Parallel and distributed data mining: the FastWeka tool

Cristian Cosmoski Rangel de Abreu, Marcio Augusto de Souza, Lilian Tais de Gouveia, Luciano José Senger

Universidade Estadual de Ponta Grossa, Ponta Grossa, PR, Carlos Cavalcanti Av., 4748, 84030-900, phone +55 42 3220 3097, Brazil. E-mails: cristiancra@gmail.com, msouza@uepg.br, ltgouveia@uepg.br, ljsenger@uepg.br

Abstract: Data mining refers to the process of extract useful information and knowledge from a given data set, using statistic techniques and machine learning algorithms. Due to the huge size of data and amount of computation involved in data mining, it is very difficult, using current data mining tools, for a single computer to efficiently deal with large data. In this scenario, parallel computers and distributed systems can be used to speed up the data mining process. This paper presents the FastWeka, a tool for speedup data mining tasks, using multicore computers and a peer-to-peer system as computing platform. By exploiting the inherent parallelism of the data mining cross-validation phase (using k-fold technique), Fastweka can achieve an improvement in the speed of data mining. Aiming to evaluate the tool, a forest cover dataset composed of 55 attributes and 581,012 records was considered as input of data mining algorithms. The computing times obtained when using FastWeka reveals a speedup of 9 when using 10 folds and 10 processing elements, without jeopardizing the classification accuracy. The experiments also show that better speedup values are obtained when the number of folds is multiple of the quantity of available processing elements and when it is processed only 1-fold per computer of a peer-to-peer system.

1 Introduction

Data mining (DM) consists of a set of techniques for the analysis and extraction of relevant information from data sets, generating knowledge by means of patterns and trends observed in these data. For instance, agricultural data can be used as input to build descriptive and predictive models. Predictive or classification models can be used for building classification systems for fruit or soil; climate forecast information; analysis of vegetation types; detection of diseases and pests. In fact, several works have considering DM techniques as a solution for agricultural data analysis.

Frequently, DM involves developing a classification model, through a process called training, which distributes data into a set of some predetermined classes called examples or samples. The accuracy of the classifier is measured by using cross-validation, which is a largely used and reliable technique for estimating the accuracy of a classification model [1]. Cross-validation is based on the concept of partitioning the database into mutually exclusive subsets called *folds*. In the *k-fold* cross-validation, the original data is randomly partitioned in *k* equal sized subsamples. The *k*-1 subsamples are used for model training and a single subsample is reserved as the validation data for testing the model. This process is repeated *k* times, varying in each repetition the fold used for validation. A mean value, calculated from the results obtained in each of the *k* steps, is used for estimating the quality of the model (mean prediction error). The advantage of the k-fold cross-validation is that all observations are used for both training and validation, and each observation is used for validation exactly once. The number of folds commonly used is 10, but in general, the value of *k* is empirically chosen.

The application of data mining algorithms requires the use of software tools. Among all data mining software, the Weka (*Waikato Environment for Knowledge Analysis*) is a commonly adopted open source software solution to data mining. The Weka software is a collection of machine learning algorithms for data mining tasks. The algorithms included in the Weka software package can be applied directly to a dataset to extract useful information and to build descriptive and predictive (classification) models. Weka contains tools for data pre-processing, classification, regression, clustering, association rules and visualization.

Data mining software response times are affected by the amount of data stored in the database and by the DM algorithm class. Works have shown that DM very often requires high processing times, and it is possible to speed up the data mining using parallel computing. Parallel computing is the simultaneous and the coordinated use of multiple compute resources to solve a computational problem [2]. The compute resources are typically a single computer with multiple processor (or multiple cores) or computers connected by a network. When using computers connected by the Internet, Peer-to-peer (P2P) systems has emerged as an attractive alternative for building systems that provide efficient use of resources, scalability and self-organization [3]. Basically, the P2P model defines a distributed system as a cooperative set of computers, called peers, which share some of their resources with other peers in the system without the need for a central server. Through the sharing of processing elements between the different peers, it is possible to run parallel applications that are flexible and that can scale up by increasing the availability of computing peers.

Aiming to speedup data mining computing times, this work presents FastWeka. FastWeka extends the Weka software using distributed and parallel computing techniques to conduct data mining using multicore computers and a P2P system. By exploiting the inherent parallelism of the data mining cross-validation phase (using k-fold cross validation), Fastweka can achieve a significant improvement in the speed of data mining tasks. Also, by using a P2P system, new threads can be created in computers connected by the Internet, regardless of the presence of firewalls separating the computers. These advantages of P2P systems are directly inherited by FastWeka.

This paper presents the implementation details and the performance obtained from the execution of FastWeka in multicore computers and a P2P system. In the section 2, related work is presented. The FastWeka tool implementation details is presented in the section 3. The evaluation of FastWeka in an agricultural data set is presented in section 4. The conclusions of this work are presented in the section 5.

2 Related work

While can be used for mining data from all domains, FastWeka was designed as a tool for agricultural data mining. Many authors have addressed agricultural data analysis. For instance, in [4], a classifier for Egyptian rice diseases is presented. In [5], DM techniques are considered to improve climate predictions. Also, the application of DM on large data sets typically involves large computational cost, as showed in [6] and [7]. Furthermore, the use of parallel computation to assist the mining of a large number of agricultural data has been explored in [8] [9] [10].

Authors also have addressed how to speed up the DM process. Pimenta *et al* [11] presented and evaluated a Weka tool extension for parallel data mining in computational grids [12]. Talia *et al* [13] developed and evaluated the Weka4WS software, which uses a Web

Services Resource Framework and allows remote execution of Weka DM algorithms. In a similar study, Zuo [16] presents a tool called Weka-Parallel, used for parallel cross-validation using TCP communication. More recently, in [14], a distributed framework for Weka is presented. The framework is implemented on top of Spark, a Hadoop-related distributed framework for big data computing.

FastWeka follows the inspiration from these tools, but it uses a different approach. The system is totally based on threads and does not depend on any software infrastructure to execute, as in [11] and [13]. The FastWeka details are presented in the following section.

3 The FastWeka Tool

FastWeka is a parallel implementation of Weka data mining tool [1], version 3.6.4, developed by the research group in machine learning at the University of Waikato, New Zealand [1]. Weka is an open source software written in Java that can be executed on different operating systems (Linux, Windows and Macintosh) and is distributed under the terms of the General Public License - GNU GLP. Weka implements a set of machine learning algorithms for DM tasks, including algorithms for data preprocessing, post-data processing, classification, regression, clustering and visualization.

FastWeka can operate in two modes: multi-threads, which uses different cores from a multi-core processor; P2P, which allows the use of computers distributed over the Internet. To implement FastWeka, it was necessary to modify the ClassifierPanel class, which belongs to the weka.gui.explorer package of Weka. In addition, the RunLocal and RunParallel classes were added and they are responsible for the multi-threads and P2P modes, respectively. Additionally, to implement the P2P mode, it was necessary to create classes that interact with P2PComp, on which FastWeka is executed.

The FastWeka are implemented using the P2PComp framework [3]. The P2PComp was developed using the JXSE library (an implementation of the JXTA standard [15] and it provides: an infrastructure for starting and monitoring parallel applications written in Java and a programming library that is included in the source code of the programs executed by the framework. P2Pcomp follows the pure P2P model: all peers have the same functionalities, so a peer can act as both scheduler, worker or monitoring peer. The P2P network components are implemented in Java classes named `Peer`, `Discovery` and `IPC` (See Figure 1). The `Peer` class initiates the execution of JXTA protocols and implements the framework protocol messages. A framework message follows a standard format defined in `Message` class and it is transmitted using JXTA sockets. The `Discovery` class organizes the advertising mechanism by publishing information about the peers and searching the advertisements of other peers. In a regular time interval, the P2P system is searched for advertisements of new peers. The information about all the discovered peers is stored in a data structure managed by the `Cache` class. The `IPC` class is responsible for the inter-process communication routines available for parallel applications.

[1] Available in: <http://www.cs.waikato.ac.nz/ml/weka/>

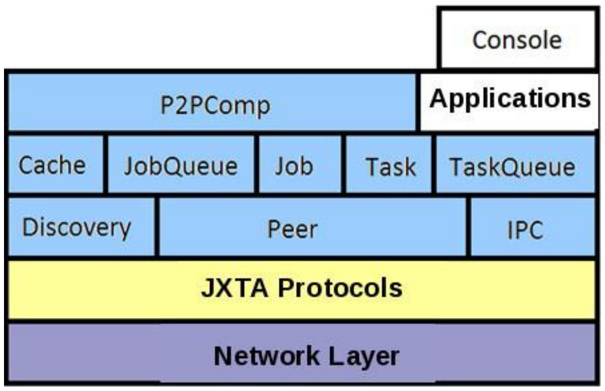

Figure 1: Main components of the P2PComp framework

The `P2PComp` defines that a parallel application is composed by cooperating tasks that execute the same source-code, according to the SPMD (single program multiple data) model. Also, the parallel tasks can exchange information by message passing. Each task of the application is identified by an integer number ranging from 0 to the total number of tasks. The task identifier is called *rank*, and it is used to specify the recipient when communicating with another task.

After choosing the group of peers that will execute the application, the framework sends the application encapsulated in a serialized Java class to all peers that will execute it. In the context of `P2PComp`, a parallel task is called a task, and the peer that initiates the execution maintains a copy of the tasks sent to other peers in a class named `RemoteJob`.

When a peer receives a new task, it stores the task in a queue defined in the class `JobQueue` and waits for configuration messages that inform the relation between application tasks and *ranks*. The *ranks* are used for exchanging messages among the tasks, and the peer must know the *ranks* of the tasks that it is executing.

For the implementation of FastWeka, the parallelism was applied on the cross-validation process. As discussed in the previous section, this technique performs *k* steps, which do not depend on each other and have a high computational cost. By distributing these steps to several threads, it is possible achieve great performance improvement.

During the execution of FastWeka, the number of steps and threads are defined by the user when the ClassifierPanel class is executed. In multi-threads mode, the threads are mapped to processor cores, using the RunLocal class. In P2P mode, the RunParallel class is activated and distributes the threads to the processing elements that are available on the P2P network.

4 Evaluation

For evaluating the FastWeka tool classification performance, a data set[2] with 581,012 records which describes a forest cover vegetation was used as input data. Each record contains 55 attributes, such as: elevation, aspect and slope of the land, solar incidence, distance to water and vegetation type. This data set was obtained from the areas of Rawah (29,628 hectares), Comanche Peak (27,389 hectares), Neota (3,904 hectares) and Cache la Poudre

[2] BLACKARD, J. A. Covertype Data Set. fev. 2012. Available in: <http://archive.ics.uci.edu/ml/datasets/Covertype>

(3,817 hectares), located approximately 113 km northwest of the city of Denver, Colorado, United States, in the Roosevelt National Forest [6]. These forest areas were selected because they remain relatively unchanged, with little human influence, and they represent a composite of forest cover types generated from natural evolutionary processes.

The DM processing performed in this work is made on the "type of forest cover" attribute, which is defined as a target attribute and can have 7 possible values: spruce/fir, lodgepine pine, ponderosa pine, cottonwood/willow, Douglas-fir, aspen and krummholz. The krummholz cover type is composed of Engelmann spruce, subalpine fir and Rocky Mountain bristlecone pine. The DM was performed for creating a classifier for forest cover types, to make predictions about the possible coverage that can occur in certain areas according to the measured values of the 55 attributes defined in the original data set.

Initially, all the 581,012 records were grouped into a single data set, from which seven subsets were created, based on the type of forest cover type. From the analysis of the partitioned data set, it was observed that only two types of forest cover accounted for about 85.22% of all data. To create a more homogeneous model, it was determined that the sets of test and validation data would be built from a data set composed by an equal number of observations of all the types of forest cover.

The number of observations of each forest cover type was defined based on the class with least representation in the database, which had 2,747 records. Therefore, 2,747 observations were randomly selected for each type, obtaining a total of 19,229 observations. A supervised classification was performed in the selected data by running a multilayer artificial neural network (ANN), known as MultiLayer Perceptron, generating a model for classifying forest cover types within each of the 7 defined classes.

The parameters used on the ANN algorithm were set according to [6], which defined them through practical experimentation: 1 single hidden layer with 120 neurons; backpropagation algorithm with learning rate of 0.05, momentum rate of 0.5 and number of training epochs equal to 1000. For the cross-validation, it was chosen 2 values for the number of folds: 10, default choice for cross-validation evaluating, and 24, the total number of processing elements used in the experiments. With all the parameters defined, FastWeka tool was executed varying the number of folds and processing elements. The obtained results are shown in the next section.

5 Results and Discussion

The experiments were performed on 6 computers, running Linux with kernel version 2.6.37, with the following configuration: Intel processor core I7 - 2600 3.4 GHz with 4 real and 4 virtual cores with Turbo Boost Technology enabled; memory of 4 GB DDR3; Gigabit Ethernet network interface connected to a local network.

The next subsections present the performance results obtained from the execution of FastWeka, in multi-threads and P2P modes, and the accuracy of the results obtained by the classifier. All the performance values measured were statistically compared, with the intent of proving the real difference between the obtained mean values.

For all the experiments, sequential and parallel runtimes were calculated, and from them it was obtained the speedup. The speedup is used to measure how much a parallel algorithm is faster than a sequential one, and it is calculated by dividing the sequential by the parallel execution time. Ideally, the speedup should have a value close to the number of processing elements used.

5.1 FastWeka using Multi-threading

The experiments showed in this subsection were performed in a single computer, so the maximum number of threads generated is 8, as each CPU have 8 cores and each core executed only one thread. The Table 1 presents the results of the execution of DM using 10 folds and varying the number of threads among 2, 4, 6 and 8. In this case, there was a good performance improvement in all the configurations. For 6 and 8 threads, the performance improvement was less significant, due to the hyperthreading enable processors (the 8-core processor used has 4 physical cores).

Table 1: Multi-threading with 10 folds

	Number of Threads				
	Sequencial	2	4	6	8
Mean execution time	301.92	157.21	98.21	81.44	90.63
Standard deviation	1.05	1.08	0.47	1.05	0.25
Speedup	−−	1.92	3.07	3.71	3.33

A graphical representation of the runtimes measured using 10 folds is showed in the Figure 2. The Table 2 shows the results obtained using 24 folds and 2, 4, 6 and 8 threads. As it can be seen, when compared to the experiment performed with 10 folds, a similar improvement of performance was achieved, but with a better speedup when 4 threads were used. The runtime of a cross-validation step is the same because the folds have the same size. Therefore, 4 cores perform 4 steps at the same time. If 10 folds are processed, then 10 steps are performed. The first 8 steps are processed at the same time (2 steps per core), but the last two will only use 2 cores, underusing the processor. As 24 is a multiple of 4, the division of steps is exact between the different cores, maintaining the processor fully used all the time.

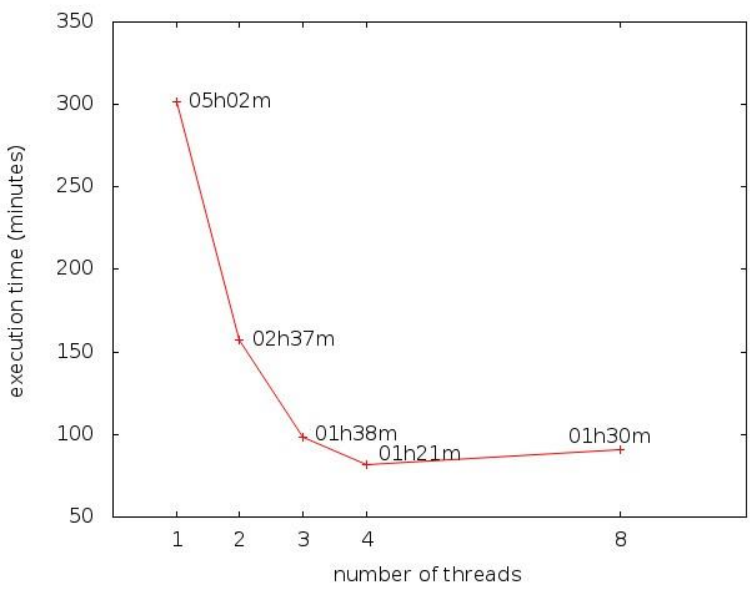

Figure 2: Multi-threading with 10 folds

Table 2: Multi-threading with 24 folds

	Number of threads				
	Sequencial	2	4	6	8
Mean execution time	763.06	408.76	218.09	203.79	192.78
Standard deviation	3.94	1.73	0.44	3.51	1.67
Speedup		1.87	3.50	3.74	3.96

The Figure 3 shows the graphical representation of the measured execution times with 24 folds.

5.2 FastWeka on a P2P network

This section presents the results obtained using FastWeka on P2P mode with different computers. Each component of a P2P network is called a peer, and, in this work, each processing element runs a single (1) P2PComp peer.

When using a P2P network, a peer is responsible for starting the application and distributing the folds to the other peers, because, in the P2PComp computing model, the peer which starts a job cannot execute tasks from this job. It is important to point out that, in all the experiments discussed in this section, one of the peers does not perform any processing. The existence of this additional peer decreases the speedup, but it will not negatively affect the runtime if there are sufficient peers for all the cross-validation steps.

When setting up a P2P network, it is possible to initiate peers on each core of the processor, or it can be defined that each computer will execute just 1 peer. When the processing involves different computers, one issue that may influence the final runtime is the communication overhead generated by the distribution of the folds for all the peers that will participate in the processing.

With the intent of measuring this possible communication overhead, the table 3 presents the results obtained by using 4 processing peers distributed in 4 cores on the same computer and in 4 different computers.

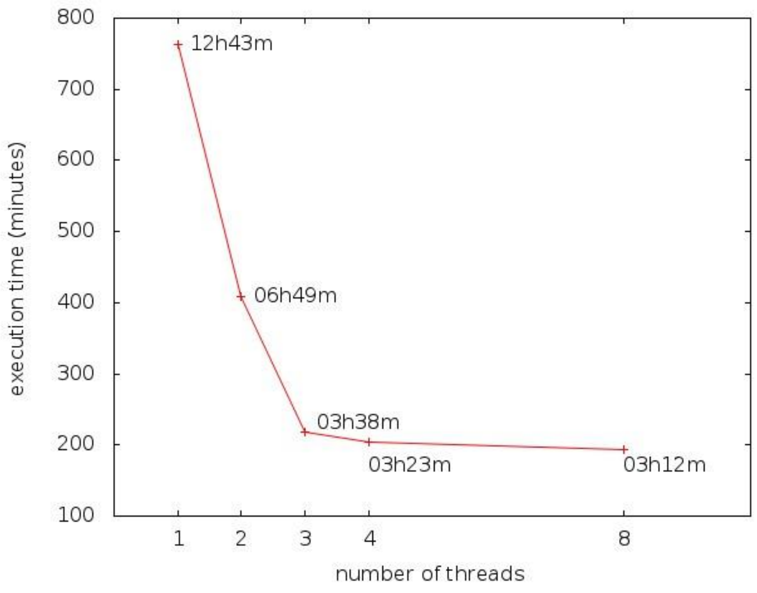

Figure 3: Multi-threading with 24 folds

Table 3: P2P-mode with 10 folds in computers and cores

	Number of threads				
	Sequencial	2	4	6	8
Mean execution time	763.06	408.76	218.09	203.79	192.78
Standard deviation	3.94	1.73	0.44	3.51	1.67
Speedup		1.87	3.50	3.74	3.96

The distribution in different computers, in despite of the communication overhead, allowed better performance than the distribution in different cores. The competition for multiple cores and for the same resources can affect the performance, so it is preferable to use all the available computers before using different cores of the same computer. Table 4 and Figure 4 show the results obtained by using 10 folds and varying the number of computers to 3, 5, 7, 9 and 11. The same experiments were made with 24 folds, and the results are showed in Table 5 and Figure 5.

Table 4: P2P network with 10 folds

Number of Peers	1	3	5	7	9	11
Mean execution time	301.92	146.07	91.22	61.44	60.98	31.97
Standard deviation	1.05	0.88	1.10	0.69	0.75	0.82
Speedup		2.07	3.31	4.91	4.95	9.44

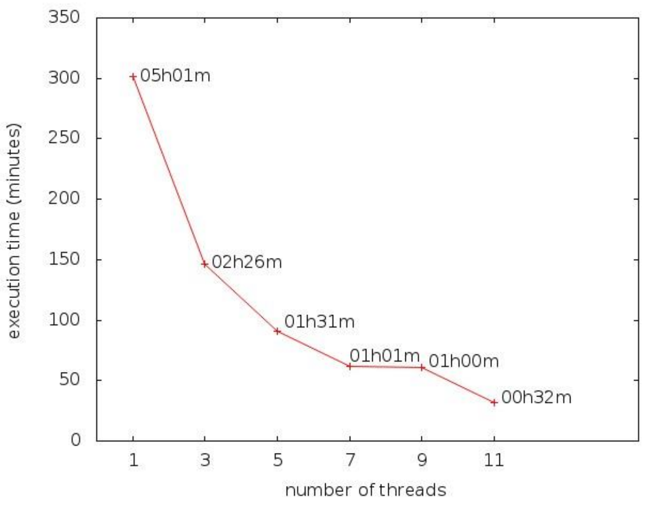

Figure 4: P2P network with 10 folds

All the runtime values obtained in this section (with 10 or 24 folds) show that the inclusion of new peers always improves the speedup, showing that this kind of application can effectively use a computational system that easily scale up.

Table 5: P2P network with 24 folds

Number of peers	1	3	5	7	9	13
Mean execution time	763.06	378.10	192.83	132.24	100.53	72.70
Standard deviation	3.94	3.88	2.87	3.71	1.72	1.23
Speedup		2.02	3.96	5.77	7.59	10.50

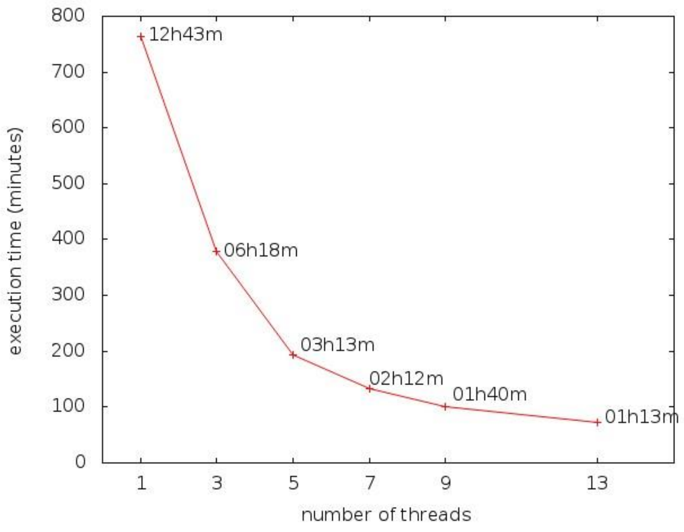

Figure 5: P2P network with 24 folds

In Table 6, is presented the qualitative results of the generated classifier, using 10 or 24 folds, expressed by: accuracy percentage and the Kappa statistics. A classifier is a prediction system, and the Kappa index measures the difference between the predictions made by the proposed classifier when compared to a reference ideal classifier.

Table 6: Qualitative information about the obtained classifier

Number of folds	10	24
Accuracy (%)	82.33%	82.79%
Kappa statistic	0.794	0.7993

The Table 7 shows the measured AUC, that is the area under the ROC (Receiver Operator Characteristic) curve. The ROC curve shows how the number of correctly classified examples varies with the number of incorrectly classified examples. While ROC is a two-dimensional representation of a model performance, the AUC reduce this information into a single scalar. As the name implies, it is calculated as the area under the ROC curve and a perfect model presents an AUC score equal to 1. As it can be seen in the Table 7, for 10 or 24 folds, all results are above 0.9 and near to 1.

Table 7: Area under ROC curve (AUC) and classification precision

Number of folds	10	24
Accuracy (%)	82.33%	82.79%
Kappa statistic	0.794	0.7993

Observing all the quality indexes, it can be noted that the number of folds has no significant influence on the accuracy of the classifier. On the other hand, there is a considerable difference when the runtime is considered. This fact shows that is preferable to use less folds when dividing the data set, as the quality is assured, and the execution time is considerably decreased. The ideal scenario is to use enough computers (peers) so each cross-validation step is processed in a different computer.

6 Conclusions

In a context of a continuous growth of the size of data sets and of the time spent for processing it, this work presented the FastWeka tool, which enables the parallel execution of DM tasks. The use of FastWeka, in multi-threads or P2P mode, allowed a satisfactory decrease in runtime, and the performance evaluation performed in this paper also showed that it is not necessary to divide the datasets in many folds. From experiments results, it is possible to obtain better performance when the number of folds is multiple of the quantity of available processing elements, especially if it is used only 1 core per computer. The classifier for types of forest cover that was created using FastWeka obtained a good level of accuracy, reaching a level of 82.79%, using only 10 folds. As a conclusion, it can be pointed out that is possible to run DM tasks on agricultural data with efficiency, quality and scalability using FastWeka on single computers or P2P systems.

7 Acknowledgments

This paper is based upon work supported by CAPES and Fundação Araucária, Brazil. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the CAPES or Fundação Araucária. The authors would like to thank the High Performance Computing Lab (LCAD) at UEPG.

References

[1] WITTEN, I. H.; FRANK, E.; HALL, M. A. **Data Mining:** Practical Machine Learning Tools and Techniques. 3. ed. Amsterdam: Morgan Kaufmann Publishers Inc., 2011.

[2] WILKINSON, B.; ALLEN, M. **Parallel Programming:** Techniques and Applications using networked workstations and parallel computers. Upper: Pearson/Prentice Hall, 2004.

[3] SENGER, L. J.; SOUZA, M. A.; FOLTRAN, D. C. J. **Towards a Peer-to-Peer Framework for Parallel and Distributed Computing**. Computer Architecture and High Performance Computing (SBAC-PAD), 2010 22nd International Symposium on. [S.l.]: [s.n.]. 2011.

[4] EL-TELBANY, M.; WARDA, M.; EL-BORAHY, M. Mining the Classification Rules for Egyptian Rice Diseases. **The International Arab Journal of Information Technology**, v. 3, p. 303-306, 2006.

[5] PESSOA, A. S. A. **Mineração de Dados Meteorologicos Pela Teoria dos Conjuntos Aproximativos na Previsão de Clima por Redes Neurais Artificiais**. Instituto Nacional de Pesquisas Espaciais. São José dos Campos, SP. 2009.

[6] BLACKARD, J. A.; DEAN, D. J. Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Types From Cartographic Variables. **Computers and Electronics in Agriculture**, v. 24, p. 131-151, 1999.

[7] CHAGAS, C. S. et al. Utilização de redes neurais artificiais na classificação de níveis de degradação em Pastagens. **Revista Brasileira de Engenharia Agrícola e Ambiental**, v. 13, p. 319-327, 2009.

[8] SOUZA, M. A. de ; Abreu, C. C. ; SENGER, L. J. ; GUIMARãES, A. M. **Considerações sobre a utilização de técnicas de computação paralela para melhorar o tempo de resposta da mineração de dados agrícolas**. Congresso Brasileiro de Agroinformáica (SBIAgro). Bento gonçalves, Rio Grande do Sul, Brasil: [s.n.]. 2011.

[9] DAMIATI, F. V. F.; SOUZA, M. A.; SENGER, L. J. **Avaliação de desempenho computacional da mineração de dados pluviométricos com a utilização da computação paralela**. 2 Encontro Anual de Iniciação Tecnologica e Inovacao. Universidade Estadual de Maringá: ANAIS DO 21 EAIC. 2012.

[10] MUCHERINO, A.; PAPAJORGJI, P. J.; PARDALOS, P. M. **Data Mining in Agriculture**. 1. ed. [S.l.]: Springer Publishing Company, 2009.

[11] PIMENTA, A. et al. WEKA - G: Mineração de Dados Paralela em Grades Computacionais. **Revista de Sistemas de Informação da FSMA**, v. 4, p. 2-9, 2009.

[12] MOWBRAY, M. How web community organisation can help grid computing. **Int. J. Web Based Communities**, Inderscience Publishers, Geneva,SWITZERLAND, v. 3, p. 44-54, 2007. ISSN ISSN: 1477-8394.

[13] TALIA, D.; TRUNFIO, P.; VERTA, O. The Weka4WS Framework for Distributed Data Mining in Service-Oriented Grids. **Concurrency and Computation: Practice \& Experience**, v. 20, p. 1933-1951, nov. 2008.

[14] KOLIOPOULOS, A.-K. et al. **A parallel distributed weka framework for big data mining using Spark**. Big Data (BigData Congress), 2015 IEEE International Congress on. [S.l.]: [s.n.]. 2015. p. 9-16.

[15] VERSTRYNGE, J. **Practical JXTA:** Cracking the P2P puzzle. [S.l.]: Lulu, 2008.

[16] ZUO, X. **HIGH LEVEL SUPPORT FOR DISTRIBUTED COMPUTATION IN WEKA**. University College Dublin. [S.l.]. 2004.