NEW ARCHIVE-BASED ANT COLONY OPTIMIZATION ALGORITHMS FOR LEARNING PREDICTIVE RULES FROM DATA

A THESIS SUBMITTED TO

THE UNIVERSITY OF KENT

IN THE SUBJECT OF COMPUTER SCIENCE

FOR THE DEGREE

OF PHD.

By

Ayah Helal

August 2019

Copyright

Acknowledgements

I would like to thank a lot of people, who supported me through my Ph.D. and helped me to navigate this interesting journey. I came from Egypt to do my Ph.D. in the UK, not imagining that I would actually start my life here. I want to start by thanking my husband, James, for giving me a home and family here. Also, I would like to thank my supervisors, Dr. Fernando Otero, and Prof. Alex Freitas for their help and support through the ups and downs in my research and writing this thesis. I want to also thank my examiners for their comments and input.

I want to thank the School of Computing department in Medway for helping me through my Ph.D and giving me amazing friends. I would like to thank Sam for the endless chats and distractions we had in the office. He also was one of the reasons I joined the exclusive lunch club where I got to know my future husband. I would like to thank Janine and Shannon for being more than administrators. They did not just support my nervous breakdown in their offices, but also they became my friends. I would like to thank Anna for being a great friend. I would like to thank our friendly retired police officer, Mick, who never fails to compliment me every morning to make my day brighter.

I want to give a special thanks to my family back home. I will never forget that my inspiration for doing a Ph.D. came from seeing my mother do hers, with my dad's support, when I was young. I couldn't have done it without you. I want to thank my brother, my friends (Enas and Menna), and my family in Egypt for always being there for me, even if they are continents away.

Abstract

Data mining is the process of extracting knowledge and patterns from data. Classification and Regression are among the major data mining tasks, where the goal is to predict a value of an attribute of interest for each data instance, given the values of a set of predictive attributes. Most classification and regression problems involve continuous, ordinal and categorical attributes. Currently Ant Colony Optimization (ACO) algorithms have focused on directly handling categorical attributes only; continuous attributes are transformed using a discretisation procedure in either a preprocessing stage or dynamically during the rule creation.

The use of a discretisation procedure has several limitations: (i) it increases the computational runtime, since several candidates values need to evaluated; (ii) requires access to the entire attribute domain, which in some applications all data is not available; (iii) the values used to create discrete intervals are not optimised in combination with the values of other attributes.

This thesis investigates the use of solution archive pheromone model, based on Ant Colony Optimization for mixed-variable ($\text{ACO}_{\mathbf{MV}}$) algorithm, to directly cope with all attribute types. Firstly, an archive-based ACO classification algorithm is presented, followed by an automatic design framework to generate new configuration of ACO algorithms. Then, we addressed the challenging problem of mining data streams, presenting a new ACO algorithm in combination with a hybrid pheromone model. Finally, the archive-based approach is extended to cope with regression problems.

All algorithms presented are compared against well-known algorithms from the literature using publicly available data sets. Our results have been shown to improve the computational time while maintaining a competitive predictive performance.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

The amount of data available for analyses has exponentially increased as a result of recent technological advances — nowadays data regarding ad clicks, shares and likes from social media platforms, user analytics from mobile apps and web searches are easily available[1]. The analysis of data has the potential to uncover useful knowledge and be an important type of support for better decision making. Data mining is the research field concerned with the design of algorithms that (semi-)automate data analysis by employing methods mainly from the areas of machine learning and statistics (Fayyad, Piatetsky-Shapiro and Smyth 1996; Piateski and Frawley 1991).

There are two broad types of data mining tasks from a machine learning perspective: *supervised* and *unsupervised*. Supervised tasks include classification and regression, where the aim of the task is to build a predictive model representing the relationship between input variables (predictive attributes, or features) and a known output (target) attribute. Unsupervised tasks do not involve any output variable and do not involve prediction (i.e., they use only input variables). Hence, unsupervised tasks are descriptive, rather than predictive. An example of unsupervised task is clustering, where the instances are grouped into clusters based on

[1] Based on the information provided by https://www.domo.com/learn/data-never-sleeps-7

their similarity, so that descriptions of the different clusters can be used to highlight differences and similarities between different groups of objects (instances). In this thesis, we will focus on the supervised tasks of classification and regression.

Data mining models can be divided into *white-box* or *black-box* models based on the type of knowledge representation that they use. The term white-box refers to interpretable models, where the model can be understood by the user when making a decision - e.g., models based on decision trees, and IF-THEN classification/regression rules. It is important to note that not all decision trees/rule sets are interpretable models, as for example a decision tree with many thousand, or a rule set with many thousands of rules will not interpretable. The term black-box refers to models whose inner working are not easily understood by the user - e.g., models based on artificial neural networks and support vector machines. The benefit of white-box models is that they help with expert acceptance, since their prediction can be potentially validated/interpreted by experts, unless the models are too large for a user's interpretation. In many domains, explaining the predictions made by a model is a requirement (Freitas 2014; Freitas, Wieser and Apweiler 2010; Pazzani, Mani and Shankle 2001).

Data stream mining is the process of extracting knowledge from a continuous flow of data (Muthukrishnan 2003) — i.e., data instances are continuously generated. As a result, a data stream needs to be processed without access to the whole data, and each data instance is processed only once or a small number of times. According to (Krempl et al. 2014), challenges in handling data streams are volume, velocity and volatility. Data stream volume (can be considered rate) incrementally increases from zero to infinite making it infeasible to store all data. Velocity impacts the mining process since data arrive quickly and continuously, limiting available processing time. Volatility is the concept of drift and change of patterns, target, and/or variables of the data being mined, which indicate the need for an algorithm to deal with a dynamically changing environment.

There are several data stream applications as shown by (Nguyen, Woon and Ng 2015). Mining query streams to provide users with better search results has attracted much research work (Zeng et al. 2004; Chien and Immorlica 2005; Mundhe and Manwade 2018). Network monitoring is a popular application area, where the analysis of traffic data can be used to discover usage patterns and unusual activities in real time (Muthukrishnan 2003; Andreoni Lopez et al. 2019). Sensor networks are involved in real-life applications such as traffic monitoring, smart homes, habitat monitoring and healthcare (Gama and Gaber 2007; Sow et al. 2010; ur Rehman et al. 2016). Social networks are becoming more and more popular, generating tremendous amounts of online data streams (Aggarwal and Philip 2005; Aggarwal and Subbian 2012; Cui et al. 2011; Sakaki, Okazaki and Matsuo 2010; Sun, Faloutsos and Papadimitriou 2007; Chen and Shang 2019; Kudo et al. 2019).

Krempl et al. (2014) highlighted the need to create simpler models, considering not only predictive accuracy, but also the interpretability of the knowledge discovered by data stream algorithms. Interpretability was one of the recommendations based on the study of real-world applications and the shortcomings of the existing approaches. Notably, current rule induction algorithms in the field follow an incremental approach (Gama and Kosina 2011; Stahl, Gaber and Salvador 2012; Le et al. 2014, 2017), which leads to large and difficult to interpret models. Ensemble approaches are shown to be successful in data stream classification (Minku and Yao 2012; Baena-Garcıa et al. 2006; Street and Kim 2001; Almeida, Kosina and Gama 2013), but an ensemble architecture increases the complexity of the models produced.

Several evolutionary approaches have been successfully applied to handle data stream classification. Vivekanandan and Nedunchezhian (2011) proposed an online genetic algorithm (OGA), an incremental rule learning algorithm that creates a rule set for data stream classification with concept drift. Vahdat et al. (2014a) proposed a GP for streaming data classification tasks with label budgets, where the GP learns a model using a limited number of labelled instances.

Classification or regression problems can be viewed as optimization problems, where the aim is to create the best model to represent the predictive patterns in the data. Many metaheuristics have been applied to create classification/regression models, including evolutionary and swarm intelligence algorithms (Vivekanandan and Nedunchezhian 2011; Vahdat et al. 2014a; Cervantes et al. 2013) — Ant Colony Optimization (ACO) (Dorigo and Stützle 2004) is amongst the most successful ones. ACO is a metaheuristic inspired by the behaviour of real ant colonies. Many ant species, despite the simplicity of their individual behaviour and lack of centralised control, are able to find the shortest path between a food source and their nest. To find such shortest paths, ants cooperate via an indirect communication mechanism by means of pheromone deposit. The path with the highest concentration of pheromone, which usually corresponds to the shortest path since ants can traverse it quicker, is preferred. ACO algorithms simulate this behaviour to find optimal or near optimal solutions for optimization problems. Most ACO algorithm convert the optimisation problem to a shortest path problem, where ants uses a graph pheromone model to deposit and traverse the solution space. Recently a different pheromone model was proposed in (Socha and Dorigo 2008; Liao et al. 2014), where rather than using a graph model to guide the ants, this approach uses an archive based pheromone model.

ACO algorithms have been successfully applied to classification problems in (Parpinelli, Lopes and Freitas 2002; Otero, Freitas and Johnson 2008, 2009, 2013; Liang et al. 2016; Yang et al. 2017; Seidlova, Poživil and Seidl 2019; Al-Behadili, Ku-Mahamud and Sagban 2018), where the use of pheromone allows the algorithm to explore the search space effectively to build accurate models. The majority of ACO-based classification algorithms are limited to cope only with categorical attributes, while continuous attributes are discretised in a pre-processing step or dynamically during the model creation.

Data stream mining is also a growing research area in terms of open source software frameworks. Massive Online Analysis (MOA) (Bifet et al. 2010) was the first framework for data stream mining. MOA includes a collection of machine learning algorithms—such as classification, regression, clustering, outlier detection, concept drift detection and recommender systems—as well as stream generators and evaluation measures. MOA is based on the well-known Waikato Environment for Knowledge Analysis (WEKA) data mining tool (Hall et al. 2009; Witten et al. 2016) and its goal is to provide a benchmark suit for the growing data stream mining community.

1.1 Overview of the Contributions of the Thesis

The focus of this thesis is to extend the current approach of ACO-based classification mining algorithms to handle mixed-variables problems, in particular improving how ACO-based algorithms handle continuous attributes. The aim is to improve the overall computational time without reducing predictive performance, allowing ACO-based algorithms to handle large data sets and the more challenging problem of mining data streams.

The first contribution is a new ACO classification algorithms for mixed-variable problems, using a pheromone model that efficiently handle continuous attributes; which eliminates the need for a discretisation procedure in ACO rule induction (creation) algorithms by using an archive-based pheromone model capable to cope with continuous attributes directly and faster. We also propose an automatic design framework to incorporate the graph-based model along with the archive-based model during the rule creation process.

The second contribution is an ACO data stream algorithm that creates classification rules; it uses a novel hybrid pheromone model which combines the archive and graph models to handle continuous attributes directly without the need for a discretisation procedure. The approach uses a Pittsburgh learning strategy to allow for rule interactions. The approach shows significant improvement in the model size compared to well-known rule induction algorithms in data stream mining, without any negative impact on predictive accuracy.

The third contribution is an initial work on an ACO regression algorithm improving the way the algorithm handles continuous attributes; using an archive pheromone model. The approach shows improvement in the computation runtime, without negative impact on predictive accuracy.

1.2 Thesis Structure

The remainder of this thesis is organised as follows. Chapter 2 presents an overview of data mining and rule induction algorithms. Chapter 3 presents an in-depth analysis of data stream mining algorithms. Chapter 4 presents an overview of ACO algorithms and ACO-based classification and regression algorithms.

Chapter 5 presents the first contribution of the thesis, namely a new ACO classification algorithm for mixed-variable problems and the results of the algorithm.

Chapter 6 present the automatic design framework for combining graph-based and archive-based ACO approaches during the rule creation process, and the results of the algorithm.

Moreover, Chapter 7 presents an ACO data stream mining algorithm, which combines and integrates the archive and graph pheromone models to learn rule lists using a Pittsburgh-based approach. The results of the new algorithm are presented in Chapter 8.

Chapter 9 presents a inital work in applying the archive-based pheromone model to regression problems and the computational results.

Chapter 10 presents conclusions and final remarks of the thesis, as well as suggestions for future research.

1.3 Publications

The list of publications from the research described in this thesis is as follows:

- Helal, A. and Otero, F. E. (2016). A Mixed-Attribute Approach in Ant-Miner Classification Rule Discovery Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ACM, GECCO '16, pp. 13–20.

- Helal, A. and Otero, F. E. B. (2017). Automatic Design of Ant-Miner Mixed Attributes for Classification Rule Discovery. In *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM Press, GECCO '17, pp. 433–440.

- Helal, A., Brookhouse, J. and Otero, F. E. B. (2018). Archive-Based Pheromone Model for Discovering Regression Rules with Ant Colony Optimization. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, pp. 1–7.

- Helal, A. and Otero, F. E. B. (2019). Data Stream Classification with Ant Colony Optimization, submitted to IEEE transactions on Evolutionary Computation, under review.

Chapter 2

Data Mining

In the era of big data, vast amounts of information are stored in large data repositories (Fan and Bifet 2013). The process of analysing and extracting useful knowledge from data is known as Knowledge Discovery in Database (KDD) (Fayyad, Piatetsky-Shapiro and Smyth 1996). KDD steps involve collecting and selecting data, preprocessing and data cleaning, transformation, data mining, interpretation and evaluation. These steps are illustrated in Figure 1. It starts with raw data, which undergo selection to produce target data; target data then undergo pre-processing to transform the data for the data mining step; data mining produces patterns that are interpreted into knowledge.

Data mining is the central step in the KDD process, which aims at discovering patterns in data and/or relationships between data attributes (Holmes, Donkin and Witten 1994). Supervised and unsupervised are the two main learning approaches in data mining. In supervised learning, the data is labelled, meaning

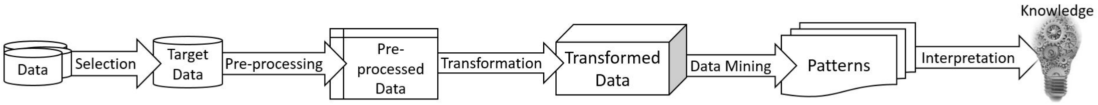


Figure 1: The main steps in Knowledge Discovery in Databases (adapted from (Fayyad, Piatetsky-Shapiro and Smyth 1996)).

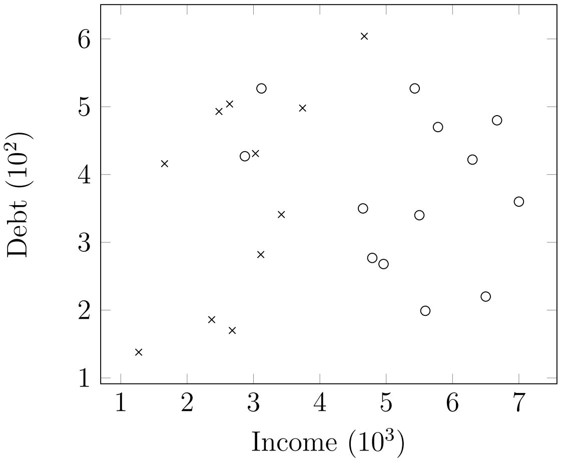
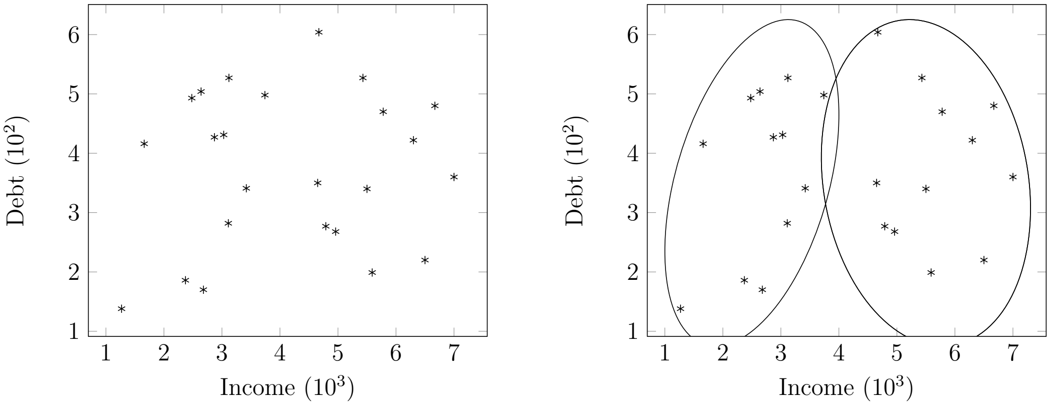


Figure 2: Example of a data set with two classes: x's represent the class that defaulted on their loans; and the o's represent the class that is in good status with the bank. The values in the debt axis are multiplied by (10^2) and the values in the income axis are multiplied by (10^3).

that a target value is known by the learning method. In unsupervised learning, the data is unlabelled, meaning that there is not a target value specified to the learning method. Figure 2 illustrates a simple artificial bank data set. Each data point represents an instance of a person who has taken a loan and the axis represent the attributes 'Debt' and 'Income'. The 38 instances are divided into two groups (classes): the x's represent the class that defaulted on their loans and the o's represent the class that is in good status with the bank.

Unsupervised learning focuses on understanding the data and visualizing the relationships between the data attributes. One of the main types of descriptive methods is clustering. Clustering algorithms partition the data into clusters, where each cluster consists of data points that are similar to each other and different data points belong to different clusters (Jain and Dubes 1988). Clustering algorithms are an unsupervised learning technique, where the data used is not labelled, as illustrated in Figure 3(a); a potential output of clustering is illustrated in Figure 3(b), where the data is divided into 2 clusters.



(a)

(b)

Figure 3: In (a) an example of an unlabelled data data set; (b) potential output of a clustering algorithm, where the data is grouped into 2 clusters.

A well-known clustering algorithm is K-means, formalised and named by MacQueen (1967). K-means partitions the data into a user-defined number of clusters k. The algorithm starts by randomly initializing k centroids in the solution space, where each centroid defines the central point of a cluster. Then it calculates the distance between each data points and the centroids. Each data point is assigned to the cluster with the nearest centroid, then the position for each centroid is recalculate based on the data points assigned to it by centralizing the centroid position. This iterative process continues until centroids positions do not change or a maximum number of iterations is reached.

Supervised learning focuses on predicting the value of a target (class) attribute for previously unseen data points. Two popular predictive tasks are classification and regression. Classification problems involve the prediction of class values from a finite set of values, such as identifying loan applicants having credit risks or not (Weiss and Kulikowski 1991). Regression problems involve the prediction of a real-valued target value, such as income (Fahrmeir et al. 2013).

In recent years, data mining started to tackle richer data formats rather than traditional stationary data (Maimon and Rokach 2010), such as data streams (Gaber, Zaslavsky and Krishnaswamy 2007), spatio-temporal (Kisilevich et al.

2009) and multimedia data (Zhang and Zhang 2008). A data stream consists of a continuous generation of data that is unbounded on time. Examples of data streams are stock market data, social media and sensor networks. Spatio-temporal data are generated from location-based or environmental devices that record position, time and environmental properties of objects. Multimedia data have complex data structure, such as images, sounds and music.

In the following sections, we will discuss classification and regression tasks in more detail, since they are the focus of our work, and present examples of well-know approaches for these tasks.

2.1 Classification

The classification task is concerned with finding patterns in data, then use those patterns to classify any new (future) data instance (Weiss and Kulikowski 1991). Each data instance is described by a set of predictor attributes, whose values are used to predict the value of a designated class (target) attribute. There are mainly three types of predictor attributes: continuous, ordinal and (unordered) categorical. A continuous attribute takes numeric real values, where each instance has potentially a different value; categorical attributes have a finite set of values, where each instance has one of the available values; and ordinal attributes also have a finite set of values, but the values are ordered (e.g., Small, Medium, Large). Integer attributes are usually treated as continuous attributes in machine learning. Classification algorithms produce a model based on predictive relationships between the set of predictor attributes and the class attribute, which represent the patterns found in the data. In the process of creating a model, the data is divided into training and test sets to create and evaluate a model, respectively. Since classification is a supervised learning task, classification algorithm has the information (value) of the class attribute of every instance in the training set.

Table 1: Data set for income classification.

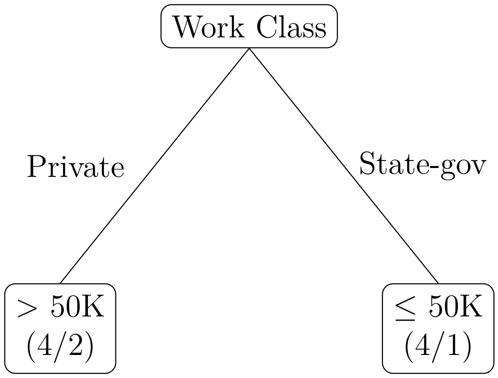
Age	Gender	Work Class	Income (Class)
39	Male	State-gov	> 50K
34	Male	Private	> 50K
52	Male	Private	≤ 50K
42	Male	State-gov	≤ 50K
59	Female	State-gov	≤ 50K
25	Female	Private	> 50K
66	Female	Private	≤ 50K
32	Female	State-gov	≤ 50K
57	Female	State-gov	≤ 50K
25	Male	Private	> 50K
36	Female	Private	> 50K

The model generated by a classification algorithm is then used to predict the class value of the instances in the test data, where the class value is not given to the algorithm. The prediction is compared to the actual class value to determine the model's quality.

Table 1 shows a small data set regarding income classification. The first attribute "Age" is a continuous attribute; the second and third attributes, "Gender" {Male, Female} and "Work Class" {Private, State-gov}, are both categorical attributes; the class attribute is the "Income" {≤ 50K, > 50K}.

Classification models can be divided into white or black box. White box models refer to interpretable models, where the model provides an explanation for the classification of an instance (or data point) and they can be used to understand the problem at hand. Black box models are models whose inner workings are not interpretable, the prediction cannot be easily explained.

The need for interpretable models arises from the need to understand/justify why a certain prediction was made (Freitas 2014; Doshi-Velez and Kim 2017), given that in several real-world applications it is not enough to get the prediction (e.g., medical diagnosis, credit scoring). In such applications, the model needs to explain how it came to the prediction, because a correct prediction does not



(a)

IF Work Class = Private *THEN* > 50K

IF Work Class = State-gov *THEN* ≤ 50K

(b)

Figure 4: An illustration of classification models: (a) decision tree with root attribute "Work Class" and two branches { "Private" , "State-gov" } leading to leaf nodes. The number of instances that are correctly/incorrectly classified by each leaf node is shown in brackets, respectively. (b) decision rules representing the same model in (a).

include the needed explanation to justify the prediction. A wide range of algorithms have been used to create classification models (Wu et al. 2008), including white box approaches: e.g., decision trees, and rule induction; and black box approaches: e.g., artificial neural networks and support vector machines, among others. In this section, we focus only on decision trees and classification rules, since both are popular forms of interpretable classification models.

In decision trees (Quinlan 1986) internal nodes represent attributes and each branch originating from a node represents a different test on an attribute's value; leaf nodes are associated with a predicted class value. Decision trees are usually built using a divide-and-conquer approach. It starts by splitting the data into subsets, which are then split into even smaller subsets, and so on until this iterative process stops when each of the subsets of data is homogeneous (with all its instances belonging to the same class) or another stopping criteria has been met. At the root node, the entire data is represented. Next a decision tree algorithm chooses an attribute to split the data, creating a decision (internal) node; ideally it chooses an attribute that divides the data into homogeneous subsets. Working down each branch, the algorithm continues to partition the data choosing the best attribute to make another decision node. Each attribute in the data set is used in at most one node. Decision trees can continue to grow indefinitely, dividing the data into smaller and smaller portions until each instance is perfectly classified or all attributes are used. However, this could lead to overfitting the training data, which results in poor predictive performance on unseen data (Mingers 1989; Malerba, Esposito and Semeraro 1996). One solution to mitigate the effects of growing a large tree is pruning, where pruning removes leaf nodes to reduce the size of the tree as long as the overall quality of the tree is not decreased.

Figure 4(a) illustrates a decision tree for the data presented in Table 1, which consist of a root node "Work Class" represented by a categorical attribute with two possible values {"Private","State-gov"}. Each value is represented by a branch leading to a leaf node. To calculate the value to be predicted by a leaf node, a common approach is to return the most frequent (majority) value of the class attribute amongs the instances that reach the leaf node. In the case of the leaf node "Private", the prediction value is "> 50K", since 4 instances (2,6,10,11) have the value "> 50K" and 2 instances (3,7) have the value "≤ 50K". The number of instances correctly/incorrectly classified is shown under the predicted class value, respectively.

Decision trees can be easily converted into a set of decision rules, where rules are represented as *IF-THEN* statements. The *IF* part consist of attribute-based conditions (also called antecedent) and the *THEN* part consist of the predicted class value. An instance satisfying all conditions of a rule antecedent is covered by the rule and has the value of the consequent as its predicted class value. Rule induction will be covered in Section 2.3. The decision tree in Figure 4(a) can be converted into a set of *IF-THEN* rules (Figure 4(4b)) by following each path from the root node to a leaf node, resulting in two rules: "*IF* Work Class = Private *THEN* > 50K" and "*IF* Work Class = State-gov *THEN* ≤ 50K". Converting the trees into rules allow for improved readability, as rules tend to be easier to understand (Mitchell 1997; Freitas 2014).

An important characteristic of classification models is their capacity in classifying unseen instances. One of the used measures to evaluate a classification model in this thesis is accuracy, given by:

$$Q = \frac{\#CorrectlyClassifiedInstances}{\#TotalInstances} \tag{1}$$

where the #CorrectlyClassifiedInstances is the number of instances where the class value predicted by the model is the same as the actual class value and #TotalInstances is the total number of instances whose class is predicted by the model. For example, the accuracy of the decision tree presented in Figure 4(a) when classifying the data in Table 1 is 73% (8 correct instances of a total of 11).

2.2 Regression

The regression task consists of building a model to predict a continuous target attribute value. Similar to classification algorithms, regression algorithms are supervised learning techniques, where the training data used is labelled.

The first difference between regression and classification tasks can be identified by looking at the income data in Table 2 and comparing it to the data in Table 1. In classification, the class attribute is represented by a categorical attribute with two values {≤50K, >50K} while in regression the target attribute is represented by a continuous attribute.

Similarly to classification, a wide range of algorithms have been used to tackle regression problems (Wu et al. 2008) such as: decision trees, rule induction, linear regression, artificial neural networks and support vector regression.

Table 2: Data set for income regression.

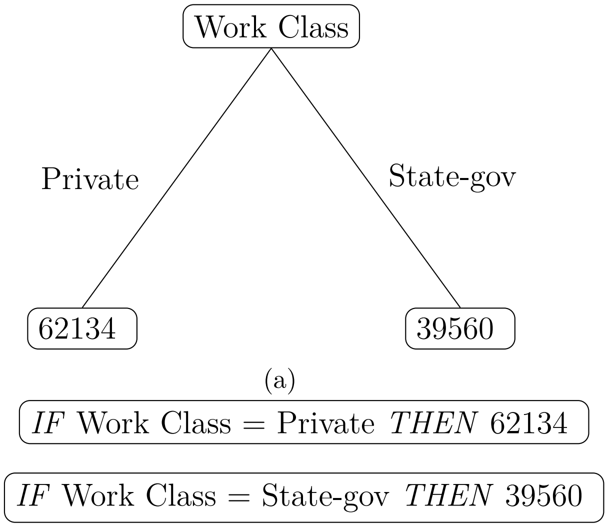
Age	Gender	Work Class	Income (target)
39	Male	State-gov	45150
34	Male	Private	34580
52	Male	Private	63691
42	Male	State-gov	45796
59	Female	State-gov	36169
25	Female	Private	85849
66	Female	Private	37066
32	Female	State-gov	39837
57	Female	State-gov	30845
25	Male	Private	77649
36	Female	Private	73969

A linear model is a standard regression technique (Press et al. 1992). Let x_i be the attribute value vector for the instance i and y_i its predicted value, a linear regression algorithm creates a function in the form:

$$y_i = \beta_0 + \epsilon + \sum_{j=1}^{n} \beta_j x_{ij} \tag{2}$$

where β_0 is the intercept (i.e, β_0 is the point were the line crosses the y axis); n is the number of predictor attributes; β_j are the linear coefficients for the predictor attributes; x_{ij} are the j-th predictor attribute values for the i-th instance, and ϵ is the error associated with each measurement. Linear regression is restrictive in modelling relationships since it tries to fit a linear relationship between attributes and predicted values. More advanced linear regression algorithms would use non-linear functions to represent attributes such as $x' = \frac{1}{x}$ to better fit the problem (Fahrmeir et al. 2013), in case a straight line does not fit the problem.

Figure 5 shows two interpretable representations of a simple regression model based on the data from Table 2. Figure 5(a) presents a regression tree model, which consists of the attribute “Work Class” as root node. “Work Class” is a categorical attribute with two possible values: “Private” and “State-gov”. Each



(a)

IF Work Class = Private *THEN* 62134

IF Work Class = State-gov *THEN* 39560

(b)

Figure 5: (a) An example of a regression tree with root attribute "Work Class", the values of that attribute are "Private" or "State-gov" and the leaf has the target attribute. The predicted value in the leaf is the average of the numeric target values of the instances covered in this subset. (b) A decision rules model representing the same model.

value is represented by a branch leading to a leaf node. To calculate the value to be predicted by a leaf node, a common approach is to calculate the average value of the target attribute among the instances that reach the leaf node. In the case of "Work Class = Private", the value predicted is 62134 (average target value of the instances 2,3,6,7,10,11). Similarly to classification, a regression tree can be converted to a rule model as shown in Figure 5(b). A more advanced tree model would use linear models in the leaf node to make predictions, which in some cases improve the overall quality of the regression model (Quinlan et al. 1992), while decreasing its interpretability.

One of the main measures to determine the quality of a regression model is the Root Mean Square Error (RMSE) (Barnston 1992), given by:

$$RMSE = \sqrt{\frac{1}{m} \cdot \sum_{i=1}^{m} (y_i - \bar{y}_i)^2} \tag{3}$$

where m is the total number of instances, $y_{i]}$ is the actual value of the i-th instance and $\bar{y_i}$ is the predicted value of the i-th instance. RMSE can measure the quality of regression models and individual regression rules. To calculate the RMSE for a regression rule, only the instances that are covered by the rule are considered. For example, the rule "*IF* Work Class = Private *THEN* 62134" covers 6 instances (2,3,6,7,10,11), the value of the Mean Square Error (MSE) of instance 2 predicted as 62134, is given by:

$$(y_2 - \bar{y_2})^2 = (34580 - 62134)^2 = 759,222,916 \tag{4}$$

The MSE values of all the instances covered by the rules are calculated, as given:

$$\begin{gathered}
(y_2 - \bar{y_2})^2 = 759,222,916 \\
(y_3 - \bar{y_3})^2 = 2,424,249 \\
(y_6 - \bar{y_6})^2 = 562,401,225 \\
(y_7 - \bar{y_7})^2 = 628,404,624 \\
(y_{10} - \bar{y_{10}})^2 = 240,715,225 \\
(y_{11} - \bar{y_{11}})^2 = 140,067,225 \\
\sum_{i=1}^{m}(y_i - \bar{y_i})^2 = 2,333,235,464
\end{gathered} \tag{5}$$

using the Equation 3, The quality of the rule is:

$$\begin{gathered}
RMSE = \sqrt{\frac{1}{m} \cdot \sum_{i=1}^{m}(y_i - \bar{y_i})^2} \\
RMSE = \sqrt{\frac{1}{6} \cdot 2,333,235,464} = 19719.9
\end{gathered} \tag{6}$$

2.3 Rule Learning

In this section, we will discuss three learning approaches to build rule models. Rule models can be represented as either rule sets (unordered rules) or rule lists (ordered rules), also known as decision lists. That is, rule sets do not impose any order on the rules, while rule lists apply a strict order on the rules. In a rule list, the order of rules plays an important role in determining the prediction of individual rules. When using a rule list to classify an instance, each rule is tested sequentially until a rule that covers the instance is found (i.e., all the attribute conditions in the rule antecedent are satisfied by the instance). This means that a rule is only used if the previous rules do not cover the instance. In the case of a rule set, all rules attempt to cover the instance. If only one rule covers the instance, the rule classifies the instance; if multiple rules cover the instance, a conflict resolution criterion is used to decide the final classification of the instance.

Michigan rule learning (Booker, Goldberg and Holland 1989) is well known learning approach, where the system learns all the rules at once. In our work we do not use Michigan learning, so we will not discussed further.

We will discuses two different rule induction approaches from the broader area of evolutionary algorithms (Freitas 2002; Alcalá-Fdez et al. 2009), each employing a different strategy to create a rule model.

2.3.1 Iterative Rule Learning

Iterative Rule Learning (IRL), also known as sequential covering (Mitchell 1997), is based on the idea that the problem of creating a rule list or rule set can be divided into a set of smaller problems consisting in creating a single rule. Starting with the complete data, a single rule is created and the instances covered by the rule are removed. The rule is added to the rule model and the procedure to create a single rule (LearnOneRule) is then repeated until a user-defined maximum number of instances are left uncovered (Threshold). Algorithm 1 presents the high-level pseudocode of the IRL approach. Since each iteration of the procedure uses a different set of instances, given that the covered instances are removed, a different rule will be created.

Algorithm 1: High-level pseudocode of Iterative rule learning.

```
  Data: Instances
  Result: Rule Model
1 RuleModel ← {}
2 while |Instances| > Threshold do
3     Rule ← LearnOneRule(Instances)
4     RuleModel ← RuleModel.addrule(Rule)
5     Instances ← Instances − Covered(Rule, Instances)
6 end
7 RuleModel ← RuleModel.addrule($Rule_{Default}$)
8 return RuleModel
```

Note that IRL can create either a set or list of rules, depending on the covered procedure used (line 5). If the covered procedure removes only the instances that are correctly classified by a rule (the prediction of the rule is the same as the instance class value), the final rule model (line 7) will be a rule set. If the covered procedure removes all instances covered by a rule, regardless if they are correctly classified or not, the final rule model will be a rule list. After the loop (lines 2-6) ended, a default rule is added to the rule model. A default rule is normally a rule with an empty antecedent, such that it will make a prediction to any instance that is not covered by the rules in the list or set. The use of a default rule ensures that the rule-based model will always make a prediction. There number of polices to decide on the default rule prediction one of them is the majority class among the uncovered instances, while in regression it is the mean value of the class attribute among the uncovered instances.

LearnOneRule can be implemented using different strategies and it is usually the main point where IRL algorithms differentiate. A greedy strategy would learn the most accurate rule, meaning it will focus on selecting whatever attribute seems best at the moment and then solve the subproblems that arise later. Usually a greedy strategy will fall into a local minima, since it does not cope well with attribute interaction (Freitas 2001). Attributes interaction exists between two attributes when the joint effect of both attributes in a model is different from that obtained by additively combining the individual effects (Chanda et al. 2009). A global strategy, such as evolutionary or swarm intelligence algorithms, allows the creation of strategies that choose other attributes (not just the best), coping better with attribute interaction.

2.3.2 Pittsburgh Rule Learning

Pittsburgh rule learning is based on the idea of learning a complete rule model using a single procedure (Smith 1983). In the context of swarm intellgence, an individual is a complete solution, and the algorithm tries to improve the complete solution as whole. A high-level pseudocode is shown in Algorithm 2, where the algorithm looks like IRL except that the LearnOneRule procedure is replaced by LearnOneRuleModel in order to create an entire rule model rather than just a single rule. The algorithm starts by creating an empty rule model, and while the algorithm has not reached the *Max Number* of iterations, LearnOneRuleModel creates a new rule model.

Algorithm 2: High-level pseudocode of Pittsburgh rule learning.

```
Data: Instances
Result: Rule Model
RuleModel$_{best}$ ← {}
while $i <$ Max Iterations do
    RuleModel ← LearnOneRuleModel
    if Quality(RuleModel) > Quality(RuleModel$_{best}$) then
        RuleModel$_{best}$ ← RuleModel
    end
end
return RuleModel$_{best}$
```

In the LearnOneRuleModel procedure, the algorithm can learn either a set or list of rules, determined by the LearnOneRuleModel procedure. Pittsburgh-based algorithms do not focus on the quality of a single rule, therefore, the selection of the final model is based on the quality of a complete model (line 4). This strategy allows the optimisation of rule interactions rather than just attribute interactions, as done by previous strategies.

2.4 An Overview of Classical Classification Algorithms

In this section, we will discuss some of the classical classification algorithms (Wu et al. 2008). Classification And Regression Trees (CART) (Breiman et al. 1984) creates binary trees, where each node contains a logical condition that evaluates to either a true or false value. CART uses the Gini diversity index to select the attribute to create a node; the Gini index provides an indication of how "pure" or "homogeneous" (with respect to their class) the instances in the leaf nodes are. CART is also able to use categorical and continuous attributes as target variables during tree construction, allowing the creation of classification or regression models, respectively.

Another well known classification algorithm that produces trees models is C5.0 (see5), which is a predecessor of (C4.5 and ID3) (Quinlan 1986, 1993). The algorithm follows a divide-and-conquer strategy to create decision trees. It uses the entropy of the class distribution (entropy is a measure of impurity in data set) to identify good splits for internal nodes by calculating the information gain ratio (measures how much information a feature gives us about the class) of attributes in the training data. Moreover, the algorithm allows for two or more outcomes from an internal tree node.

Quinlan (1987) proposed a transformation approach to convert decision trees created by C4.5 into rules and simplify the rules to create a rule set. The simplification procedure removes conditions that are satisfied by the fewest number of instances in the training set until at least one condition remains or the rule's quality deteriorates. In this case, pruning rules rather than the decision trees gives two advantages. First, it allows the removal of attribute conditions that affect a single rule. In the case of decision trees, the removal of any attribute affect its subtree. Second, it allows the removal of attribute conditions that occur at the top of the tree including the root node. In a decision tree, it is not possible to remove only the root node. After the simplification, if multiple rules classify the same instance, the approach uses the rule with the best accuracy to classify the instance.

Cendrowska (1987) presented the PRISM algorithm. The first difference between PRISM and the above algorithm is that it does not create rules using decision trees. PRISM starts by dividing the data into subsets according to the class attribute values and creates rules for each subset. In order to create rules, PRISM selects the attribute-value pair with the highest frequency of occurrence in the subset of instances being considered; it continues selecting attribute-value pairs until the subset of instances is homogeneous with respect to the class.

Frank and Witten (1998) proposed the PART algorithm, a rule induction algorithm that produces a rule list. The algorithm obtains rules from partial decision trees, using a sequential covering strategy. To build the rule, the algorithm first creates a pruned decision tree; then a rule is extracted from the tree based on the best leaf node and added to the rule list.

Support Vector Machines (SVMs) (Vapnik 1995) are one of the most robust and accurate methods in data mining (Wu et al. 2008). SVMs are referred to as black box models, since the processes that transform the input to output are difficult to understand by domain specialists compared to decision trees and classification rules. In a two-class learning task, the aim of SVMs is to find a hyperplane that maximises the margin between the two classes in order to generalize its prediction for future data. SVMs can be extended to handle multi-class problems by repeatedly using one class as the positive class and the remaining ones as negative to create multiple separating hyperplanes.

Ensemble learning (Hansen and Salamon 1990; Dietterich 1997) is based on the idea of building a predictive model by integrating multiple models, where each of the models solves the same original problem, and combining their output (predictions) to create a more robust model. Some approaches combine the prediction of different models, by means of a majority vote. One of the classical ensemble algorithms is AdaBoost (Schapire 1990; Freund and Schapire 1995). AdaBoost starts by assigning an equal weight to each instance in the training set. It then builds a classification model. Then, it updates the weights of the instances, where it increases the weights for misclassified instances and decreases the weights for correctly classified instances. Using the updated weights of instances, it builds a new classification model and adds it to the ensemble. It repeats the process of updating the weights and creating new models until a pre-set number of models have been created or no further improvement can be made on the training set by adding new models.

2.5 An Overview of Classical Regression Algorithm

As explained previously, regression aims to predict a real value rather than a categorical value as in classification. A classical decision tree regression algorithm is CART, which can be used for both regression and classification (Breiman et al. 1984). CART, when applied to regression problems, uses the Least Squares or Least Absolute Deviation (LAD) measure to select attributes for its internal nodes. LAD is the sum of absolute errors in each leaf of the tree, calculated as:

$$LAD = \sum_{i=1}^{n} \mid y_i - f(x_i) \mid \tag{7}$$

where x_i is the attribute-value vector representing the i-th instance, $f(x_i)$ is the predicted value for the i-th instance, y_i is the actual value of the target attribute for i-th instance, and n is the number of the instances in the current leaf node.

Quinlan et al. (1992) proposed the decision tree regression algorithm M5. The main difference between CART and M5 trees is that M5 uses linear models in the leaf nodes, rather than a single value to make a prediction. M5 uses a standard deviation split point generation method to choose attributes, which attempts to maximise the reduction in error of the predicted target value in a subset of the instances. Once the tree is grown and each branch is terminated with a leaf node, 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 rooted at the node. The linear model produced by M5 is not a model that uses all the available attributes, but it is simplified to use only the attributes present in the sub-tree that will potentially be pruned.

Janssen and Fürnkranz (2010b) proposed the Separate-and-Conquer Regression (SeCoReg) algorithm, which employs the commonly used sequential covering strategy to create a list of regression rules. SeCoReg uses a greedy search to learn one regression rule at a time, considering all possible conditions (attribute, operator, value) and adds the best possible condition to improve the quality of the rule.

Holmes, Hall and Prank (1999) proposed the M5'Rules algorithm, which is a wrapper for the M5 decision tree regression algorithm. The algorithm uses a sequential covering strategy to create a set of rules. It starts creating a full regression tree using the current training instances. Then, the best leaf in the tree is converted to a rule and added to the rule model. The correctly covered instances are then removed from the current training instances and the process is repeated. Similarly to M5 trees, M5'Rules uses linear models to make predictions.

2.6 Summary

In this chapter, we discussed two of the main data mining supervised learning tasks: classification and regression. In supervised learning tasks, the data is labelled, i.e., there is a target attribute and the data mining algorithm has access to its value during it's training. Both classification and regression aim to find patterns in the data to predict the value of the target attribute for unseen (future) data. Classification aims to predict a categorical value, while regression aims to predict a continuous value.

Furthermore, this chapter discussed the differences between black and white box models and presented in detail strategies to create two types of while box (interpretable) models: decision trees and rules. We focused on decision tree and rules since our aim is to build interpretable models. Finally, an overview of classical classification and regression algorithms was presented.

Chapter 3

Data Streams

In data streams, data arrives in rapid and continuous form. Data stream mining is the area concerned with extracting information from an incoming stream of data (Muthukrishnan 2003). While traditional data mining usually runs in off-line mode, characterised by slow data generation and where data storage is feasible (Fayyad, Piatetsky-Shapiro and Smyth 1996), data stream mining runs in real-time mode with rapid data generation where data storage is not feasible.

The main properties of data streams are volume, velocity and volatility—these properties present challenges in handling data streams (Krempl et al. 2014). As data streams volume incrementally increases from zero to infinity, data stream mining approaches need to incorporate data in an incremental form without storing all data. Velocity impacts the mining process, preventing the use of any off-line or time consuming procedure, due to the fact that data arrives in high velocity. Volatility is the change of patterns, target, and/or features of the data being mined, which require continuous updates of the model learned from the data. Volatility is also called concept drift. An example of volatility is the behaviour of customers in an on-line shop, where the prediction of how profitable a week is will differ with the increase of advertising and brand loyalty over time. These challenges and related work addressing them are discussed next.

3.1 Volume

Data stream cannot store all data in memory, therefore algorithms are restricted to storing small summaries of data stream. Moreover, keeping past elements could be harmful to the model since the distribution of data can change over time.

Most data stream processing make use of summarisation techniques to handle its large volume. Those techniques are used to produce approximate representations of large data sets, usually by selecting a subset of the data.

Random sampling is one of the most common techniques to reduce the data size (Motwani and Raghavan 1995). The difficulty of using sampling in context of data stream is the unknown data size. Additionally, the use of sampling is associated with poor detection of concept drifts (Gaber, Zaslavsky and Krishnaswamy 2005), since not all data is examined. Vitter (1985) presents a reservoir sampling algorithm, which keeps n number of data in memory, and with each new data instance arriving, it uses a probability to replace an old instance in memory. Chaudhuri, Motwani and Narasayya (1999) extended the reservoir sampling algorithm to weight the sampling where the weights are used to change the probability of replacing the instances. The weighting reservoir sampling was further extended by Guha et al. (2000) to apply the weighted apporach on clustring data streams.

Sketching involves building a summary of the data stream using a small amount of memory. Alon, Matias and Szegedy (1999) presented *frequency moments*, which capture the statistics of the data stream distribution in linear space. The statistics captured include the length of a sequence, the number of distinct values in a sequence, and the most frequent item multiplicity. Several approaches to estimate different *frequency moments* have been proposed by Babcock et al. (2002).

Synopsis data are any data structures substantively smaller than their original data, such as histograms and wavelets. Histograms are summary structures capable of aggregating the distribution of the attributes in a data set, where equal-width histograms aggregate data distributions to instance counts for equally wide data ranges. The most common types of histograms for data streams are V-optimal histograms (Jagadish et al. 1998), equal-width histograms (Greenwald and Khanna 2001), and end-based histogram (Fang et al. 1998).

Wavelets are used to approximate the data distribution with a given probability. Wavelet coefficients project a given signal into an orthogonal set of basis vectors. Research in data stream models with wavelet coefficients are discussed by Gilbert et al. (2002). The limitation of synopsis data is that it does not represent all the characteristics of the data sets (Gaber, Zaslavsky and Krishnaswamy 2005).

Another summarisation technique is the Hoeffding bound framework (Hoeffding 1963). The Hoeffding bound is used in data stream algorithms to obtain confidence bounds on the mean of a distribution based on a small subset of the distribution. This characteristic fits well with data stream scenario, where the distribution of the data is unknown. Therefore, the estimation of the bound is independent of the probability distribution generating the data. The price of this generality is that the bound is more conservative using more data that it would have in comparison to distribution-depended ones.

Consider an observed mean $\bar{x}$ of a random variable x whose range R is calculated from a small observed sample of n independent observations $x_1, x_2 \dots x_n$. The true mean of the variable x is at least $\bar{x} - \epsilon$ where:

$$\epsilon = \sqrt{\frac{R^2 ln(\frac{1}{\delta})}{2n}} \tag{8}$$

ϵ represents the bound on how close the estimated mean is to the true mean after n observations, with confidence of at least $1 - \delta$; δ is a value between 0 and 1, this is known as the Hoeffding bound.

Babcock et al. (2002) presented a sampling procedure on a sliding window model, while Domingos and Hulten (2000) used Hoeffding bound for sampling data in decision tree-based classification and k-mean clustering. Hoeffding bound is a probability model that is independent of the probability distribution generating the observation.

3.2 Volatility

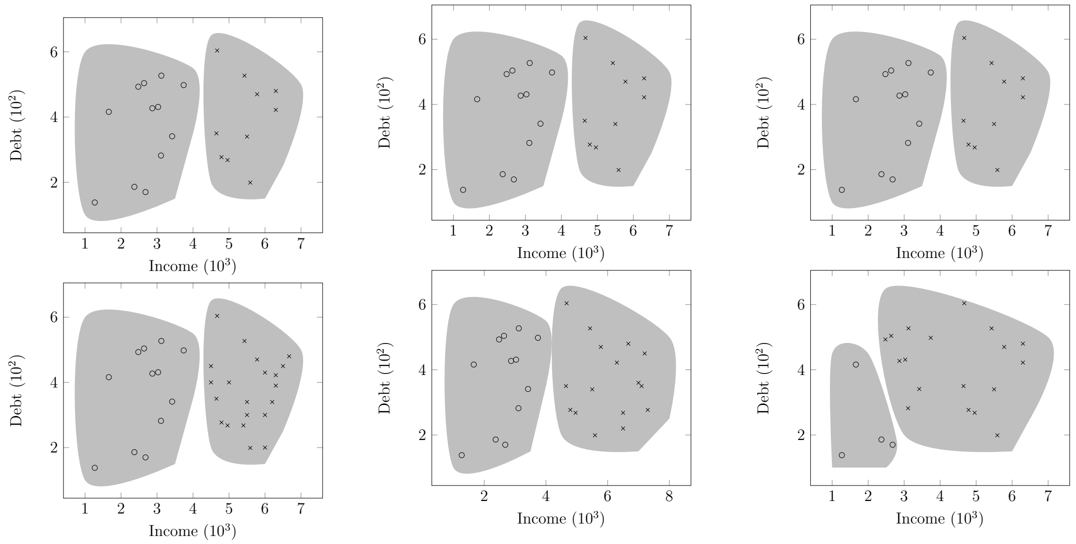
Volatility in data stream leads to concept drift, which is an unforeseen change in patterns or targets, and/or features. Let us consider a classification decision for instance X to class c_i, determined by the prior probability $P(c_i)$ of the class and the class-conditional probability density functions $P(X|c_i)$, $i = 1, \ldots, k$. The posterior probability of an instance belonging to a particular class is $P(c_i|X), i = 1...k$. Concept drift occurs in three ways according to Kelly, Hand and Adams (1999), as shown in Figure 6 :

1. Drift type 1: the probabilities of classes $P(c_1), ...P(c_k)$ change over time. Figure 6 (a) shows that the frequency of class x (c_x) is higher after a concept drift as $P_t(c_x) \neq P_{t+1}(c_x)$.

2. Drift type 2: the class-conditional probabilities distributions $P(X|c_i), i = 1...k$ change. Figure 6 (b) shows that the boundary of class x changed after concept drift.

3. Drift type 3: the posterior probabilities of classes $P(c_i|X), i = 1...k$ change over time. Figure 6 (c) shows that some objects that were classified as circle will change classification to x after the concept drift.

Some authors consider the first two types 1 and 2 as virtual drifts (Gama et al. 2014), temporary drifts (Lazarescu, Venkatesh and Bui 2004), sampling shifts (Salganicoff 1997), or feature change (Gao et al. 2007). While Type 3 is considered as real concept drift, where the class change, this drift is easier to handle so it is handled by most algorithms.

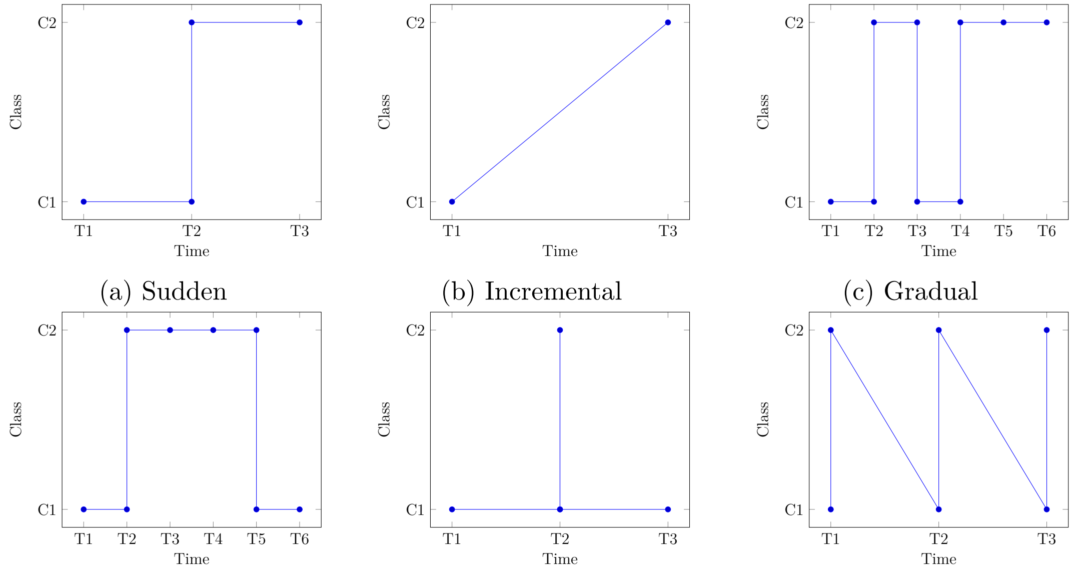


(a) Drift type 1. Note that the class x occurs more frequently after drift: $P_t(c_x) \neq P_{t+1}(c_x)$.

(b) Drift type 2. Note that the boundaries of the class x changed after drift: $P_t(X|c_x) \neq P_{t+1}P(X|c_x)$.

(c) Drift type 3. Note that the boundaries between the classes changed after drift: $P_t(c_i|X) \neq P_{t+1}P((c_i|X)$.

Figure 6: Different Classification of Concept Drift (adapted from (Hoens, Polikar and Chawla 2012)).



(a) Sudden

(b) Incremental

(c) Gradual

(d) Recurrent

(e) Blip

(f) Noise

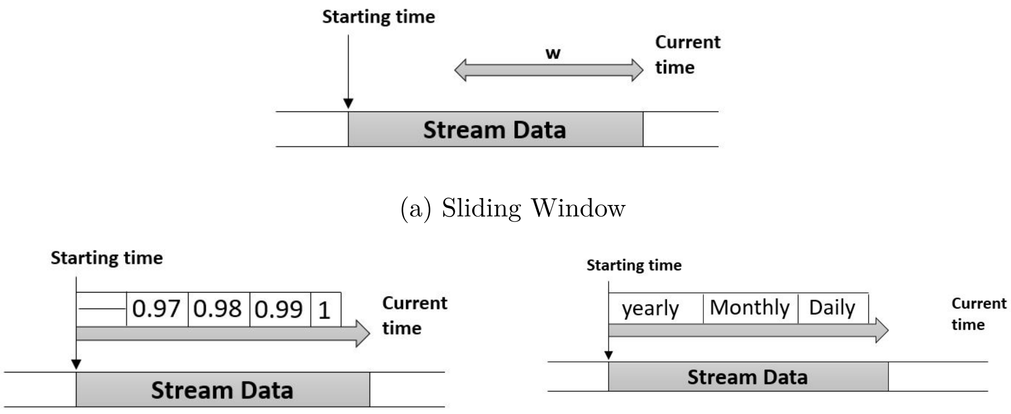
Figure 7: Different Types of Concept Drift (adapted from (Brzeziński 2010)).

Concept drift may manifest in different forms over time, as illustrated in Figure 7, showing the change in a single variable over time for a two class problem. A drift might occur suddenly, where the change in the class assignment is instant and irreversible; incremental drift slowly occurs over time, where the attributes slowly change their values; gradual drift slowly occurs over time; recurrent drift represents changes that are temporary and reverted after some time, it can be periodic and reappear again; blip is a more rare change, which could be considered as an outlier in the data and should be ignored in data stream; finally, noise should be ignored and not considered in data stream as this fluctuation is not connected to the source distribution (Brzeziński 2010).

Data stream mining is constrained by memory and time: it is thus not possible to store all data, but only to remember a small subset thereof, like a time window. This constraint could be relaxed to allow an algorithm to remember the data for a short term, like a time window. However, it will have to discard data to use newly arriving data (Aggarwal 2009; Gama 2013; Bifet et al. 2010). Moreover a near-real time response is required, so mining algorithms should be fast.

Data stream should be able to react to concept drift by forgetting outdated data, while learning new patterns. A popular alternative to data summarisation, is to use a time window. Time windows are used to process portions of the entire data streams. There are different types of time windows: landmark, sliding, fading and tilted time windows, as illustrated in Figure 8. Time windows allow an algorithm to adapt to changing concepts by forgetting outdated data.

Sliding window uses the most recent instances, eliminating older instances. With the arrival of a new data instance, the oldest instance that does not fit in the window is thrown away (Wang et al. 2003; Guha, Kim and Shim 2004; Oza 2005; Hashemi et al. 2009; Zhang et al. 2009, 2011a,b,c; Nguyen et al. 2012). Fading window assigns weight to instances according to arrival time and a decreasing exponential function is used as a fading model. Older examples receive smaller



(a) Sliding Window

(b) Fading window ($\lambda = 0.99$)

(c) Tilted-time window

Figure 8: Examples of time windows: a) sliding window b) fading window c) tilted-time window (adapted from (Nguyen, Woon and Ng 2015)).

weights and are treated as less important by the classifier (Park and Lee 2004; Cao et al. 2006; Tasoulis, Ross and Adams 2007; Chen and Tu 2007; Dang et al. 2009; Lühr and Lazarescu 2009; Rai, Daumé and Venkatasubramanian 2009; Smith and Alahakoon 2009; Leite, Costa and Gomide 2010). Tilted time window uses different summarisation with regards to the age of data. The most recent data are stored in fine scale and long-term data are stored a summarised scale e.g., a histogram structure (Aggarwal et al. 2003, 2006; Guo et al. 2011; Zhou et al. 2007).

3.3 Velocity

Velocity impacts the data stream mining processing time. Due to the high rate that data arrives, data stream algorithms are not able to do any offline or time-consuming tasks. Preprocessing tasks in traditional data mining are used to improve the quality of the data, and usually take place off-line. Preprocessing methods include feature selection, outlier definition and removal. In data streams, it is challenging to implement preprocessing frameworks (Krempl et al. 2014) that can be fully automated and easily integrated with the stream mining task being used due to the velocity. Addressing the frequency of missing feature values, how to select the best imputation method when missing values are present, and the trade-off between speed and accuracy are all challenges due to the velocity.

Computational approach in data stream is usually based on incremental learning. In incremental learning, the model evolves incrementally to adapt to changes in the incoming data. The frequency of the update can vary from being by data instance (Guha et al. 2000; Guha, Kim and Shim 2004; Leite, Jr and Gomide 2009) or by window (Hulten, Spencer and Domingos 2001; Dang et al. 2009). Its main advantage is that it provides results instantly, but it requires more computational resources.

3.4 Evaluation

In traditional data mining algorithms, where the algorithm have unrestricted access to the data, the evaluation process focuses on maximizing the use of data. *Hold-out*, *K-fold cross validation* and *leave-one-out* are standard validation methods (Kantardzic 2011). *Hold-out* methods randomly partition the data set into two subsets, one for training and one for testing — the ratio of training and testing partitions are usually $(\frac{1}{2}, \frac{1}{2})$ or $(\frac{2}{3}, \frac{1}{3})$, respectively. The k-fold *cross-validation* partitions the data into k independent and equal size[1] subsets, each subset used once for testing while the remaining $k - 1$ are used for training. This process is repeated k times and the results are averaged to provide one single output. In most cases, the k partitions maintain the same class distribution as the original data set (stratified cross validation). The *leave-one-out* is a variant form of cross validation, where k is equal to the data set size and the testing is performed on one data instance.

[1] If the data set cannot divide equally to K subsets, it can have one or more subsets not the same size

Since in data streams the data is unbounded, the validation process focuses on evaluating the model at various stages. A well-know approach is to create a learning curve by measuring the model performance over time to show how much the model improves with training data and how well it adapts to concept drift. *Hold-out* and *prequential* are two popular approaches for stream validation (Bifet et al. 2015). In the *hold-out* method, data instances are collected into chunks — each chunk is used as a testing and then used to update the model. Models that adapt to concept drift tend to use this method as it allows the model to adapt to latest changes in data. In the *Prequential* method, the data instances are used to test the model before they are used to update the model. The *prequential* method can be considered a special case of the *hold-out* method with chunk size equal to one.

Bifet et al. (2015) discussed major issues on the evaluation of data stream classifiers and proposed solutions for those problems. They report the current evaluation are not standardised, and current practice is that the comparisons of algorithms performance, although appearing acceptable, are frequently invalid. In their paper, they highlight the importance of a proper evaluation methodology for streaming classifiers, and produced some recommendations. The first recommendation is the use of prequential k-fold distributed bootstrap validation to compare different classifiers and also to use the Wilcoxon's signed-rank test (Wilcoxon 1992) for testing if any statistically significant improvement exists between different classifiers. k-fold distributed bootstrap validation runs k instances of the same classifier, where each instance is used for training in approximately two thirds of the classifiers, with a separate weight[2] in each classifier, and for testing in the rest. The use of only two third simulates drawing random samples with replacement from the original stream.

Another issue identified by Bifet et al. (2015) is the class imbalance problem,

[2] The weight is used to calculate the prequential accuracy of the classifier.

where the prior probability of one class is small compared to that of the rest of classes. This is a frequent problem in real-world applications, like fraud detection or credit scoring (Krempl et al. 2014). They propose several statistics to handle any skew in evolving data stream. The recommendations in (Bifet et al. 2015), uses 3 different measures kappa measures:

1. **Kappa** compares an algorithm's prequential accuracy to a chance classifier (one that assigns the same number of instances to each class as the algorithm being evaluated);

2. **Kappa M** compares an algorithm's prequential accuracy to a simple majority class classifier;

3. **Kappa Temporal** compares an algorithm's prequential accuracy to a persistent classifier (one that predicts the class label of the previous instance for the current instance).

Moreover, they propose an ADWIN prequential evaluation, rather than using a sliding window. ADWIN is an adaptive sliding window algorithm for detecting change and keeping updated statistics from a data stream (Gama, Sebastião and Rodrigues 2012). ADWIN prequential evaluation is used to calculate the accuracy of the model, rather than using the accuracy of all instances seen, the ADWIN averages the accuracy on different sized windows, this shows the different changes in the prefromance of the algeothrim. ADWIN keeps a variable-length window of recently seen items, an older item in the window is dropped if and only if there is enough evidence that its average value differs from that of the rest of the window.

3.5 Related Work in Data Stream Classification

Data stream classification algorithms are called adaptive classifiers. They are implemented to learn, forget and to limit the storage of data instances. All data stream algorithms implement update mechanisms to cope with new data instances arriving from the stream. They update the model with different procedures. Moreover, some algorithms implement specific procedures to be more reactive to concept drift. In Table 3, we present the approaches that we will discuss in this section, presenting the update mechanism and type, and indicating if they have a special procedure for handling concept drift.

Considering the update mechanism, algorithms are divided into evolving and trigger-based learners. Trigger-based algorithms uses a trigger to start a learning process to update the model. In this case, the algorithm could learn a new model or increment the model used. Evolving algorithms update the model gradually and always learn using the data stream. Considering the update type, algorithms are divided into incremental and replacement learners. In replacement algorithms, the current model is discarded and a new model is build from scratch based on the new data. In incremental algorithms, the model is incrementally updated to handle the data stream.

A Hoeffding decision tree approach called Very Fast Decision Tree (VFDT) was proposed by Domingos and Hulten (2000). The algorithm does not store any data instances in main memory, requiring only space proportional to the size of the tree and associated statistics to calculate the information gain for attributes. For each attribute, the statistic consists of the frequency of all values seen. VFDT builds a decision tree in a similar fashion to the classic C4.5 tree induction algorithm (Quinlan 1993), growing the decision tree as more data arrives from the stream. The Hoeffding bound is used to decide on the number of instances needed to be seen by a leaf node before making a decision to create a test and divide the node into two leaves. The attribute selected is the one that provides the best information gain. The experiments showed that VFDT using the Hoeffding bound on a subset of the data selected the same nodes to build the tree as when the entire data set used. Thus, given a stream of instances, the first ones will be used to select the root test; once the root attribute is chosen, the following instances will be passed down to the corresponding leaves and used to choose the appropriate attributes there, and so on recursively.

Table 3: Review of data stream classification algorithms employing different update mechanisms and update types.

Algorithm Name	Acronym	Citation	Model Type	Update mechanism	Update Type	Concept drift
Very Fast Decision Trees	VFDT	(Domingos and Hulten 2000)	Decision Tree	Evolving	Incremental	No
Concept-adapting Very Fast Decision Tree	CVFDT	(Hulten, Spencer and Domingos 2001)	Decision Tree	Evolving	Incremental	Yes
Hoeffding Adaptive Trees	Adaptive VFDT	(Bifet and Gavaldà 2009)	Decision Tree	Evolving	Incremental	Yes
Evolutionary Learning Classifier System	XCS	(Dam and Lokan 2007)	Rule Model	Evolving/ Trigger	Replacement	Yes
Very Fast Decision Rules	VFDR	(Gama and Kosina 2011)	Rule Model	Evolving	Incremental	No
Online Genetic Algorithm	OGA	(Vivekanandan and Nedunchezhian 2011)	Rule Model	Evolving	Incremental	Yes
Evolving set of Rules	eRules	(Stahl, Gaber and Salvador 2012)	Rule Model	Trigger	Incremental	Yes
Gaussian Evolving set of Rules	G-eRules	(Le et al. 2014)	Rule Model	Trigger	Incremental	Yes
Hoeffding Rules	HRules	(Le et al. 2017)	Rule Model	Trigger	Incremental	Yes

Table 3: Review of data stream classification algorithms employing different update mechanisms and update types.

Algorithm Name	Acronym	Citation	Model Type	Update mechanism	Update Type	Concept drift
Symbiotic Bid-Based Genetic Programming	StreamSBB	(Vahdat et al. 2014b,a)	Rule Model	Evolving	Replacement	No
Symbiotic Bid-Based Genetic Programming	StreamSBB	(Khanchi, Heywood and Zincir-Heywood 2017)	Rule Model	Evolving	Replacement	No
Growing Type 2 Fuzzy Classifier	GT2FC	(Bouchachia and Vanaret 2014)	Fuzzy Rule Model	Evolving	Incremental	Yes
Streaming Ensemble Algorithm	SEA	(Street and Kim 2001)	Ensemble	Evolving	Replacement	Yes
Dynamic Weighted Majority	DWM	(Kolter and Maloof 2007)	Ensemble	Evolving	Replacement	Yes
Online Bagging with ADWIN	OBAG$_A$	(Bifet et al. 2009)	Ensemble	Evolving	Replacement	Yes
Online Boosting with ADWIN	OBOOST$_A$	(Bifet et al. 2009)	Ensemble	Evolving	Replacement	Yes
Diversity for Dealing with Drifts	DDD	(Minku and Yao 2012)	Ensemble	Trigger	Replacement	Yes
Adaptive Random Forest	ARF	(Gomes et al. 2017)	Ensumble	Evolving	Replacement	Yes
Early Drift Detection Method	EDDM	(Baena-Garcıa et al. 2006)	Detection Method	Trigger-based	Replacement	Yes
Growing Prototype Network Classifier	GPNC	(Cervantes et al. 2013)	Clustering	Evolving	Incremental	Yes

Table 3: Review of data stream classification algorithms employing different update mechanisms and update types.

Algorithm Name	Acronym	Citation	Model Type	Update mechanism	Update Type	Concept drift
Supervised Neural Constructivist System	SNCS	(Sancho-Asensio, Orriols-Puig and Golobardes 2014)	MLPs	Evolving	Replacement	Yes
Dynamic Extreme Learning Machine	DELM	(Xu and Wang 2017)	Neural Networks	Trigger	Incremental	Yes
Ensemble OS-ELM based on combination weight	CWEOS-ELM	(Yu, Sun and Wang 2019)	Ensemble/ Neural networks	Evolving	Replacement	No
Adaptive Random Forest	ARF	(Gomes et al. 2017)	Ensumble	Evolving	Replacement	No

Hulten, Spencer and Domingos (2001) proposed the CVFDT algorithm as an extension of VFDT to cope with concept drift, using a fixed window size to determine nodes in the tree that are ageing. Fragments of the decision tree that become old and inaccurate after seeing a user defined number of instances are replaced with alternative subtrees. The process incrementally improves and updates the decision tree, while building subtrees to update the model using the data instances in the current time window. The resulting accuracy of the CVFDT is similar to what would be obtained by reapplying a VFDT to the entire window every time a new data instance arrives. The experiments showed that CVFDT is better at controlling the size of the tree throughout concept drifts, while VFDT considers many more examples and it is forced to grow larger trees to make up for the early decisions becoming incorrect.

Hoeffding Adaptive Tree or Adaptive VFDT is presented in (Bifet and Gavaldà 2009). It uses a change detection mechanism in order to define the length of a window of relevant patterns. Adaptive VFDT used ADWIN window to detect changes in a series of pattern. If changes are detected in the data, the length of the window decreases; when no change is present the window length increases.

The evolutionary learning classifier XCS was orginal introduced in (Wilson 1995). XCS was adapted for data stream by Abbass et al. (2004). Another XCS algorithm (Dam and Lokan 2007) is a rule-based system, where each rule (individual) represents a partial solution to the target problem. The typical goal of XCS is to evolve a population of rules to represent a complete solution to the target problem. XCS relies on reinforcement learning for evaluating rules in the population, and on a GA for exploring the search space and introducing new rules into the population. XCS receives a data stream instance and returns a prediction, if the prediction is good the rule is rewarded. The GA runs to produce two offspring once with every new data instance, the two offspring are added to population and compete with parents. The population size is a user-defined parameter, and when the population size reaches the maximum, the worse performing individuals are removed. Three different strategies were proposed: (1) an adaptive learning strategy to adjust the learning rate according to the prediction error; (2) re-initialize the population when a drift is recognized; and (3) re-initialize the learning rate parameter values when a drift is recognized. XCS detects concept drift by monitoring the prediction error. The results show that in the case of small drifts, the original XCS, adaptive (1), and learning rate re-initialization (3) perform better than re-initializing the population (2). While in the case of larger drifts, the number of affected rules is large, therefore the re-evaluation time is large enough so that it is more effective to re-learn rules from scratch; re-initializing population (2) performed best as a consequence.

Both original XCS and the adaptive strategies are considered evolving approaches that do not implement a drift detection procedure, while re-initialising learning rate and the re-population strategies are considered trigger-based. In all cases the model is updated by the current population following a replacement mechanism. This is due to the fact that the model is based on population, that is updated in the learning procedure.

Very Fast Decision Rules (VFDR) was proposed by Gama and Kosina Gama and Kosina (2011). It is a single pass[3] algorithm that learns ordered or unordered rules. Similar to the VFDT's approach, the statistics are saved for all values seen for each attribute. The Hoeffding bound is used to determine the number of instances seen before a rule can be expanded or a new rule can be induced from the default rule. A rule can be induced or expanded, by creating an attribute-value condition based on the current seen instances to create a homogeneous subset. A rule is expanded with the attribute-value condition that minimizes the entropy of the class labels of the instances covered by the rule. Experiments have shown

[3] A single pass uses data instance only once.

that the number of rules produced by VFDR is much smaller than the number of leaves in a VFDT tree.

Vivekanandan and Nedunchezhian (2011) proposed an Online Genetic Algorithm (OGA), an incremental rule learning algorithm that creates a rule set for data stream classification with concept drift. Each individual represents a classification rule and the algorithm builds the rule set gradually by evaluating each individual on a window of instances, adding the best individual of the population to the rule set. Rules that fall under user-defined threshold accuracy are removed from the rule set. OGA has the limitation of not handling continuous attributes in the data sets, only nominal attributes are supported.

Stahl, Gaber and Salvador (2012) proposed the eRules algorithm for rule induction in data stream. eRules uses a fixed sliding window and learns rules using the Prism algorithm (Cendrowska 1987). Prism is a greedy rule induction algorithm that creates a set of rules, where each attribute-value pair in the antecedent of the rule is chosen to maximise the probability of the target class. New instances from the stream are added to a buffer if they are not covered by the current rule set. eRules uses a user-defined limit of instances on the buffer to trigger the incremental creation of new rules. To adapt to concept drift, the rule set is validated using the current buffer. If a rule's accuracy is deteriorated as a results of misclassifying instances over time, it is removed from the rule set. eRules use a time-consuming discretisation procedure, testing multiple cut points to create a continuous attribute-value-condition.

Le et al. (2014) proposed an extension to eRules to handle continuous attributes in a more computational efficient way. The proposed G-eRule extension uses a Gaussian distribution on continuous attribute values to efficiently sample values to create continuous attribute-value conditions. Le et al. (2017) added a Hoeffding bound procedure to determine the credibility of a rule condition. A rule condition is added to the current rule if the difference of the conditional probabilities between the new rule condition and the second best possible rule condition is greater than the Hoeffding bound.

Vahdat et al. (2014a) proposed a GP for streaming data classification tasks with label budgets, where the GP learns a model using a limited number of labelled instances. Operating under a label budget assumption means requesting a label for the data instance is computationally expensive, so the algorithm tries to reduce the number of requests for labels. The GP uses sliding window with a uniform sampling procedure to request labels. The apporach have a data archive to save a sampled set of seen instances, to be used by the algorithm in updating the model. The approach uses a sampling procedure to select instances from the data stream to receive a label and add them to an archive. Then, a GP procedure is triggered to create a model on the archive instances. The best individual of the GP is selected as the anytime classifier. The GP runs a limited number of iterations on the archive. This approach handles concept drift by updating the model using the GP procedure.

Khanchi, Heywood and Zincir-Heywood (2017) proposed improvement for problems with class imbalance under the label budget. The archive and sampling polices are optimised to operate under class imbalanced context, where they incrementally introduce a bias on both the sampling of the stream and the replacement of instances in the archive to balance the class distribution and improve the model creation.

Bouchachia and Vanaret (2014) propose an on-line rule learning algorithm based on the Growing Type 2 Fuzzy Classifier (GT2FC). The algorithm is designed to operate on-line and to learn from both labelled and unlabelled data. The idea is to produce clusters that evolve over time to generate the rules antecedents. Growing Gaussian Mixture Model (2G2M) (Lee 2005) is used to generate the type-2 fuzzy membership function to predict the class. Fuzzy membership allow us to define fuzzy sets, where each element is mapped to a value between 0 and 1. This value, called membership value or degree of membership, quantifies the grade of membership of the element to the fuzzy set. The algorithm adapts to concept drift by continuously and regularly evolving its 2G2M parameters using batches of labelled data. To maintain compactness of the rules, the GT2FC classifier uses an online feature selection algorithm.

An ensemble approach is proposed by Street and Kim (2001), called Streaming Ensemble Algorithm (SEA). SEA combines a maximum of 25 unpruned decision tree classifiers. The prediction is a combined majority vote and ties are broken randomly. SEA uses a heuristic strategy to replace the "weakest" classifier based on two factors: accuracy and diversity. Accuracy is important because, as the authors state, an ensemble should correctly classify the most recent examples to adapt to a concept drift. On the other hand, diversity is the source of success of ensemble methods in static environments such as Bagging or boosting. SEA trains a new classifier on each sequential batch of data and the trained classifier is added to the fixed-sized ensemble while the worst performing classifier is discarded. Experiments showed that SEA performs better than a single tree classifier in data without concept drift, but when the target concept changes suddenly the performance of both algorithms decreases. The dynamic nature of SEA allows the accuracy to recover very quickly, while the single decision tree, which still uses data points from the old concept, recovers much more slowly, if at all.

Dynamic Weighted Majority (DWM) (Kolter and Maloof 2007) is a well cited approach for concept drift. DWM does not use a drift detection method, it maintains an ensemble of classifiers whose weights are reduced by a multiplier constant ρ, $\rho < 1$, when the classifier gives a wrong prediction. DWM allows the addition and removal of a classifier with every instance arriving. If a classifier weights falls below a specific threshold, then DWM removes it from the ensemble. The ensemble is general and can be used with any algorithm — in the paper they used a incremental naive Bayes (Witten et al. 2016) and incremental tree inducer (Utgoff, Berkman and Clouse 1997) as the base classifier.

Online Bagging with ADWIN (OBAG$_A$) and Online Boosting with ADWIN (OBOOST$_A$) where proposed by Bifet et al. (2009). OBAG$_A$ bagging trains each model in the ensemble using a randomly drawn subset of the training set. Predictions are based on the unweighted voting of each base classifier. The algorithm uses ADWIN window to adjust to concept drift. OBOOST$_A$ is a boosting algorithm using ADWIN window to adjust to concept drift, while ONSBOOST uses a fixed size window, and removes and updates a classifier, if the ensemble is performing better without that classifier. Both approaches used Adaptive Hoeffding tree as base classifiers.

Minku and Yao (2012) proposed an online leaning algorithm called Diversity for Dealing with Drifts (DDD). The algorithm starts with two modified "Online Bagging" ensembles: an ensemble with lower diversity and an ensemble with higher diversity using as a base classifier lossless Incremental Tree Inducer (ITI) online decision trees classifier (Utgoff, Berkman and Clouse 1997). The original online bagging ensemble uses each data instance that arrives and train K times (where K is random value from distribution of $Poisson(1)$) each base classifier. To create different diversity among the ensembles DDD uses the parameter λ for the $Poisson(\lambda)$ distribution, where higher/lower λ values are associated with higher/lower diversity (Minku, White and Yao 2010). λ takes a user-defined value between 0 and 1 to calculate the diversity level of each ensemble.

The DDD algorithm has two modes of running, stable stream and concept drift detection. In the stable stream mode, the lower diversity ensemble is used for model predictions since the lower diversity ensemble is more accurate on the the data stream, although both high/low diversity ensembles are trained on the incoming instances. The algorithm uses a concept drift detection method, and triggers a new learning mode when it detected a concept drift. After a drift is detected, DDD creates two new ensembles, one with low and the other with high diversity on the new coming data instances. The old high diversity ensemble starts learning with lower diversity; (lower value of λ); in order to improve its convergence to new concept. The four ensembles start to learn and their predictions are determined by weighted majority voting of three ensembles: the old high, the new low, and the old low diversity ensembles. The new high diversity ensemble is ignored, since it will have low accuracy on the current data instances. The algorithm returns to the stable stream mode when the new low ensemble has higher accuracy than the old low and old high ensemble. When this happen DDD replaces both high/low old diversity ensembles with both high/low new diversity ensembles. As a special case, if the old high diversity ensemble is perfomering better than the new low diversity ensemble, the algorithm will choose the old high as the low diversity ensemble and the new high diversity ensemble as the high diversity ensemble, discarding the new low and old low ensembles.

Random forests is a well-known algorithm in traditional data mining. Gomes et al. (2017) presents an Adaptive random forest ARF, where an online booststrap process is used in sampling the incoming data instances for each base tree. Moreover, each tree in the ensemble is limited to a random subset of features when considering node split. The algorithm uses a drift detection procedure to relearn the ensemble when a warning of concept drift is detected, and replace the ensemble when the concept drift is detected.

The Early Drift Detection Method (EDDM) (Baena-Garcıa et al. 2006) is based on the idea that the distance between two consecutive errors increases when a the data stream is stable. In EDDM, the distance is monitored and if it reduces considerably according to predefined constant value, a concept drift is detected. EDDM could be considered an online learning system, if we consider that a new online classifier system is created when the warning level is triggered, instead of storing the training instances for posterior use. The paper used three distinct learning algorithms namely J48 (Quinlan 1993) (C4.5, decision tree), IB1 (Aha, Kibler and Albert 1991) (nearest-neighbourhood, it is not able to deal with noise) and NNge (Martin 1995) (nearest-neighbourhood with generalisation).

Particle Swarm Optimization (PSO) concepts were adapted in the Growing Prototype Network Classifier (GPNC) algorithm (Cervantes et al. 2013). GPNC is an incremental learning algorithm that generates a network of linked prototypes, each labelled with one of the class labels in the training data set. Each prototype has a class label, a fitness value, a set of neighbours prototypes, as well as position and velocity vectors as in PSO. The prototypes move in the search space using a PSO simplified velocity equation, and classify nearest arriving data instances. It adapts to concept drift using a decay mechanism for the noisy prototypes. When detecting contradiction between new data and previous data, the model deletes prototypes with fitness lower than a user-defined threshold.

Sancho-Asensio, Orriols-Puig and Golobardes (2014) proposed a Supervised Neural Constructivist System (SNCS) for mining data streams with concept drift. The SNCS classifier uses a population of multilayer perceptrons (MLP) with feed forward topology (i.e., the signal propagates from inputs toward the output layer). SNCS operates in two modes, the learning mode and the prediction mode. In the learning mode, SNCS discovers and evolves new MLPs that accurately predict a desired label. In the prediction mode, SNCS uses its current knowledge to determine the best label for new input instances.

Xu and Wang proposed the Dynamic Extreme Learning Machine (DELM) for data stream classification (Xu and Wang 2017). Extreme learning machine (ELM) is a single hidden layer feedback neural network. Due to its fast training and good generalization, ELM has been applied to many fields and recently to data streams. DELM uses two hidden layers so that it can dynamically adjust the ELM layer when concept drift is detected. From the results comparing DELM with different ELM implementations for data streams, DELM can get a better balance between accuracy and time overhead than online sequence extreme learning machine (OS-ELM) (Liang et al. 2006).

Yu, Sun and Wang (2019) proposed an ensemble of OS-ELM based on combination weight. The algorithm extends the OS-ELM (Liang et al. 2006) algorithm, by introducing a two phase process: a learning step and an updating step. In the learning step, the weights of the ensemble is determined by Adaboost. In the update step, the weights are calculated using game theory analysis of the prediction of the base learners.

3.6 Summary

In this chapter we focused on the challenges of dealing with data streams in data mining. We described main concepts and methodologies found in the literature to deal with data streams. We also discussed the differences between evaluation measures for traditional (offline) data mining and (online) data stream mining, including specific evaluation measures from the literature.

We also described current approaches for data stream classification, focusing on three different aspects: (1) How they update the model to cope with new instances; (2) The update type, incremental or replacement; (3) if a special procedure is used to detect a concept drift.

We can identify recurrent themes in the discussed algorithms, starting with the use of Hoeffding bound as a heuristic to update the model under construction. Since it is infeasible to store all data, algorithms employ sampling and archive procedures to store relevant data. This is usually the case with evolutionary algorithms in order to allow the model to evolve for a limited number of iterations. ADWIN windows have proved useful for deciding the window size when evaluating the algorithms, and then deciding the size of the instance buffer. Using archive and sampling procedures to store some instances for algorithms update.

Chapter 4

Ant Colony Optimization

Combinatorial Optimization problems (Papadimitriou and Steiglitz 1998) are problems that consist of finding a combination of components from a finite set of components where the combination is optimal with respect to a given objective function. Classical combinatorial problems include shortest-path problems, where the goal is to find a minimum cost plan to deliver goods to customers, e.g., Travel Salesman Problem (Lawler 1985). In many problems, a straightforward exhaustive search to enumerate all possible combinations and select the best one is infeasible, since the number of solutions tend to grow exponentially with the size of the problem.

Ant Colony Optimization (ACO) is a metaheuristic inspired by the foraging behaviour of real ants (Dorigo and Stützle 2004). Ant colonies, and more generally social insect societies, are distributed systems that, in spite of the simplicity of their individuals' behaviour, present a highly structured social organization. As a result of this organization, ant colonies can accomplish complex tasks that in some cases far exceed the individual capabilities of a single ant. Many ant species can find shortest paths between food sources and their nest with limited or no visual aid (Blum 2005). When searching for food, ants start by exploring the area surrounding their nest at random. While moving, ants deposit a chemical pheromone trail on the ground. As soon as an ant finds a food source, it starts the journey back to the nest. During the trip back, the ant deposits more pheromone on the ground. Since an ant can move back and forth quicker when using a shorter path, over time the pheromone concentration on the shortest path will be greater. Ants can detect pheromone and they will prefer trails that are associated with strong pheromone concentration to follow, i.e., the stronger the pheromone concentration, the higher the chance that the path will be selected. This indirect communication is called *stigmergy* (Grassé 1959).

This chapter will discuses the ACO metaheuristic (Dorigo and Stützle 2004) and its extension $\text{ACO}_{\mathbf{MV}}$ (Liao et al. 2014). We will then discuses how ACO is used in data mining, covering well known approaches in the literature.

4.1 Metaheuristic

As aforementioned, ant colony optimization (ACO) is a metaheuristic where a colony of artificial ants cooperate to find good solutions to discrete optimization problems. Cooperation is a key design component of ACO, where simple agents communicate by depositing pheromones. The amount of the deposited pheromone is proportional to the quality of the solution the ant has built. Good solutions are an emergent property of the agents' cooperative interaction. ACO algorithms are based on the following steps:

- Create an appropriate representation of the problem, where the problem is represented as a construction graph. Nodes of the graph represent components of the solution and edges represent the connections between them. Ants incrementally create solutions through the use of a probabilistic transition rule, based on the amount of pheromone and a local problem-dependent heuristic associated with nodes or edges of the graph. At the start of the search, all nodes/edges are given the same probability of being selected;

- Each path followed by an ant represents a candidate solution for a given target problem;
- When an ant follows a path, the amount of pheromone deposited on that path is proportional to the quality of the corresponding candidate solution, given a problem-dependent evaluation function.
- When an ant has to choose between two or more paths, the path with a larger amount of pheromone has a greater probability of being chosen by the ant;
- Pheromone concentration decreases over time, so sub-optimal trails would have a smaller chance of being chosen by ants again.

As a result, the colony eventually converges to a good solution, generally the optimum or a near-optimum solution for the target problem. The high-level pseudocode of a ACO algorithm is shown in Algorithm 3, where the algorithm is informally divided into three procedures: Construct Ant Solutions, Update Pheromones and Daemon Actions.

Construct Ant Solutions manages a colony of ants that concurrently visit adjacent nodes by moving through the problem construction graph. They move by applying a stochastic local decision based on the values of pheromone and heuristic information. The probability of ant k to move from node i to node j is given by :

$$p_{ij}^{k} = \begin{cases} \frac{\tau_{ij}\eta_{ij}}{\sum_{l \in N_i^k} \tau_{il}}, \text{if } j \in N_i^k \\ 0, j \notin N_i^k \end{cases} \tag{9}$$

where N_i^k is the set of neighbouring nodes of ant k when at node i, τ_{ij} is the amount of the pheromone on the edge connecting nodes i and j, and η_{ij} is the heuristic information on the edge connecting nodes i and j. The neighbouring nodes of an ant k when at node i are all the nodes that are connected to node i

and are valid to visit. Nodes can be invalid based on a heuristic problem-based function or when the node has been visited by the ant before. The ants stop visiting nodes, when there is no more available nodes to visit.

After all ants build their solution, a set of the high quality solutions — usually the best solution — deposit pheromone on the edges used. The pheromone deposit is a function of the generated solution quality, which helps in directing future ants more strongly towards the components that lead to the creation of better solutions. Interestingly, the correlation between the amount of pheromone to deposit and the solution quality is also present in some ant species: (Beckers, Holland and Deneubourg 2000) found that some ant species will deposit more pheromone when returning from rich food sources than from poorer food sources. The update of the pheromone value τ_{ij} is given by:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\theta^k, \forall \tau_{ij} \in S^k \tag{10}$$

the value of $\Delta\theta^k$ is a value derived from the quality of the solution created by the ant K (S^k). Then, all pheromone values are evaporated to avoid quick convergence towards a suboptimal solution. The pheromone evaporation is important for the search of new solutions by allowing the ants to forget poor choices made at earlier stages of the search. The pheromone values are evaporated based on equation (11):

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij}, \forall (i, j) \in A \tag{11}$$

where $\rho \in (0, 1]$ is a user defined parameter and A is all the set of edges in the construction graph.

Finally, the optional daemon actions procedure is used to implement centralised actions, which can not be performed by individual ants. These include using local search procedure to further improve solutions, adding the recalculation of heuristic information about the problem, and selecting one or few ants to

Algorithm 3: High-level pseudocode of Ant Colony Optimization

```
Set parameters, initialize pheromone trails
while Termination condition not met do
    Construct Ant Solutions
    Update Pheromones
    Daemon Actions
end
```

deposit additional pheromone.

The above steps are repeated until one or more termination criteria are satisfied, which usually include a maximum number of iterations and/or a stagnation test, where it detects where the ants failed to produce better solutions after a fixed number of iterations. The best solution found is then returned as the result of the search.

Different variations of ACO algorithms were proposed in literature: the Ant System, which was the first ACO algorithm proposed in the literature (Dorigo, Maniezzo and Colorni 1996), the Ant Colony System (ACS) (Dorigo and Gambardella 1997) and the $\mathcal{MAX} - \mathcal{MIN}$ ant system (Stützle and Hoos 2000) are among the most influential ones. ACO algorithms have been applied to many different combinatorial optimization problems, such as routing problems (Dorigo, Maniezzo and Colorni 1996), assignment problems (Stützle and Hoos 2000) and scheduling problems (Merkle, Middendorf and Schmeck 2002); data mining (Parpinelli, Lopes and Freitas 2002; Martens et al. 2007; Otero, Freitas and Johnson 2008); dynamic optimization (Guntsch and Middendorf 2001; Mavrovouniotis 2013); stochastic optimization problems (Bianchi, Gambardella and Dorigo 2002); multi-objective optimization problems (Gambardella, Taillard and Agazzi 1999; López-Ibáñez and Stützle 2012), and continuous optimization problems (Socha and Dorigo 2008).

4.2 Mixed-Variable Optimization

ACO$_{\mathbf{MV}}$ (Liao et al. 2014) is an ant colony optimization for mixed-variable optimization problems, which is an extensions for continuous optimisation (Socha and Dorigo 2008). Mixed-variable optimization solutions are composed by r real-valued variables, c (unordered) categorical-valued variables and o ordinal-valued variables. As discussed in the previous section, ACO can naturally cope with combinatorial (discrete) problems, while ACO$_{\mathbf{MV}}$ is an extension to cope with mixed-variable optimization problems. The fundamental idea underlying ACO$_{\mathbf{MV}}$ is the shift from using a discrete probability distribution, i.e., a construction graph, to using a continuous one, i.e., a solution archive. ACO$_{\mathbf{MV}}$ uses a solution archive and weighted solutions to replace the graph and pheromone model in the classical ant colony optimization.

In ACO$_{\mathbf{MV}}$, the archive structure contains R previously generated solutions. Each solution S_j in the archive, where $j = \{1, 2, ..., R\}$, is a vector containing n-dimensional real-valued components, m-dimensional categorical-valued components and o-dimensional ordinal-valued components. The archive is sorted by the quality Q of solutions, so that $Q(S_1) \geq Q(S_2) \geq \cdots \geq Q(S_R)$.

Each solution S_j is associated with a weight w_j that is related to $Q(S_j)$, where w_j is calculated using the Gaussian function given by

$$w_j = \frac{1}{qR\sqrt{2\pi}} e^{\frac{-(rank(j)-1)^2}{2q^2R^2}} \tag{12}$$

where q (Local Search) is a constant that is used to control the extent of the top-ranked solution influence on the construction of new solutions. The weight of solution S_j is used during the creation of new solutions, as an indicator for the level of attractiveness of this solution. The higher the weight of the solution S_j, the higher the probability of sampling a new solution around the values of S_j.

ACO$_{\mathbf{MV}}$ starts by randomly generating m solutions in the archive R. The solution construction phase starts by each ant i generating a new S_i candidate solution. When constructing solution S_i, a probabilistic solution construction method is used to sample new values from the solution archive according to each attribute type. At the end of an iteration, all solutions created by the ants in the colony are added to the archive $(R + m)$. The archive is sorted by the solution quality and only the best R solutions are kept and the remaining solutions are removed.

4.2.1 Sampling Procedures

ACO$_{\mathbf{MV}}$ sample new values from the solution archive according to each attribute, using the following sampling procedures:

Continuous variables Continuous variables are handled by ACO$_{\mathbf{MV}}$ using ACO$_{\mathbb{R}}$ (Socha and Dorigo 2008), where each ant i probabilistically chooses a solution j given by:

$$P_j = \frac{w_j}{\sum\limits_{l=1}^{R} w_l} \tag{13}$$

where P_j is the probability of selecting the j-th solution from the archive to sample the new continuous variable value around it, R is the size of the archive and w_j is the weight associated with the j-th solution in the archive. Let S_i denote a new solution sampled by ant i around the chosen solution S_j for a continuous attribute a, the Gaussian probability density function (PDF) used to choose a continuous value is given by:

$$S_{i,a} \sim N(S_{j,a}, \sigma_{j,a}) \tag{14}$$

$$\sigma_{j,a} = \xi \sum_{r=1,j\neq r}^{R} \frac{|S_{j,a} - S_{r,a}|}{R-1} \tag{15}$$

where $S_{j,a}$ is the value of the variable a in the solution j of the archive, $\sigma_{a,j}$ is the average distance between the value of the variable a in the solution j and the value of a in all the other solutions in the archive and ξ (convergence) is a user-defined value representing the convergence speed of the algorithm. For the purpose of sampling, the value $S_{a,j}$ is considered the average value of the distribution and $\sigma_{a,j}$ is the variance.

Ordinal variables Ordinal variables are variables that do not necessarily have a numeric value but whose values' order have a meaning, e.g., *small* < *medium* < *large*. ACO$_{\mathbf{MV}}$ handles ordinal variables as continuous variables, where the continuous value is the index of the chosen value in the ordered attribute values. Then, a final step is to round up the value generated from Equation (14) to the nearest index. Using this relaxed continuous sampling allows the algorithm to take into consideration the order of ordinal attribute values.

Categorical variables Categorical variables whose values have no meaningful order, are treated differently by ACO$_{\mathbf{MV}}$. Assume that a categorical variable i has t possible values, so that each ant has to choose a value t_i from $v_l^i \in \{v_1^i, v_1, .., v_t^i\}$. The probability P_l^i to choose the lth value is given by:

$$P_l^i = \frac{\alpha_l}{\sum\limits_{j=1}^{t_i} \alpha_j} \tag{16}$$

where α_l is the weight associated with each l-th value of the categorical variable, calculated as:

$$\alpha_l = \begin{cases} \frac{w_{ji}}{u_{ij}} + \frac{q}{\kappa}, & \text{if } \kappa > 0, u_{ij} > 0, \\ \frac{w_{ji}}{u_{ij}}, & \text{if } \kappa = 0, u_{ij} > 0, \\ \frac{q}{\kappa}, & \text{if } \kappa > 0, u_{ij} = 0, \end{cases} \tag{17}$$

where w_{ji} is the weight of the first solution that uses the value v_{ij} in the archive, u_{ij} is the number of solutions that use the value v_{ij} in the archive, κ is the number of values of this attribute that are not used in the archive and q is a parameter used to control the extent of the top-ranked solution influence on the construction of new solutions (the same parameter found in Equation (12)).

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 solutions but do not occur very frequently in the archive. The second component biases the sampling towards unexplored values of the attribute.

4.3 Ant Colony Optimization for Rule Induction

Ant colony optimization algorithms have been applied to both supervised and unsupervised data mining tasks. Most work on unsupervised learning focused on clustering (Shelokar, Jayaraman and Kulkarni 2004; Abraham, Das and Roy 2008), while more work focused on supervised learning in classification (Martens, Baesens and Fawcett 2011). This section discuses mainly ACO algorithms for classification, more specifically, ACO-based rule induction algorithms.

There are two main approaches to apply ACO to create classification rules in the literature: grammar- and graph-based approaches. In grammar-based approaches, the rule creation is guided by a context-free grammar, which determines the valid structure of rules. The Grammar-Based Ant Programming (GBAP) algorithm (Olmo, Romero and Ventura 2011; Olmo, Romero and Ventura 2012) was the first implementation of a grammar-based approach. Similar to the majority of ACO-based classification algorithms, GBAP does not cope with continuous attributes directly and it uses a discretisation procedure in a preprocessing stage.

Graph-based approaches started with Ant-Miner (Parpinelli, Lopes and Freitas 2002), which was limited to discrete data sets only. Ant-Miner successfully extract IF-THEN classification rules from data. Ant-Miner uses a sequential covering approach to create a rule list model, using an ACO procedure to create rules. Each ant traverses a construction graph, where each node in the graph consists of a condition that the ant might choose to add to its rule.

4.3.1 *c*Ant-Miner

*c*Ant-Miner (Otero, Freitas and Johnson 2008) is an extension of the well known Ant-Miner approach to handle continuous attributes during rule construction. *c*Ant-Miner uses a graph-based approach to extract *IF-THEN* classification rules from data. Each rule is represented as a n-dimensional vector of terms that are joined with *AND*, such as *IF* t_1 *AND* t_2 .... *AND* t_n *THEN (class)*, where each term t_i consist of a tuple (attribute, operator, value).

The construction graph in *c*Ant-Miner consists of a fully connected graph. Let a_i be a nominal attribute and v_{ij} be the j-th value of attribute a_i. For $j = 1, ..., b_i$, where b_i is the number of values of attribute a_i, each v_{ij} is added as a node (a_i, $=$, v_{ij}) to the graph. Let c_i be a continuous attribute, only one node (c_i) is added to the graph—the operator and value are not defined for a continuous attribute node, since those will be dynamically selected during the rule construction procedure. The pheromone model is represented as a matrix, where columns and rows represent the nodes and the value in each cell represents the pheromone value of the edge connecting the node specified in the row to the node specified in the column.

*c*Ant-Miner uses the sequential coverage approach to create a list of rules. *c*Ant-Miner constructs a rule that satisfies part of the training instances using an ACO procedure, then removes those instances, and repeats until no (or very few) training instances remain.

The high-level pseudocode for *c*Ant-Miner is shown in Algorithm 4. In summary, *c*Ant-Miner works as follows. It starts with an empty rule list (line 2) and iteratively (*while* loop) (line 3) adds one rule (the best rule discovered by the ACO procedure) to that list (line 18) while the number of uncovered training instance is greater than *max_uncovered_training_instances*, a user-defined parameter. The antecedents of the rules are chosen probabilistically in line 8. Rules are then pruned in line 9 to remove irrelevant terms from the antecedent — i.e., terms that are added as result of the stochastic behaviour of the ACO procedure, but have very little or no predictive power. The consequent of a rule is computed based on the most frequent class value observed on the training instances covered by the rule. Finally, in (line 12) pheromone trails are then updated using the iteration-best rule (Current$_{best}$) (this to direct the ants to interesting areas in the search space) based on the quality measure Q until a user-specified number of iteration reached or the algorithm converges (line 7). Convergence occurs when the best rule generated in an iteration is the same for a number of conservative iterations. The training instances covered by the newly created rule are then removed from the training set and the whole rule creation procedure is repeated.

The rule creation starts with an empty rule at node i and probabilistically chooses to visit a node j based on the amount of pheromone and heuristic information on the edge E_{ij}, given by:

$$P(E_{ij}) = \frac{\tau_{ij}^{\alpha} \cdot \eta_{j}^{\beta}}{\sum\limits_{l \in N_i} \tau_{il}^{\alpha} \cdot \eta_{l}^{\beta}} \tag{18}$$

Algorithm 4: High-level pseudocode of *c*Ant-Miner

```
training_set ← all training instances
rule_list ← ∅
while |training_set| > max_uncovered_training_instances do
    $\tau$ ← initialise pheromone
    $\text{rule}_{best}$ ← ∅
    i ← 1
    while i < max_iterations OR convergence do
        CreateRules()
        PruneRules()
        ComputeConsequents()
        $\text{Current}_{best}$ ← BestRule()
        UpdatePhermones($\tau$,$\text{Current}_{best}$)
        if Q($\text{Current}_{best}$) > Q($\text{rule}_{best}$) then
            $\text{rule}_{best}$ ← $\text{Current}_{best}$
        end
        i ← i + 1
    end
    rule_list ← rule_list + $\text{rule}_{best}$
    training_set ← training_set − CoveredInstances($\text{rule}_{best}$)
end
```

where τ_{ij} is the pheromone value of the edge connecting node i to node j; η_j is the value of the heuristic information for node j; node l is a node in the neighbourhood of node N_i; the exponents α and β are used to control the influence of the pheromone and heuristic information, respectively. The heuristic information is based on the information gain of the (attribute-value) pair associated with each value.

If a node with a nominal attribute is selected, then a term in the form $(a_i = v_{ij})$ is added to the rule. The nodes with the same nominal attribute are then marked invalid, and removed from the neigbouring nodes set. If a node with a continuous attribute is selected, then a dynamic discretisation procedure based on the entropy measure is used to choose an operator and value to create a term in the form $(a_i \leq v_{ij})$ or $(a_i > v_{ij})$. This is done with a time complexity of $O(n \log n)$, where n is the number of the training instances, since the values need to be sorted and multiple candidate threshold values are evaluated. The ant continues to visit the rest of the graph, when selecting a node the ant cannot visit it again, until no node can be visited or until is not possible to visit any node.

The quality of a rule is measured as sensitivity × specificity, as used in Ant-Miner, given by

$$Q = \frac{TP}{TP + FN} \times \frac{TN}{FP + TN} \tag{19}$$

where *True Positive* TP is the number of instances covered by the rule that are correctly classified; *False Negative* FN is the number of instances that are not covered and have the same class value as predicted by the rule; *False Positive* FP is the number of covered instances that are incorrectly classified; *True Negative* TN is the number of instances that are not covered and do not have the same class value as predicted by the rule. The quality function Q measures how well the rule classifies the training instances that have the same class values as predicted by the rule and, at the same time how well the rule avoids covering training instances that have different class values.

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

The *c*Ant-Miner version based on the Pittsburgh approach (*c*Ant-Miner$_{\text{PB}}$) (Otero, Freitas and Johnson 2013) is an ACO classification algorithm that employs a different search strategy from *c*Ant-Miner. Rather than using the sequential covering approach to produce the list of *best rules* as *c*Ant-Miner does, *c*Ant-Miner$_{\text{PB}}$ searches for the *best list* of rules. This change might sound minor in words, but it does have a significant effect in the algorithm behaviour. In *c*Ant-Miner, each ant creates an individual rule and the rules compete to be the best, so that the best is added to the list. In *c*Ant-Miner$_{\text{PB}}$, each ant creates an entire list of rules, where rules are added independently of their individual qualities, considering the

Algorithm 5: High-level pseudocode of *c*Ant-Miner$_{\text{PB}}$

```
1  $\tau \leftarrow$ initialise pheromone
2  rule_list$_{gb} \leftarrow \emptyset$
3  $m \leftarrow 1$
4  while $m <$ max_iterations OR convergence do
5      rule_list$_{ib} \leftarrow \emptyset$
6      for $n \leftarrow$ 1 to colony_size do
7          rule_list$_n \leftarrow \emptyset$
8          instances $\leftarrow$ all_training_instances
9          rule_list$_n \leftarrow \emptyset$
10         while | instances | > max_uncovered_training_instances do
11             rule $\leftarrow$ CreateRule()
12             rule $\leftarrow$ PruneRule()
13             rule $\leftarrow$ ComputeConsequents()
14             rule_list$_n \leftarrow$ rule_list$_n$ + rule
15             instances $\leftarrow$ instances $-$ CoveredInstances(rule)
16         end
17         if Quality(rule_list$_n$) > Quality(rule_list$_{ib}$) then
18             rule_list$_{ib} \leftarrow$ rule_list$_n$
19         end
20     end
21     UpdatePhermones(rule_list$_{ib}$)
22     if Quality(rule_list$_{ib}$) > Quality(rule_list$_{gb}$) then
23         rule_list$_{gb} \leftarrow$ rule_list$_{ib}$
24     end
25     $m \leftarrow m + 1$
26 end
```

quality of the rule list as a whole.

The high-level pseudocode of *c*Ant-Miner$_{\text{PB}}$ is shown in Algorithm 5. The new strategy works as follows. An ant in the colony starts with empty an rule list (line 7). After creating and pruning a rule, the training instances covered by the rule are removed from the current instances (line 15) and the rule is added to the current rule list. After all ants create their rule lists, the best list of the iteration updates the pheromone values based on its quality (This allow the algorithm to converge easier); The algorithm also keeps track of the best list of rules created so far (rule_list$_{gb}$) — this is the list returned as the final rule list.

In order to use the pheromone model to create multiple rules, the pheromone matrix was extended to include a *tour* identification. This *tour* corresponds to the index of the rule being created (e.g., 1 for the first rule, 2 for the second rule, and so forth). Each entry in the pheromone matrix that corresponds to an edge of the construction graph is represented by a triple ($tour, vertex_i, vertex_j$) — where $vertex_i$ and $vertex_j$ correspond to the vertices connected by edge$_{ij}$. This way an ant will use the pheromone entries that correspond to the position of the rule in the list being created.

The probability for an ant k to follow the edge$_{ij}$ leading to the vertex v_j from the vertex v_i when creating the rule t is given by:

$$p_{tij}^{k} = \begin{cases} \frac{\tau_{tij} \cdot \eta_{v_j}}{\sum\limits_{l \in N_i^k} \tau_{til} \cdot \eta_{v_l}}, \\ 0, \text{if } j \notin N_i^k \end{cases} \tag{20}$$

where τ_{tij} is the amount of pheromone associates with the entry *(t,i,j)* in the pheromone matrix, η_{v_j} is the heuristic information associated with vertex v_j and N_i^k is the set of neighbouring vertices of vertex v_i.

The pheromone update function also takes into account the tour identification, where the pheromone values are updated by the iteration-best list. The pheromone update rule is given by:

$$\tau_{tij} = \begin{cases} \rho.\tau_{tij}, \text{ if } (t, i, j) \notin rule\_list_{ib} \\ \rho.\tau_{tij} + Q(rule\_list_{ib}), \text{ if } (t, i, j) \in rule\_list_{ib} \end{cases} \tag{21}$$

where ρ is the evaporation factor between [0,1], τ_{tij} is the amount of pheromone associated with entry *(t,i,j)* and $Q(rule\_list_{ib})$ is the quality of the iteration-best list of rules. As it can be seen in Equation 21, the search performed by *c*Ant-Miner$_{\text{PB}}$ is guided by the quality of a complete rule list, more specifically, the accuracy of the rule list measured on the training set.

4.3.3 Ant-Miner-Reg

Brookhouse and Otero introduced the first extension of the Ant-Miner algorithm for regression problems, called Ant-Miner-Reg (Brookhouse and Otero 2015). Ant-Miner-Reg uses the same sequential covering approach adopted by *c*Ant-Miner, with the dynamic discretisation procedure based on Quinlan's M5 (Quinlan et al. 1992) to adapt the rule creation to cope with continuous value prediction.

Ant-Miner-Reg creates a rule list as follows. First, n rules are created by the colony, where each ant traverses a graph of attribute nodes and values to build the antecedent of a rule — similarly to *c*Ant-Miner. If an ant visits a node representing a continuous attribute, a value is generated via a dynamic discretisation method that finds the value that minimises the variance on the generated subsets. After the antecedent of a rule is created, the prediction is generated by calculating the mean value of the class attribute over the instances covered by the rule. Once all rules are created, the best rule generated is used to update the pheromone values. The creation procedure is repeated until the maximum number of iterations is reached or the algorithm reaches stagnation, at which point the best rule is then returned and added to the list of rules under construction, removing any newly covered instances from the data set. The colony is then reset and the ACO process repeated on the remaining set of uncovered instances until all (or almost all) instances are covered by the rule list.

The quality of a regression rule is based on two factors. The first is the quality of the prediction measured using the Relative Root Mean Squared Error (RRMSE). The RRMSE of a rule is defined as:

$$L_{RRMSE} = \frac{L_{RMSE}}{\sqrt{\frac{1}{m}L_{default}}} \tag{22}$$

where 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. L_{RMSE} and $L_{Default}$ are defined as:

$$\begin{aligned} L_{RMSE} &= \sqrt{\frac{1}{m} \cdot \sum_{i=1}^{m} (y_i - \bar{y_i})^2} \\ L_{default} &= \sum_{i=1}^{m} (y_i - y')^2 \end{aligned} \tag{23}$$

where m is the number of instances covered by the rule, y_i is the value of the i-th instance, $\bar{y}$ is the predicted value of the i-th instance and y' is the mean value over all instances.

The RRMSE approximately normalises the RMSE of a rule between 0 and 1, where a value less than 1 corresponds to a rule making predictions better than predicting the mean value and a value greater than 1 corresponds to a rule making prediction worse than predicting the mean value.

The second factor is a measure of how general the rule is, i.e., the ratio of the number of covered instances over the total number of training instances. Like RRMSE, the coverage of a rule is normalised so that 0 represents a rule covering no instances and 1 represents a rule that covers all of the instances in the training data set. The relative coverage of a rule R is defined as:

$$relCov = \frac{1}{M} \cdot coverage(R) \tag{24}$$

Where M is the number of training instances. Both the RRMSE and relative coverage are combined into a single metric Q, which is used as a rule quality function, defined as:

$$Q = \alpha \cdot (1 - L_{RRMSE}) + (1 - \alpha) \cdot relCov \tag{25}$$

where α sets the weighting between RRSME and relative coverage. Varying α

between 0 and 1 will bias the rule quality towards either RRMSE or relative coverage, lower values of α will give more importance to accurate rules and greater values of α will give more importance to generic rules.

4.3.4 Other Extensions for Ant-Miner

Several other extensions of Ant-Miner have been proposed (Martens, Baesens and Fawcett 2011). Ant-Miner2 (Liu, Abbass and McKay 2002) and Ant-Miner3 (Liu, Abbas and McKay 2003) presented a simple heuristic function using density-based estimation. Ant-Miner+ (Martens et al. 2007) extended Ant-Miner in several aspects: it uses a class based heuristic, since an ant pre-selects the predicted class value and extracts a rule accordingly; it also employs a different pheromone initialization and update procedure based on the $\mathcal{MAX} - \mathcal{MIN}$ ant system ($\mathcal{MMAS}$) (Stützle and Hoos 2000), where the use of the lower and upper bound values of pheromone levels allows the algorithm to avoid early stagnation of the search; and 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).

Additionally, Ant-Miner+ employs a distinctive procedure for categorical and ordinal attributes. Categorical attributes have unordered nominal values (e.g., *male* and *female*), which were treated as a tuple $(attribute, =, value)$. Ordinal attributes have a natural order (e.g., *poor* $<$ *acceptable* $<$ *good*), where the algorithm creates upper and lower bounds on the values chosen by the ant: the first type represents a lower bound of the interval and takes the form ($attribute$, $\leq$, $value_i$); the second type represents an upper bound of the interval and takes the form ($attribute$, $\geq$, $value_j$), where $value_i$ and $value_j$ are values from the attribute domain. Continuous attributes are discretised in a pre-processing stage and then treated as ordinal attributes.

Improvements in the *c*Ant-Miner are found in (Salama et al. 2013), where the authors proposed the use of multiple pheromone levels to extract rules predicting different class values. Ant-Miner$_{mbc}$ (Liang et al. 2016) proposed the use of multiple rule lists to create an ensemble, where the ensemble uses weighted votes to provide the final classification. While further improvements on cAnt-Miner$_{\mathrm{PB}}$ in (Yang et al. 2017) by incorporating a principal of attraction and exclusion of pheromone, reaching a balance in the relation of exploration and development of constructing rules. Brookhouse and Otero (2016); Brookhouse and Otero (2018) extended their regression algorithm to enforce monotonicity constrains, a type of domain knowledge.

4.4 Summary

In this chapter, we present the ant colony optimization metaheuristic and applications in the data mining context. We covered graph-based ACO algorithms, suitable for combinatorial problems, and archive-based ACO algorithms, suitable for mixed-variable problems.

Furthermore, this chapter discussed the rule induction algorithms using ACO. The majority of the algorithms follow a sequential covering approach, using an ACO procedure to create a single rule; this procedure is then repeated to create a list. There are also algorithms that use an ACO procedure to create a complete rule list.

Noticeably, ACO approaches in the literature depended on a discertisation procedure to handle continuous attributes in the data, which is time consuming and ideally requires prior knowledge of the domain of the attribute. Discretisation procedures can be implemented in the traditional data mining scenario, where all training data is available to the algorithm and computational time is not a constraint (unless the data set is extremely large), but it is not practical to use the traditional discretisation procedure in a data stream context.

Chapter 5

Mixed-Attribute Ant-Miner For Classification Rule Discovery

While ACO classification algorithms can cope with continuous attributes, this is achieved by a time consuming discretisation procedure. To cope with this problem, this chapter introduces the first contribution of this thesis, the aim is to eliminate the discretisation procedure in ACO rule induction algorithms by using an archive-based pheromone model, which is capable to cope with continuous attributes directly and faster. Hence, this chapter proposes the Mixed-Attribute Ant-Miner Classification Rule Discovery Algorithm (Ant-Miner$_{\mathrm{MA}}$). Ant-Miner$_{\mathrm{MA}}$ was inspired and designed based on ACO for mixed-variable optimization (ACO$_{\mathbf{MV}}$) (Liao et al. 2014). Ant-Miner$_{\mathrm{MA}}$ uses a solution archive as a pheromone model, inspired by ACO$_{\mathbf{MV}}$, eliminating the need for a discretisation procedure, and in Ant-Miner$_{\mathrm{MA}}$ attributes can be treated directly as continuous, ordinal, or categorical.

We present the results for the archive pheromone model in the rule creation for classification problems. The comparison between the proposed Ant-Miner$_{\mathrm{MA}}$ and *c*Ant-Miner, where a similar predictive accuracy was observed in combination with a statistically significant improvement in runtime on large data sets.

The algorithms presented in this chapter were first presented in the following peer-reviewed papers: Ant-Miner$_{\text{MA}}$ was published in:

- Helal, A. and Otero, F. E. (2016). A Mixed-Attribute Approach in Ant-Miner Classification Rule Discovery Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ACM, GECCO '16, pp. 13–20.

The remainder of this chapter is organised as follows, Section 5.1 describes the Ant-Miner$_{\text{MA}}$ algorithm. Section 5.2 shows the results for comparison of the algorithms. Section 5.3 summarizes the chapter.

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

The proposed Ant-Miner$_{\text{MA}}$ algorithm uses an ACO$_{\mathbf{MV}}$ procedure to handle mixed attributes types, eliminating the need for an entropy-based discretisation when handling a continuous attribute as used in *c*Ant-Miner (Otero, Freitas and Johnson 2009), and also coping with ordinal attributes. ACO$_{\mathbf{MV}}$ is an ACO algorithm designed for mixed variable optimization problems; it handles ordinal, categorical, and continuous variables using a solution archive as the pheromone model. Ant-Miner$_{\text{MA}}$ uses a solution archive to sample conditions for the creation of the rules, instead of traversing a construction graph. A high-level pseudocode of Ant-Miner$_{\text{MA}}$ is shown in Algorithm 6.

Ant-Miner$_{\text{MA}}$ starts with an empty list of rules (line 1), and iteratively adds the best rule found along a sequential covering process to the list of rules in the *outer while loop*, which is executed while the number of uncovered training examples is greater than a user-defined maximum value. At each iteration, the best rule created by an ACO$_{\mathbf{MV}}$ procedure is added to the list of rules (lines 3–18). The ACO$_{\mathbf{MV}}$ procedure starts by initializing the solution archive with R random generated rules (line 3). Then, each ant generates a new rule (lines 7-11). Once

Algorithm 6: High-level pseudocode of Ant-Miner$_{\text{MA}}$.

```
   input  : training data
   output: list of rules
1  RuleList ← {}
2  while |$TrainingData$|<$MaxUncovered$ do
3      $SA$ ← Generate Random Rules
4      Restarted ← 0
5      while $t$ <$MaxIterations$ AND $Restarted$ <$2$ do
6          $SA_t$ ← {}
7          while $i$ < $number\ of\ ants$ do
8              $R_i$ ← Create New Rule
9              $R_i$ ← Prune($R_i$)
10             $SA_t$ ← $R_i$
11             $i \leftarrow i + 1$
12         end
13         $SA$ ← UpdateArchive($SA$ ,$SA_t$)
14         $t \leftarrow t + 1$
15         if $stagnation()$ then
16             Restart($SA$)
17             Restarted ← Restarted + 1
18         end
19     end
20     $R_{best}$ ← BestRule($SA$)
21     RuleList ← RuleList + $R_{best}$
22     TrainingData ← TrainingData − Covered($R_{best}$)
23 end
24 return RuleList
```

m new rules have been generated, where m is the number of ants in the colony, they are added into the solution archive (line 13). The R and m rules are sorted and the m worst ones are removed from the archive. The procedure to create a new rule is repeated until the maximum number of iterations has been reached.

5.1.1 Archive and Rule Structure

As aforementioned, the archive consist of R rules. Each rule consist of a vector of n-dimensional terms, where n is the number of attributes in the data set. Each

	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_2	...	...	F	-	-	...	...	f(S_1)	w_1
S_2	T	$<\leq$	v_3	v_4	...	...	F	-	-	...	...	T	$\leq$	v_5	...	...	f(S_2)	w_2
S_3	F	-	-	-	...	...	T	$=$	v_6	...	...	T	$\geq$	v_7	...	...	f(S_3)	w_3
⋮			⋱						⋱					⋱			⋮	⋮
S_R	T	$\leq$	v_8	-	...	...	F	-	-	...	...	T	$\leq$	v_9	...	...	f(S_R)	w_R
	Continuous attributes						Categorical attributes					Ordinal attributes						

Figure 9: Archive Structure: example of 3 rules of the archive, each rule showing a single example of different attribute type: A_r is a real-valued (continuous) attribute, A_c is a categorical attribute and A_o is an ordinal attribute.

term in a rule contains a flag to indicate if this term is enabled or not, an operator and value(s). For continuous attributes, the operator could be either $\leq$ (less than or equal to), $>$ (greater than) or $<\leq$ (in range); categorical attributes' operator is always $=$; and ordinal attributes have an operator of either $\leq$ or $\geq$ (greater/less than or equal).

Figure 9 illustrates a solution archive with 3 rules, each rule showing a single example of a different attribute type. The rules are stored according to their quality in the archive, where the best is stored at the top (highest ranking) and the worse at the bottom (worst ranking). The archive stores the quality of each rule $f(S)$ and the weight w for each solution calculated by Equation (26):

$$w_j = \frac{1}{qR\sqrt{2\pi}} e^{\frac{-(rank(j)-1)^2}{2q^2R^2}} \tag{26}$$

where q (Local Search) is a constant that is used to control the extent of the top-ranked solution influence on the construction of new solutions. The weight w_j of a solution S_j is used during the creation of new solutions, as an indicator for the level of attractiveness. The higher the weight of a solution S_j, the higher the probability of sampling a new solution around the values of S_j.

Figure 9 presents an example of a solution archive containing three rules.

Those rules have A_r a real-valued (continuous) attribute in the data set, A_c a categorical attribute in the data set and A_o is an ordinal attribute in the data set:

1. S_1 is IF $A_r > v_1$ AND $A_c = v_2$ THEN C_1
2. S_2 is IF $v_3 < A_r \leq v_4$ AND $A_o \leq v_5$ THEN C_2
3. S_3 is IF $A_c = v_6$ AND $A_o \geq v_7$ THEN C_3

The rule S_1 is a rule where the continuous attribute A_r is enabled `(Flag = T)`, the operator is "greater than" and the value v_1 is set. The categorical attribute A_c is enabled `(Flag = T)`, the default operator is "equal" and value v_2 is set. The ordinal attribute A_o is not enabled `(Flag = F)` and so it has no values for operator and value. The rule S_2 is a rule where the continuous attribute A_r is enabled `(Flag = T)`, the operator is "in range" and two values v_3 and v_4 are set. The categorical attribute A_c is not enabled showing `(Flag = F)` and so it has no values for operator and value. The ordinal attribute A_o is enabled `(Flag = T)`, the operator is "less than or equal" and value the v_5 is set. Rule S_3 follows a similar representation.

5.1.2 Archive Initialization

In the archive initialization procedure, each rule is randomly initialized. Rule initialization starts with an unbiased random probability to enable each the term, then it continues the rule initialization according to each attribute type. For all continuous terms, it uses an unbiased random probability to select the operator from the set $\{\leq, >, <\leq\}$. The value of the continuous attribute is generated using a random value sampled from a normal distribution in the domain of the attribute. In case of the $<\leq$ (range) operator, two values are generated and the values will only be accepted if they make the operator valid, e.g., the operator will not be valid with $9 < A_r \leq 4$.

Ordinal and categorical attributes values are encoded into non-negative integers, e.g., ordinal attribute values (small, medium, large) are encoded into (0, 1, 2) and for categorical attribute values (cat, dog, horse, cow) are encoded into (0, 1, 2, 3). Ordinal terms also use an unbiased random probability to choose the operator from the set $\{\leq, \geq\}$, then an unbiased random value for the index is generated. For categorical terms, a default = operator is added, then an unbiased random value for the index is generated.

After the random initialization of each rule, a rule is pruned to remove irrelevant terms enabled by the stochastic nature of the initialization. The pruning procedure is detailed in Section 5.1.5. Then, if the number of covered instances of a rule is greater or equal to a user-defined minimum limit, the rule is added to the archive. Finally, the rules in the archive are ordered according to their quality measured as the m-estimate measure in the rule selection and archive ordering (m-estimate provides a tradeoff between consistency/coverage), given by

$$Q = \frac{TP + m \cdot (\frac{P}{P+N})}{TP + FP + m} \tag{27}$$

where TP (true positives) is the number of instances covered by a rule that belong to the class predicted by the rule; FP (false positives) is the number of instances covered by a rule that do not belong to the class predicted by the rule; P and N are the total number of instances that are in the positive and negative class in the training data set, respectively.[1] The value $m = 22.466$ used in our approach has been determined experimentally in (Janssen and Fürnkranz 2010a) to be the optimum value for the m-estimate measure.

[1] An instance is considered negative if it is from a class different than the class predicted by the rule.

5.1.3 Rule Creation

Each attribute type used by Ant-Miner$_{\text{MA}}$ has a sampling procedure to create new rules. The sample procedures are based on the ACO$_{\mathbf{MV}}$ algorithm, as shown in Section 4.2.1. Rule creation uses the sampling procedures to determine which attribute and their values should be part of the antecedent of a rule. In order to create a rule, an ant performs the following steps.

1. For each term, it considers the probability of including the term or not. The decision is handled as a categorical choice, since it is dealing with boolean {`TRUE`, `FALSE`} values, where the value is sampled using the categorical sampling procedure.

2. If the term is enabled, an operator is chosen according to the attribute type. If the attribute type is categorical, the operator is =; if it is continuous, the decision is handled as a categorical choice of three operators $\{\leq, >, <\leq\}$, using the categorical sampling procedure; for ordinal attributes, the decision is handled as a categorical choice of two operators $\{\leq, \geq\}$. In both cases an operator should be sampled from the subset of rules in the archive that has the term enabled, to avoid using wrong terms that had no effect in the rule quality.

3. After selecting the operator, the value of the attribute is sampled according to the attribute type. Note that the sampling takes into consideration only the subset of the rules in the archive that have the term enabled using the same operator, which allow the terms to be correctly optimised.

4. After the creation of a term, the term is added to the antecedent of a rule and the rule is applied to the training set. If the rule covered less than a user-defined minimum number of instances, the term is disabled. This process (Steps 1-4) is repeated for the next term, until all terms are considered.

After the antecedent of a rule is created, the rule is applied to the training set. The prediction is set as the majority class of the rules covered instances.

5.1.4 Rule Creation Walk-through

Let us consider a subset of the Australian credit approval data set (Lichman 2013). This data subset has 2 attributes: A_2 is a continuous attributes; and A_1 is a categorical attribute. The target class attribute has 2 values. Table 4 shows a subset of the data set.

Table 4: A subset of the Australian credit approval data set.

A_1	A_2	Class
0	23.5	1
1	17.5	1
0	28.42	0
0	46.67	1
1	41.5	0
0	20.33	1
1	47.42	1
1	44.25	0
1	32.42	1
0	20	0
0	29.67	1
1	47.33	0
0	19.58	1
0	32.25	1
1	16.25	0

After initialisation, the archive has 4 randomly generated rules - shown in Figure 10. Using $q = 0.26$ and $\xi = 0.65$, we will go through the 3 steps that we described in Section 5.1.3.

Step 1: Starting from attribute A_1, we consider the probability to include the term or not. To decide whether or not to include a term from A_1, we use the

	Categorical attribute (A_1)			Continuous attribute (A_2)					
	Flag	Op	Value	Flag	Op	Value1	Value2	f(S)	w
S_0	T	=	0	F	_	_	_	0.64	0.21
S_1	F	_	_	T	>	24	_	0.54	0.14
S_2	T	=	1	F	_	_	_	0.23	0.021
S_3	T	=	0	T	>	21	_	0.15	0.012

Figure 10: Archive example

categorical sampling, as follows:

$$\begin{aligned} \alpha_T &= \frac{w_0}{u_T} = \frac{0.21}{3} = 0.07 \\ \alpha_F &= \frac{w_1}{u_F} = \frac{0.14}{1} = 0.14 \end{aligned} \tag{28}$$

where α_T is the weight for enabling attribute A_1, α_F is the weight for disabling attribute A_1, w_1 is the weight of the first rule in the archive that has A_1 enabled (rule index 1), w_0 is the weight of the first rule in the archive that has A_1 disabled (rule index 0), u_T is the number of rules that have A_1 enabled and u_F is the number of rules that have A_1 disabled. Using both α_T and α_F values, the probability of enabling/disabling the term is given by:

$$\begin{aligned} P_T &= \frac{\alpha_T}{\alpha_T + \alpha_F} = \frac{0.07}{0.21} = 0.33 \\ P_F &= \frac{\alpha_F}{\alpha_T + \alpha_F} = \frac{0.14}{0.21} = 0.66 \end{aligned} \tag{29}$$

where P_T is the probability to have A_1 enabled and P_F is the probability to have A_1 disabled. Using the above probabilities, we sample a random value to decide whether A_1 will be enabled or disabled. Assuming we get the value $P = 0.54$, the term for attribute A_1 is set to *false* (disabled) since $P > P_T$, and then we stop the sampling of the attribute.

Moving on to attribute A_2, we consider the probability to include the term or not using the same rationale:

$$\begin{aligned}\alpha_T &= \frac{w_1}{u_T} = \frac{0.14}{2} = 0.07\\ \alpha_F &= \frac{w_0}{u_F} = \frac{0.21}{2} = 0.105\\ P_T &= \frac{\alpha_T}{\alpha_T + \alpha_F} = \frac{0.07}{0.175} = 0.4\\ P_F &= \frac{\alpha_T}{\alpha_T + \alpha_F} = \frac{0.105}{0.175} = 0.6\end{aligned} \tag{30}$$

Using the above probabilities, we sample a random value to decide whether A_2 will be enabled or disabled. Assuming we get the value $P = 0.35$, the term for attribute A_2 is set to *true* (enabled) since $P < P_T$.

Step 2: Since the term for attribute A_2 is enabled and A_2 is a continuous attribute, an operator is then sampled. In order to choose an operator, we use categorical sampling as follows:

$$\begin{aligned}\alpha_{>} &= \frac{w_1}{u_{>}} + \frac{q}{\kappa} = \frac{0.14}{2} + \frac{0.26}{2} = 0.27\\ \alpha_{\leq} &= \frac{q}{\kappa} = \frac{0.26}{2} = 0.13\\ \alpha_{<\leq} &= \frac{q}{\kappa} = \frac{0.26}{2} = 0.13\end{aligned} \tag{31}$$

where $\alpha_{>}$ is the weight for the $>$ operator in attribute A_2, $\alpha_{\leq}$ is the weight for the $\leq$ operator in attribute A_2, $\alpha_{<\leq}$ is the weight for the $<\leq$ operator in attribute A_2, w_1 is the weight of the first rule in the archive has A_2 operator equal to $>$ (rule index 1), $u_{>}$ is the number of rules that have A_2 operator equal to $>$, q is the variable that is used to control the extend of the top-ranked rule influence on the construction of new rules and κ is the number of values of this attribute that are not used in the archive (2 in this case). Using the weights $\alpha_{>}$, $\alpha_{\leq}$, $\alpha_{<\leq}$ we calculate their probabilities,

$$\begin{aligned}P_{>} &= \frac{\alpha_{>}}{\alpha_{>} + \alpha_{\leq} + \alpha_{<\leq}} = \frac{0.27}{0.53} = 0.50\\ P_{\leq} &= \frac{\alpha_{\leq}}{\alpha_{>} + \alpha_{\leq} + \alpha_{<\leq}} = \frac{0.13}{0.53} = 0.25\\ P_{<\leq} &= \frac{\alpha_{<\leq}}{\alpha_{>} + \alpha_{\leq} + \alpha_{<\leq}} = \frac{0.13}{0.53} = 0.25\end{aligned} \tag{32}$$

where $P_{>}$ is the probability to have A_2 operator set to $>$, $P_{\leq}$ is the probability to have A2 operator set to $\leq$ and $P_{<\leq}$ is the probability to have A_2 operator set to $<\leq$. Using the above probabilities, we sample a random value to select an operator. Assuming we get the value $P = 0.152$, we set the operator to $>$ since $P < P_{>}$.

Step 3: After selecting the operator for attribute A_2, we start sampling its value. Firstly, we choose a rule from the archive to build our solution around it. There are only two rules with A_2 term enabled and the probability of selecting one of them is given by

$$\begin{aligned}P_1 &= \frac{w_1}{w_1 + w_3} = \frac{0.14}{0.152} = 0.921\\ P_3 &= \frac{w_3}{w_1 + w_3} = \frac{0.012}{0.152} = 0.078\end{aligned} \tag{33}$$

where P_1 is the probability to select the rule with index 1 and P_3 is the probability to select the rule with index 3. Using the above probabilities, we sample a random value to select a rule form the archive. Assuming we get the value $P = 0.567$, the rule index 1 is selected since $P < P_1$. Using the values from the archive to calculate the value of the term.

$$\begin{gathered}\sigma = \xi \frac{|S_{1,2} - S_{3,2}|}{1} = 0.655 \times 3 = 1.965\\ value = Gaussian(S_{1,2}, \sigma) = Gaussian(24, 1.965) = 23.23\end{gathered} \tag{34}$$

σ is the average distance between the value of the attribute A_2 in rule 1 and the value of attribute A_2 in all the other rules in the archive (only rule 3 in this case), and ξ is a user-defined value representing the convergence speed of the algorithm. $S_{1,2}$ is the selected rule value used as the mean for the Gaussian sampling.

5.1.5 Rule Pruning

Ant-Miner$_{\mathrm{MA}}$ applies different heuristics for the rule refinement and rule selection following a similar approach proposed in (Stecher, Janssen and Fürnkranz 2014). Ant-Miner$_{\mathrm{MA}}$ uses a pruning function to remove irrelevant terms that are added to the rule, due to the stochastic nature of the rule creation. The random sampling in enabling terms, could allow irrelevant terms to be added. The pruning function starts by disabling the last enabled term in the rule and, if the quality of the rule does not decrease, it permanently disable the term. The pruning function continue to remove terms until the removal of a term decreases the quality of the rule or the rule has only one term remaining. This is the same pruning procedure used in *c*Ant-Miner (Otero, Freitas and Johnson 2009).

For the pruning function, the sensitivity × specificity function is used to measure the quality of rules, as employed in Ant-Miner, given by

$$Q_{Pruning} = \frac{TP}{TP + FN} \cdot \frac{TN}{FP + TN} \tag{35}$$

where TP is the number of instances covered by a rule that belong to the class predicted by the rule; FP is the number of instances covered by a rule that do not belong to the class predicted by the rule; TN is the number of instances not covered by a rule that do not belong to the class predicted by the rule; FN is the number of instances not covered by a rule that belong to the class predicted by the rule.

Table 5: Parameter values used in the experiments. Ant-Miner$_{\text{MA}}$ uses the first three parameters in the table, while the remaining ones are used by both Ant-Miner$_{\text{MA}}$ and *c*Ant-Miner.

Parameters	Value
q (Local Search)	0.025495
ξ (Convergence)	0.6795
R (Archive Size)	90
Minimum Covered	10
Max Uncovered	10
Max Iterations	1500
Number of Ants	60
Stagnation Test	10

5.1.6 Restart Procedure

Ant-Miner$_{\text{MA}}$ uses a simple restart strategy to avoid search stagnation. The restart procedure is triggered when the best rule of the current iteration is exactly the same as the best rule constructed in a user-defined number of previous iterations, which works as a stagnation test. After the restart procedure is triggered, all rules in the archive are reinitialized except the best-so-far rule (top rule in the archive). The restart is performance only once.

5.2 Experimental Results for Ant-Miner$_{\text{MA}}$

The computational results were computed using 30 publicly available data sets from the UCI Machine Learning Repository (Lichman 2013), presented in Tables 6. Ant-Miner$_{\text{MA}}$ uses the first three parameters in Table 5 for the archive settings, while the remaining parameters are used by both Ant-Miner$_{\text{MA}}$ and *c*Ant-Miner. These were either empirically chosen based on preliminary experiments (archive setting parameters) or based on *c*Ant-Miner's default values (Otero, Freitas and Johnson 2009).

Table 6: Summary of the data sets used in the experiments: data sets from 1 to 18 are considered small data sets, while the remaining ones are considered large data sets due the larger number of attributes and/or number of instances.

				Attributes			
#	Data set	Size	#Classes	Total	#Ordinal	#Categorical	#Continuous
1	breast-tissue	106	6	9	0	0	9
2	iris	150	3	4	0	0	4
3	wine	178	3	13	0	0	13
4	parkinsons	195	2	22	0	0	22
5	glass	214	7	9	0	0	9
6	breast-l	286	2	9	4	5	0
7	heart-h	294	5	13	3	3	7
8	heart-c	303	5	13	3	3	7
9	liver-disorders	345	2	6	0	0	6
10	ionosphere	351	2	34	0	0	34
11	dermatology	366	6	34	33	0	1
12	cylinder-bands	540	2	35	2	14	19
13	breast-w	569	2	30	0	0	30
14	balance-scale	625	3	4	4	0	0
15	credit-a	690	2	14	4	4	6

Table 6: Summary of the data sets used in the experiments: data sets from 1 to 18 are considered small data sets, while the remaining ones are considered large data sets due the larger number of attributes and/or number of instances.

				Attributes			
#	Data set	Size	#Classes	Total	#Ordinal	#Categorical	#Continuous
16	pima	768	2	8	0	0	8
17	annealing	898	6	38	0	29	9
18	credit-g	1000	2	20	11	2	7
19	MiceProtein	1080	8	80	0	3	77
20	HillValley	1212	2	100	0	0	100
21	Magic	19020	2	10	0	0	10
22	Nomao	34465	2	118	0	29	89
23	bank-additional	41188	2	20	0	10	10
24	eb	45781	31	3	0	1	2
25	adult	48842	2	14	0	8	6
26	connect4	67557	3	42	0	42	0
27	diabetes	101766	3	47	2	34	11
28	SkinNonSkin	245057	2	3	0	0	3
29	ForestType	581012	7	54	0	44	10
30	PokerHand	1025010	10	10	5	0	5

The *c*Ant-Miner implementation used in the experiments was the *c*Ant-Miner2$_{MDL}$ variation (Otero, Freitas and Johnson 2009)[2], which can create intervals with lower and upper bounds for continuous attributes.

We ran both the Ant-Miner$_{\mathrm{MA}}$ and *c*Ant-Miner algorithms for 15 times in a tenfold cross-validation setting — the averaged results (over the 150 individual runs) are shown in Tables 7 and 9. We ran the Wilcoxon signed rank test (Wilcoxon 1992) on the results of the 30 data sets to show if there are statistically significant differences in terms of both predictive accuracy and runtime. For a fair comparison, both algorithms are implemented in Java running in the same environment.

According to the results presented in Table 7, Ant-Miner$_{\mathrm{MA}}$ achieved a better average rank (1.43) for the predictive accuracy, while *c*Ant-Miner has an average rank of 1.57. In terms of runtime, the results in Table 8 shows that Ant-Miner$_{\mathrm{MA}}$ has a rank of 1.10, while *c*Ant-Miner has a rank of 1.90. Most of the data sets show an order of magnitude improvement of Ant-Miner$_{\mathrm{MA}}$ over *c*Ant-Miner in terms of runtime. Considering both number of attributes and instances size, the largest data sets are *forest type*, *poker hand* and *diabetes*, respectively. Most notably, a single *c*Ant-Miner execution on *diabetes* takes up to 3.5 days, while Ant-Miner$_{\mathrm{MA}}$ takes just over 1 hour. Ant-Miner$_{\mathrm{MA}}$ would take 45 minutes for a single run in *poker hand*, while *c*Ant-Miner almost 8 hours. These results show that the use of the solution archive as a pheromone model does not affect the accuracy, while improving the computational time since the discretisation procedure is eliminated.

Table 9 shows that Ant-Miner$_{\mathrm{MA}}$ models are significantly bigger than *c*Ant-Miner, although Ant-Miner$_{\mathrm{MA}}$ achieved better runtime and similar accuracy. In MiceProtein data set, Ant-Miner$_{\mathrm{MA}}$ shows an increase in rules and terms count (25.3, 3.4) with low predictive accuracy (62.57), while *c*Ant-Miner shows much concise model (7.9, 1.5) with higher predictive accuracy (99.07). Also notably

[2] Available from `https://github.com/febo/myra`.

Table 7: Average predictive accuracy (average ± standard error) of *c*Ant-Miner and Ant-Miner$_{\mathrm{MA}}$ measured over 15 runs of tenfold cross-validation. The last row of the table shows the average rank of the algorithm. The best value of a given data set is shown in bold.

	Accuracy	
Data set	Ant-Miner$_{\mathrm{MA}}$	*c*Ant-Miner
breast-tissue	60.24±0.97	**64.24±0.24**
iris	93.6±0.31	**94.27±0.11**
wine	90.78±0.40	**93.52±0.07**
parkinsons	**86.29±0.64**	85.22±0.40
glass	**63.24±0.50**	59.18±0.32
breast-l	71.46±0.34	**76.17±0.11**
heart-h	64.37±0.29	**64.81±0.33**
heart-c	56.94±0.54	**57.42±0.32**
liver-disorders	**63.13±0.49**	62.26±0.18
ionosphere	**89.28±0.37**	88.84±0.25
dermatology	**89.42±0.47**	89.02±0.25
cylinder-bands	69.32±0.35	**70.28±0.22**
breast-w	93.53±0.22	**94.28±0.11**
balance-scale	**80.10±0.22**	68.34±0.08
credit-a	85.19±0.22	**85.74±0.11**
pima	**75.30±0.21**	67.45±0.07
annealing	96.68±0.14	**97.02±0.09**
credit-g	**74.19±0.14**	69.39±0.17
MiceProtein	62.57±0.37	**99.07±0.43**
HillValley	**52.65±0.19**	51.35±0.11
Magic	**82.67±0.05**	70.41±0.01
Nomao	87.56±0.05	**90.66±0.02**
bank-additional	89.49±0.02	**89.87±0.01**
eb	**65.00±0.05**	64.58±0.01
adult	**84.74±0.03**	79.73±0.04
connect4	**68.18±0.02**	67.83±0.01
diabetes	**55.83±0.09**	54.23±0.13
SkinNonSkin	**98.91±0.02**	97.54±0.00
ForestType	**68.55±0.07**	63.09±0.07
PokerHand	**51.97±0.04**	50.2±0.00
Rank	**1.43**	1.57

Table 8: Average computational time (average ± standard error) of *c*Ant-Miner and Ant-Miner$_{\text{MA}}$ measured over 15 runs of tenfold cross-validation. The last row of the table shows the average rank of the algorithm. The best value of a given data set is shown in bold.

	Computational time (seconds)	
Data set	Ant-Miner$_{\text{MA}}$	*c*Ant-Miner
breast-tissue	**0.38±0.01**	0.67 ±0.02
iris	**0.28±0.00**	0.49 ±0.00
wine	**0.33±0.01**	0.56 ±0.04
parkinsons	**0.78±0.01**	2.87 ±0.25
glass	**0.50±0.00**	2.66 ±0.42
breast-l	**0.54±0.01**	1.28 ±0.14
heart-h	**0.72±0.00**	12.61±0.62
heart-c	**0.76±0.02**	10.91±0.64
liver-disorders	**0.47±0.01**	1.81 ±0.08
ionosphere	**1.27±0.03**	5.97 ±0.65
dermatology	**1.31±0.02**	16.67±1.49
cylinder-bands	**3.15±0.08**	29.54±1.31
breast-w	**2.31±0.04**	5.40 ±0.29
balance-scale	**0.50±0.01**	5.95 ±0.38
credit-a	**1.10±0.01**	11.57±0.79
pima	**0.93±0.01**	3.69 ±0.42
annealing	**4.79±0.16**	10.76±0.77
credit-g	**1.69±0.02**	39.1 ±2.11
MiceProtein	26.79±0.61	**8.53±0.65**
HillValley	37.93±0.71	**19.12±0.91**
Magic	**25.56±0.61**	155.07±2.01
Nomao	2308.57±98.25	**779.77±28.15**
bank-additional	**185.27±2.49**	1970.72±60.55
eb	**25.38±0.36**	321.58±1.11
adult	**236.54±3.83**	5048.44±165.33
connect4	**1273.66±7.97**	14380.04±535.72
diabetes	**4008.11±141.34**	∗388023.6±3580.67
SkinNonSkin	**125.41±1.56**	1484.52±5.27
ForestType	**30649.92±695.20**	53336.53±3946.86
PokerHand	**2577.19±43 .07**	27872.59±2286.18
Rank	**1.10**	1.90

∗ Result of a single tenfold execution due to high computational time.

Table 9: Average rule and term count of *c*Ant-Miner and Ant-Miner$_{\mathrm{MA}}$ measured over 15 runs of tenfold cross-validation. The last row of the table shows the average rank of the algorithm. The best value of a given data set is shown in bold.

	Rule Count		Term Count	
Dataset	Ant-Miner$_{\mathrm{MA}}$	*c*Ant-Miner	Ant-Miner$_{\mathrm{MA}}$	*c*Ant-Miner
breast-tissue	**6.9**	6.9	**1.2**	**1.0**
iris	**4.9**	5.0	0.9	**0.8**
wine	5.8	**5.1**	1.4	**0.8**
parkinsons	7.2	**5.8**	1.7	**1.2**
glass	**7.8**	10.1	1.8	**1.4**
breast-l	8.3	**5.9**	1.8	**1.3**
heart-h	8.6	**8.3**	**2.0**	2.4
heart-c	9.4	**9.7**	**2.1**	2.5
liver-disorders	9.9	**7.7**	1.4	**1.2**
ionosphere	8.0	**6.0**	1.8	**1.4**
dermatology	10.3	**9.7**	2.7	**2.1**
cylinder-bands	13.4	**7.6**	2.1	**2.1**
breast-w	9.0	**8.8**	2.4	**1.3**
balance-scale	**11.3**	12.3	1.5	**1.0**
credit-a	10.6	**7.6**	2.1	**1.6**
pima	13.6	**10.0**	1.7	**1.2**
annealing	13.5	**11.2**	**1.4**	1.9
credit-g	16.3	**9.0**	1.7	**1.7**
MiceProtein	25.3	**7.9**	3.4	**1.5**
HillValley	**20.4**	21.4	1.6	**1.1**
Magic	39.4	**36.3**	1.4	**1.1**
Nomao	43.2	**15.2**	1.9	**1.6**
bank-additional	32.9	**14.7**	**1.6**	1.8
eb	**48.6**	150.4	1.6	**1.2**
adult	66.6	**19.4**	**1.4**	1.5
connect4	37.6	**17.8**	**1.9**	2.2
diabetes	92.8	**18.5**	**1.5**	2.1
SkinNonSkin	**35.7**	81.5	**1.3**	1.4
ForestType	113.9	**52.4**	1.7	**1.7**
PokerHand	39.3	**28.6**	1.5	**1.3**
Rank	1.73	**1.27**	1.70	**1.30**

Table 10: Results of the Wilcoxon Signed Rank tests at the $\alpha = 0.05$ significance level comparing Ant-Miner$_{\mathrm{MA}}$ and *c*Ant-Miner on predictive accuracy and computational time. Statistically significant differences are shown in bold, indicating the case where Ant-Miner$_{\mathrm{MA}}$'s performance is statistically significantly better than *c*Ant-Miner.

	Size	W+	W-	Z	*p*
Accuracy	30	289.5	175.5	-1.1724	0.24200
Runtime	30	59	406	-3.5686	**0.00036**
Rules Count	30	371	94	-2.85	**0.0044**
Terms Count	30	358	107	-2.58	**0.00988**

this is one of the small number of instances that shows an increase in runtime of Ant-Miner$_{\mathrm{MA}}$ as its runtime (26.8) while *c*Ant-Miner is (8.5). This case does repeat itself in instances with high attribute sizes such as HillValley, Nomao.

The results suggest that there is a small limitation when the number of attributes increases over 50, where the observed improvement in computational time is only around 25% in the *forest type* data set. In cases where there is a large number of attributes but a smaller number of instances—which means that the discretisation overhead is less noticeable—such as in the data sets *Namao* (119 attributes, 34465 instances), *Hill Valley* (101 attributes, 1212 instances) and *Mice Protein* (81 attributes, 1080 instances), Ant-Miner$_{\mathrm{MA}}$ running time increases in relation to *c*Ant-Miner. We hypothesised that this is due to the use of the graph pheromone model in *c*Ant-Miner, which allows the algorithm to quickly identify irrelevant attributes and not use them in the rule creation process.

Table 10 indicates that there is no statistically significantly differences between Ant-Miner$_{\mathrm{MA}}$ and *c*Ant-Miner in terms of predictive accuracy ($p = 0.24200$). In the case of computational time, Ant-Miner$_{\mathrm{MA}}$'s improvement is statistically significant ($p = 0.00036$). In case of the rules and terms count Ant-Miner$_{\mathrm{MA}}$ loses with statistical significance to *c*Ant-Miner. So, we believe this shows that archive model fails to identify good attributes quickly. Overall, we consider these results positive. The use of a rule creation process inspired by ACO$_{\mathbf{MV}}$ led to a statistically significant runtime improvement in Ant-Miner$_{\text{MA}}$ compared to *c*Ant-Miner, without affecting the predictive accuracy.

5.3 Summary

We introduced Ant-Miner$_{\text{MA}}$ to tackle mixed-attribute classification problems, based on ACO$_{\mathbf{MV}}$. The use of a solution archive allows the algorithm to deal with categorical, continuous and ordinal attributes directly, without requiring a discretisation procedure. The rule creation procedure then uses ACO$_{\mathbf{MV}}$ strategies to sample values for each attribute to create the antecedent of a rule.

In this chapter, we showed the effects of using an archive pheromone model. The use of a solution archive allows the algorithm to deal with continuous attributes without requiring a discretisation procedure. The results suggest that there is a small limitation when the number of attributes increases. We hypothesised that this is due to the use of the graph pheromone model in *c*Ant-Miner, which allows the algorithm to quickly identify irrelevant attributes and not use them in the rule creation process leading to larger models as shown by the rule size.

Chapter 6

Automatic Design of Ant-Miner Mixed Attributes for Classification Rule Discovery

This work in Chapter 5, identified a limitation of using the archive only as the pheromone model, as the archive pheromone model fails to identify good attributes quickly. In this chapter, I will show how we can we combine both the graph pheromone model and the archive pheromone model into a single framework, to overcome this limitation. We introduce the Automatic Design of Ant-Miner Mixed Attributes for Classification Rule Discovery (Ant-Miner$_{\mathrm{MA+G}}$). Ant-Miner$_{\mathrm{MA+G}}$ is an automatic design framework to incorporate the graph-based model along with the archive-based model in the rule creation process.

We then present results for Ant-Miner$_{\mathrm{MA+G}}$, showing that the new framework combining both the archive and the graph pheromone models achieved a statistically significant improvement over *c*Ant-Miner in runtime and predictive accuracy.

Ant-Miner$_{\mathrm{MA+G}}$ was published in :

- Helal, A. and Otero, F. E. B. (2017). Automatic Design of Ant-Miner Mixed Attributes for Classification Rule Discovery. In *Proceedings of the Genetic* *and Evolutionary Computation Conference*, ACM Press, GECCO '17, pp. 433–440.

Section 6.1 describes the framework of Ant-Miner$_{\text{MA+G}}$. Section 6.2 shows the results of the comparison between Ant-Miner$_{\text{MA+G}}$ and *c*Ant-Miner. Finally, Section 6.3 compares the ACO-based Ant-Miner$_{\text{MA+G}}$, Ant-Miner$_{\text{MA}}$ and *c*Ant-Miner against C5.0 rules (Quinlan 1993), Jrip (Cohen 1995), and PART (Frank and Witten 1998) to show their performance compared to well-known induction algorithms from the literature.

6.1 Ant-Miner$_{\text{MA+G}}$

The graph-based pheromone model was introduced to guide the ants in a discrete search space, where solution components are represented by nodes of the graph. While the archive-based pheromone model was introduced to guide the ants in the mixed variables search space, employing different sampling strategies according to the variable type on a solution archive.

The second approach, Ant-Miner$_{\text{MA+G}}$ (archive + graph) is implemented to combine both the graph and the archive approaches into one framework, since both graph-based and archive-based pheromone models have their merits. The archive-based model showed limitations when the number of attributes increased over 50, the runtime increased compared to graph-based model. Also, the graph-based model showed improvement in predictive accuracy with over 50 attributes, and the graph-based pheromone model had the advantage of finding the best attributes to use faster than the archive-based pheromone model. Combining concepts from both approaches could potentially lead to improved runtime and a better capacity to handle data sets with a large number of attributes.

There are a number of design questions when building a framework to combine both archive-based and graph-based pheromone models. Ant-Miner$_{\text{MA}}$ uses the archive-based pheromone model to sample rule term components, such as the attribute, the operator and the value; By contrast, ants in *c*Ant-Miner traverse the graph-based pheromone model to create rules, using the pheromone deposited on each edge as an indication of the current best attributes-value pairs, and in the case of continuous attributes, they use a dynamic discretisation procedure based on the entropy measure.

Algorithm 7: High-level pseudocode of Ant-Miner$_{\text{MA+G}}$.

```
input  : training data
output: list of rules
RuleList ← {}
Restarted ← 0
while |TrainingData| < MaxUncovered do
    SA ← Generate Random Rules
    Restarted ← 0
    while t <MaxIterations and Restarted <2 do
        $SA_t$ ← {}
        while i < number of ants do
            $R_i$ ← Create New Rule (Section 6.1.1)
            $R_i$ ← Prune($R_i$) (Section 6.1.2)
            $SA_t$ ← $R_i$
            $i \leftarrow i + 1$
        end
        SA ← UpdateArchive(SA, $SA_t$) (Section 6.1.4)
        SA ← UpdateGraph(SA, $SA_t$) (Section 6.1.4)
        $t \leftarrow t + 1$
        if stagnation() then
            Restart(SA) (Section 6.1.5)
            Restarted ← Restarted + 1
        end
    end
    $R_{best}$ ← BestRule(SA) (Section 6.1.2)
    RuleList ← RuleList + $R_{best}$
    TrainingData ← TrainingData − covered($R_{best}$)
end
return RuleList
```

The algorithm design questions that we are interested in this work are:

Table 11: Algorithmic components of the proposed Ant-Miner$_{\text{MA+G}}$.

Design components	
Ordinal attributes:	1: Using ordinal attribute with $<\leq$ condition 2: Using ordinal attribute without$<\leq$ condition 3: Not using ordinal attribute
Operator selection:	2: Using archive for sampling conditions 1: Using graph for choosing conditions
Categorical attributes:	1: Archive sampling 2: Archive sampling and not equal condition 3: Using graph for choosing categorical value
Prune Quality Function:	See Table 12
Selection Quality Function:	See Table 12
Pheromone limits:	1: Max-Min limits 2: No limits
Archive top rule updating graph pheromone model:	1: First iteration after the archive is created 2: At the end of each iteration 3: Never updates
Updating graph pheromone model with:	1: Best iteration rule 2: All rules added to the archive
Value used to update graph pheromone model:	1: Weight of the rule in the archive 2: Quality of the rule
Pheromone restart:	1: Restart both the pheromone models 2: No restart

1. *Should the archive pheromone model be used only for continuous values, or should it also be used for nominal and ordinal values?*

2. *Should the operator be selected using the archive pheromone model, or should it be added to the graph pheromone model?*

3. *How should both pheromone models be updated?*

Instead of following a manual approach of testing each possible configuration of Ant-Miner$_{\mathrm{MA}}$, which would require a large amount of human and computational time, we propose the use of an automated algorithm configuration tool to obtain a high-performing Ant-Miner$_{\mathrm{MA}}$ variant. We have been inspired by the work of López-Ibáñez and Stützle (2012), which used I/F-Race (Birattari et al. 2010; López-Ibáñez et al. 2016) to deal with the automatic design and configuration of parameters to obtain a multi-objective ant colony optimization algorithm. In order to use an automatic configuration tool, we created a framework of design algorithmic components from which new variants of Ant-Miner$_{\mathrm{MA}}$ could be generated. These are presented on Table 5.

Algorithm 7 shows the high-level pseudocode of the Ant-Miner$_{\mathrm{MA+G}}$ algorithm. At a high level, Ant-Miner$_{\mathrm{MA+G}}$ starts with an empty list of rules and iteratively adds the best rule found along the iterative process while the number of uncovered training examples is greater than a maximum value. It uses the same rule creation loop as Ant-Miner$_{\mathrm{MA}}$. At each iteration, a single rule is created by a new procedure combining both graph-based and archive-based pheromone models (lines 3–18). Once m new rules have been generated, where m is the number of ants in the colony, they are added into the solution archive (line 13). The graph is also updated with the same rules (line 14). The R and m rules are sorted and the m worst ones are removed from the archive. The procedure to create a new rule is repeated until the maximum number of iterations has been reached.

The following subsections present the different design choices that were implemented in Ant-Miner$_{\mathrm{MA+G}}$. We grouped the design choices into three main categories: (1) rule construction; (2) rule quality function configurations; and (3) pheromone model.

Table 12: Rule evaluation functions used for pruning and selection procedures.

Functions		Parameter
Precision (P)	$\frac{TP}{TP+FP}$	-
Confidence Coverage (CC)	$\frac{TP}{TP+FP}+\frac{TP}{S}$	-
Cost measure (CM)	$(c \times TP) - ((1-c) \times FP)$	$c = 0.437$
Fmeasure (FM)	$\frac{(1+\beta^2)\cdot\frac{TP}{TP+FN}\cdot\frac{TP}{TP+FP}}{\beta^2\cdot\frac{TP}{TP+FN}\cdot\frac{TP}{TP+FP}}$	$\beta = 0.5$
Jaccard (J)	$\frac{TP}{TP+FP+FN}$	-
Klosgen (K)	$(\frac{TP+FP}{S})^{\omega}.(\frac{TP}{TP+FP}-\frac{TP+FN}{S})$	$\omega = 0.4323$
Laplace (L)	$\frac{TP+1}{TP+FP+k}$	$k = number\ of\ classes$
MEstimate (ME)	$\frac{TP+m.\frac{TP}{S}}{TP+FP+m}$	$m = 22.466$
Relative Cost Measure (RCM)	$cr \times \frac{TP}{TP+FN} - (1-cr) \times \frac{FP}{TN+FP}$	$cr = 0.342$
Precision Inverted (PI)	$\frac{(FP+TN)-TP}{(S)-(TP+FP)}$	-
MEstimate Inverted (MEI)	$\frac{(FP+TN)+m\times\frac{TP+FN}{S}}{(S)-(TP+FP+m)}$	$m = 22.466$
Laplace Inverted (LI)	$\frac{(FP+TN)-FP+1}{(S)-(TP+FP+k)}$	$k = number\ of\ classes$
Sensitivity and Specificity(SS)	$\frac{TP}{TP+FN} \times \frac{TN}{TN+FP}$	-

6.1.1 Rule Construction

A crucial design decision when combining both graph-based and archive-based pheromone models is defining each pheromone model's contribution to solution creation. This problem is exacerbated by the fact that there are two different ways to create a solution, each with their own strengths, and there are different ways to combine them. This section shows different algorithmic choices for operator, ordinal and categorical attribute selection. Note that we do not consider using a graph-based model to select continuous attributes values, since this would involve using a discretisation procedure.

The basic framework of Ant-Miner$_{\text{MA+G}}$ consists of a fully connected construction graph. Let a_i be an attribute, $i = 1, ..., n$ where n is the number of attributes; each attribute is added as a node (a_i) to the graph. Suppose an ant l is generating a rule r_l. It starts with an empty rule at node i and probabilistically chooses to visit a node j based on the amount of pheromone on the edge E_{ij}. There are different strategies available for all attributes, as discussed next.

Continuous attributes

Ant-Miner$_{\text{MA+G}}$ has two possible approaches to handle operator selection, for the continuous attributes:

1. The archive pheromone model is used to select the operator according to the attribute configuration. For continuous attributes, one of the three possible $\{\leq, >, <\leq\}$ operators is selected.

2. The graph pheromone model is used to select the operator. In this case, each node of the graph consists of a pair (attribute, operator). Let a_r be a continuous attribute, each attribute is associated with three operators and three nodes will be added to the graph: (a_r, $\leq$), (a_r, $>$) and (a_r, $<\leq$). In this special case for continuous attributes, the categorical and ordinal attributes are affected as follows. Let a_c be a categorical attribute, each attribute is associated with the equal operator and added as a node to the graph: (a_c, $=$). Finally, let a_o be an ordinal attribute, each attribute is associated with two operators and two nodes will be added to the graph: (a_o, $\leq$) and (a_o, $\geq$).

After the attribute operator is selected, the attribute value selection is configured, we always use the archive model to sample the continuous value, using a continuous sampling procedure on the subset of the archive rules with that term enabled using the same attribute and operator.

Ordinal attributes

The Ant-Miner$_{\mathrm{MA+G}}$ algorithm implements a procedure for ordinal attributes, where it uses the continuous sampling procedure from ACO$_{\mathbf{MV}}$. This was based on ACO$_{\mathbf{MV}}$ approach to benefit from the natural order of the ordinal attribute values. The possible conditions for ordinal attributes are $\{a_i \leq v,\ a_i > v,\ v_1 < a_i \leq v_2)\}$. the Ant-Miner$_{\mathrm{MA+G}}$ algorithm implements three possible approaches to handle ordinal attributes:

1. Sampling from three possible operators: $\{\leq, >, <\leq\}$;
2. Sampling from two possible operators: $\{\leq, >\}$;
3. Handling ordinal attributes as categorical attributes without any special treatment—i.e., conditions are always in the form $a_i = v$.

Categorical attributes

Ant-Miner$_{\mathrm{MA+G}}$ implements a procedure for categorical attribute values, where it uses the discrete sampling procedure from ACO$_{\mathbf{MV}}$. The possible conditions for Ant-Miner$_{\mathrm{MA+G}}$ categorical attributes are $a_i = v$ and $a_i \neq v$. Ant-Miner$_{\mathrm{MA+G}}$ implements three possible approaches to handle categorical attributes:

1. The archive pheromone model is used to sample the value, using $=$ as the operator;
2. The archive pheromone model is used to sample the value and the operators out of two possible operators: $\{=, \neq\}$;
3. The graph pheromone model is used for categorical values — each categorical node consist of a triple (attribute, =, value) and there is a node for each value in the domain of the attribute.

6.1.2 Rule Evaluation Function

The rule quality function configurations used in the rule creation process typically represent a trade off between consistency and coverage — i.e., the quality functions promote rules that cover as few negative and as many positive instances as possible (Fürnkranz 2005; Janssen and Fürnkranz 2010a). A rule quality function is used in two different steps in this process: (i) evaluating rule quality in the pruning process, where it influences the choice of terms to be removed from the current rule; (ii) rule evaluation, where it influence the selection of the rules to be added to the list of rules.

Stecher, Janssen and Fürnkranz (2014) argued that these tasks should be treated separately and be performed using different rule quality functions. Where each of those tasks could have different trade off between consistency and coverage. In Ant-Miner$_{\mathrm{MA+G}}$ rule-pruning quality function are used to evaluate the effect of removing terms from the current rule, while rule selection functions are used in the archive sorting and selection of rules to be added to the model. We implemented 13 different rule quality functions, presented in Table 12. The same function can be used for both pruning and selection. For the parametric rule quality functions, we used the default parameter values proposed in (Janssen and Fürnkranz 2010a) — these are shown in the 'Parameter' column in Table 12. In this table we used a series of shorthand notations to condense the equations, as follows:

- **TP** (**T**rue **P**ositives): The number of instances covered by a rule that belong to the class predicted by the rule (the positive class);
- **FP** (**F**alse **P**ositives): The number of instances covered by a rule that do not belong to the class predicted by the rule;
- **TN** (**T**rue **N**egatives): The number of instances not covered by a rule that do not belong to the class predicted by the rule;

Table 13: Confusion Matrix

Rule	TP	FP	TN	FN	S
1	6	2	4	3	15
2	7	3	3	2	15

Table 14: The effect of using different rule quality functions to measure the quality of 2 rules discussed in the text. For each function, a higher value indicates better quality.

Functions	Rule 1	Rule 2
Precision (P)	**0.75**	0.70
Confidence Coverage (CC)	1.15	**1.17**
Cost measure (CM)	**1.50**	1.37
F-measure (FM)	0.68	**0.76**
Jaccard (J)	0.55	**0.58**
Klosgen (K)	**0.11**	0.08
Laplace (L)	**0.70**	0.67
MEstimate (ME)	0.49	**0.54**
Relative Cost Measure (RCM)	**0.01**	-0.06
Precision Inverted (PI)	**0.00**	-0.20
MEstimate Inverted (MEI)	**0.58**	**0.58**
Laplace Inverted (LI)	1.00	**1.33**
Sensitivity and Specificity(SS)	**0.44**	0.39

- **FN** (**F**alse **N**egatives): The number of instances not covered by a rule that belong to the class predicted by the rule;

- **S** (TP + FP + TN + FN): The total number of training instances.

6.1.3 The Effect of Different Quality Functions

We refer to the same example used in section 5.1.4, involving the following 2 rules

$$\begin{aligned} \text{RULE 1} &: \text{ IF } A_1 \ = \ 0 \text{ Then } 1 \\ \text{RULE 2} &: \text{ IF } A_2 \ < \ 35 \text{ Then } 1 \end{aligned} \tag{36}$$

The quality of these 2 rules differs according to different evaluation functions, as shown in Table 14.

Therefore, a change in the function selection will affect the algorithm performance. For example, using sensitivity and specificity, Rule 1 will have better quality than Rule 2, resulting in Rule 1 having a better rank in the archive; while if the F-measure is used, Rule 2 will have a better quality than Rule 1, and so their ranks in the archive will change. This will affect which rule has more influence in the archive (i.e., which rule ends up top of the archive) and it will change the rules sampled by ants. Also, if a different function is used by the rule-pruning procedure, the resulting rule can be different even if the same rule undergoes pruning. For example, removing a term will increase or decrease rule quality differently based on the function used.

6.1.4 Pheromone Model Configurations

There are three configurations regarding how the pheromone models are used in Ant-Miner$_{\mathrm{MA+G}}$, as discussed next.

Graph Pheromone Model

In the Ant-Miner$_{\mathrm{MA+G}}$ framework, there are two approaches for updating the graph pheromone model:

1. $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) (Stützle and Hoos 2000), where the update of the pheromone trail is done first by lowering the pheromone trail by a constant factor of evaporation and then allowing the ants to deposit pheromones on the terms they used in the rule based on the quality of the rule. The pheromone trails have min and max boundary to avoid early convergence.

2. Ant-Miner, where the pheromone associated with each term occurring in the rule created by an ant is increased in proportion to the quality of the rule in question; the pheromone associated with each term that does not occur

in the rule is decreased by normalizing all the pheromones values after the update.

Updating Pheromone Models

The level of interaction between the two pheromone models could range from no interaction at all (cases 1.3 and 2.1 and 3.2 in the following) to close interaction between them, as follows:

1. The top archive rule updates the graph pheromone model:

    1.1. In the first iteration after the archive is created;

    1.2. At the end of each iteration;

    1.3. Never updates the graph pheromone.

2. The graph pheromone model is updated with:

    2.1. The iteration-best rule;

    2.2. All rules that have been added to the archive.

3. The value used to update the graph pheromone model:

    3.1. The weight of the rule in the archive;

    3.2. A value proportional to the quality of the rule.

6.1.5 Restart Procedure

The restart procedure resets both pheromone models to the starting point without forgetting the best-so-far solution in the archive. It is used to avoid premature stagnation of the algorithm. The reset procedure is triggered (only once) by observing a number of consecutive iterations without improvement on the quality of the best-so-far rule. It works by randomly initializing the archive and resetting graph pheromone values to their initial value.

Table 15: Range of parameter values in Ant-Miner$_{\mathrm{MA+G}}$.

Parameters	range of Values
q (Local Search)	$[0.001, 1]$
ξ (Convergence)	$[0.001, 1]$
Archive size	$[5, 150]$
Maximum iterations	$[500, 2500]$
Uncovered instances	$[5, 25]$
Minimum covered by rule	$[5, 25]$
Ant colony size	$[5, 90]$
Stagnation limit	$[10, 100]$
Initial pheromone	$[1, 10]$
Evaporate factor	$[0.001, 1]$
Best pheromone	$[0.001, 1]$

Table 16: Summary of the training data sets used to automatically generate configurations of the Ant-Miner$_{\mathrm{MA+G}}$ algorithm.

Data set	Size	Ordinal	Categorical	Continuous
ionosphere	351	0	0	34
dermatology	366	33	0	1
cylinder-bands	540	2	14	19
annealing	898	0	29	9
credit-g	1000	11	2	7
MiceProtein	1080	0	3	77
HillValley	1212	0	0	100
eb	45781	0	1	2
adult	48842	0	8	6
SkinNonSkin	245057	0	0	3

6.2 Experimental Results for Ant-Miner$_{\mathrm{MA+G}}$

The computational experiments were computed using 35 publicly available data sets from the UCI Machine Learning Repository (Lichman 2013). The data sets were divided into two sets: a training set (shown in Table 16) and a testing set (shown in Table 17). When choosing the training set, we made sure we have diversity in the sizes of attributes and instances.

In the first part the experiments, our goal is automatically design a better variant for the Ant-Miner$_{\mathrm{MA}}$ algorithm using the proposed configurable framework

Table 17: Summary of the testing data sets used by both the Ant-Miner$_{\mathrm{MA}}$ and *c*Ant-Miner.

Data set	Size	Ordinal	Categorical	Continuous
breast-tissue	106	0	0	9
iris	150	0	0	4
wine	178	0	0	13
parkinsons	195	0	0	22
glass	214	0	0	9
breast-l	286	4	5	0
heart-h	294	3	3	7
heart-c	303	3	3	7
liver-disorders	345	0	0	6
breast-w	569	0	0	30
balance-scale	625	4	0	0
credit-a	690	4	4	6
pima	768	0	0	8
MolecularBiology	3189	0	60	0
ChoralsHarmony	5665	0	13	1
Mushroom	8124	0	22	0
PenDigits	10992	0	0	16
Magic	19020	0	0	10
CardClients	30000	7	2	14
Nomao	34465	0	29	89
bank-additional	41188	0	10	10
connect4	67557	0	42	0
diabetes	101766	2	34	11
ForestType	581012	0	44	10
PokerHand	1025010	5	0	5

Ant-Miner$_{\mathrm{MA+G}}$ discussed in Section 6.1 and the automatic configuration method I/F-Race. I/F-Race is a state-of-the-art automatic configuration method to deal with continuous, categorical, and discrete parameters (López-Ibáñez et al. 2016). I/F-Race generates new candidate configurations and performs races to discard the worst-performing ones. Within a single race of I/F-Race, candidate configurations are run on different instances of the algorithm at a time and a Friedman test followed by a post-test analysis is applied to discard configurations that show a sufficient statistical evidence that they perform worse than the remaining ones.

After only a small number of configurations remain in the race, the race stops. A new race starts with the best configurations previously found and with new candidate configurations generated from the best configurations using a simple probabilistic model. The automatic configuration process stops after reaching a given maximum budget, usually specified as a maximum number of runs or a time limit.

López-Ibáñez and Stützle (2012) showed that automating the selection of both the algorithmic design components and ACO parameter settings has the advantage of coping with the interaction between the design components and parameter settings. We therefore followed a similar approach, where I/F-Race optimises both the design components and ACO parameter settings—these are shown in Table 15. The configuration budget is set to 10000 runs. We perform five independent repetitions of the configuration process using the classification accuracy (the percentage of correctly classified instances) as the evaluation criterion. The best configuration found in each of the five runs were then used as seed candidates for a final I/F-Race configuration process. Therefore, we created six different configurations through six I/F-Race processes. The data sets used by I/F-Race are shown in Table 16.

The six best configurations found by the independent runs of I/F-Race are shown in Table 18. The configuration values are presented in Table 11 and the keys are used in Table 18 to describe the configurations found by I/F-Race.

Table 18: The best configurations of Ant-Miner$_{\text{MA+G}}$ found by I/F-Race, where the starred configurations values are found in Table 11.

Configurations	1	2	3	4	5	6
q (Local Search)	0.893	0.412	0.487	0.118	0.112	0.964
ξ (Convergence)	0.109	0.715	0.231	0.754	0.216	0.808
Archive Size	47	105	9	54	108	12
Max Iterations	1648	757	1442	719	704	1802
Uncovered Instance by model	16	10	14	19	7	9
Minimum covered by rule	11	10	15	10	17	8
Ant Colony Size	26	9	86	36	34	74
Stagnation limits	26	78	32	82	15	18
Prune Function	Jaccard	Confident Coverage	Cost Measure	Laplace Inverted	Sensitivity and Specificity	Jaccard
Selection Function	Sensitivity and Specificity	Sensitivity and Specificity	Sensitivity and Specificity	Sensitivity and Specificity	Sensitivity and Specificity	Sensitivity and Specificity
Ordinal Attributes	Ordinal with Range	Ordinal without Range	No Ordinal	Ordinal without Range	Ordinal with Range	Ordinal with Range

Table 18: The best configurations of Ant-Miner$_{\mathrm{MA+G}}$ found by I/F-Race, where the starred configurations values are found in Table 11.

Configurations	1	2	3	4	5	6
Operator Selection	Using Graph	Using Graph	Using Graph	Using Graph	Using Graph	Using Graph
Categorical Attributes	Archive with not equal condition	Using Graph	Archive with not equal condition	Archive with not equal condition	Using Graph	Using Graph
Archive top rule updates graph pheromone model	At the end of each iteration	At the end of each iteration	First iteration after the archive is created	Never updates	At the end of each iteration	At the end of each iteration
Updating graph pheromone model with	Best iteration rule	All rules added to the archive	All rules added to the archive	Best iteration rule	Best iteration rule	All rules added to the archive
Value used to update graph pheromone model	Weight of the rule in the archive	Weight of the rule in the archive	Weight of the rule in the archive	Weight of the rule in the archive	Weight of the rule in the archive	Weight of the rule in the archive
Restart Procedure	No Restart	No Restart	No Restart	No Restart	No Restart	No Restart
Pheromone limits	No limits	No limits	No limits	Limits	No limits	No limits
Pheromone Initial	NA	NA	NA	2.491	NA	NA

Table 18: The best configurations of Ant-Miner$_{\mathrm{MA+G}}$ found by I/F-Race, where the starred configurations values are found in Table 11.

Configurations	1	2	3	4	5	6
Pheromone Factor	NA	NA	NA	0.809	NA	NA
Pheromone Best	NA	NA	NA	0.917	NA	NA

The resulting configurations did show the impact of using a graph pheromone model, since the option of sampling operators using the graph was used in every winning configuration. This provides evidence for our first assumption that the graph pheromone model works well with nominal values. Categorical attributes showed interesting results as two options where used frequently: (1) using the graph to select the categorical value, which is the expected behaviour; and (2) sampling from archive was used when we added the not equal ($\neq$) operator. Also, using the value of the rule weight of the archive proved to produce better configurations. The configuration of the quality functions showed an interesting behaviour. While different functions were considering when pruning rules, the sensitivity and specificity dominated the configuration for evaluating rules for selection, providing a good indication of the benefit of using this function—it is the same function used in the original Ant-Miner.

Those six configurations are evaluated on the testing data sets (shown in Table 17) by running them 15 times in a tenfold cross-validation (a total of 150 individual runs) on every data set. The average results are shown in Table 19. We also measured the average runtime of the algorithms, shown in Table 20.

A particular interesting data set to look at is the *Nomao*, which is one of the largest data sets with 34465 instances and 118 attributes: This case was noted in Ant-Miner$_{\mathrm{MA}}$, where it was believed that the number of attributes did affect the Ant-Miner$_{\mathrm{MA}}$ performance. Notably, the proposed Ant-Miner$_{\mathrm{MA+G}}$ framework generated an improvement in the configuration Ant-Miner$_{\mathrm{MA+G}}$ (3), where it achieved an accuracy of 90.99% and a runtime of 846 seconds.

6.3 Comparison Against Classical Algorithms

In this section, we compared Ant-Miner$_{\mathrm{MA+G}}$ (3) — the automatic generated configuration with the best ranking — to the classical Ant-Miner$_{\mathrm{MA}}$ approach, *c*Ant-Miner and three well-known rule induction algorithms from the literature: Jrip, C5.0 rules and PART. The comparison was performed using the same 25 data sets presented in Table 17. Table 21 shows the average classification accuracy over 15 runs of tenfold cross-validation for the ACO-based algorithms, while the accuracies for the remaining algorithms (which are not stochastic) are averaged over one run of the tenfold cross-validation procedure. Table 22 shows the results considering the average number of rules in the models generated by the algorithms. There is no comparsion between runtime preformance, due to the fact that the classical algorithms are deterministic so the runtime will be fractions of the ant colony apporaches. For statistical testing of the differences in predictive accuracy and number of rules in the models, we used the Friedman test with Holm's post-hoc test (Demšar 2006). The Friedman test was used since we are comparing multiple algorithms (more than 2) over multiple data sets; the Holm's post-hoc test is used to adjust the p value given that we are performing multiple pair-wise comparison. The algorithm marked as 'Control' is the algorithm with best ranking — the remaining algorithms are compared against the control algorithm.

As seen in Table 21, C5.0 rules obtained the best average rank among all the algorithms and its performance is statistically significantly better compared to the remaining algorithms; all other algorithms achieved similar performance regarding predictive accuracy. Considering the ACO-based algorithms, the automatic generated configuration Ant-Miner$_{\text{MA+G}}$ (3) achieved the best ranking.

Interestingly, as shown in Table 22, Ant-Miner$_{\text{MA+G}}$(3) is the best rank algorithm in terms of average number of rules in this case, with a rank of 1.7, and its performance is statistically significantly better compared to all remaining algorithms except Jrip. The size of the model, measured by the number of rules in this case, can be used as a proxy measure of interpretability — models with a larger number of rules tend to be less interpretable to users that smaller ones. For example in the *diabetes* data set, as shown in Table 21, the accuracy of C5.0

rules is 57.90% while the quality of Ant-Miner$_{\mathrm{MA+G}}$(3) is 55.87%, which shows C5.0 rules with an improvement of around 2%; in terms of the average accuracy. However, Ant-Miner$_{\mathrm{MA+G}}$(3) has an average of 16 rules per model, while C5.0 rules has an average of 359 rules in the model. This supports example the idea that if a domain expert would like to understand the data, they are able to choose a model based on a trade-off between predictive accuracy and size.

6.4 Summary

In this chapter, we introduced the concept of combining the graph pheromone model and the archive pheromone model, based on the Ant-Miner$_{\mathrm{MA}}$ and *c*Ant-Miner algorithms. The use of the solution archive allows the algorithm to deal with continuous attributes without requiring a discretisation procedure, while using the graph pheromone model improves the predictive performance of algorithm for data sets containing a large number of attributes.

Instead of manually designing a new algorithm to combine both pheromone models, we proposed a fully configurable framework Ant-Miner$_{\mathrm{MA+G}}$ using an automatic design process. I/F-Race, which is a state-of-the-art automatic configuration tool, was used to generate six different configurations for the Ant-Miner$_{\mathrm{MA+G}}$ algorithm. Each one of those automatically designed configurations performed competitively well against several classical algorithms from the literature.

Our experimental results have shown that such an automatically configured design outperforms the *c*Ant-Miner algorithm to a significant level, and addressed the problem that Ant-Miner$_{\mathrm{MA}}$ faced when dealing with data sets containing a large number of attributes. The automatic framework also provides the flexibility to design an algorithm specific to a given data set.

Table 19: Average classification accuracy measured over 15 runs of tenfold cross-validation. The last row of the table shows the average rank of the algorithm. The best value for each given data set is shown in bold.

	Ant-Miner$_{\text{MA+G}}$'s configurations					
Data set	1	2	3	4	5	6
breast-tissue	64.81	63.35	66.07	65.83	65.84	**66.93**
iris	94.98	94.8	94.4	95.16	95.02	**95.33**
wine	92.61	93.05	92	92.69	92.72	**93.34**
parkinsons	83.77	86.06	**84.71**	82.4	84.39	84.09
glass	67.52	**68.72**	65.04	66.42	63.15	68.27
breast-l	71.95	**73.73**	72.24	72.94	70.23	72.54
heart-h	60.9	**61.81**	60.08	59.47	59.67	60.39
heart-c	55.59	55.99	55.97	55.64	**55.99**	55.36
liver-disorders	61.44	63.76	**64.41**	62.47	62.97	61.89
breast-w	**93.58**	93.22	93.29	91.67	93.75	93.48
balance-scale	73.56	74.38	73.3	73.34	**74.97**	74
credit-a	85.29	**85.59**	85.08	85.1	85.3	85.01
pima	73.81	73.51	**74.45**	72.8	73.63	72.75
MolecularBiology	84.09	69.92	83.46	79.46	83.66	**84.52**
ChoralsHarmony	61.42	60.44	62.18	61.31	60.16	**62.51**
Mushroom	97.46	97.05	**98.89**	96.82	93.98	98.52
PenDigits	82.15	81.07	86.18	79.21	85.76	**86.28**
Magic	80.65	81.35	**81.74**	80.1	81.31	80.61
CardClients	81.42	81.18	81.44	80.82	**81.55**	81.07
Nomao	88.74	89.76	**90.99**	86.84	88.68	90.77
bank-additional	90.6	90.32	**90.74**	90.41	90.57	90.62
connect4	67.9	67.42	**69.51**	67.72	67.49	68.37
diabetes	56.01	54.09	55.87	56.1	54.24	**55.89**
ForestType	68.92	67.25	**69.51**	67.02	69.09	69.15
PokerHand	50.23	50.25	51.39	51.56	**51.74**	50.24
Rank	3.76	3.57	**2.88**	4.38	3.3	3

Table 20: Average runtime measured over 15 runs of tenfold cross-validation. The last row of the table shows the average rank of the algorithm. The best value for each given data set is shown in bold.

	Ant-Miner$_{\text{MA+G}}$'s configurations					
Data set	1	2	3	4	5	6
breast-tissue	1.19	1.34	1.41	1.97	1.36	**0.86**
iris	**0.48**	0.94	0.86	1.04	0.93	0.75
wine	1.13	1.19	1.13	1.46	1.16	**0.85**
parkinsons	**1.18**	1.51	1.68	1.73	1.58	1.36
glass	**1.51**	2.11	2.15	3.09	1.93	2
breast-l	1.17	1.12	1.82	1.9	1.55	**1.11**
heart-h	**1.66**	2.37	3.02	3.7	2.33	2.47
heart-c	**2.05**	2.6	3.21	3.44	2.6	3.03
liver-disorders	**1.36**	1.87	2.21	2.59	1.81	1.52
breast-w	3.04	**2.64**	4.25	4.07	3.31	**2.64**
balance-scale	**1.12**	1.33	2.12	2.09	1.55	1.34
credit-a	2.44	**2.24**	4.43	4.06	3.16	2.56
pima	**2.46**	2.59	3.61	3.43	2.74	2.6
MolecularBiology	23.96	52.34	32.47	**23.29**	63.77	26.95
ChoralsHarmony	148.98	102.07	258.18	284.83	**100.44**	229.8
Mushroom	11.08	4.91	25.36	22.28	**4.6**	19.6
PenDigits	173.83	**107.12**	294.15	176	164.38	251.87
Magic	198.34	**145.74**	199.7	197.71	161.93	158.76
CardClients	458.64	**188.1**	830.95	409.17	445.07	546.12
Nomao	564.9	**310.67**	846.17	804.15	391.73	433.15
bank-additional	240.03	**121.2**	338.18	307.17	242.78	355.86
connect4	1263.02	**878.05**	4304.96	1405.23	1475.85	2950.35
diabetes	4043.66	4723.35	7208.3	**2470.67**	9669.81	6363.74
ForestType	49580.33	**29274.07**	72685.69	35223.81	64891.29	59500.34
PokerHand	6109.91	**3415.63**	11183.9	10218.68	6755	7442.81
Average Rank	**2.24**	2.36	5.24	4.64	3.36	3.16

Table 21: Average classification accuracy measured over 15 runs of tenfold cross-validation for the ACO-based algorithms, while the accuracies of the remaining algorithms are averaged over one run of tenfold cross-validation. The last row of the table shows the average rank of each algorithm. The best value for each given data set is shown in bold.

Data set	cAM	AM_{MA}	AM_{MA+G}	PART	Jrip	C5.0 rules (Control)
breast-tissue	64.24	63.15	**66.07**	64.36	59.18	64.17
iris	94.27	94.00	94.40	93.33	92.00	**94.65**
wine	**93.52**	91.39	92.00	91.54	92.68	92.71
parkinsons	85.22	85.50	84.71	**86.05**	84.53	80.87
glass	59.18	64.14	65.05	**72.81**	65.71	70.95
breast-l	**76.17**	71.85	72.25	68.94	69.26	73.06
heart-h	**64.81**	62.22	60.08	53.83	52.84	56.11
heart-c	**57.42**	56.46	55.97	53.83	52.84	56.11
liver-disorders	62.26	63.77	64.41	62.70	66.34	**68.06**
breast-w	**94.28**	93.41	93.29	94.19	93.67	93.65
balance-scale	68.34	76.70	73.30	**77.12**	73.92	72.15
credit-a	85.74	85.14	85.08	83.48	86.09	**86.39**
pima	67.45	**74.88**	74.45	71.73	73.55	74.70
MolecularBiology	71.14	69.78	83.46	92.66	93.29	**94.43**
ChoralsHarmony	60.59	63.16	62.18	**73.57**	69.83	73.53
Mushroom	98.23	99.30	98.89	**100.00**	99.98	**100.00**
PenDigits	56.90	78.87	86.18	96.89	96.35	**97.12**
Magic	70.41	82.21	81.74	85.57	84.55	**86.30**
CardClients	81.59	81.62	81.44	77.94	81.76	**81.98**
Nomao	90.66	89.38	90.99	**95.97**	95.39	95.24
bank-additional	89.87	90.11	90.74	89.71	91.26	**91.29**
connect4	67.83	68.84	69.51	78.93	75.43	**81.83**
diabetes	54.17	55.89	55.87	51.17	57.08	**57.90**
ForestType	63.07	69.06	69.85	93.48	89.55	**94.68**
PokerHand	50.20	51.70	51.38	74.44	59.13	**82.60**
Rank	4.28	3.96	3.92	3.34	3.40	2.10
p-value	**3.7E-5**	**4.4E-4**	**5.8E-4**	**0.01**	**0.02**	-
Holm's α	0.01	0.0125	0.0167	0.025	0.05	-

Table 22: Average number of rules measured over 15 runs of tenfold cross-validation for ACO-based algorithms, while the results for the remaining algorithms are average over one run of tenfold cross-validation. The last row of the table shows the average rank of the algorithm. The best value for each given data set is shown in bold.

Data set	cAM	AM_{MA}	AM_{MA+G}	PART	Jrip	C5.0 rules
			(Control)			
breast-tissue	6.90	6.45	**6.00**	10.90	**6.00**	9.50
iris	4.99	4.61	4.37	3.80	**3.30**	3.90
wine	5.10	4.83	**3.88**	4.30	3.90	5.00
parkinsons	5.84	5.69	4.15	6.40	**3.80**	6.10
glass	10.08	7.84	7.88	15.20	**7.20**	14.00
breast-l	5.88	7.00	5.74	18.40	**3.10**	4.70
heart-h	8.26	7.74	6.89	41.30	**3.30**	25.00
heart-c	9.72	8.68	7.95	41.30	**3.30**	25.00
liver-disorders	7.72	8.01	6.07	7.50	**4.30**	13.10
breast-w	8.78	6.95	**4.91**	6.70	5.40	8.30
balance-scale	12.29	8.98	**6.70**	29.90	12.10	17.90
credit-a	7.56	8.44	6.30	33.40	**4.10**	9.50
pima	10.04	10.29	7.01	7.50	**3.50**	11.80
MolecularBiology	11.70	14.48	**8.71**	100.10	16.60	51.30
ChoralsHarmony	46.25	**42.48**	42.54	334.80	129.00	197.20
Mushroom	5.83	12.12	**4.67**	12.10	8.50	18.00
PenDigits	38.30	31.76	**22.27**	80.30	75.70	122.80
Magic	36.33	25.61	**11.79**	46.20	22.70	80.50
CardClients	14.33	25.46	12.80	1334.20	**7.30**	53.90
Nomao	15.17	27.88	**12.53**	315.30	53.80	75.30
bank-additional	14.74	22.85	12.82	1020.30	**11.90**	111.10
connect4	17.77	27.31	**17.03**	3624.70	137.00	1305.30
diabetes	18.45	54.91	**16.99**	14147.60	15.20	359.50
ForestType	52.40	74.28	**28.08**	6941.30	1392.14	5517.20
PokerHand	28.63	30.47	**21.61**	66463.10	103.00	9617.60
Rank	3.52	3.44	1.7	5.2	2.14	5.00
p-value	**5.8E-4**	**0.001**	-	**3.7E-11**	0.4	**4.4E-10**
Holm's α	0.017	0.025	-	0.01	0.05	0.0125

Chapter 7

Data Stream Classification with Ant Colony Optimization

Krempl et al. (2014) highlighted the need to create simpler models, considering not only accuracy, but also considering the interpretability of the knowledge discovered by data stream algorithms. This was one of the recommendations based on the study of real-world applications and the shortcomings of the existing approaches. Notably, current rule induction algorithms in the field are implemented in an incremental approach (Gama and Kosina 2011; Stahl, Gaber and Salvador 2012), which leads to large and hard to interpret models. Ensemble approaches are shown to run successfully in data stream classification (Minku and Yao 2012; Baena-Garcıa et al. 2006; Street and Kim 2001; Almeida, Kosina and Gama 2013), but an ensemble architecture increases the complexity of the models produced. Our aim in this work is to produce interpretable simple models to be used in data stream classification.

The majority of Ant Colony Optimization (ACO) rule induction algorithms have proved to be successful in producing both accurate and interpretable classification models. Considering that the ACO-based classification algorithms are usually limited to cope only with categorical attributes, continuous attributes are usually discretised in a pre-processing stage. *c*Ant-Miner (Otero, Freitas and Johnson 2008, 2009) was the first Ant-Miner extension to cope with continuous attributes directly by employing a dynamic discretisation process, while Ant-Miner$_{\text{MA}}$ (Helal and Otero 2016, 2017) used an archive-based pheromone model to handle continuous attributes directly without discretisation process.

Both Ant-Miner$_{\text{MA}}$ and Ant-Miner$_{\text{MA+G}}$ approaches showed an improvement in runtime and competitive performance without using discretisation procedures. Attributes were treated directly as continuous, ordinal, or categorical, in the rule creation process. Those approaches are the first step in applying ACO rule induction algorithms to data stream, where we do not have access to the whole data sets and will not be able to discretise continuous attributes.

In this chapter, we propose to combine and integrate the archive and graph pheromone models to implement a Pittsburgh-based approach—a learning procedure which creates a rule list at each iteration (Section 2.3.2)—for data stream mining. The rationale is to use a construction graph to select attributes and determine their order, which is naturally a combinatorial problem; a solution archive to select values to create attribute tests, which is a mixed-variable problem; and the Pittsburgh-based learning approach to produce the best rule list, allowing the algorithm to cope with both term and rule interactions.

7.1 Stream Ant-Miner

The proposed algorithm, called stream Ant-Miner (sAnt-Miner), is an anytime prediction data stream algorithm, using a learning procedure where each ant creates a complete rule list in a single iteration. sAnt-Miner continuous learning replaces the classification model (rule list) rather than incrementing it, as illustrated in Figure 11. The high-level view of the algorithm comprises 2 layers. The

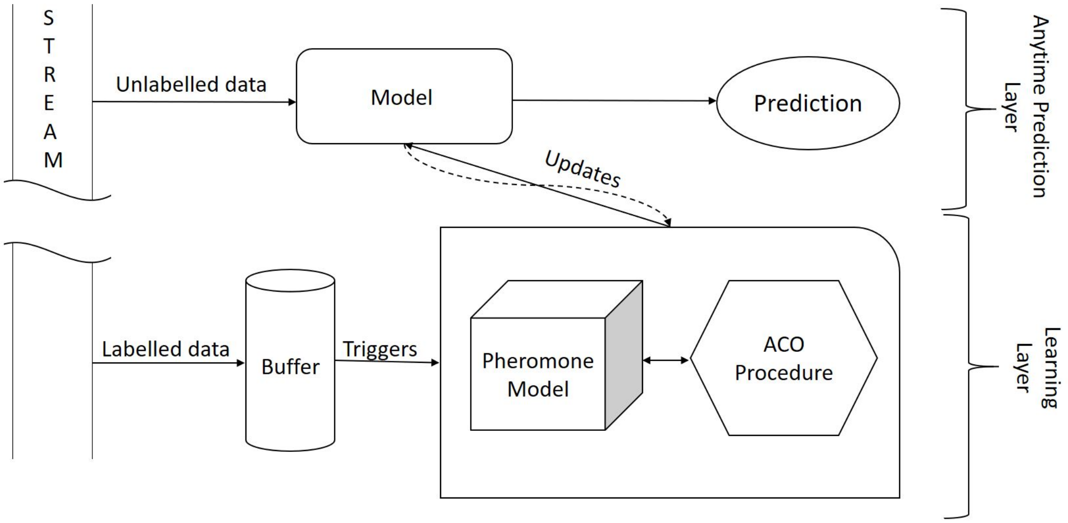


Figure 11: Overview of how sAnt-Miner works.

first layer is responsible for classifying the incoming data stream (anytime prediction layer); the second layer is responsible to update the model used in the first layer by learning from a subset of the labelled data (learning layer).

The learning layer consist of a buffer that stores the labelled data, which is used to trigger the learning and update the model. When the buffer is full or a high percentage of misclassification occurs, a learning procedure is triggered to update the model from the anytime prediction layer, allowing the algorithm to refine the model and/or adapt to a concept drift.

In real world scenarios, the unlabelled instances are first classified by the model, then either in an automated or manual fashion, these instances are labelled and feedback to the algorithm to train on them. An example is a credit score application, where an operation would be first classified by the algorithm, then receive a label (e.g., fraudulent or not) after a period of time and feedback to the model to learn from it.

Algorithm 8: High-level pseudocode of the learning procedure of sAnt-Miner

```
$R_{gb} \leftarrow$ Current model
$R_{gb}$.Quality $\leftarrow$ Evaluate(Buffer)
while $t <$ number of iterations do
    $R_{ib} \leftarrow R_{gb}$
    while $m <$ Colony size do
        $R_i \leftarrow$ Create Rule List()
        $R_i$.Quality $\leftarrow$ Evaluate($R_i$,Buffer)
        if $R_{ib}.Quality < R_i.Quality$ then
            $R_{ib} \leftarrow R_i$
        end
    end
    PheromoneModel.update($R_{ib}$)
    if $R_{gb}.Quality < R_{ib}.Quality$ then
        $R_{gb} \leftarrow R_{ib}$
    end
end
Buffer.Clear()
return $R_{gb}$
```

7.1.1 Overview of the approach

At the very first stage, the classifier used by sAnt-Miner is a majority classifier based on the labelled instances and it remains so until a learning procedure is triggered. Therefore, the initial prediction is a random choice between any of the available class values, then as each labelled instance arrives, a majority classifier is used. When a learning procedure is completed and a new classifier created, the current classifier is replaced if the new classifier has a higher predictive quality. Any subsequent prediction is then performed by the new classifier. Note that in all cases, predictions at this layer are very fast even when a classifier different than a majority one is used. Additionally, the classifier is not incrementally generated as it occurs in algorithms such as VFDT (Domingos and Hulten 2003), VFDR (Gama and Kosina 2011) and Ge-Rules (Le et al. 2014). Instead, the classifier is replaced between executions of the learning phase. Therefore, the classifier used in this layer does not suffer from the potential problem of growing indefinitely (Gama and Kosina 2011; Stahl, Gaber and Salvador 2012).

7.1.2 Learning Layer

The learning phase of sAnt-Miner uses the buffer of labelled instances and creates a new classifier using an ACO-based procedure. This procedure uses a novel hybrid graph and archive pheromone model, combining the strengths of both: the graph is used to select the attributes to be used while individual archives are used to create rule conditions. As shown in previous chapters (5,7), there are advantages in using a hybrid integrated pheromone model, as the archive provides the possibility of handling continuous attributes without the need to know the complete attribute values' distribution, while the graph helps to identify the best attributes to add to the antecedent of rules and their sequence. A high-level pseudocode of the learning algorithm is shown in Algorithm 8.

At the start of the learning procedure the current model is re-evaluated using the buffer instances (line 2). Then, each ant in the colony samples a new rule list from the current pheromone model (line 6). Once all rule lists are generated, pruned and evaluated using the buffer instances, the iteration-best rule list is used to deposit pheromone. If the quality of the iteration-best rule list is higher than the current model (R_{gb}), it replaces the current model; otherwise, the algorithm continues using the current model. Note that if the first iteration does not create a model better than the current model (R_{gb}), the current model is used to update the pheromones. The procedure to create a new rule list is repeated until the maximum number of iterations has been reached. At the end of the learning procedure, the buffer is cleared and the best model is returned. Note that the returned model could potentially be the same model in use, if any of the created models has not achieved better quality.

The learning procedure is designed to be fast, running a limited number of

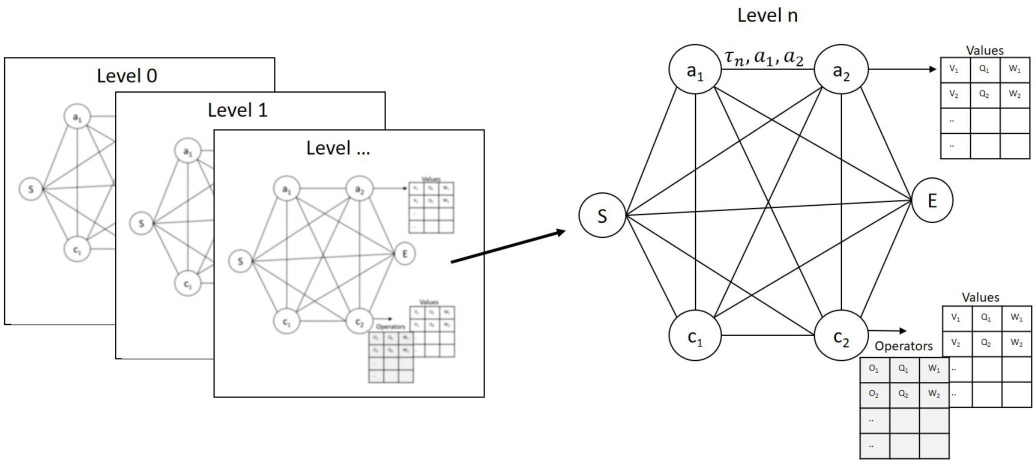


Figure 12: Simplified hybrid construction graph.

iterations each time it is triggered. If new instances requiring classification arrive before it is completed, the current model (anytime layer) performs the classification.

Pheromone Model

The hybrid construction graph consists of a fully connected graph, where each attribute in the data set is represented by a node. Each node holds an archive to sample values for the attribute. An additional archive is used by continuous attribute nodes to sample an operator (e.g.,"$\leq$" or "$>$"). Two additional nodes are added to represent the start (S) and end (E) of the antecedent of a rule. The pheromone model consists of several levels, which correspond to the indexes of the rule being created (e.g., 1 for the first rule, 2 for the second rule and so forth). This allows the algorithm to store pheromone and archive values for different rules in order to create a Pittsburgh-style rule list (Otero, Freitas and Johnson 2013). Figure 12 illustrates the hybrid construction graph and the underlying pheromone model. Pheromone is deposited on the edges of the construction graph, representing the attributes and their order to create rules; the operator and values are sampled from individual archives.

Each archive is sorted by the quality of the rule where the entry (value/operator) appears. The weight of an entry j is calculated by:

$$w_j = \frac{1}{qK\sqrt{2\pi}} e^{\frac{-(rank(j)-1)^2}{2q^2K^2}} \tag{37}$$

where q is a user-defined value used to control the extent of the top-ranked entry influence on the sampling of new values and K is the size of the archive. The weight of an entry is used during the sampling of new values as an indicator for the level of attractiveness of this value. The greater the weight of an entry, the higher the probability of sampling a new value around it.

Rule List Creation

The rule list creation is triggered when the buffer is full or the number of misclassified instances of one class is E percent of the buffer. When the rule list creation process starts, m ants create rule lists by traversing the construction graph. An ant uses the pheromone model to create multiple rules covering different training instances by specifying a tour identification, which corresponds to the index of the rule being created—0 for the first rule, 1 for the second and so forth. This way, each ant will use pheromone entries corresponding to the level of the rule (tour) being created during the rule construction process. The pheromone entries are stored in a pheromone matrix, where column and row indexes indicate the edge between two vertex. The probability of an ant to follow the edge leading to a vertex v_j when creating the rule t and located at vertex v_i is given by:

$$P^t_{v_j} = \frac{\tau^t_{v_i,v_j}}{\sum\limits_{k \in \lambda_{v_i}} \tau^t_{v_i,v_k}} \tag{38}$$

where $\tau^t_{v_i,v_j}$ is the amount of pheromone associated with the entry (t, v_i, v_j), and λ_{v_i} is the set of neighbouring vertices of vertex v_i. For each node selected by an ant, a term (operator, value) is sampled from the node's archive(s) based on the type of the attribute. The rule creation stops when the ant chooses to visit the end node of the graph. At this point, the rule is pruned to remove irrelevant attributes added due to the stochastic nature of the creation procedure, and the value predicted by the rule (consequent) is set to the majority class value observed among the covered instances. The instances covered by the rule are removed, and another rule is created until the number of instances remaining is equal to or lower than a user-defined threshold of uncovered instances.

This new hybrid model approach takes full advantage of the hybrid construction graph using the graph-based approach to select the attributes and their order, and solution archives to determine operator and values for terms. Additionally, the rule list creation process automatically learns the minimum number of attributes required by each rule, since the construction graph includes an end (E) node and pheromone is increased between the last attribute node in a rule and the end node. Therefore, every time pruning removes attributes from a rule, the pheromone update reflects this change. Over time, the creation process will not include irrelevant attributes, relying less on the pruning procedure.

Archive Sampling

At start of the learning procedure, random values are used since the solution archives are empty. Sampling only starts when an archive is full—the size of the archive is determined by a user-defined parameter. For nominal attributes, the relational operator is set to "equal" (=) and a value is sampled from the archive based on $\text{ACO}_{\mathbf{MV}}$ nominal value sampling as shown in Section 4.2.1.

For continuous attributes, each node has two archives: one for the relational operator and another for the value. The relational operator is sampled using a nominal sampling procedure, where the possible values are $\{\leq, >\}$. The value is sampled using a continuous sampling procedure as shown in Section 4.2.1.

Pruning

After the creation of a rule, the rule is pruned using a single pass procedure through the buffer to remove irrelevant rule terms. The pruning procedure calculates the coverage of each rule term based on the instances in the buffer. Then, each term is added to a new rule one at a time in the same order until the quality of the rule decreases or the addition of a term makes the rule cover less than a user-specified minimum number of instances. The remaining unused terms are then discarded.

For the purpose of pruning, the quality of a single rule is measured as sensitivity × specificity, the same measure employed in Ant-Miner, given by:

$$Q = \overbrace{\frac{TP}{TP + FN}}^{\text{Sensitivity}} \times \overbrace{\frac{TN}{FP + TN}}^{\text{Specificity}} \tag{39}$$

where TP is the number of covered instances that are correctly classified; FN is the number of instances that are not covered that have the same class as the rule; FP is the number of covered instances that are incorrectly classified; TN is the number of instances that are not covered and do not have the same class as the rule.

Pheromone update

The pheromone update is divided into two steps: the first step updates the edges of the construction graph, while the second updates each individual archive.

The graph update starts with pheromone evaporation, simulated by decreasing the amount of pheromone of each entry by a user-defined factor ρ. Then, the pheromone of the entries used in the iteration-best rule list is increased based on the quality of the rule list, which corresponds to its predictive accuracy measured on the buffer instances. The pheromone update is given by:

$$\tau_{t,v_i,v_j} = \begin{cases} \rho \times \tau_{t,v_i,v_j}, & \text{if}(t, v_i, v_j) \notin R_{ib} \\ \rho \times \tau_{t,v_i,v_j} + Q(R_{ib}), & \text{if}(t, v_i, v_j) \in R_{ib} \end{cases} \tag{40}$$

where ρ is the user-defined evaporation factor, τ_{t,v_i,v_j} is the amount of pheromone associated with the entry (t, v_i, v_j), t is the tour identification (i.e., the t rule in the rule list), v_i is the start vertex of the edge, v_j is the end vertex and $Q(R_{ib})$ is the quality of the iteration-best rule list. The values given by Equation (40) are limited to the interval $[\tau_{min}, \tau_{max}]$, following the same approach as the $\mathcal{MAX} - \mathcal{MIN}$ Ant System (MMAS) (Stützle and Hoos 2000). This procedure is the same as the update procedure in the Pittsburgh-based *c*Ant-Miner$_{\text{PB}}$ (Otero, Freitas and Johnson 2013).

After updating the graph edges, each individual archive of a node (attribute) used in a rule is updated. The update consists of adding a pair (value, quality) to the archive at the level t, where the quality corresponds to the rule list quality. Note that for continuous attributes, a pair (operator, quality) is also added to the operator archive. After all pairs are added, each archive is then sorted based on the pairs' quality. The weight associated with each pair is recalculated based on their (updated) rank. Finally, the low quality pairs are removed to resize the archive to K pairs.

7.1.3 Rule list creation walk-through

Let us consider a subset of the airline data set (Lichman 2013). This subset data set has 2 attributes: a categorical attribute (A$_1$); and a continuous attribute (A$_2$). The target class attribute has 2 values. Table 23 shows a subset of the data set.

After a number of iterations, the pheromone matrix has a depth of 2 (Table 24) as the algorithm needs to create only 2 rules to classify the data. In Table 24,

Table 23: A subset of the airline data set.

A_1	A_2	Class
CO	269	1
US	1558	1
AA	2400	1
AA	2466	1
AS	108	0
CO	1094	1
DL	1768	0
DL	2722	0
DL	2606	0
AA	2538	1
CO	223	1
DL	1646	1

S and E denote the Start and End node in the path traversed by the ant. Table 25 presents the values on the individual archives for the attributes A_1 and A_2. In order to create a new rule list, the algorithm performs the following steps.

Table 24: The pheromone matrix with 2 levels depth.

	1 depth				2 depth			
Source	Target Node				Target Node			
node	S	A_1	A_2	E	S	A_1	A_2	E
S	-	0.49	0.51	-	-	0.56	0.44	-
A_1	-	-	0.42	0.58	-	-	0.45	0.55
A_2	-	0.28	-	0.72	-	0.33	-	0.67
E	-	-	-	-	-	-	-	-

Step 1(a): Starting from the Node S at depth 1, we consider the probability that an ant will choose to visit either A_1 or A_2:

$$\begin{aligned} P^1_{v_{A_1}} &= 0.49 \\ P^1_{v_{A_2}} &= 0.51 \end{aligned} \tag{41}$$

An ant will choose A_1 with a probability of 0.49, and it will choose A_2 with a probability of 0.41. Lets consider for example, that a value of 0.51 is randomly

Table 25: Individual archive values with 2 level depth.

1 depth								
	A1		A2					
	Value			Operators			Value	
Weight	Value	Quality	Weight	Value	Quality	Weight	Value	Quality
0.379	DL	0.984	0.379	$>$	0.984	0.379	2373	0.984
0.241	AA	0.890	0.241	$>$	0.890	0.241	1608	0.890
0.062	CO	0.768	0.062	$>$	0.768	0.062	2224	0.768
0.007	US	0.443	0.007	$>$	0.443	0.007	696	0.443
2 depth								
	A1		A2					
	Value			Operators			Value	
Weight	Value	Quality	Weight	Value	Quality	Weight	Value	Quality
0.379	AA	0.984	0.379	$\leq$	0.984	0.379	3421	0.984
0.241	DL	0.890	0.241	$>$	0.890	0.241	1675	0.890
0.062	US	0.768	0.062	$>$	0.768	0.062	3347	0.768
0.007	CO	0.443	0.007	$\leq$	0.443	0.007	1253	0.443

generated, this mean that the ant will choose A$_2$ as the attribute for its first term.

Step 1(b): Since A$_2$ is a continuous attribute, there are two archives: one for the operator and one for the value. The first action is to select an operator from the archive. Using Equations (16) and (17), the probability of choosing each operator is given by[1]:

$$\begin{aligned}
\alpha^1_{A_{2>}} &= \frac{w_{ji}}{u_j} * \frac{q}{\kappa} = \frac{0.379}{4} * \frac{0.263}{1} = 0.025 \\
\alpha^1_{A_{2\leq}} &= \frac{q}{\kappa} = 0.26 \\
P^1_{A_{2>}} &= \frac{0.025}{0.285} = 0.087 \\
P^1_{A_{2\leq}} &= \frac{0.26}{0.285} = 0.923
\end{aligned} \tag{42}$$

If the ant then uses e.g. the random number 0.43, the operator used will be $\leq$.

Step 1(c): The second action is to sample the value for attribute A$_2$. The ant starts by randomly choosing a rule to build the solution around. The probability

[1] The values we use for the archive configurations are $K = 4$ and $q = 0.263$ and $\xi = 0.655$.

of choosing each entry of the archive is calculated using Equation (13), which produces the values shown in Equation (43):

$$
\begin{aligned}
P_1^1 &= \frac{0.379}{0.689} = 0.55 \\
P_2^1 &= \frac{0.241}{0.689} = 0.35 \\
P_3^1 &= \frac{0.379}{0.689} = 0.09 \\
P_4^1 &= \frac{0.379}{0.689} = 0.01
\end{aligned}
\tag{43}
$$

where P_1^1 is the probability to use the entry 1 at depth 1 for sampling the continuous attribute, P_2^1 is the probability to use the entry 2 at depth 1 for sampling the continuous attribute and so forth. Using the above probabilities, a randomly generated number will decide on which value to use as the mean value in the sampling of the new value. If the ants uses e.g. a random value of P = 0.685, the new value for A$_2$ will be sampled around the entry 2 (value 1608). The new value is sampled using Equations (14) and (15), as shown in Equation (44):

$$
\begin{aligned}
\delta^1 &= \xi \sum_{r=1, j \neq r}^{R} \frac{|S_{j,a} - S_{r,a}|}{R-1} \\
&= 0.655 * \frac{765 + 616 + 912}{3} \\
&= 0.655 * 254.7 = 166.88
\end{aligned}
\tag{44}
$$

$$V = Gaussian(1608, 166.88) = 1689$$

The final value selected is 1689, and then the ant adds the term $A_2 \leq 1689$ to its current rule.

Step 1(d): Moving from attribute A$_2$, we need to consider selecting another attribute or to end the construction of the rule:

$$\begin{aligned} P^1_{v_{A_1}} &= 0.424 \\ P^1_{v_{A_2}} &= - \\ P^1_{v_E} &= 0.58 \end{aligned} \tag{45}$$

An ant will choose A$_1$ with a probability of 0.42, and it will choose the end node with probability of 0.58. Lets consider that a random value of 0.77, for example, is selected. This mean that the ant will select the end node, finishing the construction of the rule.

Step 1(e): The final action is to calculate the rule consequent by finding the instances that are covered by the rule. Looking at Table 23, instances {1, 2, 5, 6, 10, 11} have a value lower than or equal to 1689. Therefore, the majority class value among those 6 instances is 1. The complete created rule is:

$$\text{IF } A2 \leq 1689 \text{ Then Class 1} \tag{46}$$

Step 2(a): Starting again from the node S but at depth 2, we consider the probability that an ant will choose to visit either A$_1$ or A$_2$:

$$\begin{aligned} P^2_{v_{A_1}} &= 0.56 \\ P^2_{v_{A_2}} &= 0.44 \end{aligned} \tag{47}$$

An ant will choose A$_1$ with a probability of 0.56, while it will choose A$_2$ with a probability of 0.44. Lets consider the value 0.23 is randomly generated, this means that the ant will choose A_1 as its first term.

Step 2(b): Since A$_1$ is a categorical attribute, there is only one archive for the value; the operator in this case is =(equal). Selecting the value for the attribute is calculated using the Equations (16) and (17), as shown by the following equations:

$$\begin{aligned}
\alpha^2_{A_{1AA}} &= \frac{w_{ji}}{u_j} * \frac{q}{\kappa} = \frac{0.379}{1} * \frac{0.263}{1} = 0.1 & P^2_{A_{1AA}} &= \frac{0.1}{0.4432} = 0.226 \\
\alpha^2_{A_{1DL}} &= \frac{w_{ji}}{u_j} * \frac{q}{\kappa} = \frac{0.241}{1} * \frac{0.263}{1} = 0.06 & P^2_{A_{1DL}} &= \frac{0.06}{0.4432} = 0.135 \\
\alpha^2_{A_{1US}} &= \frac{w_{ji}}{u_j} * \frac{q}{\kappa} = \frac{0.062}{1} * \frac{0.263}{1} = 0.02 & P^2_{A_{1US}} &= \frac{0.02}{0.4432} = 0.045 \\
\alpha^2_{A_{1CO}} &= \frac{w_{ji}}{u_j} * \frac{q}{\kappa} = \frac{0.007}{1} * \frac{0.263}{1} = 0.002 & P^2_{A_{1CO}} &= \frac{0.002}{0.4432} = 0.005 \\
\alpha^2_{A_{1AS}} &= \frac{q}{\kappa} = \frac{0.263}{1} = 0.263 & P^2_{A_{1AS}} &= \frac{0.263}{0.4432} = 0.593
\end{aligned} \tag{48}$$

If the ant then uses e.g. the random number 0.38, the value used will be “DL”. The second rule will decide on its consequent using the remaining instances.

Step 2(c): Moving from attribute A$_1$, we need to consider selecting another attribute or to end the construction of the rule:

$$\begin{aligned}
P^1_{v_{A_1}} &= - \\
P^1_{v_{A_2}} &= 45 \\
P^1_{v_E} &= 0.55
\end{aligned} \tag{49}$$

An ant will choose A$_2$ with a probability of 0.45, and it will choose the end node with probability of 0.55. Lets consider that the value 0.51 is randomly generated, this mean that the ant will select the end node, finishing the construction of the rule.

Step 2(d): The final action is to calculate the rule consequent by finding the instances that are covered by the rule. Looking at Table 23, instances {7,8,9} have a value higher than 1689 and have A_1 equal to “DL”. Therefore, the majority class value among those 3 instances is 0. The complete created rule is:

$$\text{IF } A_1 = DL \text{ Then Class 0} \tag{50}$$

Step 3(a): Since there are only three remaining instances {3, 4, 12}, the algorithm chooses to cover them using a default rule predicting the majority class value among them (Class 1 in this case). The final rule list and its quality on the training set are given below:

$$\begin{gathered} \text{IF } A2 \leq 1689 \text{ Then 1} \\ \text{IF } A1 = DL \text{ Then 0} \\ \text{IF no condition \ Then 1} \end{gathered} \tag{51}$$

$$\text{Quality} = \frac{\text{Correctly Classified}}{\text{Total}} = \frac{11}{12} = 0.91 \tag{52}$$

7.2 Summary

In this chapter, we introduce sAnt-Miner for rule induction in data stream classification. sAnt-Miner uses a novel hybrid pheromone model, combining both graph and archive pheromone models to creating classification model from data streams. sAnt-Miner’s hybrid pheromone model allowed the algorithm to benefit from the archive model in creating continuous attributes without the need for discretisation in a preprocessing step; and to benefit from the graph model in selecting the best attributes to use when creating rules.

Chapter 8

Results for sAnt-Miner

The proposed sAnt-Miner was implemented[1] using the Massive Online Analysis (MOA) (Bifet et al. 2010) framework for data stream mining. MOA includes a collection of machine learning algorithms — such as classification, regression, clustering, outlier detection, concept drift detection and recommender systems — as well as stream generators and evaluation measures. We used a total of 13 standard benchmark data sets in the evaluation: 6 real world data sets (Airplane, Connect4, Electricity, Diabetes, ForestType, and PokerHand) available in the UCI repository (Lichman 2013); the "Give Me Some Credit" (GMSC) data set from Kaggle repository[2]; the spam corpus data created as the result of a text mining process on an online news dissemination system (Katakis et al. 2009); and 5 synthetic data sets generated using the MOA random data generator. The details of the data sets used on the experiments are shown in Table 26.

In order to determine optimal values for sAnt-Miner's user-defined parameters, we used the I/F-Race procedure (López-Ibáñez et al. 2016). To maintain the comparison fair, 3 different synthetic data sets were used for tuning. Table 27 presents the details of the data sets used for tuning. The range of sAnt-Miner's

[1] https://github.com/AyahHelal/StreamAntMiner

[2] https://www.kaggle.com/c/GiveMeSomeCredit

Table 26: Summary of the data sets used in the experiments.

		Attributes		
Data set	# Instances	# Categorical	# Continuous	# Classes
		Real world		
Airplane	539383	4	3	2
Connect4	67557	42	0	3
Diabetes	101766	34	11	3
Electricity	45312	1	7	2
Forest Type	581012	44	10	7
GMSC	150000	0	10	2
Spam	9831	504	0	2
Poker Hand	829201	5	5	10
		Artificial		
Hyperplane Generator	1000000	0	10	2
LED Generator	1000000	0	24	10
Random RBF Generator	1000000	0	10	2
RT Generator	1000000	5	5	3
SEA Generator	1000000	0	3	2

Table 27: The data sets used in the I/F-Race procedure, all the data sets were generated using the MOA data generator.

		Attributes		
Data set	# Instances	# Categorical	# Continuous	# Classes
Wave Form Generator	100000	0	40	3
Sine Generator	100000	0	4	2
Agrawal Generator	100000	3	6	2

Table 28: sAnt-Miner's parameters range used by I/F-Race in the tuning phase.

Parameters	Range	Final value
Buffer Size	[250, 1500]	1459
Colony Size	[5, 20]	6
ξ	[0.00, 1.00]	0.367
q	[0.00, 1.00]	0.119
Archive Size	[5, 50]	17
Max Iteration	[5, 70]	54
Buffer Trigger	[0.00, 1.00]	0.748
Minimum Cases	[5, 30]	30
Uncovered Percentage	[0.00, 0.2]	0.176

parameter values used as input for I/F-Race and the final selected values of sAnt-Miner's parameters are presented in Table 28.

8.1 Experimental Setup

We compared sAnt-Miner against VFDR and Ge-Rules, both algorithms are well-known rule induction data stream classification algorithms. VFDR is available on MOA framework; Ge-Rules was downloaded from the Git-Hub repository[3] and upgraded to work on (2016.10) version of MOA. We used the prequential 10-fold bootstrap validation with ADaptive WINdowing (ADWIN) (Bifet and Gavalda 2007) evaluation window; ADWIN has theoretical guarantees that the chosen size is optimal without the need to decide beforehand on the size of the sliding window (Bifet et al. 2015). For both VFDR and Ge-Rules, the values of a single ADWIN evaluation are averaged over the 10 folds; for sAnt-Miner the values of 15 ADWIN evaluations are averaged over the 10-fold bootstrap validation with ADWIN evaluation window — a total 150 evaluations — to count for the stochastic nature of the algorithm.

1. **Prequential accuracy:** instances are first classified by the algorithm (test) before they are available for the learning procedure (training);

2. **Runtime:** runtime of a single prequential 10-fold bootstrap validation with ADWIN evaluation window;

3. **Rules count:** the number of rules in the generated model;

4. **Kappa Statistics:** it takes into account the class unbalance of the data stream asdiscussed in Section 3.4.

In order to measure the statistical significance of the differences in algorithms' predictive performance, we used the non-parametric Friedman test with

[3] https://github.com/thienle2401/G-eRules

the Holm's post-hoc test (Demšar 2006). Tables 29-34 show the average values of the ADWIN prequential evaluation for each of the different measures. In each of these tables, the best value for each data set is shown in bold; the last three rows presents the results of the Friedman statistical test.

Table 29: Average prequential accuracy computed over 15 runs of 10-folds bootstrap validation with adwin evaluation window.

Data set	Ge-Rules	VFDR	sAnt-Miner
Airplane	57.92	56.06	**59.21**
Connect4	**69.27**	65.67	66.49
Diabetes	50.81	53.83	**56.06**
Electricity	49.49	70.40	**89.18**
ForestType	63.48	64.35	**70.30**
GMSC	93.32	93.28	**93.35**
Spam	92.76	82.40	**93.00**
PokerHand	53.07	**59.58**	56.05
HyperplaneGenerator	50.00	**71.00**	69.60
LEDGenerator	**51.28**	47.72	47.86
RandomRBFGenerator	52.14	**78.04**	68.45
RTGenerator	**64.58**	60.76	57.62
SEAGenerator	66.68	80.29	**84.02**
Average rank	2.308	2.154	1.538
p-value	0.04986	0.11666	-
$Holm$'s α	0.025	0.05	-

Table 29 presents the results regarding the prequential accuracy. sAnt-Miner is the most accurate algorithm, achieving an average rank of 1.54; VFDR ranked second with 2.15, followed by Ge-Rules with 2.31. The p-value obtained by sAnt-Miner compared to Ge-Rules 0.05 and VFDR 0.12 are not statistical significant according to the non-parametric Friedman test with Holm's post-hoc test at the 5% level. Overall, all algorithms achieved a similar prequential accuracy. In both the Electricity and Forest Type data sets, sAnt-Miner achieved larger improvement over VFDR and Ge-Rules, while in the Random RBF Generator data set VFDR achieved a large improvement over sAnt-Miner and Ge-Rules.

Table 30: Average runtime in seconds computed over 15 runs of 10-folds bootstrap validation with adwin evaluation window.

Data set	Ge-Rules	VFDR	sAnt-Miner
Airplane	2407.67	**172.53**	187.72
Connect4	101.52	**13.30**	65.40
Diabetes	1248.68	**23.53**	82.37
Electricity	**15.13**	45.72	20.79
ForestType	1099.44	566.90	**538.75**
GMSC	**30.70**	327.65	58.28
Spam	88.88	**10.79**	138.78
PokerHand	1570.68	1259.51	**505.93**
HyperplaneGenerator	**180.85**	2809.90	975.91
LEDGenerator	3009.83	3663.33	**2027.74**
RandomRBFGenerator	**191.29**	6660.39	440.59
RTGenerator	**874.90**	5874.18	1240.41
SEAGenerator	**137.65**	2383.27	678.19
Average rank	1.923	2.231	1.846
p-value	0.8445	0.3267	-
$Holm$'s α	0.05	0.025	-

Table 30 presents the results regarding the runtime. sAnt-Miner is the fastest algorithm, achieving an average rank of 1.85; Ge-Rules ranked second (rank of 1.92) followed by VFDR (rank of 2.23). Again, no statistically significant differences were observed with p-values 0.85 and 0.33 for Ge-Rules and VFDR respectively. The results obtained by sAnt-Miner show that, although the proposed approach has an iterative nature, it is still fast to run. Further improvement to the runtime could be achieved by parallelising the ACO procedure.

The results of the Kappa measures depends on the nature of the data sets used. The negative indicate the the compared algorithm perform worse than a naive classifier. Table 31 presents the results regarding the Kappa measure. sAnt-Miner is the most accurate algorithm, achieving an average rank of 1.54; VFDR ranked second 2.15 followed by Ge-Rules 2.31. It is interesting to note that Ge-Rules has negative kappa values in the PokerHand and Hyperplane generator data

Table 31: Average Kappa computed over 15 runs of 10-folds bootstrap validation with adwin evaluation window.

Data set	Ge-Rules	VFDR	sAnt-Miner
Airplane	**15.27**	2.38	13.43
Connect4	**22.18**	0.00	8.92
Diabetes	2.94	0.04	**9.63**
Electricity	2.31	41.77	**78.39**
ForestType	10.27	13.05	**29.41**
GMSC	0.00	1.35	**10.31**
Spam	67.55	18.40	**73.09**
PokerHand	-0.02	7.05	**7.44**
HyperplaneGenerator	-0.08	**42.01**	39.20
LEDGenerator	**45.89**	41.95	42.09
RandomRBFGenerator	4.05	**56.08**	36.91
RTGenerator	**40.42**	32.93	28.54
SEAGenerator	9.42	54.79	**64.06**
Average rank	2.231	2.231	1.538
p-value	0.077	0.077	-
$Holm$'s α	0.025	0.05	-

sets, as negative values indicate that the algorithm is performing worse than a chance classifier. No statistical significant differences were observed with p-values of 0.077 for both Ge-Rules and VFDR.

Table 32 presents the results regarding the Kappa M measure. sAnt-Miner is the most accurate algorithm, achieving an average rank of 1.69; VFDR ranked second with 2.08 followed by Ge-Rules with 2.23. Note that there are eight imbalanced data sets, where a single class has a high proportion of instances: Airplane, Connect4, Diabetes, Electricity, Forest Type, GMSC, Poker Hand and Hyperplane Generator. It is expected that the majority class classifier perform relatively well in those data sets. Out of the eight data sets, sAnt-Miner performs better than both other algorithms in four (Airplane, Electricity, Diabetes and GMSC) while VFDR performs better in three (ForestType, Poker Hand and Hyperplane Generator) and Ge-Rules perform better in one (Connect4). Note that a negative Kappa

Table 32: Average Kappa M computed over 15 runs of 10-folds bootstrap validation with adwin evaluation window.

Data set	Ge-Rules	VFDR	sAnt-Miner
Airplane	2.23	-1.88	**5.16**
Connect4	**10.06**	-0.48	1.92
Diabetes	-6.70	-0.15	**4.68**
Electricity	-7.55	36.98	**76.95**
ForestType	-274.71	**-227.66**	-298.48
GMSC	-0.04	-0.73	**0.32**
Spam	65.39	15.93	**66.57**
PokerHand	-36.75	**-17.76**	-28.28
HyperplaneGenerator	-0.23	**41.87**	39.06
LEDGenerator	**45.80**	41.83	41.99
RandomRBFGenerator	3.82	**55.87**	36.61
RTGenerator	**32.27**	24.97	18.97
SEAGenerator	6.78	44.86	**55.30**
Average rank	2.231	2.076	1.692
p-value	0.1698	0.3268	-
$Holm$'s α	0.025	0.05	-

M value indicates that an algorithm is performing worse than a majority classifier; sAnt-Miner obtained negative Kappa M values in only two data sets (Forest Type and Poker Hand), while VFDR and Ge-Rules obtained negative Kappa M values 6 data sets, including the (Forest Type and Poker Hand) data sets. No statistically significant differences were observed with p-values of 0.17 and 0.33 for Ge-Rules and VFDR respectively.

Table 33 presents the results regarding the Kappa Temporal measure. Again, sAnt-Miner is the most accurate algorithm, achieving an average rank of 1.54; VFDR ranked second 2.15 followed by Ge-Rules 2.31. There are four data sets with known temporal nature: Electricity, Forest Type, Spam, and Poker Hand. sAnt-Miner shows the best performance compared to a persistent classifier in three out of the four data sets (Electricity, Forest Type, Spam); VFDR performs best in the remaining one (Poker Hand). Additionally, sAnt-Miner performed better than

Table 33: Average Kappa temporal computed over 15 runs of 10-folds bootstrap validation with adwin evaluation window.

Data set	Ge-Rules	VFDR	sAnt-Miner
Airplane	4.63	0.46	**7.64**
Connect4	**37.77**	30.48	32.13
Diabetes	14.67	19.91	**23.77**
Electricity	-232.25	-94.69	**28.79**
ForestType	-834.40	-800.35	**-645.48**
GMSC	48.72	48.38	**48.92**
Spam	-115.45	-423.29	**-108.11**
PokerHand	-77.20	**-56.70**	-67.45
HyperplaneGenerator	0.15	**42.09**	39.29
LEDGenerator	**45.84**	41.88	42.03
RandomRBFGenerator	4.34	**56.11**	36.95
RTGenerator	**43.93**	37.88	32.91
SEAGenerator	27.44	57.08	**65.21**
Average rank	2.307	2.153	1.538
p-value	0.04986	0.11666	-
Holm's α	0.025	0.05	-

the persistent classifier (positive value) in the Electricity data set, while both Ge-Rules and VFDR performed worse than the persistent classifier (negative values) in that data set. No statistically significant differences were observed with p-values of 0.05 and 0.11 for Ge-Rules and VFDR respectively.

In order to evaluate the simplicity of the discovered model, we focus on the results regarding the rules count, presented in Table 34. In this case the lower the number of rules, the simpler the model. sAnt-Miner discovers the smallest models; achieving an average rank of 1.15; VFDR ranked second (2.15) followed by Ge-Rules (2.69). The results obtained by sAnt-Miner are statistically significantly better than (p-value 0.9E-5) Ge-Rules and (p-value 0.011) VFDR according to the non-parametric Friedman test with Holm's post-hoc test with at the 5% significant level.

Table 34: Average Rule count computed over 15 runs of 10-folds bootstrap validation with adwin evaluation window.

Data set	Ge-Rules	VFDR	sAnt-Miner
Airplane	2760.63	110.40	**2.469**
Connect4	915.10	19.70	**2.939**
Diabetes	1621.75	13.05	**2.188**
Electricity	368.30	26.90	**3.127**
ForestType	399.08	34.90	**3.273**
GMSC	315.00	44.00	**1.781**
Spam	181.60	**2.50**	3.318
PokerHand	1187.38	101.81	**3.282**
HyperplaneGenerator	**5.74**	162.64	6.564
LEDGenerator	121.43	22.13	**8.669**
RandomRBFGenerator	17.62	203.79	**4.612**
RTGenerator	400.45	110.97	**7.645**
SEAGenerator	50.22	213.22	**5.112**
Average rank	2.6923	2.1538	1.1538
p-value	**0.876E-5**	**0.0108**	-
$Holm$'s α	0.025	0.05	-

Overall, the results obtained by the proposed sAnt-Miner are positive. It discovered smaller models while maintaining similar predictive accuracy when compared to Ge-Rules and VFDR, the latter considered the state-of-the-art rule induction data stream classification algorithm—the importance of discovering smaller models has been highlighted by Krempl et al. (2014). sAnt-Miner is competitive regarding its runtime, despite being an iterative algorithm. We attribute this to the proposed hybrid construction graph, where a solution archive handles continuous values combined with a construction graph to select attributes to compose rules. Previous work has indicated that using a solution archive only in data sets with more than 50 attributes did not produce accurate models (Helal and Otero 2016, 2017). Our results indicate that the hybrid model allows the algorithm to deal with a larger number of attributes effectively, while maintaining the advantage of the solution archive in dealing with continuous attributes without requiring a discretisation procedure; in both Forest Type and Spam (data sets with more than 50 attributes), sAnt-Miner achieved the highest prequential accuracy.

8.2 Summary

In this chapter, we compared sAnt-Miner for rule induction in data stream classification against VFDR and Ge-Rules using standard benchmark data sets. Our results showed that sAnt-Miner models had competitive accuracy and reactivity compared to VFDR and Ge-Rules models. Moreover, sAnt-Miner models were smaller, and more concise models help understanding the rules currently created to classify the data stream.

Chapter 9

Mixed-Attribute Ant-Miner for Regression Rule Discovery

In this chapter, we propose the use of an archive-based pheromone model to improve how continuous values are handled by an ACO algorithm in regression problems. By incorporating a similar $\text{ACO}_{\mathbf{MV}}$ strategy to the one used in Ant-Miner_{MA} (Chapter 5), different attributes types (categorical and continuous) can be handled directly without requiring a discretisation procedure.

Brookhouse and Otero (2015) have successfully used an ACO-based algorithm, called Ant-Miner-Reg, to create regression rules. Ant-Miner-Reg uses a sequential covering approach to create a rule list using an ACO rule creation procedure with a graph-based pheromone model. In order to handle continuous attributes, Ant-Miner-Reg uses a M5 (Quinlan et al. 1992) inspired dynamic discretisation procedure during the rule creation process rather than requiring the discretisation of continuous values as a pre-processing step. Computational experiments showed that Ant-Miner-Reg significantly outperformed SeCoReg (Janssen and Fürnkranz 2010b), a greedy sequential covering algorithm, without increasing the average number of terms required to classify an instance. This indicates that ACO algorithms have the potential for being successful in creating regression rules.

While we know that Ant-Miner$_{\text{MA}}$ improved the runtime compared to the graph-based algorithm in classification problems, improvements on the quality of the learned rule lists were not observed (Helal and Otero 2016). In this chapter, we investigate the effects of using an archive-based pheromone model to create regression rules, evaluating both the predictive performance and the runtime of the algorithm.

9.1 Archive-based Ant-Miner-Reg

The proposed Archive-based Ant-Miner-Reg (Ant-Miner-Reg$_{\text{MA}}$) algorithm uses the ACO$_{\mathbf{MV}}$ pheromone model and search procedure to sample terms to create regression rules. The high-level pseudocode of Ant-Miner-Reg$_{\text{MA}}$ is shown in Algorithm 9. Ant-Miner-Reg$_{\text{MA}}$ starts with an empty list of rules (line 1). At each iteration (lines 3-23), a single rule is created. The rule creation process starts by initialising the archive with k randomly generated rules (line 3). At each iteration, m new rules (lines 7-13) are generated, where m is the number of ants in the colony. Rules are added to the archive and the $k + m$ rules are sorted (line 14). The worst m rules are removed from the archive, limiting the archive to k best rules found so far. The procedure to create new rules is repeated until the maximum number of iterations has been reached or stagnation. Stagnation is the failure of the algorithm to find better rules for a predefined number of iterations. In the first occurrence of stagnation, a restart procedure is applied; if the algorithm reaches stagnation for a second time, the rule creation procedure stops.

9.1.1 Rule Structure

A rule R consists of an n-dimensional term vector, where n is the number of attributes in the data set. Each term t_i (i in $[1..n]$) in R contains a flag to indicate if this term is enable or not, an operator and value. For continuous attribute

Algorithm 9: High-level pseudocode of Ant-Miner-Reg$_{\text{MA}}$.

```
Data: TrainingData
Result: RuleList
RuleList ← {}
while |TrainingData| > MaxUncovered do
    A ← Generate Random Rules
    Restarted ← 0
    while t < MaxIterations and Restarted ≤ 1 do
        A_t ← {}
        while i < number of ants do
            R_i ← Create New Rule
            R_i ← Prune(R_i)
            R_i ← Set Consequent(R_i)
            A_t ← R_i
            i ← i + 1
        end
        A ← UpdateArchive(A_t)
        t ← t + 1
        if stagnation() then
            Restart(A)
            Restarted ← Restarted + 1
        end
    end
    R_best ← BestRule(A)
    RuleList ← RuleList + R_best
    TrainingData ← TrainingData − Covered(R_best)
end
return RuleList
```

terms, the operator can be either $\leq$ or $>$, representing conditions where the term's attribute value is less than or equal to a specific real value ($\leq x$) or greater than a specific real value ($> x$). Categorical attribute terms use a single operator $=$, representing conditions where the term's attribute value is equal to a specific value in the domain of the attribute ($= y$). The consequent (prediction) of a rule is a real value, calculated as the mean value of the target (class) attribute in the instances covered by the rule on the training data.

9.1.2 Rule Quality

The quality of a regression rule is based on two factors, the first is the quality of the prediction measured using a Relative Root Mean Squared Error (RRMSE). The RRMSE of a rule is defined as:

$$L_{RRMSE} = \frac{L_{RMSE}}{\sqrt{\frac{1}{m}L_{default}}} \tag{53}$$

where 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; m is the number of instances covered by the rule. L_{RMSE} and $L_{Default}$ are defined as:

$$\begin{aligned} L_{RMSE} &= \sqrt{\frac{1}{m} \cdot \sum_{i=1}^{m}(y_i - \bar{y_i})^2} \\ L_{default} &= \sum_{i=1}^{m}(y_i - y')^2 \end{aligned} \tag{54}$$

Where y is the value of the target (class) attribute in the current instance, $\bar{y}$ is the predicted value of the target (class) attribute in the current instance and y' is the mean of the target (class) attribute over all instances in the data set. The RRMSE approximately normalises the RMSE of a rule between 0 and 1, where a value less than 1 corresponds to a rule making a prediction better than the mean of the target (class) value of uncovered instances and a value greater than 1 is worse than the mean.

The second factor is a measure of how generalised the rule is, i.e., number of instances covered by the rule. Like RRMSE, the coverage of a rule is normalised so that 0 represents a rule covering no instances and 1 is a rule that covers all of the instances in the data set. The relative coverage of a rule R is defined as

$$relCov = \frac{1}{M} \cdot coverage(R) \tag{55}$$

Where M is the total number of instances in the data set and coverage(R) is the number of covered instances by rule R. Both the RRMSE and relative coverage are combined into a single metric Q, which is used as a rule's quality, defined as:

$$Q = \alpha \cdot (1 - L_{RRMSE}) + (1 - \alpha) \cdot relCov \tag{56}$$

where α sets the weighting between RRSME and relative coverage. Varying α between 0 and 1 bias the rule quality towards either RRMSE ($\alpha = 1$) or relative coverage ($\alpha = 0$).

9.1.3 Archive Structure and Initialisation

The archive consist of k rules sorted by their 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 related to its $Q(R_j)$, where ω_j is calculated using a Gaussian function given by:

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} e^{\frac{-(rank(j)-1)^2}{2q^2k^2}} \tag{57}$$

where q is used to control the influence of the top-ranked rules on the construction of a new rule. When a new rule is created, it probabilistically samples values around the rules with higher weights.

The archive is initialised with k random rules. Initialisation begins by randomly enabling each term in the vector of allowed terms. These enabled terms are then initialised according to 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 generated among the values observed in the training data for that attribute. For categorical terms, only the $=$ is added and the value set randomly set to one of the values in the domain of the attribute.

Rules are then pruned to disable irrelevant terms that might be enabled by the stochastic nature of the initialisation. 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 doesn't, a new rule is generated instead. Finally, rules are sorted according to their quality.

9.1.4 Rule Creation

The rule creation process uses the solution archive to sample values. The sampling procedures are the same as the ones used in Ant-Miner$_{\mathrm{MA}}$ (Section 4.2.1). Rule creation starts by choosing probabilistically whether to include each term or not. The decision is handled using a categorical sampling to choose a value from {`TRUE`, `FALSE`}. If the term is enabled (`TRUE` value), we set the operator according to the attribute type. If the attribute is categorical, it is set to =. If it is continuous, the decision is handled using a categorical sampling to choose an operator from the set $\{\leq, >\}$, with the only difference being that only the subset of rules that have this term enabled are considered in Equation (17).

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 as the new term. If the attribute is categorical, we use the categorical sampling procedure only considering the subset of rules that have this term enabled.

After a term is created and added to the partial rule, we apply the rule to the training data. If the number of instances covered by the rule after the addition of the new term is less than the user-defined minimum covered instances, the term is disabled. This process is repeated until all terms are considered.

Finally, a local search procedure is applied. The local search procedure is inspired by the *threshold-aware* pruner (Otero, Freitas and Johnson 2009). Firstly, the quality of the rule is calculated according to Equation (25). Then, the last term is disabled and the quality re-calculated. If the quality of the (pruned)

Table 35: Parameter values used in experiments. Ant-Miner-Reg$_{\text{MA}}$ uses the first three parameters in this table, while the remaining ones are used by both Ant-Miner-Reg$_{\text{MA}}$ and Ant-Miner-Reg.

Parameters	Value
q	0.025495
ξ	0.6795
R	90
Minimum Covered	10
Max Uncovered	10
Max Iterations	1500
Number of Ants	60
Stagnation Test	10
α	0.59

rule decreases, the term is re-enabled and the procedure stops; otherwise, the procedure is repeated until a decrease in quality is observed.

9.2 Comparison with Ant-Miner-Reg

We compared our proposed algorithm Ant-Miner-Reg$_{\text{MA}}$ against Ant-Miner-Reg. The experiments are conducted using nineteen regression data sets publicly available from the UCI Machine Learning Repository (Lichman 2013)—details are shown in Table 36. Ant-Miner-Reg$_{\text{MA}}$ uses the first three parameters in Table 35 for the archive setting, while the remaining parameters are used by both algorithms. We ran both algorithms for five times with tenfold cross-validation each time for a total of fifty runs for each data set, and reported the average performance of the models produced by each algorithm—shown in Table 37 in terms of relative root mean square error (RRMSE). The last row in Table 37 shows the average rank of each algorithm. For statistical significance testing of the difference in RRMSE and runtime, we used the Wilcoxon signed-rank test (Wilcoxon 1992) at the 5% significant level. The result of the statistical testing is shown in Table 39.

Table 36: Details 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

In terms of RRMSE (Table 37), Ant-Miner-Reg$_{\mathrm{MA}}$ did not significantly improve the RRMSE compared to Ant-Miner-Reg (p-value = 0.97) — although Ant-Miner-Reg$_{\mathrm{MA}}$ does have a better average rank (1.42) than Ant-Miner-Reg (1.52).

In terms of computational time (Table 38), Ant-Miner-Reg$_{\mathrm{MA}}$ shows an improvement in runtime compared to Ant-Miner-Reg, outperforming Ant-Miner-Reg in eighteen of the nineteen data sets. Most notably, Ant-Miner-Reg$_{\mathrm{MA}}$ achieved more than one order of magnitude improvement in both Pm 25 and Energy Data data sets: Ant-Miner-Reg$_{\mathrm{MA}}$'s runtime was 582.38 and 615.24 seconds; while Ant-Miner-Reg's runtime was 30669.21 and 55113.399 seconds, respectively. Based on our results, it is clear that the introduction of an archive-based pheromone model in Ant-Miner-Reg$_{\mathrm{MA}}$ resulted in an improvement in the model creation runtime.

Table 37: Average RRMSE (average ± standard error) measured by five runs of tenfold cross-validation. The value of the most accurate algorithm for a given data set is shown in bold.

Data set	Ant-Miner-Reg	Ant-Miner-Reg$_{\text{MA}}$
WPBC_r	1.13±0.052	**1.04±0.016**
CPU	0.76±0.048	**0.50±0.012**
Yacht	**0.15±0.030**	0.21±0.006
MPG	0.58±0.038	**0.54±0.009**
Housing	0.65±0.022	**0.60±0.014**
Forest Fire	**1.28±0.087**	1.53±0.160
Istanbul	0.82±0.016	**0.79±0.012**
Efficiency	0.34±0.009	**0.23±0.006**
Stock	0.36±0.011	**0.33±0.004**
Concrete	**0.54±0.007**	0.72±0.015
Flare	1.00±0.007	**1.00±0.001**
Airfoil	**0.74±0.002**	0.82±0.006
Red Wine	**0.90±0.020**	0.99±0.002
Skill Craft	0.88±0.013	**0.85±0.005**
Elevator	**0.70±0.004**	0.76±0.003
CCPP	0.42±0.003	**0.36±0.002**
Bike Share	0.83±0.002	**0.64±0.001**
Energy Data	**0.89±0.001**	0.98±0.001
Pm 25	**0.89±0.006**	0.94±0.001
Average Rank	1.57	1.42

This is similar to what was observed in classification problems, where the introduction of an archive-based pheromone model did significantly improve the runtime by eliminating the need for a discretisation procedure. Ant-Miner-Reg uses the M5 dynamic discretisation procedure when creating terms for continuous attributes, which is slow, 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.

Table 39 shows that Ant-Miner-Reg$_{\text{MA}}$ achieved a statistically significant improvement regarding the computational time at the 5% significance level ($p = 0.00016$) with respect to Ant-Miner-Reg, according to the Wilcoxon signed-rank

Table 38: Average computational runtime (average ± standard error) in seconds measured by five runs of tenfold cross-validation. The value of the fastest algorithm for a given data set is shown in bold.

Data set	Ant-Miner-Reg	Ant-Miner-Reg$_{\text{MA}}$
WPBC_r	1.94±0.245	**0.51±0.024**
CPU	0.91±0.178	**0.24±0.002**
Yacht	0.55±0.088	**0.20±0.004**
MPG	2.03±0.432	**0.26±0.003**
Housing	8.80±0.631	**0.44±0.011**
Forest Fire	15.24±1.013	**0.99±0.010**
Istanbul	7.70±1.021	**0.39±0.013**
Efficiency	**0.42±0.093**	0.71±0.011
Stock	8.33±0.763	**0.86±0.034**
Concrete	38.12±1.250	**1.29±0.048**
Flare	13.26±1.751	**0.85±0.027**
Airfoil	11.75±0.314	**0.93±0.030**
Red Wine	136.49±9.473	**0.94±0.016**
Skill Craft	472.19±25.562	**10.12±0.246**
Elevator	208.35±13.446	**4.65±0.219**
CCPP	7.33±0.106	**2.30±0.040**
Bike Share	7,568.04±164.315	**223.53±4.988**
Energy Data	25,718.16±1,129.573	**582.39±44.681**
Pm 25	43,919.93±2,878.527	**615.25±16.038**
Rank	1.05	1.94

test. Although Ant-Miner-Reg$_{\text{MA}}$ did not significantly improve the RRMSE of the Ant-Miner-Reg, the significant improvement in runtime shows the advantage of using an archive-based pheromone model in regression problems.

9.3 Summary

This chapter presented a new ACO-based regression algorithm, called Ant-Miner-Reg$_{\text{MA}}$. Ant-Miner-Reg$_{\text{MA}}$ is an extension of Ant-Miner-Reg, where the dynamic discretisation procedure is replaced by the use of a solution archive. This modification allows Ant-Miner-Reg$_{\text{MA}}$ to cope with different attributes types (continuous

Table 39: Results of the Wilcoxon Signed-Rank test at the $\alpha = 0.05$ significance level comparing Ant-Miner-Reg$_{\mathrm{MA}}$ and Ant-Miner-Reg. Statistically significant differences are shown in bold, indicating the case where the performance of Ant-Miner-Reg$_{\mathrm{MA}}$ is statistically significantly better than the one of Ant-Miner-Reg.

	Sample size	W+	W-	Z	*p*
RRMSE	19	94	96	0.0402	0.9681
Runtime	19	1	189	-3.7828	**0.00016**

and categorical) directly. Computational results showed that the proposed algorithm significantly improved Ant-Miner-Reg's computational time.

The effect of using a solution archive is similar to the one observed in Ant-Miner$_{\mathrm{MA}}$ (Chapter 5), where the archive-based algorithm has an improved computational time without a negative impact on the predictive performance. This is an important extension, since it allows the proposed algorithm to be applied to larger data sets and/or used in domains where computational time is limited.

Chapter 10

Conclusion

In this thesis, we introduced novel Ant Colony Optimisation (ACO) algorithms in the context of both classification and regression tasks. The research focused on unexplored research areas: (1) extending ACO algorithms to cope with continuous attributes without the need of a discretisiation procedure, either static (pre-processing) or dynamic, in both classification and regression; (2) extending ACO algorithms to cope with data stream classification, where the data is not stationary as in traditional mining algorithms and the storage of data is limited.

The aim in (1) is to improve the computational time of the algorithm, in particular when handling large data sets, by removing the discretisation procedure. At the same time, the predictive accuracy should not be negatively affected. It should be noted that this is also important to allow ACO algorithms to handle data stream problems, since both access to all the data and computational time are limited. In (2), we extend the algorithm to cope with the challenges of data stream mining. This takes advantage of the proposed mechanism to cope with continuous attributes. All proposed algorithms discover IF-THEN rules, providing the advantage of creating comprehensible models. They have been compared against state-of-the-art algorithms from the literature in terms of both predictive performance and size of the discovered model.

10.1 Contributions

We started by proposing Ant-Miner$_{\mathrm{MA}}$ to tackle mixed-attribute classification problems based on ACO$_{\mathbf{MV}}$ (Liao et al. 2014). The use of a solution archive allows the algorithm to deal with categorical, continuous and ordinal attributes directly, without a requiring discretisation procedure. The rule creation process then uses ACO$_{\mathbf{MV}}$ strategies to sample values for each attribute type to create the antecedent of a rule. Ant-Miner$_{\mathrm{MA}}$ was compared against *c*Ant-Miner, an ACO-based classification algorithm capable of dealing with continuous attributes employing a dynamic discretisation procedure, using 30 publicly available data sets. Our results show that the proposed Ant-Miner$_{\mathrm{MA}}$ statistically significantly improves the computational time of *c*Ant-Miner with no negative effects on its predictive accuracy—in most cases, an order of magnitude improvements were observed. This enables Ant-Miner$_{\mathrm{MA}}$ to be applied to much larger data sets, mitigating the restriction on computational time.

Ant-Miner$_{\mathrm{MA}}$'s results indicated that in data sets with a relatively large number of attributes (greater than 50), there are no gains in computational time. In cases where there is a large number of attributes but a smaller number of instances, which results in the discretisation overhead being less noticeable, the graph-based algorithms in combination with the dynamic discretisation performed well. This observation indicates that the construction graph is useful to quickly select attributes that are effective, while the archive has the advantage of being able to handle multiple attribute types directly. To address the archive limitation, we introduced a novel approach to combine both graph and archive pheromone models. The use of the solution archive allows the algorithm to deal with all attribute types directly, including continuous attributes without requiring a discretisation procedure, while the graph pheromone model improves the selection of attribute conditions in data sets containing a large number of attributes. Instead of manually designing a new algorithm, we proposed a fully configurable framework (called Ant-Miner$_{\mathrm{MA+G}}$) using an automatic design process based on I/F-Race (López-Ibáñez et al. 2016), which is a state-of-the-art automatic algorithm configuration tool. Experiments using five different automatically designed configurations of Ant-Miner$_{\mathrm{MA+G}}$ show that the proposed framework performed competitively well against baseline algorithms.

For data stream mining, we proposed the Stream Ant-Miner (sAnt-Miner) algorithm for classification rule induction. sAnt-Miner uses a novel hybrid pheromone model, combining both graph and archive pheromone models to create classification rules. sAnt-Miner's hybrid pheromone model allowed the algorithm to benefit from the archive model for handling multiple attribute types and the graph model for selecting the best attributes to use when creating rules. This also allowed the rule construction process to be extended to include a Pittsburgh-based approach, where each ant creates a complete rule list and the search is guided by the quality of the rule list instead of guided by the quality of individual rules. As a result, the algorithm copes effectively with rule interactions (Otero, Freitas and Johnson 2013). Additionally, given the gains in computational time by using the archive to handle continuous attributes, sAnt-Miner iteratively improves the quality of the model using a sample of the data over a limited number of iterations — the current model is replaced when an improved one is created. sAnt-Miner was compared against the well-known VFDR (Gama and Kosina 2011) and Ge-Rules (Le et al. 2014) using standard benchmarks data sets. Our results showed that sAnt-Miner models had competitive predictive accuracy and reactivity[1] compared to VFDR and Ge-Rules models. Moreover, sAnt-Miner models were statistically significantly smaller when compared to VFDR and Ge-Rules — smaller models contribute to interpretability, potentially allowing users to understand the reasons for the predictions of the model.

[1] Based on the Kappa measures in MOA, which measures how fast the algorithm can react to different stream challenges.

Finally, we proposed a new regression algorithm based on our work with mixed-variable classification. The proposed algorithm Ant-Miner-Reg$_{\text{MA}}$, an extension of Ant-Miner-Reg (Brookhouse and Otero 2015), uses an archive-based pheromone model to handle both categorical and continuous attributes. Similarly to the results in the classification task, Ant-Miner-Reg$_{\text{MA}}$ results showed that the use of an archive-based pheromone model improved the runtime without negative effects on the algorithm's predictive accuracy.

10.2 Future Research

Future investigation is required to realise the full potential of adding the archive-based pheromone model to rule discovery in traditional data mining. Using an archive-based pheromone model improved the runtime of the proposed ACO-based algorithm. It would be interesting to further investigate the effect of incorporating a graph pheromone model in combination with an archive-based pheromone model, where the graph pheromone model is responsible for selecting attributes and the archive pheromone model for optimising their values, following a similar approach to the one used in sAnt-Miner. Moreover, the Pittsburgh approach of generating rule lists in each iteration of the algorithm, instead of a single rule, allowing rule interactions to be optimised, is also an interesting research direction worth further exploration. Pittsburgh-based rule construction procedures have achieved good results in *c*Ant-Miner$_{\text{PB}}$ (Otero, Freitas and Johnson 2013) and in the proposed sAnt-Miner. This research direction has the potential to lead to improved ACO-based algorithms for both classification and regression tasks.

In relation to data stream mining, there are several interesting directions. Currently the pheromone model on sAnt-Miner gets re-initialised when learning on a new buffer of instances, while the only link between the two different learning phases is the current best model. One possible extension is adding an adaptive evaporation rate on the graph, while keeping the pheromone between learning phases. Moreover, we could add different sampling or archiving strategies to cope with the unbalanced classes in the data sets, this was successfully implemented in a GP stream algorithm (Khanchi, Heywood and Zincir-Heywood 2017).

Currently, sAnt-Miner generates many rule lists at each iteration, but only one is used to make predictions. Instead of discarding older models, they could form an ensemble and a voting approach could be used to make the predictions — each rule list makes a prediction and the final prediction is a combination of them (Dietterich 2000). A related approach is to extend sAnt-Miner to operate as a rule ensemble (Hastie et al. 2005), where rules from different rule lists are combined. Both approaches have the potential to improve the predictive accuracy — although a trade-off between the size of the model (number of lists/rules) and accuracy should be considered.

sAnt-Miner does not have an explicit mechanism to detect concept drift. The current model gets replaced once a better one is created, but this only happens when the buffer of instances is full and the learning procedure is executed. Using an explicit mechanism to detect and trigger the model update could improve the overall performance of the algorithm. We can make the analogy of concept drift to dynamic optimisation problems, where the optimal solution dynamically changes over time. ACO algorithms have been shown to perform well under dynamic optimisation problems, employing pheromone evaporation strategies and direct communication mechanisms (Mavrovouniotis 2013; Mavrovouniotis and Yang 2014). It would be interesting to see if these approaches can be incorporated to cope with concept drift.

Finally, extending sAnt-Miner to work with regression problems is a research direction worth further exportation and likely to lead to interesting research questions, such as how to detect concept drift, what sampling and archive polices are more suitable. Our results show that ACO algorithms have the potential of being successful in dealing with data streams and we hope others would be encouraged to improve them.

Bibliography

Abbass, H. A., Bacardit, J., Butz, M. V. and Llora, X. (2004). Online adaptation in learning classifier systems: stream data mining. *Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign.*

Abraham, A., Das, S. and Roy, S. (2008). Swarm intelligence algorithms for data clustering. In *Soft computing for knowledge discovery and data mining*, Springer, pp. 279–313.

Aggarwal, C. (2009). Data streams: An overview and scientific applications. *Scientific Data Mining and Knowledge Discovery*, pp. 377–397.

Aggarwal, C. and Philip, S. (2005). Online analysis of community evolution in data streams. *SDM.*

Aggarwal, C., Han, J., Wang, J. and Yu, P. (2006). A framework for on-demand classification of evolving data streams. *IEEE Transactions on Knowledge and Data Engineering*, 18(5), pp. 577–589.

Aggarwal, C. C. and Subbian, K. (2012). Event detection in social streams. In *Proceedings of the 2012 SIAM international conference on data mining*, SIAM, pp. 624–635.

Aggarwal, C. C., Han, J., Wang, J. and Yu, P. S. (2003). A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference* *on Very Large Data Bases - Volume 29*, VLDB Endowment, VLDB '03, pp. 81–92.

Aha, D. W., Kibler, D. and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6(1), pp. 37–66.

Al-Behadili, H. N. K., Ku-Mahamud, K. R. and Sagban, R. (2018). Rule pruning techniques in the ant-miner classification algorithm and its variants: A review. In *2018 IEEE Symposium on Computer Applications Industrial Electronics (ISCAIE)*, pp. 78–84.

Alcalá-Fdez, J. et al. (2009). Keel: a software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, 13(3), pp. 307–318.

Almeida, E., Kosina, P. and Gama, J. (2013). Random rules from data streams. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, ACM, pp. 813–814.

Alon, N., Matias, Y. and Szegedy, M. (1999). The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1), pp. 137 – 147.

Andreoni Lopez, M., Mattos, D. M., Duarte, O. C. M. and Pujolle, G. (2019). Toward a monitoring and threat detection system based on stream processing as a virtual network function for big data. *Concurrency and Computation: Practice and Experience*, p. e5344.

Babcock, B., Babu, S., Datar, M., Motwani, R. and Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, New York, NY, USA: ACM, PODS '02, pp. 1–16.

Baena-Garcıa, M. et al. (2006). Early drift detection method. In *Fourth international workshop on knowledge discovery from data streams*, vol. 6, pp. 77–86.

Barnston, A. G. (1992). Correspondence among the correlation, rmse, and heidke forecast verification measures; refinement of the heidke score. *Weather and Forecasting*, 7(4), pp. 699–709.

Beckers, R., Holland, O. E. and Deneubourg, J.-L. (2000). Fom local actions to global tasks: Stigmergy and collective robotics. In *Prerational Intelligence: Adaptive Behavior and Intelligent Systems Without Symbols and Logic, Volume 1, Volume 2 Prerational Intelligence: Interdisciplinary Perspectives on the Behavior of Natural and Artificial Systems, Volume 3*, Springer, pp. 1008–1022.

Bianchi, L., Gambardella, L. M. and Dorigo, M. (2002). An ant colony optimization approach to the probabilistic traveling salesman problem. In *Parallel Problem Solving from Nature—PPSN VII*, Springer, pp. 883–892.

Bifet, A. and Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, SIAM, pp. 443–448.

Bifet, A. and Gavaldà, R. (2009). Adaptive learning from evolving data streams. In *Proceedings of the 8th International Symposium on Intelligent Data Analysis: Advances in Intelligent Data Analysis VIII*, Berlin, Heidelberg: Springer-Verlag, IDA '09, pp. 249–260.

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R. and Gavaldà, R. (2009). New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, KDD '09, pp. 139–148.

Bifet, A., Holmes, G., Kirkby, R. and Pfahringer, B. (2010). MOA: Massive online analysis. *Journal of Machine Learning Research*, 11, pp. 1601–1604.

Bifet, A., de Francisci Morales, G., Read, J., Holmes, G. and Pfahringer, B. (2015). Efficient online evaluation of big data stream classifiers. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, KDD '15, pp. 59–68.

Birattari, M., Yuan, Z., Balaprakash, P. and Stützle, T. (2010). F-race and iterated f-race: An overview. In *Experimental Methods for the Analysis of Optimization Algorithms*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 311–336.

Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, 2(4), pp. 353–373.

Booker, L., Goldberg, D. and Holland, J. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, 40(1), pp. 235 – 282.

Bouchachia, A. and Vanaret, C. (2014). GT2FC: An online growing interval type-2 self-learning fuzzy classifier. *IEEE Transactions on Fuzzy Systems*, 22(4), pp. 999–1018.

Breiman, L., Friedman, J., Olshen, R. and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth.

Brookhouse, J. and Otero, F. E. (2015). Discovering regression rules with ant colony optimization. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, New York, NY, USA: ACM, GECCO Companion '15, pp. 1005–1012.

Brookhouse, J. and Otero, F. E. (2016). Using an ant colony optimization algorithm for monotonic regression rule discovery. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, New York, NY, USA: ACM, GECCO '16, pp. 437–444.

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

Brzeziński, D. (2010). *Mining data streams with concept drift*. Ph.D. thesis, MS thesis, Dept. of Computing Science and Management, Poznan University of Technology, Poznan, Poland.

Cao, F., Ester, M., Qian, W. and Zhou, A. (2006). Density-based clustering over an evolving data stream with noise. In *In 2006 SIAM Conference on Data Mining*, pp. 328–339.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4), pp. 349 – 370.

Cervantes, A., Isasi, P., Gagné, C. and Parizeau, M. (2013). Learning from non-stationary data using a growing network of prototypes. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pp. 2634–2641.

Chanda, P., Cho, Y., Zhang, A. and Ramanathan, M. (2009). Mining of attribute interactions using information theoretic metrics. In *2009 IEEE International Conference on Data Mining Workshops*, pp. 350–355.

Chaudhuri, S., Motwani, R. and Narasayya, V. (1999). On random sampling over joins. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA: ACM, SIGMOD '99, pp. 263–274.

Chen, L. and Shang, S. (2019). Region-based message exploration over spatio-temporal data streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 873–880.

Chen, Y. and Tu, L. (2007). Density-based clustering for real-time stream data. In

Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA: ACM, KDD ’07, pp. 133–142.

Chien, S. and Immorlica, N. (2005). Semantic similarity between search engine queries using temporal correlation. In *Proceedings of the 14th international conference on World Wide Web*, ACM, pp. 2–11.

Cohen, W. W. (1995). Fast effective rule induction. In *Machine learning proceedings 1995*, Elsevier, pp. 115–123.

Cui, W. et al. (2011). Textflow: Towards better understanding of evolving topics in text. *IEEE transactions on visualization and computer graphics*, 17(12), pp. 2412–2421.

Dam, H. H. and Lokan, H. A., Chrisand Abbass (2007). Evolutionary online data mining: An investigation in a dynamic environment. In *Evolutionary Computation in Dynamic and Uncertain Environments*, Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 153–178.

Dang, X. H., Lee, V., Ng, W. K., Ciptadi, A. and Ong, K. L. (2009). An em-based algorithm for clustering data streams in sliding windows. In *Proceedings of the 14th International Conference on Database Systems for Advanced Applications*, Berlin, Heidelberg: Springer-Verlag, DASFAA ’09, pp. 230–235.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7(Jan), pp. 1–30.

Dietterich, T. G. (1997). Machine-learning research. *AI magazine*, 18(4), p. 97.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, Springer, pp. 1–15.

Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, KDD '00, pp. 71–80.

Domingos, P. and Hulten, G. (2003). A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12(4), pp. 945–949.

Dorigo, M. and Gambardella, L. M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1), pp. 53–66.

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

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. Cambridge: MIT Press.

Doshi-Velez, F. and Kim, B. (2017). Towards A Rigorous Science of Interpretable Machine Learning. *arXiv e-prints*, arXiv:1702.08608, `1702.08608`.

Fahrmeir, L., Kneib, T., Lang, S. and Marx, B. (2013). *Regression: models, methods and applications*. Springer Science & Business Media.

Fan, W. and Bifet, A. (2013). Mining big data: Current status, and forecast to the future. *SIGKDD Explor Newsl*, 14(2), pp. 1–5.

Fang, M., Shivakumar, N., Garcia-Molina, H., Motwani, R. and Ullman, J. D. (1998). Computing iceberg queries efficiently. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., VLDB '98, pp. 299–310.

Fayyad, U., Piatetsky-Shapiro, G. and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI magazine*, 17(3), p. 37.

Frank, E. and Witten, I. H. (1998). Generating accurate rule sets without global optimization. In *Proceedings of the Fifteenth International Conference on Machine Learning*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., ICML '98, pp. 144–151.

Freitas, A. A. (2001). Understanding the crucial role of attribute interaction in data mining. *Artificial Intelligence Review*, 16(3), pp. 177–199.

Freitas, A. A. (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Berlin, Heidelberg: Springer-Verlag.

Freitas, A. A. (2014). Comprehensible classification models: A position paper. *SIGKDD Explor Newsl*, 15(1), pp. 1–10.

Freitas, A. A., Wieser, D. C. and Apweiler, R. (2010). On the importance of comprehensible classification models for protein function prediction. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 7(1), pp. 172–182.

Freund, Y. and Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, London, UK, UK: Springer-Verlag, EuroCOLT '95, pp. 23–37.

Fürnkranz, J. (2005). From local to global patterns: Evaluation issues in rule learning algorithms. In *Local pattern detection*, Springer, pp. 20–38.

Gaber, M., Zaslavsky, A. and Krishnaswamy, S. (2005). Mining data streams: a review. *ACM Sigmod Record*, 34(2), p. 18.

Gaber, M. M., Zaslavsky, A. and Krishnaswamy, S. (2007). A survey of classification methods in data streams. In *Data streams*, Springer, pp. 39–59.

Gama, J. (2013). Data stream mining: the bounded rationality. *Informatica*, 37(1).

Gama, J. and Gaber, M. M. (2007). *Learning from data streams: processing techniques in sensor networks*. Springer.

Gama, J., Sebastião, R. and Rodrigues, P. (2012). On evaluating stream learning algorithms. *Mach Learn*, 90(3), pp. 317–346.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M. and Bouchachia, A. (2014). A survey on concept drift adaptation. *Acm Comput Surv*, 46(4), pp. 1–37.

Gama, J. a. and Kosina, P. (2011). Learning decision rules from data streams. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, AAAI Press, IJCAI'11, pp. 1255–1260.

Gambardella, L. M., Taillard, E. and Agazzi, G. (1999). Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. Tech. rep., Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale.

Gao, J., Fan, W., Han, J. and Philip, S. Y. (2007). A general framework for mining concept-drifting data streams with skewed distributions. In *SDM*, SIAM, pp. 3–14.

Gilbert, A. C. et al. (2002). Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, New York, NY, USA: ACM, STOC '02, pp. 389–398.

Gomes, H. M. et al. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9), pp. 1469–1495.

Grassé, P.-P. (1959). La reconstruction du nid et les coordinations interindividuelles chezbellicositermes natalensis etcubitermes sp. la théorie de la stigmergie: Essai d'interprétation du comportement des termites constructeurs. *Insectes Sociaux*, 6(1), pp. 41–80.

Greenwald, M. and Khanna, S. (2001). Space-efficient online computation of quantile summaries. *SIGMOD Rec*, 30(2), pp. 58–66.

Guha, S., Kim, C. and Shim, K. (2004). Xwave: Optimal and approximate extended wavelets. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB Endowment, VLDB '04, pp. 288–299.

Guha, S., Mishra, N., Motwani, R. and O'Callaghan, L. (2000). Clustering data streams. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pp. 359–366.

Guntsch, M. and Middendorf, M. (2001). Pheromone modification strategies for ant algorithms applied to dynamic tsp. In E. Boers, ed., *Applications of Evolutionary Computing*, *Lecture Notes in Computer Science*, vol. 2037, Springer Berlin Heidelberg, pp. 213–222.

Guo, J., Zhang, P., Tan, J. and Guo, L. (2011). Mining frequent patterns across multiple data streams. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, New York, NY, USA: ACM, CIKM '11, pp. 2325–2328.

Hall, M. et al. (2009). The weka data mining software: An update. *SIGKDD Explor Newsl*, 11(1), pp. 10–18.

Hansen, L. K. and Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10), pp. 993–1001.

Hashemi, S., Yang, Y., Mirzamomen, Z. and Kangavari, M. (2009). Adapted one-versus-all decision trees for data stream classification. *IEEE Transactions on Knowledge and Data Engineering*, 21(5), pp. 624–637.

Hastie, T., Tibshirani, R., Friedman, J. and Franklin, J. (2005). The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2), pp. 83–85.

Helal, A., Brookhouse, J. and Otero, F. E. B. (2018). Archive-Based Pheromone Model for Discovering Regression Rules with Ant Colony Optimization. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, pp. 1–7.

Helal, A. and Otero, F. E. (2016). A Mixed-Attribute Approach in Ant-Miner Classification Rule Discovery Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ACM, GECCO '16, pp. 13–20.

Helal, A. and Otero, F. E. B. (2017). Automatic Design of Ant-Miner Mixed Attributes for Classification Rule Discovery. In *Proceedings of the Genetic and Evolutionary Computation Conference*, ACM Press, GECCO '17, pp. 433–440.

Helal, A. and Otero, F. E. B. (2019). Data Stream Classification with Ant Colony Optimization, submitted to IEEE transactions on Evolutionary Computation, under review.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301), pp. 13–30.

Hoens, T. R., Polikar, R. and Chawla, N. V. (2012). Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence*, 1(1), pp. 89–101.

Holmes, G., Donkin, A. and Witten, I. H. (1994). Weka: a machine learning workbench. In *Intelligent Information Systems,1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pp. 357–361.

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

Hulten, G., Spencer, L. and Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, KDD '01, pp. 97–106.

Jagadish, H. V. et al. (1998). Optimal histograms with quality guarantees. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., VLDB '98, pp. 275–286.

Jain, A. K. and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Janssen, F. and Fürnkranz, J. (2010a). On the quest for optimal rule learning heuristics. *Machine Learning*, 78(3), pp. 343–379.

Janssen, F. and Fürnkranz, J. (2010b). Separate-and-conquer regression. In *LWA*, pp. 81–88.

Kantardzic, M. (2011). *Data mining: concepts, models, methods, and algorithms*. John Wiley & Sons.

Katakis, I., Tsoumakas, G., Banos, E., Bassiliades, N. and Vlahavas, I. (2009). An adaptive personalized news dissemination system. *Journal of Intelligent Information Systems*, 32(2), pp. 191–212.

Kelly, M. G., Hand, D. J. and Adams, N. M. (1999). The impact of changing populations on classifier performance. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, KDD '99, pp. 367–371.

Khanchi, S., Heywood, M. I. and Zincir-Heywood, A. N. (2017). Properties of a gp active learning framework for streaming data with class imbalance. In

Proceedings of the Genetic and Evolutionary Computation Conference, New York, NY, USA: ACM, GECCO ’17, pp. 945–952.

Kisilevich, S., Mansmann, F., Nanni, M. and Rinzivillo, S. (2009). Spatio-temporal clustering. In O. Maimon and L. Rokach, eds., *Data Mining and Knowledge Discovery Handbook*, Boston, MA: Springer US, pp. 855–874.

Kolter, J. Z. and Maloof, M. A. (2007). Dynamic weighted majority: An ensemble method for drifting concepts. *J Mach Learn Res*, 8, pp. 2755–2790.

Krempl, G. et al. (2014). Open challenges for data stream mining research. *SIGKDD Explor Newsl*, 16(1), pp. 1–10.

Kudo, R. et al. (2019). Real-time event search using social stream for inbound tourist corresponding to place and time. *International Journal of Big Data Intelligence*, 6(3-4), pp. 248–258.

Lawler, E. (1985). *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley-Interscience series in discrete mathematics and optimization, John Wiley & sons.

Lazarescu, M. M., Venkatesh, S. and Bui, H. H. (2004). Using multiple windows to track concept drift. *Intell Data Anal*, 8(1), pp. 29–59.

Le, T., Stahl, F., Gomes, J. B., Gaber, M. M. and Fatta, G. D. (2014). *Research and Development in Intelligent Systems XXXI: Incorporating Applications and Innovations in Intelligent Systems XXII*, Cham: Springer International Publishing, chap. Computationally Efficient Rule-Based Classification for Continuous Streaming Data. pp. 21–34.

Le, T., Stahl, F., Gaber, M. M., Gomes, J. B. and Fatta, G. D. (2017). On expressiveness and uncertainty awareness in rule-based classification for data streams.

Neurocomputing, 265, pp. 127 – 141, new Trends for Pattern Recognition: Theory & Applications.

Lee, D.-S. (2005). Effective gaussian mixture learning for video background subtraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5), pp. 827–832.

Leite, D., Costa, P. and Gomide, F. (2010). Evolving granular neural network for semi-supervised data stream classification. In *Neural Networks (IJCNN), The 2010 International Joint Conference on*, pp. 1–8.

Leite, D. F., Jr, P. and Gomide, F. (2009). Evolving granular classification neural networks for semi-supervised data stream classification. *IEEE*, pp. 1736–1743.

Liang, N.-Y., Huang, G.-B., Saratchandran, P. and Sundararajan, N. (2006). A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 17(6), pp. 1411–1423, cited By 1112.

Liang, Z. et al. (2016). A novel multiple rule sets data classification algorithm based on ant colony algorithm. *Appl Soft Comput*, 38(C), pp. 1000–1011.

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

Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of California, School of Information and Computer Science. [http://archive.ics.uci.edu/ml].

Liu, B., Abbas, H. and McKay, B. (2003). Classification rule discovery with ant colony optimization. In *Intelligent Agent Technology, 2003. IAT 2003. IEEE/WIC International Conference on*, pp. 83–88.

Liu, B., Abbass, H. A. and McKay, B. (2002). Density-based heuristic for rule discovery with ant-miner. In *The 6th Australia-Japan joint workshop on intelligent and evolutionary system*, vol. 184, Citeseer, pp. 180–184.

López-Ibáñez, M. and Stützle, T. (2012). The automatic design of multiobjective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6), pp. 861–875.

López-Ibáñez, M., Dubois-Lacoste, J., Caceres, L. P., Stützle, T. and Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3, pp. 43–58.

Lühr, S. and Lazarescu, M. (2009). Incremental clustering of dynamic data streams using connectivity based representative points. *Data Knowl Eng*, 68(1), pp. 1–27.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Berkeley, Calif.: University of California Press, pp. 281–297.

Maimon, O. and Rokach, L. (2010). *Data Mining and Knowledge Discovery Handbook*. Springer Publishing Company, Incorporated, 2nd edn.

Malerba, D., Esposito, F. and Semeraro, G. (1996). A further comparison of simplification methods for decision-tree induction. In *Learning from data*, Springer, pp. 365–374.

Martens, D., Baesens, B. and Fawcett, T. (2011). Editorial survey: swarm intelligence for data mining. *Machine Learning*, 82(1), pp. 1–42.

Martens, D. et al. (2007). Classification with ant colony optimization. *Evolutionary Computation, IEEE Transactions on*, 11(5), pp. 651–665.

Martin, B. (1995). Instance-based learning : Nearest neighbor with generalization. Tech. rep., University of Waikato Research Commons, working Paper.

Mavrovouniotis, M. (2013). *Ant Colony Optimization in Stationary and Dynamic Environments*. Ph.D. thesis, University of Leicester.

Mavrovouniotis, M. and Yang, S. (2014). Interactive and non-interactive hybrid immigrants schemes for ant algorithms in dynamic environments. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, IEEE, pp. 1542–1549.

Merkle, D., Middendorf, M. and Schmeck, H. (2002). Ant colony optimization for resource-constrained project scheduling. *Evolutionary Computation, IEEE Transactions on*, 6(4), pp. 333–346.

Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2), pp. 227–243.

Minku, L. and Yao, X. (2012). DDD: A new ensemble approach for dealing with concept drift. *Knowledge and Data Engineering, IEEE Transactions on*, 24(4), pp. 619–633.

Minku, L. L., White, A. P. and Yao, X. (2010). The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 22(5), pp. 730–742.

Mitchell, T. M. (1997). *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1st edn.

Motwani, R. and Raghavan, P. (1995). *Randomized Algorithms*. New York, NY, USA: Cambridge University Press.

Mundhe, R. V. and Manwade, K. B. (2018). Continuous top-k monitoring on document streams. In *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 1008–1013.

Muthukrishnan, S. (2003). Data streams: Algorithms and applications. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, SODA '03, pp. 413–413.

Nguyen, H., Woon, Y. and Ng, W. (2015). A survey on data stream clustering and classification. *Knowledge and information systems*.

Nguyen, H.-L., Woon, Y.-K., Ng, W.-K. and Wan, L. (2012). Heterogeneous ensemble for feature drifts in data streams. In *Proceedings of the 16th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*, Berlin, Heidelberg: Springer-Verlag, PAKDD'12, pp. 1–12.

Olmo, J., Romero, J. and Ventura, S. (2012). Classification rule mining using ant programming guided by grammar with multiple pareto fronts. *Soft Computing*, 16(12), pp. 2143–2163.

Olmo, J. L., Romero, J. R. and Ventura, S. (2011). Using ant programming guided by grammar for building rule-based classifiers. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 41(6), pp. 1585–1599.

Otero, F. E., Freitas, A. A. and Johnson, C. G. (2008). cant-miner: An ant colony classification algorithm to cope with continuous attributes. In M. Dorigo, M. Birattari, C. Blum, M. Clerc, T. Stützle and A. Winfield, eds., *Ant Colony Optimization and Swarm Intelligence*, *Lecture Notes in Computer Science*, vol. 5217, Springer Berlin Heidelberg, pp. 48–59.

Otero, F. E., Freitas, A. A. and Johnson, C. G. (2009). Handling continuous attributes in ant colony classification algorithms. In *Computational Intelligence and Data Mining, 2009. CIDM '09. IEEE Symposium on*, pp. 225–231.

Otero, F. E., Freitas, A. A. and Johnson, C. G. (2013). A new sequential covering strategy for inducing classification rules with ant colony algorithms. *IEEE Trans Evolutionary Computation*, 17(1), pp. 64–76.

Oza, N. C. (2005). Online bagging and boosting. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, vol. 3, pp. 2340–2345 Vol. 3.

Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial optimization: algorithms and complexity*. Courier Corporation.

Park, N. H. and Lee, W. S. (2004). Statistical grid-based clustering over data streams. *SIGMOD Rec*, 33(1), pp. 32–37.

Parpinelli, R., Lopes, H. and Freitas, A. (2002). Data mining with an ant colony optimization algorithm. *Evolutionary Computation, IEEE Transactions on*, 6(4), pp. 321–332.

Pazzani, M. J., Mani, S. and Shankle, W. R. (2001). Acceptance of rules generated by machine learning among medical experts. *Methods of information in medicine*, 40(05), pp. 380–385.

Piateski, G. and Frawley, W. (1991). *Knowledge discovery in databases*. MIT press.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. (1992). *Numerical Recipes in C (2Nd Ed.): The Art of Scientific Computing*. New York, NY, USA: Cambridge University Press.

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

Quinlan, J. R. (1987). Generating production rules from decision trees. In *ijcai*, vol. 87, Citeseer, pp. 304–307.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Quinlan, J. R. et al. (1992). Learning with continuous classes. In *5th Australian joint conference on artificial intelligence*, vol. 92, World Scientific, pp. 343–348.

Rai, P., Daumé, H. and Venkatasubramanian, S. (2009). Streamed learning: One-pass svms. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., IJCAI'09, pp. 1211–1216.

Sakaki, T., Okazaki, M. and Matsuo, Y. (2010). Earthquake shakes twitter users: Real-time event detection by social sensors. In *Proceedings of the 19th International Conference on World Wide Web*, New York, NY, USA: ACM, WWW '10, pp. 851–860.

Salama, K. M., Abdelbar, A. M., Otero, F. E. and Freitas, A. A. (2013). Utilizing multiple pheromones in an ant-based algorithm for continuous-attribute classification rule discovery. *Applied Soft Computing*, 13(1), pp. 667 – 675.

Salganicoff, M. (1997). Tolerating concept and sampling shift in lazy learning usingprediction error context switching. *Artif Intell Rev*, 11(1-5), pp. 133–155.

Sancho-Asensio, A., Orriols-Puig, A. and Golobardes, E. (2014). Robust on-line neural learning classifier system for data stream classification tasks. *Soft Computing*, 18(8), pp. 1441–1461.

Schapire, R. E. (1990). The strength of weak learnability. *Mach Learn*, 5(2), pp. 197–227.

Seidlova, R., Poživil, J. and Seidl, J. (2019). Marketing and business intelligence with help of ant colony algorithm. *Journal of Strategic Marketing*, 27(5), pp. 451–463.

Shelokar, P., Jayaraman, V. and Kulkarni, B. (2004). An ant colony approach for clustering. *Analytica Chimica Acta*, 509(2), pp. 187 – 195.

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*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., IJCAI'83, pp. 422–425.

Smith, T. and Alahakoon, D. (2009). *Foundations of Computational Intelligence Volume 4: Bio-Inspired Data Mining*, Berlin, Heidelberg: Springer Berlin Heidelberg, chap. Growing Self-Organizing Map for Online Continuous Clustering. pp. 49–83.

Socha, K. and Dorigo, M. (2008). Ant colony optimization for continuous domains. *European Journal of Operational Research*, 185(3), pp. 1155 – 1173.

Sow, D., Biem, A., Blount, M., Ebling, M. and Verscheure, O. (2010). Body sensor data processing using stream computing. In *Proceedings of the International Conference on Multimedia Information Retrieval*, New York, NY, USA: ACM, MIR '10, pp. 449–458.

Stahl, F., Gaber, M. M. and Salvador, M. M. (2012). *Research and Development in Intelligent Systems XXIX: Incorporating Applications and Innovations in Intelligent Systems XX Proceedings of AI-2012, The Thirty-second SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, London: Springer London, chap. eRules: A Modular Adaptive Classification Rule Learning Algorithm for Data Streams. pp. 65–78.

Stecher, J., Janssen, F. and Fürnkranz, J. (2014). Separating rule refinement and rule selection heuristics in inductive rule learning. In T. Calders, F. Esposito, E. Hüllermeier and R. Meo, eds., *Machine Learning and Knowledge Discovery in Databases*, *Lecture Notes in Computer Science*, vol. 8726, Springer Berlin Heidelberg, pp. 114–129.

Street, W. N. and Kim, Y. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, KDD '01, pp. 377–382.

Stützle, T. and Hoos, H. H. (2000). Max-min ant system. *Future Gener Comput Syst*, 16(9), pp. 889–914.

Sun, J., Faloutsos, C. and Papadimitriou, S. (2007). Graphscope: parameter-free mining of large time-evolving graphs. *Proceedings of the 13th ...*.

Tasoulis, D. K., Ross, G. and Adams, N. M. (2007). Visualising the cluster structure of data streams. In *Proceedings of the 7th International Conference on Intelligent Data Analysis*, Berlin, Heidelberg: Springer-Verlag, IDA'07, pp. 81–92.

ur Rehman, M. H., Chang, V., Batool, A. and Wah, T. Y. (2016). Big data reduction framework for value creation in sustainable enterprises. *International Journal of Information Management*, 36(6), pp. 917–928.

Utgoff, P. E., Berkman, N. C. and Clouse, J. A. (1997). Decision tree induction based on efficient tree restructuring. *Mach Learn*, 29(1), pp. 5–44.

Vahdat, A., Atwater, A., McIntyre, A. R. and Heywood, M. I. (2014a). On the application of gp to streaming data classification tasks with label budgets. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, New York, NY, USA: ACM, GECCO Comp '14, pp. 1287–1294.

Vahdat, A., Atwater, A., McIntyre, A. R. and Heywood, M. I. (2014b). On the application of GP to streaming data classification tasks with label budgets. In *Proceedings of the Companion Publication of the 2014 Annual Conference on*

Genetic and Evolutionary Computation, New York, NY, USA: ACM, GECCO Comp '14, pp. 1287–1294.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. New York, NY, USA: Springer-Verlag New York, Inc.

Vitter, J. S. (1985). Random sampling with a reservoir. *ACM Trans Math Softw*, 11(1), pp. 37–57.

Vivekanandan, P. and Nedunchezhian, R. (2011). Mining data streams with concept drifts using genetic algorithm. *Artificial Intelligence Review*, 36(3), pp. 163–178.

Wang, H., Fan, W., Yu, P. S. and Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, KDD '03, pp. 226–235.

Weiss, S. M. and Kulikowski, C. A. (1991). *Computer Systems That Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Wilcoxon, F. (1992). Individual comparisons by ranking methods. In *Breakthroughs in statistics*, Springer, pp. 196–202.

Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evol Comput*, 3(2), pp. 149–175.

Witten, I. H., Frank, E., Hall, M. A. and Pal, C. J. (2016). *Data Mining: Practical machine learning tools and techniques*. San Francisco, CA, USA: Morgan Kaufmann.

Wu, X. et al. (2008). Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1), pp. 1–37.

Xu, S. and Wang, J. (2017). Dynamic extreme learning machine for data stream classification. *Neurocomput*, 238(C), pp. 433–449.

Yang, L., Li, K., Zhang, W. and Ke, Z. (2017). Ant colony classification mining algorithm based on pheromone attraction and exclusion. *Soft Computing*, 21(19), pp. 5741–5753.

Yu, H., Sun, X. and Wang, J. (2019). Ensemble os-elm based on combination weight for data stream classification. *Applied Intelligence*, 49(6), pp. 2382–2390.

Zeng, H.-J., He, Q.-C., Chen, Z., Ma, W.-Y. and Ma, J. (2004). Learning to cluster web search results. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, pp. 210–217.

Zhang, P., Zhu, X., Shi, Y. and Wu, X. (2009). *Advances in Knowledge Discovery and Data Mining: 13th Pacific-Asia Conference, PAKDD 2009 Bangkok, Thailand, April 27-30, 2009 Proceedings*, Berlin, Heidelberg: Springer Berlin Heidelberg, chap. An Aggregate Ensemble for Mining Concept Drifting Data Streams with Noise. pp. 1021–1029.

Zhang, P., Gao, B. J., Zhu, X. and Guo, L. (2011a). Enabling fast lazy learning for data streams. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pp. 932–941.

Zhang, P. et al. (2011b). Enabling fast prediction for ensemble models on data streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA: ACM, KDD 2011, pp. 177–185.

Zhang, P., Zhu, X., Shi, Y., Guo, L. and Wu, X. (2011c). Robust ensemble learning for mining noisy data streams. *Decis Support Syst*, 50(2), pp. 469–479.

Zhang, Z. and Zhang, R. (2008). *Multimedia data mining: a systematic introduction to concepts and theory*. Chapman and Hall/CRC.

Zhou, A., Cao, F., Qian, W. and Jin, C. (2007). Tracking clusters in evolving data streams over sliding windows. *Knowledge and Information Systems*, 15(2), pp. 181–214.