CD-MOA: Change Detection Framework for Massive Online Analysis

Albert Bifet[1], Jesse Read[2], Bernhard Pfahringer[3], Geoff Holmes[3], and Indrė Žliobaitė[4]

[1] Yahoo! Research Barcelona, Spain
abifet@yahoo-inc.com
[2] Universidad Carlos III, Spain
jesse@tsc.uc3m.es
[3] University of Waikato, New Zealand
{bernhard,geoff}@waikato.ac.nz
[4] Aalto University and HIIT, Finland
indre.zliobaite@aalto.fi

Abstract. Analysis of data from networked digital information systems such as mobile devices, remote sensors, and streaming applications, needs to deal with two challenges: the size of data and the capacity to be adaptive to changes in concept in real-time. Many approaches meet the challenge by using an explicit change detector alongside a classification algorithm and then evaluate performance using classification accuracy. However, there is an unexpected connection between change detectors and classification methods that needs to be acknowledged. The phenomenon has been observed previously, connecting high classification performance with high false positive rates. The implication is that we need to be careful to evaluate systems against intended outcomes–high classification rates, low false alarm rates, compromises between the two and so forth. This paper proposes a new experimental framework for evaluating change detection methods against intended outcomes. The framework is general in the sense that it can be used with other data mining tasks such as frequent item and pattern mining, clustering etc. Included in the framework is a new measure of performance of a change detector that monitors the compromise between fast detection and false alarms. Using this new experimental framework we conduct an evaluation study on synthetic and real-world datasets to show that classification performance is indeed a poor proxy for change detection performance and provide further evidence that classification performance is correlated strongly with the use of change detectors that produce high false positive rates.

Keywords: data streams, incremental, dynamic, evolving, online.

1 Introduction

Real-time analytics is a term used to identify analytics performed taking into account recent data that is being generated in real time. The analytical models should be up-to-date to match the current distribution of data. To be able to do that, models should be able to adapt quickly. Drift detection is a very important component in adaptive modeling, detecting a change gives a signal about when to adapt models. Typically, the streaming error of predictive models is monitored and when the detector raises a change alarm, then the model is updated or replaced by a new one.

Currently, drift detection methods are typically evaluated by the final classification accuracy [1,2]. For example, in [2], the authors notice that for a real dataset, the Electricity Market Dataset [3], performance increases when there is a large number of false positives (or low ARL_0): "*Interestingly, the fact that the best performance is achieved with a low ARL_0 suggests that changes are occurring quite frequently.*" Thus, evaluating drift detection methods only using classifiers may not be informative enough, since the adaptation strategy occludes change detector performance. Further, given that classification is not the only context for change detection we need to match our drift detection evaluation methodologies with what it is that we want to achieve from the task as a whole.

This paper investigates change detection for real time predictive modeling, and presents the following contributions:

1. CD-MOA, a new experimental framework for evaluating concept drift detection methods,
2. MTR, a new measure of performance of a concept drift detection method.

It is important to note that the framework generalises to other tasks but in this paper our focus is on change detection in the context of classification. The proposed framework is intended to serve as a tool for the research community and industry data analysts for experimentally comparing and benchmarking change detection techniques on synthetic data where ground truth changes are known. On real data, the framework allows to find changes in time series, and monitor the error in classification tasks. The framework and the proposed techniques are implemented in the open source data stream analysis software MOA and are available online[1].

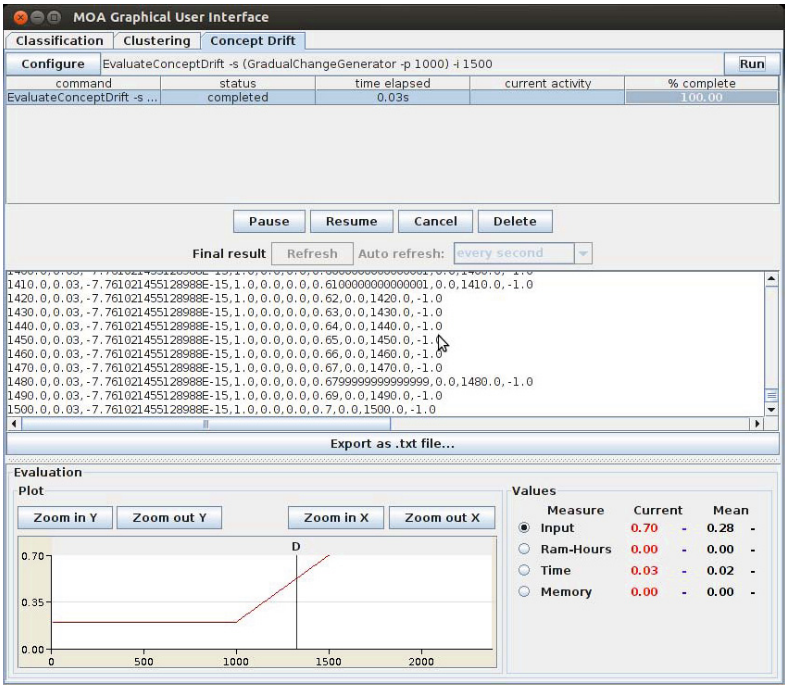
In Section 2 we present the new change detection framework and propose a new evaluation measure for change detection. Section 3 presents the results of our experimental evaluation. We conclude the study in Section 4.

2 Experimental Framework

CD-MOA is a new framework for comparing change detection methods. It is built as an extension of MOA. **M**assive **O**nline **A**nalysis (MOA) [4] is a software environment for implementing algorithms and running experiments for online learning from data streams.

CD-MOA contains a graphical user interface where experiments can be run. Figure 1 shows the GUI. It contains three different components. The first is the

[1] `http://moa.cs.waikato.ac.nz/`



```
EvaluateConceptDrift -s (GradualChangeGenerator -p 1000) -i 1500
```

command	status	time elapsed	current activity	% complete
EvaluateConceptDrift -s ...	completed	0.03s		100.00

```
1410.0,0.03,-7.761021455128988E-15,1.0,0.0,0.0,0.6100000000000001,0.0,1410.0,-1.0
1420.0,0.03,-7.761021455128988E-15,1.0,0.0,0.0,0.62,0.0,1420.0,-1.0
1430.0,0.03,-7.761021455128988E-15,1.0,0.0,0.0,0.63,0.0,1430.0,-1.0
1440.0,0.03,-7.761021455128988E-15,1.0,0.0,0.0,0.64,0.0,1440.0,-1.0
1450.0,0.03,-7.761021455128988E-15,1.0,0.0,0.0,0.65,0.0,1450.0,-1.0
1460.0,0.03,-7.761021455128988E-15,1.0,0.0,0.0,0.66,0.0,1460.0,-1.0
1470.0,0.03,-7.761021455128988E-15,1.0,0.0,0.0,0.67,0.0,1470.0,-1.0
1480.0,0.03,-7.761021455128988E-15,1.0,0.0,0.0,0.6799999999999999,0.0,1480.0,-1.0
1490.0,0.03,-7.761021455128988E-15,1.0,0.0,0.0,0.69,0.0,1490.0,-1.0
1500.0,0.03,-7.761021455128988E-15,1.0,0.0,0.0,0.7,0.0,1500.0,-1.0
```

	Measure	Current		Mean	
●	Input	0.70	-	0.28	-
○	Ram-Hours	0.00	-	0.00	-
○	Time	0.03	-	0.02	-
○	Memory	0.00	-	0.00	-

Fig. 1. Example of the new CD-MOA framework graphical user interface

panel where the user can specify the experiment they would like to run. Another panel in the middle shows the numeric results of the experiment, and the panel at the bottom displays a plot of the experiment, showing graphically where the change has been detected.

CD-MOA contains a Java API for easier customization, and implementation of new methods and experiments. The main components of CD-MOA are:

– Tasks: experiments to run combining change detectors and streams
– Methods: change detection algorithms used to detect change
– Streams: time series used to run the experiment. If they have been artificially generated and have ground truth, then the system will output the statistics about detection time.

CD-MOA is connected to MOA, and it is easy to use the change detector methods in CD-MOA to evaluate classifiers, checking how their accuracy evolves. For evaluation purposes, all the methods in CD-MOA have a measure of the resources consumed: time, memory, and RAM-Hours, a measure of the cost of the mining process that merges time and memory into a single measure.

2.1 Evaluation of Change Detection

Change detection is a challenging task due to a fundamental limitation [5]: the design of a change detector is a compromise between detecting true changes and avoiding false alarms.

When designing a change detection algorithm one needs to balance false and true alarms and minimize the time from the change actually happening to detection. The following existing criteria [5,6] formally capture these properties for evaluating change detection methods.

Mean Time between False Alarms (MTFA) characterizes how often we get false alarms when there is no change. The false alarm rate FAR is defined as 1/MTFA. A good change detector would have high MTFA.

Mean Time to Detection (MTD) characterizes the reactivity of the system to changes after they occur. A good change detector would have small MTD.

Missed Detection Rate (MDR) gives the probability of not receiving an alarm when there has been a change. It is the fraction of non-detected changes in all the changes. A good detector would have small or zero MDR.

Average Run Length (ARL(θ)) generalizes over MTFA and MTD. It quantifies how long we have to wait before we detect a change of size θ in the variable that we are monitoring.

$$ARL(\theta = 0) = MTFA, \quad ARL(\theta \neq 0) = MTD$$

Our framework needs to know ground truth changes in the data for evaluation of change detection algorithms. Thus, we generate synthetic datasets with ground truth. Before a true change happens, all the alarms are considered as false alarms. After a true change occurs, the first detection that is flagged is considered as the true alarm. After that and before a new true change occurs, the consequent detections are considered as false alarms. If no detection is flagged between two true changes, then it is considered a missed detection. These concepts are graphically illustrated in Figure 2.

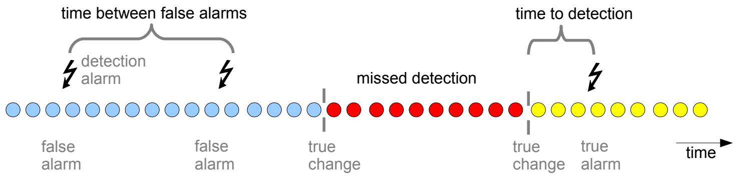


Fig. 2. The setting of change detection evaluation

We propose a new quality evaluation measure that monitors the compromise between fast detection and false alarms:

$$MTR(\theta) = \frac{MTFA}{MTD} \times (1 - MDR) = \frac{ARL(0)}{ARL(\theta)} \times (1 - MDR). \tag{1}$$

This measure MTR (Mean Time Ratio) is the ratio between the mean time between false alarms and the mean time to detection, multiplied by the probability of detecting an alarm. An ideal change detection algorithm would have a low false positive rate (which means a high mean time between false alarms), a low mean time to detection, and a low missed detection rate.

Comparing two change detectors for a specific change θ is easy with this new measure: the algorithm that has the highest $MTR(\theta)$ value is to be preferred.

2.2 Change Detectors

A *change detector* or *drift detector* is an algorithm that takes a stream of instances as input and outputs an alarm if it detects a change in the distribution of the data. A detector may often be combined with a predictive model to output a prediction of the next instance to come. In general, the input to a change detection algorithm is a sequence $x_1, x_2, \ldots, x_t, \ldots$ of data points whose distribution varies over time in an unknown way. At each time step the algorithm outputs:

1. an estimate of the parameters of the input distribution, and
2. an alarm signal indicating whether a change in this distribution has occurred.

We consider a specific, but very frequent case, of this setting with all x_t being real values. The desired estimate is usually the current expected value of x_t, and sometimes other statistics of the distribution such as, for instance, variance. The only assumption about the distribution of x is that each x_t is drawn independently from each other. This assumption may be not satisfied if x_t is an error produced by a classifier that updates itself incrementally, because the update depends on the performance, and the next performance depends on whether we updated it correctly. In practice, however, this effect is negligible, so treating them independently is a reasonable approach.

The most general structure of a change detection algorithm contains three components:

1. *Memory* is the component where the algorithm stores the sample data or data summaries that are considered to be relevant at the current time, i.e., the ones that describe the current data distribution.
2. *Estimator* is an algorithm that estimates the desired statistics on the input data, which may change over time. The algorithm may or may not use the data contained in Memory. One of the simplest Estimator algorithms is the *linear estimator,* which simply returns the average of the data items contained in Memory. Other examples of run-time efficient estimators are Auto-Regressive, Auto Regressive Moving Average, and Kalman filters [7].
3. *Change detector* (hypothesis testing) outputs an alarm signal when it detects a change in the input data distribution. It uses the output of the Estimator, and may or may not in addition use the contents of Memory.

There are many different algorithms to detect change in time series. Our new framework contains the classical ones used in statistical quality control [6], time series analysis [8], statistical methods and more recent ones such as `ADWIN`[9].

2.3 Statistical Tests with Stopping Rules

These tests decide between the hypothesis that there is change and the hypothesis that there is no change, using a stopping rule. When this stopping rule is achieved, then the change detector method signals a change. The following methods differ in their stopping rule.

The CUSUM Test. The cumulative sum (CUSUM algorithm), which was first proposed in [10], is a change detection algorithm that raises an alarm when the mean of the input data is significantly different from zero. The CUSUM input ϵ_t can be any filter residual, for instance the prediction error from a Kalman filter.

The stopping rule of the CUSUM test is as follows:

$$g_0 = 0, \qquad g_t = \max\,(0, g_{t-1} + \epsilon_t - \upsilon), \qquad \text{if } g_t > h \text{ then alarm and } g_t = 0$$

The CUSUM test is memoryless, and its accuracy depends on the choice of parameters υ and h. Note that CUSUM is a one sided, or *asymmetric* test. It assumes that changes can happen only in one direction of the statistics, detecting only increases.

The Page Hinckley Test. The Page Hinckley Test [10] stopping rule is as follows, when the signal is increasing:

$$g_0 = 0, \qquad g_t = g_{t-1} + (\epsilon_t - \upsilon), \qquad G_t = \min(g_t, G_{t-1})$$

$$\text{if } g_t - G_t > h \text{ then alarm and } g_t = 0$$

When the signal is decreasing, instead of $G_t = \min(g_t, G_{t-1})$, we should use $G_t = \max(g_t, G_{t-1})$ and $G_t - g_t > h$ as the stopping rule. Like the CUSUM test, the Page Hinckley test is memoryless, and its accuracy depends on the choice of parameters υ and h.

2.4 Drift Detection Method

The drift detection method (DDM) proposed by Gama et al. [1] controls the number of errors produced by the learning model during prediction. It compares the statistics of two windows: the first contains all the data, and the second contains only the data from the beginning until the number of errors increases. Their method doesn't store these windows in memory. It keeps only statistics and a window of recent errors data.

The number of errors in a sample of n examples is modelled by a binomial distribution. For each point t in the sequence that is being sampled, the error rate is the probability of misclassifying (p_t), with standard deviation given by $s_t = \sqrt{p_t(1-p_t)/t}$. They assume that the error rate of the learning algorithm (p_t) will decrease while the number of examples increases if the distribution of the examples is stationary. A significant increase in the error of the algorithm, suggests that the class distribution is changing and, hence, the actual decision model is supposed to be inappropriate. Thus, they store the values of p_t and s_t when $p_t + s_t$ reaches its minimum value during the process (obtaining p_{min} and s_{min}). DDM then checks if the following conditions trigger:

- $p_t + s_t \geq p_{min} + 2 \cdot s_{min}$ for the warning level. Beyond this level, the examples are stored in anticipation of a possible change of context.
- $p_t + s_t \geq p_{min} + 3 \cdot s_{min}$ for the drift level. Beyond this level the concept drift is supposed to be true, the model induced by the learning method is reset and a new model is learnt using the examples stored since the warning level triggered. The values for p_{min} and s_{min} are reset.

In the standard notation, they have two hypothesis tests h_w for warning and h_d for detection:

- $g_t = p_t + s_t$, if $g_t \geq h_w$ then alarm warning, if $g_t \geq h_d$ then alarm detection, where $h_w = p_{min} + 2s_{min}$ and $h_d = p_{min} + 3s_{min}$.

The test is nearly memoryless, it only needs to store the statistics p_t and s_t, as well as switch on some memory to store an extra model of data from the time of warning until the time of detection.

This approach works well for detecting abrupt changes and reasonably fast changes, but it has difficulties detecting slow gradual changes. In the latter case, examples will be stored for long periods of time, the drift level can take too much time to trigger and the examples in memory may overflow.

Baena-García et al. proposed a new method EDDM (Early Drift Detection Method) [11] in order to improve DDM. It is based on the estimated distribution of the distances between classification errors. The window resize procedure is governed by the same heuristics.

2.5 EWMA Drift Detection Method

A new drift detection method based on an EWMA (Exponential Weighted Moving Average) chart, was presented by Ross et al. in [2]. It is similar to the drift detection method (DDM) described previously, but uses an exponentially weighted moving average chart to update the estimate of error faster.

This method updates the following statistics for each point t in the sequence:

$$p_t = p_{t-1}(t-1)/t + \epsilon_t/t, \qquad s_t = \sqrt{p_t(1-p_t)}$$

$$g_0 = p_0, \qquad g_t = (1-\lambda)g_{t-1} + \lambda\epsilon_t, \qquad s_t^{(g)} = s_t\sqrt{\lambda(1-(1-2\lambda)^{2t})/(2-\lambda)}$$

EWMA uses the following trigger conditions:

- $g_t > h_w$ for the warning level, where $h_w = p_t + 0.5 L_t s_t^{(g)}$.
- $g_t > h_d$ for the drift level, where $h_d = p_t + L_t s_t^{(g)}$.

The values of L_t are computed using a different polynomial for each choice of $MTFA$ of the form $L(p_t) = c_0 + c_1 p_t + \cdots + c_m p_t^m$ using a Monte Carlo approach. A value of $\lambda = 0.2$ is recommended by the authors of this method.

2.6 ADWIN: ADaptive Sliding WINdow Algorithm

ADWIN[12] is a change detector and estimator that solves in a well-specified way the problem of tracking the average of a stream of bits or real-valued numbers. ADWIN keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis "there has been no change in the average value inside the window".

More precisely, an older fragment of the window is dropped if and only if there is enough evidence that its average value differs from that of the rest of the window. This has two consequences: one, that change can reliably be declared whenever the window shrinks; and two, that at any time the average over the existing window can be reliably taken as an estimate of the current average in the stream (barring a very small or very recent change that is still not statistically visible). These two points appears in [12] in a formal theorem.

ADWIN is data parameter- and assumption-free in the sense that it automatically detects and adapts to the current rate of change. Its only parameter is a confidence bound δ, indicating how confident we want to be in the algorithm's output, inherent to all algorithms dealing with random processes.

Table 1. Evaluation results for an experiment simulating the error of a classifier, that after t_c instances with a probability of having an error of 0.2, this probability is increased linearly by a value of $\alpha = 0.0001$ for each instance

Method	Measure	No Change	$t_c = 1,000$	$t_c = 10,000$	$t_c = 100,000$	$t_c = 1,000,000$
ADWIN	1-MDR		0.13	1.00	1.00	1.00
	MTD		111.26	1,062.54	1,044.96	1,044.96
	MTFA	5,315,789				
	MTR		6,150	5,003	5,087	5,087
CUSUM(h=50)	1-MDR		0.41	1.00	1.00	1.00
	MTD		344.50	902.04	915.71	917.34
	MTFA	59,133				
	MTR		70	66	65	64
DDM	1-MDR		0.44	1.00	1.00	1.00
	MTD		297.60	2,557.43	7,124.65	42,150.39
	MTFA	1,905,660				
	MTR		2,790	745	267	45
Page-Hinckley(h=50)	1-MDR		0.17	1.00	1.00	1.00
	MTD		137.10	1,320.46	1,403.49	1,431.88
	MTFA	3,884,615				
	MTR		4,769	2,942	2,768	2,713
EDDM	1-MDR		0.95	1.00	1.00	1.00
	MTD		216.95	1,317.68	6,964.75	43,409.92
	MTFA	37,146				
	MTR		163	28	5	1
EWMA Chart	1-MDR		1.00	1.00	1.00	1.00
	MTD		226.82	225.51	210.29	216.70
	MTFA	375				
	MTR		2	2	2	2

Table 2. Evaluation results for an experiment simulating the error of a classifier, that after $t_c = 10,000$ instances with a probability of having an error of 0.2, this probability is increased linearly by a value of α for each instance

Method	Measure	No Change	$\alpha = 0.00001$	$\alpha = 0.0001$	$\alpha = 0.001$
ADWIN	1-MDR		1.00	1.00	1.00
	MTD		4,919.34	1,062.54	261.59
	MTFA	5,315,789.47			
	MTR		1,080.59	5,002.89	20,320.76
CUSUM	1-MDR		1.00	1.00	1.00
	MTD		3,018.62	902.04	277.76
	MTFA	59,133.49			
	MTR		19.59	65.56	212.89
DDM	1-MDR		0.55	1.00	1.00
	MTD		3,055.48	2,557.43	779.20
	MTFA	1,905,660.38			
	MTR		345.81	745.15	2,445.67
Page-Hinckley	1-MDR		1.00	1.00	1.00
	MTD		4,659.20	1,320.46	405.50
	MTFA	3,884,615.38			
	MTR		833.75	2,941.88	9,579.70
EDDM	1-MDR		0.99	1.00	1.00
	MTD		4,608.01	1,317.68	472.47
	MTFA	37,146.01			
	MTR		7.98	28.19	78.62
EWMA Chart	1-MDR		1.00	1.00	1.00
	MTD		297.03	225.51	105.57
	MTFA	374.70			
	MTR		1.26	1.66	3.55

ADWIN does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique storing a window of length W using only $O(\log W)$ memory and $O(\log W)$ processing time per item.

3 Comparative Experimental Evaluation

We performed a comparison using the following methods: DDM, ADWIN, EWMA Chart for Drift Detection, EDDM, Page-Hinckley Test, and CUSUM Test. The two last methods were used with $\upsilon = 0.005$ and $h = 50$ by default.

The experiments were performed simulating the error of a classifier system with a binary output 0 or 1. The probability of having an error is maintained as 0.2 during the first t_c instances, and then it changes gradually, linearly increasing by a value of α for each instance. The results were averaged over 100 runs.

Tables 1 and 2 show the results. Every single row represents an experiment where four different drifts occur at different times in Table 1, and four different drifts with different incremental values in Table 2. Note that MTFA values come

Table 3. Evaluation results of a prequential evaluation using an adaptive Naive Bayes classifier on Electricity and Forest Covertype datasets: accuracy, κ, and number of changes detected

		Forest Covertype			Electricity		
Change Detector	Warning	Accuracy	κ	Changes	Accuracy	κ	Changes
`ADWIN`	No	83.24	73.25	1,151	81.03	60.79	88
CUSUM	No	81.55	70.66	286	79.21	56.83	28
DDM	Yes	88.03	80.78	4,634	81.18	61.14	143
Page-Hinckley	No	80.06	68.40	117	78.04	54.43	10
EDDM	Yes	86.08	77.67	2,416	84.83	68.96	203
EWMA Chart	Yes	**90.16**	**84.20**	6,435	**86.76**	**72.93**	426

from the no-change scenario. We observe the tradeoff between faster detection and smaller number of false alarms. Page Hinckley with $h = 50$ and `ADWIN` are the methods with fewer false positives, however CUSUM is faster at detecting change for some change values. Using the new measure MTR, `ADWIN` seems to be the algorithm with the best results.

We use the EWMA Chart for Drift Detection with L_t values computed for a MTFA of 400. However it seems that this is a very low value compared with other change detectors. EDDM has a high number of false positives, and performs worse than DDM using the new measure MTR.

This type of test, has the property that by increasing h we can reduce the number of false positives, at the expense of increasing the detection delay.

Finally, we use the change detector algorithms inside the MOA Framework in a real data classification task. The methodology is similar to the one in [1]: a classifier is built with a change detector monitoring its error. If this change detector detects a warning, the classifier begins to store instances. After the change detector detects change, the classifier is replaced with a new classifier built with the instances stored. Note that this classifier is in fact similar to a data stream classification algorithm that exploits a window model. The size of this window model is not fixed and depends on the change detection mechanism. We test the change detectors with a Naive Bayes classifier, and the following datasets:

Forest Covertype Contains the forest cover type for 30 x 30 meter cells obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. It contains $581,012$ instances and 54 attributes. It has been used before, for example in [13,14].

Electricity Contains $45,312$ instances describing electricity demand. A class label identifies the change of the price relative to a moving average of the last 24 hours. It was described by [3] and analysed also in [1].

The accuracy and κ statistic results using prequential evaluation are shown in Table 3. The classifier that uses the EWMA Chart detection method is the method with the best performance on the two datasets. It seems that having a large amount of false positives, detecting more changes, and rebuilding the classifier more often with more recent data helps to improve accuracy for these datasets. For the classification setting, the fact that the detector has a warning signal detection helps to improve results. Also, the success of EWMA Chart may be due to the fact that Naive Bayes is a high-bias algorithm, which can attain good performance from smaller batches of data.

However, as we have seen, the fact that a change detection algorithm produces high accuracy figures in a classification setting does not necessarily imply a low false alarm rate or a high MTR value for that algorithm. It should be possible, for example, to tune all the detectors in the framework to produce better classification results by deliberately raising their false positive rates. Additionally, it should be possible to demonstrate that under normal circumstances, low-bias algorithms suffer high false positive rates.

4 Conclusions

Change detection is an important component of systems that need to adapt to changes in their input data. We have presented a new framework for evaluation of change detection methods, and a new quality measure for change detection. Using this new experimental framework we demonstrated that classification performance is a poor proxy for change detection performance and provide further evidence that if high classification performance is a requirement then using a change detector that produces a high false positive rate can be beneficial for some datasets. We hope that the new framework presented here will help the research community and industry data analysts to experimentally compare and benchmark change detection techniques.

Acknowledgements. I. Žliobaitė's research has been supported by the Academy of Finland grant 118653 (ALGODAN).

References

1. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171, pp. 286–295. Springer, Heidelberg (2004)
2. Ross, G.J., Adams, N.M., Tasoulis, D.K., Hand, D.J.: Exponentially weighted moving average charts for detecting concept drift. Pattern Recognition Letters 33(2), 191–198 (2012)
3. Harries, M.: Splice-2 comparative evaluation: Electricity pricing. Technical report, The University of South Wales (1999)
4. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: Massive online analysis. Journal of Machine Learning Research 11, 1601–1604 (2010)
5. Gustafsson, F.: Adaptive Filtering and Change Detection. Wiley (2000)
6. Basseville, M., Nikiforov, I.V.: Detection of abrupt changes: theory and application. Prentice-Hall, Inc., Upper Saddle River (1993)

7. Kobayashi, H., Mark, B.L., Turin, W.: Probability, Random Processes, and Statistical Analysis. Cambridge University Press (2011)
8. Takeuchi, J., Yamanishi, K.: A unifying framework for detecting outliers and change points from time series. IEEE Transactions on Knowledge and Data Engineering 18(4), 482–492 (2006)
9. Bifet, A., Gavaldà, R.: Adaptive learning from evolving data streams. In: Adams, N.M., Robardet, C., Siebes, A., Boulicaut, J.-F. (eds.) IDA 2009. LNCS, vol. 5772, pp. 249–260. Springer, Heidelberg (2009)
10. Page, E.S.: Continuous inspection schemes. Biometrika 41(1/2), 100–115 (1954)
11. Baena-García, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldá, R., Morales-Bueno, R.: Early drift detection method. In: Fourth International Workshop on Knowledge Discovery from Data Streams (2006)
12. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: SIAM International Conference on Data Mining (2007)
13. Gama, J., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 523–528 (2003)
14. Oza, N.C., Russell, S.J.: Experimental comparisons of online and batch versions of bagging and boosting. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 359–364 (2001)