Change Detection in Categorical Evolving Data Streams

Dino Ienco
IRSTEA, Montpellier, France
LIRMM, Montpellier, France
dino.ienco@teledetection.fr

Albert Bifet
Yahoo! Research Barcelona,
Catalonia, Spain
abifet@yahoo-inc.com

Bernhard Pfahringer
University of Waikato,
Hamilton, New Zealand
bernhard@cs.waikato.ac.nz

Pascal Poncelet
LIRMM, Montpellier, France
IRSTEA, Montpellier, France
poncelet@lirmm.fr

ABSTRACT

Detecting change in evolving data streams is a central issue for accurate adaptive learning. In real world applications, data streams have categorical features, and changes induced in the data distribution of these categorical features have not been considered extensively so far. Previous work on change detection focused on detecting changes in the accuracy of the learners, but without considering changes in the data distribution.

To cope with these issues, we propose a new unsupervised change detection method, called *CDCStream* (*Change Detection in Categorical Data Streams*), well suited for categorical data streams. The proposed method is able to detect changes in a batch incremental scenario. It is based on the two following characteristics: (i) a summarization strategy is proposed to compress the actual batch by extracting a descriptive summary and (ii) a new segmentation algorithm is proposed to highlight changes and issue warnings for a data stream. To evaluate our proposal we employ it in a learning task over real world data and we compare its results with state of the art methods. We also report qualitative evaluation in order to show the behavior of *CDCStream*.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Knowledge Management Applications—*data exploration and discovery*

General Terms

Theory, Algorithms, Experimentation

Keywords

Evolving data stream, categorical data, unsupervised change detection, statistical test, concept drifts

1. INTRODUCTION

Many real world applications continuously generate huge amounts of data, such as web logs, sensor networks, business transactions, etc. These *data streams* [1], due to the big volumes of information they contain, pose serious issues for the research community in order to extract useful and update-to-date knowledge in real-time. Due to its intrinsic temporal dimension, the information available in data streams changes and evolves over the time. More precisely, this phenomenon impacts on the performance of any supervised (or unsupervised) model learnt over these evolving data streams: previous models may not be suitable for newly incoming data [1]. Therefore we need to be able to adapt models both quickly and accurately. In particular, different types of change may happen in the stream. For instance, classes or concepts that can be underrepresented during a short period can become overrepresented after a longer period. Detecting change is essential for adapting learned models.

Most of the time, a common assumption made by many research works is to take into account only the aposteriori probability of the class given the data [8]. Actually, this formulation of the change detection task does not exploit information coming from the underlying data distribution and, generally, it is not suitable for unsupervised tasks where the class information is not available at all. As it is generally known, obtaining labels for data (especially in a stream context) can be very time-consuming and may require intensive human effort that is not always possible [11]. This issue can limit the use of such techniques that depend on proper class labels.

Secondly, in real world applications, data is heterogeneous and often can be represented over set of categorical attributes as well as numerical ones. In the last decade, lots of approaches have been defined to monitor classification accuracy as evidence or an indication for change in streams of numerical data (e.g. [8]). There are very few approaches dealing with the same problem which are able to handle categorical data [4].

The necessity to address all these points motivates us to propose a new change detection method especially tailored for categorical data: *CDCStream* (**C**hange **D**etection in **C**ategorical data **Stream**). Our approach is able to detect and highlight changes in categorical data streams in a fully unsupervised setting. Interestingly, our method is able to efficiently exploit historical data to make its decision. It works in the batch size scenario: when a new batch arrives,

SAC'14 March 24-28, 2014, Gyeongju, Korea.

firstly *CDCStream* compresses it by extracting a descriptive summary and it then performs a statistical test to evaluate if change has happened in the data distribution, or not. The developed algorithm supplies a segmentation approach that can also work with other statistics. This means that it can be coupled with any other measure we want to monitor.

The remainder of this paper is organized as follows. Section 2 explores the state of the art in categorical change detection for data streams. The proposed methodology is presented in Section 3. Section 4 presents experimental results for real world datasets and we also report an analysis of its qualitative behavior. Section 5 concludes.

2. RELATED WORK

The problem of detecting concept drift or change is an important issue in data stream mining [1]. The research community has spent a lot of effort to detect change in numerical streams [8], while dealing with categorical data has not been addressed all that much [4]. A first attempt to mining concept drift in categorical data is presented in [6]. Their proposed approach determines concept drift by measuring the difference in cluster distribution between two contiguous batches of stream data. An important drawback of this method, that limits its usability in practise, is the difficulty encountered when having to customize the settings of its many parameters. In [7], a framework was presented for detecting the change of a primary clustering structure which was indicated by the best number of clusters in categorical data streams. However, setting the decaying rates to adapt to different types of clustering structures is very difficult. Cao proposed two different rough set theory based methods to detect concept drifts in streams of categorical data [5, 4]. The former [5] defines the distance between two concepts as the difference value of the degree of membership of each object belonging to two different concepts, respectively. This distance can only detect concept drift but not the reasons leading to the change. The later strategy presented in [4], evaluates the difference of distributions between two concepts appearing in two consecutive batches. The difference is defined by the approximation accuracy of rough set theory which can also be used to measure the speed of change of concepts. If the difference is bigger than a threshold a concept drift is revealed. Both methods require only a threshold parameter but, unfortunately: i) no hints on how set this parameter are supplied and ii) the parameter threshold has no clear associated meaning. These two factors do not facilitate the analyst to use these approaches. As has been pointed out, all these methods have several issues related to parameter settings (too many parameters or no hints about the value to use). Furthermore, all these approaches assume that, for detecting a concept drift it is enough to consider only contiguous batches and no more historical information coming from the data stream.

3. METHODOLOGY

In this section we describe the general setting and we introduce our new method. Suppose that we have an infinite data stream divided in batches. Each batch S_i is associated with the arrival time denoted by the index i: $S = \{S_1, S_2, ..., S_n, ...\}$. This scenario is general enough to model real data streams. Note that, even in the case of continuous streams of data, we can employ a buffer-based procedure for collecting examples in order to build batches and then process one batch at a time. In addition, each example belonging to the data stream is defined over a set of categorical attributes. This means that each attribute A_j is defined over a discrete set of nominal values. As previously mentioned, the two important points that we address in this work are the following: (i) extracting useful information from each batch in order to summarize it and (ii) keeping trace of this information to be able to detect change. All these steps are addressed in a fully unsupervised scenario where no class information is available differently from many previous strategies that work in supervised context [1]. To deal with the first point we exploit the method proposed in [12], named *DILCA*. This method is able to extract numerical statistics from a set of categorical examples in order to summarize the underlying data distribution. To tackle the second point we monitor the statistics extracted from the batches by employing the *Chebyshev's Inequality* [1]. The combination of (i) and (ii) allows *CDCStream* to summarize categorical batches and to detect changes as they happen in the underlying data distribution, and to issue appropriate warnings. In the following we describe in detail the steps that *CDCStream* is based on.

3.1 Computing the distance model

Here we revise the *DILCA* (**D**Istance **L**earning for **C**ategorical **A**ttributes) framework that we can employ to summarize a batch of categorical data. *DILCA* was first introduced by [12] and it was mainly used for clustering categorical data. The result of this method is a set of matrices, one for each attribute. Each matrix contains the learnt distances between each pair of values of a specific attribute. Let us now consider the set $F = \{X_1, X_2, \ldots, X_m\}$ of m categorical attributes over which the batch S_i is defined. We denote by Y the target attribute, which is a specific attribute in F that constitutes the target of the method, that is, on whose values we need to compute the distances. The notation $|X_l|$ refers to the cardinality of the attribute X_l. *DILCA* allows to compute a context-based distance between any pair of values (y_i, y_j) of the target attribute Y on the basis of the similarity between the probability distributions of y_i and y_j given the context attributes, called $\mathcal{C}(Y) \subseteq F \setminus Y$. For each context attribute X_i it computes the conditional probability for both the values y_i and y_j given the values $x_k \in X_i$ and then it applies the Euclidean distance. The Euclidean distance is normalized by the total number of considered values:

$$d(y_i, y_j) = \sqrt{\frac{\sum_{X \in \mathcal{C}(Y)} \sum_{x_k \in X} (P(y_i|x_k) - P(y_j|x_k))^2}{\sum_{X \in \mathcal{C}(Y)} |X|}} \quad (1)$$

The selection of a good context is not trivial, particularly when data are high-dimensional. To select a relevant and non redundant set of features w.r.t. a given one, [12] propose to adopt *FCBF*, a feature-selection approach originally presented by [15]. We let the reader refer to the cited paper for more details about the context selection step. At the end of the process, *DILCA* returns a distance model $\mathcal{M} = \{M_{X_l} \mid l = 1, \ldots, m\}$, where each M_{X_l} is the matrix containing pairwise distances between the values of attribute X_l, computed using Eq. 1. The distances contained in the matrix M_{X_l} resume the underlying distribution of attribute X_l. Each generated matrix M_{X_l} has some interesting characteristics: it is symmetrical, the diagonal is filled by 0s

(logically the distance between a value and itself is 0) and each value is bounded between 0 and 1. The set $\mathcal{M}$ of matrices is aggregated into one single measure by the following formula, which only considers the upper triangular part of each matrix, discarding the all-zeroes diagonal (matrices are symmetric and their diagonals are filled by 0s):

$$extractSummary(M) = \frac{\sum_{M_l \in M} \frac{2 \times \sqrt{\sum_{i=0}^{|X_l|} \sum_{j=i+1}^{|X_l|} M_{X_l}(i,j)^2}}{|X_l| * (|X_l|-1)}}{|F|} \quad (2)$$

This formula extracts a single statistic starting from the matrices produced by the *DILCA* method over a batch of examples. For each matrix M_l the squares of all values is summed and then rescaled by using the root. This value is then divided by the number of values that are summed ($\frac{|X_l|*(|X_l|-1)}{2}$). This operation normalizes the coefficient extracted from each matrix forcing them back into the range between 0 and 1.

The outer loop sums the coefficients of all the matrices and then the normalization step allows to scale back the value into the range between 0 and 1 again. The result of this formula numerically represents a summary of the whole batch taking into account both the correlation among the attributes and the attributes' distributions themselves.

3.2 Change Detection through the Chebyshev's Inequality

Statistical tests are often used in data stream mining in order to capture changes in the distribution [1]. Most of the literature focuses its attention on the supervised scenario where the goal is to detect changes as a decrease in classification accuracy. When that happens, it is time to re-adapt or start to re-learn a model. To manage this phenomenon, researchers usually exploit one-sided tests. These tests are usually chosen since changes (in a supervised scenario) are associated with a decrease only of classification accuracy. If a classifier increases its accuracy, we have no reason to modify it. Unfortunately, in our context, we work in a totally unsupervised setting: we want to detect general variation on both sides of the distribution, because the monitored variable is not the accuracy but an estimation of the data distribution. To deal with this issue, in order to understand if a batch of data deviates from the actual distribution or not, we employ techniques from Statistical process control [8], that are based on Chebyshev's Inequality [1]. This statistical test does not assume any data distribution. It is a two tailed test and it considers both sides of the distribution to understand if a change is appearing.

LEMMA 1 (CHEBYSHEV'S INEQUALITY). *Let X be a random variable with expectation μ_X and standard deviation σ_X. Then for any $k \in R^+$,*

$$Pr(|X - \mu_X| \geq k\sigma_X) \leq \frac{1}{k^2}$$

In particular, we consider the statistic computed over each batch S_i as a value coming from a randomly distributed variable. In this way, without loss of generality, we can use *Chebyshev's Inequality* to monitor if change is happening, or not, in the data stream distribution.

As we can observe, to use this inequality, we need to compute the average μ_X and the standard deviation σ_X. As we assume that a data stream evolves and that it is infinite, we can never retain all values. To tackle this important issue, we need to design an intelligent method to understand how many previous batches we need to use in order to monitor the actual one. Our solution is sketched in Algorithm 1. The procedure takes as input parameter the stream of data S and the values k_c and k_w to compute *Chebyshev's Inequality* in order to detect, respectively, changes and warnings. While a change can be associated with a drift in the data stream, a warning is a small variation or fluctuation that will not necessarily result in a shift of the data distribution. In addition to actual change, noise may also cause such warnings.

We can observe that the first step of the algorithm is devoted to initialize the list (L) and the standard deviation used as default in the case where we cannot compute it ($\sigma_{default}$). The list L contains the summary values up until the previous batch: this means that if we are analyzing batch S_i, L contains statistics until batch S_{i-1}. As we enter the main loop, given the batch S_i we extract the actual summary statistic using Formula 2. Once we obtain the statistic we check the size of L. We did not check the case in which L is empty because it can only be empty at the begin. After that it will always contain at least one value. When L contains more than a value we can compute both an average and a standard deviation. In the case where L contains only one value, and we have already computed the maximum (σ_{MAX}) and the minimum (σ_{MAX}) standard deviation seen until now, the actual summary value is used as the average and the standard deviation is set to $(\sigma_{MAX} + \sigma_{MIN})/2$.

If the value of σ_{MAX} and σ_{MIN} are not defined, than we are in the first or second batch of the stream, and we have not yet collected enough evidence to compute these statistics, as we need to see at least two batches before we can compute them. In this case we need to continue to accumulate evidence before being ready to detect changes or warnings. Line 22 checks if σ is not null. If so, we employ the statistical process control technique based on *Chebyshev's Inequality* to evaluate if a change or a warning happens. In particular, *Chebyshev's Inequality* is used with two different thresholds k_c to detect a change and k_w to detect a warning. Logically, as a warning is a variation in the distribution less important than a change (which is more critical by definition) k_c is greater that k_w and for this reason, the inequality is first used to check if a change happens and successively, if that first test is negative, the second test determines the presence of a warning. At the end of the procedure the algorithm adds the new summary value x to the list L.

If a change happens the values (i.e. the statistics/summary values until batch S_i) in the list L are removed and we restart to fill L with the one extracted from batch S_i. Practically, if a change is detected, the actual batch S_i is statistically different from the previous sequence of batches and, for this reason, we do not need to keep the history before S_i anymore. In the case of a warning the algorithm only reports it to the user, without clearing L.

The final algorithm is a method able to automatically segment the sequence of summary statistic generated to summarize the batches of the stream.

4. EXPERIMENTS

In this section we evaluate the performance of our approach *CDCStream* under different aspects. We compare the behavior of our algorithm with the method proposed in [4]

Algorithm 1 $ChangeDetectionProcess(S, k_c, k_w)$

Require: S: stream of instances
Require: k_c: stat. parameter to detect a change
Require: k_w: stat. parameter to detect a warning ($k_w < k_c$)
1: $\sigma_{MAX} = \sigma_{MIN}$ = nil
2: $L = \emptyset$
3: **while** hasMoreInstances(S) **do**
4: S_i = extractNextBatch(S)
5: //the summary statistic is extracted as explained in Sec. 3.1
6: x = extractSummaryStatistic(S_i)
7: $\sigma = \mu$ = nil
8: **if** ($L.size > 1$) **then**
9: μ = avg(L)
10: σ = stdev(L)
11: **if** (σ_{MAX} == nil or $\sigma > \sigma_{MAX}$) **then**
12: $\sigma_{MAX} = \sigma$
13: **end if**
14: **if** (σ_{MIN} == nil or $\sigma < \sigma_{MIN}$) **then**
15: $\sigma_{MIN} = \sigma$
16: **end if**
17: **end if**
18: **if** ($L.size$ == 1 and $\sigma_{MAX} \neq$ nil and $\sigma_{MIN} \neq$ nil) **then**
19: μ = avg(L)
20: $\sigma = \frac{\sigma_{MAX}+\sigma_{MIN}}{2}$
21: **end if**
22: **if** ($\sigma \neq nil$ and $|x - \mu| \geq k_c\sigma$) **then**
23: // *CHANGE HAPPENS*
24: $L = \emptyset$
25: **else if** ($\sigma \neq nil$ and $|x - \mu| \geq k_w\sigma$) **then**
26: // *WARNING HAPPENS*
27: **end if**
28: $L.add(x)$
29: **end while**

that we name $RSCD$. $RSCD$ is the most recent approach to detect changes in categorical data streams. It obtains good performance compared to previous methods.

To objectively measure the quality of the two methods, we compare them in a learning scenario in which a classifier is learnt using a warning model as introduced in [8]. In particular, at the beginning a classifier is learnt and tested over the stream of data. Every time a warning is detected, a background classifier is created and trained in parallel with the main classifier. When a change is detected, the main classifier is replaced by the background one (and a new background classifier is also started). As a base learner for both *CDCStream* and $RCSD$ we use the Naive Bayes classifier [14]. To evaluate the different methods we employ the final accuracy of the learnt model. To assess the accuracy we use the prequential evaluation: each time an instance arrives, we first test the algorithm on it, and then we subsequently use the example as training input for the classifier.

$RCSD$ does not implicitly manage warnings, for this reason we adapt it to manage both changes and warnings. We consider that a change happens when $RCSD$ returns values bigger than 0.01 (the same threshold is used in the original work [4]) while we highlight warnings when $RCSD$ returns values bigger than 0.005 but lower than 0.01. As these values represent a threshold over a distance, it is reasonable that the change threshold is larger than the warning one. *CDCStream* is designed to manage both warnings and changes. For our approach we set the value of k_c equals to 3 and k_w equals to 2. We evaluate the two approaches using different batch sizes. In particular we employ the following values: $\{50, 100, 500, 1\,000\}$.

In order to deeply understand the behaviour of our strategy, we also employ as another competitor the supervised method presented in [8], that we name $SDriftClassif$.

Dataset	# of Inst.	# of Feat.	# of Cl.
Airlines	539 383	7	2
Electricity	45 312	8	2
Forest	581 012	54	7
KDD99	148 517	41	2

Table 1: Dataset characteristics

It also implements a warning/change mechanism monitoring the accuracy of the classifier. All our experiments are performed using the MOA data stream software suite [3]. MOA is an open source software framework in Java designed specifically for data stream mining.

4.1 Datasets

To evaluate all the algorithms we use four real world datasets: *Electricity*, *Forest*, *Airlines*, *KDD99*.

Electricity data [10] is a popular benchmark in stream classification. The task is to predict the rise or fall of electricity prices (demand) in New South Wales (Australia), given recent consumption and prices in the same and neighboring regions. The task proposed in the *Forest* dataset [2] is to predict forest cover types of vegetation from cartographic variables. Inspired by [13] we constructed an *Airlines* dataset using the raw data from US flight control. The task is to predict whether a given flight will be delayed, given the information of the scheduled departure. The last dataset we use is *KDD99*. One of the big problems with this dataset is the big amount of redundancy among instances. To solve this problem we use the cleaned version named NSL-KDD[1]. To build the final dataset we join both training and test data. A summary of the characteristics of these datasets is reported in Table 1. We observe that this collection of datasets contains both binary and multi-class problems, datasets with different size (varying between 42k to 580k instances) and different dimensionality (from 7 to 54 attributes).

For analysis purposes, we also introduce one more dataset, named *Forest Sorted*, in which the instances of the *Forest* dataset are reordered using the attribute *elevation*. Due to the nature of the underlying problem, sorting the instances by the *elevation* attribute induces a natural gradual drift on the class distribution, because at higher elevation some types of vegetation disappear while other types of vegetation appear smoothly.

As general pre-processing for all numerical attributes we perform unsupervised equi-width discretization using a number of bins equal to 5.

4.2 Analysis of Accuracy

In table 2 we report the accuracy performance of the different methods using the warning/change framework as explained at the beginning of the experimental section. For each dataset, we highlight in bold the best result obtained for it. As a first point, we can observe that the supervised approach did not outperform the unsupervised ones. This is a remarkable result because $SDriftClassif$ is the only method that exploits the accuracy to make a decision in order to detect a change and to re-learn the model. The only case in which $SDriftClassif$ obtains slightly better results is over *KDD99*, but the final performances are comparable as the difference is very low (around one point of accuracy).

[1]http://nsl.cs.unb.ca/NSL-KDD/

Dataset		b=50		b=100		b=500		b=1 000	
	SDriftClassif.	*CDCStream*	*RSCD*	*CDCStream*	*RSCD*	*CDCStream*	*RSCD*	*CDCStream*	*RSCD*
Electricity	70.58%	**73.84%**	73.77%	71.23%	72.19%	68.18%	64.45%	66.01%	63.40%
KDD	**91.38%**	90.81%	90.65%	90.04%	90.06%	90.59%	90.06%	90.87%	90.06%
Forest	80.27%	**82.99%**	82.25%	81.18%	80.39%	74.46%	64.08%	80.05%	64.08%
Forest Sort.	67.44%	70.57%	68.35%	70.51%	68.35%	**72.10%**	68.35%	69.33%	68.35%
Airlines	65.25%	66.43%	62.71%	64.50%	64.41%	67.16%	66.73%	**67.66%**	67.64%

Table 2: Accuracy of models learnt by different strategies using warning/change framework under different batch size

In all the other cases, we can observe that *CDCStream* outperforms the other competitors (both supervised and unsupervised).

Considering the comparison between *CDCStream* and $SDriftClassif$ we can note that, most of the time, our approach obtains interesting performances. This is especially true for *Forest Sorted* in which for all values of batch sizes it outperforms $SDriftClassif$. For both *Electricity* and *Forest*, *CDCStream* obtains improvements using smaller batch size ($b = 50$ and $b = 100$). This similar behavior is shared by the classifier learnt using the $RCSD$ approach. This is coherent with the results presented in [11]. The authors show that for these two datasets smaller batches outperform bigger ones as they note that small batches forces the learner to adapt faster to possible changes. This is particularly true for *Electricity*, where multiple levels of periodicity are present, over 24 hours, over 7 days, and over 4 seasons. Drift can be visually presented by plotting class conditional distributions over time. Also for *Airlines* we can observe that our approach obtains results higher or comparable to the one achieved by $SDriftClassif$. This time the better performances are reached considering larger batch size ($b = 500$ and $b = 1\,000$), gaining in the best case more than two points of accuracy.

Analysing the performance of *CDCStream*, we can note that most of the time the classifier learnt by our approach obtains better performance. Also in this case, considering *Forest Sorted*, for all batch values *CDCStream* outperforms $RCSD$. Another interesting point is that *CDCStream* obtains interesting stability results w.r.t. $RCSD$. This behavior is very clear considering the *Forest* dataset. We can see that for *CDCStream* the accuracy varies between 82.99% ($b = 50$) and 74.46% ($b = 500$) while for the rough set based method the accuracy vary from 82.25% ($b = 50$) to a minimum of 64.08% ($b = 500$ or $b = 1\,000$), which is a variation of 18 points of accuracy. A similar, but less drastic behavior can be observed for the *Electricity* dataset, where larger batch sizes impact negatively on $RCSD$ while *CDCStream* remains more stable.

Generally, we can note that monitoring changes in data distribution ($P(X)$) with *CDCStream* allows to learn a good model. In supervised approaches, as $SDriftClassif$, the detector reacts to changes in the posterior probability ($P(Y|X)$). Most of the time supervised methods assume that only $P(Y|X)$ varies and it is the only useful quantity to monitor in order to make a decision [9]. An interesting future work will be to combine both sources of information (detected changes in $P(X)$ and in $P(Y|X)$) in order to learn better stream classifiers.

To summarize, we can state that the classifier learnt using *CDCStream*, under the warning/change detection framework, obtains interesting accuracy results and also generally improves the classifier quality compared to a previous supervised approach. We can also underline that, considering the benchmark used in the experiments, *CDCStream* is robust considering the batch size. This parameter did not have a strong influence on the final performance.

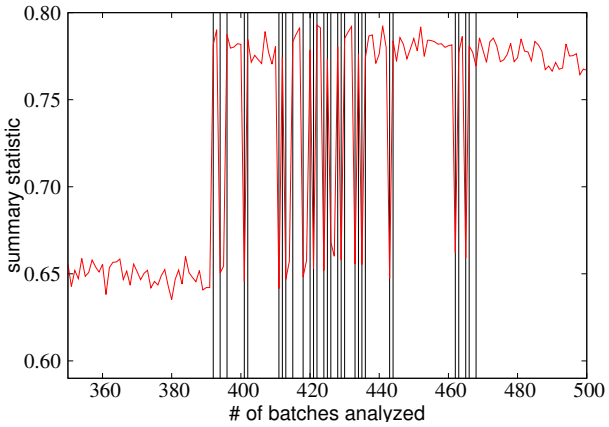

Figure 1: A sample of changes detected by *CDCStream* on the *Forest Sort.* dataset with batch size equals to 1 000 between the batches 350 and 500

4.3 Analysis of Change Detection Behavior

We report the results concerning the behaviors of *CDCStream* and $RCSD$ in Table 3. In this experiment we measure the percentage of changes detected by both methods, for each dataset considering different batch sizes. The percentage of changes is the number of changes detected divided by the possible number of changes that can happen in the stream. If a dataset is divided in N batches, the number of possible changes is equal to $N - 1$ because we can have, at most, one change for each batch except the first one as we cannot start to detect changes before the second batch. The first aspect we point out is that the two methods have different behaviors. This means that when one approach increases the number of changes detected, the other one does not exhibit the same trend. For instance, if we analyze the results on *Electricity*, when we use a batch size equals to 50 the number of changes are similar but, when the batch size increases, they show different trends. In particular, for this dataset, we can observe that bigger batch size ($b = 500$ and $b = 1\,000$) drastically impacts the percentage of changes discovered by $RCSD$. This is also confirmed by the decrease in accuracy for this strategy over large batch sizes. If we continue to analyze the results, we can note that for *Forest Sort.* $RCSD$ has problems to detect the smooth changes that are intrinsically present in this dataset. This is underlined by the fact that for $b = 50$ and $b = 100$ it does not detect changes (a percentage equals to 0%) and also for

Dataset	b=50		b=100		b=500		b=1 000	
	CDCStream	*RSCD*	*CDCStream*	*RSCD*	*CDCStream*	*RSCD*	*CDCStream*	*RSCD*
Electricity	96.57%	92.93%	64.67%	100%	55.55%	6.66%	44.44%	8.88%
KDD	7.97%	1.34%	1.01%	0.06%	85.52%	0.33%	75.67%	0%
Forest	96.3%	77.63%	34.49%	71.15%	0%	0.17%	18.24%	0.34%
Forest Sort.	12.65%	0%	13.87%	0%	96.21%	0.08%	8.26%	0.17%
Airlines	14.68%	100%	94.12%	100%	2.78%	100%	58.07%	44.52%

Table 3: Change behavior: number of changes triggered over the number of possible changes (Number of batches - 1)

larger values of batch size it does not react as much (a percentage lower than 0.2%). This phenomenon explains why the model learnt using the *RCSD* approach always obtains the same accuracy over all the values of batch size. The fact that the rough sets based approach does not underline changes, in this case, impacts the performance of the final classifier in a bad way. A totally different behavior is evident for *Airlines*. For this dataset *RCSD* (for batch size of 50, 100, 500) detects as many changes as theoretically possible. This means that, each time, the model previously learnt is discarded and replaced with a new one while *CDCStream* adopts a different strategy, and it adapts itself to the value of batch size in order to discover changes. As a practical consequence, the strategy adopted by *CDCStream* outperforms the one supplied by *RCSD* to learn the classification model for the stream data. In general we can say that, most of the times, *RCSD* detects smaller percentages of changes than the ones detected by *CDCStream*. When the opposite happens, *RCSD* always saturates the percentage of changes, reaching 100%. In Figure 1 we report the behavior of *CDCStream* on the *Forest Sort.* dataset considering a batch size of 1 000 instances. For lack of space we report only this example as a representative. We observe similar behavior over all the other datasets. On the Y axis we plot the value of the summary statistic (Formula 2) monitored by *CDCStream*. The X axis represents the # of batches analyzed during the stream. Vertical lines are used to highlight when *CDCStream* detects a change. In order to easily visualize some interesting behavior we supply the trend between the first 350 and 500 batches. We can observe that our metod detects the important change observed around the batch 390. After that we can note that *CDCStream* continues to detect reasonable changes of the monitored statistic segmenting the time series in a reasonable way. We underlined that the segmentation algorithm implemented in *CDCStream* is general and can be employed to monitor any numerical time series in order to detect variations (increase or decrease) in the data distribution.

5. CONCLUSION

Detecting changes in a stream of categorical data is not straightforward. In this paper we presented a new algorithm in order to deal with this task. We adapt a previous technique in order to extract summaries from a batch of categorical instances. After that, we develop a new warning/change detection approach based on Chebyshev's Inequality that automatically segments the stream of statistics produced by the summarization step. A strong point of our approach is its ability to exploit historical information for decision making in a dynamic fashion, adjusting the history size automatically. The only parameters needed by *CDCStream* are related to Chebyshev's Bound and they have therefore clear semantics. The experiments fully validate the quality of *CDCStream* showing interesting results obtained by our strategy. As future work we plan to apply our method in categorical data stream clustering to update the model when changes are detected.

6. REFERENCES

[1] C. C. Aggarwal, editor. *Data Streams - Models and Algorithms*. Advances in Database Systems. Springer, 2007.

[2] A. Asuncion and D. Newman. UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences, 2007.

[3] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis. *J. Mach. Learn. Res.*, 11(May):1601–1604, 2010.

[4] F. Cao and J. Z. Huang. A concept-drifting detection algorithm for categorical evolving data. In *PAKDD (2)*, 2013.

[5] F. Cao, J. Liang, L. Bai, X. Zhao, and C. Dang. A framework for clustering categorical time-evolving data. *IEEE T. Fuzzy Syst.*, 18(5):872–882, 2010.

[6] H. Chen, M. Chen, and S. Lin. Catching the trend: A framework for clustering concept-drifting categorical data. *IEEE TKDE*, 21(5):652–665, 2009.

[7] K. Chen and L. Liu. He-tree: a framework for detecting changes in clustering structure for categorical data streams. *VLDB J.*, 18(6):1241–1260, 2009.

[8] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with drift detection. In *SBIA*, 2004.

[9] J. Gao, W. Fan, J. Han, and P. S. Yu. A general framework for mining concept-drifting data streams with skewed distributions. In *SDM*, 2007.

[10] M. Harries, C. Sammut, and K. Horn. Extracting hidden context. *Mach. Learn.*, 32(2):101–126, 1998.

[11] D. Ienco, A. Bifet, I. Zliobaite, and B. Pfahringer. Clustering based active learning for evolving data streams. In *Discovery Science*, 2013.

[12] D. Ienco, R. G. Pensa, and R. Meo. From context to distance: Learning dissimilarity for categorical data clustering. *ACM TKDD*, 6(1):1:1–1:25, mar 2012.

[13] E. Ikonomovska, J. Gama, and S. Dzeroski. Learning model trees from evolving data streams. *Data Mining and Know. Disc.*, 23(1):128–168, 2010.

[14] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Data Management Systems. Morgan Kaufmann, 2005.

[15] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *ICML*, 2003.