2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology

F-TRADE 3.0: An Agent-Based Integrated Framework for Data Mining Experiments

Peerapol Moemeng, Longbing Cao, Chengqi Zhang
Data Sciences & Knowledge Discovery Research Lab
Faculty of Engineering and Information Technology
University of Technology, Sydney, AUSTRALIA
{peerapol, lbcao, chengqi}@it.uts.edu.au

Abstract

Data mining researches focus on algorithms that mine valuable patterns from particular domain. Apart from the theoretical research, experiments take a vast amount of effort to build. In this paper, we propose an integrated framework that utilises a multi-agent system to support the researchers to rapidly develop experiments. Moreover, the proposed framework allows extension and integration for future researches in mutual aspects of agent and data mining. The paper describes the details of the framework and also presents a sample implementation.

1. Introduction

Agent and data mining integration and interaction (AMII) is an emerging area[3] due to the constant acceptance of agent computing and data mining as well as the convergence of industries, such as financial market, capital market, insurance, etc. Regarding data mining research, apart from the theoretical improvement, the work must be tested and we realise two difficulties in this scenario: software development and testing. (1) Firstly, the wrong person may be doing the wrong job. Software development may not be a common skill for all researchers especially expecting the product to be a well-managed one with usability and reusability. Software reusability is a common concept however troublesome since it requires a well preparation and implementation in software development life cycle[9]. (2) Secondly, algorithm optimisation is a major component in data mining research. This is a time consuming task as it takes an unpredictable number of turnarounds to refine parameters and run the test iteratively. Therefore, a desired functionality is automatic parameter tuning, which could enable processes to autonomously perform decisions on parameter tuning based on a given set of constraints.

With these problematic foundations, we propose a framework to support data mining and intelligent agent research in an integrated platform. Intelligent agent (agent, for short) plays an important role in presenting autonomy and intelligence in several components of the proposed system. In addition, the system allows extensibility and reusability by integrating external components into the running instance without having to interrupt the system and all those components are stored for future uses.

The main contributions of this work are (1) formulation of a formal framework for multi-agent systems that allows allows extensibility, reusability, integrity of system components varied upon particular task; (2) providing a platform for data mining researches which enhances research processes; and (3) providing a platform for agent researches to extend functionalities of the system incorporated with data mining capability.

2. Related Work

A nature of data mining is that systems are task-oriented. Adding agents to them requires modifications of agents, for instances [2, 11]. System implementations are guided by high-level system architectures which lack of detailed structures of the agents, which prevents the systems from extensibility and reusability. In this review, we cover two aspects of AMII: agent-based data mining and vise versa.

IDM[2] is an agent-based data mining system and is designed to support predefined and ad hoc data access, data analysis, data presentation, and data mining request from non-technical users over a data warehouse. IDM distributed environment is a result of using JATLite[7] which allows communications of agents through its message routing scheme which operates over a networked environment. The IDM's test system shows that IDM can be applied to a problem domain. The authors claim that IDM can rapidly analyse data because it delegates data mining tasks to mul-

978-0-7695-3496-1/08 $25.00 © 2008 IEEE
DOI 10.1109/WIIAT.2008.230

tiple agents. A similar system is iJADE[11], which is an integrated environment implemented with IBM Aglets[10] and Java Servlets, providing structured layers of system abstraction. iJADE can operate over various environments and configurations.

The other aspect of AMII is data mining-based agent. It is an agent system that embeds data mining capability into agents, in this case, agents are able to learn and reason respecting the specific domain. A work contributed to the idea is Agent Academy[12] (AA). AA is an integrated framework that interacts with its users through the system console application and the Web. It also provides a generic tool to create an AA system which is very useful as it shortens time in development. AA's approach fully utilises data mining intelligence to build better agents. Nevertheless, data domain used in AA is specific to a particular task. Variation of the system is made with change in core agent repository unit and data miner modules.

Our review shows the need for a framework that provides formal models to pose the system specifications into components that can be integrated and reused. In our work, we try to blend both aspects of AMII to create a formal framework that allows extensibility of both agent and data mining. System components e.g. algorithms, optimisation methods, datasets are stored in the repository for later use.

3. F-TRADE 3.0

The earlier version of F-TRADE was discussed in [4] and shows potential of system extensibility. In this section, we formulate the framework of F-TRADE 3.0.

3.1. Framework Modelling

The model is described in Z notation[13]. At central of the system, we introduce 4 basic sets that are central of interests in this universe, namely experiments (EXPE), algorithms (ALGO), datasets (DS), and optimisation methods (OPME). Universal set *[EXPE, ALGO, DS, OPME]* is set up as follows;

$$\begin{array}{|l} \textit{EXPE} \\ \hline algo : ALGO, ds : DS, opme : OPME, state : STATE, \\ cons : CONS, param : PARAM, output : OUTPUT \\ \hline \end{array}$$

Where ALGO, DS, and OPME are sets of identifiers referred to physical objects of the particular system, e.g.

$$algo == \{(package.algorithm.name)\}$$
$$ds == \{(directory.filename)\}$$

Regarding system configuration, we use paired attributes, keys (KEY) and values (VALUE), defined as follows.

$$\begin{array}{|l} \textit{KEY} \\ \hline key : STRING \\ \hline key = \{k : STRING; \bullet k \in (\mathrm{dom}\, attribute(ALGO) \\ \quad \vee \mathrm{dom}\, attribute(OPME))\} \\ \hline \end{array}$$

$$\begin{array}{|l} \textit{VALUE} \\ \hline value : ANY \\ \hline value = \{v : ANY; \bullet v \in (\mathrm{ran}\, attribute(ALGO) \\ \quad \vee \mathrm{ran}\, attribute(OPME))\} \\ \hline \end{array}$$

A supplement set *[PARAM, EVAL, OUTPUT, CONS]* contains contains domain values for parameters (PARAM), evaluations (EVAL), outputs (OUTPUT), and constraints (CONS). States of experiment (STATE) and operations (OP) for constraint checking are as follows.

$$STATE ::= ready \mid wait \mid finished \mid failed$$
$$OP ::= \leq | < | \geq | > | = | \neq | \in | \notin$$

System configuration sets are described as follows.

$$EVAL, PARAM, OUTPUT : KEY \leftrightarrow VALUE$$
$$CONS : KEY \leftrightarrow VALUE \leftrightarrow OP$$

These sets are to be instantiated specifically for each experiment, *for example eval : EVAL; cons : CONS; param : PARAM*.

SetupEXPE creates a new experiment at store into EXPE. Initial *state* of a new experiment is *ready* and *output* is *empty*.

$$\begin{array}{|l} \textit{SetupEXPE} \\ \hline \Delta EXPE \\ algo? : ALGO, ds? : DATASET, opme? : OPME, state : STATE, \\ param? : PARAM, cons? : CONS, output : OUTPUT \\ \hline state = ready \\ output = \emptyset \\ EXPE' = EXPE \cup \\ \quad \{(algo?, ds?, opme?, param?, cons?, state, output)\} \\ \hline \end{array}$$

The followings are in-line operators. *execute* executes an algorithm provided with a set of parameters and outputs a set of OUTPUT. However, algorithm execution must be specifically defined therefore the definition is left blank and yet to be extended.

evaluate evaluates the provided output against experiment's constraints and yields an evaluation result.

$$execute\_, \_ : ALGO \times PARAM \rightarrow OUTPUT$$
$$evaluate\_, \_ : OUTPUT \times CONS \rightarrow EVAL$$

fetch takes an experiment from EXPE. However, the detail specification can be extended as to compile with the system policy. For example, the system may determine load balancing in a distributed environment, therefore the behaviour of *fetch* is varied.

$$\begin{array}{|l} fetch\_ : EXPE \rightarrow experiment \\ \hline experiment \in EXPE \\ EXPE' = EXPE \setminus experiment \\ ..specific\ extension.. \\ \hline \end{array}$$

tune performs parameter alterations in order to achieve the experiment's constraints. It produces a new set of parameters from the given experiment. The optimisation method plays an important role here due to the optimisation computation determining input parameters with output against the constraints is performed here, furthermore, historical data need to be taken into concern. With regard to the extensibility of *tune*, its definition must be implemented specifically.

$$tune\_ : EXPE \rightarrow PARAM$$

..specific extension..

satisfy is a binary operation producing a boolean value by determining whether the provided evaluation results are accepted by designated operations of the experiment's constraints.

$$\_satisfy\_ : EVAL \times CONS \rightarrow BOOLEAN$$

$$\forall (ek, ev) : EVAL;\ (ck, cv, op) : CONS \bullet ek = ck \land ev\ op\ cv$$

Finally, *BackTest* runs repeatedly to obtain evaluation results that satisfy the experiment's constraints. It outputs a state of experiment as finished if all evaluation results satisfy, otherwise failed.

$$BackTest : EXPE \rightarrow STATE$$

$$expe : EXPE, output : OUTPUT, eval : EVAL, state! : STATE$$

$$\begin{aligned}
&expe = fetch\ EXPE \\
&state = failed \\
&\forall\ eval : EVAL;\ output : OUTPUT \bullet \\
&\quad eval = evaluate\ output \\
&\quad \land\ output = execute\ expe.algo,\ expe.param \\
&\quad \land\ if\ eval\ satisfy\ expe.cons\ then \\
&\qquad \land\ state = finished \\
&\quad else \land\ expe.param' = tune\ expe
\end{aligned}$$

3.2. System Architecture

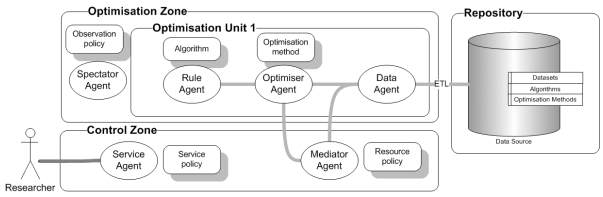

Figure 1. F-TRADE 3.0 architecture

Figure 1 shows the architecture of F-TRADE 3.0. The system is divided into three parts: control zone, optimisation zone, and repository. Each part contains a collection of system components, while agents work across these zones. The system flow starts from the control zone which determines the direction of the execution flow. *Control Zone* is where the user requests for experiment service and uploads his algorithm to the system, configures parameters, and submits into an execution queue. *Optimisation Zone* contains several instances of *optimisation units* in which may execute concurrently. *Repository* is a collection of system components, i.e. data sets, algorithms, and optimisation methods; furthermore, it can also maintain links to on-line resources, e.g. remote databases. The repository concerns a high degree of concurrency control and security as some components are restricted to some particular users.

Figure 1 also shows a group of agents communicating in a defined sets of channels as described in follows. *Rule agent* implements *execute()* and is responsible for running an equipped algorithm. *Optimiser agent* implements *evaluate()* and *tune()* and prepares experimental data for a rule agent. *Data agent* collects requested data for an optimiser agent since the data may be distributed, partitioned, and spatial, the agent is responsible to collect proper data in which it may block until collection conditions are met. *Spectator agent* keeps an eye on the optimisation zone and periodically checks for the progress. The agent is also responsible to alert the user, e.g. e-mail, when alert conditions are met. *Mediator agent* implements *fetch()* and is a single-instance agent in the system which acts as the optimisation manager. The agent is responsible in synchronising multiple optimisation zones, instructs a zone to start, pause, or stop. *Service agent* is a single-instance agent in the system which determines the privileges of the user in accessing the services of the system.

3.3. System Implementation

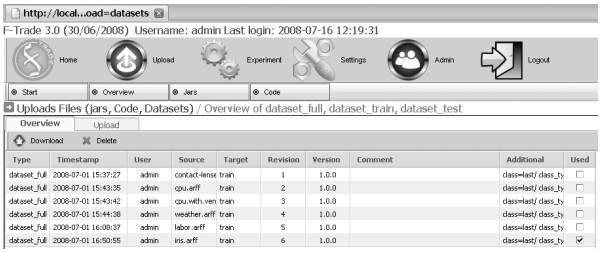

Figure 2. Sample screen

Figure 2 shows a sample screen of F-Trade 3.0. The system interface is a web-based implemented with PHP and AJAX to enhance usability and the underlying database is MySQL. JADE[1] is used as the core agent container. Due to JADE's FIPA compliance and extensibility, F-TRADE can be integrated further with other FIPA compliant applications. Nevertheless, we have integrated standard data mining algorithms from WEKA[14] into our system as to provide the user with more options. During this development, F-TRADE is published under GNU Public Licence as to compile with our regarding tools[1].

[1] DHTMLX (http://www.dhtmlx.com/) provides AJAX components for advanced Web UI

4. Case Study

In this section, we create an experiment and run a test by demonstrating how system formulation works. We use functions available from WEKA into our demonstration; We refer to John Platt's sequential minimal optimization algorithm for training a support vector classifier[8][6], the algorithm is known as weka.classifiers.functions.SMO in Weka package, to train a support vector machine on Iris Plants dataset[5]. The class of of the dataset, $class = \{setosa, versicolor, virginica\}$, and the dataset structure is $iriss =< sl, sw, pl, pw, class >$, where sepal length (sl), sepal width (sw), petal length (pl), and petal width (pw) are real numbers. WEKA SMO receives a range of parameters and we have chosen some that relevant as constraints to the experiment as follows; (1) **-C *double***: the complexity, (2) **-N**: Whether to 0=normalize/1=standardize/2=neither, (3) **-L *double***: the tolerance parameter, (4) **-P *double***: the epsilon for round-off error, (5) **-V *double***: the number of folds for the internal cross-validation, and (6) **-W *double***: the random number seed.

The kernel for the SVM we choose WEKA's classifier function, supportVector.PolyKernel, with following parameters: (1) **-D**: enables debugging output (if available) to be printed, and (2) **-E *double***: the Exponent to use.

Providing the constraints and parameters, we can now form the sets for an experiment.

$$
\begin{aligned}
&cons == \{(-C, 1.0, =), (-L, 0.0010, =),\\
&\quad (-P, 1.0E, =), (-N, 0), (-V, -1), (-W, 1)\}\\
&param == \{(-C, 250007), (-E, 1.0)\}\\
&ds == \{(iris.arff)\}, algo == \{(weka....PolyKernel)\},\\
&opme == \{(weka.....SMO)\}
\end{aligned}
$$

After creation of a new experiment, mediator agent that periodically perform fetch operation will eventually receive an instance of the experiment. Mediator agent then forwards the instance to a rule agent to perform execution operation. The results of the experiment are stored in the database as key-value records as shown in figure 3 which are ready for further analysis.

key	value
IR_recall	1
Serialized_Model_Size	19881
Mean_absolute_error	0.2305184
Root_mean_squared_error	0.28648731
Relative_absolute_error	51.86666657
False_negative_rate	0

Figure 3. Experiment result

5. Conclusions

This work shows significance of agent and data mining integration and interaction in enhancing complex systems allowing extensibility, reusability, and intelligence to embed. We will need to investigate and gather further requirements to improve the usability and friendliness of the system. Finally, this work provides a platform for agent research as a playground of agents incorporating with data mining capability.

References

[1] F. Bellifemine, G. Caire, and D. Greenwood. *Developing multi-agent systems with JADE*. Springer, 2007.

[2] R. Bose and V. Sugumaran. IDM: an intelligent software agent based data mining environment. *Systems, Man, and Cybernetics, 1998. 1998 IEEE International Conference on*, 3, 1998.

[3] L. Cao, C. Luo, and C. Zhang. Agent-Mining Interaction: An Emerging Area. *LECTURE NOTES IN COMPUTER SCIENCE*, 4476:60, 2007.

[4] L. Cao, J. Ni, J. Wang, and C. Zhang. Agent Services-Driven Plug and Play in the F-TRADE. *17th Australian Joint Conference on Artificial Intelligence, LNAI*, 3339:917–922, 2004.

[5] R. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[6] T. Hastie and R. Tibshirani. Classification by pairwise coupling. In M. I. Jordan, M. J. Kearns, and S. A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. MIT Press, 1998.

[7] H. Jeon, C. Petrie, and M. Cutkosky. JATLite: A Java Agent Infrastructure with Message Routing. 2000.

[8] S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(3):637–649, 2001.

[9] C. Krueger. Software reuse. *ACM Computing Surveys (CSUR)*, 24(2):131–183, 1992.

[10] D. Lange and D. Chang. IBM Aglets Workbench—Programming Mobile Agents in Java. *White Paper, IBM Corporation, Japan, August*, 1996.

[11] R. Lee and J. Liu. iJADE eMiner-A Web-Based Mining Agent Based on Intelligent Java Agent Development Environment (iJADE) on Internet Shopping. *Cheung, GJ Williams, and Q. Li (Eds.): PAKDD*, pages 28–40, 2001.

[12] P. Mitkas, D. Kehagias, A. Symeonidis, and I. Athanasiadis. A framework for constructing multi-agent applications and training intelligent agents. *Proc. of the 4th Int. Workshop on Agent-Oriented Software Engineering (AOSE-2003)*, pages 96–109, 2003.

[13] J. Spivey. The Z notation: a reference manual. *Prentice-Hall International Series In Computer Science*, page 155, 1989.

[14] I. Witten and E. Frank. Data mining: practical machine learning tools and techniques with Java implementations. *ACM SIGMOD Record*, 31(1):76–77, 2002.