Mining Changes for Real-Life Applications

Bing Liu, Wynne Hsu, Heng-Siew Han and Yiyuan Xia

School of Computing
National University of Singapore
3 Science Drive 2
Singapore 117543
{liub, whsu, xiayy}@comp.nus.edu.sg

Abstract. Much of the data mining research has been focused on devising techniques to build accurate models and to discover rules from databases. Relatively little attention has been paid to mining changes in databases collected over time. For businesses, knowing what is changing and how it has changed is of crucial importance because it allows businesses to provide the right products and services to suit the changing market needs. If undesirable changes are detected, remedial measures need to be implemented to stop or to delay such changes. In many applications, mining for changes can be more important than producing accurate models for prediction. A model, no matter how accurate, can only predict based on patterns mined in the old data. That is, a model requires a stable environment, otherwise it will cease to be accurate. However, in many business situations, constant human intervention (i.e., actions) to the environment is a fact of life. In such an environment, building a predictive model is of limited use. Change mining becomes important for understanding the behaviors of customers. In this paper, we study change mining in the contexts of decision tree classification for real-life applications.

1. Introduction

The world around us changes constantly. Knowing and adapting to changes is an important aspect of our lives. For businesses, knowing what is changing and how it has changed is also crucial. There are two main objectives for mining changes in a business environment:

1. To follow the trends: The key characteristic of this type of applications is the word "follow". Companies want to know where the trend is going and do not want to be left behind. They need to analyze customers' changing behaviors in order to provide products and services that suit the changing needs of the customers.
2. To stop or to delay undesirable changes: In this type of applications, the keyword is "stop". Companies want to know undesirable changes as early as possible and to design remedial measures to stop or to delay the pace of such changes. For example, in a shop, people used to buy tea and creamer together. Now they still buy tea, but seldom buy creamer. The shopkeeper needs to know this information so that he/she can find out the reason and design some measures to attract customers to buy creamer again.

In many applications, mining for changes can be more important than producing accurate models for prediction, which has been the focus of existing data mining research. A model, no matter how accurate, in itself is passive because it can only predict based on patterns mined in the old data. It should not lead to actions that may change the environment because otherwise the model will cease to be accurate.

Building models for prediction is more suitable in domains where the environment is relatively stable and there is little human intervention (i.e., nature is allowed to take its course). However, in many business situations, constant human intervention to the environment is a fact of life. Companies simply cannot allow nature to take its course. They constantly need to perform actions in order to provide better services and products. For example, in a supermarket, there are always discounts and promotions to raise sale volume, to clear old stocks and to generate more sales traffic. Change mining is important in such situations because it allows the supermarket to compare results before and after promotions to see whether the promotions are effective, and to find interesting changes and stable patterns in customer behaviors.

Even in a relatively stable environment, changes (although in a slower pace) are also inevitable due to internal and external factors. Significant changes often require immediate attention and actions to modify the existing practices and/or to alter the domain environment. Let us see a real-life example.

A company hired a data mining consultant firm to build a classification model from their data. The model was built using the decision tree engine in a commercial data mining system. The accuracy was 81% at the time when the model was built. However, after the Asia financial crisis, the model only worked 60% of the time. The company asked the consultant why the classification model did not work any more. The reason, of course, is simple, i.e., the training data used to build the model (or classifier) was collected before the financial crisis. The consultant firm then built a new model (a classifier) for the company using the data collected after the financial crisis. The model was again accurate. However, after a while the company realized that the new model did not help. The reason is also simple. The company's profit is dropping and an accurate model could not stop this decline. What they really need is to know what has changed in the customer behaviors after the financial crisis so that they can perform some actions to reverse the situation. This requires change mining to compare the data collected from different periods of time.

In this paper, we study change mining in the contexts of decision tree classification. The study is motivated by two real-life data mining applications (see Section 3.2). In these applications, the users want to find changes in their databases collected over a number of years. In our literature search, we could not find suitable techniques to solve the problems. We thus designed a method for change mining in decision tree classification. This method has been incorporated into a decision tree algorithm to make it also suitable for mining changes.

There are existing works that have been done on learning and mining in a changing environment. Existing research in machine learning and computational learning theory has been focused on generating accurate predictors in a drifting environment [e.g., 14, 5, 7, 17]. It does not produce the explicit changes that have occurred. In data mining, [1, 4] addressed the problem of monitoring the support and confidence changes of association rules. [6] gave a theoretical framework for measuring changes. We will discuss these and the other related works in Section 4.

2. Mining Changes in the Decision Tree Model

Decision tree construction is one of the important model building techniques. Given a data set with a fixed discrete class attribute, the algorithm constructs a classifier of the domain that can be used to predict the classes of new (or unseen) data.

Traditionally, misclassification error rate is used as the indicator to show that the new data no longer conforms to the old model. However, the error rate difference does not give the characteristic descriptions of changes, as we will see below. Additional techniques are needed.

In a decision tree, each path from the root node to a leaf node represents a hyper-rectangle region. A decision tree essentially partitions the data space into different class regions. Changes in the decision tree model thus mean changes in the partition and changes in the error rate (see Section 2.3). Our objective in change mining is:

- to discover the changes in the new data with respect to the old data and the old decision tree, and present the user with the exact changes that have occurred.

The discovered changes should also be easily understood by the user. Our application experiences show that changes are easily understood if they are closely related to the old decision tree or the old partition.

2.1 Approaches to change mining

Below, we present three basic approaches to change mining in the decision tree model: *new decision tree*, *same attribute and best cut*, and *same attribute and same cut*. The first two approaches modify the original tree structure, which makes the comparison with the original structure difficult. The third approach is more appropriate and it is the method that we use. Here, we discuss these three approaches. The detailed algorithm for the third approach is presented in Section 2.2. Note that the basic decision tree engine we use in our study is based on that in C4.5 [15]. We have modified it in various places for change mining purposes.

1. **New decision tree**: In this method, we generate a new decision tree using the new data, and then overlay the new decision tree on the old decision tree and compare the intersections of regions. The intersection regions that have conflicting class labels are the changes. This idea was suggested in [6].
2. **Same attribute and best cut**: This method modifies the decision tree algorithm so that in generating the new tree with the new data, it uses the same attribute as in the old tree at each step of partitioning, but it does not have to choose the same cut point for the attribute as in the old tree. If the algorithm has not reached the leaf node of a particular branch in the old tree and the data cases arrive here are already pure (with only one class), it stops, i.e., no further cut is needed. If any branch of the new tree needs to go beyond the depth of the corresponding branch in the old tree, the normal decision tree building process is performed after the depth.
3. **Same attribute and same cut:** In this method, we modify the decision tree engine so that in building the new tree, it not only uses the same attribute but also the same cut point in the old tree. If the algorithm has not reached the leaf node of a particular branch in the old tree and the data cases arrive here are already pure, it stops. If any branch of the new tree needs to go beyond the depth of the corresponding branch in the old tree, the normal process is performed.

In all three approaches, decision tree pruning is also performed. Let us use an example to show the differences among the three approaches. We use the iris data set from UC Irvine machine learning repository [12] in the example. The iris data has 4 attributes and 3 classes. We only use two attributes (and all 3 classes) here for illustration. The data points in the original data set are drawn in Figure 1 together with the partition produced by the decision tree engine. Next, we introduce changes in the data by shifting some data points of setosa class in region 1 (Figure 1) toward left and put some versicolor class points in the space vacated, see the shaded area in Figure 2.

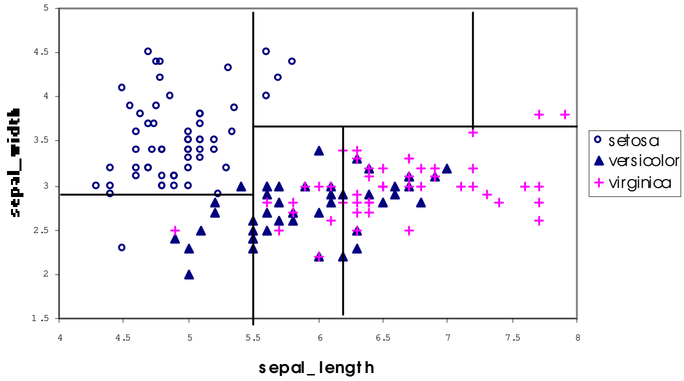


Fig. 1. Partition produced by decision tree on the original data

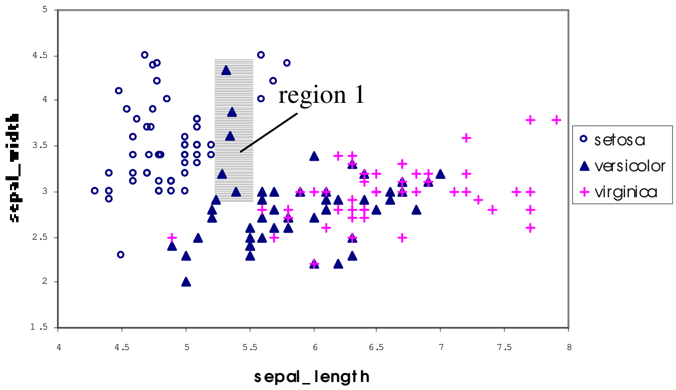


Fig. 2. The introduced change

We look at approach 1 first. Figure 3 shows the partition produced by approach 1 on the new data after the changes have been introduced. From Figure 3 alone, it is not clear what the changes are.

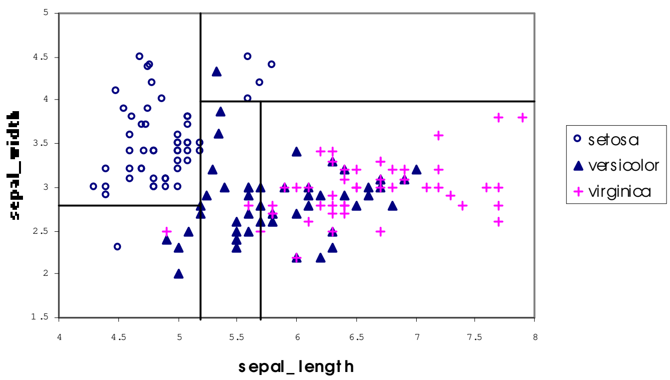


Fig. 3. Partition produced by decision tree on the new data

Figure 4 shows the overlay of this partition (Figure 3) on the old partition (Figure 1). Dashed lines represent the old partition (produced from the old data). The shaded areas are the conflicting regions, which represent changes.

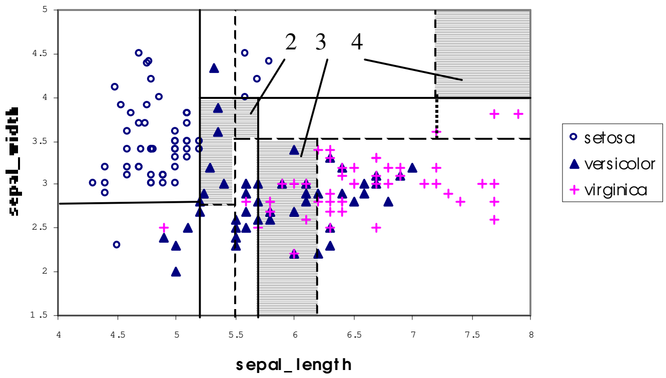


Fig. 4. Overlay of the new partition on the old partition

Clearly, the result is not satisfactory. It produces changes that do not actually exist, i.e., the intersection areas marked 2, 3, and 4. These problems become more acute when the number of attributes involved is large. The reason is that the changes in data points caused the decision tree engine to produce a completely new partition that has nothing to do with the old partition (it is well known that the decision tree algorithm can result in a very different tree even if only a few data points are moved slightly).

The same problem exists for the second approach, same attribute and best cut. Figure 5 shows the new partition. Although the algorithm tries to follow the attribute sequence in the old tree, but the cut points can change drastically, which make it hard to compare with the old tree. Again we need to use overlay to find changes. Hence, it has the same problem as the first approach.

The third approach (same attribute and same cut), on the other hand, does not have the problems. Figure 6 shows the partition obtained. The shaded area represents the change, which is precisely what has been introduced (see Figure 2). The change is also closely related to the old tree, and thus easily understood. We present the details of the third approach below, which is what we use in our system.

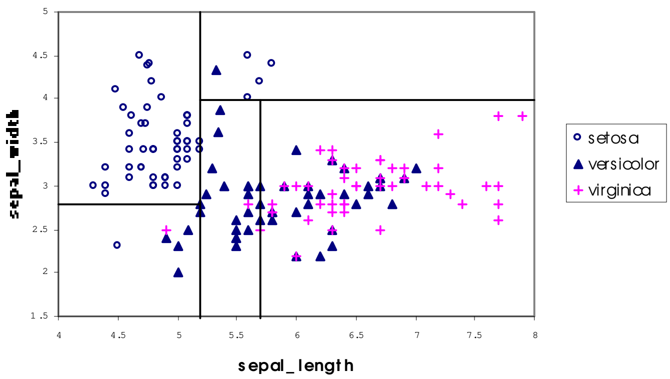


Fig. 5. Partition produced by the second approach

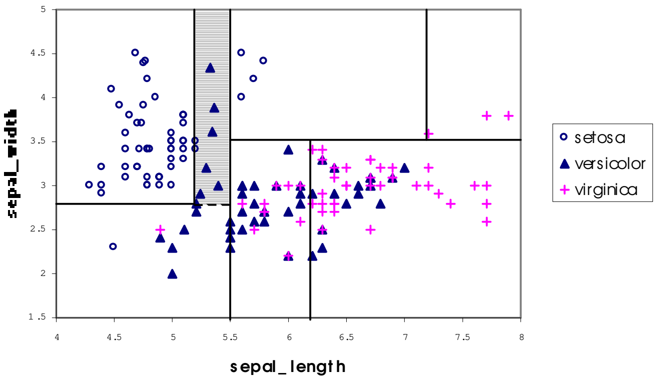


Fig. 6. Partition produced by the third approach

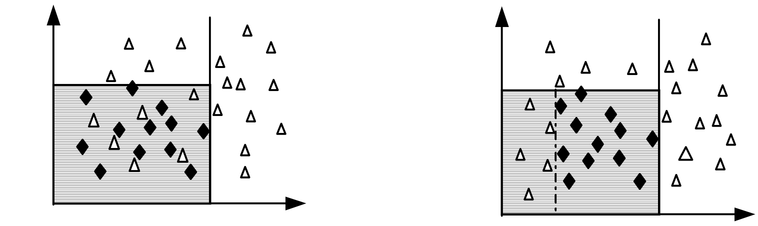
2.2. Change mining for the third approach: same attribute and same cut

The conceptual algorithm for the third approach is given in Figure 7. Let O_T be the old tree, and N_D be the new data. Let the decision tree building algorithm be buildTree().

Algorithm mineChange(O_T, N_D)
1 Force buildTree(N_D) to follow the old tree O_T (both the attributes and cut points) and stop earlier if possible;
2 Test for significance of the error rate change at each old leaf node and show the user those leaves that have significant error rate changes;
3 Grow the new tree further (not only those branches that have changed significantly);
4 Prune the tree to remove those non-predictive branches;
5 Traverse the new tree and compare it with O_T to identify changes;

Fig. 7. The proposed algorithm

Five points to note about the algorithm:
a. In line 1, the algorithm basically tries to follow the old tree.
b. Line 2 uses chi-square test [13] to test the significance of the changes in error rates.
c. In line 3, we allow the decision tree algorithm to grow the new tree further, not only those branches that have significant changes. The reason is that those branches that do not result in significant changes may be further partitioned to result in more homogenous regions, which is not possible in the old data.



(a) Partition from the old data (b) Partition from the new data

Fig. 8. Partitioning the region further

For example, in the old data in Figure 8(a), the shaded region cannot be partitioned further because the error points (Δ) are randomly positioned. However, in the new data, the error rate for the shaded region remains the same as in the old partition, but it can now be further refined into two pure regions (Figure 8(b)).

d. In line 4, the new tree is subjected to pruning. Pruning is necessary because the new tree can grow much further and result in overfiting the new data. As in the normal decision tree building, we need to perform pruning to remove those non-predictive branches and sub-trees. There are two ways to perform pruning in normal decision tree building [15]. They are:
   1) discarding one or more sub-trees and replacing them with leaves;
   2) replacing a sub-tree by one of its branches.

   For change mining, the pruning process needs to be restricted. We still use (1) as it is because it only results in join of certain regions, which is understandable. However, (2) needs to be modified. It should only prune till the leaf nodes of the old tree. Those nodes above will not be subjected to the second type of pruning because otherwise it can result in drastic modification in the structure of the old tree, thus making the changes hard to identify and to understand.

e. In line 5, the algorithm traverses the new tree and compares it with the corresponding nodes in the old tree to report the changes to the user. Below, we present the types of changes that we aim to find using the proposed approach.

2.3 Identify different types of changes

There are many kinds of changes that can occur in the new data with respect to the old data. Below, we identify three main categories of changes in the context of decision tree building: *partition change*, *error rate change*, and *coverage change*. The first four types of changes presented below are partition changes, type 5 is the error rate change and type 6 is the coverage change. Their meanings will be made clear later.

Type 1. **Join of regions**: This indicates that some cuts in the old tree (or partition) are no longer necessary because the data points in the new data set arrive in the split regions are now homogeneous (of the same class) and need no further partitioning.

Type 2. **Boundary shift**: This indicates that a cut in the old tree is shifted to a new position. It only applies to numeric attributes. Boundary shifts only happen at those nodes right above the leaf nodes of the old tree. It is not desirable to allow boundary shifts in the earlier nodes because otherwise, the whole tree can be drastically changed resulting in the problem discussed in Section 2.1.

Type 3. **Further refinement**: This indicates that a leaf node in the old tree can no longer describe the new data cases arrive at the node (or the region represented by the node). Further cuts are needed to refine the node.

Type 4. **Change of class label**: This indicates that the original class of a leaf node in the old tree has changed to a new class in the new tree. For example, a group of people who used to buy product-1 now tend to buy product-2.

Partition changes basically isolate the changes to regions (which can be expressed as rules). They provide the detailed characteristic descriptions of the changes. They are very useful for targeted actions. Though useful, partition changes may not be sufficient. Error rate change and coverage change are also necessary. The reasons are two-fold. First, sometime we cannot produce a partition change because the changes of the data points are quite random, i.e., we cannot isolate the changes to some regions. Nevertheless, there is change, e.g., the error rate has increased or decreased, or the proportion of data points arrives at a node has increased or decreased. Second, even if we can characterize the changes with regions, the data points in the regions may not be pure (i.e., with different classes of points). In such cases, we need error rate change to provide further details to gauge the degree of change.

Type 5. Error rate change (and/or class distribution change): This indicates that the error rate of (and/or the class distribution of the data points arrive at) a node in the new tree is significantly different from the error rate of the same node in the old tree. For example, the error rate of a node in the old tree is 10%, but in the corresponding node of the new tree it is 40%.

Type 6. **Coverage change**: This indicates that the proportion of data points arrives at a node has increased or decreased significantly. For example, in the old tree, a node covers 10% of the old data points, but now it only covers 2% of the new data points.

The partition changes (the first four types) can be easily found by traversing the new tree and comparing it with the old tree. The information on the last two types of changes is also easily obtainable from the final new tree and the old tree.

3. Experiments and Applications

We evaluate the proposed techniques using synthetic data sets and real-life data. The goal is to assess the effectiveness of the proposed technique (the third approach in Section 2.1). Efficiency is not an issue here because it uses an existing technique for decision tree building [15], which has been proven very efficient.

3.1 Synthetic data test

We implemented a synthetic data generator to produce data sets of 2 classes. It takes as input, the number of attributes (all attributes are numeric attributes), range of values for each attribute, the number of data regions with attached classes, the locations of the data regions to be generated, and the number of data points in each region. Basically, the data generator generates a number of regions with data points in them, and each data point is also labeled with a class. Each region is in the shape of a hyper-rectangle (each surface of the hyper-rectangle is parallel to one axis and orthogonal to all the others). The data points in each region are randomly generated using a uniform distribution. In the new data generation, we introduce changes by modifying the input parameters used for the old data set generation.

In our experiments (the results are summarized in Table 1), we used data sets with 3, 5, 8 and 10 dimensions. We first generate 3 old data sets (i.e., No. of Expts in Table 1) with 6, 8 and 10 data regions for each of the 3, 5, 8 and 10 dimensional spaces. For each old data set, we then generate one new data set and introduce all 6 types of changes at different locations. We then run the system to see if the planted changes can be identified. Experiment results show that all the changes embedded are found.

Table 1. Synthetic data test

No. of Expts	No. of dimensions	Types of changes introduced	Changes found
3	3	6	✔
3	5	6	✔
3	8	6	✔
3	10	6	✔

3.2 Real-life Data Tests

We now describe our two real-life data tests. Due to confidentiality agreement, we could not disclose the detailed findings. The first application is for an educational institution that wishes to find out more about its students. We were given the data from the past few years. The data includes the student's examination results, family background, personal particulars etc. Our user was interested in knowing how the performances of different groups of students had changed over the years. This requires change mining. A base year was chosen to build the old tree and the program was run by comparing the subsequent years with the base year decision tree. Some interesting trends immediately became apparent using our technique. For instance, we were able to tell our user that the performances of certain group (with some characteristics) of students in a particular subject had steadily deteriorated over the years. In addition, we also discovered that in a particular year, a group of students suddenly outperformed another group that had consistently been the better students.

The second application involves the data from an insurance company. The user knew that the number of claims and the amounts per claim had risen significantly over the years. Yet, it was not clear whether there were some specific groups of people who were responsible for the higher number of claims or that the claims were just random. In order to decide suitable actions to be taken, our user wanted to know what are the claiming patterns of the insurers over the years. Using data from the past five years, our system again discovered some interesting changes. We found that certain groups of insurers had gradually emerged to be the major claimers over the years. On the other hand, there exists another group of insurers that no longer had any claims even though in the beginning, they did put in claims.

4. Related Work

Mining and learning in a changing environment has been studied in machine learning [e.g., 14, 17], data mining [1, 4] as well as in computational learning theory [5, 7]. In machine learning, the focus is on how to produce good classifiers in on-line learning of a drifting environment [14, 17]. The basic framework is as follows: The learner only trusts the latest data cases. This set of data cases is referred to as the *window*. New data cases are added to the window as they arrive, and the old data cases are deleted from it. Both the addition and deletion of data cases trigger modification to the current concepts or model to keep it consistent with the examples in the window. Clearly, this framework is different from our work as it does not mine changes.

In computational learning theory, there are also a number of works [5, 7] on learning from a changing environment. The focus is on theoretical study of learning a function in a gradually changing domain. They are similar in nature to those works in machine learning, and they do not mine changes as the proposed method does.

[6] presented a general framework for measuring changes in two models. Essentially, the difference between two models (e.g., two decision trees, one generated from data set D_1 and one generated from data set D_2) is quantified as the amount of work required to transform one model into the other. For decision trees, it computes the deviation by overlaying of two trees generated from two data sets respectively. We have shown that overlaying of one tree on another is not satisfactory for change mining. The proposed method is more suitable.

Another related research is subjective interestingness in data mining. [10, 11, 16] gives a number of techniques for finding unexpected rules with respect to the user's existing knowledge. Although the old model in our change mining can be seen as "the user's existing knowledge" (it is not from the user), interestingness evaluation techniques cannot be used for mining changes as its analysis only compares each newly generated rule with each existing rule to find the degree of difference. It does not find which aspects have changed and what kinds of changes have occurred.

5. Conclusion

In this paper, we study the problem of change mining in the contexts of decision tree classification. This is motivated by two real-life applications. We could not find an existing technique to solve the problems. This paper proposed a technique for the purpose. Empirical evaluation shows that the methods are very effective.

We believe change mining will become more and more important as more and more data mining applications are implemented in production mode. In our future work, we plan to address the problem of evaluating the quality of the changes and to detect reliable changes as early as possible by on-line monitoring.

References

[1] Agrawal, R. and Psaila, G. "Active data mining." *KDD-95*, 1995.

[2] Agrawal, R., Imielinski, T., Swami, A. “Mining association rules between sets of items in large databases.” *SIGMOD-1993*, 1993, pp. 207-216.

[3] Cheung, D. W., Han, J, V. Ng, and Wong, C.Y. “Maintenance of discovered association rules in large databases: an incremental updating technique.” *ICDE-96*, 1996.

[4] Dong, G. and Li, J. “Efficient mining of emerging patterns: discovering trends and differences.” KDD-99, 1999.

[5] Freund, Y and Mansour, Y. “Learning under persistent drift” Computational learning theory: Third European conference, 1997.

[6] Ganti, V., Gehrke, J., and Ramakrishnan, R. "A framework for measuring changes in data characteristics" *POPS-99*.

[7] Helmbold, D. P. and Long, P. M. “Tracking drifting concepts by minimizing disagreements.” *Machine Learning*, 14:27, 1994.

[8] Johnson T. and Dasu, T. "Comparing massive high-dimensional data sets," *KDD-98*.

[9] Lane, T. and Brodley, C. "Approaches to online learning and concept drift for user identification in computer security." *KDD-98*, 1998.

[10] Liu, B., Hsu, W., “Post analysis of learnt rules." *AAAI-96*.

[11] Liu, B., Hsu, W., and Chen, S. “Using general impressions to analyze discovered classification rules.” *KDD-97*, 1997, pp. 31-36.

[12] Merz, C. J, and Murphy, P. UCI repository of machine learning databases [http://www.cs.uci.edu/~mlearn/MLRepository.html], 1996.

[13] Moore, D.S. “Tests for chi-squared type.” In: R. B. D’Agostino and M. A. Stephens (eds), *Googness-of-Fit Techniques*, Marcel Dekker, New York, 1996, pp. 63-95.

[14] Nakhaeizadeh, G., Taylor, C. and Lanquillon, C. “Evaluating usefulness of dynamic classification”, *KDD-98*, 1998.

[15] Quinlan, R. *C4.5*: *program for machine learning.* Morgan Kaufmann, 1992.

[16] Silberschatz, A., and Tuzhilin, A. “What makes patterns interesting in knowledge discovery systems.” *IEEE Trans. on Know. and Data Eng.* 8(6), 1996, pp. 970-974.

[17] Widmer, G. "Learning in the presence of concept drift and hidden contexts." *Machine learning*, 23 69-101, 1996.