A Comparison on How Statistical Tests Deal with Concept Drifts

Paulo M. Gonçalves Jr.¹,², Roberto S. M. Barros¹
¹Centro de Informática, Universidade Federal de Pernambuco, Cidade Universitária, 50.740-560, Recife, Brasil
²Instituto Federal de Pernambuco, Cidade Universitária, 50.740-540, Recife, Brasil

Abstract—RCD is a framework proposed to deal with recurring concept drifts. It stores classifiers together with a sample of data used to train them. If a concept drift occurs, RCD tests all the stored samples with a sample of actual data, trying to verify if this is a new context or an old one that is recurring. This is performed by a non-parametric multivariate statistical test to make the verification. This paper describes how two statistical tests (KNN and Cramer) can distinguish between new and old contexts. RCD is tested with several base classifiers, in environments with different rates-of-change values, with gradual and abrupt concept drifts. Results show that RCD improves single classifiers accuracy independently of the statistical test used.

Keywords: Multivariate non-parametric statistical tests, concept drifts, data streams.

1. Introduction

Concept drift is a situation that frequently occurs in data streams environments [1]. One informal definition, as stated by Kolter and Maloof [2], informs that “concept drift occurs when a set of examples has legitimate class labels at one time and has different legitimate labels at another time.” Another definition of concept drift describes that, in machine learning, “the term concept refers to the quantity that a learning model is trying to predict, i.e., the variable. Concept drift is the situation in which the statistical properties of the target concept change over time.” [3]

Therefore, dealing with concept drift is a task of increasing importance to the areas of data mining and machine learning, as increasing amounts of data are being stored in the form of data streams instead of static databases. Also, it is not very common for data distributions and concepts to keep stable over a long period of time [4], [5].

This paper extends Recurring Concept Drifts (RCD) framework [6] with the implementation of the Cramer multivariate statistical test [7], and shows how it compares to tests using KNN [8]. The performance of the implemented tests were analyzed in terms of accuracy and evaluation time. RCD was tested with several base classifiers in commonly used data sets in the concept drift research area, including abrupt and gradual concept drifts.

The rest of this paper is organized as follows: Section 2 presents approaches to deal with concept drifts, algorithms, and how they work. Section 3 briefly describes how RCD works. Section 4 presents the algorithms used in the comparison, the parameters set, and the data sets used in the tests. Section 5 presents the accuracy, number of detected concept drifts, and evaluation time between RCD and the other algorithms in the selected data sets. Finally, Section 6 presents our conclusions and proposes future work.

2. Background

There are several approaches to handle concept drifts. One is to create a model that adapts its internal structure as instances arrive. CVFDT (Concept-adapting Very Fast Decision Tree) [9] is an example. It extends VFDT [10] to handle concept drifts. VFDT is a decision tree built to deal with data stream and uses a Hoeffding bound [11], [12], to decide exactly how many examples are necessary at each node to find the best attribute to test. VFDTc [13] is another proposal, which also extends VFDT with the ability to deal with continuous attributes and the use of naive Bayes classifiers at tree leaves. OCVFDT (One-class Very Fast Decision Tree) [14] is an algorithm to extend VFDT to deal with one-class problems, based on the fact that fully labeled data streams are expensive to obtain. It only deals with discrete attribute values.

Another common approach to deal with concept drifts is to identify when it occurs and create a new classifier. Thus, only classifiers trained on a current concept are maintained. Algorithms that follow this approach work in the following way: each arriving training instance is first evaluated by the base classifier. Internal statistics are updated with the results and two thresholds are computed: a warning level and an error level. As the base classifier makes mistakes, the warning level is reached and instances are stored. If the behavior continues, it will reach the error level, indicating that a concept drift has occurred. At this moment, the base classifier is destroyed and a new base classifier is created and initially trained on the stored instances. On the other hand, if the classifier starts to correctly evaluate instances, this situation is considered a false alarm and stored instances are flushed. Algorithms that follow this approach can work with any type of classifier as they only analyze how the classifier evaluates instances. Proposals that use this approach are DDM (Drift Detection Method) [15], EDDM (Early Drift Detection Method) [16], and ECDD (Exponentially Weighted Moving Average for Concept Drift Detection) [17].

Several proposals try to deal with concept drifts by the use of ensemble classifiers. This approach maintains a collection
of learners and combine their decisions to make an overall decision. To deal with concept drifts, ensemble classifiers must take into account the temporal nature of the data stream.

One of these proposals was the Streaming Ensemble Algorithm (SEA) [18]. It builds separate classifiers, each one trained on a different sequential chunk of data. These classifiers are then combined into a fixed-size ensemble using a heuristic replacement strategy. If there is free space in the ensemble, SEA adds the newly created classifier to the ensemble. If the ensemble is full, the new classifier is only added if it outperforms a stored classifier, substituting it. The performance is measured on the current batch of examples.

The Accuracy Weighted Ensemble (AWE) classifier [19] is another proposal of ensemble classifier. It uses batch classifiers and each one is built in different chunks of data. Classifiers weights are computed based on their expected accuracy in test data. Thus, they proposed to infer the weights by estimating the error rate in the most recent data blocks.

Accuracy Updated Ensemble (AUE) [20] is an enhancement of the AWE classifier. Both use classifier ensembles and associate to them weights that are updated as data arrive. AUE improves AWE in three directions: (a) Instead of batch classifiers, AUE uses on-line classifiers, which adapt to data while they arrive; (b) AUE updates both classifiers and their weights, while AWE only updates the weights; (c) AUE changed the weight computation to avoid situations where no classifier should have precision higher than the stipulated threshold, as in abrupt concept drifting environments, zeroing the weights of all classifiers and no class being predicted.

The Weighted Majority Algorithm (WMA) [21] implements a weighted ensemble classifier, which extends previous work [22] to specifically handle concept drifts. In WMA all the arriving instances are passed to all the classifiers in the ensemble. The initial weight of the classifiers is 1, and, if a classifier makes an error, its weight is reduced by a factor of \( \beta \) if higher than a specified minimum threshold. Then, the classifier is trained. After all the ensemble classifiers have been trained, their weights are normalized.

Dynamic Weighted Majority (DWM) [2] is another example of ensemble classifier. It extends WMA to add and remove classifiers according to the algorithms global and local performance. If the ensemble commits an error, then a classifier is added. If one classifier commits an error, its weight is reduced. If after many examples a classifier continues with a low accuracy, indicated by a low weight, it is removed from the ensemble. This method is general and, in principle, can be used with any classifier.

A common situation regarding concept drifts is its recurrence. It happens when the actual context is similar to a previously seen context. One approach commonly used to treat recurring concept drifts is to store information about the contexts, and continuously identify if actual context is similar to old contexts. If it is so, obtain stored information and use it again, as it is expected that it also represents actual context. Algorithms that use this approach to deal with context recurrence are FLORA [4], Prediction Error Context Switching (PECS) [23], and SPLICE [24].

3. The RCD Framework

The RCD framework, proposed in [6], deals with concept drifts by storing classifiers together with data samples used to train the classifiers. It starts with one single classifier and uses a concept drift detector, like DDM or EDDM for example, to identify the occurrence of a concept drift. While no drift is detected, the arriving instances are used to train the actual classifier and are stored in a FIFO data buffer with fixed length. If the drift detector enters the warning level, RCD stops storing instances in the classifier’s buffer and stores them in a new buffer. If the error level is reached, indicating a concept drift, a multivariate non-parametric statistical test is performed to compare the instances obtained in the warning level with the instances associated to each classifier to identify if both samples come from the same data distribution. If not, a new classifier is stored alongside with the instances in the new buffer. If both samples come from the same data distribution, it is considered that this is an old concept that is occurring again. In this situation, the stored classifier is used as the actual classifier.

In [6], besides presenting how RCD works, we also show how RCD deals with concept drifts, comparing it with single and ensemble classifiers in environments with abrupt and gradual concept drifts. Artificial and real world data sets were tested using KNN as statistical test.

In this paper, RCD is tested using different base learners, in four different artificial data sets, with abrupt and gradual concept drift with various rates-of-change, and three real world data sets. One new implemented statistical test, Cramer, is also analyzed. It was based on the implementation of the Cramer package [7] of The R Project for Statistical Computing [25] tool.

RCD was configured with the following parameters: classifiers buffer size (100), test frequency (500), maximum number of classifiers to store (15), concept drift detection method (DDM), and minimum amount of similarity between distributions (0.05). RCD was implemented using of the MOA framework [26]. Instructions on how to download and use RCD can be found at https://sites.google.com/site/moaextensions/.

4. Algorithms and Datasets

In this section we present the algorithms used in the tests, their parameters, and information about the data sets.

4.1 Algorithms

Four base learners were used to test RCD: Hoeffding tree with naive Bayes at the leaves [13], naive Bayes, and
Multilayer Perceptron (MLP). From now on we will reference the decision tree as HTNB. We tested each base learner against rcd with KNN and Cramer statistical tests.

The parameters used by HTNB in the experiments are the default values present in the MOA framework: number of instances a leaf should observe between split attempts (200), split criterion to use (information gain), allowable error in split decision \((10^{-7})\), and threshold below which a split will be forced to break ties (0.05).

The parameters used by MLP in the experiments are the default values present in the WEKA tool [27]: learning rate (0.3), momentum (0.2). Only the number of epochs to train was reduced from 500 to 100 to increase its speed.

The following sections presents the data sets used in the comparison of the framework. The data sets used are well known in the field and have been used in several previous experiments.

4.2 Artificial data sets

We first describe the artificial data sets used to test the algorithms. All these data sets are available through the MOA framework.

4.2.1 Hyperplane

Hyperplane [9] is an artificial data set that simulates concept drifts through a moving hyperplane. A hyperplane in a \(d\) dimensional space is the set of points \(x\) that satisfies \(\sum_{i=1}^{d} w_i x_i = w_0\), where \(x_i\) is the \(i^{th}\) coordinate of \(x\). Examples where \(\sum_{i=1}^{d} w_i x_i \geq w_0\) are classified as positive, and examples where \(\sum_{i=1}^{d} w_i x_i < w_0\) are classified as negative. Hyperplanes are used to simulate gradual concept drifts where it is possible to smoothly change the orientation and position of the hyperplane by modifying its weights. It is possible to introduce changes in the data set changing the weight of each attribute \(w_i = w_i + d \sigma\), where \(\sigma\) is the chance the direction of change be inverted and \(d\) is the amount of change applied to each example.

4.2.2 LED

The LED [28] data set represents the problem of predicting the digit shown by a seven-segment LED display. It is composed of 24 nominal attributes, where 17 are irrelevant, and one nominal class with ten possible values. Noise was added by including a 10% probability of each attribute being inverted. The version of LED used in the experiments is available at MOA that includes concept drifts to the data sets by simply changing the attributes positions. The number of drifting attributes chosen were 1, 3, 5, and 7.

\[(((LED_1 W LED_3) W LED_5) W LED_7)\]

4.2.3 SEA

The SEA [18] concepts are commonly used to test abrupt concept drifts. The values of each of its three attributes are in the interval \([0,10)\), but the third one is irrelevant. In each concept, a data point belongs to class 1 if \(f_1 + f_2 \leq \theta\), where \(f_1\) and \(f_2\) represent the first two features and \(\theta\) is a threshold value between the two classes. For the experiments, we used the thresholds proposed by [18]: 8, 9, 7, 9.5 to represent four concepts. Noise was inserted by randomly changing the class value of 10% of the instances. In the following tests, SEA concepts are defined as follows:

\[(((SEA_0 W SEA_8) W SEA_7) W SEA_{9.5})\]

4.2.4 Random RBF

RBF (Radial Basis Function) [29] creates complex concept drifts that are not straightforward to approximate with a decision tree model. It works as follows: a fixed number of random centroids are generated where each center has a random position, a single standard deviation, a class label and a weight. New examples are generated by selecting a center at random, taking weights into consideration so that centers with higher weight are more likely to be chosen. A random direction is chosen to offset the attribute values from the central point. The length of the displacement is randomly drawn from a Gaussian distribution with standard deviation determined by the chosen centroid. The chosen centroid also determines the class label of the example. This effectively creates a normally distributed hypersphere of examples surrounding each central point with varying densities. Only numeric attributes are generated. Drift is introduced by moving the centroids with constant speed. This speed is initialized by a drift parameter.

4.3 Real data sets

Here we present three real data sets used in the experiments. It is not easy to find large real-world datasets for public benchmarking, especially with substantial concept change. Another problem is that we do not know when drift occurs or if there is any drift at all. The data sets used are: Forest Covertype [30], Poker-Hand [29], and Electricity [15]. They were obtained from the MOA web site, in the following address: http://moa.cs.waikato.ac.nz/.

4.3.1 Forest Covertype

This data set contains the forest cover type for 30 x 30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. The goal is to predict the forest cover type from cartographic variables. It contains 581,012 instances and 54 attributes.
4.3.2 Poker Hand

The Poker Hand data set is constituted of five categoric and five numeric attributes plus one categoric class with 10 possible values informing the value of the hand, for example, one pair, two pairs, a sequence, a street flush, etc. It represents the problem of identifying the value of five cards in the Poker game. Bifet [31] describes that “in the Poker hand data set, the cards are not ordered, i.e., a hand can be represented by any permutation, which makes it very hard for propositional learners, especially for linear ones”. So, in the experiments, we used a modified version, where the cards are sorted by rank and suit, and duplicates were removed. This data set is composed of 829,201 instances.

4.3.3 Electricity

This data set presents data collected from the Australian New South Wales Electricity Market. In that market, prices vary according to market demand and supply. Prices are set every five minutes and the class label identifies the change of the price related to a moving average of the last 24 hours. The goal of the problem is to predict if the price will increase or decrease. It is composed of 45,312 instances.

5. Empirical Study and Results

The evaluation methodology used in the presented data sets were the Interleaved Test-Then-Train approach. Every example is used for testing the model, then it is used to train it. The accuracy was measured as the final percentage of instances correctly classified over the interleaved evaluation. For the Hyperplane and Random RBF data sets, 10 million instances were generated; for LED and SEA, 1 million instances. The experiments were repeated 10 times. The parameters of these streams are the following:

- $RBF(x,v)$: Random RBF data stream with $x$ centroids moving at speed $v$.
- $HYP(x,v)$: Hyperplane data stream with $x$ attributes changing at speed $v$.
- $SEA(v)$: SEA data set, with length of change $v$.
- $LED(v)$: LED data set, with length of change $v$.

The chosen evaluation methodology, data sets, and configurations are exactly the same as used in [29], [31]. For the tests, we used an Intel Core i3 330M processor (with two cores and emulating two other), with 4GB of main memory.

Table 1 presents how long each statistical test needs to evaluate samples of sizes varying from 100 to 300 instances. Samples were obtained from the Electricity data set. We can see that the time spent by the Cramer test almost doubles each time the sample size is increased by 50 instances while KNN is less impacted. Comparing both tests, it is clear that KNN is considerably faster than Cramer performing the statistical tests. Thus, considering the data set sizes used in the experiments (1 to 10 million, with 10 repetitions), the buffer size used in RCD was 100 to increase its speed.

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>250</th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cramer</td>
<td>0.413</td>
<td>0.947</td>
<td>1.835</td>
<td>3.263</td>
<td>6.375</td>
</tr>
<tr>
<td>KNN</td>
<td>0.003</td>
<td>0.007</td>
<td>0.012</td>
<td>0.017</td>
<td>0.024</td>
</tr>
</tbody>
</table>

Tables 2, 3 and 4 present the average evaluation times in seconds and the accuracy of the HTNB, naive Bayes and MLP base classifiers, respectively. For the artificial data sets, the accuracy also contains the 95% confidence interval. The second column presents the results for the base classifier, the third one, RCD using the Cramer test, and the latter, RCD using the KNN test. The last column of Cramer and KNN presents the average number of drifts detected in all repetitions in the artificial data sets, and the total number of drifts in the real-world data sets. Best results in performance are highlighted in boldface and the values marked by an asterisk (*) represent statistically significant differences between RCD and the base learner.

Analyzing the results presented at Table 2, it is possible to observe that RCD outperformed the HTNB base classifier in the majority of the artificial data sets, independently of the statistical test used. Statistically, RCD had better performance in HYP(10,0.0001), HYP(10,0.001) (when using the KNN statistical test), and in the Random RBF configurations with 50 centroids. In all the other configurations, both algorithms had statistical similar performances.

In the Random RBF configuration without concept drifts, RBF(0,0), all algorithms had exactly the same results. This is an expected result of RCD because, when no concept drift is detected, no statistical test is performed and RCD performs the same as the base learner. In the other RBF data sets with concept drift, the configurations that RCD performed better than the base learner was the ones with 50 centroids. For this data set, the higher the number of moving centroids, the bigger the number of concept drifts. Thus, RCD had better performance exactly in the configurations with higher number of concept drifts while in the configurations with 10 centroids, the ones with lower number of concept drifts, the performances were similar.

It is possible to check the influence of the number of moving centroids compared to the average number of detected concept drifts in each configuration. In the RBF(50,0.001) configuration, using the Cramer test, 17.2 concept drifts were detected, while using KNN there were 15.4 in the ten repetitions. In the RBF(10,0.001) configuration, only 0.4 and 0.3 concept drifts were detected using the Cramer and KNN statistical tests, respectively, in the ten repetitions.

Comparing the usage of the two statistical tests, KNN had a better performance in two data sets: in RBF(50,0.0001), and in the LED data set. In the rest, both statistical tests had comparable performances. Analyzing the number of detected concept drifts, KNN is much more sensible than Cramer. Using KNN, 40,480 concept drifts were detected.
while Cramer detected 33,755 in all data sets, considering all repetitions.

Regarding the time spent in the evaluation procedure, as expected, the base learner was faster than the RCD framework. In average, HTNB was approximately 75.41% faster than RCD using the Cramer test and 45.19% using KNN. In 12 out of 13 data sets, KNN was faster than Cramer.

The results presented for the naive Bayes base learner in Table 3, show that, again, the RCD framework had better performance in the majority of the data sets and configurations. Statistically, the results were similar to the ones obtained using the HTNB base learner: RCD also had higher accuracy values in both configurations of Random RBF with 50 centroids and, in this case, in both configurations of the Hyperplane data set. In the other data set configurations, RCD and naive Bayes obtained statistically similar results.

Considering RBF(0,0), two concept drifts occurred in the ten repetitions. The results of the algorithms are practically the same, only differing in the fifth decimal. This is a similar result compared to the same configuration using HTNB: when no concept drift was found, the same results occurred; when only two concept drifts were detected, the accuracy difference was negligible. Again, it is possible to notice that higher number of centroids gives higher number of concept drifts. Using the Cramer test, there were an average of 10.0 concept drifts in RBF(50.0.0001) and 5.7 with 10 centroids. The same behavior occurred when using the KNN test: 16.7 and 5.5 concept drifts, respectively.

Comparing the statistical tests, KNN was statistically more accurate than Cramer in HYP(10.0.0001) and in both configurations of Random RBF with 50 centroids. In all other situations, both statistical tests had comparable performances. Comparing the number of detected concept drifts, similar results to the ones using HTNB base learner were obtained. Using KNN, 60,240 concept drifts were detected while Cramer detected 38,985 in all data sets.

Again, the base learner was faster than RCD in evaluating the data sets. The base learner was 2.5 times faster than RCD using the Cramer test, and 72% faster using KNN. The KNN statistical test was faster than the Cramer test in 9 out of 13 data set configurations.

Table 4 presents the results for the MLP base learner. The evaluation results show that, statistically, RCD had higher accuracy values compared to MLP only in RBF(50.0.0001). In all other situations, RCD and MLP had comparable performances, independently of the statistical test used.

In RBF without concept drifts, both versions of RCD performed the same, like when using the other base learners. In the 10 repetitions, only 2 concept drifts were found. These were false positives raised by the drift detector, giving the slightly different results from RCD and MLP. Similarly from the other base learners, RCD was statistically better in RBF(50.0.0001). Again it is possible to verify that the number of centroids has a much higher influence in the number of concept drifts than the speed of change. For example, using the Cramer test, an average of 6.8 concept drifts were identified in the RBF(10.0.001) configuration. Increasing the number of centroids to 50, the number of concept drifts raised to 14.0, while changing the speed of change to 0.0001 augmented the number of concept drifts to 8.0.

Comparing Cramer to KNN, only in the RBF(50.0.0001) configuration the tests had statistical different performances: KNN was better. In all other data sets, they performed similarly. Analyzing the evaluation time, the base learner was again faster. Here, Cramer was faster than KNN in average. While Cramer spent more than two times compared to MLP in the evaluation procedure, KNN was more than three times slower than the base learner. Again, KNN identified more concept drifts than Cramer: 34,348 versus 32,398.

6. Conclusion

This paper presented the implementation of a new multivariate non-parametric statistical test, the Cramer test, in the RCD framework. We compared RCD using two multivariate non-parametric statistical tests (KNN and Cramer) with three
base learners: Hoeffding trees, naive Bayes, and Multilayer Perceptron. The tests were performed in seven data sets: four artificial data sets, with several rates-of-change, and three real world data sets.

The RCD configuration tested presented better performance than the Hoeffding tree base learner with naive Bayes at the leaves in 8 out of 13 possible situations using the Cramer test, in 4 situations the Hoeffding tree performed better, and in 1 situation the performance was similar. Using KNN, RCD has beaten the base learner by 9 to 2. In two situations, the performance was similar. Statistically, in three data set configurations, RCD had a better performance than the base learners considering both statistical tests. In all other situations, the performances were similar.

Using the naive Bayes base learner, the results of RCD are even better considering the Cramer test: of the 13 data set configurations, RCD performed better than naive Bayes in 11. In one situation they performed similarly and in one naive Bayes was better. Using KNN, RCD was better in ten data set configurations and in three the results were similar. Statistically, RCD performed better in four situations: both versions of Hyperplane and Random RBF with 50 centroids.

The results indicate that RCD tends to perform better than the base learners in environments with many concept drifts. This is an expected behavior because RCD stores classifiers and reuses them if actual data is similar to the ones used to build it. In environments without concept drifts, RCD’s performance is similar to the base classifier, as the results from the tests shows in the Random RBF(0,0) configuration using the Hoeffding tree as base learner.

Regarding the two statistical tests in the artificial data sets, KNN statistically outperformed Cramer in six situations. In the others, both tests performed similarly. Considering the real data sets, KNN has beaten Cramer by 8 to 1. Results from the experiments clearly show that KNN is better suited for concept drift detection than Cramer.

Considering the evaluation times, as expected, the base learner was faster than using RCD. Comparing the two statistical tests, using the Hoeffding tree and naive Bayes, KNN was faster, while Cramer was faster when using Multilayer Perceptron. In all the tests, KNN was faster than Cramer in 28 situations, while the opposite occurred 11 times.

These results confirmed previous findings that the RCD approach to handle concept drifts is promising and improves single classifiers results when using them as base learners of the framework, independently of the statistical test used.

### Table 3: Results using the naive Bayes base classifier.

<table>
<thead>
<tr>
<th>Time</th>
<th>Accuracy</th>
<th>Time</th>
<th>Accuracy</th>
<th># Drifts</th>
<th>Time</th>
<th>Accuracy</th>
<th># Drifts</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBF(0,0)</td>
<td>53.90</td>
<td>72.02±0.02</td>
<td>93.94</td>
<td>72.02±0.02</td>
<td>0.2</td>
<td>92.68</td>
<td>72.02±0.02</td>
</tr>
<tr>
<td>RBF(50,0.001)</td>
<td>117.17</td>
<td>53.18±0.01</td>
<td>161.20</td>
<td>53.19±0.01*</td>
<td>10.0</td>
<td>162.71</td>
<td>53.22±0.02*</td>
</tr>
<tr>
<td>RBF(10,0.0001)</td>
<td>66.40</td>
<td>75.78±0.03</td>
<td>105.95</td>
<td>75.79±0.03*</td>
<td>5.7</td>
<td>103.61</td>
<td>75.78±0.03</td>
</tr>
<tr>
<td>RBF(50,0.0001)</td>
<td>116.78</td>
<td>53.23±0.05</td>
<td>160.61</td>
<td>53.33±0.03*</td>
<td>13.6</td>
<td>160.86</td>
<td>53.76±0.31*</td>
</tr>
<tr>
<td>RBF(10,0.0001)</td>
<td>66.43</td>
<td>75.77±0.04</td>
<td>106.29</td>
<td>75.76±0.04</td>
<td>6.8</td>
<td>105.09</td>
<td>75.77±0.04</td>
</tr>
<tr>
<td>HYP(10,0.001)</td>
<td>54.91</td>
<td>87.10±3.14</td>
<td>96.88</td>
<td>87.64±2.78*</td>
<td>7.6</td>
<td>96.14</td>
<td>87.91±2.47*</td>
</tr>
<tr>
<td>HYP(10,0.0001)</td>
<td>54.90</td>
<td>88.34±1.23</td>
<td>97.08</td>
<td>88.71±1.24*</td>
<td>9.3</td>
<td>95.03</td>
<td>88.84±1.25*</td>
</tr>
<tr>
<td>SEA(50)</td>
<td>2.60</td>
<td>87.71±1.33</td>
<td>4.35</td>
<td>87.86±1.22</td>
<td>0.3</td>
<td>4.29</td>
<td>87.86±1.22</td>
</tr>
<tr>
<td>SEA(50000)</td>
<td>2.51</td>
<td>87.71±1.32</td>
<td>4.36</td>
<td>88.42±0.35</td>
<td>0.8</td>
<td>4.20</td>
<td>88.42±0.35</td>
</tr>
<tr>
<td>LED(50000)</td>
<td>17.77</td>
<td>72.58±2.36</td>
<td>31.89</td>
<td>73.13±1.25</td>
<td>0.2</td>
<td>32.59</td>
<td>73.13±1.25</td>
</tr>
<tr>
<td>Covertype</td>
<td>29.13</td>
<td>60.52</td>
<td>391.97</td>
<td>61.68</td>
<td>251.3</td>
<td>128.54</td>
<td>85.47</td>
</tr>
<tr>
<td>Poker hand</td>
<td>15.09</td>
<td>59.55</td>
<td>231.88</td>
<td>60.25</td>
<td>1151</td>
<td>45.97</td>
<td>70.88</td>
</tr>
<tr>
<td>Electricity</td>
<td>0.44</td>
<td>73.36</td>
<td>26.01</td>
<td>73.58</td>
<td>180</td>
<td>2.56</td>
<td>82.21</td>
</tr>
</tbody>
</table>

### Table 4: Results using the MLP base classifier.

<table>
<thead>
<tr>
<th>Time</th>
<th>Accuracy</th>
<th>Time</th>
<th>Accuracy</th>
<th># Drifts</th>
<th>Time</th>
<th>Accuracy</th>
<th># Drifts</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP RCD</td>
<td>56.46</td>
<td><strong>87.70±0.57</strong></td>
<td>101.86</td>
<td>87.44±0.53</td>
<td>0.2</td>
<td>101.90</td>
<td>87.44±0.53</td>
</tr>
<tr>
<td>MLP RCD</td>
<td>120.26</td>
<td>50.28±0.24</td>
<td>174.82</td>
<td>50.27±0.19</td>
<td>14.0</td>
<td>188.73</td>
<td>50.30±0.33</td>
</tr>
<tr>
<td>MLP RCD</td>
<td>69.09</td>
<td>83.95±0.47</td>
<td>126.76</td>
<td>84.20±0.43</td>
<td>6.8</td>
<td>133.79</td>
<td>83.93±0.45</td>
</tr>
<tr>
<td>MLP RCD</td>
<td>120.23</td>
<td>49.92±0.26</td>
<td>176.99</td>
<td>50.10±0.26*</td>
<td>10.9</td>
<td>321.69</td>
<td>53.32±1.94*</td>
</tr>
<tr>
<td>MLP RCD</td>
<td>69.12</td>
<td>84.41±0.67</td>
<td>122.14</td>
<td>84.40±0.48</td>
<td>8.0</td>
<td>121.69</td>
<td>84.78±0.40</td>
</tr>
<tr>
<td>MLP RCD</td>
<td>56.45</td>
<td><strong>82.55±6.70</strong></td>
<td>111.98</td>
<td>82.24±7.33</td>
<td>8.6</td>
<td>110.47</td>
<td>82.27±7.36</td>
</tr>
<tr>
<td>MLP RCD</td>
<td>56.64</td>
<td>84.31±2.14</td>
<td>113.46</td>
<td>84.34±2.22</td>
<td>8.9</td>
<td>112.71</td>
<td><strong>84.43±2.22</strong></td>
</tr>
<tr>
<td>MLP RCD</td>
<td>4.13</td>
<td>87.99±1.68</td>
<td>6.97</td>
<td><strong>88.75±0.98</strong></td>
<td>1.0</td>
<td>7.03</td>
<td>88.74±0.98</td>
</tr>
<tr>
<td>MLP RCD</td>
<td>4.08</td>
<td>87.99±1.78</td>
<td>7.37</td>
<td><strong>88.04±1.70</strong></td>
<td>0.9</td>
<td>7.14</td>
<td><strong>88.04±1.70</strong></td>
</tr>
<tr>
<td>MLP RCD</td>
<td>27.48</td>
<td>63.67±4.73</td>
<td>50.84</td>
<td>63.68±4.72</td>
<td>0.5</td>
<td>51.05</td>
<td><strong>65.17±2.98</strong></td>
</tr>
<tr>
<td>Covertype</td>
<td>49.00</td>
<td><strong>60.69</strong></td>
<td>410.19</td>
<td>49.02</td>
<td>1661</td>
<td>388.15</td>
<td>49.05</td>
</tr>
<tr>
<td>Poker hand</td>
<td>23.35</td>
<td>43.03</td>
<td>265.26</td>
<td>59.80</td>
<td>1421</td>
<td>918.81</td>
<td><strong>63.47</strong></td>
</tr>
<tr>
<td>Electricity</td>
<td>1.53</td>
<td><strong>57.51</strong></td>
<td>20.76</td>
<td>49.30</td>
<td>98</td>
<td>7.68</td>
<td>55.49</td>
</tr>
</tbody>
</table>
There was no situation where the base learner had a statistically better performance compared to using RCD in the experiments. Tests using other two base learners, J48 [32] and a pure Hoeffding tree, also had the same results (not presented here due to limitations of space).

6.1 Future Work

Even though RCD is actually highly configurable, as future work, further research might be made to allow the user to choose the best configuration for a specific problem:

- Analyze the influence of the drift detection method, implementing and testing alternatives to DDM and EDDM, like EWMA.
- Create an ensemble classifier based on the individual learners associated with each distribution. As more than one stored classifier can match the actual context, one approach would be to set higher weights to classifiers where their samples return higher confidence levels.
- Implement a pruning strategy different than FIFO and analyze the influence of the number of stored classifiers in terms of accuracy and performance.
- Use the two available statistical tests in conjunction and analyze if they provide better results than their individual use.

References