DISTRIBUTED DATA MINING: A MULTIAGENT APPROACH

by

Cuong Trung Tong

BIT, GradDip (InfSc)

A thesis submitted in fulfilment of the requirements for the degree of Master of Information Science.

Faculty of Information Sciences and Engineering
University of Canberra

June 2011

Copyright

Acknowledgement

I feel so grateful to numerous people who have helped me with their guidance, time, support, and encouragement through out this research.

First and foremost, I would like to express my sincere gratitude to my supervisory panel Professor Dharmendra Sharma and Dr Fariba Shadabi for the professional and personal guidance that go far beyond their responsibilities. It is their patient guidance, gentle encouragement and advices that led me through this journey that seemed impossible at times.

I would like to thank my research colleagues and the staff at Faculty of Information Science and Engineering, University of Canberra, especially Associate Professor Dat Tran, Dr Kim Le , Professor John Campbell and Dr Wan Li Ma for their time and supports during my time at the University.

Finally, I would like to express my deepest gratitude to my parents, for their loves, sacrifices and encouragements.

Abstract

Data mining on large datasets using a batch approach is time consuming and expensive. Training a large dataset can be time-consuming and in some cases may not be practical or even possible. In addition, batch learning introduces a single point of failure – this means that the training process may crash at any one point during the job and the whole process would need to be restarted.

This research advances the understanding of a multi-agent approach to data mining of large datasets. An agent mining model called DMMAS (Distributed Mining Multi-Agent System) is developed for the purpose of building accurate and transparent classifiers and improving the efficiency of mining a large dataset.

In our case study utilising the DMMAS model, the Pima Indian Diabetes dataset and US Census Adult dataset were used. They are well-known benchmark data from the UCI (University of California, Irvine) machine learning repository. This study found that the processing speed is improved as the result of the multi-agent mining approach, although there can be a corresponding marginal loss of accuracy. This loss of accuracy gap tends to close over time as more data becomes available.

The DMMAS approach provides a new, innovative data mining model, with great research and commercial potential for distributing mining across several agents and possibly different data sources. This research also reinforces the idea that combining multiagent and data mining approaches is a logical extension for large scale data mining applications.

Table of Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1. Background

The rapid development of computer technology in recent decades has introduced a data explosion challenge. The explosion of data has occurred in many domains, ranging from browser mouse clicks, scientific data, medical data, demographic data, financial data, web search queries, network traffic logs, to structured medical data (Muthukrishnan 2005). In the medical domain, for example, a particle super-collider system used for atomic research accumulates thousands of gigabytes of data in less than a year (Dwivedi 2001). The Internet search company, Google processed more than twenty petabytes (1 petabyte is 1000 terabytes, 1 terabyte is 1000 gigabytes) of data per day in 2008 (Dean & Ghemawat 2008).

This data explosion phenomenon has attracted a lot of interest in the area of data mining research, in particular from a data mining and multiagent integration perspective – or so called agent mining. Agent mining is a hybrid approach that aims to address the efficiency and scalability challenges of mining large datasets (Klusch, Lodi & Moro 2003). This approach has enjoyed success in the development of large complex systems (Shadabi & Sharma 2008).

1.2. Motivation

The current data mining approaches concentrate on methods for one-pass training of a dataset. In order to mine new data, previously trained models need to be retrained with the updated data. This approach is acceptable for a small dataset; however when applying to large datasets, this can be inefficient. In some instances, it may not be practical due to resource and time constraints. This motivates the need for distributed data mining algorithms. Such algorithms aim to partition and divide data mining using multiple agents, and build classifiers, by integrating outputs from each agent. Such proposed model would optimise the process and could be used to build classifier incrementally without the need to run the data mining algorithm on existing dataset.

In a distributed environment, relevant data often resides in different physical databases. When using a traditional approach, all the distributed datasets need to be aggregated to one central location prior to executing a data mining job. Aggregating all data to a central location for learning incurs high communication costs, making it a less efficient way to deal with distributed databases (Han & Kamber 2001).

1.3. Objectives

This research proposes an agent mining approach called DMMAS (Distributed Mining Multi-Agent System), which uses an agent mining approach to mine large datasets in a distributed environment. This work aims to make a scholarly contribution to the area of data mining and multi-agent systems. The research will produce findings that will be of interest in large dataset classification mining research.

1.4. Research Questions

- How can we perform data mining tasks on large datasets that do not fit in main memory?
- Can we use multi-agents for distributing data mining tasks?
- Can multiple classifiers be combined into one main classifier for classification tasks?

- How can we optimise combination of sub classifiers as generated by multiple agents?

1.5. Research Scope

This research contributes to the field of agent mining integration. The approach presented in this research is suitable for situations where efficiency is vital, such as in time critical data mining processes. The proposed model aims to support incrementally and classification of data through different sources and channels.

We have chosen to limit the scope of this research to consider only classification task.

1.6. Thesis Roadmap

Next chapter briefly outlines some background concepts that are used throughout the thesis. We explore various relevant concepts and techniques in the area of multi-agent systems (MAS) and data mining (DM) that are related to our research questions. We then identify research opportunities. In Chapter 3, we present DMMAS: a distributed agent mining prototype. In Chapter 4, we will look at the DMMAS implementation and evaluate the system with the experiments. We report the experiment and present the results of our analysis in Chapter 5. Chapter 6 wraps up the research with the conclusion and suggestions for future works.

Chapter 2

DATA MINING AND MULTI-AGENTS SYSTEM: A REVIEW

2.1. Introduction

Research in Data Mining (DM) integrates techniques from several fields, including machine learning (ML), statistics, pattern recognition, artificial intelligence (AI) and database systems, for the analysis of large volumes of data. DM is applied in many fields such as medical science, natural science and finance. Within the computer science domain, the data mining process expands to other research areas, such as distributed computing, parallel computing and multi-agent systems. This chapter aims to draw together previous relevant works and recent research results, with a view towards developing a methodology for the generation of better classification and prediction systems on large datasets. We evaluate some popular Multi-agents System (MAS) toolkits that are being used in the research community as well as in industry. We conclude the chapter by identifying some gaps in existing knowledge and subsequent research opportunities.

2.2. Classification in Data Mining

Data mining is a subset in Knowledge Discovery (KD) (Han, Kamber & Pei 2006). Knowledge discovery is a complex process aiming to extract previously unknown and implicit knowledge from large datasets (Fayyad 1996). It allows new information to be derived from a combination of previous knowledge and relevant data (Goebel & Gruenwald 1999). DM algorithms fall in the following typical task categories: *classification*, *clustering* and *association rules* (Larose et al. 2005).

Classification, as defined by Han & Kamber (Han, Kamber & Pei 2006), is used to predict a class label of an instance from a set of predefined labels, where the labels are nominal or categorical. Classification is called regression when the class labels are continuous values (Han & Kamber 2001). For example, classification is used when a bank wants to determine whether a credit card application is a good or bad credit risk. Regression could be used to predict the fluctuation of a stock market based on historic data. Classification uses supervised learning. Supervised learning is when the method operates under supervision by being provided with the actual outcome for each of the training examples (Witten & Frank 2005). On the other hand, techniques that analyse data without consulting known class labels are referred to as unsupervised learning (Witten & Frank 2005).

For classification, if the classifier correctly predicts the class of an instance, then it is counted as success, otherwise it is counted as error (Witten & Frank 2005). The error rate is the proportion of errors made over a whole set of instances and it measures the overall performance of the classifier. In order to predict the performance of a classifier on new data, we need to assess its error rate based on instances that were not used to train the classifier. This is achieved by partitioning the dataset into two subsets. One subset is used for training and the other one is for testing the classifier. The training set usually consists of 66.6% (two third) of the original dataset and the test set set takes up the remaining 33.3% (one third) of the original dataset. This is referred to as hold out method (Witten & Frank 2005). Without splitting the dataset into training set and testing set, our error rate is called resubstitution error. This means it is calculated by resubstituting the training instances into a classifier that was constructed from them.

There are three steps in the classification process. The first step is to build a model based on a set of training data where the attribute of interest is known. In the second step, after the model is constructed, it can be used to predict the test data. The test data outcome is known, however the model has not previously seen the test data. In the third step, typically in a production environment, the model is used to predict the attribute of interest for unseen data. The model's performance on test data usually provides a good estimate of how well it will perform on new data (Poncelet, Masseglia & Teisseire 2007).

There are a number of different classification algorithms, such as decision tree algorithms and Naïve Bayesian algorithms. Wu et al. (2007) did a comprehensive survey of the most ten influential algorithms for data mining. In this survey, for classification algorithms, only CART (Classification and Regression Trees) and C4.5 (Quinlan 1993) made it in the list. These algorithms are good candidates to consider for our base classifier later.

2.3. Classification Algorithms

How large is a large dataset? There is no definitive answer to this question, as it depends on the context and the problem being investigated. Han and Kamber (2001) suggest that datasets with large training sets of millions of samples are common (Han & Kamber 2001). In the scope of this research, we define large datasets as a dataset with a minimum of 1 million instances. In discussions about mining large datasets, two critical dimensions are often mentioned: space and time (Witten & Frank 2005). The algorithm should return the result as quickly as possible while still scaling well to large datasets. A limitation with a number of algorithms is that they require the dataset to reside in main memory; which is not feasible for large datasets. In addition, the enormity and complexity of the data makes the data mining model construction process very computationally expensive (Alsabti, Ranka & Singh 1998). Despite the availability of state-of-the-art data mining algorithms and techniques, mining large databases is still an active research area, particularly in terms of performance (Han & Kamber 2001).

There are several approaches attempt to address this challenge, such as distributed DM, parallel DM and incremental mining. Each approach has a variety of potential algorithms (Han & Kamber 2001). This section discusses some of these algorithms.

2.3.1. Decision Tree

Decision trees describe a dataset in a tree-like structure. It was first developed by Quinlan (Quinlan 1986) and is well known for solving classification problems. Decision trees are popular for several reasons. They are simple, easy to understand (Negnevitsky 2002), can be constructed relatively fast and with better accuracy when compared to other classification methods (Mehta, Agrawal & Rissanen 1996; Taniar & Rahayu 2002).

Decision trees consist of nodes, non leaf nodes and branches. The non-leaf nodes represent the attributes and leaf nodes represent the values of the attribute to be classified. Decision tree algorithms such as ID3 (Quinlan 1986), C4.5 (Quinlan 1993) and CART (Breiman et al. 1984) are constructed in 2 phases: tree building and tree pruning. Tree building is a greedy recursive process. The tree construction starts by selecting an attribute and setting that attribute as a root node. A branch is created for each possible value of the selected attribute. The process is repeated for each branch until all instances of a node of that branch have the same classification (Witten & Frank 2005). A key step in the decision tree construction process is the selection of which attribute to split. Each algorithm has a different way of determining the split. Some popular splitting criteria include information gained or entropy based (ID3) and Gini Index (CART) (Quinlan 1986).

Breiman et al. (1984) suggest that tree complexity has a crucial effect on its accuracy; therefore, it is preferred to keep the decision tree as simple as possible. Tree complexity is controlled by the stopping criteria and the pruning method used. Pruning methods can be broadly classified as post pruning or pre-pruning methods. Pre-pruning takes the preventive approach. It tries not to develop a branch where possible. Post pruning on the other hand tries to compact a fully completed tree. There are two forms of post pruning; subtree replacement and subtree raising. Subtree replacement replaces a subtree with a single leaf, which reduces the size of the tree. Subtree raising pulls one of the children nodes up to replace the parent node (Witten & Frank 2005).

Decision trees are criticised for their lack of scalability; despite of their popularity. Han and Kamber (2001) remark that the efficiency of existing decision trees are well established for relatively small datasets. But they are unable to deal with large real world datasets with millions of instances and possibly hundreds of attributes (Han & Kamber 2001). The reason for this limitation is that decision trees require the training samples to reside in main memory.

2.3.2. SLIQ

SLIQ proposed by Mehta et al.in 1996. SLIQ is a decision tree classifier that can handle both numeric and categorical attributes. It pre-sorts the data in the tree construction phase to reduce the cost of evaluating numeric attributes. SLIQ generates 2 lists; a disk resident attribute-list and a memory resident class-list. For each attribute of the training data, SLIQ generates an attribute list, indexed by a record identifier. Each tuple is represented by a linkage of one entry from each attribute list to an entry in the class list which is in turn linked to a leaf node in the decision tree (Han & Kamber 2001; Mehta, Agrawal & Rissanen 1996).

SLIQ’s tree-pruning algorithm is inexpensive; producing a compact and accurate tree. SLIQ can scale to large datasets with large numbers of classes, attributes and data (Mehta, Agrawal & Rissanen 1996). SLIQ’s performance decreases, however, when faced with a dataset with hundreds of attributes and a class list that will not fit in main memory. SLIQ is also limited by the use of its memory resident data structure (Han & Kamber 2001).

2.3.3. SPRINT

Announced in the same year as SLIQ (1996), SPRINT is superior to SLIQ in terms of computational efficiency (Shafer, Agrawal & Mehta 1996). SPRINT is designed to be easily parallelised. SPRINT can handle categorical and continuous valued attributes. SPRINT is criticised for keeping hash tables in memory when sorting along each attribute, and splitting the sorted attribute lists at every level (Han & Kamber 2001). The algorithm has to perform multiple passes over the entire dataset which creates a bottleneck for datasets that are larger than the available memory.

2.3.4. CLOUDS

CLOUDS is a decision tree algorithm that aims to address the loss of information introduced by using discritisation and sampling techniques (Alsabti, Ranka & Singh 1998). CLOUDS samples the splitting point for the numeric attributes, followed by an estimation to narrow the search of the best split. Sampling and splitting point and sampling splitting point with estimation are two methods used in determining splitting points in CLOUDS.

The attributes are sorted beforehand. Each attribute is then tested for the best splitting point. This results in several passes on the data. The sampling and splitting point divides the range of each numeric attribute into intervals such that each interval contains approximately the same number of instances. Split points are calculated at interval boundaries, reducing the number of calculations. The sampling splitting point with estimation improves upon sampling and splitting point by pruning unlikely candidates for split points (Alsabti, Ranka & Singh 1998). CLOUDS produce a slightly less accurate decision tree than SPRINT, as splitting points are determined by estimation.

2.3.5. Meta Decision Tree

Meta Decision Trees (MDT) proposed by Todorowvski and Dzeroski (2001), combine multiple models of learnt decision trees. Instead of giving a prediction like conventional models, MDT leaves specify which model should be used to obtain a prediction. MDT's algorithm is based on the C4.5 algorithm for learning ordinary decision trees. MDTs combine models better than voting and stacking. In addition, MDTs are much more concise than normal trees used for stacking and are thus a step towards comprehensible combinations of multiple models (Todorovski & Džeroski 2000).

2.3.6. Ensemble Learning

Popular methods for ensemble learning are stacking, bagging and boosting. This section does not consider stacking as it is less widely used (Witten & Frank 2005).

2.3.6.1. Bagging

Bagging is a shorthand notation for b*ootstrap aggregating.* Bagging employs bootstrap sampling with replacement on the training data. Each generated set of training data is used to build a learning model.

In bagging the models receive equal weight. The way bagging works is that for each test instance, each generated model takes a vote. If one class of the test instance receives more votes from the voting models than others, it is taken as the correct one. Classification made by voting becomes more reliable as more votes are taken into account. When a new training set is added, and a new model is built and takes part in the vote, the new results are generally more accurate. The combined classifier from multiple models in bagging often performs significantly better than a single classifier model and is never substantially worse.

Bagging is suitable for unstable learning methods such as decision trees. Small changes in the input data for decision trees can lead to quite different classifiers (Witten & Frank 2005).

Table 2.1 provides pseudo code for the bagging algorithm. Each classifier is trained on a sample of instances taken with replacement from the training set. Usually each sample size is equal to the size of the original training set.

```
Input: I (an inducer), T (the number of iterations), S (the training set), N
  (the subsample size).
Output: Ct; t = 1,…, T
1: t ← 1
2: repeat
3:   St ← Sample N instances from S with replacement.
4:   Build classifier Ct using I on St
5:   t + +
6: until t > T
```

Table 2.1 Bagging Algorithm (Breiman 1996)

As sampling with replacement is used, some of the original instances of *S* may appear more than once in *St* and some may not be included at all. So the training sets *St* are different from each other, but they are not independent. To classify a new instance, each classifier returns the class prediction for the unknown instance. The composite bagged classifier, I*, returns the class that has been predicted most often (voting method). The result is that bagging produces a combined model that often performs better than the single model built from the original data. Breiman (1996) notes that this is true especially for unstable inducers because bagging can eliminate their instability. In this context, an inducer is considered unstable if perturbing the learning set can cause significant changes in the constructed classifier. However, the bagging method is rather hard to analyse and it is not easy to understand by intuition the factors and reasons behind the improved decisions.

2.3.6.2. Boosting

Boosting is similar to bagging, however the key difference in boosting is that weighting is used to give more influence to the more successful models. Boosting models are built iteratively where as bagging models are built separately. In boosting, each new model is influenced by the performance of those built previously. A new model in boosting takes into account the instances that were handled incorrectly by the previous models. Boosting weights a model's contribution by its performance rather than treating every model with an equal weight as in bagging (Witten & Frank 2005).

Wang et al. (2003) apply boosting in a weighted-classifier ensemble algorithm. Classifier weights depend on the data distribution of the windows used to train them, so that the classifiers built from data having a distribution similar to the current distribution are assigned higher weights. Clearly, this addresses the concept-drift problem, as the classifiers representing the current and recent distributions are more heavily favoured.

Each classifier is given a weight that is inversely proportional to its expected classification error using the mean-square error measure. The expected error is approximated by comparing the new classifier results to the results of the previous classifier. After assigning the weights, only the best *k* classifiers are considered for the testing phase, and their results are combined using weighted averaging.

2.4. Multi-agents System

The term 'agent', or software agent, has been gaining popularity and is used within the literature of a number of technologies including artificial intelligence, databases, operating systems and computer networks. Although there is no single definition of an agent (see, for example, Genesereth and Ketchpel, 1994; Wooldridge and Jennings, 1995; Russell and Norvig, 2003), general consensus agrees that an agent is essentially a special software component that has autonomy, provides an interoperable interface to an arbitrary system or behaves like a human agent. An agent can automatically figure out what to do in order to satisfy its design objectives, rather than having to be told explicitly what to do at any given moment (Wooldridge 2002). An agent is autonomous, because it operates without the direct intervention of humans or others and has control over its actions and internal state. An agent is social, because it cooperates with humans or other agents in order to achieve its tasks. An agent is reactive, because it perceives its environment and responds in a timely fashion to changes that occur in the environment. An agent is proactive, because it does not simply act in response to its environment but is able to exhibit goal-directed behaviour by taking the initiative. Moreover, if necessary an agent can be mobile, with the ability to travel between different nodes in a computer network. It can be truthful, providing the certainty that it will not deliberately communicate false information. It can be benevolent, always trying to perform what is asked of it. It can be rational, always acting in order to achieve its goals and never to prevent its goals being achieved. Finally, it can learn, adapting itself to fit its environment and the desires of its users (Wooldridge 2002).

Multiagent systems consist of a network of problem solvers which are modelled in agents that work together to solve problems which are beyond one agent's capability (Durfee & Montgomery 1989). MAS provides a powerful abstraction that can be used to model systems where multiple entities exhibiting self-directed behaviours must coexist within an environment, and achieve the system-wide objective of that environment (Sugumaran 2008). MAS can be used to model complex systems. Agents in MAS may interact with each other both indirectly (by acting on the environment and event) or directly (via direct communication and negotiation). Agents may decide to cooperate for mutual benefits or may compete to serve their own interests. Agents typically interact and collaborate with one another by exchanging messages through computer network infrastructure. Generally, each agent will be representing or acting on behalf of a user with individual goals and motivations. In order to successfully interact, agents need to co-operate, coordinate and negotiate with each other just like human beings (Wooldridge 2002).

2.3.7. MAS Motivation

According to Luck et al (2007), MAS's motivation is to automate and improve existing tasks, to anticipate desired action on human behalf, to undertake them, while enable human to retain as much control as required. In recent years agent-based systems, particularly as implemented in a distributed framework, have attracted considerable interest. Jennings and

Wooldridge (1998) state that agent-based computing has been hailed as '*the next significant breakthrough in software development*' and '*the new revolution in software*'

2.3.8. Features and Capabilities

Wooldridge (2002) suggests that an intelligent agent should have the following capabilities in order to satisfy their design objectives: *reactivity, pro-activeness* and *social ability*. *Reactivity* means agents are able to perceive their environment, and respond in a timely fashion to changes that occur in it. *Pro-activeness* assumes agents are able to exhibit goal-directed behaviour by taking the initiative. *Social ability* defines agents are capable of interacting with other agents (Wooldridge 2002).

Durfee, Lesser and Corkill (1989) stress the importance of collaboration in MAS, as no single agent has sufficient expertise, resources, and information to solve the problem. Different agents might have expertise for solving different parts of the problem. Each problem-solving agent in the network is capable of sophisticated problem-solving and can work independently, but the problems faced by the agents cannot be completed without cooperation (Durfee, Lesser & Corkill 1989).

2.3.9. Multi-agents Toolkits

Agent technology is considered one of the most innovative technologies for the development of distributed software systems (Baik, Bala & Cho 2004). Although it is not yet a mainstream approach in software engineering, much research has been done, and many applications have been developed and presented. Some software applications originally developed for research purposes are finding their way into industry; for example JADE, Agent Builder, FIPA-OS, Jack and ZEUS. In this chapter we will review the most popular toolkits JADE, JACK and MASDK. Reviewing all of MAS toolkits is out of the scope of this research. A more complete list of more than 100 agent toolkits can be found in (Hamburg 2009) and at Agentlink.com.

Agent toolkits are software containing tools for deploying an agent infrastructure and for aiding in the development of agent application. There are many multi-agent toolkits available in different languages, such as Java, C++ and .NET (Sharma 2005). This section reviews popular MAS toolkits which conform to the Foundation for Intelligent Physical Agents (FIPA) specification.

2.3.9.1. JADE

Java Agent Development Framework (JADE) was developed by Bellifemine, Poggi and Rimassa (Bellifemine, Poggi & Rimassa 2001). It is a popular well-established Java-based FIPA-compliant agent platform (Chmiel et al. 2005). JADE originated as a research project from TILAB and is well known in the research community. It is a software framework that facilitates development of interoperable intelligent MAS. It is distributed under an Open Source License. JADE is a mature product, used by both the research and industrial communities (Bellifemine et al. 2008).

Some of the main features of JADE are its portability, mobile support and support for distributed agents. JADE leverages the portability that the underlying Java virtual machine offers. JADE agents in one operating system can clone themselves and migrate to other operating systems at runtime. Agents communicate with each other through message exchange. JADE agents can also be distributed to several machines. Each agent runs in its own thread -potentially on different remote machines - while still being able to communicate with each other. JADE also has a lighter set of API functions called LEAP, which is designed for agents running on mobile devices. LEAP is considered as one of

JADE’s leading distinguishing features. Another useful feature of JADE is its subscription mechanism. JADE’s subscription mechanism allows agents and external applications to subscribe to notifications of platform events.

A JADE platform contains one container at the minimum, of which one must be a MainContainer. One JADE container can host zero to many agents (see Figure 2.0.1). The following figure illustrates the relationships between AgentPlatform, Container, and Agent (Bellifemine, Caire & Greenwood 2007).

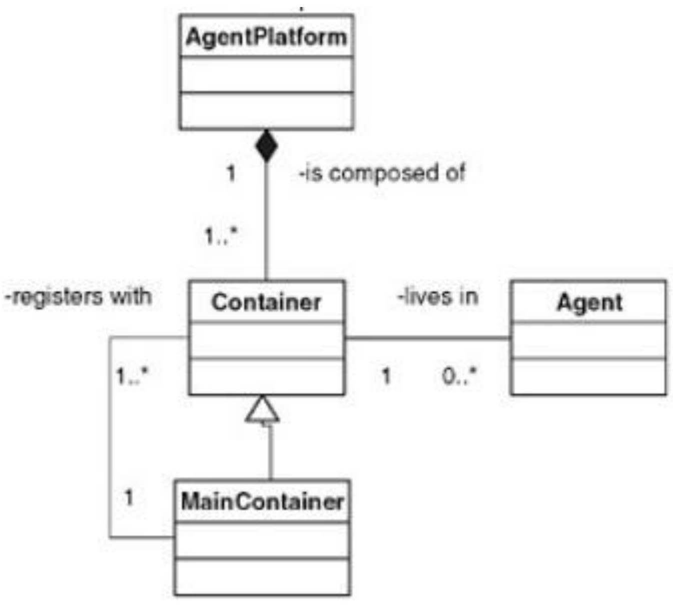


Figure 2.1 Platform, Container and Agent Relationship (Bellifemine, et al., 2007)

The MainContainer is different to a Container in that MainContainer is the first container to start up and perform initialisation tasks. MainContainer manages the container table, global agent descriptor table and hosts two main platform agents: Agent Management Service agent and Directory Facilitator agent. The Agent Management Service is the core agent that keeps track of all JADE programs and agents in the system. Besides providing the white pages service as specified by FIPA, it also plays the role of authority in the platform. The Directory Facility is a yellow pages service, where agents can publish their services.

2.3.9.2. JACK

JACK is developed by the Agent Oriented Software (AOS) Group in Melbourne, Australia. JACK's design philosophy is to be an extension of object-oriented development. As a result, JACK agents are written in the JACK agent language, an extension to the Java programming language. JACK agent language implements a beliefs-desires-intentions (BDI) architecture whose design is root in philosophy (Bellifemine, Caire & Greenwood 2007).

JACK programs are compiled to normal Java source files with a precompiler. These then can subsequently be translated to Java classes using the normal Java compiler.

2.3.9.3. MASDK

MASDK is an acronym for Multi-agent Software Development Kit. It implements the Gaia methodology and is written in C++. It is a fully Integrated Development Environment (IDE). MASDK aims to become a MAS tool capable of supporting the complete life cycle of industrial MAS comprising analysis, design, implementation, deployment and maintenance which others like AgentBuilder, JADE, ZEUS, FIPA-OS and agent tool do not support (Gorodetski et al. 2005).

2.4. Why Agent Mining?

Multiagent technology complements DM for several reasons. First, MAS allows the DM task to be divided to each DM agent. The divide and conquer approach (Smith and Davis, 1981) enables the data mining task to scale to a massive distributed dataset. In addition, DM agent may be able to pick up new data mining techniques dynamically and automatically select the technique most appropriate to the mining task (Klusch, Lodi & Moro 2003).

MAS can be used to solve problems that are too large for a centralised agent to solve due to resource limitations. MAS also avoids a one point bottleneck (or failure). In addition, the agents are not self-contained, they are able to interact with outside agents e.g., buying and selling, contract negotiation, meeting scheduling (Garrido & Sycara, 1996).

MAS is good at enhancing performance in the area of computational efficiency. This is achieved through concurrency, reliability via redundancy, extensibility by changing the number and capabilities of the agents, maintainability via modularity and reuse of agents in different agent societies (Ira 2004).

MAS and DM have become two of the most prominent, dynamic and exciting research areas in information science in the last 20 years (Cao, Gorodetsky & Mitkas 2009). Both MAS and DM face challenges that can be alleviated by leveraging other technologies. The DM processes such as data selection and data pre-processing can be enhanced by using agent technology (Klusch, Lodi and Moro 2003). Klusch et al argue that agent technology is best able to cope with these processes in terms of autonomy, interaction, dynamic election and gathering, scalability, multi strategy, and collaboration (Klusch, Lodi and Moro 2003).

The most important features that agent technology brings to data mining include; decentralised control, robustness, simple extendibility, sharing of knowledge, sharing of resources, process automation, data and task matching and result evaluation.

Decentralised control is arguably the most significant feature of MAS. This feature implies that individual agents operate in an autonomous manner and are self deterministic.

Robustness is an important feature, allowing the system to continue to operate, even though the agents may have crashed or died. Simple extendibility is achieved by adding more agents to the framework.

There has been increasing interest in multi-agent and data mining research, as evidenced by the increasing number of research projects in these areas in the last few years. The following section looks at some projects from the research community.

2.4.1. Applications

CoLe is a cooperative data mining approach for discovering knowledge. It employs a number of different data mining algorithms, and combines the results to enhance the mined knowledge. A multi-agent framework is used to run these multiple data mining algorithms simultaneously and cooperatively. The agents communicate their results, which are combined into hybrid knowledge. Gao, Denzinger and James (2005) claimed that the results achieved by CoLe were efficient and promising. Unlike our approach which emphasizes on incremental mining and large dataset, CoLe concentrates more on getting hybrid knowledge that cannot be generated by a single data mining algorithm.

Tian and Tianfield, (2003) introduced a multi-agent approach to the design of an E-medicine system. They implemented a MAS prototype for assistance in delivering telemedicine services for the care of diabetes sufferers. The medical services include monitoring the patient in real time and transmitting the information to a physician, providing the patient with the relevant therapy, and responding to patient enquiries. These services are implemented by a monitoring agent, data processing agent, diagnosis agent, therapy agent, consultation agent, training agent, archival agent, department agent and interface agent, respectively (Tian & Tianfield 2003).

Agent Academy (AA) is an integrated development platform that allows developers to create agent-based applications (Mitkas et al. 2002). AA aims to fill a gap in the market for integrated high level abstraction tools for the design and development of agent based applications. AA is implemented as a MAS, running on JADE for the multi agent part and Weka for the data mining part. AA focuses on MAS developers rather than addressing any particular domain problem.

JAM (Java Agents for Meta-learning) is a meta-learning agent based system proposed by Stoflo & Chan (1993). It is a general technique that combines the results of multiple learning algorithms, each applied to a set of training data (Chan & Stolfo 1993). It deals with the problem of computing a global classifier from large and possibly distributed databases. JAM aims to compute a number of independent classifiers by applying learning algorithms to a collection of independent and distributed databases in parallel. The main classifiers are then collected and combined by another learning process. The meta-learning seeks to compute a meta classifier that integrates the separately-learned classifiers to boost overall predictive accuracy (Chan & Stolfo 1993; Prodromidis, Chan & Stolfo 2000). Although this approach reduces the running time significantly, Chan & Stolfo (1993) and Stolfo et al. (1997), report that the JAM meta classifier did not achieve the accuracy of a single classifier in the batch learning mode, using all the available data.

2.5. Summary

Although, the research community has attempted a number of solutions to solve the scalability problem, there are still several gaps that can be addressed. Current approaches for solving classification tasks do not tend to perform well with large dataset. In addition, literature search revealed that little research had been conducted into MAS and DM integration. Multi-agent systems are a powerful way to model and build a complex system.

Data mining tasks are generally complex, and as such MAS are an appropriate tool. The interaction and integration between MAS and DM has the potential to not only strengthen one another, but open up new techniques for developing more powerful intelligence and intelligent information systems across different domains.

This chapter has reviewed state of the art agent toolkits, common approaches to mining large datasets and some agent mining applications. We also identified that, as an important operational problem, large dataset mining is made more challenging by increasingly distributed information and computing resources. It becomes impractical to process these distributed and dynamic information resources. Centralised systems will still be useful in processing archived data collections, however, distributed systems are needed to deal with highly distributed and dynamic online information.

If we can create a successful multi-agents data mining system for one domain, it is likely to be useful to apply in other domains.

Chapter 3

THE RESEARCH PROBLEM AND THE PROPOSED DMMAS SOLUTION

3.1. Introduction

This chapter describes an approach that leverages the capabilities of multi agents applied to data mining tasks. We look, firstly, at large datasets and present a strategy for transforming traditional batch learning algorithms into distributed agent learning algorithms. Next we will setup an experiment to do a comparative study between batch mining and multiagent mining on a large dataset with two dimensions; time and space.

3.2. The Problem

The problem that the research attempts to address is the data mining scalability challenges. Databases have been growing in size, and more of them are distributed among geographical locations. There are existing algorithms such as C4.5 requires all training examples to reside in main memory. The large size of training data can make it impossible for the algorithm to run. This problem can be explored by looking at its main characteristics.

3.3. The Problem Characteristics

Dealing with large dataset has some unique challenges. One of the training large dataset challenge is how do we handle the large training data. Does the algorithm load all the data to main memory or the training data at once or the data can be read incrementally?

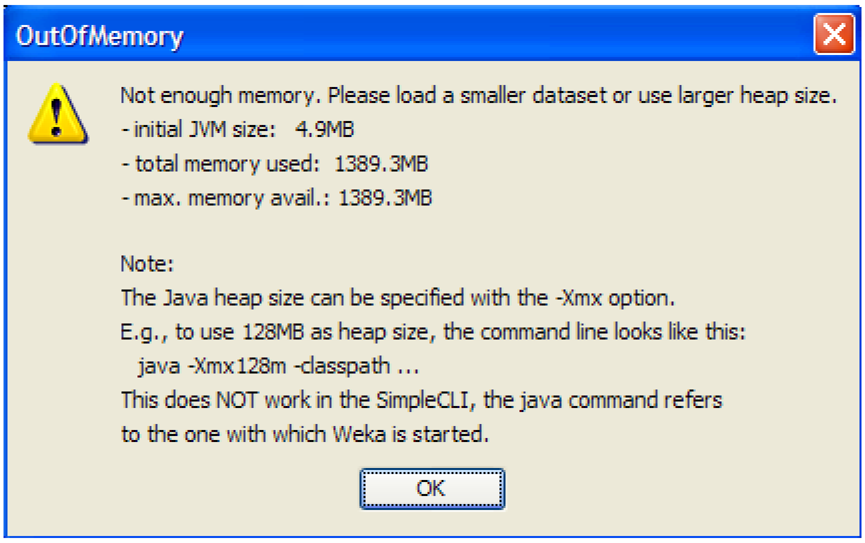
Another characteristic is the communication cost. If the dataset is geographically distributed, accessing data over this distributed geographically will be expensive and not optimal. This means that the communication cost needs to be kept as low as possible. The following section discuses the problem in more details.

3.4. Analysis of the Problem

The main focus of this research is to improve efficiency on a large dataset classification in a cooperative multi-agent environment.

To deal with large datasets, works such as SPRINT try to utilize parallelism by distributing the workload among different processors, while CLOUDS tries to exploit properties of an algorithm for better efficiency.

As mentioned in the previous chapter, some data mining algorithms such as ID3 and C4.5 have to load the entire dataset into memory before they can process and analyse the data. DMMAS can be considered where the processing node does not have enough primary memory to load in the entire dataset. Typically, if a program does not have enough memory it will simply crash. If the error is handled gracefully, it will throw an exception such as the one in Figure 3.1



```
Not enough memory. Please load a smaller dataset or use larger heap size.
- initial JVM size:   4.9MB
- total memory used:  1389.3MB
- max. memory avail.: 1389.3MB

Note:
The Java heap size can be specified with the -Xmx option.
E.g., to use 128MB as heap size, the command line looks like this:
   java -Xmx128m -classpath ...
This does NOT work in the SimpleCLI, the java command refers
to the one with which Weka is started.
```

Figure 3.1 Out of memory thrown by Weka

Another challenge that we aim to address with DMMAS is the stale training model problem. In batch learning mode, the algorithm builds model from a set of training dataset and classifies unknown instance(s) base on the trained model. In most circumstances, the dataset will evolve with new data or existing data is removed. The limitation with the batch approach is whenever new training instances are available; the database will have to be taken offline for the data mining task to execute. This means that users will not be able to access the dataset while this exercise is being performed.

In order to address the scalability and memory limitation, we will take a look at the algorithm in the next section.

3.5. Algorithm

The algorithm used to approach this problem is based on the divide and conquer technique. The idea is to divide the dataset to multiple partitions; each partition is allocated to a data mining agent. The data mining agent is trained with the allocated data by creating a model based on the data partition. When there is a request for classification, the request is broadcast to all agents. Each agent would handle its own classification task and return the class it thinks best match the request. The classification responses from the agents are aggregated and returned to the requester.

The algorithm makes the following assumption regarding “agents” and “multi-agent environment”

Agents

- Agents are independent of each other.
- Each agent has its own dataset, from which an initial classifier is learned.
- Agents can learn using a different algorithm (C4.5 as a default algorithm)
- Agents can communicate.
- Agents are working together cooperatively.

Multi-agent Environment

- Multiple agents exist in an environment.
- Agents are interconnected; an agent is able to send messages to all other agents.

In addition to the above assumption, it is important to define some basic notions that are important in the multi-agent paradigm, particularly in relation to DMMAS. The first important basic notion and is the agent’s goal. An agent can have one or more goal. Goals constitute the motivational attitude of rational agents and they form the key concept in explaining and generating their pro-active behaviour. Pursuing multiple goals simultaneously might pose problems for agents as the plans for achieving them may conflict (Dastani et al). In DMMAS implementation, an agent’s goal can have status of achieved, no longer desired, waiting or unachievable. These goal statuses are mutually exclusive, which mean an agent can only have one status at anyone time.

An agent’s belief represents the informational state of the agent, in other words its beliefs about the world around it. Beliefs can also include inference rules, allowing forward chaining to lead to new beliefs. The term belief is used rather than knowledge recognizes that what an agent believes may not necessarily be true and may change in the future (Woolridge, 2002).

An agent’s plans are sequences of actions that an agent can perform to achieve one or more of its intentions. Plans may include other plans. A data mining agent plan to extract rules from data can be that: find the dataset, run one or more algorithms, extract rules from each result.

Another important notion that triggers agent’s actions is event. An event may update beliefs, trigger plans or modify goals. Events may be generated externally and received by sensors or integrated systems.

More discussion on the algorithm is in the next chapter (Chapter 4). In the next section, we will look at how we use the algorithm in the design of the system.

3.6. System Architecture

Our solution utilises an agent oriented software engineering approach, as it is an attractive approach for implementing modular and extensible distributed computing systems (Jennings 2001). We employed the Prometheus methodology developed by Winikoff and Padgham, (2004) to design the DMMAS. This methodology prescribes the elements necessary for the development of a software system. It has two important components: a description of the process elements of the approach, and a description of the deliverable products and their documentation. Bresciani et al (2004) suggests that using agent oriented (AO) methodologies such as Prometheus may be beneficial even if the implementation is not in an AO language but, for example uses object-oriented design and programming.

There are five main components in the DMMAS: agent, behaviour, performance counter, data source and configuration (Figure 3.2).

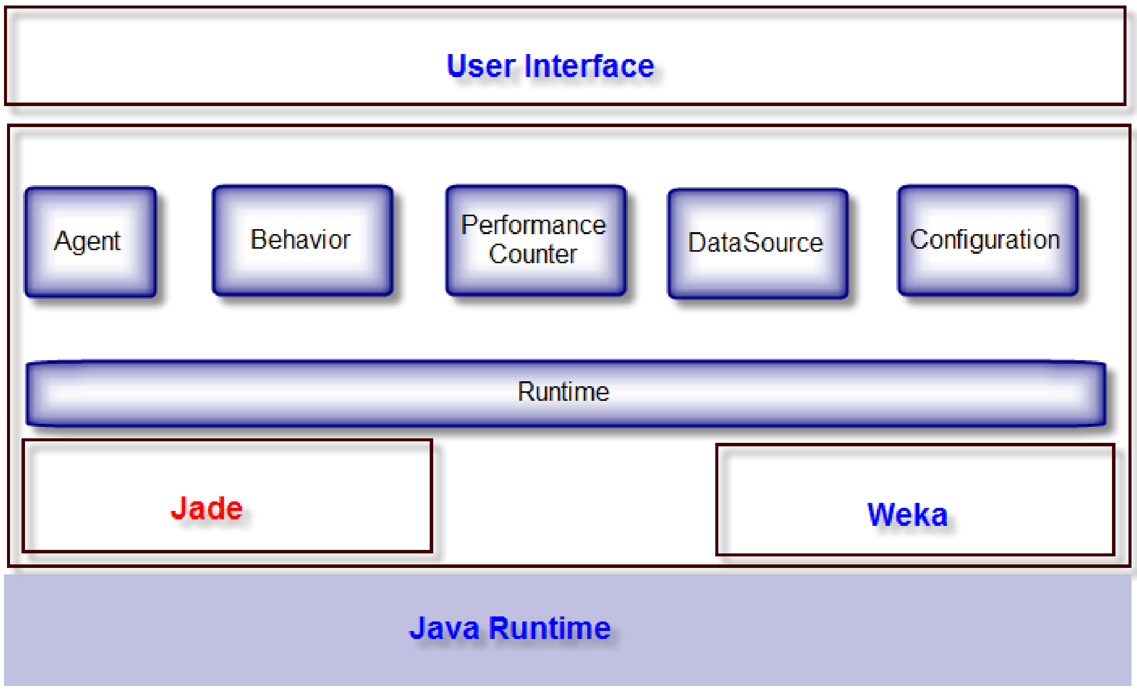


Figure 3.2 System Architecture Overview

Agent component is a group of all the agent types in DMMAS such as an AgentManager agent, DataMiningAgent agent, DataManager agent and DataGenerator agent. The agents in this package can be inherited and their behaviour can be overridden.

Behaviour package contains different behaviours of an agent. An agent can have different behaviours by having the behaviour applied to it. For example, an agent can have a classification behaviour which implements the decision tree algorithm. In addition, a new behaviour can be added by inheriting from the behaviour base class.

Performance Counter components are responsible for measuring the performance of an agent. It measures the duration of an operation. It also can measure the accuracy of a data mining task, based on given test data.

DataSource is a logical group of dataset related functionalities. A dataset in DMMAS can be in the form of files or a database. In case of a file, the type of file can range from a text file, xml file to an arff file (Weka proprietary format). The database data source can by MySQL or Microsoft SQL Server. There is also minimal support for NoSQL dataset. Support for NoSQL is very limited at this stage and is envisaged to be added in future work.

A configuration package stores configuration settings that are necessary to enable DMMAS to run successfully. Configurations such as setting the data source, the agent container host name, the data mining algorithms for each agent, and the agent partition size are looked after by the configuration package.

The runtime component is responsible for loading Weka and Jade. It also initialises the Jade agent platform ready to be used in DMMAS.

3.7. DMMAS Design

The software is implemented using the Java programming language to leverage JADE and WEKA. The learning algorithm would be faster if it was written in a lower level language such as C. It's slower in Java because Java byte code must be translated into machine code before it can be executed (Witten & Frank 2005).

The database technology used to store the dataset is MySQL, an open source database management system. DMMAS is developed as a library that can be embedded and consumed by other applications. DMMAS uses JADE for the agent platform and WEKA as the data mining engine. DMMAS settings can be set programmatically, however a user interface has also been developed to demonstrate the capabilities of the system. Java, MySQL, JADE and WEKA were all selected as they are readily available and well supported technologies with active user bases.

WEKA is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from Java code. WEKA contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. (Wakaito 2008). The version used in DMMAS is 3.5.7.

The DMMAS code is written in Net Bean. Net Bean is a free integrated development environment (IDE).The functionalities are written using traditional java classes. These classes are then converted into agent based behaviour.

In addition to the high level architecture discussed in section 3.6, the design of the multiagent system can be examined in more details by looking at the UML diagram in Appendix E. It should be noted that there is no user interface element or class is included in the UML diagram. The diagram strictly contains the important relevant classes in DMMAS.

As can be seen from the diagram, all the agents in DMMAS inherit from the parent Agent class. This inheritance relationship allows all the sub-class agents to inherit the parent Agent behaviours and characteristics.

As should be noted in the class diagram, there should only be one agent platform at anyone time. The agent platform contains one or more containers. Each container can have more than one agent. The agent platform can be split among several hosts. Only one Java application, and therefore only one Java Virtual Machine, is executed on each host. Agents are implemented as Java threads and live within Agent Containers that provide the runtime support to the agent execution.

DMMASEnvrionment class is the façade to DMMAS. It is responsible in setting up the agent platform and initialises the first container. It also has the ability to save the configuration to file system as well as loading the configuration back when it starts.

3.8. Summary

This chapter identified the research problem and discussed proposed algorithms. It also looked at DMMAS as a system and its various components. The next chapter provides the implementation details of the DDMAS; when it can be used; and how it works with static and dynamic datasets.

Chapter 4

IMPLEMENTATION AND EXPERIMENTS

4.1. Introduction

The first section of this chapter discusses DMMAS implementation which includes data compression, agents' communication and the user interfaces (UI). The second section discusses the experiment framework to compare DMMAS approach with the traditional batch mining approach.

4.2. DMMAS Implementation

This section describes the main implementation of DMMAS. The UI controls are what the users see on the screen. In DMMAS these are the main tabs, initialisation tab and dataset tab. The internal aspects of the system such as data compression and agents communication are also discussed.

4.2.1. Platform Initialization

An agent platform contains one main container and at least one satellite container. Each container contains one or more agents. One machine can host both the main container and a satellite container. For simplicity and scalability we will assume that one machine corresponds to one container. This relationship is illustrated in Figure 4.1.

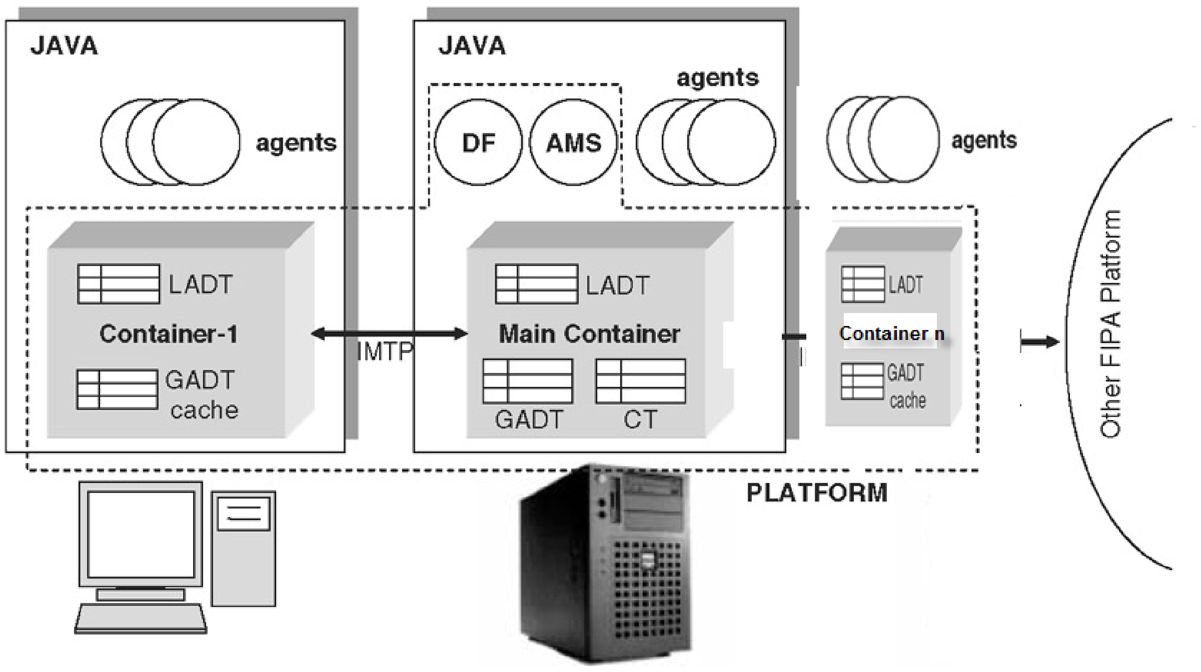
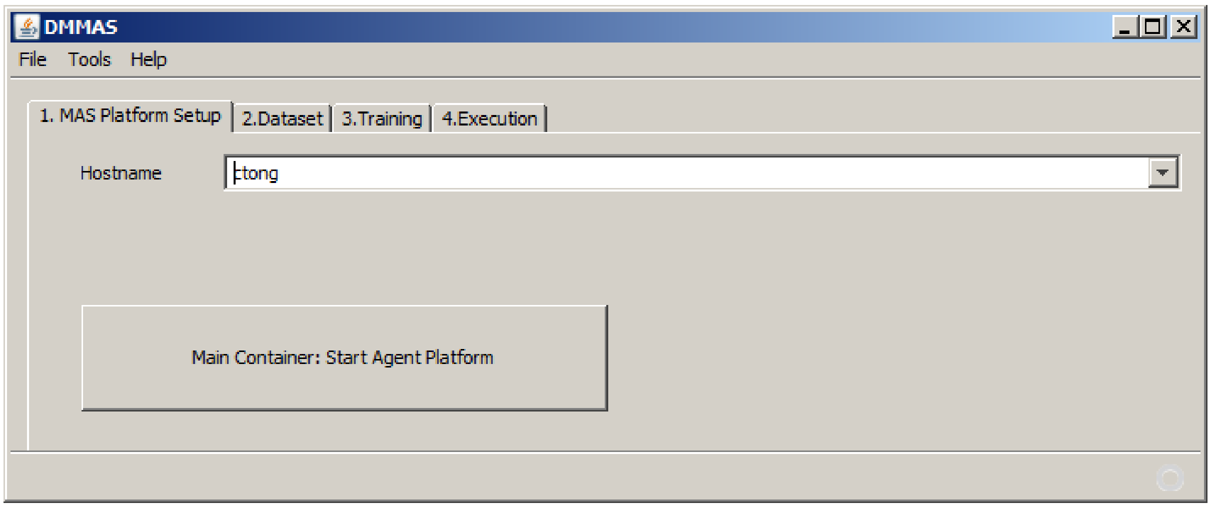


Figure 4.1 Container Relationship

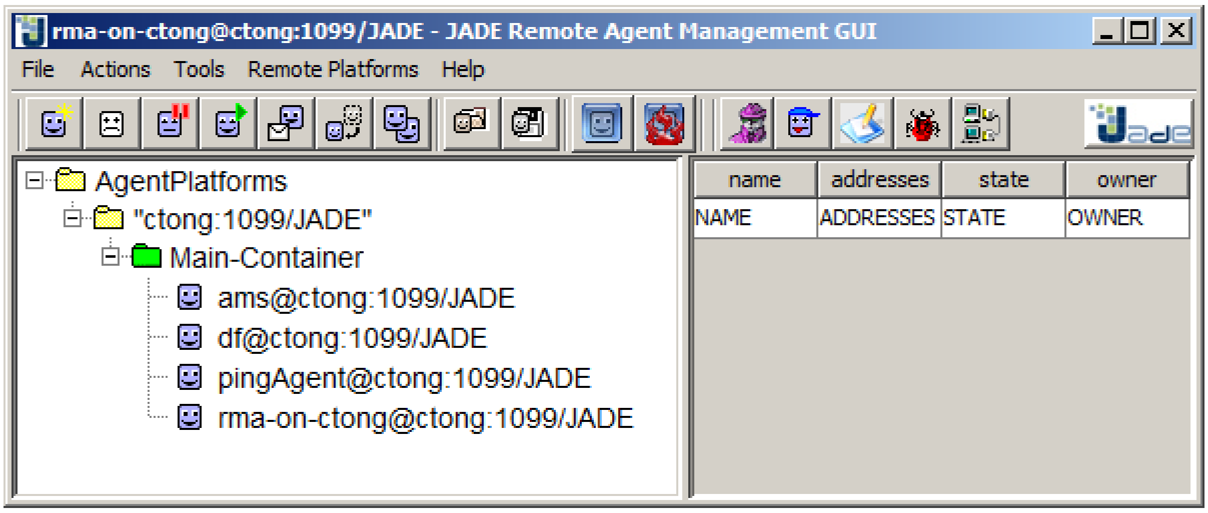
The platform initialisation is performed in the first tab "MAS Platform Setup" (see Figure 4.2 – Platform Setup).



Hostname: ttong

Figure 4.2 Platform Setup

This creates a new platform and adds a new "Main-Container" on the current machine. Note that there is only one "Main Container" in the platform. After the initialisation, the platform now contains one container: the "Main Container". The platform initialisation in a sense is like a network socket listening for connections (also see Figure 4.3). Here, the platform ‘listens’ for satellite containers wishing to join. This is a powerful mechanism that makes it extremely easy to add additional processing nodes. A new machine can just start up and join the platform anytime. Users have the option whether or not to see the Jade Agent UI when initialising the platform. It is useful to display this UI in debug mode for troubleshooting and viewing agents’ statuses.

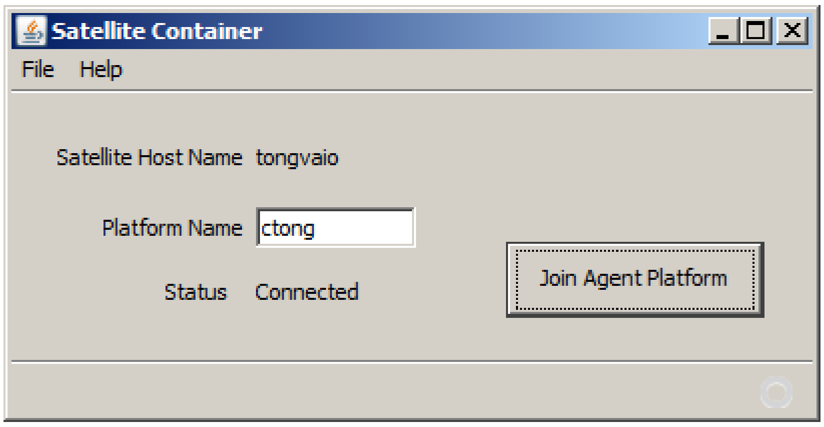

rma-on-ctong@ctong:1099/JADE - JADE Remote Agent Management GUI
File
Actions
Tools
Remote Platforms
Help
Jade

```
AgentPlatforms
  "ctong:1099/JADE"
    Main-Container
      ams@ctong:1099/JADE
      df@ctong:1099/JADE
      pingAgent@ctong:1099/JADE
      rma-on-ctong@ctong:1099/JADE
```

name	addresses	state	owner
NAME	ADDRESSES	STATE	OWNER

Figure 4.3 DMMAS Platform Ready

Now, other satellite containers can join the platform by specifying the machine name that hosts the platform. The satellite container (see Figure 4.4) must be initialised separately in another process or on another computer on the same network. It is suggested that the firewall to be turned off or set to enable traffic on default port 1099. Without having disabled the firewall or enabled port 1099, the satellite container will not be able to join the agent platform.



Satellite Host Name tongvaio

Platform Name ctong

Status Connected

Figure 4.4 Non Main Container Joining

When the satellite container successfully connects to the host machine, the platform will acknowledge the connection and update the container list. If this is the first time the container has joined the platform, there will be no agents residing in this container. The 'Main-Container' has the ability to create agents which run on the satellite container. This means that the agents that the 'Main-Container' spawned on the satellite container can leverage the satellite container resources. The satellite container pictured in figure 4.5 has no agents in it. The satellite container name is the machine name that initiated the connection, in this case 'tongvaio'.

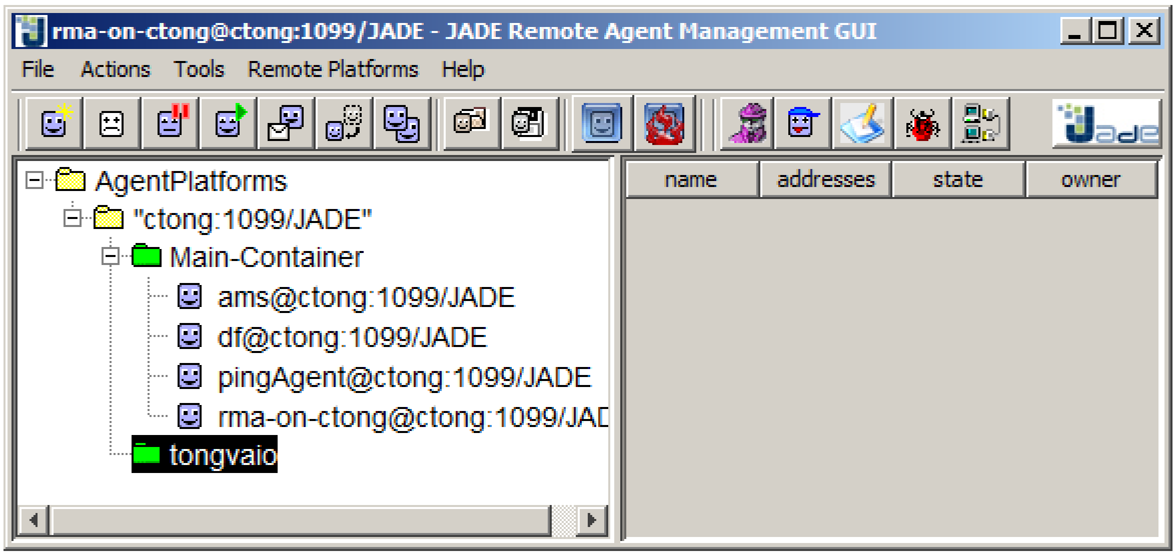
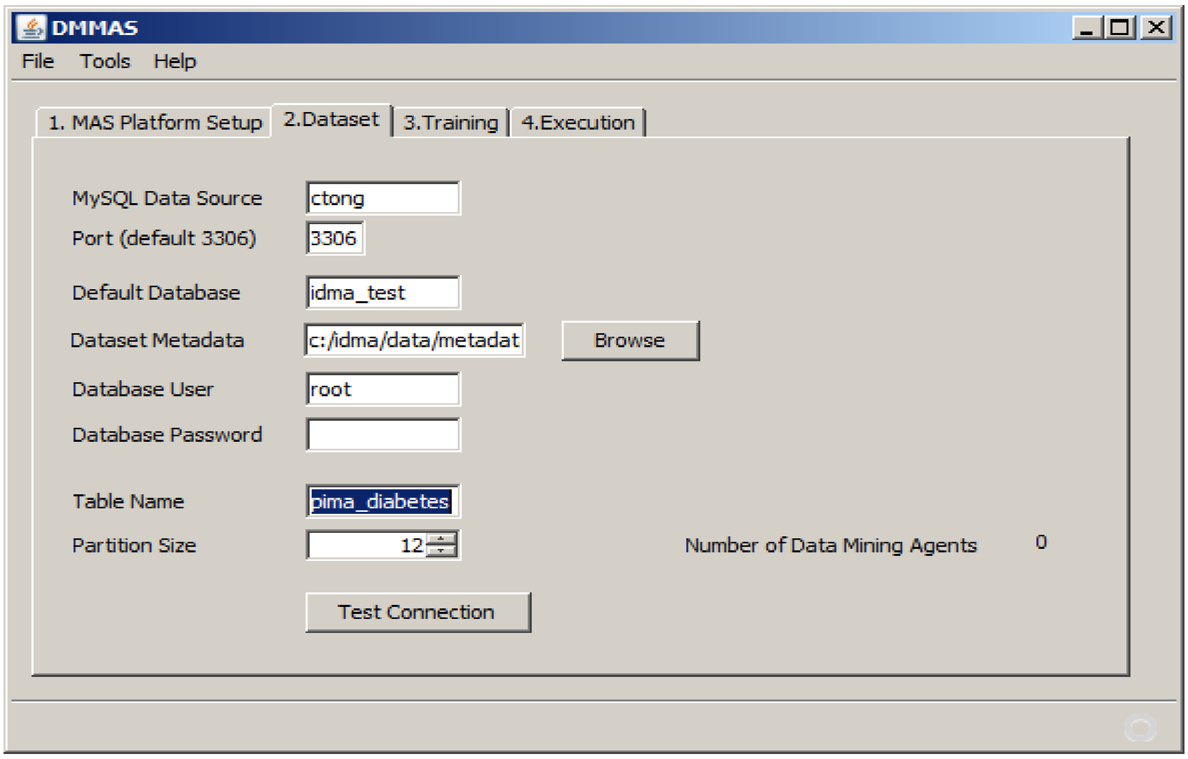


Figure 4.5 Platform is ready

In order to add additional processing nodes or non-main containers, the satellite container program can simply be run on a new machine and connects to the platform.

4.2.2. Data Source Configuration

The Dataset configuration tab (as shown in Figure 4.6) allows users to specify the data source. The data source can either be: a SQL data source or a text file data source. SQL data sources accepted are MySQL, MSSQL and PostgreSQL databases. Text file data sources accepted are Weka ARFF proprietary format and CSV files. We also considered NoSQL technologies that employ Map Reduce algorithms – such as Cassandra, MongoDB and CouchDB – due to their massive scaling capabilities. There are some basic implementations that leverage the NoSQL technologies; however they are still very basic. We will discuss this further in the future work section.



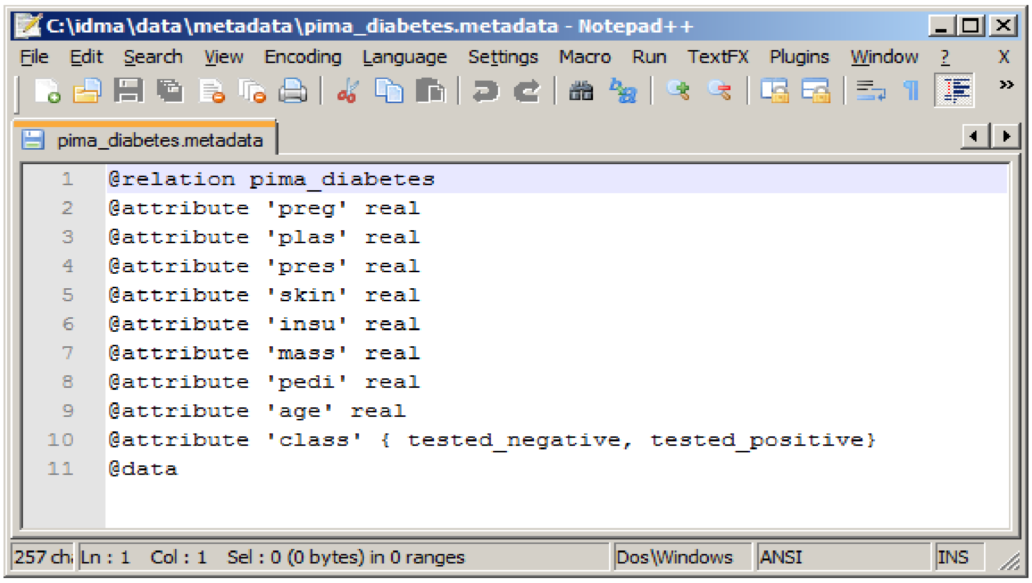
MySQL Data Source: ctong
Port (default 3306): 3306
Default Database: idma_test
Dataset Metadata: c:/idma/data/metadat
Database User: root
Database Password:
Table Name: pima_diabetes
Partition Size: 12
Number of Data Mining Agents: 0

Figure 4.6 Dataset Configuration

MySQL is the default data source used in DMMAS. In order to establish a connection to a MySQL database, there are some required properties that we need to fill in: MySQL server name, MySQL port (default to 3306), the default dataset and the database user name and password. In addition, the table name field, the dataset metadata and the partition size also need to be configured.

DMMAS requires the table name to be entered so the system knows where to get the data from. The dataset metadata is a small text file that defines the dataset properties and data type. We have adapted the Weka ARFF format for our metadata definition. Refer to Figure 4.7 for a sample metadata file. For a comprehensive explanation of the syntax in this metadata file consult Witten and Frank (2001).

The partition size field is particularly important. This field dictates how many partitions the whole dataset is to be divided into. Ideally, the number of partitions would be determined dynamically based on prior knowledge. At the moment, however, this parameter still needs to be manually configured. Dynamically determining a partition number will be discussed in the future work section.


C:\idma\data\metadata\pima_diabetes.metadata - Notepad++
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins Window ? X
pima_diabetes.metadata
257 ch Ln : 1 Col : 1 Sel : 0 (0 bytes) in 0 ranges
Dos\Windows
ANSI
INS

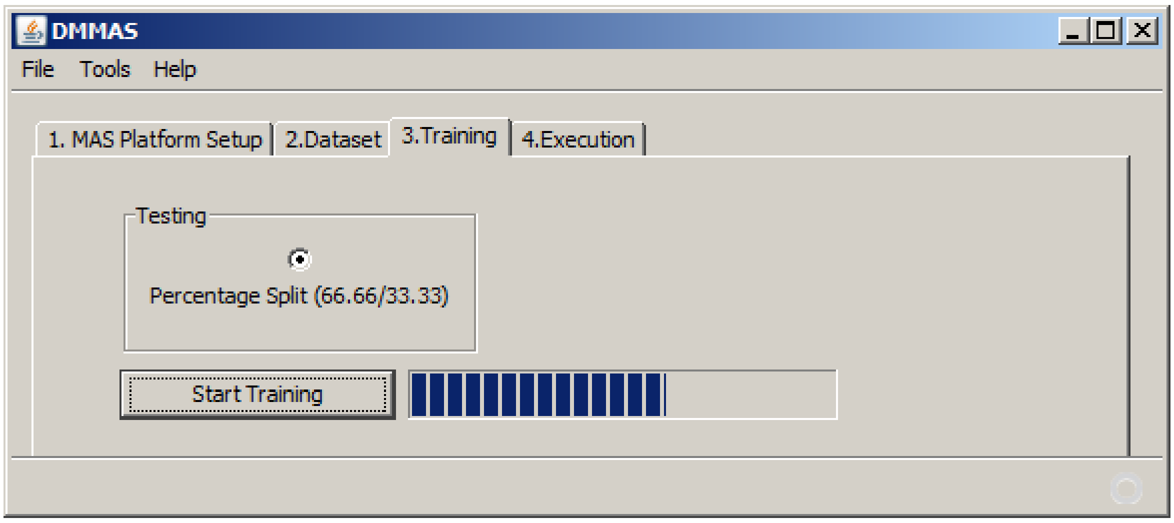
```
@relation pima_diabetes
@attribute 'preg' real
@attribute 'plas' real
@attribute 'pres' real
@attribute 'skin' real
@attribute 'insu' real
@attribute 'mass' real
@attribute 'pedi' real
@attribute 'age' real
@attribute 'class' { tested_negative, tested_positive}
@data
```

Figure 4.7 Dataset Metadata

Increasing the number of partitions / agents adversely affects efficiency, as this also increases the communication overhead. A study by Mukhopadhyay et al. (2003) shows that as the number of agents increases, the communication overhead increases almost linearly while the classification performance quickly saturates.

4.2.3. Agents Training

Agent training is configured using the Training tab (see Figure 4.8). Users have the ability to specify whether testing should be performed after the completion of training. The only testing mechanism provided at the moment is to perform a percentage split to partition the data into training and test sets. This means 66.6% of the agent's allocated partition is reserved for training and the remaining 33.3% is for testing.



Percentage Split (66.66/33.33)

Figure 4.8 Training

In order to perform the training process, it is necessary to determine how we partition the dataset. One of the challenges we faced is in determining the size of each data partition. There are two possible approaches; a static partition size determination approach and a dynamic partition size determination approach.

With a static partition size determination approach, we divide the partition into *i* partitions. The number of *i* is determined ahead of time. For instance, we can set *i* to any number in the array of { 2, 4, 6, 8 }; meaning the dataset is divided in to 2, 4 , 6 and 8 partitions respectively. This approach means that the user will need to determine the most effective partition size by trial and error. As the domain of each dataset is different, we cannot generalise the best partition size for every dataset.

With a dynamic partition size determination approach, DDMAS will automatically attempt to find out what the best partition size is. In this mode, DDMAS will run N iterations, each time using a different value for *i where,* $i > 1$, $i < N$, $i = x * 2$ and where x = the current iteration. The *best partition size* is calculated by either one of following criteria: the highest accuracy or the greatest efficiency. Depending on which criterion is selected by the user, the partition size which results in the best performance against the selected criteria is nominated as the *best partition size*.

The training process is explained as follow. Let say the dataset D consist of some very large number of tuple, let call this number of tuple *n*, so we have *n* tuple. Let *i* be the number of partition. Let *s* be the user configurable number of tuple per partition. Let ALG be user specified algorithm (eg: Decision Tree). The basic training algorithm for DMMAS will be:

1. The number of tuples is divided into *i* equally size partitions of size *s*.
   $i = n/s$ where $s = 1$ to n.
2. Agent Manager creates *i* numbers of agents. Agent A_i is allocated a dataset partition P_i.
3. Each agent performs its own data mining task with ALG and generates its own model M on the dataset partition that it held responsible.

This algorithm will reduce the training time by having the dataset to be trained in parallel independently. This algorithm is illustrated in Figure 4.9.

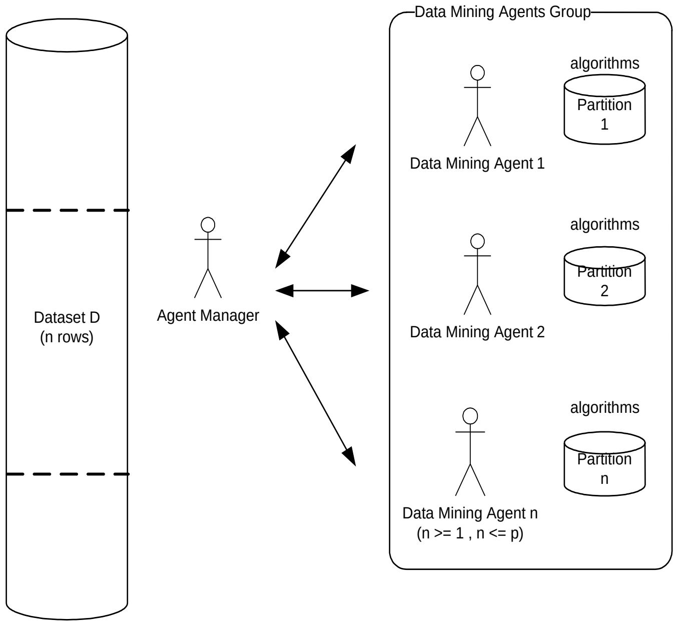


Figure 4.9 Dataset Allocation

4.2.4. Execution

The execution concept is similar to a network socket in the” listening” stage. This is where the DMMAS system is waiting for incoming instances. When an instance (tuple) is received, it is placed in to a queue. The Agent Manager then forwards each item in the queue to all available agents on the system asynchronously.

There is a time out period associated with each classification request that the Agent Manager sends to data mining agents. By the end of this time out period, if a response has been returned the Agent Manager will not take that vote into account. Due to time constraints no retry mechanism has been implemented. This is discussed in the section on future work.

Occasionally, the decision of the agents can result in a tie. This is when 50% of agents classify instance X as class Y and the other 50% agents classify instance X as class Z. This scenario only happens when the number of agents is even. DMMAS handles tied situations by randomly selecting an agent's response and using this as the returned response. In our experiments this scenario rarely happened.

Having looked at the setup of the agent's infrastructure, it is time to look at how this approach handles data updates.

In step 1of the training algorithm discussed above, we did not mention the case where tuples remain from the divide algorithm. If we have an odd number of tuples; for example 11 tuples and a page size of 2, then we will end up with 5 partitions, and 1 last tuple as a remainder. We call this an *orphan* tuple. This tuple will need to wait until there are enough tuples to make a partition. For this situation we use an update algorithm, which requires the training algorithm as a prerequisite.

4.2.5. Dataset Update

The update algorithm waits until there is enough data to fill up a partition before the Agent Manager creates a new agent (also see Table 4.1).

```
Let $o$ be the number of orphan tuples
For each tuple inserted:
    If $o$ < page size s
        Increment $o$ by 1
    Else
        Agent Manager creates new Agent $A_i$
        Agent $A_i$ executes training process.
        Reset $o$ to 0
```

Table 4.1 Dataset Update Algorithm

The assumption we make here is that there will be no removal of existing tuple and the algorithm only handles tuple insertion.

We have just explored the heavy lifting components of DMMAS: the training process and the data update handling algorithm. Now we will look at how we utilise this infrastructure to a classification task.

4.2.6. Classification in DMMAS

We will only deal with classification in this thesis. One can argue that this is the most important phase of the algorithm because this is where the result is predicted and synthesised. Classification process is initiated by fetching an unseen instance into DDMAS. We employ majority voting to process the instance. We run the instance through all the agents in the system and interpret the returned result from the agent. Majority voting is a popular proven simple approach. In this research we employ majority voting and rules aggregation.

For the experiment in the thesis, we selected ID4.5 decision tree as our learning algorithm. This algorithm was selected for its popularity and simplicity as mentioned in the review chapter.

Classification in DMMAS is handled largely by the Data Mining Agent (DMA). The DMA's job is to handle the mining (learning) and classifying processes. The DMA starts after it is allocated a data partition via message exchange delivered from the Agent Manager. The message contains information such as the location of the data, the start index and the size of the data block. If the underlying storage mechanism is file-system, the instructions will include the file line number. If the underlying storage mechanism is a database, the name of the database and the table, as well as the row index will be included.

After retrieving the data block from the data source, DMA starts mining the data. The default algorithm is decision tree C4.5[1]. This algorithm is known as J48 in the Weka 3.6

[1] C.45 with revision number 8, the last version before C.45 becomes C.5 which is a commercial product.

data mining package . Once the decision tree is generated, DMA will convert the tree into decision rules.

In this study, each agent performs the learning task independently and in parallel. The agent learns from a set of training data using a designated algorithm. A computational node (e.g. a network computer that is part of a cluster) can have one or many agents.

Our objectives are to create a technique that can mine large datasets by leveraging the power of distributed computing. In addition we want to be able to mine the dataset in real-time, without having to take the dataset offline and without having to rebuild the learnt models.

4.2.7. Data Compression

We performed experiments to test the effect of compression of agent communications on performance. We observed that the CPU cycle and time to process compression at one end and uncompress at the other end outweighed the benefit of compression in most cases. This was due to the small size of data being transferred.

4.2.8. Agents Communication

In JADE, agent communication can be performed in three different ways: string, Java serialisation and XML. String representation such as (Book: title “Hello World”) can be used and parsed by agents; however it is not adequate for more complex representations. DMMAS uses Java serialisation to encode and decode the content object in agent’s message. Java serialisation is simple and yet powerful enough to express complex representations. However, its shortcoming is that it is only applicable in the Java environment and it produces nonhuman readable data. The fact that Java serialisation only works in the Java environment limits the system interoperability with other Agent platforms that do not understand Java. Non readable serialisation object makes debugging of agents' messages more difficult.

We also considered Google Protocol Buffer (PB) to encode the agent messages. PB is a way of encoding structured data in an efficient yet extensible format. Due to time constraints, however, this will be investigated further in future work.

4.3. DMMAS Experiments

4.3.1. Experiment Design

The experiment we perform on DMMAS explores the efficiency of multi agent mining in comparison to the classical batch mining. We are also interested in the efficiency convergence point; the point at which adding additional agents ceases to improve efficiency.

The critical factors that dictate performance of an algorithm are its accuracy and speed. Typically, accuracy is calculated by the percentage of correct classifications that a model makes on a given set of data.

Each experiment has two phases: testing batch mining and testing DMMAS. The first phase uses traditional batch learning. This phase is not applicable to a dataset where the number of instances is greater than five hundred thousand (500,000). Preliminary testing has demonstrated that on current hardware, the decision tree algorithm is not able to scale up to this number of instances. The second phase uses our multiagent approach on the same dataset. Before conducting the experiment, we performed a preliminary study using the Indian Pima diabetes dataset (see Appendix B for a discussion of the diabetes application problem). The results were promising (Tong, Sharma and Shadabi 2008). Due to the small size of the dataset (768 instances), however, we need to perform the experiment on a larger dataset. In this research we will repeat the experiment, this time using the Adult dataset with varying numbers of instances. The dataset sizes are: 48,842 instances (original dataset), 500 000 instances, 200,000 instances, 500,000 instances and 1,000,000 instances. The dataset size for each repetition is reported in Table 4.2 below.

Repetition	Dataset Size	Batch Mining	DMMAS
1	48,842	Yes	Yes
2	100,000	Yes	Yes
3	200,000	Yes	Yes
4	500,000	Yes	Yes
5	1,000,000	NA	Yes

Table 4.2: Experiment Repetition Dataset Parameters

Datasets larger than the original dataset size of 48,842 instances are generated by repetitively drawing with replacement from the original dataset. For more information on this procedure, see Section 4.3.

The first step of the experimental process is dataset initialisation. Dataset initialisation ensures that the dataset is online and ready for the mining job. If the dataset has less than 1 000,000 instances we will perform the batch mining job (Step 2); otherwise we go straight to the next process, which is mining using multi agents (Step 3).

The batch mining is a single process that utilises the decision tree from the Weka framework (weka.classifiers.trees.J48) to do the training and testing. 1/3 of the dataset is reserved for testing and the remaining 2/3 is used for training. At completion of the process the result (ie training duration and accuracy percentage) is logged to a comma separated value text file.

When the batch mining process is completed, we proceed to the next step; performing the classification task using DMMAS.

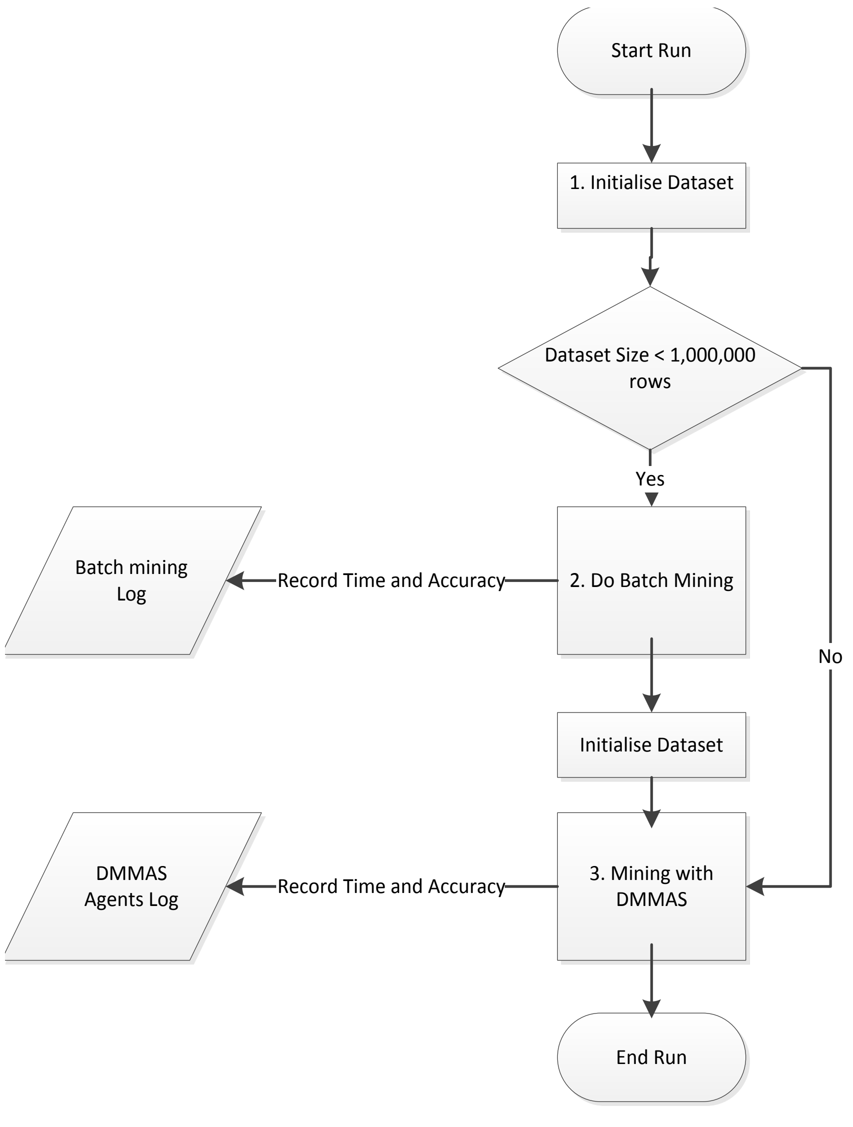


Figure 4.10 Experiment Iteration Flowcharts

The sub process of Step 3: Mining with DMMAS is illustrated in Figure 4.11 below.

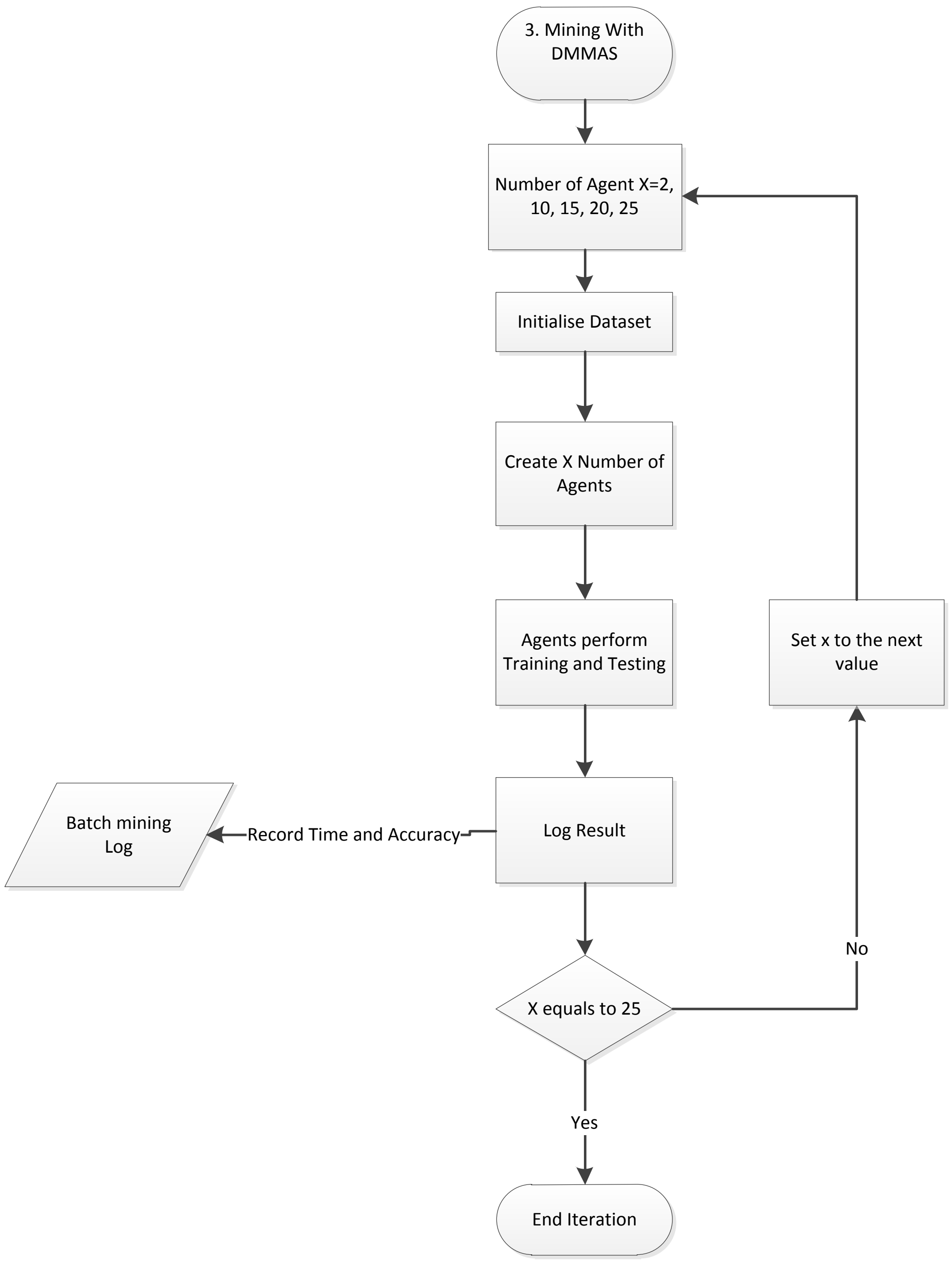


Figure 4.11 DMMAS Mining Sub Process

The DMMAS experiment involves a number of runs against the same dataset, with increasing numbers of agents used on each run. Each run will divide the dataset into 2, 5, 10, 20, 25 agents successively. Each agent uses decision tree (j48) for training and testing its local classifier.

The accuracy measurement is calculated by averaging the accuracy percentages of all agents. To measure the training duration, the duration of the agent that is last to finish is used as the overall speed. For example, say there are two agents; agent A and agent B. Agent A finished training in 2 minutes, while agent B finished training in 2.5 minutes. The training duration will therefore be 2.5 minutes.

4.3.2. Infrastructure

The experiment was performed on two computers, connected via a direct Ethernet 100Mbs link.

The naming convention for processing nodes is Cx where *x is an integer and greater than 0.* C1 is the name of computer 1 and C2 is the name of computer 2 and so on. This naming convention makes it easier to manage any additional processing nodes that may be required at a later stage. Both C1 and C2 serve as processing nodes. C1 hosts the data source on a MySQL database. The test machines run Windows and the software is independent from the underlying operating system.

It is important to point out that it is easy to add additional processing nodes. This needs to be set up before the training step. The agent manager will allocate data based on the number of processing nodes available.

Table 4.3 below depicts the hardware and software configuration for each computer. The experiment runs on the following hardware and software infrastructure:

Computer 1 (C1)	
Processor	Intel Dual Core 2.9 Ghz
Memory	4 GBs
Hard disk	500 GBs
Software	
Operating System	Windows Server 2008
Java runtime	1.6.010-beta
Agent Framework	JADE 3.6
Data mining library	Weka 3.5.7
Development Environment	Netbean 6.7.1
Database server	MySQL 5.1.37

Computer 2 (C2)	
Processor	Intel Core 2 Duo 2Ghz
Memory	2 GBs
Hard disk	120 GBs
Software	
Operating System	Microsoft Window Vista Business 32 Bits
Java runtime	1.6.010-beta
Agent Framework	JADE 3.6
Data mining library	Weka 3.5.7
Development Environment	Netbean 6.7.1

Table 4.3 Infrastructure of computers used for the experiments

4.3.3. Agent Container Setup

In both the batch mining and the stream mining experiment, the agent platform is composed of two containers: the main-container and a satellite container. Each container is hosted on a separate machine (C1 and C2 in this case) and each is capable of hosting multiple agents. The main-container contains the JADE housekeeping agents such as the Directory Facilitator, Agent Management System (AMS) agent and Remote Agent Management agent. Refer to (Bellifemine et al 2008) for more information about these JADE agents.

A satellite container consists of a Data Mining Agent, a Data Agent and an Agent Factory. JADE cannot create remote containers; hence the JADE container on each machine will need to be manually started from the DDMAS user interface.

4.3.4. Experiment Data

The UCI Repository of Machine Learning Database is a repository of databases, domain theories and data generators that are used by the machine learning community for the empirical analysis of machine learning algorithms. The experiments used the Pima Indian diabetes dataset and UCI Adult census dataset. More information about these datasets can be found in Appendix D.

The default formats for the datasets obtained from the UCI are in C4.5 file format. This presents some immediate challenges. Firstly, the format needs to be converted to arff file format which is a format understood by Weka. Weka then uses the arff file to export the data into a MySQL database table. This enables the agents to perform input, output on the data concurrently; something that cannot be done with flat file formats such as C4.5 or arff format.

Additional records were generated to create larger datasets than the original for the purposes of the experiment. The additional records were created using the technique of random sampling with replacement strategy.

In order to automate the generation of larger databases, we created a utility called Data Generator which can be accessed via the user interface (Tools -> Data Generator). The data generator’s job is randomly selecting and inserting an existing instance from the source dataset into the destination dataset until the required number of rows in the destination dataset is reached.

Generating a large amount of data is a challenge in itself. There were a number of problems in generating large datasets. The first problem is with the memory constraints. The second problem is the disk swapping required for the read and write operations. As a result, this made the data generation process very slow. Importing 100,000 randomly selected tuples from one dataset to another took approximately 10 minutes.

The process of bagging (discussed in section 2.3.6.1), involves selecting several training sets by randomly drawing from the original dataset with replacement. This means it is likely that an instance can appear multiple times in one or more training sets. We are looking at very large to potentially infinite databases, and this approach does not scale to the size of our datasets. Therefore, instead of drawing instances with replacement, we just simply divide the dataset into *x* number of data partitions, where each partition represent a training set.

When classifying an unseen instance, the Agent Manager broadcasts the instance to all DMAgents. Each DMAgent will classify the unseen instance based on their knowledge, and then send a response back to the Agent Manger. The Agent Manager will determine the outcome using majority voting and return the result to the user.

4.3.5. Batch Mining

This is where a single process performs training and testing on the generated dataset. A dataset is divided into two parts: training and testing. 66.6% of the dataset is selected randomly for training and the remaining 33.3% is used for testing.

4.3.6. DMMAS Algorithm

The learning algorithm used for this research is Quinlan’s C4.5 extension 8 (Quinlan 1993). This algorithm is implemented in the Weka data mining software as J4.8 class.

Each mining agent is allocated some training data, called a training partition. The agent will learn from the given training partition and produce a decision tree classifier. This rule set becomes the agent's knowledge or it's perception about the world. The agent together with other agents forms a network of agents whose expertise are required to perform classification and prediction.

Within the test datasets, the dataset is divided into smaller partitions. Each agent is allocated a distinct data block. The agent uses the allocated data block for training the classifier and a common test classifier for testing. Each agent uses the same classifier algorithm, in this case a decision tree.

4.3.7. Evaluation Method

In our experiment, we made observations on two factors: the time taken and the accuracy. At the system level, the overall time is defined as the time taken for DMMAS to complete its classification tasks. To measure the time taken by different parts of the system, including classification and communication, the overall time is decomposed into two parts: the classification time and the communication time.

It is important to point out that the time measures can be influenced by many factors, such as the computational power of the machines used, the workload of the machine where the agent runs, and the network bandwidth and latency.

In measuring classification time we measure the time taken to train the model and classify an unseen instance. Accuracy is taken as the classification accuracy.

When measuring speed, it is important to differentiate between learning time and processing time. In static large datasets, learning time refers to the time taken for all agents to finish learning the data. Processing speed refers to the time taken to process an unseen instance.

4.4. Summary

This chapter presented the DMMAS implementation which includes the UI and some internal aspect of the system. The UI-controls enable user to initialise agent platform, train agents and perform classification. The internal aspects of the system are data compression, agents' communication and classifying in the system. It also discussed the experiments framework.

The experiment section explained the experiment design and the setup of the hardware and software infrastructure used in the comparison experiment. The experiments benchmark the traditional batch mining and compare the result to the DMMAS approach.

Chapter 5

RESULTS AND ANALYSIS

5.1. Introduction

This chapter examines the result from the experiments discussed in section 4.3 on the Adult census dataset. The analysis considers the impacts on performance in terms of the efficiency and accuracy. The preliminary result for the type II diabetes Pima Indian dataset was reported in (Tong, Sharma & Shadabi 2008).

5.2. Batch Mining Results

Table 5.1 presents the result of running a single process against 4 derived dataset from the Adult Census dataset. The first, second, third and fourth datasets consist of 48,842 instances, 100,000 instances, 200,000 instances and 500,000 instances, respectively. We could not perform the experiment on the dataset with 1,000,000 instances due to memory constraints - even after the Java platform virtual heap was set to the maximum of 2GBs.

In our discussion, the training time is the time taken to build a model. The testing time is the time taken to finish classifying the testing set.

For each database size, 3 runs were performed; each time the training time and testing time were recorded. The results were averaged and recorded in Table 5.1 below.

Experiment ID	Dataset Size (instances)	Training Size (instances)	Testing Size (instances)	Training Time (sec)	Testing Time (sec)	Accuracy (%)
1	48,842	32,561	16,281	4.5	2	85.84%
2	100,000	66,666	33,334	23,67	15	94.82%
3	200,000	133,333	66,667	80	95	98.62%
4	500,000	333,333	166,667	265	240	99.1653 %
5	1,000,000	N/A	N/A	N/A	N/A	N/A

Table 5.1 Adult Dataset Single Process Experiment's Result

In Figure 5.1, in terms of training and testing time, we observed that the training time often takes a little longer than testing time. The duration for training and testing increases exponentially from only 2 seconds for 48,842 instances to 265 seconds for 500,000 instances. The dataset size grows roughly 10 times however; the duration it takes to train the half million instances dataset is about 130 times longer. This is because all the instances were loaded in memory to construct the decision tree. The bigger the data (hence the bigger the decision tree), the slower the training time will take.

The data in Table 5.1 is further illustrated in Figure 5.1, Figure 5.2 and Figure 5.3.

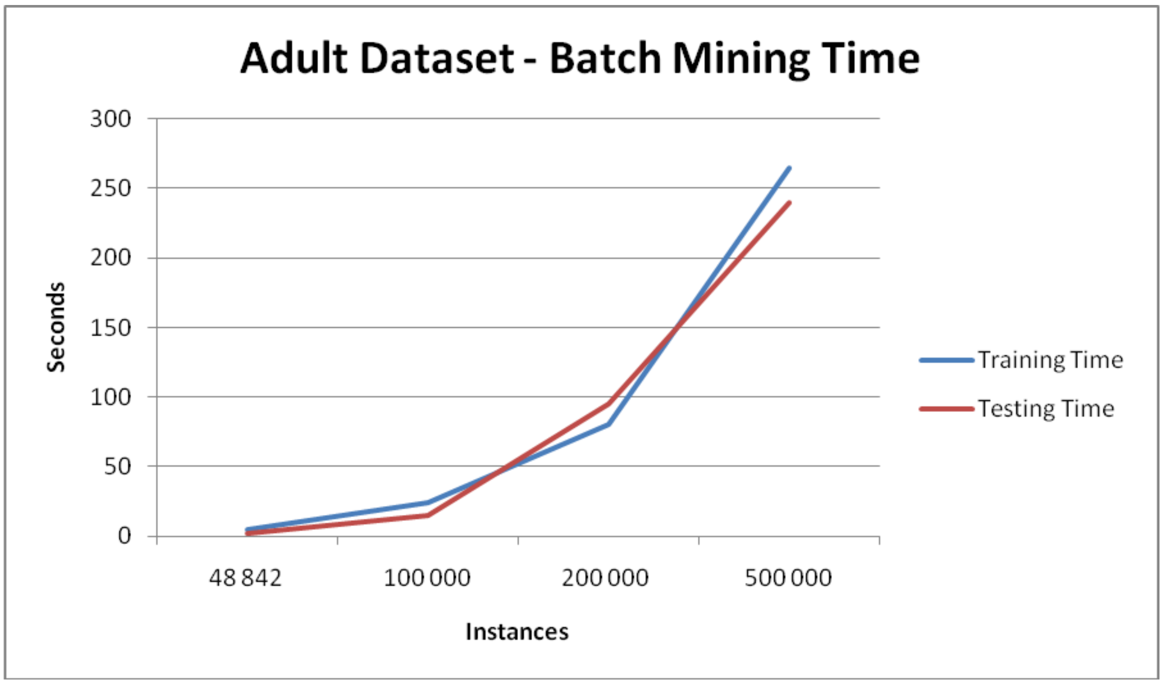


Figure 5.1 Adult Dataset – Single Process Training Time and Testing Time

Figure 5.2 illustrates the batch mining accuracy that use C4.5 decision tree in a process. The x-axis represents the dataset and the y-axis represents the accuracy (in percentage). The accuracy rises steeply from 84% to 95% in 48,842 instances to 100,000 instances. The accuracy then slows down from 95% to 98% in 100,000 instances to 200,000 instances. After this rise, the subsequent rise is very insignificant, that is from 98% to 99.1% in 200,000 instances to 500,000 instances.

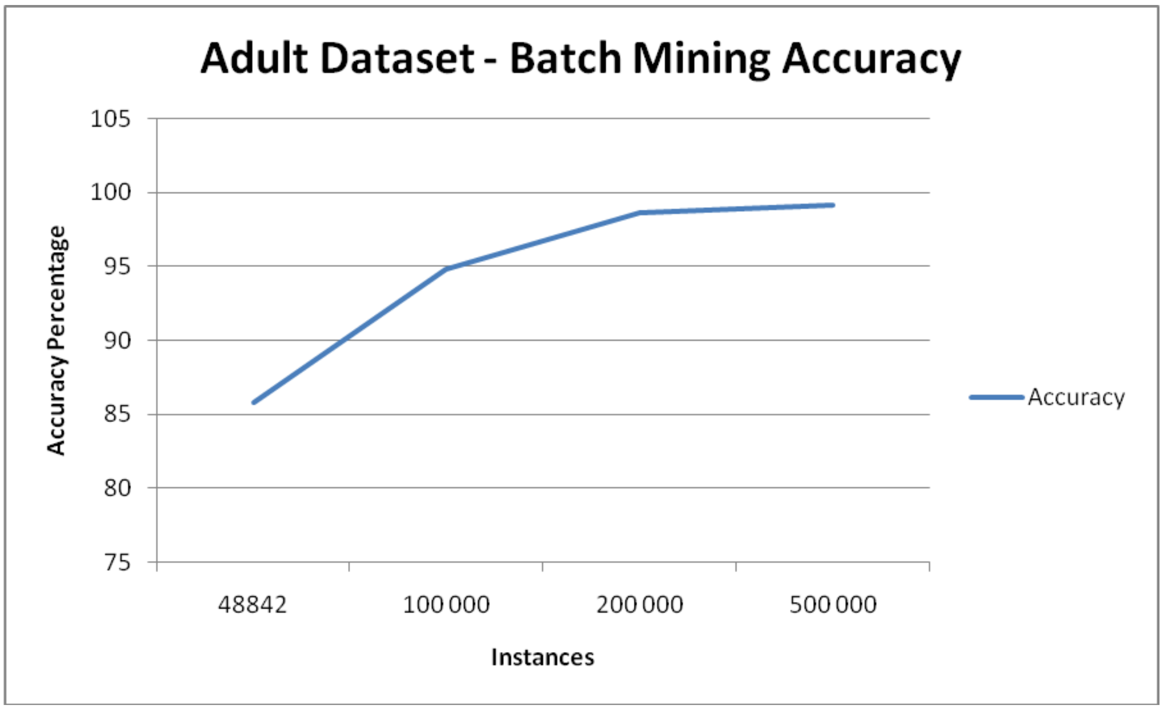


Figure 5.1 Adult Dataset – Batch Mining Accuracy

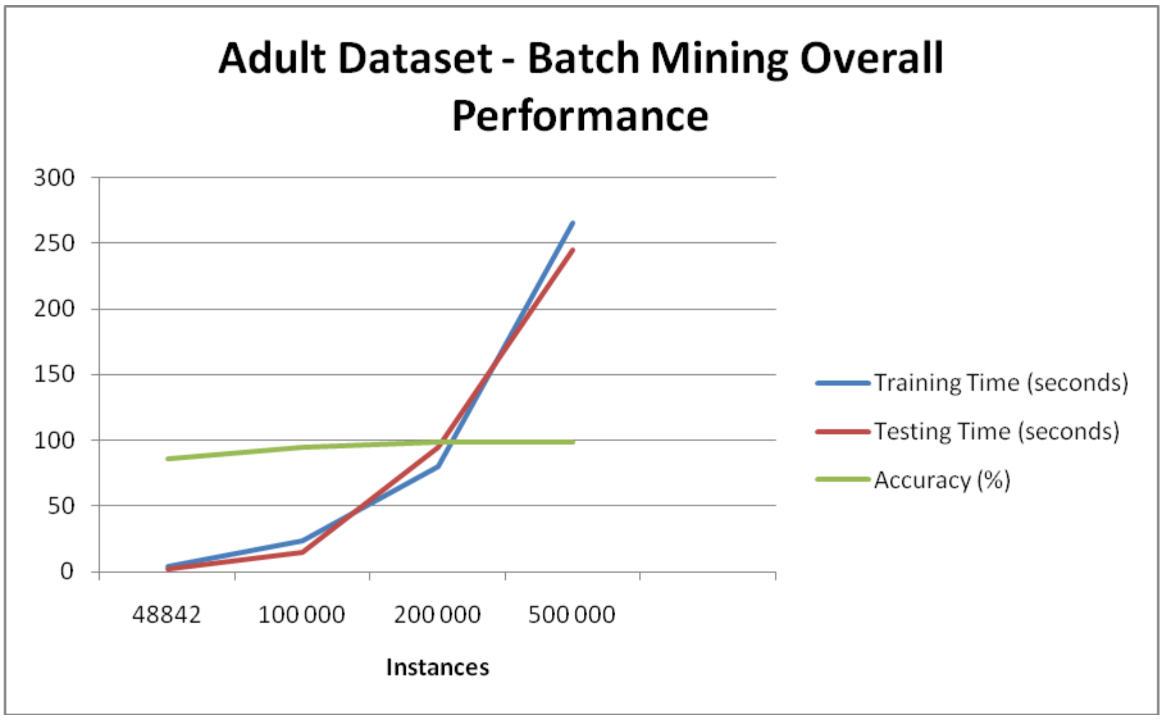


Figure 5.2 Adult Dataset – Batch Mining Overall Performance

We put Figures 5.1 and 5.2 together to create Figure 5.3. This allows readers to have better visual on the graph. Both testing time and training time grow as the dataset becomes larger. What stands out is the point from 200,000 instances to 500,000 instances. The training and testing time here is quite steep. It is almost 2.5 times from about 100 seconds (for each training time and testing time) to roughly 250 seconds. An explanation for this is because of high number of page fault where the data is not in physical memory and the operating system needs to access it from disk. Nevertheless, the accuracy rises marginally from 94% to 99%. This is only 5% gain in accuracy at the cost of almost triple the time.

5.3. DMMAS Results

The following table presents the same experiment conducted with batch mining approach, but this time using the DMMAS with a number of different agents.

Dataset Size (no of instances)	Number of Agents														
	2			5			10			20			25		
	Training (second)	Testing (second)	Accuracy (%)	Training (second)	Testing (second)	Accuracy (%)	Training (second)	Testing (second)	Accuracy (%)	Training (second)	Testing (second)	Accuracy (%)	Training (second)	Testing (second)	Accuracy (%)
48 842	2	1	83%	1.5	1.6	80.03%	0.7	1.3	76.45%	0.8	0.5	73%	0.5	0.30	72.5%
100 000	11	17.45	90%	8.5	19	87%	6.5	22	85%	3.5	23.4	83.45%	2.2	30	82.4%
200 000	22.32	30.5	96%	15.2	33.5	95.4%	10	34	94.9%	6.3	37	89%	4.9	37.9	87.9%
500 000	125	60	99.65%	45	63	98.3%	30	66	95.5%	18.5	68	93.6	16.7	75.4	92.5%
1 000 000	280	80	99.91%	214.5	90	98.4%	185	95	95.4%	45	93	93%	20	96	92%

Table 5.2 DMMAS Results

The general observation here is that the higher the number of agents, the faster the training time. A more detailed comparison is discussed in the next section.

The data in Table 5.2 presents the results that we observed with the DMMAS approach. The first column is the list of datasets that we use for the experiments. Next column is the number of agents that will be running on each dataset. For each dataset, we repeat the experiment with 2, 5, 10, 20 and 25 agents. For each repetition, we record the training time, testing time, and the accuracy. These are the sub columns in the number of agent column. The numbers recorded in these three columns are the result of averaging the recorded number in each of the 3 repetitions. For clarity, the data is further illustrated in Figure 5.5, Figure 5.6 and Figure 5.7. It is easier to look at the graph than looking at the raw data.

Figure 5.4 looks at the multi-agents training. The x axis represents the number of agents. The y axis represents the training time in seconds. For smaller dataset size (48,842 instances, 100,000 instances, 200,000 instances), adding more agents did not make much of a difference in training time. However we start to see more significant gain in time as dataset becomes larger. For larger dataset size (1 million instances), the duration is reduced from 214 seconds (2 agents) to 16.7 seconds with 25 agents. This is roughly 12 times gained in training time.

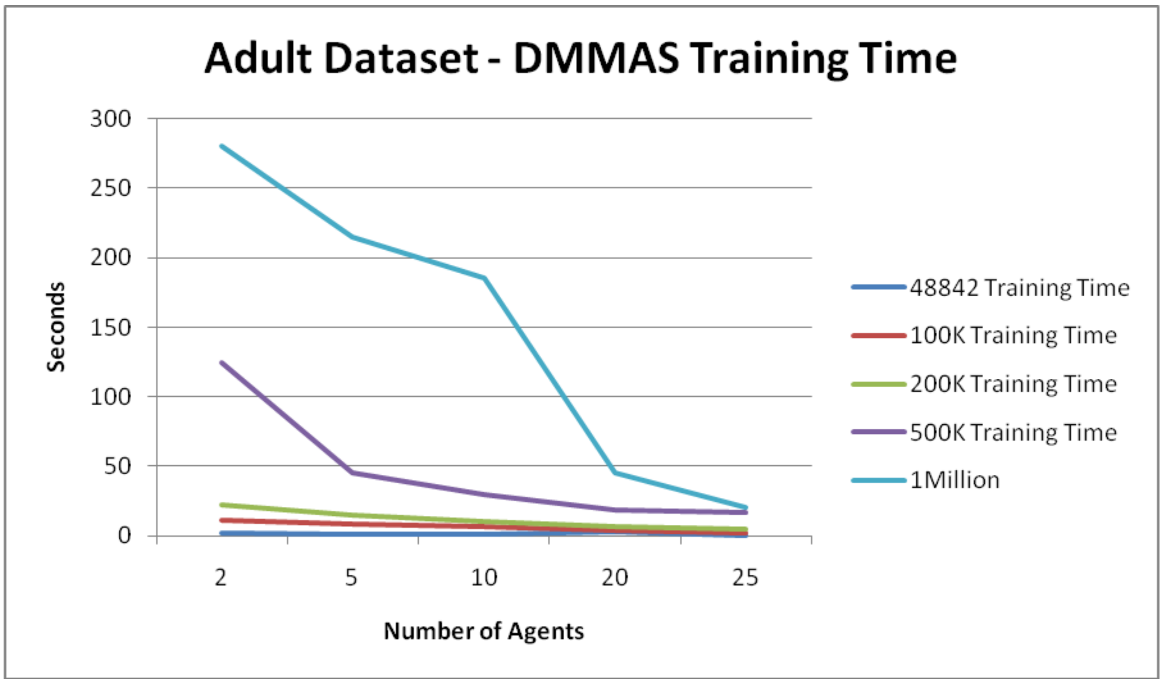


Figure 5.3 DMMAS Training Time

Figure 5.5 looks at the multi-agents testing time. Again here, the x axis represents the number of agents. The y axis represents the training time in seconds. There are five lines in this graph. Each line represent the testing time on each dataset. The bottom line represents the testing time for dataset with 48,842 instances, starting with 2 agents, 5 agents and goes up to 25 agents.

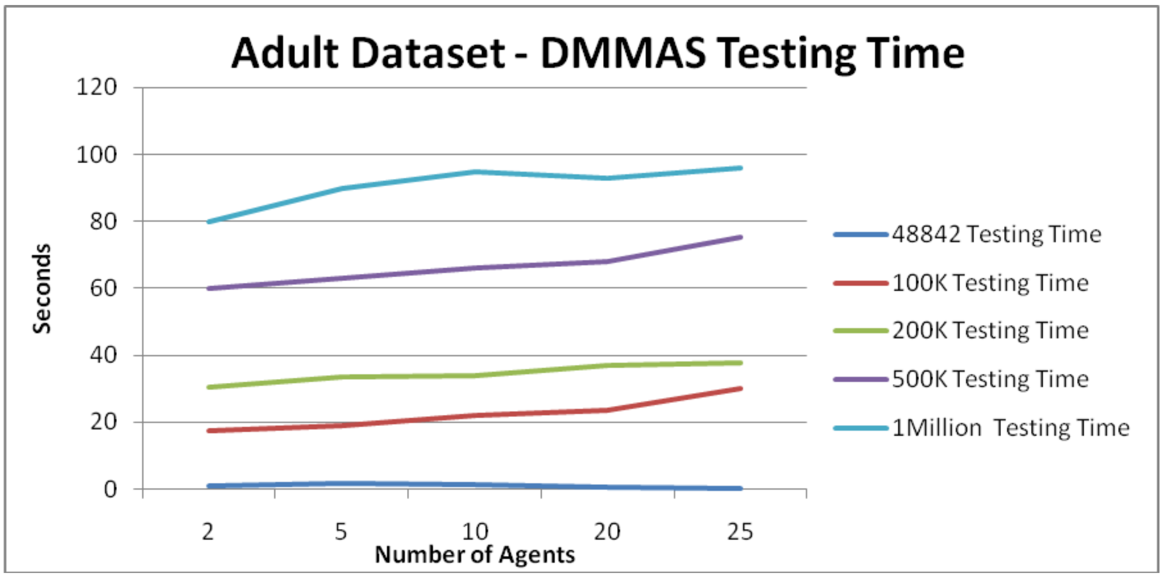


Figure 5.4 Adult Dataset DMMAS Testing Time

Figure 5.6 below looks at the accuracy of the DMMAS approach. Similar to previous two graphs, x-axis represents the number of agents. Y-axis is the accuracy (in percentage). The five lines represent the multiagent accuracy on 5 datasets. From the graph, we can see that the accuracy decreases as the number of agents increases.

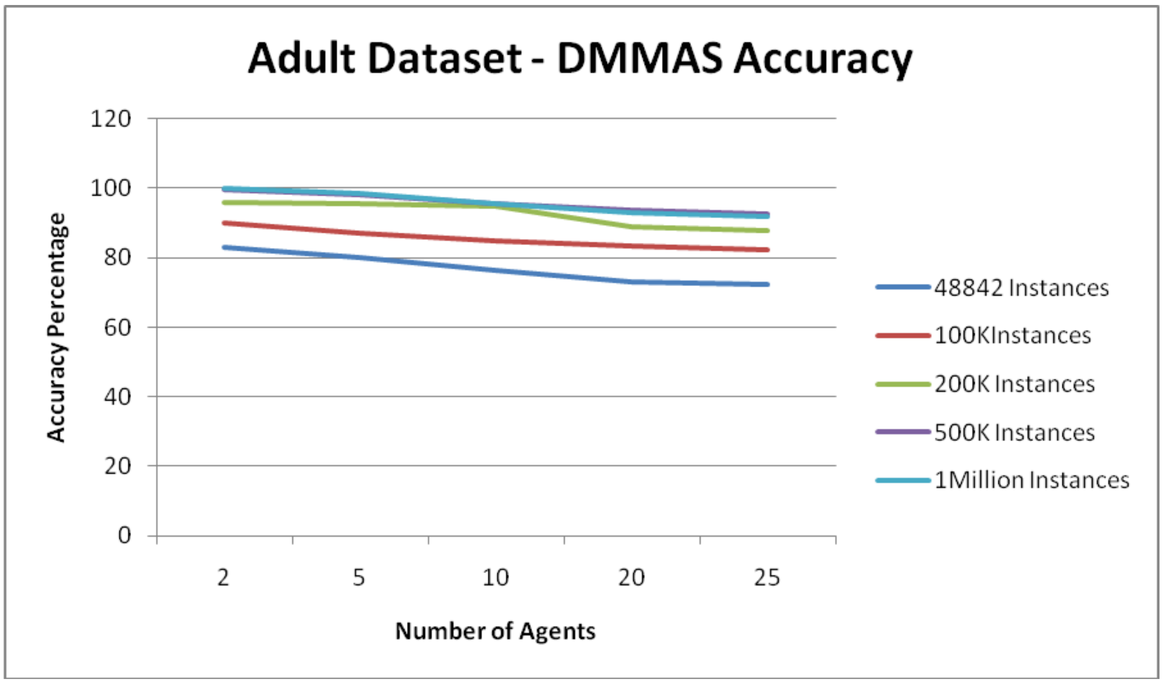


Figure 5.5 DMMAS Accuracy

5.4. Comparison Analysis

The following table analyses the performance gain and loss in terms of training time, testing time and accuracy.

Dataset Size	Number of Agents														
	2			5			10			20			25		
	Training	Testing	Accuracy (%)	Training	Testing	Accuracy	Training	Testing	Accuracy	Training	Testing	Accuracy	Training	Testing	Accuracy (%)
48 842	+2.5 s +55.56%	+1s +50%	-2.84% -3.3%	+3s + 66.67%	+0.4s +40%	-5.01s -5.836%	+3.8s +84.44%	+1.3s +65%	-9.39% -10.98%	+ 3.7s +82%	+1.5s +75%	-12.84s -15.02	+4s +88.8%	+1.7s +85%	13.34% -15.54%
100 000	+12.67s +53.52%	-2.45s -16.3%	-4.82% -5.08%	+15.17s +64.08%	-4 -126%	-7.82% -8.24%	+17.17 +72.5%	-7	-9.82% -10.3%	+20.17 +85.2%	-8.4s	-11.37% -11.99%	+21.47 +90.7%	-15	-12.42% -13%
200 000	+57.68 +72%	+64.5	-2.62% -2.65%	+64.8 +81%	+76s	-3.22% -3.26%	+70 +87.5%	+61	-3.72% -3.77%	+73.7 +92.12%	+68s	-9.62% -9.76%	75.1 +93.85%	+57.1s	-10.72% -10.87%
500 000	+140 +52.83%	+180s	-0.5 -0.55%	+220 +83%	+167s	-0.865% -0.872%	+235 +90.3%	+164s	-3.65% -3.68%	+246.5 +93%	+162s	-5.56% -5.62%	+248.3 +93.69%	+174s	-6.65% -6.72%
1 Million	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA		NA	NA

Table 5.3 DMMAS Performance Gain

Table 5.3 above highlights the gain of DMMAS with different number of agents in comparison to traditional batch mining approach. The first column is the size of the dataset. The second column shows the number of agents and the associated training time, testing time and accuracy. In each of the cells for training time, testing time and accuracy there are two values. The first value is the raw difference (in seconds for training and testing and as a percentage for accuracy) between the agent performance and the single batch mining performance. The second value is the difference expressed as a percentage. The plus (+) sign means that employing DMMAS has resulted in an improvement relative to single batch mining, and the minus (-) sign means that employing DMMAS resulted in a reduction of outcomes.

For example, with 2 agents on a 48,842 instances dataset, DMMAS gained 2.5 seconds in training time (55.56%), and gained 1second in testing (50%). This results, however, in the loss of 2.84% (-3.3%) of accuracy in comparison to batch mining. So the trade off here is a 55.56% reduction in training time, a 50% reduction in testing time and a 3.3% loss of accuracy.

Based on our experiments, increasing the number of agents allocated to a dataset results in improved performance in training and testing. This means the more agents allocated to the dataset, the shorter the training and testing times. The data suggests, however, that there is a point beyond which there is no significant gains to be had when more agents are added. This point is around the 20 agent mark. On a dataset of 500,000 instances, the gain of training time from 20 to 25 agents is 0.39% faster (from 93% to 93.39%).

This provides us with some basic confidence to infer that as more data become available, we can employ the DMMAS approach for faster result with little loss of accuracy. The loss of accuracy can only be determined by running DMMAS on the given dataset.

There were several studies on classification performance in the literature. STATLOG is probably the best known study (King et al., 1995). STATLOG was a very comprehensive study when it was performed, but since then important new learning algorithms have been introduced such as bagging, boosting and multiagent integration. In addition, the Adult census database was not available at the time hence it was not included in the STATLOG study. The closest performance study is a more recent empirical study from Caruana & Niculescu-Mizil that included the adult census dataset.

There were not many performance studies on the Adult dataset. The closest study on this dataset is the study from Caruana et al (2004). However, experiment is also slightly different to our experiment. For the basic experiment, the authors use 5000 instances for training whereas we use 16000 instances. In addition, the hardware that the experiment ran on is also different. Considering those difference into account, it is not feasible to perform a thorough comparison. A thorough comparison with past study on the same hardware and experiment parameter is a research project in itself. This however, can be scoped for future work. King et al pointed out that there is no universally best learning algorithm. Even the best models perform poorly on some problems, and models that have poor average performance perform well on a few problems or metrics.

5.5. Summary

This chapter reported 2 experiments running on derived Adult Census Income datasets. The first experiment runs batch mining and the second experiment runs DMMAS on all 5 derived Adults Census Income datasets.

Our experiments have demonstrated the advantages of using DMMAS for classification tasks. The primary advantage is that DMMAS was able to perform data mining tasks with base algorithms that are not able to load large data in memory. The improvement in efficiency of DMMAS becomes increasingly apparent as dataset grows larger. In addition, the loss of accuracy that occurs with DMMAS compared to batch mining also becomes smaller as more data is made available.

The results of the performed experiments are by no means conclusive. They provide, however, a model which can be used for further exploration of the benefits that multi agent technology can bring to data mining tasks. For large datasets, the benefit of improved efficiency which DMMAS can bring will potentially outweigh any negative effects as a result of a loss of accuracy.

Chapter 6

CONCLUSION AND FUTURE WORK

As an important operational problem, large dataset classification is made even more challenging by the increasingly distributed information and computing resources. It becomes impractical to process these distributed and dynamic information resources, such as the web logs, using centralized systems. Centralized systems will still be useful in processing archived data collections, however, distributed systems are in great need to deal with highly distributed and dynamic online information.

The multi-agents framework is one of the computing paradigms to tackle automatic text classification in distributed environments. The multi-agent framework provides the methodology to build intelligent agents and the communication mechanism that connects those individual agents into a coherent community. This study was motivated by the lack of research in multiagent framework in the area of data mining.

To advance the research on distributed classification using a multi-agent framework, this study attempted to solve the following challenges:

- How can we perform data mining tasks on large datasets that do not fit in main memory?
- Can we use multi-agents for distributing data mining tasks?
- Can multiple classifiers be combined into one main classifier for classification tasks?

- How can we optimise combination of sub classifiers as generated by multiple agents?
- How to coordinate classification activities in a distributed environment?
- How to achieve efficient classification in a distributed environment?

To solve those challenges, this study proposed a distributed approach to tackle classification using a multi-agent framework. This approach aimed to achieve improved classification efficiency as its centralised counterpart with acceptable classification effectiveness. This approach was implemented in a prototype system and evaluated using comparative experiments on a standard data collection.

In this study, we looked at two approaches: the traditional centralised classification and the proposed DMMAS system. We have found, on a large dataset, DMMAS performs well with both efficiency and accuracy. In addition, DMMAS also avoids the problem of single point of failure and the performance bottleneck in the centralised approach. DMMAS approach demonstrated more efficient classification performance, but its effectiveness result was marginally lower than the centralised approach. In conclusion, as the size of the data increases or changes, centralised approach may not be feasible, while the DMMAS approach can sacrifice a little effectiveness to maintain high efficiency.

The contributions of this study can be summarized as follows. First, this study demonstrated that in dealing with distributed classification the multi-agent approach can achieve the same level of effectiveness with better efficiency. Therefore, the multi-agent approach can serve as alternative for the centralised approach when the centralised approach is infeasible. Second, we can easily apply DMMAS approach to different domains not limited to health and census data. Lastly, an evaluation methodology for evaluating classification in a multi-agent framework was proposed and provided successful results throughout the reported experiments.

Reflecting the main research questions in Chapter 1, we can perform data mining tasks on large datasets that do not fit in main memory. This can be achieved by using the DMMAS algorithm to partition the data and leverage the multiagent system capabilities for data mining task. This research did not combine multiple classifiers into one main classifier; however, it leverages the classifiers responses to build a unified response.

Research Limitations

Due to the relative small size of the data collection used in this study, the proposed approach was tested only in relatively small-scale experiments (up to 1 million instances). Although it was demonstrated that the advantage of the multi-agent approach was improved efficiently, the accuracy is not yet clearly and confidently established. In order to be able to support real-world applications, further experiments on a much larger scale must be conducted.

The experiments we conducted assumed that the data is clean and contains no missing values. In the real world this is not always the case.

Future work

In addition to addressing the existing limitations that are outlined in the limitation section, the following items are scoped for future work:

- Each agent may utilise different reasoning techniques depending on the situation. For instance, data mining agent 1 can use a decision tree algorithm, data mining agent 2 can use regression, data mining agent 3 can use support vector machine. Each technique usually has its own advantages and disadvantages under different circumstances. It will be useful if the agent is able to determine which algorithm to use based on its belief (based on past experience).

- It should not also be ruled out that the greater the number of agents, the higher the communication costs which could have a negative impact on the system. This needs to be investigated further in future work.
- For the current model, the Data Generator process runs in the same thread as the main UI thread. When a data generation process task takes too long, this will cause the main UI to freeze, leading to poor user experiences. This limitation can be addressed by having the Data Generator task run on a separate thread.
- Agent management can be improved by reporting what each agent is doing in real-time.
- Running DMMAS on better infrastructure, such as 64 bits operating system to leverage 64 bit memory addressing to improve performance.
- We ran DMMAS on a 100Mbps Ethernet link. The agent communication and data retrieval efficiency can be improved by upgrading this to a 1Gbps direct Ethernet link.
- The current agent communities are static. It would be desirable to investigate dynamic agent communities, where agents can join and leave at any time. The discovery of new agents, the removal of existing agents, and the existence of duplicated or not mutually exclusive agents are the main issues in a dynamic agent community. Such behaviour would break the current DMMAS implementation.

BIBLIOGRAPHY

Alsabti, K., Ranka, S. & Singh, V. 1998, *CLOUDS: A Decision Tree Classifier for Large Datasets*.

American Diabetes, A., Gale, G. & HighWire, P. 1978, 'Diabetes care'.

Baik, S., Bala, J. & Cho, J. 2004, 'Agent Based Distributed Data Mining', *PDCAT*, vol. 3320, eds K. Liew, H. Shen, S. See, W. Cai, P. Fan & S. Horiguchi, Springer, pp. 42-5<citeulike-article-id:3725083>.

Bellifemine, F., Caire, G., Poggi, A. & Rimassa, G. 2008, 'JADE: A software framework for developing multi-agent applications. Lessons learned', *Inf. Softw. Technol.*, vol. 50, no. 1-2, pp. 10-21.

Bellifemine, F., Poggi, A. & Rimassa, G. 2001, 'JADE: a FIPA2000 compliant agent development environment', *Proceedings of the fifth international conference on Autonomous agents*, pp. 216-7<citeulike-article-id:3725119>.

Bellifemine, F.L., Caire, G. & Greenwood, D. 2007, *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*, John Wiley \& Sons.

Breiman, L. 1996, 'Bagging Predictor', *Machine Learning*.

Breiman, L., Friedman, J., Stone, C. & Olshen, R.A. 1984, *Classification and Regression Trees*, {Chapman & Hall/CRC}.

Cao, L., Gorodetsky, V. & Mitkas, P.A. 2009, 'Agent Mining: The Synergy of Agents and Data Mining', *IEEE Intelligent Systems*, vol. 24, no. 3, pp. 64-72.

Chan, P.K. & Stolfo, S.J. 1993, 'Experiments on multistrategy learning by meta-learning', paper presented to the *Proceedings of the second international conference on Information and knowledge management*, Washington, D.C., United States.

Chmiel, K., Gawinecki, M., Kaczmarek, P., Szymczak, M. & Paprzycki, M. 2005, 'Efficiency of JADE agent platform', *Sci. Program.*, vol. 13, no. 2, pp. 159-72.

Dean, J. & Ghemawat, S. 2008, 'MapReduce: simplified data processing on large clusters', *Commun. ACM*, vol. 51, no. 1, pp. 107-13.

Dixon, T., Australian Institute of, H., Welfare, Welfare. Cardiovascular Disease, D. & Risk Factor Monitoring, U. 2005, *Costs of diabetes in Australia, 2000-01*, Australian Institute of Health and Welfare, Canberra.

Durfee, E.H., Lesser, V.R. & Corkill, D.D. 1989, 'Trends in Cooperative Distributed Problem Solving', *IEEE Transactions on Knowledge and Data Engineering*, vol. 01, no. 1, pp. 63-83.

Dwivedi, P. 2001, 'Archive - Where it started and the Problems of Perpetuity', paper presented to the *Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*.

Fayyad, U.M. 1996, *Advances in knowledge discovery and data mining*, Menlo Park, Calif. : AAAI Press : c1996.

Goebel, M. & Gruenwald, L. 1999, 'A survey of data mining and knowledge discovery software

tools', *SIGKDD Explor. Newsl.*, vol. 1, no. 1, pp. pp. 20-33.

Gorodetski, V., Karsaev, O., Samoilov, V., Konushy, V., Mankov, E. & Malyshev, A. 2005, 'Multi-agent system development kit: MAS software tool implementing Gaia methodology', in, *Intelligent information processing II*, Springer-Verlag, pp. 69-78.

Hamburg, U.o. 2009, *Agents Platform Links*, <http://vsis-www.informatik.uni-hamburg.de/projects/jadex/links.php>.

Han, J. & Kamber, M. 2001, *Data mining : concepts and techniques*, Morgan Kaufmann Publishers, San Francisco.

Han, J., Kamber, M. & Pei, J. 2006, *Data Mining: Concepts and Techniques, Second Edition*, Morgan Kaufmann.

Indian, U.P. 2008, *Pima Indians Diabetes Dataset*, viewed 1 Feb 2008, <http://archive.ics.uci.edu/ml/support/Pima+Indians+Diabetes>.

Institute, I.D., Education, D. & Region, C.P. 1986, *Sweet talk*.

Ira, S.R. 2004, 'INTELLIGENT AGENTS', *Communications of the Association for Information Systems*, vol. Volume14, 2004, pp. 275-90.

Jennings, N.R. 2001, '{An agent-based approach for building complex software systems}', *Communications of the ACM*, vol. 44, no. 4, pp. 35-41.

King, R., Feng, C., & Shutherland, A. (1995). *Statlog: comparison of classification algorithms on large real- world problems.* Applied Arti_cial Intelligence, 9.

Klusch, M., Lodi, S. & Moro, G. 2003, 'Issues of agent-based distributed data mining', *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, vol. Melbourne, Australia, ACM, New York, NY, USA, pp. 1034-5<http://doi.acm.org.ezproxy2.canberra.edu.au/10.1145/860575.860782>.

Larose, D.T., John, W., Sons & Wiley, I. 2005, *Discovering knowledge in data : an introduction to data mining*, Wiley-Interscience, Hoboken, N.J.

Mathers, C., Penm, R., Australian Institute of, H. & Welfare 1999, *Health system costs of cardiovascular diseases and diabetes in Australia 1993-94*, Australian Institute of Health and Welfare, Canberra.

Mehta, M., Agrawal, R. & Rissanen, J. 1996, 'SLIQ: A Fast Scalable Classifier for Data Mining', paper presented to the *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*.

Mitkas, P.A., Kehagias, D., Symeonidis, A.L. & Athanasiadis, I.N. 2002, 'Agent Academy: An integrated tool for developing multi-agent systems and embedding decision structures into agents'.

Muthukrishnan, S. 2005, *Data Streams: Algorithms and Applications*, Now Publishers Inc.

Negnevitsky, M. 2002, *Artificial Intelligence A guide to Intelligent Systems*.

Poncelet, P., Masseglia, F. & Teisseire, M. 2007, *Successes and New Directions in Data Mining*, Information Science Reference - Imprint of: IGI Publishing.

Prodromidis, A., Chan, P. & Stolfo, S. 2000, 'Meta-learning in distributed data mining systems: Issues and approaches', *Advances in Distributed and Parallel Knowledge Discovery*.

Quinlan, J.R. 1986, 'Induction of Decision Trees', *Mach. Learn.*, vol. 1, no. 1, pp. 81-106.

Quinlan, R. 1993, *C4.5: Programs for Machine Learning (Morgan Kaufmann Series in Machine Learning)*, Morgan Kaufmann.

Rich Caruana and Alexandru Niculescu-Mizil, *"An empirical comparison of supervised learning algorithms"*. ICML 2006: 161-168

Shadabi, F. & Sharma, D. 2008, 'Data Mining and Intelligent Multi-Agent Technologies in Medical Informatics', in, *Success in Evolutionary Computation*, pp. 73-92.

Shafer, J.C., Agrawal, R. & Mehta, M. 1996, 'SPRINT: A Scalable Parallel Classifier for Data Mining', paper presented to the *Proceedings of the 22th International Conference on Very Large Data Bases*.

Sharma, N. 2005, *A multi agent system framework for.NET*, University of Canberra. Information Sciences & Engineering, <http://erl.canberra.edu.au./public/adt-AUC20060726.153250>.

Sugumaran, V. 2008, *Distributed Artificial Intelligence, Agent Technology, and Collaborative Applications*, Information Science Reference - Imprint of: IGI Publishing.

Swan, N. & Rural Health Education, F. 2006, *The Diabetes series*, Rural Health Education Foundation, Canberra.

Taniar, D. & Rahayu, J.W. 2002, 'Parallel Data Mining'.

Tian, J. & Tianfield, H. 2003, 'A Multi-agent Approach to the Design of an E-medicine System', in, *Multiagent System Technologies*, pp. 1093-4.

Todorovski, L. & Džeroski, S. 2000, 'Combining Multiple Models with Meta Decision Trees', in, *Principles of Data Mining and Knowledge Discovery*, pp. 69-84.

Tong, C., Sharma, D. & Shadabi, F. 2008, 'A Multi-agents Approach to Knowledge Discovery', paper presented to the *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 03*.

Victoria, D. 2007, 'Diabetes type 2 - Better Health Channel', pers. comm.

Wakaito, U.o. 2008, *Weka Website*, viewed 28 July 2008, <http://www.cs.waikato.ac.nz/ml/weka/>.

Wiley, I. 1995, 'Practical diabetes international'.

Witten, I.H. & Frank, E. 2005, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann.

Wooldridge, M. 2002, *An Introduction to MultiAgent Systems. John Wiley & Sons Ltd, 2002*, John Wiley & Sons.

Appendices

APPENDIX A: PUBLICATION

The following refereed publication was achieved from the thesis work:

Cuong Tong, Dharmendra Sharma, Fariba Shadabi: A Multi-agents Approach to Knowledge Discovery *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology* (WI/IAT) Dec 2008

APPENDIX B: TYPE II DIABETES

Diabetes is chronic health condition where the body is unable to automatically regulate blood glucose levels, resulting in continual elevated glucose level in the blood. "Insulin" is a chemical hormone in pancreas (Victoria 2007). There are two types of diabetes: Type I and Type II. *Type I Diabetes* is where the pancreas stops producing insulin or doesn't produce enough insulin. The diabetic patient with Type I Diabetes needs to be injected with insulin to control blood glucose level. Historically, Type 1 Diabetes was known as Juvenile Onset Diabetes or Insulin Dependent Diabetes. It usually occurs before the age of 30. Statistic shows that it affects ten percent to fifteen percent of those with diabetes.

Type II Diabetes is when the body does not respond properly to insulin. Type II Diabetes patient's cells in the body usually become insulin resistant. This means insulin is no longer able to control blood glucose levels effectively. This is the most common type of diabetes affecting about eighty five percent to ninety percent of those with diabetes. It affects people over fifty who have a family history or who are overweight diabetes (American Diabetes, Gale & HighWire 1978).

The common symptoms for diabetes are increased thirst, frequent urination, tiredness, lack of energy, blurred vision, infections and weight loss (Sharma & Shadabi, 2007). If diabetes is left undiagnosed or poorly treated it increases the chances of complications, which include heart disease, kidney disease, nerve and circulation damage, impotence, blindness and lower limb amputation (Institute, Education & Region 1986)

A research conducted by (Mathers et al. 1999) in 1993 reports that diabetes mellitus is a serious and growing health problem among Australians, affecting almost 4% of the population. It significantly increases the risk of cardiovascular disease and some of the health system costs of cardiovascular disease can be attributed to diabetes. Cardiovascular disease and its risk factors, including high blood cholesterol, cost the Australian community $3.9 billion in direct health system costs in 1993-94. The estimated average annual health system cost of cardiovascular is around $570 per diagnosed case, compared with around $210 per case of high blood cholesterol (in Australia).

The direct Australia health system costs of diabetes mellitus are estimated to be $372 million in 1993-94 (Mathers et al. 1999). In 2000-2001, the cost quickly climbed up to $784 million, ranking diabetes fifteenth out of around 200 disease groups. This equated to almost $1,500 per diagnosed case of diabetes, or $42 per Australian in that year (Dixon et al. 2005). As we can see, the cost climbed more than double just in 7 years from 1993-1994 to 2001.

Treatment

The aim of diabetes treatment is to maintain blood glucose levels within the normal range, which is between 3.5 and 6 mmol/L before meals and 3.5 and 8mmol/L two hours after meals. This will help prevent possible long-term health problems as mentioned earlier. In addition, keeping a blood pressure and cholesterol within the recommended range is also very important to help prevent these long term problems (Swan & Rural Health Education 2006).

Summary

Diabetes places a substantial financial burden on people with the condition, their families and the societies. In Australia, over one million Australians have diabetes. Fifty percent of people with diabetes are not even aware they have it since the symptoms of diabetes may not appear until blood glucose level is reached above certain point (usually 15 mmol/l or higher) (Wiley 1995). While prevention is certainly ideal, early diagnosis and treatment is the next best thing.

APPENDIX C: DATASETS

Pima Indian Dataset

The first dataset that we used was the UCI Pima Indian dataset (Indian 2008). This dataset contains 768 instances of Pima Indian heritage females who were diagnosed for diabetes. In the dataset, there are 268 instances that were diagnosed with diabetes. Each instance has 8 attributes and the diagnostic result (diabetes negative or diabetes positive).

The attributes are as follow: number of times pregnant, plasma glucose concentration, diastolic blood pressure (mm Hg), triceps skin fold thickness (mm), 2-Hour serum insulin (mu U/ml), body mass index, diabetes pedigree function, age (years) and finally the test result as class variable (0 or 1). Class variable value is mutually exclusive, either diabetes negative or diabetes positive

Adult Dataset

The dataset contains 32,561 instances (about 4MB in text). In 76.07% of instances earnings are <= 50K. 23.93% of records have earnings of more than 50K. The dataset consists of 14 attributes and the result (>50K or <=50K). The prediction task for this dataset is to determine whether a person makes over 50K a year.

The attributes are as follows: age, work class, fnlwgt [final sampling weight], education marital-status, occupation, relationship, race, sex, capital-gain, capital-loss, hours-per-week, native-country and finally the income class. For a more complete description refer to Kohavi (1996). Class variable values are mutually exclusive- earnings are either >50K or <=50K.

APPENDIX D: UTILITIES FEATURES

ARFF to SQL

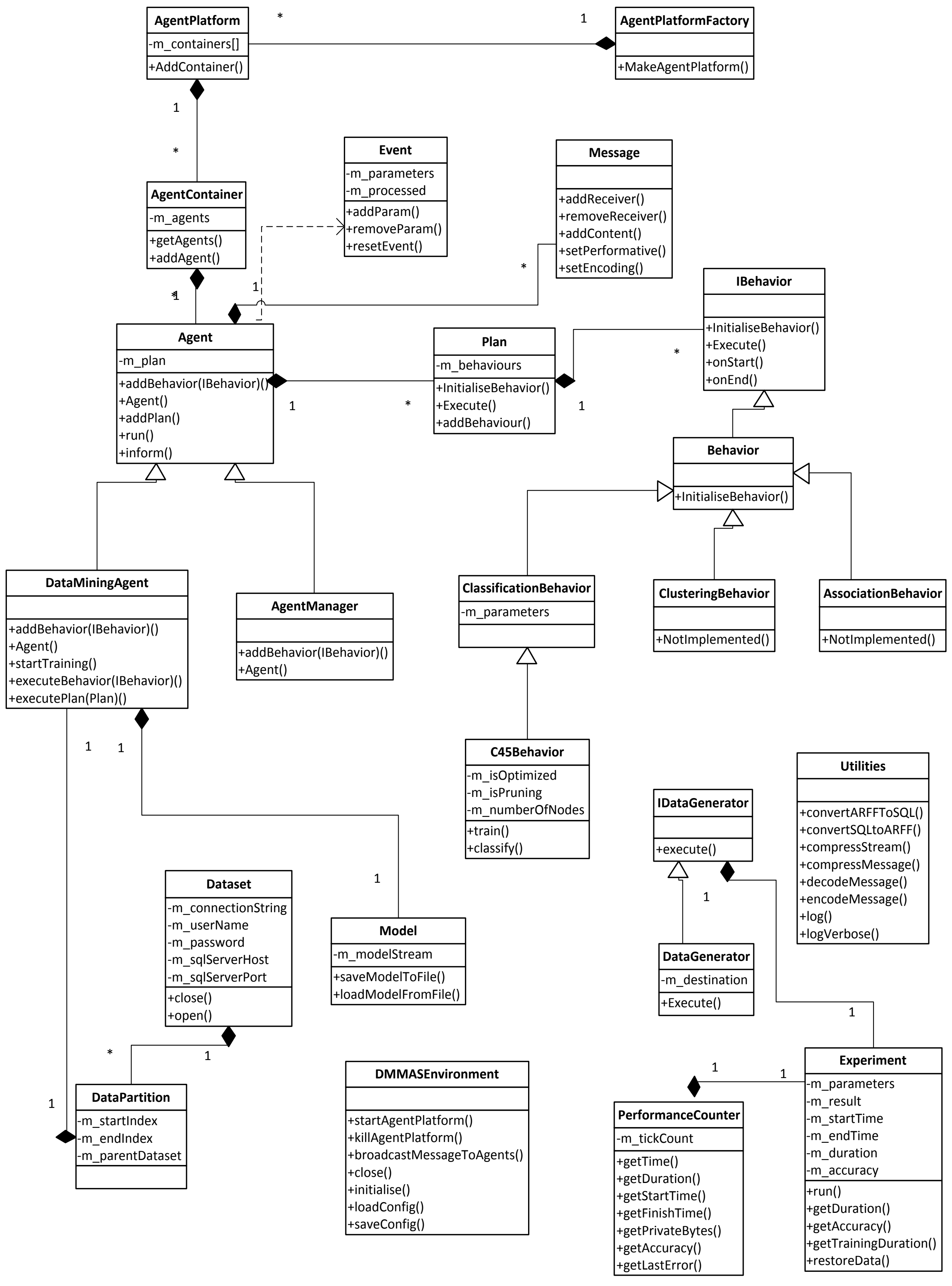
Attribute Relation File Format (ARFF) is a Weka data format. It is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF to SQL is a utility that converts datasets that are in ARFF format to a MySQL database. Each line in the ARFF file is copied across as a row in a MySQL table.

This utility was created because DMMAS cannot process ARFF data sources. ARFF data source files are not suitable for distributed and concurrent processing, as the file is locked to one process at a time. This limitation is avoided when migrating to a SQL data source.

Data Generator

As the focus of this research is performance with large datasets, it is essential for the dataset to be sufficiently large to see some meaningful improvement. This is where we use the Data Generator feature. We use the Data Generator to generate a new dataset from the old dataset. The Data Generator fills up the new dataset by randomly selecting an instance from the source dataset (with replacement). This means that an instance in the source dataset can appear multiple times in the new generated dataset.

APPENDIX E: UML DIAGRAM


AgentPlatform
-m_containers[]
+AddContainer()
AgentPlatformFactory
+MakeAgentPlatform()
AgentContainer
-m_agents
+getAgents()
+addAgent()
Event
-m_parameters
-m_processed
+addParam()
+removeParam()
+resetEvent()
Message
+addReceiver()
+removeReceiver()
+addContent()
+setPerformative()
+setEncoding()
IBehavior
+InitialiseBehavior()
+Execute()
+onStart()
+onEnd()
Agent
-m_plan
+addBehavior(IBehavior)()
+Agent()
+addPlan()
+run()
+inform()
Plan
-m_behaviours
+InitialiseBehavior()
+Execute()
+addBehaviour()
Behavior
+InitialiseBehavior()
DataMiningAgent
+addBehavior(IBehavior)()
+Agent()
+startTraining()
+executeBehavior(IBehavior)()
+executePlan(Plan)()
AgentManager
+addBehavior(IBehavior)()
+Agent()
ClassificationBehavior
-m_parameters
ClusteringBehavior
+NotImplemented()
AssociationBehavior
+NotImplemented()
C45Behavior
-m_isOptimized
-m_isPruning
-m_numberOfNodes
+train()
+classify()
Utilities
+convertARFFToSQL()
+convertSQLtoARFF()
+compressStream()
+compressMessage()
+decodeMessage()
+encodeMessage()
+log()
+logVerbose()
IDataGenerator
+execute()
Dataset
-m_connectionString
-m_userName
-m_password
-m_sqlServerHost
-m_sqlServerPort
+close()
+open()
Model
-m_modelStream
+saveModelToFile()
+loadModelFromFile()
DataGenerator
-m_destination
+Execute()
DMMASEnvironment
+startAgentPlatform()
+killAgentPlatform()
+broadcastMessageToAgents()
+close()
+initialise()
+loadConfig()
+saveConfig()
DataPartition
-m_startIndex
-m_endIndex
-m_parentDataset
PerformanceCounter
-m_tickCount
+getTime()
+getDuration()
+getStartTime()
+getFinishTime()
+getPrivateBytes()
+getAccuracy()
+getLastError()
Experiment
-m_parameters
-m_result
-m_startTime
-m_endTime
-m_duration
-m_accuracy
+run()
+getDuration()
+getAccuracy()
+getTrainingDuration()
+restoreData()
*
1