DXCS: an XCS System For Distributed Data Mining

Hai H. Dam
Artificial Life and Adaptive Robotics Laboratory,
School of ITEE,
UNSW@ADFA
Canberra, ACT 2600, Australia
z3140959@itee.adfa.edu.au

Hussein A. Abbass
Artificial Life and Adaptive Robotics Laboratory,
School of ITEE,
UNSW@ADFA
Canberra, ACT 2600, Australia
abbass@itee.adfa.edu.au

Chris Lokan
Artificial Life and Adaptive Robotics Laboratory,
School of ITEE,
UNSW@ADFA
Canberra, ACT 2600, Australia
c.lokan@itee.adfa.edu.au

ABSTRACT

XCS is a flexible system for data mining due to its ability to deal with environmental changes, learn online with little prior knowledge and evolve accurate and maximally general classifiers. In this paper, we propose DXCS which is an XCS-based distributed data mining system. A MDL metric is proposed to quantify and analyze network load, and study the balance between network load and classifier accuracy in the presence of noise. The DXCS system shows promising results.

Categories and Subject Descriptors: I.2.6 [Learning]: Knowledge Acquisition.

General Terms: Algorithms, Performance, Measurement.

Keywords: Learning Classifier System, XCS, MDL, Distributed Data Mining.

1. INTRODUCTION

In the last few decades, the amount of data in organizations has been doubling every few months. However, extracting useful information from massive databases can be a tedious, time consuming and expensive exercise. John Naisbitt stated that *we are drowning in information, but starving for knowledge* [23]. Data mining, the process of discovering novel patterns in databases [13], assists companies and organizations to discover the tacit knowledge hidden in the overwhelming amount of data.

Patterns in a database can be represented in different forms such as neural networks, decision trees, or rule-based systems. It is easier to understand patterns represented using the latter representation than those for example represented using a neural network. Once a rule-based system is learnt, one can use it to classify new data as they arrive.

Classifier systems and genetics based machine learning (GBML) techniques have been successful in data mining problems due to their ability to do global search [14] and learn with little prior knowledge [16]. Classifier systems fall in two categories: Pittsburgh and Michigan style classifier systems. In the former, an individual in evolution is a complete set of rules while in the latter, the whole population is a complete set of rules and an individual is only a single rule. XCS [25] is the current state-of-the-art GBML technique and is widely accepted as one of the most reliable Michigan-style learning classifier system for data mining. The reasons for the robust performance of XCS are: it is an online-learner, the fitness is based on the accuracy of the prediction reward, and it is able to evolve rules by interacting with the environment [1, 9].

Wilson [26] has shown that the evolved rules of XCS are accurate and maximally general in which the patterns of interest are easily recognized. Moreover, XCS maintains a *complete action map* which is able to adapt quicker to a change in the environment because it only has to relearn the quality of the rule, not the rule itself [4]. Other papers have shown that XCS performs better than other machine learning algorithms such as C4.5 [3, 12].

Nowadays, most databases in large companies are distributed. With the large amount of data generated at each location, it is not possible to transfer all the data to a central location to do data mining. In many cases, it is not feasible to transfer data from distributed sites into the centralized database due to security issues, limited network bandwidth, or even because of the internal policies for some organizations. Distributed Data Mining (DDM) is an extension of data mining techniques in distributed environments. Even if the data is not physically distributed, DDM can be used effectively in speeding up the data mining process. However, the primary purpose of DDM is to discover and combine useful knowledge from databases that are physically distributed across multiple sites [24].

Many researchers have been extending traditional data mining approaches to distributed environments. However, most of the traditional data mining methods are off-line techniques and thus require initial time for training. Also, they take time to recover from changes in the environment. In this paper, we will discuss an extension of the XCS approach [25] for DDM.

We will adopt an ensemble method called *knowledge probing* [18, 19] in order to build the global view of the distributed environment.

The effect of noise is a major issue in data mining. Therefore, we also investigate the system's performance in the presence of different noise levels. We also compare our performance against the performance of a centralized XCS system with different levels of noise.

The paper is structured as follows. The next section pro-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
GECCO'05, June 25–29, 2005, Washington, DC, USA.
Copyright 2005 ACM 1-59593-010-8/05/0006 ...$5.00.

vides a brief overview of DDM. Section 3 introduces XCS then the distributed system is introduced in Section 4. The data transmission in the network is investigated in Section 5. The experiments and results are discussed in Sections 6 and 7 respectively. Conclusions and directions for future work are presented in Section 8.

2. RELATED WORK

DDM is a relatively new area but has been receiving much attention, especially in distributed environments where trust between sites is not always complete or mutual [20]. In many applications, data are privacy-sensitive, so that centralizing the data is usually not acceptable [15]. Therefore, DDM technology needs to be adopted in these applications to reduce the transmission of raw data and thus protect raw data.

Data in DDM can be divided into two categories: homogeneous and heterogeneous. In homogeneous DDM, the databases located at different sites have the same attributes in the same format, while in heterogeneous DDM, the attributes at each site are different or in different format. The focus of this paper is on homogeneous DDM.

In homogeneous data, a large fraction of DDM approaches focuses on centralized ensemble-based methods [22], which first form local models at the local sites and then combine these models at the centralized site. For example, a company may have different branches. Each local model represents an independent data mining for each branch, then the central office of the company combines the models sent by each branch to gain an overall view of the company as a whole. Giannella et al [15] state two main advantages of DDM using ensembles. The first advantage can be obviously seen when the local model is much smaller than the local data: sending only the model thus reduces the load on the network and the network bandwidth requirement. The second one is that sharing only the model, instead of the data, gains reasonable security for some organizations since it overcomes issues of privacy.

Breiman [6] proposed *bagging* to aggregate models. The *bagging* approach was initially used in [5] to increase the accuracy of a learning algorithm by averaging the outputs from different models generated with different data subsets (not necessarily distinct).

Chan and Stolfo [11] introduced another approach for combining models called *meta-learning*. In this approach, each local site may employ a different inductive learning algorithm for learning its local model. A meta-classifier is trained using data generated by the local models. This process is applied recursively to produce an arbiter tree, which is a hierarchy of meta-classifiers. Meta-learning has been used for fraud detection in the banking domain.

An alternative to *meta-learning* is the *knowledge probing* method developed by Guo and Sutiwaraphun [18]. This method is similar to the meta-learning but it does not build an arbiter tree. Instead, one descriptive model is generated from the predictions of local models from a distributed environment.

Both *meta-learning* and *knowledge probing* methods were applied to traditional inductive learning algorithms such as decision tree learning: ID3, CART, C4.5 ; memory-based learning: WPEBLS; Bayesian learner based on conditional probabilities: BAYES, etc [19, 24]. These approaches are efficient and effective. However, they are normally used as off-line learning mechanisms, where the training of the model needs to be completed before one can use the model for mining. Moreover, the model structure can be sensitive to small modifications in the data.

Evolutionary algorithms are natural candidates for ensemble learning approaches because they can adapt quickly to changes in the environment. Several researchers are working on applying evolutionary algorithms for ensemble learning [17, 21]. However, in traditional ensemble learning, the members of the ensemble are usually trained on the same datasets. Moreover, the training is usually done in batch off-line mode. A more specific implementation of classifier systems for distributed environments was examined in [7], where Bull et al developed a distributed control system for traffic signal. The system simulates the distributed system where each traffic light is operated by a classifier system and the time needed for an object to travel is investigated.

However, the previous studies were not focused in particular on distributed data mining. In this paper, we propose the evolutionary based online-learning system called XCS, in conjunction with the *knowledge probing* technique, for DDM. XCS is a genetic based machine learning algorithm that applies a reinforcement learning (RL) scheme. Thus much of its theory is inherited from the RL literature. The following section gives a brief overview of XCS.

3. XCS DESCRIPTION

A population in XCS is a collection of rules or classifiers, in which each classifier consists of two components: *Condition C* and *Action A*. Both the *Condition* and *Action* parts are implemented in a binary representation with the use of wildcards or *don't care* symbol in the *Condition* as a generalization mechanism. XCS is able to handle both single- and multiple-step tasks, but this paper focuses on single-step tasks only.

Each classifier is associated with three main parameters as follows:

- *Prediction p*: an average of the payoff received when the action associated with the classifier is chosen.
- *Prediction error* ε: a measure of the error in the prediction parameter.
- *Fitness F*: an inverse function of the prediction error - the lower the prediction error, the higher the fitness.

A classifier in XCS is a macro–classifier, which contains a distinct combination of the *Condition* and *Action* parts in the population. Whenever a new classifier is introduced, the population is scanned to see if the new classifier already exists. If so, the new classifier is not added to the population, but a *numerosity* parameter of its copy in the population is incremented by one; otherwise, the new classifier is added to the population and its *numerosity* parameter set to 1.

Given an input, the match set $[M]$ is formed as a collection of classifiers in the population whose conditions match the input. The system then forms a system prediction set $[P]$ for each action that appears in $[M]$ using a fitness–weighted average of the predictions. The action with the largest prediction is selected and exported to the environment. The action set $[A]$ is then formed of the classifiers in $[M]$ that have the selected action.

XCS is an accuracy–based GBML, in which a new fitness is measured based on the accuracy of the classifier instead of

the reward itself from the environment. After receiving the reward R from the environment, XCS updates the values of prediction p, prediction error ε, and fitness F parameters of each classifier appeared in the current action set $[A]$.

XCS applies GA for introducing a new rule into the population. During the selection process, two parents from the action set $[A]$ are selected with probability proportional to their fitness. Two offspring are generated by reproducing, crossing–over, and mutating the parents. Parents continue to stay in the population competing with their offsprings. If the population size is less than a certain number, offsprings are inserted into the population; otherwise, two of the most inaccurate classifiers are deleted from the population before the offspring can be inserted.

4. THE DXCS FRAMEWORK

In this section we describe the main features of our proposed DXCS system. The system consists of a number of clients and a server. A client is placed at each distributed site and is responsible for gathering local knowledge and transferring information back to the server. The server combines the information from the clients to form a descriptive model for the global environment.

4.1 Distributed Sites

Each client has a complete XCS that is trained independently using the local database. The population of XCS starts from an empty set and keeps evolving during the training. A training instance is input to the local XCS, which will choose a suitable output. The output from XCS is then compared against the real class of that input instance. If the system predicts correctly, that training instance is discarded; otherwise, it is kept in a memory so that it gets transferred to the server at a later stage.

The client's models are then sent to the server to be aggregated. The server has its own XCS. For training the server, each client sends also a small sample of its local data in addition to its own misclassified cases.

The transmission of information between the clients and the server consists of:

- **The most updated rules - population of each client's XCS:** The XCS population is the collection of rules, which take the form of a set of $condition \rightarrow class$. Each rule is also associated with several parameters. Therefore, the rules and their parameters represent the complete view of the client of the local data to the server.

  There are two possibilities for sending the local XCS model to the server. The first is by sending the whole model. The second is by sending a partial model only. The problem with the second is to decide which information to send and which not to send. The trade–off between the performance of the server and the data transferred to the server will be discussed later in this paper.

- **Misclassified instances**. Any training instances which were misclassified by the local models are sent also to the server. These instances are used to train XCS at the server so that the gate is trained to deal with difficult problems. Because the population starts from an empty set so that it is totally inexperienced, we can expect that the misclassified instances are reduced as training proceeds. The more training the model receives the more it becomes accurate and thus the less misclassified instances get sent to the server.

- **Unused training instances**. The misclassified instances generated at the clients may have been produced because of noise in the data. Hence, sending the misclassified instances alone to train the server may actually cause problems for the server. Therefore, we also send a few instances not used for training at the clients to the server. The unused instances and the misclassified instances from all clients generate the training set for the server.

4.2 Server Site

The server holds copies of all clients' models, then trains a gate using another XCS. The server combines the misclassified and untrained instances from clients and uses them as a training set.

After receiving the updated models from all clients, the server applies the *knowledge probing* approach [18] to combine the local models. The key idea of this approach is to derive a descriptive model using the output of the local models. All training data received by the server are used as inputs for all copies of local models available at the server, then the server trains an XCS to learn the mapping between the output of these local models and the target class. In other words, the training instances for the server's XCS are created online at the server by all local models. In order to reduce confusion in the rest of this paper, we call the samples for training XCS at the server as the ensemble training set and the training samples get transferred from local clients to the server as the server training set. Each server's training instance is inserted to each local XCS and the class exported out of the model becomes an attribute of the ensemble training instance. Figure 1 shows the architecture of the server and the process to create a new ensemble instance.

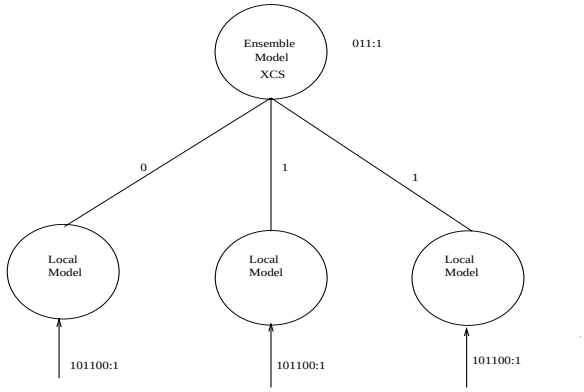

Figure 1: Server architecture with 3 local models. 101100 **is a server training instance to input into each local model.** 1 **is a target class associated with that training sample. Outputs from local models are** $0; 1; 1$**. Thus** $011 : 1$ **is an ensemble training instance.**

The algorithm at the server can be described as follows:
Inputs:

- A set **T** of **n** server training instances
  $T = \{(t_1, c_1); (t_2, c_2); ..; (t_n, c_n)\}$ where t is a training instance and c is an associated class.
- A set **L** of **m** local XCS models $L = \{l_1, l_2, .., l_m\}$
- A learning algorithm XCS, which provides the descriptive output and represents the global view of the database.

Prediction Phase: Obtain outputs from each model in **L** for each data item in **T** and form an ensemble training instances. The outputs of the n^{th} instance t_n from set **L** models are a set O_n: $O_n = \{o_{n1}, o_{n2}, .., o_{nm}\}$. The set O consists of the ensemble training instances for XCS $O = \{O_1, O_2, ..O_n\}$. The new server training set becomes $S = \{(O_1, c_1); (O_2, c_2); ...; (O_n, c_n)\}$

Learning Phase: Learning from data entries in the server training set **S**, $L* = XCS\{(O_1, c_1); (O_2, c_2); ...; (O_n, c_n)\}$

Output: descriptive model L* obtained from the learning phase.

5. FORMALIZING THE DATA TRANSMISSION

The complexity of the data transmitted between the clients and the server can be estimated by the MDL principle, a formula used to evaluate the complexity and accuracy of the model in terms of data compression. We modify the MDL formula discussed by Bacardit [2] to our system. In Bacardit's system, the sender and the receiver exchange training examples in the same order because each of them have identical copies of the data. Therefore, all they need to exchange is the set of indexes. However, this is not the case in distributed data mining, where the data actually travel from the clients to server. Hence, we updated the MDL formula by Bacardit to accommodate the specific features of DDM.

The data needed for the transmission between the client and the server includes the model, misclassified samples, and training samples. Therefore, the cost of transmission is equivalent to the number of bits needed to encode the model (theory bits) plus misclassified and training samples (exception bits).

$$\text{MDL} = \text{theory bits} + \text{exception bits}$$

The length of the theory bits (TL) is the length of classifiers travelling to the server. The classifiers have a common structure: *Condition* $\longrightarrow$ *Action* and therefore their lengths are defined as follows:

$$TL = \sum_{i=1}^{nr}(TL_i) + \sum_{i=1}^{nr}(CL_i)$$

Where nr is the number of rules of the model; TL_i and CL_i are the length of condition and class respectively. They can be estimated as follows:

$$TL_i = nb$$

$$CL_i = log_2(nc)$$

where nb is the number of bits required to encode a complete set of conditions and nc is the cardinality of the set of possible classes. For example, in the 20-multiplexer problem, $nc = 2$ while $nb = log_2(3^{20})$ since we have an alphabet of 3 symbols $\{0, 1, \#\}$ to encode 20 attributes corresponding to the 3^{20} instances. Therefore, the theory bits are estimated as:

$$TL = nr(nb + log_2(nc))$$

Similarly, the exception part of the MDL principle (EL) is estimated as follows:

$$EL = (nm + nu)(na + log_2(nc))$$

where nm is the number of misclassified samples, nu is the number of unused training samples at the clients and na is the number of bits required to encode a training example.

Thus, the length of data sent from one client to its server is:

$$MDL = (nr)(nb)+(nm+nu)(na)+(nr+nm+nu)(log_2(nc))$$

6. EXPERIMENTS

6.1 Systems Setup

Two algorithms are investigated in a simulated distributed environment with two clients and one server. The first algorithm is our proposed DXCS and the second one is a centralized XCS. The centralized XCS system is equivalent to having one XCS in the server and the clients keep sending all local data to the server.

A complete enumeration of the 20-multiplexer problem is used as the dataset. The multiplexer is widely used in the literature of learning classifiers due to its interesting function properties [8]. The data set is divided into two non-overlapping sets for training and testing. The training set is then divided into two non-overlapping sets for the two clients. The testing set is used for testing at the server.

We compare the performance of DXCS against the performance of centralized XCS with different levels of noise. Three levels of noise are used in the comparison: 0.00 (noise free), 0.10, and 0.20. The noise is incorporated in the input attributes by generating a random number between 0 and 1 according to a uniform distribution for each of the input bits. If the random number is less than or equal to the noise level, the corresponding input is flipped.

In the DXCS version, each XCS at each client is trained online using the noisy training sets. This means that every data instance is presented only once to each local XCS. Transmission between the client and the server occurs using a fixed window. We call the training of each local XCS using each window an epoch. After each epoch, the clients send their updated model, misclassified cases, and server training instances, to the server. The system at the server is trained with the incoming models and data from all clients and tested with the testing set of that epoch. A similar process is followed for the centralized XCS system, where all data (since there is no local learning) get transmitted from the clients to the server. For each noise level, 30 repeats are carried out by varying the noise in the data.

The testing set for each epoch contains 40 instances. The training size is 120 instances (changed later to 1020 in an experiment to learn about the effect of epoch size on system performance and traffic load). One hundred instances (or 1000) are used to train the client; the other $nu = 20$ instances are passed to the server for server training.

6.2 Parameters Setup

The DXCS is setup with two clients: client 0 and client 1. The default parameters [10] for XCS at the clients are chosen as follows: maximum population size $N = 2000$; tolerance error in prediction error $\varepsilon_0 = 10$; learning rate for F, p, ϵ is $\beta = 0.2$; $\alpha = 0.1$; the subsumption threshold $\theta_{sub} = 20$; the probability of using a # in one attribute in covering process $P_{\#} = 0.8$; mutation rate is 0.01; one point crossover with the crossover rate 1.0; θ_{GA} (threshold for GA occurs for the average time since last GA of classifiers in $[A]$) is 25; tournament selection with tournament sizes proportional to the current action set size with $P_{TNM} = 0.4$. The default parameters for XCS at the server are almost similar to the ones setup for XCS at clients except for the population size $N = 100$ and $P_{\#} = 0.3$ due to the smaller size of the search space. The centralized XCS is setup by the same parameter values of XCS used at the clients of DXCS.

Each XCS starts with an empty population at the initial step. The covering process occurs at the beginning to introduce new classifiers. The initial value of parameters associated with the new classifiers are set as follows: the fitness F is 0.01; prediction p is 10.0; and prediction error ϵ is 1.0. During the reward feedback process, the correct classification results in the payoff of 1000 from the environment while the incorrect one receives the payoff of 0.

7. RESULTS AND DISCUSSION

7.1 DXCS compared to Centralized XCS

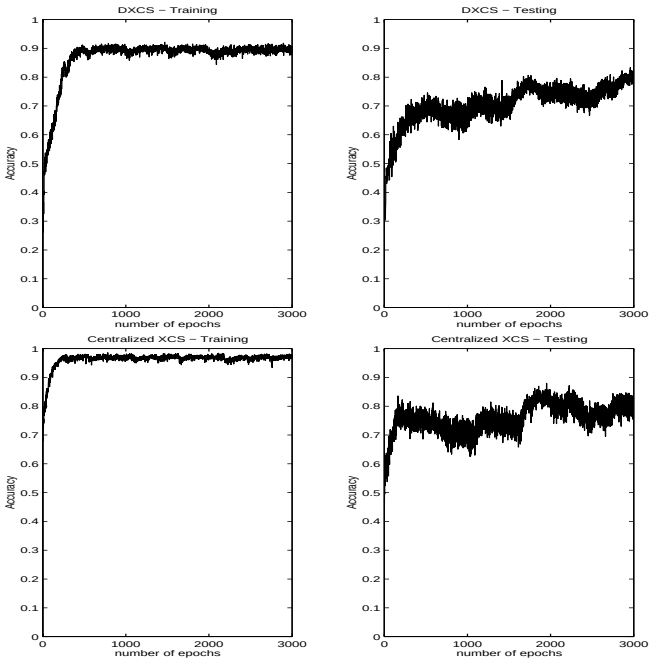

Figure 2: Training and testing performance of the server of DXCS and centralized XCS - Noise Free

In the first experiment, we consider a noise-free environment. Figure 2 shows the training and testing performance at the server of DXCS and the centralized XCS in terms of the accuracy. The results showed in this figure as well as other figures in this paper are averaged of 30 runs. We visualize the average performance over an entire epoch over time. It is important to emphasize here that the data in each epoch are non-overlapping and training is done in an online mode.

The training accuracy of DXCS is almost as good as the training accuracy of the centralized XCS. During testing, DXCS seems to generalize slower than the centralized XCS at the start of the time interval. This behavior is expected since DXCS needs more time in the beginning to combine correctly the models from the clients. However, with time the testing performance of DXCS achieves the same level as the centralized XCS.

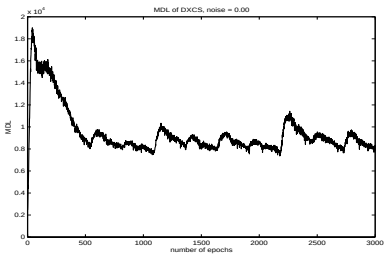

Figure 3: Data transmission from one client to server - Noise Free

In Figure 3 we plot the MDL values over time to measure the data transmission between the clients and the server after each epoch. Because we start with an empty population, the MDL value is high at the start. The covering technique is invoked so often, creating new classifiers to match each of the previously unseen data. During this stage, which we will call the heating-up stage, the population is full of inaccurate macro-classifiers and many misclassified instances are sent to the server. As training proceeds, the fitness pressure of XCS pushes its population towards accurate and general classifiers. There are fewer misclassified instances, and the model becomes smaller as the system discards inaccurate and specific classifiers. After about 500 generations, MDL seems more stable.

Overall, the results show that DXCS is very competitive when compared with the centralized XCS in a noise free environment. Data transmission in DXCS is high to start with, but stabilizes rapidly.

7.2 Effect of Epoch Size

We have shown that data transmission in DXCS decreases over time. However, if the epoch size is small, we will need to send the model more frequently. In addition, the size of the data used to train the model in each epoch may actually be smaller when compared to the size of the model itself. Therefore, it may sound as though there is not any real saving in terms of network load.

In this subsection, we decided to increase the epoch size; thus, more data is used to train each client before the model get sent to the server. In this way, the model size would be smaller than the size of the data used, and we can also estimate the effect of the epoch size on the performance of the server.

Figure 4 depicts the training and testing performance of DXCS and centralized XCS when the epoch size is increased from 120 to 1020. The training accuracy of DXCS is almost as good as the centralized XCS. The two curves may look

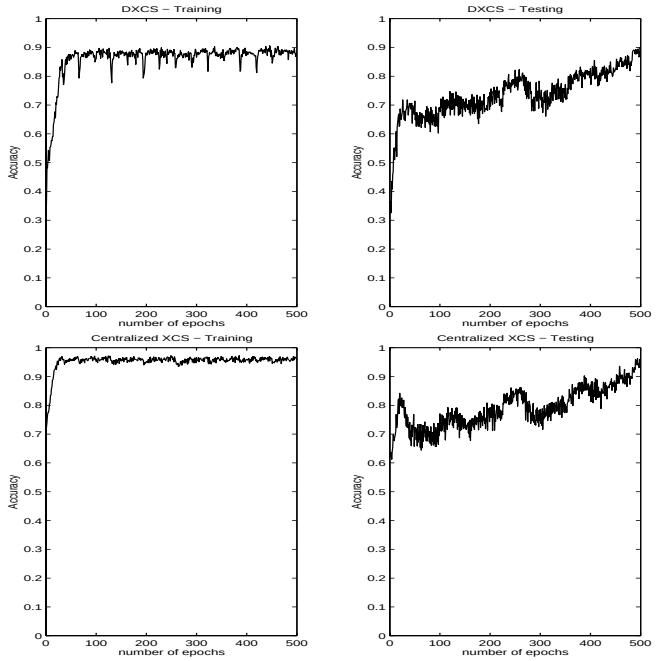

Figure 4: Epoch's size 1000 - Training and testing performance of the server of DXCS and centralized XCS - Noise Free

smoother than the corresponding ones with epoch size of 120 because we now have less number of epochs and we average over more data. The real differences are not much when we inspected the data. In terms of generalization, however, it seems that both the decentralized and centralized versions improved their generalization when the epoch size increase. This is logical since each model is exposed to more data before it is sent to the server, allowing it to learn better and improve its generalization.

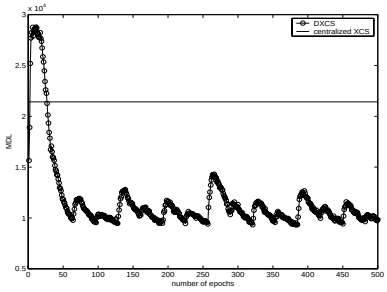

Figure 5: Epoch's size 1000 - Data transmission from one client to server - Noise Free

Figure 5 shows the MDL of both DXCS and centralized XCS with the new epoch size. Since centralized XCS sends a constant amount of data to the server (as we assume data arrives at the client and is passed onto the server at a constant rate), its curve is a horizontal line. Data transmission in DXCS quickly drops below that line, and settles at less than half the data transmission for the centralized XCS. The results show that an increase in the epoch size improves generalization and reduces data transmission. However, we cannot generalize these findings to other problems.

Increasing the epoch size means the server is updated less frequently, delaying its response to changes in the environment. The trade-off between accuracy, data transmission, and up-to-date server will vary depending on the nature of the problem we are dealing with.

7.3 Effect of Noise

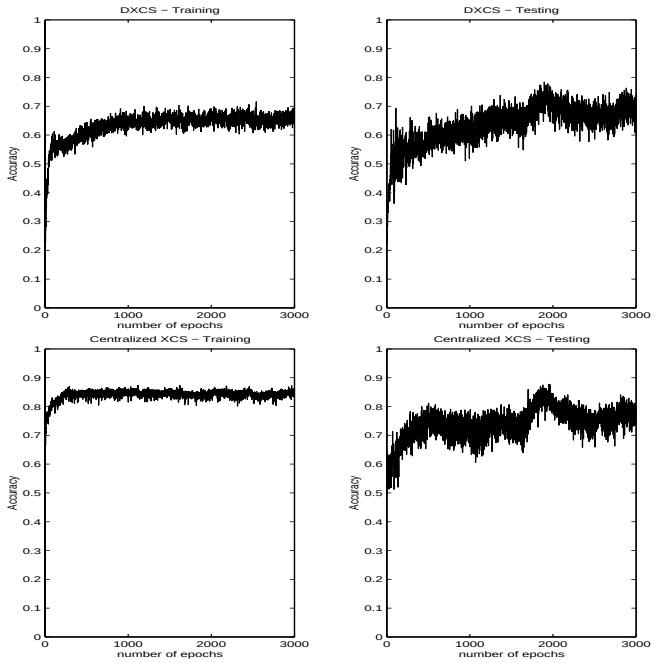

Figure 6: Training and testing performance of the server of DXCS and centralized XCS - Noise 0.10

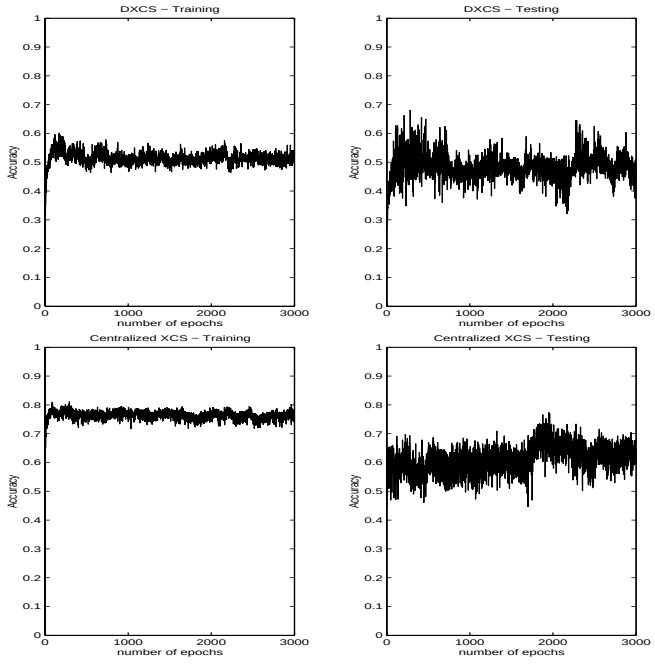

Figure 7: Training and testing performance of the server of DXCS and centralized XCS - Noise 0.20

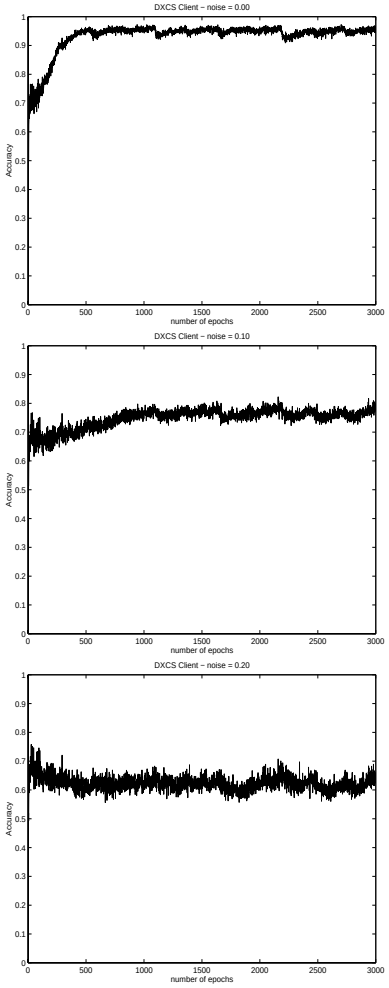

Figure 8: Training performance of the client of DXCS - Noise 0.00, 0.10, 0.20

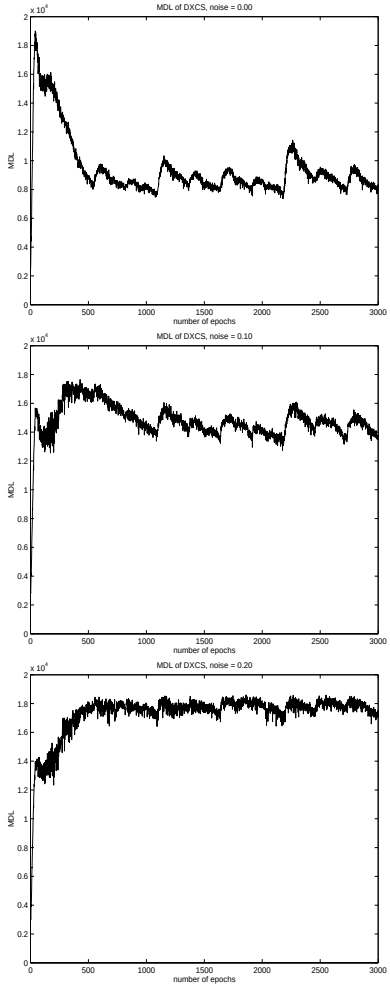

Figure 9: Data transmission from the client to server - Noise 0.00, 0.10, 0.20

This section will examine the effect of a Gaussian noise added to the training set. We observe that when noise is small (noise = 0.10) (Figure 6) the performance of DXCS is acceptable compared to centralized XCS . However, when the noise level is high (noise = 0.20), centralized XCS definitely shows better performance than DXCS in Figure 7. It can be explained by viewing the performance of each client of DXCS in Figure 8. When noise is increased, learning time required for XCS increases. Since the test sets are unchanged, the performance of XCS is decreased. Centralized XCS is always trained with twice the data as one client of DXCS receives. As a result, centralized XCS learns quicker than any of the clients could. Since XCS at the clients has not learnt enough, the performance of the server is inferior to the corresponding performance of the centralized XCS.

Data transmission, Figure 9, to the server also increases as the noise increases. This reflects the load resultant from transferring more misclassified instances as well as an increase in the size of the model.

Overall, the performance of DXCS is competitive to the centralized XCS when the noise level is small. When we increase the level of noise in the data, DXCS needs more time to recover.

8. CONCLUSION

This paper has introduced DXCS, an extension of XCS for distributed data mining. In order to validate DXCS, we have compared the system with a centralized XCS. We also examined the effect of noise and epoch size on the system's performance. The results reveal that DXCS is competitive as a distributed data mining system. Moreover, DXCS would be favored as the epoch size increases.

In our future work, we will be looking at three issues. First, the case where clients send only partial models. There are different ways to transmit partial models, which will improve more the time and bandwidth requirement for data travelling to the server. Second, we will also look into how the performance and traffic load of the system vary as we increase the number of clients. Finally we will look at the issue of improving the generalization of the server. In the current paper, the inferior generalization of the server as compared to the centralized XCS can be attributed to the gate we use at the server level. Since the gate is simply deciding between the decision of each client, its generalization could be improved if we allow it also to encode some of the data directly or if we improve the knowledge fusion procedure.

9. ACKNOWLEDGEMENT

This research has been supported by the Australian Research Council Linkage grant number LP0453657.

10. REFERENCES

[1] J. Bacardit and M. V. Butz. *Data Mining in Learning Classifier Systems: Comparing XCS with GAssist*, June 2004. IlliGAL Report No. 2004030.

[2] J. Bacardit and J. M. Garrell. Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system. In *Sixth International Workshop on Learning Classifier Systems (IWLCS-2003)*, Chicago, July 2003.

[3] E. Bernadó, X. Llorà, and J. M. Garrell. XCS and GALE: a Comparative Study of Two Learning Classifier Systems with Six Other Learning Algorithms on Classification Tasks. In *Proceedings of the 4th International Workshop on Learning Classifier Systems (IWLCS-2001)*, pages 337–341, 2001. Short version published in Genetic and Evolutionary Compution Conference (GECCO2001).

[4] E. Bernado-Mansilla and J. Garrell-Guiu. Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks. *Evolutionary Computation*, 11(3):209–238, 2003.

[5] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[6] L. Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1–2):85–103, 1999.

[7] L. Bull, J. Sha'Aban, A. Tomlinson, P. Addison, and B. Heydecker. Towards distributed adaptive control for road traffic junction signals using learning classifier systems. In L. Bull, editor, *Applications of Learning Classifier Systems*, pages 276–299. Springer-Verlag, 2004.

[8] M. V. Butz. *Rule-Based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction*. PhD thesis, University of Illinois at Urbana-Champaign, 2004.

[9] M. V. Butz, D. E. Goldberg, and K. Tharakunnel. Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*, 11(3):239–277, 2003.

[10] M. V. Butz and S. W. Wilson. An algorithmic description of XCS. In *IWLCS '00: Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems*, pages 253–272. Springer-Verlag, 2001.

[11] P. K. Chan and S. J. Stolfo. Toward parallel and distributed learning by meta-learning. In *Working Notes AAAI Work. Knowledge Discovery in Databases*, pages 227–240, Washington, DC, 1993.

[12] P. W. Dixon, D. Corne, and M. J. Oates. A preliminary investigation of modified XCS as a generic data mining tool. In *IWLCS '01: Revised Papers from the 4th International Workshop on Advances in Learning Classifier Systems*, pages 133–150. Springer-Verlag, 2002.

[13] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, pages 1–36. The MIT Press, 1996.

[14] A. A. Freitas. A survey of evolutionary algorithms for data mining and knowledge discovery. *Advances in evolutionary computing: theory and applications*, pages 819–845, 2003.

[15] C. Giannella, R. Bhargava, and H. Kargupta. Multi-agent systems and distributed data mining. In *Cooperative Information Agents VIII: 8th International Workshop, CIA 2004*, pages 1–15, Erfurt, Germany, 2004.

[16] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addision-Wesley Publishing Company, INC., 1989.

[17] C. Guerra-Salcedo and D. Whitley. Genetic approach to feature selection for ensemble creation. In *GECCO'99*, pages 236–243, 1999.

[18] Y. Guo, S. Rueger, J. Sutiwaraphun, and J. Forbes-Millott. Meta-learning for parallel data mining. In *Proceedings of the Seventh Parallel Computing Worksop*, 1997.

[19] Y. Guo and J. Sutiwaraphun. Distributed classification with knowledge probing. In *Advances in Distributed and Parallel Knowledge Discovery*, pages 115–132. AAAI Press/ The MIT Press, 2000.

[20] C. Jones, J. Hall, and J. Hale. Secure distributed database mining: Principles of design. In *Advances in Distributed and Parallel Knowledge Discovery*, pages 277–294. The MIT Press, 2000.

[21] K. Jong, J. Mary, A. Cornuejols, E. Marchiori, and M. Sebag. Ensemble feature ranking. In *Proceeding ECML-PKDD'04*, Pisa, Italy, 2004. IEEE.

[22] L. I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley, 2004.

[23] J. Naisbitt. *Megatrends : Ten New Directions Transforming Our Lives*. Warner Books; Rei edition, 1988.

[24] A. L. Prodromidis, P. K. Chan, and S. J. Stolfo. Meta-learning in distributed data mining systems: Issues and approaches. In *Advances in Distributed and Parallel Knowledge Discovery*, pages 81–114. The MIT Press, 2000.

[25] S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.

[26] S. W. Wilson. Generalization in the XCS classifier system. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674, University of Wisconsin, Madison, Wisconsin, USA, 22-25 1998. Morgan Kaufmann.