, where n is determined by the user. It achieves this by creating clusters that minimise the mean absolute errors of the target values in the following manner. First, it creates clusters containing a single pair (i.e., a cluster for each different attribute value) and the instances are then sorted by attribute value. Next, the method searches either side of each cluster looking for the merger that increases the error (where the error is defined as the mean absolute error (MAE)) by the minimum amount. This process is repeated until the number of clusters is reduced to $n + 1$.

The generated clusters have an upper and lower bound for the attribute values contained within. For each cluster, the mid point between its upper bound and the next's lower bound is calculated and these values are then returned as the n split points for an attribute. The split points are then combined with the $<$ (less than) and $\geq$ (greater than or equal to) operators. Finally, each term is

[1]Supervised clustering is the grouping of (attribute, target) pairs based on the similarity of their target values and not the similarity of the attribute values.

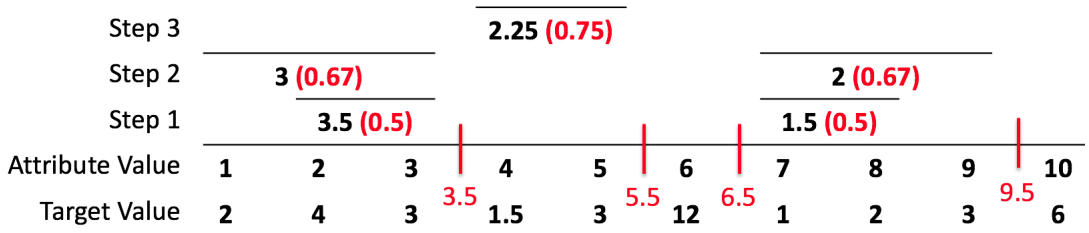


Figure 5.2: Example SeCoReg Split Point Generation.

temporarily added to the rule being constructed and the quality measured. The (operator, split point) pair that yields the highest quality —measured by equation 33— is chosen as the new term and added to the rule.

Split Point Generation Walkthrough

Figure 5.2 illustrates a fully worked example of split point generation using the SeCoReg Split Point generation method. In this example, the user has (hypothetically) asked for the generation of four split points. The algorithm needs to therefore create 5 clusters from the 10 pairs it has been given.

Initially, each (attribute, target) pair is assigned its own individual cluster. In Step 1 we can see that the smallest absolute error that can be achieved by merging values is 0.5. In this case, two clusters with this minimum error can be created. We therefore create the clusters `(2,3)` and `(7,8)` both with an absolute error of 0.5. The values 3.5 an 1.5 in step one represent the predictions made by the combined clusters.

We then compute the number of clusters that now remain, which is 8, so the process continues. In Step 2, we can see that merging the cluster `(1)` into the cluster `(1,2,3)` and `(9)` into the cluster `(7,8)` achieve the smallest increase in the absolute error, with both clusters now having an error of 0.67.

Again, we calculate the number of remaining clusters as 6, which is still more than our target of 5. The process is repeated again, creating a cluster of `(4,5)`

with an absolute error of 0.75 in Step 3. Recalculating the number of clusters, we find that we now have 5, allowing the generation of the 4 split points.

Split points are generated by finding the midpoint between the lower bound of one cluster and the upper bound of the next. In this case we would have the following 5 clusters with upper and lower bounds of: `(1,3.5)`, `(3.5,5.5)`, `(5.5,6.5)`, `(6.5,9.5)` and `(9.5,10)`.

5.2.4.2 Standard Deviation Split Point Generation

The second split point generation method is based on the M5 regression tree algorithm (Quinlan 1992). This method attempts to maximise the expected reduction in the error of the prediction of the target value in the current set of training instances. The expected error reduction is given by:

$$\Delta error = \sigma(T) - \sum_i \frac{|T_i|}{|T|} \cdot \sigma(T_i) \tag{39}$$

where T is the entire current set of instances, while T_i is the i-th subset and σ is the standard deviation of subset. To find the optimal split point p for a continuous attribute att_c, the current covered instances are scanned from beginning to end (in particular the entire instance set is scanned in the case that a continuous attribute is the first attribute selected to be added to the antecedent of a rule). Each split point generates two candidate subsets T_i: one subset containing the instances that satisfy the condition $att_c < p$ and another containing the instances that satisfy the condition $att_c \geq p$. Once the optimal p has been identified, the operator that is associated with the subset with the lowest standard error is the one used to create the rule term.

Note that the split point p is not stored in the construction graph (nor used to update pheromone values), since the split point generation is a deterministic procedure—the same split point p will be generated if att_c is selected using the same set of instances (Otero, Freitas and Johnson 2008).

5.2.5 Rule Creation

Each ant in the colony creates an individual rule at each iteration. The probability that an ant selects a node (term) is proportional to the amount of pheromone currently deposited on each edge. Therefore, edges that are used more often when constructing good rules have a higher probability of being selected. First, an ant starts at a 'virtual' start node and probabilistically selects the next node to travel to, then this node is added to the antecedent of the rule being constructed. The ant then selects the next node to add to the rule based on the pheromone value on the edges leading to the neighbouring nodes. Neighbouring nodes are those that correspond to terms using attributes not present in the rule antecedent, since once an attribute has been added to the rule, it cannot be revisited. If an ant selects a node that represents a continuous attribute, a split point generation method is executed —these methods are described in Section 5.2.4.

The ant will continue to add terms to the current rule until either the ant has added all attributes to its rule antecedent or the rule under construction covers less than the minimum number of instances (a user-defined parameter). Once an ant has finished its traversal of the construction graph, the rule is pruned to remove terms that are unnecessary.

5.2.5.1 Heuristic Information

Typically ACO-based algorithms use both pheromone and heuristic information to guide the traversal of the construction graph. This heuristic information can be static or dynamic. Static heuristic information is typically generated before algorithm execution and remains constant for the entire ACO execution. Dynamic heuristic information is generated as the ACO algorithm executes and is updated as the ant traverses the graph providing up to date information.

It is generally assumed that heuristic information is beneficial to the generation of good models, however experimental results comparing dynamic, static and no

heuristic information have shown that heuristic information does not improve performance and in fact static heuristic information can decrease model accuracy (private communication, Otero 2015). One hypothesis for this unexpected result, is that the colony quickly discovers this information and effectively encodes this into the pheromone model. While the static heuristic information can mislead the colony as it does not take into account term interaction. For these reasons Ant-Miner-Reg and its derivatives do not utilise any heuristic information during rule construction.

5.2.5.2 Rule Pruner

Ant-Miner-Reg will continue to add terms to a rule until there are no more terms to add to a rule or the number of instances covered by a rule drops below a user defined threshold. This leads to rules with many terms, which also specialise on few instances, leading to over-fitting. Therefore, a rule pruner is required to remove unnecessary terms and create more generic rules covering many more instances. Ant-Miner-Reg uses a simple backtrack pruner to achieve this (Otero, Freitas and Johnson 2009).

The backtrack pruner first calculates a rule's quality using Equation 33. It then removes the last term in the antecedent and calculates the modified quality. If this produces a rule of increased quality the change is kept. The pruner then repeats this process until no improvement in the rule's quality is observed.

5.3 Ant-Miner-Reg$_{\mathrm{M}}$: A Michigan-based ACO Regression Algorithm

The Michigan-based ACO algorithm creates a population of rules in each iteration from which it selects a subset of these rules to create a rule list in a niching operation. The niching operation allows rules to compete for their inclusion in

Algorithm 5.2: Michigan-based ACO high-level pseudo-code.

```
   Data: Training Instances
   Result: RuleList_gb
 1 RuleList_gb ⟵ ∅
 2 RuleList_cb ⟵ ∅
 3 PheromoneInitialization()
 4 for i = 1 to ant_iterations do
 5 |   RuleList_i ⟵ ∅
 6 |   for j = 1 to colony_size do
 7 |   |   Rule ⟵ CreateRule()
 8 |   |   Prune(Rule)
 9 |   |   RuleList_i ⟵ RuleList_i ∪ Rule
10 |   end
11 |   UpdatePheromone(RuleList_i)
12 |   RuleList_cb ⟵ Niche(RuleList_cb ∪ RuleList_i)
13 |   if Quality(RuleList_cb) > Quality(RuleList_gb) then
14 |   |   RuleList_gb ⟵ RuleList_cb
15 |   end
16 end
17 return RuleList_gb
```

the final rule list of the current iteration.

Algorithm 5.2 shows the high-level pseudo-code for the Michigan-based Ant-Miner-Reg$_{\mathrm{M}}$ ACO algorithm. Lines 1 and 2 initiate the global best rule list (RuleList$_{gb}$) and the current best rule list (RuleList$_{cb}$) with an empty list. Then in each iteration, each ant in the colony constructs a single rule with the $CreateRule()$ procedure. This procedure is the same $CreateRule()$ procedure found in the IRL Ant-Miner-Reg algorithm; rule creation is described in Section 5.2.5. This creates a list of rules for this iteration (lines 6-10), and this list is stored in the list $RuleList_i$.

The generation of each rule is performed on the entire training data-set, previously generated rules do not lead to the removal of any instances during the rule construction process. Therefore, each ant in every iteration is able to cover any of the instances in the training data-set, setting apart the Michigan-based algorithm

from the IRL algorithm.

Rules that have a quality above a pre-determined threshold (a user-defined parameter) have the pheromone of their terms increased. Like Ant-Miner-Reg, evaporation occurs via the normalisation of the pheromone values. The previous best rules (stored in RuleList_{cb}) and the rules produced by the current iteration are combined and a niching operation is then performed on this list to produce a new list of rules (line 12). If this new list is better than the global best (RuleList_{gb}), it becomes the new global best rule list (lines 13-15). While the ACO procedure is still responsible for creating individual rules, it is the niching procedure that is responsible for the creation and evaluation of the complete rule list.

5.3.1 Niching for the Michigan Approach

Once a list of rules has been produced at each iteration, a niching operation is performed. This operation uses the rules from the current best rule list and the rules produced in the current iteration. The niching operation is based on the classification niching operation in GBAP (Olmo, Romero and Ventura 2010) an Ant Programming algorithm that focuses on the construction of programs from an allowed grammar and then allows these programs to compete against each other. The rules are first sorted and then forced to compete for instances by claiming them before other rules. Rules with a higher quality are given the opportunity to cover instances first, and therefore claim them before lower quality rules.

Algorithm 5.3 presents the high-level pseudo-code for the niching operation used in Ant-Miner-Reg_{M}. First, rules are ordered by their quality measured on the full training data set (line 2), before iterating over all rules and instances in the training data looking for an instance that the rule matches (line 6). If a match is found, the instance is removed from the training data and the rule prediction is updated to reflect the newly covered instance (lines 7-9). This is to say that after all instances have been claimed by a rule in the list, the rule will cover a different

Algorithm 5.3: High-level Ant-Miner-Reg$_M$ niching pseudo-code

```
   Data: RuleList, Instances
   Result: RuleList_best
 1 RuleList_best ⟵ null
 2 Sort(RuleList)
 3 for Rule in RuleList do
 4 |   NumCovered ⟵ 0
 5 |   for Instance in Instances do
 6 |   |   if Covers(Rule, Instance) then
 7 |   |   |   UpdatePrediction(Rule,Instance)
 8 |   |   |   Instances = Instances - Instance
 9 |   |   |   NumCovered++
10 |   |   end
11 |   end
12 |   if NumCovered > 0 then
13 |   |   RuleList_best ⟵ RuleList_best ∪ Rule
14 |   end
15 end
16 return RuleList_best
```

set of instances than before it had to compete for an instance. Therefore, the prediction of a rule is updated so that it is now the mean of all the instances that it now covers when used as part of a rule list and not independently as a single rule.

After scanning the training data, if a rule is found to cover at least one instance it is added to the new list of rules, which is then returned at the end of the niching operation. Rules that do not cover a single instance are now discarded. These discarded rules will include poor quality rules, e.g. over-fitting rules whose job was performed better by a simpler (more generic) rule.

5.3.1.1 Michigan Niching Walkthrough

In this section an example of the niching algorithm will be shown using the set of rules shown in Table 5.2 on the sample data given in Table 5.1, which is an extended data-set of the housing data shown in Chapter 2.

Table 5.1: Sample data-set for niching a list of rules.

Instance	Target Attribute	Predictor Attributes		
Number	Rental Value	Floor Area	Location	Garage
1	£300	45	2	No
2	£600	80	1	Yes
3	£250	33	3	No
4	£400	65	2	Yes
5	£350	54	2	Yes
6	£550	120	2	No
7	£150	54	3	Yes
8	£200	54	3	No
9	£400	60	1	No

The first step of the niching algorithm requires the rules (Table 5.2) to be sorted by their RRMSE. This results in the rule ordering of `(3,1,6,5,2,4)`. Next, the best rule (`Rule 3`) is allowed to claim as many instances as it can cover from the data-set (Table 5.1), namely instances `(3,7,8)`. These instances are then removed from the data-set and `Rule 3` is added to our partial rule list as the top rule.

Now the second best rule is selected, `Rule 1`, it can still cover all the instances

Table 5.2: Rule set to be niched using the data in Table 5.1

Number	Rule	RRMSE	Instances
1	`Floor Area>55 THEN 487.5`	0.627	4
2	`Floor Area>55 AND Garage=Yes THEN 500`	0.703	2
3	`Location=3 THEN 200`	0.287	3
4	`Floor Area>45 THEN 378.5`	1.0766	7
5	`Location=2 THEN 400`	0.657	4
6	`Floor Area>55 THEN 487.5`	0.627	4

Table 5.3: Final niched Rule List for the sample house rental data-set in Table 5.1.

Number	Rule	Instances
3	`Location=3 THEN 200`	3
1	`Floor Area>55 THEN 487.5`	4
5	`Location=2 THEN 325`	2
-	`THEN 355.5`	0

that it covered in the full training set, as there was no overlap between the instances it covers and those covered by `Rule 3`. Therefore, it claims instances `(2,4,6,9)` and is also added to the partial rule list under construction, and then the covered instances in the training set are removed.

The next rule in the list (`Rule 6`) attempts to cover as many instances as it can. However, `Rule 6` is a duplicate of `Rule 1`. This means that there are no available instances to cover. When a rule fails to cover a single instance it is discarded, i.e., not included in the final rule list.

Next, `Rule 5` is evaluated, which originally covered instances `(1,4,5,6)`. However Rule `1` has already covered instances `4` and `6`. The rule claims instances `1` and `5` and recalculates its prediction based on the instances that it has covered. The modified Rule `5` is then added to the list.

With the evaluation of `Rule 5`, all instances have been covered by a rule. Since `Rule 2` and `Rule 4` are unable to cover any instances, they are discarded. Table 5.3 shows the final rule list that has been created by the niching algorithm and the number of instances that each rule now covers.

Note that Table 5.3 contains an additional rule: `{ THEN 355.5}` . This is the default rule that will always make a prediction as the antecedent is empty, and will therefore always be satisfied. In a typical rule induction algorithm, the value of the prediction for the default rule is calculated based on all remaining uncovered instances in the training set, after all the previous rules have claimed

any instances they can. However, in this case there are no uncovered instances, so we calculate the prediction based on the mean of all instances in the training set. The final rule list (after niching) is considered the rule list for the current iteration of Algorithm 5.2.

5.4 Ant-Miner-Reg$_{\text{PB}}$: A Pittsburgh-Based ACO Algorithm

The Pittsburgh-based ACO regression algorithm (Ant-Miner-Reg$_{\text{PB}}$) uses a similar strategy from *c*Ant-Miner$_{\text{PB}}$ (Otero, Freitas and Johnson 2013). Recall that (as discussed in Section 5.4) the key differences between an IRL and a Pittsburgh-based ACO algorithm is that the latter creates a complete rule list in a single ACO iteration, rather than creating a single rule in each ACO iteration. This is achieved by moving of the *While* loop (line 2 of Algorithm 5.1) that checks the number of uncovered instances to inside the ACO model construction phase, and the addition of depth to the pheromone matrix, where the depth corresponds to the position of a rule in the rule list. This allows different term interactions (i.e., co-occurrence of terms) at different positions in the list, as an interaction that is useful at the top of a list may not be as useful lower down.

The creation of a complete rule list in each iteration allows for the optimisation of the interactions of both rules and terms when creating the best rule list for a specific data set, where two rules constructed together may outperform two rules constructed separately using a greedy approach. In a greedy sequential covering approach, such as Ant-Miner-Reg, the first rule attempts to cover as many instances as possible with no consideration to subsequent rules. The high-level pseudo-code for Ant-Miner-Reg$_{\text{PB}}$ is shown in Algorithm 5.4.

Lines 14 and 19 of Algorithm 5.4 show that the quality measure used in the ACO optimisation procedure is not that of a single rule but of an entire rule

Algorithm 5.4: Pittsburgh-based ACO Regression Algorithm high-level pseudo-code.

```
   Data: Instances
   Result: RuleList_gb
 1 RuleList_gb ⟵ null
 2 PheromoneInitialization()
 3 for i = 1 to ant_iterations do
 4 |   RuleList_ib ⟵ null
 5 |   for j = 1 to colony_size do
 6 |   |   Instances ⟵ Training Instances
 7 |   |   RuleList ⟵ null
 8 |   |   while /Instances/ > maximum_uncovered do
 9 |   |   |   Rule ⟵ CreateRule()
10 |   |   |   Prune(Rule)
11 |   |   |   Instances ⟵ Instances − Covered(Rule)
12 |   |   |   RuleList ⟵ RuleList ∪ Rule
13 |   |   end
14 |   |   if Quality(RuleList) > Quality(RuleList_ib) then
15 |   |   |   RuleList_ib ⟵ RuleList
16 |   |   end
17 |   end
18 |   UpdatePheromone(RuleList_ib)
19 |   if Quality(RuleList_ib) > Quality(RuleList_gb) then
20 |   |   RuleList_gb ⟵ RuleList_ib
21 |   end
22 end
23 return RuleList_gb
```

list, in contrast to the IRL-based Ant-Miner-Reg Algorithm 5.1. In Ant-Miner-Reg$_{\text{PB}}$, we use RRMSE as a measure of list quality which we aim to minimise. It should be mentioned that the procedure to create a single rule and the pruning of each rule (lines 9 and 10 of Algorithm 5.4) are the same ones that are found in Ant-Miner-Reg.

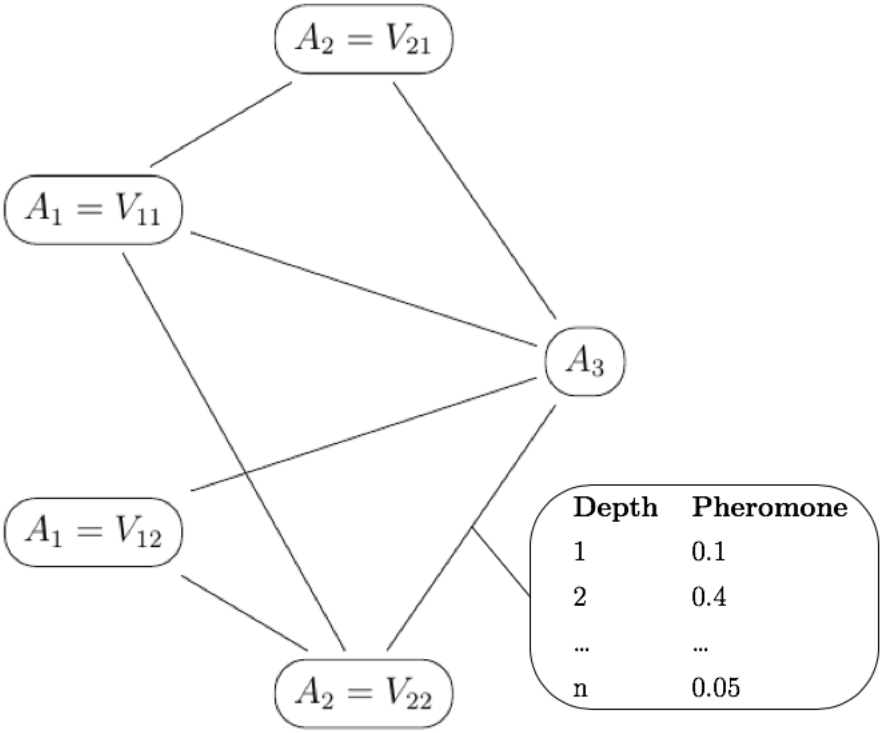


Figure 5.3: An illustration of a construction graph for the Pittsburgh-based algorithm Ant-Miner-Reg$_{\text{PB}}$. Each edge contains a table of rule depth and pheromone values to be used in the list construction.

5.4.1 Extended ACO Construction Graph

Ant-Miner-Reg$_{\text{PB}}$ still uses a construction graph, which the artificial ants traverse when constructing their rules. The pheromone amount is then updated with the best rule list created in each iteration —denoted RuleList$_{ib}$ in Algorithm 5.4. The overall structure is the same as in Ant-Miner-Reg. However, each edge on the construction graph now contains the concept of depth, this depth can be seen in Figure 5.3. Each node in the construction graph shown in Figure 5.3 represents a term that could be added to a rule in the same way as in Figure 5.1, the differences between the two construction graphs are to be found on the graph's edges, where a new concept of depth is found.

Each depth on the construction graph represents a rule position in a rule list.

When an ant wants to create the first rule in the list, then the pheromone values it uses comes from depth of one, while the n-th rule pheromone values are sampled from depth n.

The concept of depth allows the colony to compartmentalise its knowledge of the search space for each rule into separate levels, as terms that are good for rules near the top of the list may not be good for rules lower down in the list.

If an ant creates a rule at a depth that has not been used by any previous ants in any of the previous iterations, a new level is created and initial pheromone is deposited onto each edge as described in Section 5.2.2.

When updating the pheromone values the best rule list generated in an iteration is used and not the best single rule generated for each depth of the construction graph. This ensures that the algorithm optimises the entire rule list and not each rule individually, allowing for rule and term interactions to be optimised by the algorithm. This is because an individual rule does not have to cover as many instances as possible to improve its relative coverage and hence its overall quality. Rules can leave an instance uncovered for a lower rule to cover and make a more accurate prediction, improving the rule list quality and by extension its own quality.

Chapter 6

Computational Results for ACO Regression

In this chapter the computational results for ACO algorithms tackling the regression problem will be presented. The results have been divided into two sections. The first section presents the results of the dynamic discretisation procedure used by Ant-Miner-Reg, while the second section presents a comparison of the three learning strategies to create a list of regression rules in ACO-based algorithms.

6.1 Experimental Setup

The Ant-Miner-Reg algorithms presented in this chapter have been compared using the fifteen UCI Machine Learning Repository data-sets (Lichman 2013) shown in Table 6.1 to evaluate the most effective strategy for the regression task. All experiments used 10-fold cross-validation. Recall that cross-validation involves dividing the data-set into ten partitions; a model is created using nine of these partitions as training data, while holding the tenth partition as an unseen testing set. This is repeated ten times using each of the partitions as a testing set. The stochastic ACO algorithms are ran five times (varying the random seed) on each

Table 6.1: Attribute makeup of the UCI Machine Learning Repository data sets for the regression task used in the experiments (Lichman 2013).

		Predictor Attributes		
Data Set	Instances	Real	Nominal	Total
Air-foil	1502	5	0	5
CCPP	9568	0	5	5
Concrete	1029	0	8	8
CPU	209	1	8	9
Efficiency	767	0	8	8
Elevators	9516	0	6	6
Flare	1065	10	1	11
Housing	452	1	13	14
Istanbul Stock	535	0	7	7
MPG	392	3	5	8
R_WPBC	193	0	32	32
Red Wine	1599	0	12	12
Skill Craft	3337	0	19	19
Stock	949	0	9	9
Yacht	308	0	7	7

cross-validation fold, for a total of fifty runs on each data-set. Multiple runs are performed with varying random seeds to ensure that any variability in algorithm performance due to the probabilistic nature of an algorithm is reduced. While 10-fold cross-validation is used to ensure that any benefit (or penalty) due to variability in partition makeup between training and testing data is minimised.

Table 6.2 contains the user-defined parameters of the ACO-based algorithms (Ant-Miner-Reg, Ant-Miner-Reg$_{\mathrm{M}}$ and Ant-Miner-Reg$_{\mathrm{PB}}$), including the additional parameter required by the Michigan-based algorithms for the pheromone update threshold. The values for these parameters have not been optimised.

Table 6.2: Parameters used in the ACO-based algorithms.

Parameter	Value
Minimum Covered Rule	10
Minimum Uncovered	10
Iterations	500
Colony Size	10
Quality Weighting (α)	0.59
Ant-Miner-Reg$_{\mathrm{M}}$	
Pheromone Update Threshold	0.5

The values for the first four parameters mentioned are based on the parameters used in the original *c*Ant-Miner (Otero, Freitas and Johnson 2009) and Ant-Miner (Parpinelli, Lopes and Freitas 2002). Whilst the values for the last two parameters are based on other algorithms as mentioned later.

The parameters `Minimum Covered Rule` and `Minimum Uncovered` represent stopping criteria for rule construction and rule list construction respectively, setting the minimum number of instances a rule must cover and the maximum uncovered examples in the current training set before list construction can end. `Iterations` and `Colony Size` manage the ACO procedure, setting the size of the colony (which is the number of solutions produced in each iteration) along with the maximum number of iterations. The last parameter for all ACO algorithms is the `Quality Weighting`, which sets the balance between RRMSE and relative coverage. The value used here is the same as the one used in SeCoReg (Janssen and Fürnkranz 2010b). Finally, there is an additional parameter for the Michigan-based ACO algorithm Ant-Miner-Reg$_{\mathrm{M}}$, namely `Pheromone Update Threshold`, which sets the quality threshold for a rule to be used during the pheromone update procedure. The value used is the same as the one used in GBAP (Olmo, Romero and Ventura 2010).

For the regression task, the Relative Root Mean Squared Error (RRMSE) is used to compare the predictive performance of different algorithms. It is calculated by first applying the constructed model to the data-set and the RRMSE between the prediction $\bar{y_i}$ and the true value y_i is calculated as:

$$RRMSE = \frac{\sqrt{\frac{1}{m} \cdot \sum_{i=1}^{m} (y_i - \bar{y_i})^2}}{\sqrt{\frac{1}{m} \sum_{i=1}^{m} (y_i - y')^2}} \tag{40}$$

where y' is the mean value of the target attribute in the data-set and m is the number of instances in the data set. This produces a normalised value, where 0 represents no error and 1 is the error produced if the mean is predicted for all instances. Therefore, the closer the value to 0, the better the algorithm's performance

6.2 Dynamic Discretisation Experiments

Ant-Miner-Reg requires a dynamic dicretisation procedure to create good split points when coping with continuous attributes. Two strategies for dynamic discretisation have been implemented. The first is the method used by M5 (Quinlan 1992), which aims to partition the data in such a way that it maximises the expected reduction in error by minimising the standard deviation of the target variable in the two sub-sets. The second split point generation method uses a group merging process that attempts to merge groups of instances that result in the minimum increase in group error in relation to the target value.

Table 6.3 shows the RRMSE of the two Ant-Miner-Reg variants. Ant-Miner-Reg+ERSP uses the error reduction split point generation, while Ant-Miner-Reg+ GMSP uses the group merging split point procedure. Each algorithm was tested on the fifteen data-sets shown in Table 6.1 using the experimental setup outlined in Section 6.1.

Table 6.3: RRMSE of the rule list produced by each of the algorithms on each of the fifteen UCI Machine Learning repository data-sets. A bold value signifies the smallest error produced by either algorithm. The standard deviation is shown in brackets.

Data set	Ant-Miner-Reg+ERSP	Ant-Miner-Reg+GMSP
Air-foil	**0.5512 [0.0138]**	0.7869 [0.0109]
CCPP	**0.3484 [0.0007]**	0.3592 [0.0005]
Concrete	**0.4182 [0.0046]**	0.4987 [0.0106]
CPU	0.5624 [0.0450]	**0.4469 [0.0289]**
Efficiency	**0.2038 [0.0006]**	0.2044 [0.0028]
Elevators	**0.6345 [0.0221]**	0.7587 [0.0563]
Flare	**1.0017 [0.0012]**	1.0027 [0.0007]
Housing	0.5547 [0.0354]	**0.4873 [0.0090]**
Istanbul	**0.8563 [0.0246]**	0.9287 [0.0175]
R_WPBC	**1.13477 [0.0494]**	1.3265 [0.0375]
MPG	0.5432 [0.0149]	**0.5322 [0.0083]**
Red wine	**0.9048 [0.0216]**	0.9161 [0.0114]
Skill Craft	**0.8219 [0.0209]**	0.8324 [0.0267]
Stock	**0.2457 [0.0106]**	0.2540 [0.0081]
Yacht	0.5273 [0.0015]	**0.3576 [0.0050]**

From Table 6.3 we can see that the error reduction split point procedure performed better in eleven of the fifteen data-sets when compared to the group merging procedure. Statistical testing was performed and the results of the Wilcoxon Signed-Rank test can be seen in Table 6.4.

6.2.1 Discussion

The GMSP procedure provided the algorithm with a number of potential split point options; allowing a number of different rules to be evaluated compared to

Table 6.4: Wilcoxon Signed-Rank test (at the $\alpha = 0.05$ level) on RRMSE, for the results in Table 6.3.

	W+	W-	Z	p
RRMSE	38	82	-1.2495	0.2113

the ERSP procedure, which produces a single split point. However, this did not make the GMSP procedure more successful at identifying the best split point. This may be due to the ACO procedure generating many candidate rules containing continuous attributes during each iteration of the IRL strategy, causing the split point procedures to be executed many times. GMSP generates many split points during each execution of the discretisation procedure compared to ERSP, however does not give the algorithm an advantage, we can hypothesise that ERSP produces a single split point of generally higher quality.

The comparison of the two discretisation procedures shows that the error reduction split point procedure was the most successful, outperforming the group merging procedure found in SeCoReg (Janssen and Fürnkranz 2010b,a) in eleven of the fifteen data-sets. While this change was not significant (p value of 0.2113), it is deemed successful enough and show enough promise to be used as the default split point generation procedure in future Ant-Miner-Reg-derived algorithms.

6.3 Learning Strategy Experiments

Three different learning strategies for Ant-Miner-Reg were investigated. This produced three algorithms different ACO-based algorithms:

(1) Ant-Miner-Reg, the original ACO algorithm for the regression task that uses the IRL strategy;

(2) Ant-Miner-Reg$_{\mathrm{M}}$, an ACO algorithm that uses the Michigan learning strategy;

(3) Ant-Miner-Reg$_{\mathrm{PB}}$, which uses the Pittsburgh-based strategy.

Table 6.5: RRMSE of the four algorithms being tested on the 15 UCI Machine Learning Repository (Lichman 2013) regression data sets averaged across all 10 cross-validation folds. The standard deviation is shown in brackets. The ACO-based algorithms were ran 5 times (varying the random seed) on each fold to reduce stochastic effects. The best result on each data set is shown in bold.

Data Set	Ant-Miner-Reg	Ant-Miner-Reg$_{M}$	Ant-Miner-Reg$_{PB}$	M5' Rules
Air-foil	0.9438 [0.0065]	0.9453 [0.0095]	0.6147 [0.0241]	**0.4797 [0.0358]**
CCPP	**0.2275 [0.0109]**	0.2746 [0.0136]	0.2385 [0.0090]	0.2375 [0.0136]
Concrete	0.9285 [0.0568]	0.9151 [0.0579]	0.4887 [0.0377]	**0.3850 [0.1024]**
CPU	0.6779 [0.2447]	1.2514 [0.3384]	0.3694 [0.1302]	**0.1707 [0.1438]**
Efficiency	0.1839 [0.0290]	0.1940 [0.0474]	**0.0721 [0.0087]**	0.1036 [0.0067]
Elevators	0.6213 [0.0337]	0.6271 [0.0351]	**0.5410 [0.0136]**	0.6014 [0.0134]
Flare	0.9974 [0.0135]	1.0105 [0.0151]	**0.9603 [0.1105]**	1.0086 [0.0257]
Housing	0.9569 [0.0160]	0.9418 [0.0315]	**0.3744 [0.1291]**	0.4396 [0.1154]
Istanbul Stocks	0.6958 [0.0490]	0.7377 [0.0464]	0.6354 [0.0767]	**0.4811 [0.0564]**
MPG	0.4120 [0.0822]	0.4203 [0.0606]	**0.2713 [0.0578]**	0.3723 [0.0455]
R_WPBC	1.0452 [0.0738]	1.0559 [0.0849]	**1.0348 [0.1588]**	1.0555 [0.2687]
Red Wine	0.9519 [0.0174]	0.9576 [0.0121]	**0.7800 [0.0436]**	0.8068 [0.0354]
Skill Craft	0.7943 [0.0221]	0.9028 [0.0376]	**0.7408 [0.0395]**	0.7628 [0.0897]
Stock	0.5526 [0.1562]	0.5706 [0.1229]	**0.1069 [0.0150]**	0.1441 [0.0676]
Yacht	1.0241 [0.0316]	1.0339 [0.0145]	**0.0694 [0.1344]**	0.0833 [0.0264]

Table 6.6: Average ranks of the four regression algorithms tested, based on the average RRMSE of the models produced. Statistically significant results using the Holm post-hoc test for the significance level $\alpha = 0.05$ are shown in bold.

Algorithm	Average Rank	p	Holm
Ant-Miner-Reg$_{\text{PB}}$	1.4000	-	-
M5' Rules	1.8667	0.3222	0.05
Ant-Miner-Reg	**2.8667**	$\mathbf{1.8628 \times 10^{-3}}$	**0.025**
Ant-Miner-Reg$_{\text{M}}$	**3.8667**	$\mathbf{1.6715 \times 10^{-7}}$	**0.0167**

These three algorithms were compared to the classical M5' Rules regression algorithm, which also produces lists of rules, on the fifteen UCI Machine Learning repository regression data-sets (shown in Table 6.1) using the experimental setup discussed in Section 6.1.

The average RRMSE for each model produced by the four regression algorithms is shown in Table 6.5, where the value of the algorithm that achieved the smallest average error on each data set is shown in bold. The value found in the square brackets is the standard deviation of the RRMSE across all executions on each data-set.

When comparing the three ACO-based regression algorithms, Ant-Miner-Reg$_{\text{PB}}$ outperformed the other two algorithms in fourteen of the fifteen data sets. When we compare the best performing ACO algorithm Ant-Miner-Reg$_{PB}$ against M5' Rules we find that Ant-Miner-Reg$_{PB}$ wins in ten data sets and looses to M5' Rules in the other five. The IRL-based Ant-Miner-Reg wins in only one data-set.

Table 6.6 shows the results of the Friedman statistical test (Friedman 1937) of the four algorithms tested. This includes the average rank and the p-value for each algorithm. Statistical significance was tested at the 0.05 level using the Holm post-hoc test, which compares the best algorithm (Ant-Miner-Reg$_{\text{PB}}$) against each of the other three algorithms.

Table 6.7: Average number of rules in the final rule list of the four algorithms being tested on the 15 regression data sets averaged across all 10 cross-validation folds. The standard deviation is shown between brackets. The smallest rule list for each data-set is shown in bold.

Data Set	Ant-Miner-Reg	Ant-Miner-Reg$_M$	Ant-Miner-Reg$_{PB}$	M5' Rules
Air-foil	13 [0.6207]	**6 [0.5771]**	28 [2.2165]	22 [1.8754]
CCPP	130 [25.1768]	**4 [0.5406]**	126 [15.1274]	14 [1.2856]
Concrete	4.9 [0.7783]	**4 [0.7827]**	18 [2.7486]	10 [0.5687]
CPU	6.4 [0.5253]	8 [1.1824]	12 [1.4514]	**5.4 [0.2318]**
Efficiency	7.3 [0.5869]	**5 [1.1911]**	10 [0.8401]	18 [1.4864]
Elevators	9.3 [0.7010]	**6 [0.7690]**	14 [1.8211]	**6 [0.4686]**
Flare	3.2 [0.5345]	6 [2.0259]	11 [1.1037]	**3 [0.6874]**
Housing	**3 [0.1414]**	5 [1.2149]	19 [1.9821]	7 [0.8694]
Istanbul Stocks	6 [1.4213]	6 [0.8571]	11 [1.7291]	**2.5 [1.5694]**
MPG	**4.5 [0.5436]**	8 [1.3739]	13 [1.5546]	6 [1.1965]
R_WPBC	4.3 [0.6263]	7 [1.1578]	11 [1.2103]	**3 [0.7854]**
Red Wine	10 [2.3296]	8 [0.9649]	33 [3.8322]	**4 [2.6574]**
Skill Craft	9.4 [1.1761]	8 [1.2289]	32 [3.7541]	**7 [4.6451]**
Stock	**5.4 [0.8609]**	6 [1.0882]	15 [2.2348]	8 [1.5728]
Yacht	7.4 [0.8514]	**6 [0.6465]**	14 [1.6309]	6.2 [1.0348]

Table 6.8: Average ranks of the four regression algorithms tested based on the average number of rules in the list of rules produced by each algorithm. Statistical significant results using the Holm post-hoc test for the significance level $\alpha = 0.05$ are shown in bold.

Algorithm	Average Rank	p	Holm
Ant-Miner-Reg$_M$	1.9333	-	-
M5' Rules	1.9667	0.9436	0.05
Ant-Miner-Reg	2.2333	0.8245	0.025
Ant-Miner-Reg$_{PB}$	**3.8667**	$\mathbf{4.1098 \times 10^{-5}}$	**0.0167**

It can be seen that Ant-Miner-Reg$_{\mathrm{PB}}$ achieves the lowest average rank and statistically outperforms the other two ACO based algorithms. However, the difference in mean ranks between Ant-Miner-Reg$_{\mathrm{PB}}$ and M5' Rules is not statistically significant.

The number of rules produced in the list of rules was also investigated —the average number of rules for each algorithm is found in Table 6.7, with the smallest average number of rules shown in bold. It was found that Ant-Miner-Reg$_{\mathrm{PB}}$ produced the largest number of rules in thirteen of the data-sets. Ant-Miner-Reg$_{\mathrm{M}}$ produced the smallest list of rules in six of the data-sets, best result (in terms of minimising rule list size) for all the ACO-based algorithms. Ant-Miner-Reg produced the smallest list in three of the data-sets. M5' Rules produced the smallest average rule list size in the largest number of data-sets, with seven data-sets.

A Friedman statistical test was performed on the results for rule list size, this can be seen in Table 6.8. It can be seen that the algorithm with the best average rank was Ant-Miner-Reg$_{\mathrm{M}}$. Using the Holm post-hoc test to compare the average rank of Ant-Miner-Reg$_{\mathrm{M}}$ against the average rank of the other three algorithms, the only significant result was against Ant-Miner-Reg$_{\mathrm{PB}}$, which produced the largest rule lists on average.

6.3.1 Discussion

First we will discuss the three different ACO-based approaches and then discuss the performance of the best approach against the classical regression rule learner M5' Rules.

The results show that, when the goal is to minimise the RRMSE (as a measure of predictive performance), the most effective search strategy is the Pittsburgh-based algorithm: Ant-Miner-Reg$_{\mathrm{PB}}$ statistically outperforms the other two ACO-based algorithms in the regression task. In addition, the Michigan-based approach showed no improvements when compared to the original Ant-Miner-Reg, this was surprising as both Michigan and Pittsburgh-style approaches have been shown to be competitive for data-mining applications using Learning Classifier System (LCS) algorithms (Bacardit and Butz 2007). We hypothesis that while the Michigan-based constructs the rules together and then performs a niching operation allowing for a limited form of rule interaction, the multiple rules being used for pheromone update spread too much new pheromone over the graph preventing the ants from converging on a solution during rule construction. This is due to many more terms having their pheromone increased, which will spread out the pheromone, preventing the convergence towards a single path.

Another hypothesis as to why Ant-Miner-Reg$_{\mathrm{M}}$ performed poorly could be the repeated construction by the colony of rules that cover the same region, any repeated rules are ignored and the work performed by the ant that discovered the discarded rule is wasted and the effort not used to explore other regions of the search space to produce rules that cover different instances.

Ant-Miner-Reg$_{\mathrm{PB}}$ allows ants to consider term and rule interactions while traversing the pheromone graph. This is achieved by modifying the pheromone model by adding a concept of depth, separating out the pheromone being deposited by each rule, unlike the multiple rule update mechanism found in the Michigan-based ACO algorithm. This difference allows each depth of the pheromone

graph, with each depth corresponding to an individual rule. We hypothesis that this allows the colony to successfully converge on a single (optimal or near optimal) path through the graph. In the IRL approach the algorithm's aim is to create a list of best rules as at each stage the IRL procedure generates the best rule it can based on the set of instances that are currently uncovered. By contrast, the Pittsburgh-based algorithm's objective is to create the best list of rules. In the classification task it was also found that using the Pittsburgh-based algorithm produced more accurate models than the IRL strategy (Otero, Freitas and Johnson 2013).

When comparing the size of the regression models produced by the ACO algorithms, Ant-Miner-Reg$_{\text{PB}}$ produces the largest lists of rules on average. An increase in the size of the rules is to be expected as the objective during construction has changed. In the IRL strategy, the objective is to produce the best rule at each stage, with no regard to subsequent rules. As rule coverage contributes to a rule's quality, each rule will attempt to cover as many instances as possible, which tends to produce fewer rules in the list. Similarly in the Michigan-based strategy, the niching algorithm gives the best quality rules the chance to cover as many of the instances as possible, reducing the size of the list as subsequent good quality rules are unable to claim any instances. However, in the Pittsburgh-based strategy, the model's quality is determined by the best list of rules, with less pressure on the rules to cover as many instances as possible. This allows the generation of more specialist rules (with smaller coverage) to provide better predictions. This increase in model size was also found when comparing the Pittsburgh-based strategy to the IRL-strategy in the classification task (Otero, Freitas and Johnson 2013). While the the size of a model is important, with smaller models being better, the predictive accuracy of a model is often more important.

6.3.1.1 Comparison against M5' Rules

When comparing the best ACO-based rule learner against a classical regression rule learning approach, we find that Ant-Miner-Reg$_{\text{PB}}$ is very competitive against the well-known algorithm M5' Rules, achieving a lower RRMSE in ten data-sets out of the fifteen (Table 6.1). Additionally, the models produced by Ant-Miner-Reg$_{\text{PB}}$ will be generally more comprehensible than those of M5' Rules Freitas (2014). This is due to the rule predictions made by Ant-Miner-Reg$_{\text{PB}}$ being easily understandable, as they are made by a single value (the average value of the target variable over all the covered instances in the training data). M5' Rules uses linear models that, while decreasing the RRMSE, add complexity to the rule's consequent, allowing them to predict different values fo different instances satisfying the rule's antecedent. Users therefore tend to have more difficulty in understanding the final prediction, since predictions are not only determined by rule antecedent, but also the linear model. Finally, linear models allow the prediction of values outside the known domain of an attribute, which may produce unexpected results.

M5' Rules produces rule lists similar in size to the original Ant-Miner-Reg algorithm, while also being the second best performing algorithm in terms of RRMSE. The addition of linear models to a rule's prediction gives individual rules greater power as they are not restricted to predicting a single mean value for all instances satisfying the rule's antecedent; like Ant-Miner-Reg and its derivatives. This allows a single rule to perform the function of a number of simpler single-valued rules at the cost of comprehensibility.

In summary, three new ACO-based regression rule algorithms have been introduced and compared to each other and to a well established classical regression rule algorithm. The Pittsburgh-based Ant-Miner-Reg$_{\text{PB}}$ has achieved the best average rank of all four algorithms producing the best models (in terms of RRMSE) in ten of the fifteen data-sets and statistically outperforming the other two ACO-based algorithms.

Chapter 7

Incorporating Monotonic Constraints

Monotonic constraints encapsulate existing domain knowledge as either increasing or decreasing correlations between predictor attributes and the target attribute. This encapsulated domain knowledge can then be used during the learning phase to influence the model that is being created, ensuring that it does not contradict the existing knowledge.

The enforcement of monotonic constraints in this thesis is performed in two different ways. First, a use of soft monotonic constraints in the learning (rule construction) phase, where the quality of a rule incorporates a notion of how monotonic the rule is. The quality is used to influence the amount of pheromone on the construction graph, which in turn influences ants in future iterations, guiding them towards monotonic regions of the search space but not overly constricting the search.

The second enforcement of constraints is done in a post-processing phase in the form of hard constraint pruners. These pruners are used to ensure that the final rule list produced by the algorithm conforms to the constraints imposed upon it. The result of these pruners do not influence the learning phase of the algorithm,

since they are applied after all rules have been created. Additionally, they operate on a complete model (list of rules), instead of individual rules, as monotonicity is a global property of the model, not a property of individual rules.

This chapter is based on a number of published papers, with the algorithms Ant-Miner-Reg$_{\text{MC}}$ and cAnt-Miner$_{\text{PB+MC}}$ along with the soft constraint implementation being published in GECCO 2016 (Brookhouse and Otero 2016b) for the former regression algorithm and ANTS 2016 (Brookhouse and Otero 2016a) for the latter classification algorithm. The hard pruning suite was first introduced at WCCI 2018 in a special IJCNN session on "Ordinal and Monotonic Classification" along with the algorithm cAnt-Miner$_{\text{PB+MCP}}$ (Brookhouse and Otero 2018).

7.1 Soft Monotonic Constraint Enforcement

In both data mining tasks investigated in this thesis, namely regression and classification rule discovery, soft constraints can be incorporated into the learning phase when rules are constructed.

Previously in Chapter 5, the algorithms Ant-Miner-Reg and Ant-Miner-Reg$_{\text{PB}}$ were introduced to create regression rules. These algorithms, along with cAnt-Miner$_{\text{PB}}$, are modified in this chapter to incorporate soft constraints. The soft constraint algorithms work as follow. Each ant starts with an empty list of rules and iteratively adds a new rule to this list. In order to create a rule, an ant adds one term at a time to the rule antecedent by choosing terms to be added to the current partial rule based on the amount of pheromone (τ) and a problem-dependent heuristic information (η). Once a rule is created, it undergoes a pruning procedure. Pruning aims at removing irrelevant terms that might be added to a rule due to the stochastic nature of the construction process: it starts by removing the last term that was added to the rule and the removal process is repeated until the rule quality decreases when the last term is removed or the rule has only one

term left. Finally, the created rule is added to current list of rules and the training examples covered by the rule are removed.

At the end of an iteration, when all ants have created a solution —for Ant-Miner-Reg this is a single rule and for Ant-Miner-Reg$_{\text{PB}}$ each ant creates a list of rules— the best solution (determined by an error-based quality function) is used to update pheromone values, providing a positive feedback on the terms present in the solution. The higher the pheromone value of a term, the more likely it will be chosen to create subsequent solutions. This iterative process is repeated until a maximum number of iterations is reached or until the search stagnates.

Recall that the rule construction phase of Ant-Miner-Reg contained a quality function. When incorporating soft constraints, this quality function is modified to include a monotonic correctness measure along with the original quality measure used. This revised quality function is also used in the soft rule pruner, to show a preference for monotonic solutions without absolutely requiring it. The quality function is given by equations 41 and 42. The soft monotonic constraint pruner that uses a modified quality function to prune the rules during the construction phase will be discussed in detail in Section 7.1.1.

By softly enforcing monotonic constraints within the ACO procedure in the rule creation stage, optimisation of monotonicity should reduce the need to rely on the more aggressive and potentially more damaging hard pruners later on. At the same time, soft constraints allow the algorithm to explore non-monotonic regions of the search space that might lead to higher predictive power, and may also allow the discovery of new monotonic regions with good predictive power.

7.1.1 Soft Enforcement and Rule Quality

With the addition of monotonic constraints, a new notion of rule quality that incorporates both predictive accuracy and monotonic compliance is required. This is achieved through the addition of a measure of non-monotonicity and combined

with a weighted measure as follows.

Given a list of rules, the non-monotonicity index (NMI) is defined as:

$$NMI = \frac{\sum_{i=1}^{k} \sum_{j=1}^{k} m_{ij}}{k^2 - k} \ , \tag{41}$$

where m_{ij} is 1 if the pair of rules $rule_i$ and $rule_j$ violate the constraint and 0 otherwise; k is the number of rules in the model. The NMI of a model is constrained between zero and one: representing the ratio of monotonicity violating pairs over the total possible number of prediction pairs present in the model being tested —the lower a NMI is, the better a model is considered. The NMI is then incorporated into the quality metric by:

$$Q_{Mod} = (1 - \omega) \cdot Quality + \omega \cdot (1 - NMI) \ , \tag{42}$$

where Q_{Mod} is the modified quality of a model (rule list) and ω is an adjustable weighting that sets the importance of monotonicity and the original non-monotonic quality measure to the overall rule list quality. Note that Equation 42 can be used to calculate the quality of either a single rule or a complete list of rules. When used to calculate the quality of the i-th rule the NMI calculated in Equation 41 is replaced by Equation 43.

Soft enforcement is also present in the rule pruner. This local search operator allows violations of the monotonic constraints if the consequent's improvement in accuracy is large enough. The pruner operates on an individual rule and iteratively removes the last term until no improvement in the rule quality is observed. Applying soft enforcement during model creation allows the search to be guided towards monotonic models while still allowing exploration of the entire search space by the colony.

As monotonicity is a global property of the model and a rule in isolation can not violate itself, the rule being pruned is temporarily added to the current partial

list of rules. Its non-monotonicity index (NMI) can then be used as a metric to assess the rule's monotonicity, calculated using a slightly modified NMI metric given by:

$$NMI = \frac{\sum_{j=1}^{k} m_{ij}}{k-1} \ , \tag{43}$$

where Rule_i is the current rule being tested; m_{ij} is 1 if Rule_j causes a violation with Rule_i and 0 otherwise, and k is the number of rules in the list. When testing an individual rule, there are $k-1$ possible interactions instead of the larger k^2-k interactions found when calculating the NMI of a complete list of rules.

7.2 Hard Monotonic Constraint Enforcement

Monotonicity is a global property of a model and in rule models, at least two rules are required to create a violation. Therefore, pruners that operate on a complete rule list are preferential to those operating on individual rules, since the latter can only modify a single rule to fix the violations present in the model (Feelders 2000). Initially, a simple monotonic backtrack pruner, now referred to as the Naive Pruner (NP) was used to strictly enforce monotonic constraints by fixing any violations that were not removed by the soft constraint enforcement. The Naive Pruner can be very destructive if the violating rule occurs towards the top of a rule list. This is because the bottom section of the list is discarded due to the backtracking nature of the pruner, as it will keep removing the last rules and terms until it encounters the monotonic violation and creates a monotonic rule list.

7.2.1 Naive Pruner

The hard monotonic Naive Pruner enforces the monotonic constraints rigidly as it does not allow any violations to exist within the rule list. It operates on a list of

Algorithm 7.1: High-level pseudo-code for the Naive Pruner, where a term is removed from the list until the NMI is zero.

```
Data: non − monotonic list
Result: monotonic list
while NMI(list) > 0 do
    PruneLastTerm(list)
    if LastRuleLength(list) = 0 then
        RemoveLastRule(list)
    end
end
return list
```

rules as follows: (1) the NMI of a list is first calculated (Equation 41); (2) if it is non-zero, the last term of the final rule is removed or, if the rule contains only one term, the rule is removed; (3) the NMI is then recalculated for the modified list of rules. This is repeated until the NMI of the rule list is zero, this will always be achieved as (if necessary) eventually a single rule will be present in the rule list and two rules are required for a violation to occur. Finally, the default rule is added to the end of the list if it has been removed and the new monotonic rule list is returned. The high-level pseudo-code for the Naive Pruner is shown in Algorithm 7.1. The pruner is known as the Naive Pruner as it is a very simple pruner that, unlike the pruners introduced later, does not use any monotonic information when deciding how the rule list should be pruned.

7.2.2 Most Violations Pruner (MVP)

The Most Violations Pruner (MVP) prunes the worst rule in terms of NMI in the list of rules. The rationale is that by pruning the worst rules first, the list of rules can be made monotonic with fewer changes. It works as follows: (1) the NMI of each rule is calculated (Equation 43); (2) the rule with the highest NMI has its last term removed or the complete rule is removed if it has only one term; (3) the NMI of each rule is then recalculated and the procedure continues until the

Algorithm 7.2: High-level pseudo-code for the Most Violations Pruner. In each iteration the rule with the worst NMI has its last term removed.

```
Data: non − monotonic list
Result: monotonic list
while NMI(list) > 0 do
    rule_worst ⟵ {}
    for n ⟵ 1 to list_size do
        if NMI(rule_n) ≥ NMI(rule_worst) then
            rule_worst ⟵ rule_n;
        end
    end
    PruneFinalTerm(rule_worst)
    if RuleLength(rule_worst) == 0 then
        RemoveRule(list, rule_worst)
    end
end
return list
```

model's NMI is zero. In the case of a draw, e.g., a single pair of rules violating each other, the rule appearing lower in the list is preferentially pruned. This decision is made as rules towards the top of the list were generated based on a larger training set and therefore likely to be more accurate than those towards the end, which are classifying fewer remaining instances. The high-level pseudo-code for the MVP pruner is shown in Algorithm 7.2.

7.2.3 Best Fix Pruner (BFP)

The third global pruner, called Best Fix Pruner (BFP), attempts to fix the rule that would give the greatest reduction in the model's NMI. Note that the rule that gives the greatest reduction is not necessarily the rule with the highest NMI. The high-level pseudo-code for the best fix pruner is shown in Algorithm 7.3. Each non-monotonic rule in the complete model is pruned backwards from the last term until a change in the model's NMI is detected. The pruned rule that led to the largest decrease in NMI is kept and the remaining rules are restored to their

Algorithm 7.3: High-level pseudo-code for the Best Fix Pruner. In each iteration the rule that decreases the NMI by the largest amount will be pruned.

```
   Data: non − monotonic list
   Result: monotonic list
 1 while NMI(list) > 0 do
 2 |   rule_bi ⟵ {}
 3 |   best_improvement ⟵ 0
 4 |   for n ⟵ 1 to list_size do
 5 |   |   rule_n ⟵ list[n]
 6 |   |   rule_prune ⟵ PruneRule(rule_n)
 7 |   |   improvement ⟵ NMI(list) − NMI(list_prune)
 8 |   |   if improvement ≥ best_improvement then
 9 |   |   |   rule_bi ⟵ rule_n
10 |   |   |   best_improvement ⟵ improvement
11 |   |   end
12 |   end
13 |   rule_bi ⟵ PruneRule(rule_bi)
14 end
15 return list
```

original state. This process is repeated until the list becomes monotonic. For the same reasons explained in the MVP approach, draws are solved by pruning the rule lower in the list.

7.2.4 Monotonic Pruning Walk-through

In this section we will use a classification example[1] to demonstrate how the different pruners would create different rule lists, showing how the wrong pruner could easily disrupt the predictive power of a model.

The introduction of soft monotonic constraints during the learning phase of an ACO algorithm should produce a rule list that is mostly monotonic when compared to a rule list generated by a monotonically unaware algorithm. This should reduce the need for additional aggressive pruning as a post processing step.

[1] pruning a regression list would be similar with only the rules' consequents differing

However, as the soft constraints do not guarantee a monotonic list hard constraint pruners are still required.

An initial approach to generate a monotonic list is to prune the non-monotonic list until it becomes monotonic. This is the approach represented by the Naive Pruner. Given its naive nature, this pruning can be quite destructive. Two new pruners have been created with the aim to reduce the potential destructive effects of pruning. Consider the following example using a car efficiency data set, with a constraint that more powerful cars will have a lower fuel efficiency. An execution of *c*Ant-Miner$_{\text{PB}}$ could possibly produce the list of rules below:

```
1) IF Power ≥ 250 THEN High
2) IF Cylinders = 6 AND Power ≤ 200 THEN Low
3) IF Doors = 2 THEN Medium
4) IF Power ≤ 200 THEN High
5) IF <empty> THEN Medium
```

We can see that there is a monotonic violation between rules 1 and 2, as a car with a lower power can have a worse fuel efficiency (rule 2) than one with more power (rule 1). To ensure monotonicity, the Naive Pruner would keep removing the last term in the rule list, checking the monotonicity of the list at each step. When the pruner removes the term '`Power ≤ 200`' it produces the following monotonic rule list in the process (the default rule being automatically re-added to ensure full coverage):

```
1) IF Power ≥ 250 THEN High
2) IF Cylinders = 6 THEN Low
3) IF <empty> THEN Medium
```

The more sophisticated monotonic pruners MVP and BFP would identify that a single violation exists between the first and second rules, at which point they

would begin removing the terms at the end of the lowest ranked rule until the violation is fixed. Whenever multiple rules have the same NMI as is the case with MVP; or would provide the same reduction in NMI as with BFP, the lowest rule in the list will be chosen. This is because rules higher in a rule list are likely to make more accurate predictions than rules lower in the list, therefore, pruning lower rules will reduce any damage caused by the pruners. Both pruners would then produce the following monotonic rule list:

1) IF Power $\geq$ 250 THEN High

2) IF Cylinders = 6 THEN Low

3) IF Doors = 2 THEN Medium

4) IF Power $\leq$ 200 THEN High

5) IF <empty> THEN Medium

As can be seen this is a far less destructive change than the Naive Pruner as the change created by the new pruners is significantly smaller. This should lead to smaller change in the rule list's predictive accuracy compared to the large changes caused by the Naive Pruner.

The three pruners are computationally inexpensive in comparison to the main ACO optimisation procedure so a viable approach to rule list pruning is to use all three hard pruners and select the rule list that has the highest predictive accuracy on the training set after the pruning phase. At this point we can focus only on accuracy since all rule lists at this stage are guaranteed to be monotonic.

7.3 Monotonic Constraint ACO Algorithms

In this section five new monotonically aware ACO-inspired algorithms will be introduced, combining the proposed soft and hard monotonicity constraint enforcement for both classification and regression tasks. All five algorithms incorporate soft monotonic constraints during the learning phase and then implement a hard

Algorithm 7.4: High-level pseudo-code for Ant-Miner-Reg$_{\text{MC}}$. Changes from the base algorithm (which ignores monotonicity constraints) are highlighted in yellow.

```
   Data: Training instances
   Result: List of Rules
 1 List ⟵ ∅
 2 while | Instances | > maximum_uncovered do
 3 |   PheromoneInitialization()
 4 |   Rule_gb ⟵ null
 5 |   for n = 0 to ant_iterations do
 6 |   |   Rule_ib ⟵ null
 7 |   |   for j = 0 to colony_size do
 8 |   |   |   Rule ⟵ CreateRule()
 9 |   |   |   Prune(Rule)
10 |   |   |   if ModifiedQuality(Rule) > ModifiedQuality(Rule_ib) then
11 |   |   |   |   Rule_ib ⟵ Rule
12 |   |   |   end
13 |   |   end
14 |   |   UpdatePheromone(Rule_ib)
15 |   |   if ModifiedQuality(Rule_ib) > ModifiedQuality(Rule_gb) then
16 |   |   |   Rule_gb ⟵ Rule_ib
17 |   |   end
18 |   end
19 |   Instances ⟵ Instances - Covered(Rule_gb)
20 |   List ⟵ List ∪ Rule_gb
21 end
22 List ⟵ List ∪ Rule_Default
23 NaiveMonotonicPruner(List)
24 return List
```

monotonic post-processing phase. The first three algorithms (Ant-Miner-Reg$_{\text{MC}}$ Ant-Miner-Reg$_{\text{PB+MC}}$ and *c*Ant-Miner$_{\text{PB+MC}}$) use the simple Naive Pruner during this post-processing phase. The other two algorithms, Ant-Miner-Reg$_{\text{PB+MCP}}$ and *c*Ant-Miner$_{\text{PB+MCP}}$, have an expanded pruning suite to strictly enforce constraints.

7.3.1 Ant-Miner-Reg$_{\text{MC}}$

The first proposed algorithm is Ant-Miner-Reg$_{\text{MC}}$. It is an extension of the IRL-based ACO algorithm Ant-Miner-Reg (Section 5.2). The high-level pseudo-code for Ant-Miner-Reg$_{\text{MC}}$ is shown in Algorithm 7.4, where the high-lighted lines show the changes made to the algorithm to enforce the monotonic constraints, in comparison to the original Ant-Miner-Reg algorithm.

Lines 9 and 10 of Algorithm 7.4 contain the modified rule quality function. This function now combines three measures. While the original Ant-Miner-Reg rule quality measure uses RRMSE and relative coverage, these have been augmented with NMI and now form a single weighted quality measure given by:

$$Q_{Mod} = \alpha \cdot RRMSE + \beta \cdot relative_coverage + \omega \cdot (1 - NMI) \ , \qquad (44)$$

where the importance of each measure can be adjusted. Line 9 has a rule pruner that uses the modified monotonic quality measure when making term removal decisions. The final modification is the addition of the Naive Pruner on line 23. This simple backtrack pruner will iteratively remove the last term in the rule list until the model is completely monotonic.

7.3.2 Ant-Miner-Reg$_{\text{PB+MC}}$ and cAnt-Miner$_{\text{PB+MC}}$

For both classification and regression rule discovery, the introduction of a Pittsburgh-based approach has been shown to increase the quality of the rule lists produced by allowing the global optimisation of the complete list. As monotonicity is a global property of a model, a Pittsburgh approach will be able to better optimise the monotonic constraints during the soft constraint learning phase. Two Pittsburgh-based algorithms have been created: Ant-Miner-Reg$_{\text{PB+MC}}$ for regression rule discovery and cAnt-Miner$_{\text{PB+MC}}$ for the classification task. The high-level pseudo-code for both Pittsburgh-based monotonic ACO algorithms is shown

Algorithm 7.5: High-level pseudo-code for Ant-Miner-Reg$_{\text{PB+MC}}$ and cAnt-Miner$_{\text{PB+MC}}$. Changes from the base algorithm (which ignores monotonicity constraints) are highlighted in yellow.

```
   Data: training instances
   Result: list of rules
1  RuleList_gb ⟵ null
2  PheromoneInitialization()
3  for i = 1 to ant_iterations do
4      RuleList_lb ⟵ null
5      for j = 1 to colony_size do
6          Instances ⟵ Training Instances
7          RuleList ⟵ null
8          while |Instances| > maximum_uncovered do
9              Rule ⟵ CreateRule()
10             Prune(Rule)
11             Instances ⟵ Instances − Covered(Rule)
12             RuleList ⟵ RuleList ∪ Rule
13         end
14         if ModifiedQuality(RuleList) > ModifiedQuality(RuleList_ib) then
15             RuleList_ib ⟵ RuleList
16         end
17     end
18     UpdatePheromone(RuleList_ib)
19     if ModifiedQuality(RuleList_ib) > ModifiedQuality(RuleList_gb) then
20         RuleList_gb ⟵ RuleList_ib
21     end
22 end
23 NaiveMonotonicPruner(RuleList_gb)
24 return RuleList_gb
```

in Algorithm 7.5.

In order to incorporate monotonic constraints, the Pittsburgh-based approach has been changed as follows. On lines 14 and 19 there are modified rule list quality functions, which combine the original list quality measure with the NMI measure. For the regression rule discovery algorithm Ant-Miner-Reg$_{\text{PB+MC}}$, this modified list quality measure has two components: the RRMSE of the list and the NMI of the rule list, which are then combined to give a single value using a

weighted sum where the weights of the two measures can be tuned by the user. For the classification task, cAnt-Miner$_{\text{PB+MC}}$ also uses two measures: a pessimistic accuracy measure based on the C4.5's error estimation (Quinlan 1993) and the NMI measure, which again is consolidated into a single value with a weighted sum that can be tuned by the user to give a greater preference to either accuracy or monotonicity.

Like the previously discussed Ant-Miner-Reg$_{\text{MC}}$, the two Pittsburgh-based algorithms have a soft monotonic pruner (line 10) that uses the modified quality function; and hard enforcement of constraints is achieved via the Naive Pruner in a post-processing phase (line 23).

7.3.3 Ant-Miner-Reg$_{\text{PB+MCP}}$ and cAnt-Miner$_{\text{PB+MCP}}$

The algorithms introduced previously in this chapter use a simple naive pruner to strictly enforce monotonic constraints. As discussed in section 7.2, depending on the location of the violation this simple back track pruner can make significant changes to the rule list with no regard to the list's quality. To counteract the effect of large changes during hard constraint pruning, two new pruners were proposed, Most Violations Pruner (MVP) and Best Fix Pruner (BFP). Instead of selecting a single pruning technique to use, the three list pruners work in conjunction to increase the accuracy of the model returned by cAnt-Miner$_{\text{PB+MC}}$ and Ant-Miner-Reg$_{\text{PB+MC}}$: all three pruners are applied in turn to the constructed list of rules and the pruner that achieves the highest accuracy on the training set is used for the final prune. The algorithms with the new pruning suite are called Ant-Miner-Reg$_{\text{PB+MCP}}$ and cAnt-Miner$_{\text{PB+MCP}}$ for the regression and classification tasks, respectively. Algorithm 7.6 contains the high-level pseudo-code for the two algorithms with the monotonic changes highlighted in yellow.

The soft constraint enforcement from the previous Pittsburgh-based ACO algorithms are carried over to the new algorithms, as can be seen by lines 10, 14 and

Algorithm 7.6: High-level pseudo-code for Ant-Miner-Reg$_{\text{PB+MCP}}$ and *c*Ant-Miner$_{\text{PB+MCP}}$. Changes from base algorithm (which ignores monotonicity constraints) are highlighted in yellow.

```
   Data: training instances
   Result: list of rules
 1 RuleList_gb ⟵ null
 2 PheromoneInitialization()
 3 for i = 1 to ant_iterations do
 4 |   RuleList_lb ⟵ null
 5 |   for j = 1 to colony_size do
 6 |   |   Instances ⟵ Training Instances
 7 |   |   RuleList ⟵ null
 8 |   |   while |Instances| > maximum_uncovered do
 9 |   |   |   Rule ⟵ CreateRule()
10 |   |   |   Prune(Rule)
11 |   |   |   Instances ⟵ Instances − Covered(Rule)
12 |   |   |   RuleList ⟵ RuleList ∪ Rule
13 |   |   end
14 |   |   if ModifiedQuality(RuleList) > ModifiedQuality(RuleList_ib) then
15 |   |   |   RuleList_ib ⟵ RuleList
16 |   |   end
17 |   end
18 |   UpdatePheromone(RuleList_ib)
19 |   if ModifiedQuality(RuleList_ib) > ModifiedQuality(RuleList_gb) then
20 |   |   RuleList_gb ⟵ RuleList_ib
21 |   end
22 end
23 BestList ⟵ null
24 BestListQuality ⟵ 0
25 for MonotonicPruner in MonotonicPruningSuite do
26 |   PrunedRuleList ⟵ MonotonicPruner(RuleList_gb)
27 |   if Quality(PrunedRuleList) > BestListQuality then
28 |   |   BestListQuality ⟵ Quality(PrunedRuleList)
29 |   |   BestList ⟵ PrunedRuleList
30 |   end
31 end
32 return BestRuleList
```

19 of Algorithm 7.6, which contain the soft constraint pruner and the modified rule list quality function. The key changes in the algorithm can be seen on lines 23-31, with the implementation of the pruning suite.

First, the best rule list and best rule list quality variables are initialised (lines 23 and 24). The algorithm then iterates through the pruners in the pruning suite (line 25), pruning the best list that was created during the learning phase (line 26). If the pruned list is better on the training data than all the pruners tested so far, the pruned list is kept and the best list quality updated (lines 27-30). Once all of the pruners have been tested and the best pruned list obtained, the best list (the list with the highest accuracy on the training data) is returned by the algorithm.

7.4 Summary

Enforcing monotonic constraints at two different phases of the algorithm allows the production of better models, which still fully enforce the monotonic constraints. If a single constraint phase was proposed instead of a two-phase soft/hard constraint approach, either the ACO would be deprived of useful information of the domain it is operating in with the removal of soft constraints or it would face an overly constrained search space that could limit its search exploration to monotonic regions of the search space.

7.4.1 Proposed Monotonic Algorithm Variants

In this chapter, five new algorithms have been proposed which all incorporate both soft constraints during model construction and a hard constraint post-processing step. Three algorithms tackle the regression task, the first is the IRL algorithm Ant-Miner-Reg_{MC} that uses the Naive Pruner to strictly enforce constraints. Next,

Ant-Miner-Reg$_{\text{PB+MC}}$ used the same Naive Pruner but changed the learning strategy from IRL to a Pittsburgh-based strategy. Finally for the regression task a pruning suite was added to create the algorithm Ant-Miner-Reg$_{\text{PB+MCP}}$, which applies three rule pruners and selects the pruner producing the rule list with the highest accuracy on the training list.

The final two algorithms proposed in this chapter are classification algorithms. *c*Ant-Miner$_{\text{PB+MC}}$ and *c*Ant-Miner$_{\text{PB+MCP}}$ both use the Pittsburgh learning strategy and differ in the hard constraint pruning strategies. The former uses the Naive Pruner while *c*Ant-Miner$_{\text{PB+MCP}}$ uses the pruning suite.

Chapter 8

Computational Results for Problems with Monotonic Constraints

In this chapter I will present the results for the proposed ACO-based monotonic algorithms in both regression (Ant-Miner-Reg$_{\text{MC}}$, Ant-Miner-Reg$_{\text{PB+MC}}$ and Ant-Miner-Reg$_{\text{PB+MCP}}$) and classification (cAnt-Miner$_{\text{PB+MC}}$ and cAnt-Miner$_{\text{PB+MCP}}$) tasks. The results have been divided into two sections. The first will present the results obtained for the regression task, looking at both the ACO learning paradigm and the effect of the additional monotonic pruning suite presented in Chapter 7. The second section will present the results for the classification task, starting with the cAnt-Miner$_{\text{PB+MC}}$ algorithm and then cAnt-Miner$_{\text{PB+MCP}}$, where the latter incorporates the monotonic pruning suite.

The results presented in this chapter have previously been published in a number of papers. For the regression task, these papers include (Brookhouse and Otero 2016b), and for the classification task they include (Brookhouse and Otero 2016a) for cAnt-Miner$_{\text{PB+MC}}$ and (Brookhouse and Otero 2018) for cAnt-Miner$_{\text{PB+MCP}}$.

Table 8.1: Parameter settings used for Ant-Miner-Reg$_{\text{MC}}$ derived algorithms.

Parameter	Value
Iterations	500
Colony Size	10
Minimum Covered Rule	10
Minimum Uncovered Rule List	0.1
Error Weighting (α)	0.4
Coverage Weighting (β)	0.3
Constraint Weighting (γ)	0.3

8.1 Results for Regression with Monotonic Constraints

In this section I will present the results for monotonic constraints in the regression task. The results will be divided into two subsections. The first (Section 8.1.1) presenting results obtained in the first set of experiments, combining soft constraints during the learning phase and a naive pruner that uses backtracking to ensure all constraints are enforced rigidly. The second (Section 8.1.2) will present the results obtained when the pruning suite was added to both ACO-based algorithms and traditional regression algorithms.

Table 8.1 shows the parameter settings used in all Ant-Miner-Reg$_{\text{MC}}$ derived algorithms: colony size, number of iterations, minimum coverage parameters, error weighting, and coverage weighting were derived from those used by Ant-Miner-Reg and *c*Ant-Miner$_{\text{PB}}$, so may not be the optimal settings on the test data-sets used here. Table 8.2 lists the data-sets selected from the UCI Machine Learning repository (Lichman 2013). The table gives information on the size of the data set and the number of continuous and nominal attributes. The constrained attribute is also shown along with the constraint direction, where it can be either

monotonically increasing or decreasing. The constrained attributes were chosen by calculating the NMI of all the attributes in a data-set. Attributes with an NMI of less than 0.1 were considered for constraint selection, the NMI for the selected attribute is also shown in Table 8.2. Once a list of low NMI attributes was created for each data-set, a constraint could be selected. Any potential attributes were checked to make sure they were consistent with common sense before being verified as a constraint. A single constraint was chosen rather than multiple constraints to minimise the risk of enforcing pattens that do not exist in the data and are actually the result of random noise during the testing of the proposed algorithms, future work would entail additional constraints once the robustness of the algorithms has been demonstrated on single constraints.

As in Chapter 6, ten-fold cross-validation is performed and the predictive accuracy (on the test set) from each of the ten folds is averaged. ACO-based algorithms are executed five times on each fold and again the average is taken. Five executions are performed with different random seeds due to the stochastic nature of ACO-based algorithms, since the multiple runs will mitigate against random increases and decreases in performance of the algorithm. Combining both cross validation folds and repeat executions means that the ACO-based algorithms will be executed fifty times on each data-set.

Table 8.2: Attribute make-up and constraint information of the eight monotonic regression UCI data sets used in the experiments (Lichman 2013). In each data set a single attribute was constrained. The attribute name, whether it is monotonically increasing (↑) or decreasing (↓) and the non-monotonicity index (NMI) of the attribute is given.

		Attributes		Constraint		
Name	Instances	Nominal	Continuous	Constrained Attribute	Direction	NMI
CCPP	9568	0	5	V	↓	0.080
CPU	209	1	8	MMax	↑	0.074
Elevators	9516	0	6	ClimbRate	↑	0.080
Flare	1065	10	1	LargestSpot	↑	0.065
Housing	452	1	13	LSTAT	↓	0.087
MPG	392	3	5	Horsepower	↓	0.084
Red Wine	1599	0	12	Alcohol	↑	0.090
Yacht	308	0	7	Froude	↑	0.035

8.1.1 Ant-Miner-Reg$_{\text{MC}}$ and Ant-Miner-Reg$_{\text{PB+MC}}$ Results

Initially, a combination of soft constraints during the learning phase and a hard pruner was used to satisfy the imposed monotonic constraints in the models constructed. Table 8.3 contains the computational results of these experiments. The algorithms being tested include the IRL-based algorithm Ant-Miner-Reg$_{\text{MC}}$ and the Pittsburgh-based algorithm Ant-Miner-Reg$_{\text{PB+MC}}$. These two algorithms incorporate monotonic soft constraints in the learning phase and the simple backtrack naive pruner in a post-processing step to rigidly enforce the monotonic constraints. SeCoReg+MC is a modified greedy IRL learner that uses soft constraint information in the rule construction phase along with the naive pruner like the two previous ACO-based algorithms.

M5' Rules is also included as a comparison to a classical rule induction algorithm to show that the new algorithms are competitive and the introduction of constraints is not detrimental to the predictive accuracy of the models produced. It should be noted that the models created by M5' Rules are not guaranteed to be monotonic.

Table 8.4 shows that Ant-Miner-Reg$_{\text{PB+MC}}$ achieves the best average rank of all the algorithms, statistically outperforming both SeCoReg+MC and Ant-Miner-Reg$_{\text{MC}}$ in the data-sets that were used. For each data-set each algorithm was assigned a rank from 1 to 4 with the best algorithm on that particular data-set having a rank of 1 and the worst a rank of 4. These ranks were then averaged over all the test data-sets.

Table 8.3: RRMSE of the model produced by each algorithm in each of the eight data sets. The bold value indicates the smallest error of the four algorithms; the standard deviation is shown in square brackets.

Data Set	Ant-Miner-Reg$_{MC}$	Ant-Miner-Reg$_{PB+MC}$	SeCoReg+MC	M5' Rules
CCPP	**0.1715 [0.0126]**	0.1956 [0.0152]	0.2853 [0.0150]	0.2375 [0.0136]
CPU	0.3564 [0.2269]	0.2451 [0.0451]	0.4194 [0.2074]	**0.1707 [0.1438]**
Elevators	0.8574 [0.1421]	**0.4468 [0.0287]**	1.0017 [0.0020]	0.6014 [0.0134]
Flare	0.9480 [0.0283]	**0.9221 [0.0512]**	1.0099 [0.0144]	1.0086 [0.0257]
Housing	0.5088 [0.2732]	**0.3986 [0.0157]**	0.8936 [0.2282]	0.4396 [0.1154]
MPG	0.4639 [0.1724]	**0.2544 [0.0897]**	0.6203 [0.0588]	0.3723 [0.0455]
Red Wine	1.0745 [0.5425]	**0.7759 [0.0651]**	1.0099 [0.0160]	0.8068 [0.0354]
Yacht	0.1811 [0.0385]	**0.0767 [0.0985]**	0.4597 [0.1168]	0.0833 [0.0264]

Table 8.4: Non-parametric Friedman test with Holm's post-hoc test results based on the average RRMSE of the four algorithms used in the experiments. Statistically significant results at the $\alpha = 0.05$ level are shown in bold.

Algorithm	Avg. Rank	p-value	Holm
Ant-Miner-Reg$_{\text{PB+MC}}$	1.25	–	–
M5' Rules	2.125	0.1752	0.05
Ant-Miner-Reg$_{\text{MC}}$	**2.75**	**0.0201**	**0.025**
SeCoReg+MC	**3.875**	$\mathbf{4.7697 \times 10^{-5}}$	**0.0166**

Discussion

Ant-Miner-Reg$_{\text{MC}}$ and Ant-Miner-Reg$_{\text{PB+MC}}$ have been shown to produce models that preserve the monotonic constraints imposed upon them. The results also show that this can be achieved without negatively impacting the predictive accuracy of the produced models; since Ant-Miner-Reg$_{\text{PB+MC}}$ achieved the best average rank of all algorithms, including the classical rule induction algorithm M5' Rules. While there is no statistically significant difference between the two algorithms (Table 8.4), Ant-Miner-Reg$_{\text{PB+MC}}$ outperforms M5' Rules in seven of the eight data sets. In addition, as shown in Table 8.4, Ant-Miner-Reg$_{\text{PB+MC}}$ has been shown to be significantly better than the greedy search algorithm SeCoReg+MC, which has been modified to preserve the same constraints as Ant-Miner-Reg+MC.

The results in Table 8.4 also show that the Pittsburgh-based approach Ant-Miner-Reg$_{\text{PB+MC}}$ statistically outperformed the IRL learner Ant-Miner-Reg$_{\text{MC}}$. This increased predictive performance shows that the ability to optimise inter-rule interactions produces more accurate models. The additional process of enforcing monotonicity also benefits from the optimisation of these interactions, as monotonicity is a global property of the model and not the property of a single rule.

The algorithms whose results were presented in this section all use a naive hard monotonic pruner to ensure that any models produced are monotonic. However,

this simple pruner has the potential to drastically alter the list when ensuring constraints are satisfied. In the next section additional pruning strategies have been added to the algorithms to create a pruning suite to enforce the constraints with the minimum number of changes to the models discovered during the learning phase.

8.1.2 Results for Monotonic Algorithms with Proposed Pruning Suite

The proposed pruning suite consists of the three pruners outlined in Section 7.2, where each pruner is applied in a post-processing stage to the rule list produced in the learning phase. This creates three pruned rule lists, which are evaluated on the training data, and the best list is selected as the final model by the algorithm.

The pruning suite has been applied to three different algorithms. Two ACO-based algorithms: Ant-Miner-$\text{Reg}_{\text{PB+MCP}}$, which contains the soft constraint learning phase; and Ant-Miner-Reg_{PB}, which is the conventional monotonically unaware regression ACO-based algorithm. Additionally, the pruning suite has been applied to M5'Rules to produce monotonic models that can be compared to the proposed ACO-based approaches.

Table 8.5: RRMSE averages for the six algorithms being tested. Data-sets were chosen from the UCI Machine Learning Repository (Lichman 2013) as having attributes with low NMI values, allowing the enforcement of constraints. The best result (lowest RRMSE value) achieved for each data-set is shown in bold.

Data Set	Ant-Miner-$\text{Reg}_{\text{PB+MC}}$	Ant-Miner-$\text{Reg}_{\text{PB+MCP}}$	Ant-Miner-Reg_{PB} + Pruners	Ant-Miner-Reg_{PB}	M5' Rules + Pruners	M5' Rules
CCPP	**0.1715 [0.013]**	**0.1715 [0.013]**	0.2953 [0.054]	0.2385 [0.009]	0.3768 [0.028]	0.2375 [0.014]
CPU	0.2451 [0.045]	0.1957 [0.068]	0.4149 [0.095]	0.3694 [0.130]	**0.1658 [0.097]**	0.1707 [0.144]
Elevators	**0.4468 [0.029]**	0.4545 [0.049]	0.4568 [0.035]	0.5410 [0.014]	0.5941 [0.867]	0.6014 [0.013]
Flare	**0.9221 [0.051]**	**0.9221 [0.051]**	1.0095 [0.067]	0.9603 [0.111]	1.0086 [0.026]	1.0086 [0.026]
Housing	0.3986 [0.016]	**0.3324 [0.057]**	0.4218 [0.824]	0.3744 [0.129]	0.4768 [0.034]	0.4396 [0.115]
MPG	0.2544 [0.090]	**0.2432 [0.058]**	0.2706 [0.104]	0.2713 [0.058]	0.3657 [0.040]	0.3723 [0.046]
Red Wine	0.7759 [0.065]	**0.7267 [0.078]**	0.7792 [0.057]	0.7800 [0.044]	0.8157 [0.029]	0.8068 [0.035]
Yacht	**0.0757 [0.099]**	**0.0757 [0.099]**	0.0954 [0.048]	0.0694 [0.134]	0.1547 [0.069]	0.0833 [0.026]

Table 8.6: Non-parametric Friedman test with Holm's post-hoc test results based on the average RRMSE of the six algorithms used in the experiments. Statistically significant results at the $\alpha = 0.05$ level are shown in bold.

Algorithm	Avg. Rank	p-value	Holm
Ant-Miner-Reg$_{\text{PB+MCP}}$	1.6875	–	–
Ant-Miner-Reg$_{\text{PB+MC}}$	2.1875	0.5929	0.05
Ant-Miner-Reg$_{\text{PB}}$	3.375	0.0712	0.025
Ant-Miner-Reg$_{\text{PB}}$ + Pruners	**4.375**	**0.0041**	**0.0166**
M5' Rules	**4.6875**	$\mathbf{3.2834 \times 10^{-3}}$	**0.0125**
M5' Rules + Pruners	**4.9375**	$\mathbf{5.1200 \times 10^{-4}}$	**0.01**

Table 8.5 shows the average RRMSE of the six algorithms tested. Three of the algorithms (Ant-Miner-Reg$_{\text{PB+MCP}}$, Ant-Miner-Reg$_{\text{PB}}$+ Pruners, and M5' + Pruners) use the new pruning suite. The non-monotonic Ant-Miner-Reg$_{\text{PB}}$ and M5' Rules algorithms have been included to allow a comparison between monotonic and non-monotonic variants of the same algorithm.

The Friedman statistical test and the Holm post-hoc test were performed, and the results of these statistical tests along with the average rank of each algorithm can be seen in Table 8.6. It shows that the algorithm with the best average rank was Ant-Miner-Reg$_{\text{PB+MCP}}$, which produced the lowest average RRMSE in six of the eight data-sets. The additional pruning suite managed to match or improve the RRMSE of Ant-Miner-Reg$_{\text{PB+MC}}$, in seven of the eight data-sets.

To assess the changes performed by the pruners on the complete rule lists, a comparison of the rule list lengths was performed, before and after the pruners were applied. The results of this comparison can be found in Table 8.7. The table includes the average number of rules present in the list before the post-processing pruners are applied and the number present after the pruning. It also shows the average number of terms removed by the pruners to ensure the resultant list is completely monotonic.

Table 8.7: Average rule list size of the ACO-based rule learners with different monotonic constraint pruners. The first two algorithms use a soft constraint learning phase, while the last has no knowledge of monotonicity during the learning stage. The results include the number of rules before and after pruning along with the total number of terms removed by the pruner.

Data Set	Ant-Miner-Reg$_{\text{PB+MC}}$			Ant-Miner-Reg$_{\text{PB+MCP}}$			Ant-Miner-Reg$_{\text{PB}}$+ Pruners		
	Before	After	-Terms	Before	After	-Terms	Before	After	-Terms
CCPP	110	108	4	110	108	4	126	115	35
CPU	13	9	11	13	12	2.4	12	8	10
Elevators	18	16	9	18	18	3	14	11	8
Flare	10	10	1.2	10	10	1.2	11	9	7
Housing	25.3	20	16	25.3	24	3	19	13	19
MPG	15	14	4	15	15	1.1	13	12.5	0.75
Red Wine	31	28	8.6	31	31	1.2	33	32	3
Yacht	15.5	15	1.5	15.5	15	1.5	14	13	1.5

Discussion

The results show that, for both Ant-Miner-Reg$_{\text{PB}}$ and M5' Rules, the addition of the pruning suite hurts the performance of the algorithms, resulting in a worse average rank than the original non-monotonic algorithms, as shown in Table 8.6. This was not observed in the performance of Ant-Miner-Reg$_{\text{PB+MC}}$, where the additional pruning suite performed better on average, as can be seen by comparing the average rank of Ant-Miner-Reg$_{\text{PB+MCP}}$ (1.69) against the average rank of Ant-Miner-Reg$_{\text{PB+MC}}$ (2.19).

The statistical test results in Table 8.6 shows that the combination of soft constraints during the learning phase of an algorithm coupled with the post-processing pruning suite statistically outperforms the base ACO algorithm Ant-Miner-Reg$_{\text{PB}}$ with the monotonic pruning suite. This shows that combining both soft and hard constraint approaches is better than just enforcing hard constraints for these regression data-sets. The statistical test also finds no difference in predictive performance between non-monotonic and monotonic ACO-based algorithms, showing that enforcing monotonic constraints did not negatively affect performance, contra to previous work in the literature that has suggested that monotonic constraints may affect the accuracy of models produced by machine learning algorithms (Ben-David, Sterling and Tran 2009).

Finally, Table 8.7 shows the effects of the hard monotonic pruners. It can be seen in general that the use of the pruning suite results in a small change to the rule list size, with a reduced number of rules and terms being removed, compared to using the single naive pruner. The more "intelligent" pruners introduced in the pruning suite are able to ensure a monotonic rule list, while performing fewer changes to the rule list created during the learning phase. When the pruning suite is applied to the non-monotonic Ant-Miner-Reg we see an increase in the number of rules and terms removed. This is expected as the rule lists generated by Ant-Miner-Reg were not optimised for their monotonic features, but purely

for accuracy. This shows that the soft constraint learning phase present in Ant-Miner-Reg$_{\text{PB+MC}}$ and Ant-Miner-Reg$_{\text{PB+MCP}}$ positively affects the monotonicity of the rule list produced and therefore requires less pruning.

8.2 Results for Classification with Monotonic Constraints

The classification results have also been divided into two sections. First I will compare the monotonic algorithms and then compare the best monotonic algorithm to traditional non-monotonic rule learners. As I am concentrating on rule induction and the comprehensible models that they produce, I will only be considering the performance of our proposed algorithms against other rule induction algorithms. This allows a fair comparison, removing any biases that may be present due to model representation.

In all experiments, *c*Ant-Miner variations were configured with a colony size of 5 ants, 500 iterations, minimum cases covered by an individual rule of 10, uncovered instance ratio of 0.1, and constraint weighting (ω) of 0.5 (only used by *c*Ant-Miner$_{\text{PB+MC}}$ and its derivatives). The values for the colony size, minimum coverage and number of iterations were taken from those used by the original *c*Ant-Miner(Otero, Freitas and Johnson 2008).

The eight chosen algorithms were tested on thirteen data sets taken from the UCI Machine Learning Repository (Lichman 2013). Table 8.8 presents the details of the chosen data sets, including a summary of the constraints used. All predictor attributes had their NMI calculated to discover good monotonic relationships—the NMI results guided the choice of constrained attribute reported in Table 8.8. Classification data-sets were chosen at random from the UCI Machine Learning repository with the only requirement for a data set is an appropriate target attribute. An acceptable target is one that is ordinal in nature e.g. a credit score

target attribute with values `low`, `medium`, and `high`. Like the previously presented results for regression, ten-fold cross validation was used for each algorithm tested, along with the stochastic ACO-based algorithms being run five times (varying the random initialisation seed) on each fold, with the average of all the folds and runs (i.e. the average over fifty results) being reported in the results tables.

8.2.1 Results of Monotonic Algorithms

To test the effectiveness of both additive and subtractive monotonic post-processing methods, four algorithms have been proposed: the first two use cAnt-Miner$_{\text{PB}}$ as the base, one of them using the monotonic pruners, and the other using RULEM as post-processing steps. The other two algorithms use cAnt-Miner$_{\text{PB+MC}}$ as the base, which incorporates the soft constraints into the model construction, and then uses either the monotonic pruners or RULEM to enforce the constraints rigidly.

These four algorithms have also been compared against the monotonic rule learner OLM (Ben-David, Sterling and Tran 2009). Table 8.9 shows the predictive accuracy of all the algorithms on the thirteen data sets, with the standard deviation shown in brackets. The highest accuracy achieved on each data set is shown in bold.

In summary, the algorithms that incorporate soft constraints, cAnt-Miner$_{\text{PB+MCP}}$ and cAnt-Miner$_{\text{PB+MC}}$+RULEM, achieve the best result of all the algorithms in seven and three of the thirteen data-sets, respectively; cAnt-Miner$_{\text{PB}}$+Pruners achieves the best result in two data-sets; cAnt-Miner$_{\text{PB}}$+RULEM and OLM achieve the best result in one apiece.

Table 8.8: Monotonic classification data sets from the UCI Machine Learning repository (Lichman 2013) used in experiments, including attributes and constraint information. In each data-set a single attribute was constrained. The constraint information contains the attribute's name, direction of the constraint, either ↑ (increasing) or ↓ (decreasing) and its corresponding NMI.

		Attributes		Constraint		
Name	Size	Nominal	Continuous	Attribute	Direction	NMI
Abalone	4176	1	7	Shell Weight	↑	0.8062
Australian Credit	689	9	6	A8	↓	0.9925
Bank Marketing	4520	9	7	Loan	↓	0.9859
Cancer	698	0	10	USize	↑	0.0059
Car	1727	6	0	Safety	↑	0.0460
Credit Screen	689	9	6	A4	↑	0.9444
German Credit	689	9	6	Credit History	↓	0.9189
Haberman	305	0	3	PosNode	↑	0.0861
MPG	397	0	7	Horsepower	↓	0.0566
Pima	767	0	8	PGC	↑	0.0947
User Knowledge	402	0	5	PEG	↑	0.9764
Wine	177	0	13	Flavanoids	↓	0.964
Wine Quality	1598	0	11	Alcohol	↑	0.8373

Table 8.9: Accuracy of the five monotonic rule learners. OLM is an existing monotonic learner, the other four algorithms are ACO-based algorithms using a combination of soft constraints and hard constraints at different stages of the learning process. The best result for each data set is shown in bold.

Data set	OLM	*c*Ant-Miner$_{\text{PB}}$+ Pruners	*c*Ant-Miner$_{\text{PB}}$+ RULEM	*c*Ant-Miner$_{\text{PB+MCP}}$	*c*Ant-Miner$_{\text{PB+MC}}$+ RULEM
Abalone	0.1609 [0.0164]	0.2500 [0.0152]	0.1354 [0.0065]	**0.2583 [0.0083]**	0.1294 [0.0056]
Australian Credit	0.6449 [0.0646]	0.8501 [0.0562]	0.8345 [0.0097]	**0.8554 [0.0402]**	**0.8554 [0.0402]**
Bank Marketing	0.8828 [0.0482]	0.7954 [0.0242]	0.8717 [0.0097]	**0.8949 [0.0131]**	0.8746 [0.0343]
Cancer	0.8355 [0.0149]	0.9465 [0.0214]	0.7241 [0.0158]	**0.9574 [0.0166]**	0.7743 [0.0354]
Car	**0.9055 [0.0187]**	0.8452 [0.0274]	0.7958 [0.0096]	0.8964 [0.0149]	0.8179 [0.0356]
Credit Screen	0.5681 [0.0654]	0.8546 [0.0546]	**0.8645 [0.0564]**	0.8612 [0.0385]	0.8356 [0.2565]
German Credit	0.6700 [0.0153]	**0.7465 [0.0674]**	0.7000 [0.6874]	0.7416 [0.0369]	0.6946 [0.0645]
Haberman	0.6993 [0.0781]	0.7405 [0.0791]	0.7097 [0.6741]	0.7417 [0.0917]	**0.7419 [0.0654]**
MPG	0.7663 [0.0367]	0.7641 [0.0641]	0.7555 [0.0664]	**0.9256 [0.0274]**	0.7587 [0.0124]
Pima	0.7161 [0.0589]	0.7456 [0.0665]	0.6623 [0.0695]	**0.7494 [0.0707]**	0.7013 [0.0963]
User Knowledge	0.4839 [0.0398]	0.9242 [0.0157]	0.8987 [0.0678]	0.9271 [0.0355]	**0.9346 [0.0646]**
Wine	0.3202 [0.0201]	**0.9875 [0.0264]**	0.8889 [0.0345]	0.9605 [0.0377]	0.5555 [0.0564]
Wine Quality	0.2808 [0.0276]	0.5412 [0.0447]	0.3183 [0.0248]	**0.5743 [0.0391]**	0.3178 [0.0641]

Table 8.10: Friedman statistical test with Holm's post-hoc test results. Average rank and p values of the monotonic algorithms tested. Results that showed a statistically significant difference according to the $\alpha = 0.05$ level are shown in bold.

Algorithm	Avg. Rank	p	Holm
cAnt-Miner$_{\text{PB+MCP}}$	1.4910	-	-
cAnt-Miner$_{\text{PB}}$+Pruners	2.5385	0.0940	0.05
cAnt-Miner$_{\text{PB+MC}}$+RULEM	**3.2692**	$\mathbf{4.3336 \times 10^{-3}}$	**0.025**
OLM	**3.7692**	$\mathbf{2.5318 \times 10^{-4}}$	**0.0167**
cAnt-Miner$_{\text{PB}}$+RULEM	**3.9231**	$\mathbf{9.3412 \times 10^{-5}}$	**0.0125**

Table 8.10 shows the results of a Friedman statistical test with Holm's post-hoc test performed on the accuracy results presented in Table 8.9, where we can see that cAnt-Miner$_{\text{PB+MCP}}$ achieves the lowest (best) average rank and significantly outperforms the monotonic learner OLM and both ACO variants that use the RULEM post-processing method.

The last set of results in this section concerns the effect of different post-processing procedures to the size of the rule lists produced by cAnt-Miner$_{\text{PB+MC}}$. Table 8.11 shows the average number of rules in the rules list before and after post-processing along with the number of terms removed by the pruning suite, or added by RULEM.

Discussion:

Our results show that algorithms that used subtractive pruners performed better than the ones using RULEM, which is an additive approach. RULEM adds additional rules to a rule list, which could lead to over-fitting of the data — if the rules added by RULEM are good rules and therefore increase predictive accuracy, it would be reasonable to expect the learning algorithms to discover them. These additionally created rules that are added to the top of a list reduce the effectiveness

Table 8.11: Average number of rules in the rule lists created by *c*Ant-Miner$_{\text{PB+MC}}$ with the additional monotonic pruning suite and RULEM added as post-processing steps to enforce constraints. The results include the number of rules before and after post-processing along with the number of terms removed or added.

Data Set	*c*Ant-Miner$_{\text{PB+MCP}}$			*c*Ant-Miner$_{\text{PB+MC}}$ + RULEM		
	Before	After	−Terms	Before	After	+Terms
Abalone	25.0	22.0	6.0	25.0	48.0	33.5
Australian Credit	6.5	6.5	0.0	6.5	6.5	0.0
Bank Marketing	12.0	11.5	1.5	12.0	13.0	2.0
Cancer	6.0	5.5	3.0	6.0	7.2	4.0
Car	26.2	24.0	7.0	26.2	34.5	13.0
Credit Screen	10.0	9.0	2.5	10.0	16.2	19.4
German Credit	7.5	7.2	1.2	7.5	12.0	8.2
Haberman	8.0	7.0	2.0	8.0	18.4	12.0
MPG	9.2	8.5	3.0	9.2	18.0	24.5
Pima	5.2	4.8	0.5	5.2	9.1	5.0
User Knowledge	8.0	6.5	4.0	8.0	9.5	2.0
Wine	5.0	4.8	1.0	5.0	16.5	12.0
Wine Quality	12.5	11.0	4.0	12.5	22.0	17.0

of the previously generated rules, as rules at the top of the list will preferentially make predictions over those lower in the list. Subtractive pruners, instead, can only generalise a rule (by removing terms from it), allowing it to cover more instances. While overly generalised rule's will hurt a models predictive accuracy, the monotonic pruners here aim to minimise changes to the model.

Previous experiments involving RULEM have been focused on algorithms that employ the sequential covering technique, which generally ignores rule interactions when constructing a model. In fact, this is one of the reasons RULEM's authors focused on post-processing, as monotonicity is a global property (Verbeke, Martens and Baesens 2017). However, *c*Ant-Miner$_{\text{PB}}$ and its derivatives generate an entire

rule list in each iteration of the algorithm. This allows for rule interactions to be optimised and, therefore, the additional rules generated by RULEM may disrupt these rule interactions present in the models, negatively affecting the accuracy.

Due to the global optimisation of models by *c*Ant-Miner$_{\text{PB}}$, a logical step is to introduce monotonic constraints to the learning phase. The decision to implement a soft constraint regime at this stage is to nudge (bias) ants towards good monotonic solutions while not restricting the search space they operate in. Our experiments show that incorporating those constraints into the learning phase minimises the changes required in a potentially destructive post-processing step to fix the model. Embedding constraints into the learning phase allows the ant colony to optimise the rule list based on all the requirements that a user wishes and not to enforce new requirements, or at least minimise the additional requirements, after the model has been optimised.

Table 8.12: Comparison of the model accuracy of the best monotonic rule learner *c*Ant-Miner$_{\text{PB+MCP}}$ to traditional non-monotonic rule learners, including the original *c*Ant-Miner$_{\text{PB}}$. The best result for each data set is shown in bold.

Data set	*c*Ant-Miner$_{\text{PB+MCP}}$	JRip	C5.0 Rules	*c*Ant-Miner$_{\text{PB}}$
Abalone	**0.2583 [0.0083]**	0.1906 [0.0284]	0.2303 [0.0310]	0.2562 [0.0215]
Australian Credit	0.8554 [0.0402]	0.8507 [0.0315]	**0.8639 [0.0363]**	0.8580 [0.0501]
Bank Marketing	**0.8949 [0.0131]**	0.8936 [0.0146]	0.8919 [0.0125]	0.8938 [0.014]
Cancer	**0.9574 [0.0166]**	0.9542 [0.0256]	0.9527 [0.0223]	0.9566 [0.0181]
Car	0.8964 [0.0149]	0.8646 [0.0134]	**0.9543 [0.0137]**	0.8929 [0.0151]
Credit Screen	0.8612 [0.0385]	**0.8936 [0.0485]**	0.8612 [0.0393]	0.8493 [0.0479]
German Credit	0.7416 [0.0369]	0.7350 [0.0468]	0.7120 [0.0444]	**0.7490 [0.0509]**
Haberman	**0.7417 [0.0917]**	0.7222 [0.0387]	0.7288 [0.0764]	0.7405 [0.0791]
MPG	**0.9256 [0.0274]**	0.9095 [0.0856]	0.9247 [0.0353]	0.9200 [0.0293]
Pima	0.7494 [0.0707]	**0.7513 [0.0715]**	0.7377 [0.0698]	0.7493 [0.0564]
User Knowledge	0.9271 [0.0355]	0.9280 [0.0269]	**0.9281 [0.0473]**	0.9254 [0.0486]
Wine	**0.9605 [0.0377]**	0.9494 [0.0156]	0.9436 [0.0594]	0.9444 [0.0586]
Wine Quality	0.5743 [0.0391]	0.5860 [0.0212]	**0.6128 [0.0543]**	0.5523 [0.0477]

Table 8.13: Average rank and p values of the best monotonic algorithm cAnt-Miner$_{\text{PB+MCP}}$ and three non-monotonic rule learners according to the non-parametric Friedman test. Holm's post-hoc test was used to check for significance at $\alpha = 0.05$.

Algorithm	Avg. Rank	p	Holm
cAnt-Miner$_{\text{PB+MCP}}$	1.8077	-	-
C5.0 Rules	2.6538	0.0947	0.05
cAnt-Miner$_{\text{PB}}$	2.6923	0.0806	0.025
JRip	2.8462	0.0403	0.0167

8.2.2 Results of cAnt-Miner$_{\text{PB+MCP}}$ Against Classical Rule Induction algorithms

The best monotonic algorithm from the previous experiments, cAnt-Miner$_{\text{PB+MCP}}$, was also compared to three traditional non-monotonic algorithms, namely JRip (Cohen 1995), Quinlan's C5.0 Rules[1], and the original ACO-based algorithm cAnt-Miner$_{\text{PB}}$(Otero, Freitas and Johnson 2013), to show weather any loss of predictive accuracy has occurred or not due to the addition of monotonic constraints. The results of these experiments are shown in Table 8.12, with the statistical analysis shown in Table 8.13. Table 8.12 shows that cAnt-Miner$_{\text{PB+MCP}}$ as the best performing algorithm achieving the best accuracy of all the four algorithms in six data-sets with C5.0 Rules second, achieving the best results in four data-sets. While JRip and cAnt-Miner$_{\text{PB}}$ achieving the best results in two and one data-sets respectively.

To summarise, while no statistical significance was observed in Table 8.13, cAnt-Miner$_{\text{PB+MCP}}$ achieved the lowest (best) average rank and managed to outperform the other algorithms in six of the thirteen data-sets. The results also show that cAnt-Miner$_{\text{PB+MCP}}$ has not suffered a drop in predictive accuracy compared to the original algorithm cAnt-Miner$_{\text{PB}}$ with the inclusion of additional

[1] `https://www.rulequest.com/see5-unix.html`

constraints into the learning process, in fact it has achieved a substantially better average rank than the original base algorithm.

The proposed post-processing pruning suite (Section 7.2) and RULEM (Verbeke, Martens and Baesens 2017) can be added to any learning algorithm that produces a list of rules to create a monotonic rule list. Table 8.14 presents the results of adding a monotonic post-processing stage to the classical rule induction algorithms JRip and C5.0 to compare them to the best monotonic algorithm presented earlier, cAnt-Miner$_{\text{PB+MCP}}$.

A Friedman statistical test with Holm's post-hoc test was performed on the results of the non-monotonic learners that have had a post-processing constraint procedure added, the results are shown in Table 8.15. cAnt-Miner$_{\text{PB+MCP}}$ was found to be statistically better than the traditional algorithms using either the subtractive pruning suite proposed earlier in Section 7.2 or the existing additive procedure RULEM.

Table 8.14: Comparison of the model accuracy of the best monotonic rule learner *c*Ant-Miner$_{\text{PB+MCP}}$ to traditional non-monotonic rule learners that have had the additional monotonic post-processing techniques added. The best accuracy obtained for each data set is shown in bold.

Data set	*c*Ant-Miner$_{\text{PB+MCP}}$	JRip + Pruners	JRip + RULEM	C5.0 Rules + Pruners	C5.0 Rules + RULEM
Abalone	**0.2583 [0.0083]**	0.2137 [0.0346]	0.1934 [0.0352]	0.2403 [0.0416]	0.2315 [0.0469]
Australian Credit	**0.8554 [0.0402]**	0.8237 [0.0475]	0.7924 [0.0352]	0.8234 [0.0345]	0.6945 [0.0156]
Bank Marketing	**0.8949 [0.0131]**	0.8675 [0.0354]	0.8742 [0.0157]	0.8454 [0.0468]	0.8735 [0.0234]
Cancer	**0.9574 [0.0166]**	0.9423 [0.0130]	0.9124 [0.0271]	0.9004 [0.0231]	0.9312 [0.0496]
Car	0.8964 [0.0149]	0.8327 [0.0534]	0.8421 [0.0374]	**0.9224 [0.0453]**	0.8837 [0.0341]
Credit Screen	0.8612 [0.0385]	**0.9017 [0.0558]**	0.8537 [0.0370]	0.8357 [0.0359]	0.8218 [0.0289]
German Credit	**0.7416 [0.0369]**	0.7010 [0.0654]	0.6953 [0.0561]	0.6745 [0.0444]	0.6950 [0.0552]
Haberman	**0.7417 [0.0917]**	0.7157 [0.0481]	0.7057 [0.0351]	0.7036 [0.0638]	0.7041 [0.0472]
MPG	**0.9256 [0.0274]**	0.8865 [0.0537]	0.7952 [0.0675]	0.9035 [0.0468]	0.8465 [0.0341]
Pima	**0.7494 [0.0707]**	0.7245 [0.0367]	0.6542 [0.2337]	0.7337 [0.0187]	0.6782 [0.1237]
User Knowledge	**0.9271 [0.0355]**	0.8935 [0.0541]	0.8935 [0.0541]	0.9057 [0.0357]	0.8947 [0.0156]
Wine	**0.9605 [0.0377]**	0.9524 [0.0145]	0.9341 [0.0194]	0.9436 [0.0594]	0.9567 [0.0357]
Wine Quality	0.5743 [0.0391]	0.5314 [0.0359]	0.4913 [0.0474]	**0.6021 [0.0451]**	0.5567 [0.0481]

Table 8.15: Average ranks and p values of the best monotonic algorithm cAnt-Miner$_{\text{PB+MCP}}$ and two non-monotonic rule learners with a monotonic post-processing procedure applied. A Friedman test with Holm's post-hoc test was used to check for significance at $\alpha = 0.05$ — significant results are shown in bold.

Algorithm	Avg. Rank	p	Holm
cAnt-Miner$_{\text{PB+MCP}}$	1.4615	-	-
JRip + Pruners	**2.9615**	**0.0156**	**0.05**
C5.0 Rules + Pruners	**3.0769**	**0.0092**	**0.025**
C5.0 Rules + RULEM	**3.4615**	**0.0013**	**0.0167**
JRip + RULEM	**4.0385**	$\mathbf{3.2505 \times 10^{-5}}$	**0.0125**

Discussion

The results show that cAnt-Miner$_{\text{PB+MCP}}$ achieved a higher average rank than cAnt-Miner$_{\text{PB}}$. This is particular interesting as Ben-David, Sterling and Tran (2009) have previously suggested that enforcing monotonic constraints may harm predictive accuracy. However, we hypothesise that if constraints are correctly identified, this additional knowledge should allow the construction of more accurate and generalised models, helping algorithms ignore some of the noise present in real world data sets.

When Ben-David, Sterling and Tran (2009) investigated monotonic constraints, they focused on the post-processing of models, removing any non-monotonic features after a model has been created. The results presented in Table 8.14 show that limiting enforcement to a post-processing stage may indeed harm the performance of the resultant model and a more holistic approach should be embraced.

The combination of hard and soft constraints allows a broader exploration of the search space. This may lead to the discovery of good monotonic regions that could be missed if algorithms were prohibited from exploring non-monotonic regions of the search space, as they attempt to find monotonic solutions. The soft constraints bias the ant colony to explore monotonic regions more thoroughly,

however as it is a soft suggestion, the colony is also able to explore non-monotonic regions that produce accurate models. The hard constraints as a post-processing step ensure the final model does not violate the constraints that were imposed on it, which guarantees the creation of monotonic models.

In this context, ACO is a good meta-heuristic to incorporate both soft and hard constraints. The pheromone matrix represents the memory of the colony acting as a guide to good solutions for its individual ants. However, each ant is still able to take different paths and construct individual solutions, since pheromone values bias the selection of terms while maintaining the stochastic nature of the selection. The soft constraints implemented during the learning phase, modify the pheromone levels throughout the matrix, naturally suggesting good monotonic solutions. Once the colony has obtained its best solution, the hard constraint pruners can make (ideally) small changes to the rule list to ensure it is completely monotonic.

Chapter 9

Towards an Archive-Based ACO for Regression

.

An initial investigation into using an archive-based approach for learning regression rules has been performed with the preliminary results obtained shown in Section 9.2. This approach replaces the traditional graph-pheromone model with an archive-pheromone model (Discussed in Section 4.4), which is then sampled during rule creation and updated with the good rules constructed in each iteration.

The work in this section was a collaboration with Ayah Helal, who has previously used the archive-pheromone model to tackle the classification task (Helal and Otero 2016). The base mechanism of this algorithm was combined with the regression-specific functionality found in Ant-Miner-Reg, creating Ant-Miner-Reg_{MA}. Ant-Miner-Reg_{MA} was first introduced in WCCI 2018 (Helal, Brookhouse and Otero 2018).

9.1 Archive-Based ACO - Ant-Miner-Reg$_{\text{MA}}$

The previous proposed Ant-Miner-Reg algorithms use the traditional graph-based Ant Colony Optimization approach to creating rules using an internal pheromone graph model. This model works well when dealing with categorical attributes, which are naturally represented by nodes in the graph. However, when presented with continuous attributes an extra step is required, either a pre-processing step or a dynamic discretization step. The previously presented ACO-based algorithms for regression, in Chapter 5, use the latter graph-based combinatorial approach. Ant-Miner-Reg$_{\text{MA}}$ uses the IRL strategy to generate a list of rules by splitting the task of creating a list into many smaller tasks of creating a single rule.

Liao et al. (2014) introduced the Ant Colony Optimization for Mixed-Variable (ACO$_{\mathbf{MV}}$) algorithm, which was designed for mixed variable optimisation problems. ACO$_{\mathbf{MV}}$ uses an archive-based pheromone model and a number of sampling procedures to create solutions, allowing the algorithm to directly work with categorical, ordinal and continuous (real-valued) attributes. The archive-based pheromone model is implemented as a solution archive (A), which contains the k previously generated best solutions. The solution archive is used to derive a probability distribution to sample from, biasing the search towards good solutions.

A colony of m ants start generating candidate solutions. During the solution construction, each ant uses a probabilistic method to sample new values from the solution archive. The sampling mechanism is chosen based on the type of attribute. In Ant-Miner-Reg$_{\text{MA}}$ two types are attributes are used, categorical attributes and continuous attributes. Ordinal attributes are treated in the same way as categorical attributes where the order is not taken into account. This decision was made to ensure feature parity with Ant-Miner-Reg and allow a fair comparison between graph-based and archive-based approaches. After each ant in a colony has created a solution, the m new solutions (where m is the size of the colony) are added to the archive. The archive is then sorted, with the best k

Algorithm 9.1: High-level pseudo code of Ant-Miner-Reg$_{\text{MA}}$

```
   Data: Instances
   Result: RuleList
 1 RuleList ← {}
 2 while |Instances| <MaxUncovered do
 3 |   A ← Generate k Random Rules
 4 |   t ← 0; i ← 0; Restarted ← True
 5 |   while t < MaxIterations and not Restarted do
 6 |   |   A_t ← {}
 7 |   |   while i < m do
 8 |   |   |   R_i ← Create New Rule
 9 |   |   |   R_i ← Prune(R_i)
10 |   |   |   R_i ← Set Consequent(R_i)
11 |   |   |   i ← i + 1
12 |   |   |   A_t ← A_t ∪ R_i
13 |   |   end
14 |   |   A ← UpdateArchive(A_t, k)
15 |   |   t ← t + 1
16 |   |   if stagnation() then
17 |   |   |   Restart(A)
18 |   |   |   Restarted ← True
19 |   |   end
20 |   |   if stagnation() and Restarted then
21 |   |   |   Break
22 |   |   end
23 |   end
24 |   R_best ← BestRule(A)
25 |   RuleList ← RuleList ∪ R_best
26 |   Instances ← Instances − covered(R_best)
27 end
28 return RuleList
```

solutions retained for the next iteration.

Ant-Miner-Reg$_{\text{MA}}$ uses the pheromone model and search procedures found in ACO$_{\mathbf{MV}}$ to create regression rules by sampling terms. The high level pseudo-code of Ant-Miner-Reg$_{\text{MA}}$ is shown in Algorithm 9.1. Ant-Miner-Reg$_{\text{MA}}$ starts with an empty list of rules (line 1). During each iteration (lines 2 - 27), a single rule is created. The initial archive is populated with k randomly generated rules (line

3). The ants then generate m new rules (lines 7-13), where m is the size of the ant colony (line 6). These new rules are then added to the archive (line 14), and all $k + m$ rules are sorted. The worst m rules are removed, leaving the k best rules found so far in the archive, which also returns the archive to its original size. This rule creation procedure is repeated until the maximum number of iterations has been reached or stagnation occurs. Stagnation is the failure of the algorithm to find better rules for a number of iterations. The first time stagnation occurs, a restart procedure is applied; if stagnation is detected a second time, the rule creation procedure is halted and the best rule discovered so far, which is also the rule at the top of the archive, is added to the partial rule list.

9.1.1 Archive Structure and Initialisation

Figure 9.1 shows an example archive used in Ant-Miner-Reg$_{\text{MA}}$, where each rule is represented by a row in the archive and each term is represented by three elements: a flag to signify if the term is currently in use, the operator used by the term and the comparison value selected for that term. The archive shown in Figure 9.1 contains examples of both continuous attributes and categorical attributes and the allowed operators for each attribute.

The archive contains k rules that are sorted by descending quality (Q), so that $Q(R_1) \geq Q(R_2) \geq \ldots \geq Q(R_k)$. Each rule (solution) j is associated with a weight ω_j that is related to its rank in the archive, where the best rule is assigned rank 1 and the worst rule is assigned rank k. A Gaussian function is used to calculate ω_j, which is given by:

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} e^{\frac{-(rank(j)-1)^2}{2q^2k^2}} \tag{45}$$

where q controls the influence of the top-ranked rules on the construction of a new rule. When a new rule is created, the algorithm probabilistically samples values

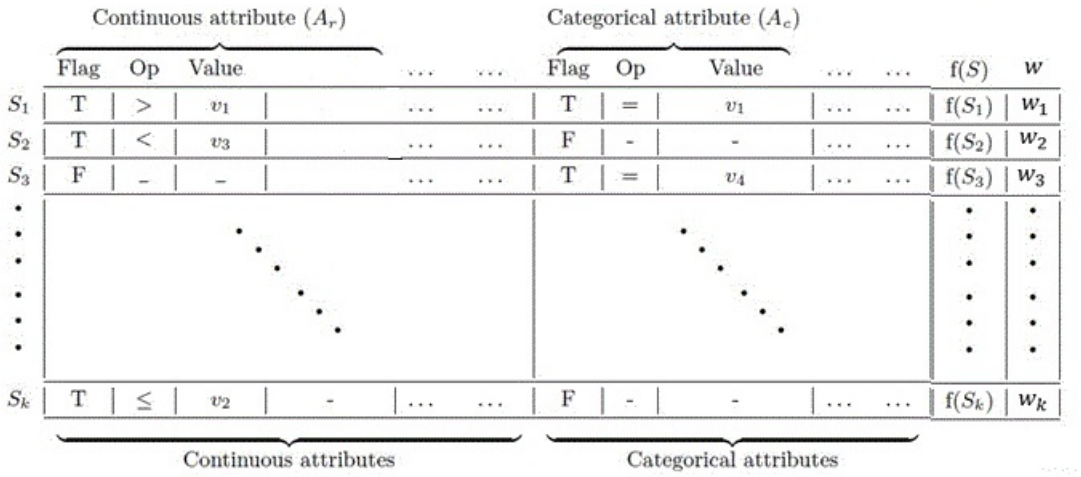


Figure 9.1: Example archive used in Ant-Miner-Reg$_{\text{MA}}$ including the structure of terms with of continuous and categorical attributes.

from a single rule for each attribute based on the weights of the rules and if the rule in the archive has the attribute activated. Rules with higher weights are more likely to be selected for sampling.

Initially k random rules are used to populate the archive. Initialisation begins with randomly enabling each term in the vector of allowed terms. These enabled terms are then initialised based on their types.

If the term is continuous, then an unbiased random probability is used to set the operator from the set $\{\leq, >\}$. The value of the continuous term is a random value uniformly generated from the range found in the training data for that attribute. For categorical terms, there is a single allowed operator, "=", which is added, and the value is set randomly to one of the allowed values for that attribute.

Rules are then pruned to disable irrelevant terms that might be enabled by the stochastic nature of the initialisation procedure. If the number of instances covered by a rule is greater or equal to a user-defined minimum limit, the rule is added to the archive, if it does not a new rule is generated instead. Finally, rules are sorted according to their quality and assigned a ranking and weight.

9.1.2 Sampling Procedures

There are two types of sampling procedures used in Ant-Miner-Reg$_{\text{MA}}$ to select values for rule terms: categorical and continuous sampling.

9.1.2.1 Categorical sampling

The categorical sampling is implemented using the same approach as ACO$_{\mathbf{MV}}$. Given a categorical attribute i that has t_i possible values, an ant chooses probabilistically a value v_l^i of the available $\{v_1^i, \ldots, v_{t_i}^i\}$ values. The probability of selecting a value v_l^i is given by:

$$p_l^i = \frac{\alpha_l}{\sum_{j=1}^{t_i} \alpha_j} \tag{46}$$

where α_l is the weight associated to each value of the categorical attribute, calculated as:

$$\alpha_l = \begin{cases} \frac{\omega_{jl}}{u_l^i} + \frac{q}{\eta} & , \text{if}(\eta > 0, u_l^i > 0) \\ \frac{\omega_{jl}}{u_l^i} & , \text{if}(\eta = 0, u_l^i > 0) \\ \frac{q}{\eta} & , \text{if}(\eta > 0, u_l^i = 0) \end{cases} \tag{47}$$

where ω_{jl} is the weight of the best rule that uses the value v_l^i for attribute i in the archive, u_l^i is the number of rules that use the value v_l^i for attribute i in the archive ($u_l^i = 0$ corresponds to the special case where v_l^i is not used by the rules in the archive), η is the number of values from t_i that are not used in the archive ($\eta = 0$ corresponds to the special case where all values are used), and q is the same parameter used in Equation (45). The categorical sampling procedure allows an ant to consider two components when sampling a new value. The first component biases the sampling towards values that are used in high-quality rules, but do not occur very frequently in the archive. The second component biases the

sampling towards unexplored values in that attribute, preventing the algorithm from ignoring unexplored regions of the search space.

9.1.2.2 Continuous sampling

Continuous sampling implements the same approach found in $\text{ACO}_{\mathbb{R}}$ (Socha and Dorigo 2008), which is also used in $\text{ACO}_{\mathbf{MV}}$. First, an ant probabilistically chooses a rule from the archive, before the sampling procedure. A rule is selected each time a continuous attribute is sampled from the archive. The probability of choosing rule j is given by:

$$p_j = \frac{\omega_j}{\sum_{l=1}^{k} \omega_l} \tag{48}$$

where ω_j is the weight associated with the j-th rule in the archive, calculated according to Equation (45). Let R_i denote a new solution sampled by ant i around the chosen rule R_j for continuous attribute a, the Gaussian probability density function (PDF) is given by:

$$R_{i,a} \sim N(R_{j,a}, \sigma_{j,a}) \tag{49}$$

$$\sigma_{j,a} = \xi \sum_{l=1, j \neq l}^{k} \frac{|R_{l,a} - R_{j,a}|}{K - 1} \tag{50}$$

where $R_{j,a}$ is the value of the attribute a in the selected rule j of the archive, $\sigma_{j,a}$ is the average distance between the value of the attribute a in the rule j and the value of a in all the other rules in the archive, given by Equation 50, where k is the number of rules in the archive and ξ is a user-defined value representing the convergence speed of the algorithm. Higher values of ξ will increase the standard deviation of the Gaussian distribution, leading to increased variability in the values selected, therefore allowing a greater exploration of the search space. While lower

values of ξ will result in the sampling of values closer to the current value, leading to a more thorough search of the current local area.

9.1.3 Rule Creation

Rule creation starts by probabilistically choosing if a term should be included in the current rule's antecedent or not. The decision is handled by the categorical sampling procedure which chooses between the {`TRUE`, `FALSE`} values. If the term is enabled (`TRUE` value), the operator is set according to the attribute type. If the attribute is categorical, it is set to "=". If it is continuous, the categorical sampling procedures selects an operator from the set $\{\leq, >\}$, with the only difference being only the subset of rules that have this term enabled are considered in Equation (47).

The value of the new rule's term is then sampled. If the term is continuous, we use the continuous sampling procedure only considering the subset of rules that have this term enabled and use the same operator which was selected earlier for the new term. If the attribute is categorical, we use the categorical sampling procedure, again only considering the subset of rules that have this term enabled.

After a term is created, it is added to the partial rule, and then the rule is applied to the training data. If the number of instances covered by the rule after the addition of the new term is less than the minimum covered instances (a user-defined parameter), the term is disabled. This is done to ensure that rules retain some generality and do not become over specified. This process is repeated until all terms are evaluated. The rule's consequent is then generated by calculating the mean of the target variable in the instances in the training data-set that are covered by the rule, in the same way as Ant-Miner-Reg.

The last step is the application of a local search procedure. The local search procedure is inspired by the *threshold-aware* pruner found in (Otero, Freitas and

Johnson 2009). Firstly, the quality of the rule is calculated according to Equation (33). Then, the last term added to the rule is disabled and the quality re-calculated. If the quality of the (pruned) rule decreases, the term is re-enabled and the procedure stops. This pruner continues until a decrease in quality is observed.

9.1.4 Restart Procedure

If the algorithm stagnates a restart procedure is performed. Stagnation occurs when the algorithm fails to find a better rule for a number of iterations. The exact number of iterations can be tuned by the user and set before run time.

The restart procedure involves removing all but the best rule generated so far from the archive and replacing the removed rules with randomly generated rules in the same way as during archive initialisation as described in Section 9.1.1.

This leaves an archive containing the best rule and $k - 1$ random rules. The algorithm then proceeds as normal. The restart procedure is applied a maximum of once. If the archive stagnates for a second time the rule creation procedure stops and returns the best rule created so far.

9.2 Archive based Pheromone Model ACO Computational Results

We compared our proposed algorithm Ant-Miner-Reg$_{\text{MA}}$ against Ant-Miner-Reg. Ant-Miner-Reg was chosen to enable a fair comparison of pheromone models as both algorithms use the same IRL strategy to construct a list of rules and the same rule consequent generation methods. This ensures that any difference in performance can be attributed to the change in pheromone model.

The experiments were conducted using nineteen regression data-sets publicly available from the UCI Machine Learning Repository (Lichman 2013). Details of

Table 9.1: Number of instances and attribute makeup of the nineteen data-sets used in the experiments

		Attributes	
Name	Instances	Categorical	Continuous
WPBC_r	194	0	33
CPU	209	1	8
Yacht	308	0	7
MPG	410	2	5
Housing	452	1	13
Forest Fire	517	2	11
Istanbul	536	0	8
Efficiency	768	0	9
Stock	950	0	10
Concrete	1030	0	9
Flare	1066	10	1
Airfoil	1503	0	6
Red Wine	1599	0	12
Skill Craft	3338	0	20
Elevator	9517	0	7
CCPP	9568	0	5
Bike Share	17379	0	13
Energy Data	19735	0	25
Pm 25	41757	1	12

Table 9.2: Parameters: Ant-Miner-Reg$_{\text{MA}}$ uses the first three parameters in table, while remaining are used by both Ant-Miner-Reg$_{\text{MA}}$ and Ant-Miner-Reg.

Archive Parameters	Value
q	0.025495
ξ	0.6795
R	90
General Parameters	**Value**
Minimum Covered Instances	10
Max Uncovered Instances	10
Max Iterations	1500
Number of Ants	60
Stagnation Test (Number of Iterations)	10
α	0.59

the data-sets chosen are shown in Table 9.1, including the number of instances, the number of categorical attributes and the number of continuous attributes present in each data-set.

Table 9.2 contains the experimental parameters used for both Ant-Miner-Reg$_{\text{MA}}$ and Ant-Miner-Reg. Ant-Miner-Reg$_{\text{MA}}$ uses the first three parameters in Table 9.2 for the archive and the associated sampling procedures. The remaining parameters are used by both algorithms.

For each of the data-sets, 10-fold cross validation was performed with each algorithm being executed five time (varying the random initialisation seed) on each fold, for a total of fifty runs of each algorithms per data-set. The average RRMSE (Relative Root Mean Squared Error) was then calculated and presented in Table 9.3. Repeated runs and 10-fold cross validation was performed due to the stochastic nature of ACO algorithms, which causes variability in their performance. Performing many runs and averaging the results mitigates against these

differences.

As shown in Table 9.3, Ant-Miner-Reg$_{\text{MA}}$ shows an improvement in RRMSE compared to Ant-Miner-Reg, outperforming Ant-Miner-Reg in sixteen of the nineteen data-sets. Most notably, Ant-Miner-Reg$_{\text{MA}}$ improved the RRMSE by 80% in the Yacht data-set: Ant-Miner-Reg's RRMSE is 1.0120, while Ant-Miner-Reg$_{\text{MA}}$'s RRMSE is 0.2091. Based on our results, it is clear that the introduction of archive-based pheromone model in Ant-Miner-Reg$_{\text{MA}}$ resulted in an improvement in the model creation. Ant-Miner-Reg uses the M5 algorithm's dynamic discretisation procedure when creating terms for continuous attributes, while Ant-Miner-Reg$_{\text{MA}}$'s archive-based pheromone model is responsible for generating and improving the values chosen for the continuous attributes terms.

For statistical significance testing of the difference in RRMSE, we used Wilcoxon signed-rank test. The Wilcoxon signed-rank test is a non-parametric statistical test that makes no assumption that the samples are normally distributed. However it can only be used in a single comparison of a pair of algorithms for statistical significance. Unlike the Friedman statistical test used earlier in Chapter 6, the Wilcoxon signed-rank test takes into account the magnitude of the differences and not just the number of wins and losses for each algorithm (Wilcoxon 1945). The result of the statistical testing is shown in Table 9.4 which shows that Ant-Miner-Reg$_{\text{MA}}$ significantly outperforms Ant-Miner-Reg at the usual significance level of $\alpha = 0.05$.

In terms of computational time, Ant-Miner-Reg$_{\text{MA}}$ did not improve the run-time in comparison with Ant-Miner-Reg, as seen in Table 9.5. This is different than what was observed in classification problems, where the introduction of an archive-based pheromone model did significantly improve the run-time by eliminating the need for a discretisation procedure. Looking at the data-sets where Ant-Miner-Reg$_{\text{MA}}$'s run-time was significantly higher — Bike Share (17379 instances), Energy Data (19735 instances) and Pm 25 (41757 instances) — we noticed that

Table 9.3: Average RRMSE of the regression model produced by each algorithm over five runs of tenfold cross-validation. The best result (smallest RRMSE) for each data-set is shown in bold.

Dataset	Ant-Miner-Reg$_{MA}$	Ant-Miner-Reg
WPBC_r	1.0356 [0.0674]	**1.0224 [0.1074]**
CPU	**0.5038 [0.1734]**	0.8233 [0.2347]
Yacht	**0.2091 [0.0256]**	1.0120 [0.0448]
MPG	**0.5374 [0.0967]**	0.6419 [0.0377]
Housing	**0.5986 [0.0056]**	0.9782 [0.0125]
Forest Fire	1.5326 [0.3682]	**1.0334 [0.2481]**
Istanbul	**0.7948 [0.0360]**	0.8341 [0.0687]
Efficiency	**0.2348 [0.0358]**	0.4288 [0.0987]
Stock	**0.3258 [0.1357]**	0.7434 [0.1562]
Concrete	**0.7239 [0.0381]**	0.9636 [0.1267]
Flare	**0.9956 [0.0297]**	0.9987 [0.0831]
Airfoil	**0.8165 [0.0057]**	0.9715 [0.0119]
Red Wine	0.9898 [0.0674]	**0.9757** [0.0520]
Skill Craft	**0.8536 [0.0312]**	0.8912 [0.0100]
Elevator	**0.7585 [0.0214]**	0.7882 [0.0952]
CCPP	**0.3557 [0.0189]**	0.4769 [0.0317]
Bike Share	**0.6412 [0.1010]**	0.9941 [0.0968]
Energy Data	**0.9775 [0.2354]**	0.9971 [0.1586]
Pm 25	**0.9389 [0.3324]**	0.9982 [0.2896]

Table 9.4: Wilcoxon Signed-Rank test (at the $\alpha = 0.05$ significance level) on RRMSE. Statistically significant differences are shown in bold.

	Size	W+	W-	Z	p
RRMSE	18	23	167	-2.8974	**0.00374**

Ant-Miner-Reg produces very generalised rules with a RRMSE closer to the mean of the entire data-set (0.9941, 0.9971, and 0.9982 respectively), while Ant-Miner-Reg_{MA} produces more specific rules for those data-set with an improved RRMSE (0.6412, 0.9775, and 0.9389 respectively).

9.2.1 Archive based Pheromone Model ACO Discussion

We hypothesise that when the data-set is more complex, Ant-Miner-Reg struggles to find good split points and produces very simple, over-generalised rules that cover large sections of the search space. This can be seen in the RRMSE of models produced for large datasets, identifying a potential limitation of using M5's dynamic discretisation procedure to create regression rules. The dynamic discretisation procedures in classification and regression Ant-Miner algorithms operate differently. In regression, the dynamic discretisation procedure aims to find the optimal split point for a continuous attribute in the set of uncovered instances, without considering how other attributes will alter the final prediction the rule. This limits the interaction between the creation of condition and the rule's final consequent, which is unknown during rule creation. In classification, the dynamic discretisation procedure aims to find the optimal split point for an attribute in the set of uncovered instances taking into account the association between the attribute and a known target class, which improves the rule's prediction. The archive-based approach overcomes this difficulty as the values chosen for continuous attributes are optimised in conjunction with all attributes and not

Table 9.5: Average run-time in seconds of the model produced by each algorithm over five runs of tenfold cross-validation. The best result (smallest time) for each data-set is shown in bold.

Data-Set	Ant-Miner-Reg$_{MA}$	Ant-Miner-Reg
WPBC_r	0.51	**0.25**
CPU	**0.23**	0.46
Yacht	**0.20**	0.31
MPG	0.26	**0.24**
Housing	0.43	**0.26**
Forest Fire	0.99	**0.42**
Istanbul	0.39	**0.38**
Efficiency	0.70	**0.39**
Stock	0.85	**0.40**
Concrete	1.29	**0.44**
Flare	0.84	**0.25**
Air-foil	0.92	**0.57**
Red Wine	0.94	**0.61**
Skill Craft	10.12	**0.93**
Elevator	4.65	**2.28**
CCPP	**2.29**	12.22
Bike Share	223.52	**2.13**
Energy Data	582.38	**4.62**
Pm 25	615.24	**6.81**

in isolation.

Although Ant-Miner-Reg$_{\text{MA}}$ did not improve the run-time when compared to Ant-Miner-Reg, the improvement in RRMSE shows great promise for regression problems. This confirms the hypothesis that the archive-based pheromone model improves the values chosen for the continuous attributes of rule conditions in regression problems producing better rules with overall lower RRMSE. Table 9.4 shows that Ant-Miner-Reg$_{\text{MA}}$ achieved a statistically significant improvement with a value of $p = 0.00374$ with respect to Ant-Miner-Reg, using the Wilcoxon signed-rank test (at the $\alpha = 0.05$ level) on RRMSE.

Chapter 10

Conclusions and Future Work

The conclusions for this thesis have been split into two main sections. First, I will discuss the introduction of new ACO-based algorithms that tackle the regression problem and future directions this work could take. The second section will discuss the addition of monotonic constraints to ACO-based algorithms for both the classification and regression tasks.

10.1 ACO-Based Algorithms for Regression

A summary of the contributions to ACO algorithms applied to regression is presented in this section. The new ACO-based algorithms for regression rule induction proposed in this thesis, namely Ant-Miner-Reg, Ant-Miner-Reg$_{\mathrm{M}}$ and Ant-Miner-Reg$_{\mathrm{PB}}$, use a graph-based combinatorial approach to model construction, where the problem is represented as a construction graph. Individual ants traverse this graph selecting nodes to create solutions, guided by pheromone deposited on the graph's edges.

This thesis also explored the use of different learning strategies to create regression rules. The motivation is to explore different learning strategies to investigate whether increased rule interaction outweighed the increase in task complexity

found in more advanced learning strategies or not.

The first ACO-based algorithm proposed to construct regression rule lists was Ant-Miner-Reg. Ant-Miner-Reg uses an IRL approach to construct rule lists by breaking down the optimisation problem to smaller problems of constructing a single rule at a time. This algorithm was implemented with two different dynamic discretisation procedures: (1) an error reduction procedure and (2) a group merging procedure that attempts to cluster instances based on the target value. It was found that the error reduction procedure performed better in eleven out of the fifteen data-sets tested, showing that creating many different good potential split points as was done by the second procedure does not out weigh the benefit of finding a single best split point.

The IRL strategy only allows limited rule interactions, as each rule is created in isolation on the current uncovered training set. However, in the literature there are two other learning strategies, namely Michigan and Pittsburgh. Both deal with the problem of rule interaction in a different way. In a Michigan-based algorithm, a single rule is represented as a population member which in the case of ACO-based algorithms corresponds to an individual ant, limited rule interactions are allowed as the individual rules compete for conclusion into the final rule list in a niching operation. While a Pittsburgh approaches define a evolve-able unit as a complete list of rules, where the task for each ant would be to create this complete list of rules. Ant-Miner-Reg$_{\mathrm{M}}$ was developed as a Michigan-based algorithm, where the ant colony creates a colony of rules which then undergo a niching procedure to produce an ordered rule list. Ant-Miner-Reg$_{\mathrm{PB}}$ is a Pittsburgh-based approach that tasks individual ants to create an entire rule list with the addition of depth concept to the construction graph.

The three Ant-Miner-Reg variants were compared to each other, where it was found that the Pittsburgh strategy statistically outperformed both IRL and

Michigan-based algorithms. The algorithms were also compared against the well-known regression rule induction algorithm M5' Rules, where Ant-Miner-Reg$_{\text{PB}}$ achieved the best average rank of all the algorithms tested. When comparing the size of the rule lists created by the algorithms, it becomes clear that the additional performance obtained by Ant-Miner-Reg$_{\text{PB}}$ has increased the size of the models, this increased size may be explained by the different objectives of IRL and Pittsburgh-based algorithms. Where as an IRL algorithm aim to find the best rule that covers as many instances as possible, a Pittsburgh-based algorithm aim to find the best rule list and has less pressure to create single rules that cover the largest number of instances. These results show that a trade-off has to be made between decreasing the error of the models and their size.

While both IRL and Michigan learning strategies allow the optimisation of term interactions, they do not allow rule interactions to be optimised when rules are created. Although the Michigan strategy does allow limited rule interactions to be optimised during the niching procedure, it is only the Pittsburgh strategy that allows both term and rule interactions to be optimised together. The Pittsburgh strategy tackles the problem directly optimising the rule list in one step rather than optimising the creation of the best rule for each sub-problem (subset of the data). This is important as the best possible rule at each position in terms of rule coverage and error is not necessarily the correct rule in that position for the best rule list.

The comparison of the three different learning strategies using the same base ACO construction procedure is important as it allows a direct comparison of the strategies in the setting of ACO-based algorithms for the regression task, where in terms of predictive quality, the Pittsburgh strategy is the strongest. The proposed family of Ant-Miner-Reg algorithms are important, as to the best of my knowledge, they are the first regression rule induction algorithms that use an ACO procedure to create their rules.

10.1.1 Future Work

None of the ACO-based algorithms presented in this thesis make use of any heuristic information that can be generated from the training data-set when traversing the pheromone graph. Using heuristic information may allow the algorithms to find better rules and/or allow the algorithms to converge quicker towards good areas fo the search space, resulting in a faster execution time.

Currently the pruning procedures for all algorithms operate on a single rule and comprise of a simple backtrack pruner, where the last term in a rule is removed while an increase in quality is observed. Smarter pruning mechanisms could be developed that operate on entire rule lists, pruning terms from rules that reduce the error of the entire rule list. This will allow pruners to globally optimise the rule lists created with respect to rule interactions rather than the goal of optimising the current rule.

Furthermore, the predictions made by Ant-Miner-Reg and its derivatives correspond to a single value for all the instances that satisfy a rule's antecedent. An alternative is to modify the rule's consequent to that of a linear model, in a similar way to M5' Rules. This would allow each rule to be more powerful and predict different values for each covered instance. While this would decrease the interpretability of the rules,it could be balanced against the expected increase in predictive quality.

10.1.1.1 Archive Pheromone model

In Chapter 9, Ant-Miner-Reg$_{\text{MA}}$ was introduced as a new regression rule induction algorithm, where the traditional construction graph and corresponding pheromone matrix were replaced by a solution archive. The incorporation of a solution archive allows Ant-Miner-Reg$_{\text{MA}}$ to remove the need for a separate dynamic discretisation procedure, as continuous attributes are now optimised within the solution archive along with nominal attributes.

Ant-Miner-Reg$_{\text{MA}}$ was compared to Ant-Miner-Reg to allow a fair comparison of the two pheromone models, as the same IRL learning strategy was used to construct a rule list. The results of this comparison show that Ant-Miner-Reg$_{\text{MA}}$ achieves a smaller RRMSE in 84% of the data-sets tested, showing a statistically significant improvement in performance. In the classification task, it has been shown by Helal and Otero (2016) that the introduction of a solution archive significantly speeds up the execution time of the algorithm. When applied to regression rule induction, this speed up was not observed. It is hypothesised that this is due to the differences in dynamic discretisation procedures used in classification and regression tasks.

The replacement of the dynamic discretisation procedure is believed to be fundamental to the increase in performance between the two pheromone models. Unlike the classification task, where the target values are known, the dynamic discretisation procedures for regression problems do not know what predictions will be made by the rule, as these are not generated until the rule is completed. As a consequence, the discretisation procedure is not optimising continuous attribute split points to the value predicted since it does not have enough knowledge to which attributes are going to be selected in the future that may alter the target value. The introduction of a solution archive allows the incorporation and optimisation of continuous attributes directly into the rule construction process, i.e. the values chosen for continuous attributes can be optimised along with the selection of other attributes and the generation of the rule's prediction.

It should be noted that Ant-Miner-Reg$_{\text{MA}}$ uses the IRL learning strategy to construct ordered rule lists, in a one-rule-at-a-time fashion. However, experiments with learning strategies with algorithms that use the traditional graph-based pheromone model show an improvement to the accuracy of models when using the Pittsburgh-based learning strategy. Therefore, one potential extension to Ant-Miner-Reg$_{\text{MA}}$ is a transition to the Pittsburgh learning strategy to evaluate

whether an improvement in performance will be found compared to the original IRL algorithm and the graph-based algorithm Ant-Miner-Reg$_{\text{PB}}$ or not.

10.2 Monotonic Constraints in ACO Algorithms

Another major contribution of this thesis is the incorporation of monotonic constraints into ACO algorithms. Constraints have been implemented in two different stages of the learning process. Firstly, soft constraints have been added during model construction, where the quality function was modified to include a notion of non-monotonicity. This modified quality was used during pheromone update, allowing a preference for the creation of monotonic solutions. These soft constraints do not prevent the creation of non-monotonic rule lists, therefore, hard constraint enforcement is required. This was achieved by adding hard constraint pruners as a post-processing procedure that would guarantee the production of monotonic models.

10.2.1 Soft Constraint Enforcement

The adoption of soft constraints during model construction ensures that the ACO procedure can fully explore the search space when constructing rule lists. During soft constraint enforcement, a preference is shown for monotonic solutions. However, the algorithm is not constrained to the current monotonic regions and it can explore non-monotonic regions if they help produce high quality models. The exploration of non-monotonic regions may also result in finding new fully monotonic areas corresponding to monotonic solutions with a higher quality than the ones produced in the current best monotonic area of the search space.

These soft constraints are implemented by modifying the quality measure of the algorithms that penalises non-monotonic rule pairs present in the rule list. This modified quality is then used when pheromone levels are updating, influencing

term selection in subsequent iterations. The proposed quality measure uses the notion of NMI adapted to the context of regression (classification) rules, inspired by the work of Ben-David (1995) in decision trees. The NMI of a list of rules can be calculated by comparing each pair of rules in the list and checking for a violation, the number of violations are then summed and divided by the total number of rule pairs giving a violation index for the list of rules.

10.2.2 Hard Constraint Enforcement

Hard constraints were initially implemented as a naive backtrack pruner. When given a non-monotonic rule list, this naive pruner repeatedly removes the final term from the rule list until the entire rule list becomes monotonic. This will always be possible as when all but one rule remains the list will be monotonic as a rule cannot violate itself. This pruner is potentially very destructive if the violating pair is near the top of the rule list. To counter this, a pruning suite was proposed that three different pruners was proposed: (1) the original Naive Pruner; (2) the Most Violations Pruner (MVP), which prunes the rule that has the highest NMI; and (3) the Best Fix Pruner (BFP), which prunes the rule that will result in the biggest reduction in a list's NMI.

This pruning suite can then be applied in a post-processing phase to the model constructed in previous phases. Each of the pruners is applied to the rule list individually and the pruned list that achieves the highest quality on the training data is retained. As this pruning suite operates as a post-processing operation, it can be applied to any algorithm that produces a list of regression (classification) rules, or any model that can be converted into a list of rules. This enables any rule induction algorithm to become a monotonic rule induction algorithm.

10.2.3 Monotonic Constraint Experiments

Both soft and hard constraint enforcement has been combined into ACO-based algorithms that construct rule lists for both regression (Ant-Miner-Reg$_{\text{MC}}$, Ant-Miner-Reg$_{\text{PB+MC}}$ and Ant-Miner-Reg$_{\text{PB+MCP}}$) and classification (cAnt-Miner$_{\text{PB+MC}}$ and cAnt-Miner$_{\text{PB+MCP}}$) tasks.

On the regression task, Ant-Miner-Reg$_{\text{PB+MCP}}$ was shown to the be the best performing algorithm, significantly outperforming the base algorithm with a pruning suite as a post-processing step (Ant-Miner-Reg$_{\text{PB}}$ + Pruners). This shows that the combination of soft and hard constraints are able to work together to improve the quality of the rule lists produced, with the soft constraints creating rule lists that require less pruning than rule lists created by Ant-Miner-Reg$_{\text{PB}}$. The algorithms proposed in this thesis were also compared against the well-known classical rule induction algorithm M5' Rules with and without the pruning suite as a post processing step. Ant-Miner-Reg$_{\text{PB+MCP}}$ statistically outperformed both monotonic and non-monotonic variants.

For the classification task, a comparison between the additive post-processing monotonic technique used by RULEM and the subtractive techniques employed by the proposed pruning suite could be compared. Both post-processing techniques were applied to the base cAnt-Miner$_{\text{PB+MC}}$ algorithm, where it was shown that the pruning suite statistically outperformed the algorithms using RULEM. These two post-processing techniques were applied to the existing algorithms JRip and C5.0 Rules, and compared to cAnt-Miner$_{\text{PB+MCP}}$. The results of these experiments showed that applying a post-processing hard constraint procedure may harm the accuracy of the models produced, as has been suggested previously by Ben-David, Sterling and Tran (2009), since JRip and C5.0 predictive accuracy decreases.

The results obtained for both the classification and regression tasks show that a two step soft-hard constraint approach is more successful than applying a single hard constraint post-processing step. While post-processing procedures can be

applied to any algorithm that produces a compatible model, they can suffer from a reduction in performance. Many algorithms can be modified to alter the quality used during model construction, although ACO-based algorithms lend themselves to this approach as they allow the soft constraints to be continually reinforced over many iterations, slowly refining the models produced while performing a global search.

10.2.4 Future Work

Currently, only monotonic features that increase (or decrease) across the entire attribute domain are considered as constraint candidates. However, many other constraint types exist (as presented in Figure 3.1), which could be implemented and enforced using the proposed two-step methodology. One candidate constraint type to investigate would be piece-wise monotonic constraints, where an attribute may change between monotonically increasing to monotonically decreasing (or vice versa), allowing the modelling of more complex relationships. Another possible constraint type that could be of interest is a preference, where a monotonic constraint is only enforced when another attribute has a particular value.

Another direction of future work could involve the introduction of multi-objective methods during the model creation phase. Currently, in both regression and classification tasks, the soft constraints have been implemented during the learning phase using a weighted formula that balances the monotonicity of a model against its quality. This balance can be modified by the user before execution, however the optimum value for each data-set is unknown. This tuning parameter could be removed by the implementation of a multi-objective optimisation procedure where the ant colony creates a non-dominated Pareto front (Coello et al. 2007) of solutions based on the quality and monotonicity of a rule list. After the front has been generated, the pruning suite presented earlier can be applied to each rule list. This removes one of the objectives as all the rule lists will be

monotonic, at which point they can be ranked by quality on the training set and the best model presented to the user.

Another area of future work would include the incorporation of constraints into the archive-based ACO algorithm Ant-Miner-Reg_{MA}, introduced in Appendix 9. The archive-based ACO algorithm was shown to increase performance compared to the traditional combinatorial graph-based algorithm when using the IRL learning strategy. It can therefore be expected to produce better monotonic rule lists and it is a future research direction worth investigating.

Also, the investigation into the effects of using noisier constraints where the input data has more violations between a constrained attribute and the target attribute. This work has concentrated on selecting the attribute with the least amount of noise and the effect of noise on algorithmic performance requires further investigation. We would expect that the two phase approach outlined in Chapter 7 would be more robust than a post-processing only technique.

Finally, the new algorithms presented here concentrate on a single constraint relationship. However, in reality there are often many monotonic relationships. These relationships may interact with each other and will decrease the available search space. The introduction of multiple constraints on algorithmic performance would be an interesting direction for further work.

Bibliography

Albinati, J., Oliveira, S. E., Otero, F. E. and Pappa, G. L. (2015). An ant colony-based semi-supervised approach for learning classification rules. *Swarm Intelligence*, 9(4), pp. 315–341.

Augusto, D. and Barbosa, H. (2000). Symbolic regression via genetic programming. In *Proceedings of the Sixth Brazilian Symposium on Neural Networks*, IEEE, pp. 173–178.

Bacardit, J. and Butz, M. V. (2007). Data mining in learning classifier systems: Comparing xcs with gassist. In *Learning Classifier Systems*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 282–290.

Barros, R. C., Basgalupp, M. P., De Carvalho, A. C. and Freitas, A. A. (2012). A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3), pp. 291–312.

Ben-David, A. (1995). Monotonicity maintenance in information-theoretic machine learning algorithms. *Machine Learning*, 19, pp. 29–43.

Ben-David, A., Sterling, L. and Pao, Y.-H. (1989). Learning and classification of monotonic ordinal concepts. *Computational Intelligence*, 5(1), pp. 45–49.

Ben-David, A., Sterling, L. and Tran, T. (2009). Adding monotonicity to learning

algorithms may impair their accuracy. *Expert Systems with Applications*, 36, pp. 6627–6634.

Bibal, A. and Frénay, B. (2016). Interpretability of machine learning models and representations: an introduction. In *24th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pp. 77–82.

Bloomfield, P. and Steiger, W. (1980). Least absolute deviations curve-fitting. *SIAM Journal on scientific and statistical computing*, 1(2), pp. 290–301.

Bologna, G. and Hayashi, Y. (2018). A comparison study on rule extraction from neural network ensembles, boosted shallow trees, and SVMs. *Applied Computational Intelligence and Soft Computing*, 2018, pp. 1–20.

Bonabeau, E. et al. (1999). *Swarm intelligence: from natural to artificial systems*. 1, Oxford University Press.

Booker, L. B. (1982). *Intelligent behavior as an adaptation to the task environment*. PhD Thesis, Dept Electic. Eng. Comput. Sci., University of Michigan.

Booker, L. B., Goldberg, D. E. and Holland, J. H. (1989). Classifier systems and genetic algorithms. *Artificial intelligence*, 40(1-3), pp. 235–282.

Boser, B. E., Guyon, I. M. and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, ACM, pp. 144–152.

Breiman, L., Friedman, J., Stone, C. and Olshen, R. (1984). *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series, Taylor & Francis.

Brookhouse, J. and Otero, F. E. (2018). Post-processing methods to enforce monotonic constraints in ant colony classification algorithms. In *2018 International Joint Conference on Neural Networks*, pp. 1–8.

Brookhouse, J. and Otero, F. E. B. (2015). Discovering regression rules with ant colony optimization. In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference (GECCO 2015)*, pp. 1005–1012.

Brookhouse, J. and Otero, F. E. B. (2016a). Monotonicity in ant colony classification algorithms. In *10th International Conference on Swarm Intelligence (ANTS 2016)*, Springer, pp. 137–148.

Brookhouse, J. and Otero, F. E. B. (2016b). Using an ant colony optimization algorithm for monotonic regression rule discovery. In *Genetic and Evolutionary Computation Conference (GECCO 2016)*, ACM Press, pp. 437–444.

Cao-Van, K. (2003). *Supervised ranking: from semantics to algorithms*. PhD Thesis, Ghent University.

Chen, C.-C. and Li, S.-T. (2014). Credit rating with a monotonicity-constrained support vector machine model. *Expert Systems with Applications*, 41(16), pp. 7235–7247.

Coello, C. A. C., Lamont, G. B., Van Veldhuizen, D. A. et al. (2007). *Evolutionary algorithms for solving multi-objective problems*, vol. 5. Springer.

Cohen, W. W. (1995). Fast effective rule induction. In *Twelfth International Conference on Machine Learning*, Morgan Kaufmann, pp. 115–123.

Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C. (2009). *Introduction to algorithms*. MIT press.

Cortez, P. and Morais, A. d. J. R. (2007). A data mining approach to predict forest fires using meteorological data. In *Proceedings of the 13th EPIA 2007*, Associação Portuguesa para a Inteligência Artificial (APPIA), pp. 512–523.

Croes, G. A. (1958). A method for solving traveling-salesman problems. *Operations research*, 6(6), pp. 791–812.

Daniels, H. and Velikova, M. (2006). Derivation of montone decision models from noisy data. *IEEE Transactions on Systems, Man, and Cybernetics*, 36(5), pp. 705–710.

Deneubourg, J. et al. (1992). The dynamics of collective sorting: Robot-like ants and ant-like robots. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pp. 353–363.

Deneubourg, J.-L., Pasteels, J. M. and Verhaeghe, J.-C. (1983). Probabilistic behaviour in ants: a strategy of errors? *Journal of Theoretical biology*, 105(2), pp. 259–271.

Deneubourg, J.-L., Aron, S., Goss, S. and Pasteels, J. M. (1990). The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior*, 3(2), pp. 159–168.

Dorigo, M. (1992). *Optimization, learning and natural algorithms*. PhD Thesis, Politecnico di Milano.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1), pp. 29–41.

Dorigo, M. and Stutzle, T. (2004). *Ant Colony Optimization*. A Bradford Book.

Duivesteijn, W. and Feelders, A. (2008). Nearest neighbour classification with monotonicity constraints. In *Machine Learning and Knowledge Discovery in Databases*, vol. 5211 5211, Springer, pp. 301–316.

Fahrmeir, L., Kneib, T., Lang, S. and Marx, B. (2013). *Regression: Models, Methods and Applications*. Springer.

Fayyad, U., Piatetsky-Shapiro, G. and Smith, P. (1996). From data mining to knowledge discovery: an overview. In U. Fayyad, G. Piatetsky-Shapiro, P. Smith and R. Uthurusamy, eds., *Advances in Knowledge Discovery & Data Mining*, MIT Press, pp. 1–34.

Feelders, A. (2000). Prior knowledge in economic applications of data mining. In *Principles of Data Mining and Knowledge Discovery*, *Lecture Notes in Computer Science*, vol. 1910, Springer, pp. 395–400.

Feelders, A. and Pardoel, M. (2003). Pruning for monotone classification trees. In *Advances in intelligent data analysis V*, Springer, pp. 1–12.

Fernández-Navarro, F., Riccardi, A. and Carloni, S. (2014). Ordinal neural networks without iterative tuning. *IEEE Transactions on Neural Networks and Learning Systems*, 25(11), pp. 2075–2085.

Frank, E. and Witten, I. H. (1998). Generating accurate rule sets without global optimization. In J. Shavlik, ed., *Fifteenth International Conference on Machine Learning*, Morgan Kaufmann, pp. 144–151.

Frank, E., Wang, Y., Inglis, S., Holmes, G. and Witten, I. H. (1998). Using model trees for classification. *Machine learning*, 32(1), pp. 63–76.

Frawley, W. J., Piatetsky-Shapiro, G. and Matheus, C. J. (1992). Knowledge discovery in databases: An overview. *AI magazine*, 13(3), pp. 57–57.

Freitas, A. (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer.

Freitas, A. (2014). Comprehensible classification models: a position paper. *ACM SIGKDD Explorations Newsletter*, 15(1), pp. 1–10.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200), pp. 675–701.

Fürnkranz, J. and Widmer, G. (1994). Incremental reduced error pruning. In *Machine Learning Proceedings 1994*, Elsevier, pp. 70–77.

González, S., Herrera, F. and García, S. (2015). Monotonic random forest with an ensemble pruning mechanism based on the degree of monotonicity. *New Generation Computing*, 33(4), pp. 367–388.

Goss, S., Aron, S., Deneubourg, J.-L. and Pasteels, J. M. (1989). Self-organized shortcuts in the argentine ant. *Naturwissenschaften*, 76(12), pp. 579–581.

Helal, A., Brookhouse, J. and Otero, F. E. (2018). Archive-based pheromone model for discovering regression rules with ant colony optimization. In *2018 IEEE Congress on Evolutionary Computation*, pp. 1–7.

Helal, A. and Otero, F. E. (2016). A mixed-attribute approach in ant-miner classification rule discovery algorithm. In *Genetic and Evolutionary Computation Conference (GECCO 2016)*, ACM Press, pp. 13–20.

Holmes, G., Hall, M. and Frank, E. (1999). Generating rule sets from model trees. In *Proceedings 12th Australian Joint Conference on Artificial Intelligence*, Springer, pp. 1–12.

Hoover, K. and Perez, S. (2000). Three attitudes towards data mining. *Journal of Economic Methodology*, 7(2), pp. 195–210.

Janssen, F. and Fürnkranz, J. (2010a). Seperate-and-conquer regression. In *Proceedings of the German Workshop on Lernen*, pp. 81–89.

Janssen, F. and Fürnkranz, J. (2010b). Seperate-and-conquer regression. Tech. Rep. TUD-KE-2010-01, Knowledge Engineering Group, Technische Universitat Darmstadt.

Koza, J. R. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2), pp. 87–112.

Kubat, M., Holte, R. C. and Matwin, S. (1998). Machine learning for the detection of oil spills in satellite radar images. *Machine learning*, 30(2-3), pp. 195–215.

Liao, T., Socha, K., Montes de Oca, M., Stutzle, T. and Dorigo, M. (2014). Ant colony optimization for mixed-variable optimization problems. *IEEE Transactions on Evolutionary Computation*, 18(4), pp. 503–518.

Lichman, M. (2013). UCI machine learning repository. `http://archive.ics.uci.edu/ml`, University of California, Irvine, School of Information and Computer Sciences.

Lievens, S., De Baets, B. and Cao-Van, K. (2008). A probabilistic framework for the design of instance-based supervised ranking algorithms in an ordinal setting. *Annals of Operations Research*, 163(1), pp. 115–142.

Liu, H. and Cocea, M. (2018). Induction of classification rules by gini-index based rule generation. *Information Sciences*, 436, pp. 227–246.

Liu, Y.-C. and Yeh, I.-C. (2017). Using mixture design and neural networks to build stock selection decision support systems. *Neural Computing and Applications*, 28(3), pp. 521–535.

Manderick, B. and Moyson, F. (1988). *The collective behavior of ants: An example of self-organization in massive parallelism*. Proceedings of the AAAI Spring Symposium on Parallel Models of Intelligence, Stanford University, CA.

Mangasarian, O. L., Street, W. N. and Wolberg, W. H. (1995). Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4), pp. 570–577.

Martens, D. and Baesens, B. (2010). Building acceptable classification models. In *Data Mining*, Springer, pp. 53–74.

Martens, D., Baesens, B. and Fawcett, T. (2011). Editorial survey: swarm intelligence for data mining. *Machine Learning*, 82(1), pp. 1–42.

Martens, D., Baesens, B. and Van Gestel, T. (2009). Decompositional rule extraction from support vector machines by active learning. *IEEE Transactions on Knowledge and Data Engineering*, 21(2), pp. 178–191.

Martens, D. et al. (2006a). Ant-based approach to the knowledge fusion problem. In *Ant Colony Optimization and Swarm Intelligence*, Springer, pp. 84–95.

Martens, D., De Backer, M., Haesen, R., Baesens, B. and Holvoet, T. (2006b). Ants constructing rule-based classifiers. In *Swarm intelligence in data mining*, Springer, pp. 21–43.

Martens, D. et al. (2007). Classification with ant colony optimiztion. *IEEE Transactions on Evolutionary Computation*, 11(5), pp. 651–665.

Mavrovouniotis, M., Müller, F. M. and Yang, S. (2017). Ant colony optimization with local search for dynamic traveling salesman problems. *IEEE transactions on cybernetics*, 47(7), pp. 1743–1756.

Minnaert, B. and Martens, D. (2012). Towards a particle swarm optimization-based regression rule miner. In *Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on*, pp. 961–963.

Minnaert, B., Martens, D., De Backer, M. and Baesens, B. (2015). To tune or not to tune: rule evaluation for metaheuristic-based sequential covering algorithms. *Data mining and knowledge discovery*, 29(1), pp. 237–272.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill Science.

Montgomery, D. C., Peck, E. A. and Vining, G. G. (2012). *Introduction to linear regression analysis*, vol. 821. John Wiley & Sons.

Olmo, J. L., Romero, J. R. and Ventura, S. (2010). A grammar based ant programming algorithm for mining classification rules. In *IEEE Congress on Evolutionary Computation*, pp. 1–8.

Otero, F. and Freitas, A. (2013). Improving the interpretability of classification rules discovered by an ant colony algorithm. In *2013 Genetic and Evolutionary Computation Conference (GECCO 13)*, pp. 73–80.

Otero, F., Freitas, A. and Johnson, C. (2008). cant-miner: An ant colony classification algorithm to cope with continuous attributes. In *Ant Colony Optimization and Swarm Intelligence (Proc. ANTS 2008)*, pp. 48–59.

Otero, F., Freitas, A. and Johnson, C. (2009). Handling continuous attributes in ant colony classification algorithms. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Data Mining (CIDM 2009)*, IEEE, pp. 225–231.

Otero, F., Freitas, A. and Johnson, C. (2012). Inducing decision trees with an ant colony optimization algorithm. *Applied Soft Computing*, 12, pp. 3615–3626.

Otero, F., Freitas, A. and Johnson, C. (2013). A new sequential covering strategy for inducing classification rules with ant colony algorithms. *IEEE Transactions on Evolutionary Computation*, 17(1), pp. 64–76.

Otero, F. E. (2017). Myra: a java ant colony optimization framework for classification algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ACM, pp. 1247–1254.

Parpinelli, R., Lopes, H. and Freitas, A. (2002). Data mining with an ant colony optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 6(4), pp. 321–332.

Potharst, R., Ben-David, A. and van Wezel, M. (2009). Two algorithms for generating structured and unstructured monotone ordinal data sets. *Engineering Applications of Artificial Intelligence*, 22(4), pp. 491–496.

Qian, Y., Xu, H., Liang, J., Liu, B. and Wang, J. (2015). Fusing monotonic decision trees. *Knowledge and Data Engineering, IEEE Transactions on*, 27(10), pp. 2717–2728.

Quinlan, J. (1992). Learning with continuous classes. In *Proceedings 5th Australian Joint Conference on Artificial Intelligence*, World Scientific, pp. 343–348.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine learning*, 5(3), pp. 239–266.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Quinlan, J. R. (1996). Improved use of continuous attributes in c4. 5. *Journal of artificial intelligence research*, 4, pp. 77–90.

Russell, S. J. and Norvig, P. (2016). *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61, pp. 85–117.

Smith, S. F. (1980). *A learning system based on genetic adaptive algorithms*. PhD Thesis, Dept. Comput. Sci., University of Pittsburgh.

Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'83, pp. 422–425.

Socha, K. and Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3), pp. 1155 – 1173.

Specht, D. (1991). A general regression neural network. *Neural Networks, IEEE Transactions on*, 2(6), pp. 568–576.

Stützle, T. and Hoos, H. (1999). The max-min ant system and local search for combinatorial optimization problems. In *Meta-heuristics*, Springer, pp. 313–329.

Stützle, T. and Hoos, H. H. (2000). Max–min ant system. *Future generation computer systems*, 16(8), pp. 889–914.

Uy, N., Hoai, N., O'Neil, M., McKay, R. and Galván-López, E. (2011). Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines*, 12(2), pp. 91–119.

Van De Kamp, R., Feelders, A. and Barile, N. (2009). Isotonic classification trees. In *International Symposium on Intelligent Data Analysis*, Springer, pp. 405–416.

Vapnik, V. (2013). *The nature of statistical learning theory*. Springer science & business media.

Verbeke, W., Martens, D. and Baesens, B. (2017). Rulem: A novel heuristic rule learning approach for ordinal classification with monotonicity constraints. *Applied Soft Computing*, 60, pp. 858–873.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6), pp. 80–83.

Witten, I. H., Frank, E., Hall, M. A. and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques 4th Ed.* Morgan Kaufmann.

Wu, X. et al. (2008). Top 10 algorithms in data mining. *Knowledge Information Systems*, 14, pp. 1–37.

Yu, B. and Xu, Z.-b. (2008). A comparative study for content-based dynamic spam classification using four machine learning algorithms. *Knowledge-Based Systems*, 21(4), pp. 355–362.

Appendix A

Copyright License

Attribution-NonCommercial-NoDerivatives 4.0 International

==========

Creative Commons Corporation ("Creative Commons") is not a law firm and does not provide legal services or legal advice. Distribution of Creative Commons public licenses does not create a lawyer-client or other relationship. Creative Commons makes its licenses and related information available on an "as-is" basis. Creative Commons gives no warranties regarding its licenses, any material licensed under their terms and conditions, or any related information. Creative Commons disclaims all liability for damages resulting from their use to the fullest extent possible.

Using Creative Commons Public Licenses

Creative Commons public licenses provide a standard set of terms and conditions that creators and other rights holders may use to share

original works of authorship and other material subject to copyright and certain other rights specified in the public license below. The following considerations are for informational purposes only, are not exhaustive, and do not form part of our licenses.

> Considerations for licensors: Our public licenses are intended for use by those authorized to give the public permission to use material in ways otherwise restricted by copyright and certain other rights. Our licenses are irrevocable. Licensors should read and understand the terms and conditions of the license they choose before applying it. Licensors should also secure all rights necessary before applying our licenses so that the public can reuse the material as expected. Licensors should clearly mark any material not subject to the license. This includes other CC-licensed material, or material used under an exception or limitation to copyright. More considerations for licensors: wiki.creativecommons.org/Considerations_for_licensors

> Considerations for the public: By using one of our public licenses, a licensor grants the public permission to use the licensed material under specified terms and conditions. If the licensor's permission is not necessary for any reason--for example, because of any applicable exception or limitation to copyright--then that use is not regulated by the license. Our licenses grant only permissions under copyright and certain other rights that a licensor has authority to grant. Use of the licensed material may still be restricted for other

and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.

b. Copyright and Similar Rights means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.

c. Effective Technological Measures means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.

d. Exceptions and Limitations means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.

e. Licensed Material means the artistic or literary work, database, or other material to which the Licensor applied this Public

License.

f. Licensed Rights means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.

g. Licensor means the individual(s) or entity(ies) granting rights under this Public License.

h. NonCommercial means not primarily intended for or directed towards commercial advantage or monetary compensation. For purposes of this Public License, the exchange of the Licensed Material for other material subject to Copyright and Similar Rights by digital file-sharing or similar means is NonCommercial provided there is no payment of monetary compensation in connection with the exchange.

i. Share means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.

j. Sui Generis Database Rights means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of

the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.

k. You means the individual or entity exercising the Licensed Rights under this Public License. Your has a corresponding meaning.

Section 2 -- Scope.

a. License grant.

1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:

a. reproduce and Share the Licensed Material, in whole or in part, for NonCommercial purposes only; and

b. produce and reproduce, but not Share, Adapted Material for NonCommercial purposes only.

2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.

3. Term. The term of this Public License is specified in Section 6(a).

4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.

    a. Offer from the Licensor -- Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.

    b. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed

Material.

6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. Other rights.

1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.

2. Patent and trademark rights are not licensed under this Public License.

3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties, including when

the Licensed Material is used other than for NonCommercial purposes.

Section 3 -- License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. Attribution.

1. If You Share the Licensed Material, You must:

a. retain the following if it is supplied by the Licensor with the Licensed Material:

i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);

ii. a copyright notice;

iii. a notice that refers to this Public License;

iv. a notice that refers to the disclaimer of warranties;

v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;

b. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and

c. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.

For the avoidance of doubt, You do not have permission under this Public License to Share Adapted Material.

2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.

3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.

Section 4 -- Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database for NonCommercial purposes only and provided You do not Share Adapted Material;

b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and

c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 -- Disclaimer of Warranties and Limitation of Liability.

a. UNLESS OTHERWISE SEPARATELY UNDERTAKEN BY THE LICENSOR, TO THE EXTENT POSSIBLE, THE LICENSOR OFFERS THE LICENSED MATERIAL AS-IS AND AS-AVAILABLE, AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE LICENSED MATERIAL, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHER. THIS INCLUDES, WITHOUT LIMITATION,

WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OR ABSENCE OF ERRORS, WHETHER OR NOT KNOWN OR DISCOVERABLE. WHERE DISCLAIMERS OF WARRANTIES ARE NOT ALLOWED IN FULL OR IN PART, THIS DISCLAIMER MAY NOT APPLY TO YOU.

b. TO THE EXTENT POSSIBLE, IN NO EVENT WILL THE LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY (INCLUDING, WITHOUT LIMITATION, NEGLIGENCE) OR OTHERWISE FOR ANY DIRECT, SPECIAL, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, EXEMPLARY, OR OTHER LOSSES, COSTS, EXPENSES, OR DAMAGES ARISING OUT OF THIS PUBLIC LICENSE OR USE OF THE LICENSED MATERIAL, EVEN IF THE LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES, COSTS, EXPENSES, OR DAMAGES. WHERE A LIMITATION OF LIABILITY IS NOT ALLOWED IN FULL OR IN PART, THIS LIMITATION MAY NOT APPLY TO YOU.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 -- Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.

b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:

   1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or

   2. upon express reinstatement by the Licensor.

   For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.

d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 -- Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.

b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 -- Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.

b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.

c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.

d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.

==========

Creative Commons is not a party to its public licenses.
Notwithstanding, Creative Commons may elect to apply one of its public licenses to material it publishes and in those instances will be considered the "Licensor." Except for the limited purpose of indicating that material is shared under a Creative Commons public license or as otherwise permitted by the Creative Commons policies published at creativecommons.org/policies, Creative Commons does not authorize the use of the trademark "Creative Commons" or any other trademark or logo of Creative Commons without its prior written consent including, without limitation, in connection with any unauthorized modifications to any of its public licenses or any other arrangements, understandings, or agreements concerning use of licensed material. For the avoidance of doubt, this paragraph does not form part of the public licenses.

Creative Commons may be contacted at creativecommons.org.