

Fastest Association Rule Mining Algorithm Predictor (FARM-AP)

Metanat HooshSadat
Dept of Computing Science
University of Alberta
Edmonton, Alberta, Canada
hooshsad@ualberta.ca

Hamman W. Samuel
Dept of Computing Science
University of Alberta
Edmonton, Alberta, Canada
hwsamuel@cs.ualberta.ca

Sonal Patel
Dept of Electrical and
Computer Engineering
University of Alberta
Edmonton, Alberta, Canada
sonal1@ualberta.ca

Osmar R. Zaiane
Dept of Computing Science
University of Alberta
Edmonton, Alberta, Canada
zaiane@cs.ualberta.ca

ABSTRACT

Association rule mining is a particularly well studied field in data mining given its importance as a building block in many data analytics tasks. Many studies have focused on efficiency because the data to be mined is typically very large. However, while there are many approaches in literature, each approach claims to be the fastest for some given dataset. In other words, there is no clear winner. On the other hand, there is panoply of algorithms and implementations specifically designed for parallel computing. These solutions are typically implementations of sequential algorithms in a multi-processor configuration focusing on load balancing and data partitioning, each processor running the same implementation on its own partition. The question we ask in this paper is whether there is a means to select the appropriate frequent itemset mining algorithm given a dataset and if each processor in a parallel implementation could select its own algorithm provided a given partition of the data.

Categories and Subject Descriptors

I.2 [Artificial Intelligence]: Miscellaneous; H.2.8 [Database Applications]: Data Mining

General Terms

Algorithms

Keywords

data mining, association rules, classification

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.

C³S²E-11 2011, May 16-18, Montreal [QC, CANADA]

Editors: Abran, Desai, Mudur

Copyright 2011 ACM 978-1-4503-0626-3/11/05 ...\$10.00.

1. INTRODUCTION

Association Rule Mining (ARM) is one of the most important canonical tasks in data mining and probably one of the most studied techniques for pattern discovery. Association rules express associations between observations in transactional databases and the process of extracting them consists of initially finding frequent itemsets. Consequently, rules are extracted from these discovered frequent itemsets. A plethora of frequent itemset mining algorithms have been devised, such as Apriori [1], FP-Growth [21], and Eclat [19], to mention a few. However, no one algorithm is a clear winner in terms of speed because the characteristics of the database being mined contribute significantly to the running times [17, 14]. Given a transactional database to mine for association rules, the best and most efficient association rule mining algorithm varies based on the target dataset. Arbitrary or trial-and-error selection of which algorithm best fits a given dataset is inefficient, especially when huge datasets are involved, such as those in supermarkets, or web logs to mention a few.

We claim that selecting the most appropriate algorithm for frequent itemsets is also relevant to parallel computing. Given the very large datasets to mine, many have supported high performance and parallel computing. Yet, all parallel algorithms that exist for mining frequent itemsets consist of the same algorithm running on different processors either sharing memory or not sharing memory, where the algorithm focuses on partitioning the dataset to split among the processors and managing and effective load balancing between these processors. However, since all frequent itemset mining algorithms theoretically should discover the same set of patterns from an identical dataset and their only difference is their efficiency, the idea that we advocate is to allow different algorithms running on different processors based on the partition of data that is provided to them. To do so, a processor needs to select the algorithm to execute based on the characteristics of the data it is given. This is a task of predicting the best algorithm (i.e. supervised classification). Research into the area of predicting the best ARM algorithm is new. In fact, we found very little previous work directly suggesting the approach we present here. However, there was relevant literature to guide our work.

Zaïane *et al.* [26] develop a framework for comparatively studying the performance of various data mining algorithms against real and synthetic datasets. The framework allows performance analysis of both new and established algorithms by running them for each dataset and determining frequent closed itemsets. Our study may seem similar in concept, but our focus is on using machine learning to predict the optimal performing algorithms for real and synthetic datasets. Our approach to predicting the fastest algorithm is not based on performance benchmarks or finding frequent closed itemsets within a given dataset, but on the characteristics of the dataset itself. In our case, the algorithm is predicted without actually running it against any datasets.

Hipp *et al.* [12] compared various ARM algorithms based on how the algorithms traverse the search space for the dataset to find itemsets and how support for the itemsets is determined. For traversal, either breath-first search or depth-first search is used, while for determining support, either counting or intersection via TIDs is used. These observations were very important for our work. Fundamentally, one association mining algorithm is better than another because of the search method it uses and how the dataset is organized in line with the search method. The counting method also plays an equally important role. The paper also presented experiments done to compare 4 algorithms with varying supports. The graphs presented were important for our research as they showed that changing support can significantly affect the algorithm performances [12].

Veloso *et al.* [17] have performed an analysis of how to determine which association rule mining algorithm is likely to make a good match with a given database by a systematic experimental evaluation. Characteristics between real and synthetic datasets are compared. This work was directly related to our research and gave us many insights into the database characteristics that could serve as classifier features [17].

In addition, research has been focused on using sampling to estimate frequent itemsets [27, 28]. While these methods can potentially reduce running time, there are 2 shortfalls: 1) estimations do not provide an exhaustive list of patterns, and 2) the accuracy of estimators might fluctuate with the nature of the dataset and the sampling strategies. Our proposed approach allows the efficient use of traditional data mining methods without having to rely on good sampling estimations of frequent itemsets.

Various algorithms exist for association rule mining. Some of these algorithms are more efficient than others in terms of running time. Given some dataset, one algorithm generally outperforms the others. In fact, the time efficiency of these algorithms is closely dependent on the nature of dataset supplied, assuming memory is sufficient. Through this study we have come up with a classifier which would predict the fastest association rule mining algorithm, with a high degree of accuracy and significantly low overhead.

2. APPROACH AND METHODOLOGY

We used the classification approach to predict which association rule mining algorithm is fastest. The particular algorithms which we used were Apriori, Eclat and FP-Growth. We used a variety of transactional datasets from different domains as training data.

The main aim in classification is to build a prediction mechanism which can map objects into pre-defined classes

depending upon the attributes of objects. In the classification approach, we have the following main steps:

- Identification of candidate features
- Feature extraction
- Model training
- Model testing

First, we looked for candidate features that are relevant to the ARM algorithms. This was followed by running the selected algorithms on our datasets. Our classes mapped to the selected algorithms based on the fastest running time for a given dataset, also allowing for ties. We explain the notion of a tie in Section 2.7. Next we used machine learning techniques to determine which features give better accuracy and which need to be pruned. This led to a model that would be able to predict the fastest algorithm for a given dataset. Ultimately, we appraise this model against a real dataset by comparing our predicted results with actual results. Figure 1 summarizes our approach. We present the artifacts of the classification approach in the following sections.

2.1 Datasets

We used two types of datasets: real and synthetic. This was because the nature of the artificially generated transaction databases is different from real-world datasets [17]. We also wanted to diversify our datasets to build a better classifier. We used 23 synthetic datasets that have been used in other related literature as well. These datasets have been artificially generated with the IBM Generator [22]. Also, we used a total of 13 real datasets, most of which were obtained from the Frequent Itemset Mining Dataset Repository [22] and other sources such as [23]. Table 1 shows a listing of these real-world datasets and their sizes.

Name	Size
chess	369KB
connect	9.70MB
gaz	1.63MB
mushroom	650KB
pumsb	16.5MB
pumsb_star	11.4MB
BMS-POS	13.7MB
BMS-WebView	3.87MB
kosarak	30.5MB
retail	3.97MB
webdocs	1.37GB
Proteins	29.7MB
worldcup98	359.9MB

Table 1: List of Real Datasets used

2.2 Candidate Features

In a usual classifier building study, the features of the dataset would be selected depending on the context of the domain. For our work, our candidate features are *meta features*, since they contain information about the dataset. We chose 10 candidates related to a dataset’s properties. For easier reference, we gave each feature a meaningful name. Table 2 shows the features and their descriptions. As mentioned above, there are already various characteristics of

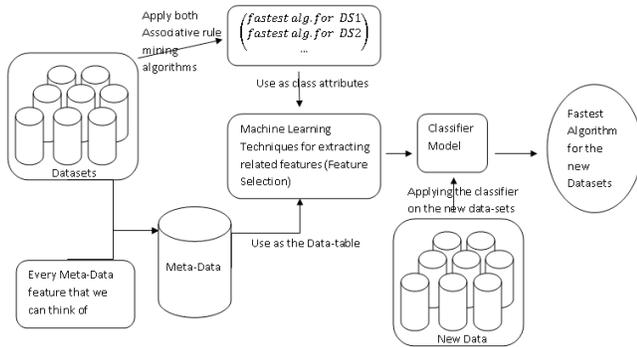


Figure 1: Overview of Classification Approach

datasets that are known to affect how ARM algorithms perform, and we leveraged this knowledge while selecting our candidate features. In addition, we determined that Apriori is sensitive to the *Bucket* feature.

Name	Description
Width	Average size of transactions
MaxWidth	Largest size of transaction
MinWidth	Smallest size of transaction
Height	Number of transactions
Span	Number of unique transaction items
MaxSpan	Largest size of transaction with unique items
MinSpan	Smallest size of transaction with unique items
Mass	Number of items
Support	Usual meaning of support
Bucket	Largest number of items per rule

Table 2: Candidate Features

2.3 Feature Selection

In this section, we will discuss the features that *probably* influence our classifier’s accuracy [9]. To find whether the features are correlated with the label used for predicting the fastest algorithm, correlation based feature selection was used. Correlation based feature selection defines the measure of goodness of a feature to its correlation to the label. The features are ranked based on this measure. We used two different correlation based methods: *Information Gain Feature Ranking* and *CFS*. Information Gain Feature Ranking is derived from Shannon entropy as a measure of correlation between the feature and the label to rank the features. Information Gain is a famous method and used in many papers [2]. The other method is CFS which uses the standardized conditional Information Gain to find the most correlated features to the label and eliminates those whose correlation is less than a user defined threshold [10]. Additionally, using a subset search, it eliminates those that are more correlated to the other features than the label (redundant features). Thus it does both dependency and redundancy filtering. Various tests showed that the algorithm is successful in reducing the feature space without decreasing the accuracy [10].

2.4 Classifiers

Our methodology suggests to use a classifier on a dataset to determine the fastest algorithm. The choice of the classi-

fier is very important, both from time and accuracy point of view. The accuracy should be obviously high. The time consumed by the classifier in the testing phase will be counted in the overhead of our system if it is to be used in a parallel setting for each processor to determine the algorithm to utilize. We used a broad set of different classifiers to choose between.

- Bayes Net: This algorithm uses various search algorithms and uses the Bayes Network data structure for learning. The algorithm is very flexible and intuitive and has been successfully used in many studies [6].
- C4.5: This algorithm which performs the learning by building decision trees, is commonly used for both discrete and continuous features [18]. It is one of the most influential algorithms selected by ICDM. It utilizes two heuristics (information gain and gain ratio) to build the decision tree. The training time of this algorithm is high, but it has a relatively low test time.
- Decision Table: Decision Table uses a very simple set of hypotheses to build a model. There is an underlying algorithm in the decision table schema which finds the best subset of features to put in the table. This algorithm can be rather simple, like a wrapper using best first search algorithm. Decision Tables are proven to be successful in datasets with continuous features [13]. The algorithm has a high training time, but small testing time.
- SVM: The functional classifier of SVM is one of the most influential classification methods [18]. Not only can it overrule other methods when data is linearly separable, but it also has the well known ability of being able to serve as a multivariate approximate of any function to any degree of accuracy. SVM uses a linear hyperplane to create a classification of maximal margin. The other important point about SVM is that it is basically known as a good comparison line for other classifiers [16]. SVM can be used as a kernelized version, but the test time will increase in that case. Thus, we will only use linear SVM.
- KNN: KNN or the K-nearest-neighbor algorithm votes between K geometrically close training points for every test point. This algorithm assumes that points in proximity to each other have the same class label. It is often successful when each class has many possible prototypes and the decision boundary is very irregular [16]. KNN belongs to the family of lazy classifiers, meaning that it does not build a model, so the training time is always 0. This feature is not desired, but as we use only small number of neighbors in range of 1,...,9 along with Manhattan Distance, the runtime will not be too high.
- NB: Naive Bayes classifier is a simple classifier which assumes that the features are completely independent and applies the Bayes rule based on this assumption. Despite its unrealistic independence assumption, the naive Bayes classifier is surprisingly effective in practice since its classification decision may often be correct even if its probability estimates are inaccurate [15]. This classifier is also very fast in both training and testing.

2.5 Feature Extraction

We use the term ‘feature extraction’ to describe getting the values for the candidate features for a given dataset. For instance, getting the value for the Span, Width, or Height for the BMS-POS dataset, and so on. This step of feature extraction is a key step because the data retrieved goes into our classifier, as depicted in Figure 1. The efficiency of our feature extractor is important because we want to keep overheads low.

2.6 Association Rule Mining Algorithms

Although there are various algorithms for ARM, we used 3 algorithms:

- Apriori: Generates candidate itemsets to be counted in a pass by using only the itemsets with frequency more than minimum support in the previous pass, because any subset of a large itemset must be large itself [1]
- FP-Growth: Uses a prefix tree representation of the given database (FP-tree). In a preprocessing step, all the single items that are not frequent are eliminated, then all the transactions that contain least frequent items are selected from among those which are frequent. The same steps are recursively applied, remembering that the itemsets found in recursion share the eliminated items [3]
- Eclat: Uses a bit matrix to represent the transactions, and uses a prefix tree to search in depth-first order. One main difference of Eclat from Apriori and FP-Growth is that it uses TID intersections in counting itemsets, which leads to very fast support counting [4, 19]

Our choice was guided by the popularity of these algorithms and their relative efficiencies found in literature. We used optimized C implementations of these algorithms provided by Borgelt [24]. These implementations provide running time information after execution that we used. We ignored the time taken for read-writes of the datasets, and used the time taken to generate frequent itemsets.

It should be noted that when execution is interrupted due to out-of-memory conditions, the algorithms return a value of -1 for the running time. For other unexpected situations, the algorithms return other negative values. For our experiments, when we encountered negative values other than -1, we ignored the results because they were meaningless. Another notable point is that the *Bucket* feature is a parameter in all the algorithms. It controls when the algorithms should stop generating itemsets, depending on the size of the itemsets. To get running times for the different datasets, these algorithms were run on a cluster with 8 GB of memory. We reserved 3 GB of memory and used 1 node to run all our scripts to call the algorithms on different datasets in parallel.

2.7 Training Set

The training set is created after feature extraction and running the algorithms on each dataset. For each dataset, the fastest algorithm determines the class. We allowed ties, so that for a given dataset, two or more algorithms could be relatively the same running time. However, not only running times that are exactly the same are considered equivalent.

We also incorporated the notion of *significance* between running times. If the difference between algorithms is greater than some *significance delta*, then we say that one is faster than the other, otherwise they are the same. This follows intuitively as well, since a difference of some milliseconds does not necessarily mean that one algorithm is significantly faster. More formally,

Let T_1 and T_2 be the running times for algorithms A_1 and A_2 respectively

Let ϕ be the setting for the significance delta

Let \supset define fastness of one algorithm over the other, for instance $A_1 \supset A_2$

If $|T_1 - T_2| > \phi$, then we say $A_1 \supset A_2$

Based on the notion of ties, we have 7 classes. Table 3 shows the class abbreviations and corresponding interpretations.

Class	Description
A	Apriori algorithm is fastest
F	FP-Growth algorithm is fastest
E	Eclat algorithm is fastest
AF	Apriori and FP-Growth tied and either is fastest
AE	Apriori and Eclat tied and either is fastest
FE	FP-Growth and Eclat tied and either is fastest
AFE	All algorithms tied

Table 3: Classes used in Training Set

We computed significance based on the range of times recorded after our experiments. This is because a uniform value of ϕ may not be suitable for all results. For instance, if the running times for all algorithms are very small, such that $T_1, T_2 < \phi$, then the choice of algorithm would become insignificant, which is not necessarily true. We use the following formula to compute ϕ , and introduce the notion of a *significance percentage*, $P : \phi = \text{MIN}(T_1, T_2) \times P$.

Given a dataset, we vary the significance percentage, P to get the optimal results for which algorithm was faster. Figure 2 shows the distribution of classes for different values of P .

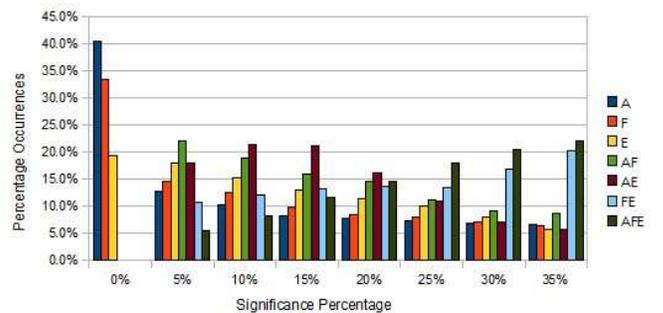


Figure 2: Distribution of Classes using different Significance Percentages

There were a total of 996 data points in our training set, generated from the datasets by varying support values to 1%, 5%, 10%, 30%, 50%, and 70%. In addition, the values for the *Bucket* candidate feature are parameters to the algorithms, and these were varied to 5, 10, 15, and default, which effectively means no limit.

2.8 Weka Machine Learning Tool

Weka is a machine learning tool implemented in Java which provides many classifiers and feature selection techniques [11]. In this study, we used Weka for both classification and feature selection tasks, as well as resampling. Even though there are other tools available for data mining and machine learning, our choice of Weka is based on its popularity within the research community.

In a dataset with discrete class labels, the majority class is the one that has the most number of members. The classifier which predicts the majority class for all of the test instances, is called Zero-R and its accuracy is called baseline [11]. Resampling produces a random subsample of a dataset using either sampling with replacement or without replacement. It is commonly used when a dataset is unbalanced, i.e. the majority class is a lot more populated than the other classes, thus the baseline is too high. In this case the prediction will be deviated to the majority class, causing the decrease in the accuracy of the test set. Resampling training data with a bias toward uniform distribution balances the dataset [11]. As described in Section 2.7 and in Figure 2 our dataset is unbalanced and needs resampling. Weka has all of the features we need, but Weka’s accuracy calculation has a major difference with the accuracy system we need to use in our study. As mentioned in Section 2.7, the labels of the dataset are one of the $\{A, E, F, AE, AF, EF, AFE\}$. The purpose of the study is to predict a fast algorithm. Hence if algorithms A_1 and A_2 are tied for running time, if the classifier predicts only one of the A_1 or A_2 our goal is met and we can declare it as an accurate prediction. Although we put the label as A_1A_2 in the training set to enhance the training task, in the testing part, a prediction of A_1 or A_2 should be counted as accurate. Table 4 shows these cases. To implement this we had to change some parts of the Weka source code [25].

Predicted Class	Actual Class
A	A, AF, AE, AFE
F	F, AF, FE, AFE
E	E, AE, FE, AFE
AF	AF, AFE
AE	AE, AFE
FE	FE, AFE
AFE	AFE

Table 4: Handling Ties in Weka

3. CLASSIFIER EVALUATION AND RESULTS

To evaluate the classifier, we measured the precision of the classifier model. In the classifier approach, there are two test phases: first for classification and feature extraction, and second for the new datasets. The first test phase guided us in selecting the better approach, but the true accuracy was found in the second phase. In addition, after prediction by our classifier, we actually ran the algorithms on the test datasets and compared the predictions with the actual empirical results. For evaluating our sampling predictions, we compared actual results with predicted results. In addition to evaluation based on accuracy, we also evaluated the overhead in using the classifier model or in sampling and frequent patterns mining. Given that prediction aims to increase the overall time taken to run an association rule mining algorithm, the overhead needed to be low enough.

3.1 Balancing

As mentioned in Section 2.8, using Weka, we applied a biased resampling filter to remove 50% of the training data and get to a lower baseline. To see the difference between the balanced and the unbalanced baselines, look at Figure 3. The numbers shown are the accuracies of the Zero-R classifier run under 10-fold cross validation for 10 times, when applied to the training set with different significance values.

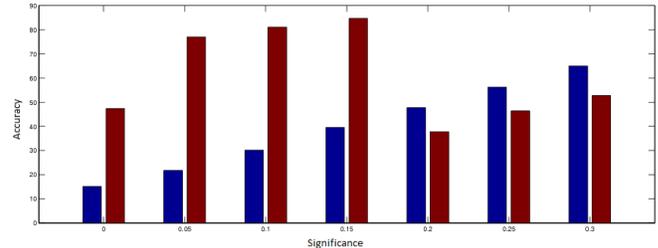


Figure 3: Baseline before and after balancing. The red bars show the baseline in unbalanced training sets, while the blue bars shows the baseline in balanced training sets

3.2 Training Results

Changing the value of significance in a 10 fold-CV scheme, we filtered the training set using Weka’s resampling, ran the feature selection algorithms using rank search, and applied the classifier. The feature selection algorithms are supervised, thus applied in-fold. For each significance value, the maximum accuracy, the classifier which achieved that accuracy, and the number of features are shown in Table 5. Figure 4 shows the highest accuracy for each significance value.

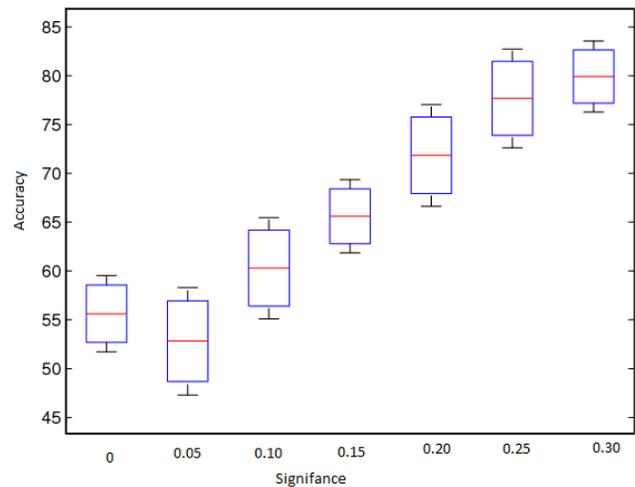


Figure 4: The best accuracy achieved for each significance value, maximum between all classifiers

3.3 Discussion

The results of the classifications were presented in the previous section. In choosing the classifier, two important

Significance Value	Accuracy	Classifier	Parameters	Number of Features
0.00	55.62(3.91)	C4.5	Confidence factor = 0.25 Minimum instance per leaf = 0.2	10
0.05	52.81(5.51)	C4.5	Confidence factor = 0.25 Minimum instance per leaf = 0.2	10
0.10	60.29(5.18)	C4.5	Confidence factor = 0.25 Minimum instance per leaf = 0.2	10
0.15	65.61(3.75)	C4.5	Confidence factor = 0.25 Minimum instance per leaf = 0.2	10
0.20	71.84(5.21)	Decision Table	Evaluation measure: Accuracy and RMSE Search: Best First	10
0.25	77.67(5.06)	Decision Table	Evaluation measure: Accuracy and RMSE Search: Best First	10
0.30	79.91(3.64)	Decision Table	Evaluation measure: Accuracy and RMSE Search: Best First	10

Table 5: The maximum accuracy reached with different significance values, the classifier reaching to the accuracy, its parameters, and the number of features

factors are effective, the accuracy and the runtime, assuming memory is sufficient. Fortunately, there was a clear winner as the best accuracies belong to Decision tree and C4.5 which also have a relatively small running time.

As shown in Figure 4 the significance measure is very important. As the significance grows higher, the accuracy of the prediction model grows. This can be because of two reasons: the dataset becomes more predictable because the labels get closer to the reality (reliability growth), or all of the class labels change into *AFE* (convergence).

Based on Figure 2, different classes still exist in the dataset in all of the significance values, so it is more likely that reliability growth causes the increase of the accuracy. Additionally as mentioned before the run time is not very different with C4.5. Based on these reasons, our model will use the significance percentage of 30%, Decision Table classifier, and 10 features.

3.4 Model Testing

In this section, we talk about how we tested our classifier against new datasets. This testing serves two purposes: testing the model in an actual real-world scenario, and also for evaluating overheads. Overall, tests performed on the model are meant to quantify its efficiency.

3.4.1 Simulation

The following scenario was used to test our model. Suppose we have a large dataset that needs to be mined. This dataset requires deployment over a cluster so that it can be processed efficiently in parallel. Normally the cluster would be configured to run the same algorithm that is arbitrarily selected, since there is no pre-knowledge of which algorithm suits the dataset. In addition, since parallelization is being used, the dataset will be broken down into smaller chunks. However, even though a guessed algorithm may suit the overall dataset, it may not be true for the sub-datasets on each node. Figure 5 shows the general setup of the scenario.

In our simulation, we first use a large dataset *accidents* using the typical setup of running the same algorithm for each node on a cluster. We then use our proposed solution of using a classifier to predict the best algorithm and use this classifier on each node. Finally, we make a comparative analysis of these 2 runs. The *accidents* dataset we used was for a region in Belgium and covers the period of 1991 to 2000.

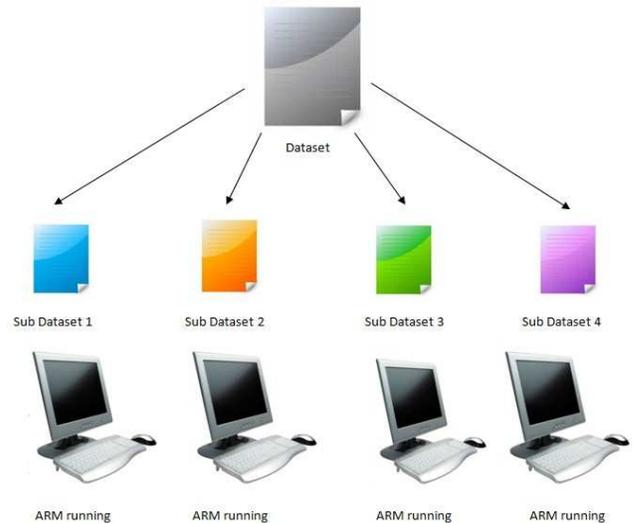


Figure 5: Setup of Simulation

Police officers fill in a form for each traffic accident they were covering. The dataset contains 340, 184 records of traffic accidents. Also, the dataset contains 572 different attribute values, but on average only 45 attributes have data [8].

3.4.2 Results

We split the *accidents* dataset and run an algorithm using different values of the parameters mentioned in Section 2.7. In one instance of our simulation, shown in Table 6, if an algorithm is guessed (for example Apriori) the time taken to complete the batch job on a cluster is 2966s. However, when FARM-AP is used, the total time to complete the job is 1.07s. This is because FARM-AP predicts that Eclat is better. In this instance, FARM-AP drastically speeds up the mining process. Also, in other instances of the simulation, with different values of the bucket and support, FARM-AP generally beats the guessing strategy when the best algorithm is not guessed.

3.5 Evaluation of FARM-AP

For evaluation of FARM-AP, we used 3 scenarios for best, average and worst cases. It should be noted that in all of

Node#	Apriori(sec)	FPGrowth(sec)	Eclat(sec)	Prediction	Feature Extraction Overhead(sec)	Classification Overhead(sec)	Total Overhead(sec)
1	1746	0.49	0.36	Eclat	0.607	0.01	0.617
2	1938	0.48	0.39	Eclat	0.668	0.01	0.678
3	2369	0.58	0.35	Eclat	0.587	0.01	0.597
4	2966	0.48	0.36	Eclat	0.670	0.01	0.680

Table 6: Results of the simulation for one case with support of 5 and no limit bucket. The model predicts the best algorithm which is Eclat

the instances of our simulation, FARM-AP predicted either FP-Growth or Eclat, which had running times much shorter than Apriori. This showed that FARM-AP was able to predict the better algorithm, but was limited by the overhead in feature extraction and classification. For classification overhead we recorded `Elapsed_time_testing` in Weka.

Best Case: For the best case in the simulation, i.e if the algorithm guessed is one with the longest running time (Apriori), FARM-AP guarantees that it will predict a better algorithm. The overhead is significantly low compared to the running time of the slower algorithm. The running time of the guessing scenario will be the same as the slowest algorithm.

Average Case: In the average case, the probability of guessing any algorithm is the same. The running time of the guessing scenario will be the mean of the algorithms. FARM-AP still outperforms guessing in this scenario.

Worst Case: Obviously in the worst case, if the fastest algorithm is guessed, then FARM-AP is beaten because of the overheads. This is because the running time of this guessing scenario will be the same as the fastest algorithm.

From the simulation, FARM-AP can predict the fastest algorithm with 83% accuracy. More generally, if the difference between the slowest and fastest algorithms is significantly high, and more than FARM-APs overhead (around 1s in the simulation), then FARM-AP beats the guessing strategy in the best and average scenarios. Even though Apriori was faster than FP-Growth and Eclat for some datasets during training, in our model testing simulation, Apriori was the slowest on all nodes. However, there may be special cases where an algorithm is fast on some nodes, and slow on other nodes due to the way the dataset is split, and the characteristics of the sub-datasets. In other words, different algorithms are optimal for different dataset partitions. Consequently, a different algorithm will be predicted for each sub-dataset, with a different overhead value for the sub-dataset. In this special case as well, if the difference between the running times of the fastest and slowest algorithm on each node is more than the largest overhead, FARM-AP will outperform arbitrary guessing, based on our high accuracy on predicting the best algorithm. This would be investigated in future research.

4. CONCLUSION AND FUTURE WORK

We have successfully presented a new technique using a classifier which can predict the fastest ARM algorithm with a high degree of accuracy of 80%, and very low overhead. Generally, FARM-AP outperforms arbitrary selection of an algorithm for a given dataset. We can make this claim based on the results of our simulation on the *accidents* dataset. For future work, the accuracy of FARM-AP can be increased, and the overhead can be decreased. Also, we suggest training FARM-AP with more features such as average maximal pattern length of the datasets.

5. REFERENCES

- [1] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *International Conference on Very Large Data Bases*, pages 487–499, 1994.
- [2] Jacek Biesiada, Wlodzislaw Duch, and Google Duch. Feature Selection for High-Dimensional Data: A Kolmogorov-Smirnov Correlation-Based Filter. In *Proceedings of the International Conference on Computer Recognition Systems*, 2005.
- [3] Christian Borgelt. An Implementation of the FP-growth Algorithm. In *International Workshop on Open Source Data Mining*, pages 1–5, 2005.
- [4] Christian Borgelt. Efficient Implementations of Apriori and Eclat. In *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.
- [5] Doug Burdick, Manuel Calimlim, and Johannes Gehrke. MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases. In *International Conference on Data Engineering*, pages 443–452, 2001.
- [6] Luis M. de Campos, Juan M. Fernandez-Luna, and Juan F. Huete. Bayesian Networks and Information Retrieval: An Introduction to the Special Issue. *Information Processing and Management*, 40(5):727–733, 2004.
- [7] Manoranjan Dash and Huan Liu. Feature Selection for Classification. *Intelligent Data Analysis*, pages 131–156, 1997.
- [8] Karolien Geurts, Geert Wets, Tom Brijs, and Koen Vanhoof. Profiling High Frequency Accident Locations Using Association Rules. In *Proceedings of the 82nd Annual Transportation Research Board*, page 18, 2003.
- [9] Isabelle Guyon and Andre Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research*, pages 1157–1182, 2003.
- [10] Mark A. Hall. Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 359–366, 2000.
- [11] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The Weka Data Mining Software: An Update. *SIGKDD Explorations*, 11(1).
- [12] Jochen Hipp, Ulrich Guntzer, and Gholamreza Nakhaeizadeh. Algorithms for Association Rule Mining - A General Survey and Comparison. *SIGKDD Explorations*, pages 58–64, 2000.
- [13] Ron Kohavi. The Power of Decision Tables. In *Proceedings of the European Conference on Machine Learning*, pages 174–189, 1995.

- [14] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham. A Portfolio Approach to Algorithm Selection. In *International Joint Conferences on Artificial Intelligence*, pages 1542–1542, 2003.
- [15] Irina Rish. An Empirical Study of the Naive Bayes Classifier. In *Workshop on Empirical Methods in Artificial Intelligence*, 2001.
- [16] Robert Tibshirani Trevor Hastie and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, corrected edition, August 2003.
- [17] Adriano Veloso, Bruno Gusmao Rocha, Marcio de Carvalho, and Wagner Meira Jr. Real World Association Rule Mining. In *Proceedings of the British National Conference on Databases*, pages 77–89, 2002.
- [18] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 Algorithms in Data Mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.
- [19] Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New Algorithms for Fast Discovery of Association Rules. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, 1997.
- [20] Zijian Zheng, Ron Kohavi, and Llew Mason. Real World Performance of Association Rule Algorithms. In *SIGKDD*, pages 401–406, 2001.
- [21] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining Frequent Patterns without Candidate Generation. In *Data Mining and Knowledge Discovery*, 8:53–87, 2004.
- [22] Frequent Itemset Mining Dataset Repository. Available at <http://fimi.cs.helsinki.fi/data/>, 2010.
- [23] Mohammed J. Zaki. Real and Synthetic Datasets. Available at <http://www.cs.rpi.edu/~zaki/www-new/pmwiki.php/Software/Software#toc32>, 2010.
- [24] Christian Borgelt. Software for Frequent Pattern Mining. Available at <http://www.borgelt.net/fpm.html>, 2010.
- [25] The Weka Source Code. Available at http://www.cs.waikato.ac.nz/~ml/weka/index_downloading.html, 2010.
- [26] Osmar R. Zaiane, Mohammad El-Hajj, Yi Li, Stella Luk. Scrutinizing Frequent Pattern Discovery Performance. In *IEEE ICDE*, pages 1109–1110, 2005.
- [27] Ruoming Jin, Scott McCallen, Yuri Breitbart, Dave Fuhry, and Dong Wang. Estimating the Number of Frequent Itemsets in a Large Database. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, 2009.
- [28] Congnan Luo and Soon M. Chung. A Scalable Algorithm for Mining Maximal Frequent Sequences using a Sample. In *Knowledge Information Systems*, 15(2):149–179, 2008.