Context-Based Abrupt Change Detection and Adaptation for Categorical Data Streams

Sarah D'Ettorre[1](✉), Herna L. Viktor[1], and Eric Paquet[1,2]

[1] School of Electrical Engineering and Computer Science,
University of Ottawa, Ottawa K1N 6N5, Canada
{sdett026,hviktor}@uottawa.ca, eric.paquet@nrc-cnrc.gc.ca

[2] National Research Council of Canada, Ottawa, ON K1A 0R6, Canada

Abstract. The identification of changes in data distributions associated with data streams is critical in understanding the mechanics of data generating processes and ensuring that data models remain representative through time. To this end, concept drift detection methods often utilize statistical techniques that take numerical data as input. However, many applications produce data streams containing categorical attributes, where numerical statistical methods are not applicable. In this setting, common solutions use error monitoring, assuming that fluctuations in the error measures of a learning system correspond to concept drift. Context-based concept drift detection techniques for categorical streams, which observe changes in the actual data distribution, have received limited attention. Such context-based change detection is arguably more informative as it is data-driven and directly applicable in an unsupervised setting. This paper introduces a novel context-based algorithm for categorical data, namely **FG-CDCStream**. In this unsupervised method, multiple drift detection tracks are maintained and their votes are combined in order to determine whether a real change has occurred. In this way, change detections are rapid and accurate, while the number of false alarms remains low. Our experimental evaluation against synthetic data streams shows that **FG-CDCStream** outperforms the state-of-the art. Our analysis further indicates that **FG-CDCStream** produces highly accurate and representative post-change models.

Keywords: Data streams · Categorical data · Concept drift · Context-based change detection · Unsupervised learning · Ensembles · Online learning

1 Introduction

Data streams, that are characterized by a continuous flow of high speed data, require learning methods that incrementally update models as new data become available. Further, such algorithms should be able to adapt appropriately as underlying concepts in the data evolve over time. Explicit detection of this evolution, known as concept drift, is beneficial in ensuring that models remain accurate and provide insights into the mechanics of the data generating process [6]. Thus, concept drift detection has been of continuous interest to machine learning researchers.

The majority of concept drift detection algorithms utilize statistical methods requiring numerical input [5]. However, real world data attributes are often categorical [9]. For example, point-of-sales streams include categorical attributes such as colour {red, green, blue} or size {small, medium, large, x-large}. Environmental data attributes might contain categorical attributes like predators {eagle, owl, fox} or land cover {desert, forest, tundra}. This prevalence of categorical data poses a challenge for change detection researchers. Currently, the majority of research on change detection in categorical data streams utilize error changes in the learning system as an indicator of concept drift [9]. While these techniques have proven to be reasonably successful, it remains that fluctuations in error measures cannot be definitively attributed to concept drift alone. Relatively few studies in the literature have examined context-based change detection in categorical data streams [9]. In this case, concept drift is detected when changes in the actual data distribution are observed, providing more precise information about particular changes. This opens the door to unsupervised change detection [9], a non-trivial task [4]. Since class information is not always available, facilitating unsupervised change detection broadens the spectrum of categorical data which may be analyzed.

To this end, this paper focuses on improving the quality of knowledge extraction from evolving streams of categorical data through the use of context-based change detection and adaptation strategies. This paper introduced the **FG-CDCStream** technique, which extends the **CDCStream** algorithm [9], in order to rapidly detect abrupt changes, using a fine-grained drift detection technique. The **FG-CDCStream** method employs a voting-based algorithm in order to track the evolution of the data as the stream evolves. Adaptation is improved by ensuring that a post-change classifier is trained on a reduced batch of highly relevant data. This ensures that the evolving classification models are more representative of the post-change concept. Our experimental evaluation confirms that our algorithm is highly suitable for abrupt change detection.

This paper is organized as follows. Section 2 introduces background work. In Sect. 3, we detail the **FG-CDCStream** algorithm. Section 4 discusses our experimental evaluation and results. Finally, Sect. 5 presents our conclusion and highlights future work.

2 Background

This section discusses related works in terms of measuring the similarity of categorical data and context-based drift detection.

2.1 DILCA Context-Based Similarity Measure

Categorical variables are abundant in real-world data and this fact has lead to a large and diverse collection of proposed distance measures spanning various fields of study, many of which arising in the context of categorical data clustering. Similarity measures can be context-free or context-sensitive, supervised or unsupervised. Context-free measures do not depend on relationships between instances while context-sensitive measures do. Based on [4,5], distance measures for categorical values may be classified into six groups that are not necessarily mutually exclusive: simple matching approaches, frequency-based approaches, learning approaches, co-occurrence approaches, probabilistic approaches, information theoretic approaches and rough set theory approaches.

DILCA is a recent state-of-the-art similarity measure that is purely context-based, that makes no assumptions about data distribution and that does not depend on heuristics to determine inter-attribute relationships [8]. These properties make **DILCA** attractive as it minimizes bias and can be applied to a great range of data sets with a wide range of characteristics. The **DILCA** similarity measure is computed in two steps, namely context selection and distance computation. Informative context selection is non-trivial, especially for data sets with many attributes. Consider the following set of m categorical attributes: $F = \{X_1, X_2, ..., X_m\}$. The context of the target attribute Y is defined as a subset of the remaining attributes: $context(Y) \subseteq F/Y$. **DILCA** uses the Symmetric Uncertainty (SU) feature selection method [10] to select a relevant, non-redundant set of attributes which are correlated to the target concept in the context.

$$SU(Y, X) = 2\frac{IG(Y; X)}{H(Y) + H(X)} \tag{1}$$

DILCA selects a set of attributes with high SU with respect to the target Y. A strength of the SU measure, as defined in Eq. 1, is that it is not biased towards features of greater cardinality and is normalized on $[0, 1]$. Once the context has been extracted, the distances between attribute values of the target attribute are computed using Eq. 2:

$$d(y_i, y_j) = \sqrt{\frac{\Sigma_{X \in context(Y)} \Sigma_{x_k \in X} (P(y_i|x_k) - P(y_j|x_k))^2}{\Sigma_{X \in context(Y)} |X|}} \tag{2}$$

For each value of each context attribute, the Euclidean distance between the conditional probabilities for both values of Y is computed and summed. This value is then normalized by the total number of values in X. The pairwise distances computation between each of the values of Y results in a symmetric and hollow (where diagonal entries are all equal to 0) matrix $M_i = |Y|$ x $|Y|$.

2.2 CDCStream Categorical Drift Detector

The **CDCStream** algorithm, as created by [9], utilizes the above-mentioned **DILCA** method [9] in order to detect drift in categorical streams, as follows.

Consider an infinite data stream S where each attribute X is categorical. A buffer is used to segment the stream into batches: $S = \{S_1, S_2, ..., S_n, ...\}$. (Note that, if the class information is present in the stream, it is removed during segmentation, thus creating an unsupervised change detection context.) The set of distance matrices $M = \{M_1, M_2, ..., M_s\}$ produced by **DILCA** is aggregated to numerically summarize the data distribution of each batch in a single statistic using Eq. 3. The resulting statistic, in $[0, 1]$, represents both intra- and inter-attribute distributions of a batch.

$$extractSummary(M) = \frac{\Sigma_{M_l \in M} \frac{2 * \sqrt{\Sigma_{i=0}^{|X_l|} \Sigma_{j=i+1}^{|X_l|} M_{X_l}(i,j)^2}}{|X_l| * (|X_l| - 1)}}{|F|} \tag{3}$$

Data are analyzed in batches, by using a dynamic sliding window method. The historical window L consists of all batch summaries, except the most recent batch which constitutes the current window. The dynamic window grows and shrinks appropriately based on the change status, by forgetting the historical window and/or absorbing the most recent batch. In periods of stability, the historical window continues to grow summary by summary. When a change is detected, abrupt forgetting is employed and the historical window is dropped and replaced with the current window.

As noted in [9], a two-tailed statistical test, which does not assume a specific data distribution is required for context-based, unsupervised change detection. To this end, Chebychev's inequality was adopted. Formally, Chebychev's Inequality states that if X is a random variable with mean μ_X and standard deviation σ_X, for any positive real number k:

$$Pr(|X - \mu_X| \geq k\sigma_X) \leq \frac{1}{k^2} \tag{4}$$

In this equation, $\frac{1}{k^2}$ is the maximum number of the values that may be beyond k standard deviations from the mean. **CDCStream** utilizes this property in order to warn for, and subsequently detect, changes in the distributions. Values of k representing warning and change thresholds, are denoted by k_w and k_c, respectively. These values were empirically set to $k_w = 2$ and $k_c = 3$. That is, in order for a warning to be flagged, at least 75% of the values in the distribution must be within two standard deviations of the mean. For a change to be flagged, at least 88.89 % of the values must be within three standard deviations of the mean. The **CDCStream** adaptation method employs global model replacement. When a warning is detected (k_w), a new background classifier is created and updated alongside the current working model. Once a change is detected (k_c), the current model is entirely replaced with the background model.

Limitations of CDCStream for Abrupt Drift Detection. A recent study showed that **CDCStream** performs competitively in terms of accuracy and adaptability, when compared to two state-of-the art algorithms with similar goals but different structures [9]. In this prior study, the main focus was on detecting gradual drifts. **CDCStream** does, however, have a number of limitations, notably when aiming to address abrupt drift. Firstly, Chebychev's Inequality is conservative, leading to high detection delay, due to the fact that the grain of the bound must be quite coarse. Secondly, aggregation into a single summary statistic may have a diluting effect, depending on factors such as change magnitude, duration and location, which may result in increased missed detections and detection delays. In an abrupt drift setting, a main goal is to offer a fast response to change. Lastly, temporal bounds on the post-change replacement classifier training data are unnecessarily broad, potentially leading to warnings caused by noise or minor fluctuations. This broadness could severely effect any post-change classifications. Collectively, these limitations have the greatest effect on the detection of abrupt concept drifts. An increased detection delay due to the coarseness of Chebychev's inequality or aggregation dilution would be most detrimental in the case of abrupt drift. Since the distributions change so rapidly, predictions based on the previous distribution would be quite erroneous. This is more so than in the gradual change case, where the stream still contains some instances of the previous distribution. Aggregation dilution would also be more likely to miss changes altogether if the transition period was shorter. Finally, introducing a new classifier that remains partially representative of the previous distribution, is unlikely to be effective in classifying the data of an abrupt change. In the next section, we present our **FG-CDCStream** algorithm that extends **CDCStream** to address the abrupt drift scenario.

3 FG-CDCStream Algorithm

The aims of the **FG-CDCStream** approach, as depicted in Algorithm 1, are to improve detection delay and reduce missed detections of the batch-based scenario and its associated aggregation. Ergo, **FG-CDCStream** overlays a series of batches, or *tracks*, each shifted by one instance from the previous track in order to simulate an instance by instance analysis. A detailed technical explanation follows.

Our **FG-CDCStream** technique uses a dynamic list L of i contiguous batches S of fixed size n to detect a change between the current batch and the previous batches remaining in the sliding window. To solve the grain size problem, we employ overlapped dynamic lists deemed tracks. More formally, **FG-CDCStream** uses a series of n tracks, each overlapping the previous track's most recent $n-1$ instances. This allows a change test to be performed as each instance is received, while incorporating the use of batches. Note that there is a single initial delay of $2n$ before the first two batches may be compared.

Figure 1 displays the batch and track construction process as instances arrive. For simplicity, this example shows only the structure of batches and tracks and does not include responses to concept drift. Let us assume a batch size of three instances (a batch size much too small in reality, but sufficient to understand track construction). Until the first three instances are collected, no batches exist. Once the third instance arrives, the first batch, represented by the purple rectangle, is complete. From this point on, a batch is completed upon the arrival of a new instance. This is demonstrated with the creation of the blue and the green batches. Upon the arrival of the sixth instance ($2n$), a second batch is added to

Algorithm 1. FG-CDCStream

```
Input: S: stream of instances, W: window size
 1: Initialize $trackPointer = 0$, $k = 0$, $k_w = 2$, $k_c = 3$, $votes = 0$, $votes_w = 1$,
    $votes_c = 15$, $inst = null$
 2: for $count = 0 \rightarrow W$ do
 3:     $tracks.add$(new change detector)
 4: while $hasMoreInstances(S)$ do
 5:     $inst = S.nextInstance$
 6:     $buffer.add$(new batch container)
 7:     for $count = 0 \rightarrow buffer.size$ do
 8:         $buffer.get(count).add(inst)$
 9:     if $buffer.size == W$ then
10:         $k = tracks.get(trackPointer).getKValue(buffer.get(0))$
11:         if $k \geq k_c$ then
12:             $votes++$
13:             if $votes == votes_w$ then
14:                 $warningPeriod = true$; $initiateBackgroundClassifier(buffer.get(0))$
15:             if $votes == votes_c$ then
16:                 $changePeriod = true$; $warningPeriod = false$
17:                 replace classifier with background classifier; nullify background classifier
18:         else if $k < k_w$ then
19:             $votes = 0$
20:             if $warningPeriod$ then
21:                 $warningPeriod = false$; nullify background classifier
22:             if $changePeriod$ then
23:                 $changePeriod = false$
24: $updateClassifier(inst)$; $updateBackgroundClassifier(inst)$
25: if $buffer.size == W$ then
26:     $buffer.remove(0)$ ;
27:     update track pointer
```

the original purple track. Note that the track with the newest batch is shown at the front and the least recently updated track in the rear. This process continues until the end of the stream (or to infinity).

It should be noted that, in reality, instances are stored explicitly only until a batch is complete. A buffer with a container for each track stores instances until n have been collected. When a batch is complete, it is input to the corresponding track's change detector object, which maintains summary statistics, as described in Eq. 3. That track's buffer is then cleared and the main algorithm begins building its next batch upon the arrival of the next instance. The appearance of the next instance fills a batch belonging to the next track and the process continues.

The algorithm requires a data stream and a user-defined window size parameter. An integer variable *trackPointer* keeps an account of which track the current batch (the batch completed by the current instance) belongs to. The k variable refers to the value calculated by Chebychev's inequality, which is initially zero. The *votes* variable stores the accumulated votes of the tracks seen so

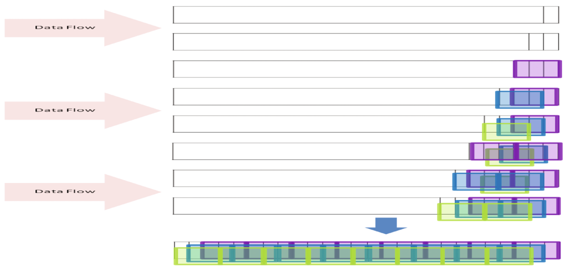

Fig. 1. Track and batch building visualization

far. As each instance arrives, a buffer, which is a list of instance containers (or batches), adds a new batch object to the list. A copy of the current instance is then added to each of the existing batches. Note that upon the arrival of the first instance, only one batch container exists. When the second instance arrives, a new batch is added to the buffer and that instance is added to both batches. At this point, the first batch contains the first and second instances, and the second batch contains only the second instance. This produces contiguous batches that contain data that are shifted by one instance. This process continues until the buffer contains n batches. At this point the first batch is complete, i.e. it contains n instances. The complete batch is processed, as described below. After the batch is processed and summarized, it is removed from the buffer.

To process a completed batch, the batch is sent to the change detector associated with the current track. The change detector returns the value of k for this batch, as calculated using Chebychev's Inequality. Recall that, in **CDCStream**, if this value is equal to two (k_w), a warning occurs. Similarly, a value of three (k_c) indicates that a change is detected by this track. Otherwise, a value of zero is returned.

The **FG-CDCStream** algorithm differs from **CDCStream** in two aspects. Firstly, it does not immediately flag a warning when a track encounters $k = k_w = 2$, but only maintains this statistic. Secondly, a value of $k = k_c = 3$ initiates a warning period, rather than reporting a change, if this is the first change detection in a series. This launches the creation of the background classifier which is built from the current batch. Subsequent reports of k_c in this warning period increment the *votes* count. Reported values of k_w are permitted within a warning period, but do not terminate it nor increment the vote count. If a value of less than k_w is reported by a track during the warning period, the warning period is terminated, the *votes* count is reset to zero and the background classifier is removed. This initiates a static period, until the next warning occurs.

If at least $votes_c$ tracks confirm a value of k_c, the system acknowledges the change. (It follows that the value of the $votes_c$ parameter is domain-dependent and determined through experimentation.) A confirmed change triggers adaptation by replacing the current classifier with the background classifier. The change period remains, whereby no new warnings or changes may occur, until a value of $k < 2$ is reported, initiating the next static period. A fall in the value of k signifies that the current change is fully integrated into the system, i.e. the change detectors have forgotten the past distribution and the current model represents the current distribution.

Whether or not adaptation occurs, a forgetting mechanism is applied to any change detector that produces a value of k_c. If change is not confirmed, the original change detection was likely incorrect and thus forgetting the corresponding information for that specific track is assumed to be reasonable. This effectively resets change detectors that are not performing well and permits outlier information to be discarded.

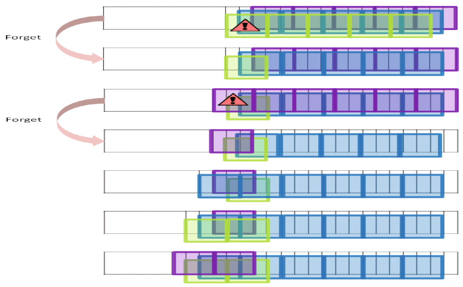

Fig. 2. Forgetting mechanism example

Figure 2 illustrates the system's forgetting mechanics. Firstly, the green track's most recent batch detects a change (represented by the exclamation mark in the red triangle). It then forgets all of its past batches retaining only the current one. The next batch to arrive, belonging to the purple track, also detects the change and forgets its past batches. The next batch, belonging to the blue track does not detect the change so it forgets nothing. The next green batch also does not detect change, so track building (or remembering) proceeds. The same is true for the next purple batch. This process occurs regardless of the state of the system: in-control, warning or out-of-control. In this example, the green and purple tracks may have detected a change due to outlier interference. It is beneficial for these two tracks to forget this information. On the other hand, the blue track, whose summary statistics may not have been as affected by the outlier(s) due the shifted sample, would retains this information.

4 Experimentation

Experimentation was conducted using the **MOA** framework for data stream mining [1], an open source software closely related to its offline counterpart **WEKA** [7]. We used both synthetic and real data streams in our evaluation. Due to space restrictions, we are only reporting the results against the synthetic data streams. Five of **MOA**'s synthetic data set generators, summarized in Table 1, were used in various configurations to produce the synthetic data. Varying degrees of noise were tested using synthetic data in order to test and compare the algorithms' robustness to noise. Each synthetic data set was injected with 0, 1, 2, 3, 4, 5, 10, 15, 20 and 25% noise using the **WEKA** "addNoise" filter. This noise was applied to every attribute but the class attribute, since **CDC-Stream** and **FG-CDCStream** are unsupervised change detectors. Experimentation was performed on a machine with an Intel i7-4770 processor, 16GB of memory, using the Windows 10 Pro x64 Operating System. The original **CDC-Stream** study [9] tested its strategy using only the Naïve Bayes classifier. For a more comprehensive understanding of the behaviour of both **CDCStream** and **FG-CDCStream**, we employed the Naïve Bayes classifier, the Hoeffding tree incremental learning, and the K-NN lazy learning strategy. Each of the classifiers used for experimentation are available in **MOA**.

Four measures [2] were used to evaluate change detection strategies, as follows. The mean time between false alarms (MTFA) describes the average distance between changes detected by the detector that do not correspond to true changes in the data. It follows that a high MTFA value is desirable. The mean time to detection (MTD) describes how quickly the change detector detects a true change and a low MTD value is sought. Further, the missed detection rate (MDR) gives the probability that the change detector will not detect a true change when it occurs and it follows that a low value is preferred. Finally, the calculated mean time ratio (MTR) describes the compromise between fast detection and false alarms, as shown in the equation, and a higher MTR value is required.

$$MTR = \frac{MTFA}{MTD}(1 - MDR) \tag{5}$$

These performance measures allow for a detailed examination of a change detector's effectiveness in detecting true changes quickly while remaining robust to noise and issuing few false alarms, thus providing researchers with a way to directly assess a change detector's performance [2].

A more indirect way of evaluating change detection methods, and the most common in the literature, is the measuring of accuracy-type performance measures. We considered the classification accuracy, κ and κ^+ accuracy-type measures focusing on the κ^+ results due to their comprehensiveness. The κ statistic considers chance agreements, and the κ^+ statistic [3] considers the temporal dependence often present in data streams.(Interested readers are referred to [3] for a detailed discussion on the evaluation of data stream classification algorithms.) We considered the progression of accuracy-type statistics throughout the stream, not only the final values, in order to gain more insights into change detector performance. For instance, the steepness of the drop in accuracy-type performance at a change point, and the swiftness of recovery provides more information than a single value representative of an overall accuracy.

Abrupt changes were injected and streams of one, four and seven changes were studied in various orders. Different stream sample sizes, change widths, patterns and distances between changes (as well as magnitudes and orderings) were studied on account of comprehensiveness. For single change scenarios, changes

Table 1. Synthetic data (basic characteristics)

Dataset	Classes	Features	Categorical	Numerical
LED	10	24	24	0
Stagger	2	3	3	0
Mixed	2	4	2	2
Agrawal	2	9	3	6
ConceptDriftStream	Varies	Varies	Varies	Varies

were injected half way through the stream in order to observe system behaviour well before and well after the change. For multiple abrupt changes, four different distances between changes were studied, namely 500, 1000, 2500 and 5000 instances. This was done in order to assess the change detectors' abilities to detect changes in succession and to compare recovery times.

4.1 Results

Table 2 shows the average performance of the classifiers in abrupt drift scenarios. The table shows that the Hoeffding Tree incremental learner performs the best, overall, in all cases. Further, the **FG-CDCStream** algorithm generally outperforms **CDCStream**. The greater gap between the performance in Hoeffding Tree and Naïve Bayes change detectors in **CDCStream** compared to **FG-CDCStream**, is likely a consequence of the Hoeffding Tree's superior ability to conform to streaming data naturally, through data acquisition and without explicit concept drift detection.

Table 2. Average algorithm performance comparison by classifier

Change type	Algorithm	Classifier	κ^+	κ	Accuracy (%)
Abrupt	FG-CDC	HT	**93.74**	93.67	95.72
		NB	91.42	91.45	94.55
		IbK	79.93	79.92	84.55
	CDC	HT	71.70	76.28	83.81
		NB	57.58	60.05	71.65
		IbK	60.33	69.34	80.72

Next, we focus on our experimental results against single abrupt change data streams generated using the LED data set, as shown in Figs. 3 and 4. The value of $votes_c$ was set to 15, by inspection. Note that the information for MTFA, MTD and MTR performance measures are not available for **CDCStream**. This is because **CDCStream** was unsuccessful in detecting changes in the case of the

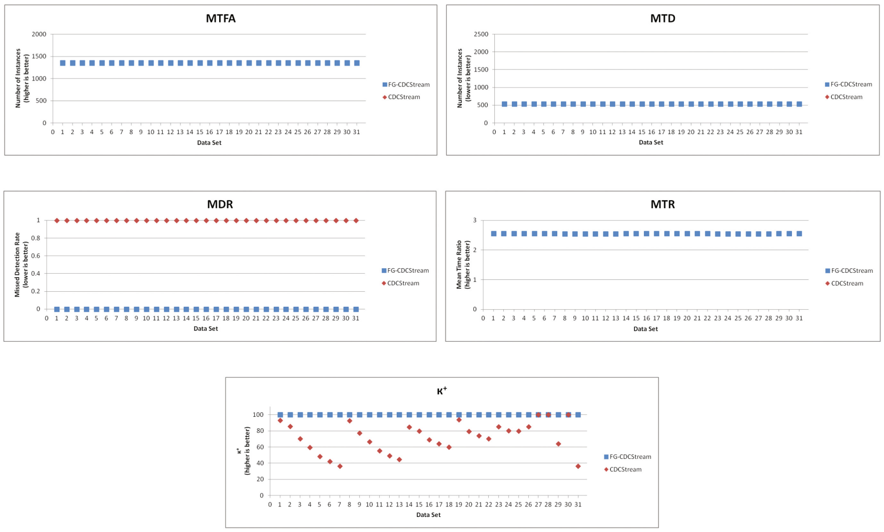

Fig. 3. A graphical comparison of performance statistics of FG-CDCStream and CDC-Stream on 31 different data streams containing a single abrupt change each.

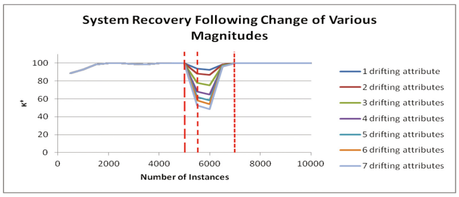

Fig. 4. System recovery following a single abrupt change. The long dashed line indicates the true change, the medium dashed line the approximate change detection and the short dashed line the full classifier recovery.

single abrupt drifting data streams. In contrast, **FG-CDCStream** produces good change detection statistics in the single abrupt change scenario. These measures remain consistent across all of the streams. Specifically, our algorithm issues false alarms at a low rate (with MTFA values around 1400) and successfully detects the true change in every case.

Next, we consider the multiple change scenario. Recall that multiple changes were injected into streams at varying distances from one another, namely 500, 1000, 2500 and 5000 instances. The averaged results and the corresponding trends may be observed in Fig. 5. Similar to the single abrupt change scenario, **CDCStream** failed to detect concept drift in the multiple abrupt change scenario. (Note that our evaluation confirms that its accuracy-type statistics are equivalent to those of a regular incremental Naïve Bayes classifier with no explicit change detection functionality.) The κ^{+} graph in Fig. 5 shows that as distance

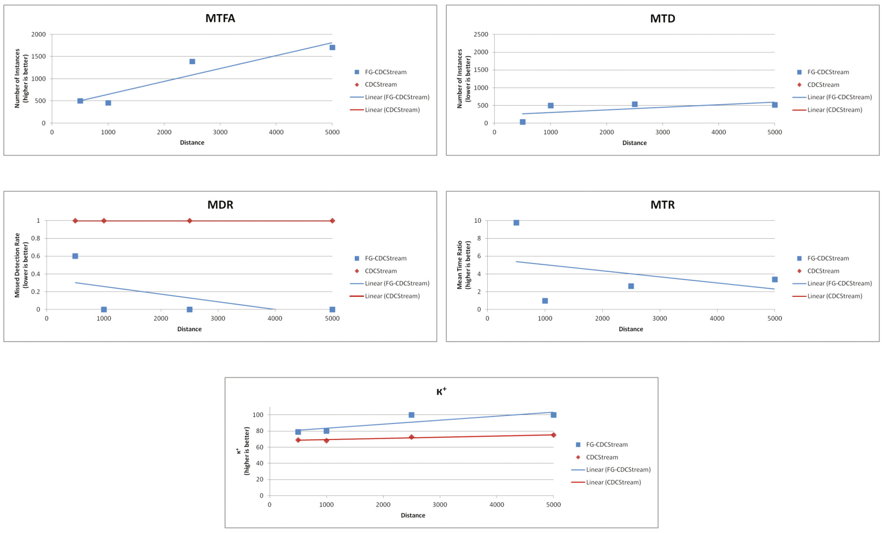

Fig. 5. A graphical comparison of averaged performance statistics of FG-CDCStream and CDCStream for multiple abrupt drifts and the associated trends as distance between the changes increases.

between changes increases, the κ^+ increases only slightly for **CDCStream**. This slight increase is due entirely to the natural adaptation of the incremental classifier over time.

For the **FG-CDCStream** change detector, performance and distance between changes generally correlate positively, as one would expect. If there is a longer period between the changes, **FG-CDCStream** has more information to detect and recover from change. This leads to a more streamlined representation of the current concept by growing its more accurate tracks' windows of the current concept and pruning the less accurate ones. A clear increase in MTFA occurs with increasing distance between changes. This trend corresponds to the systems' ability to forget more of previous concepts (discard more of the less accurate tracks) when more time is available between changes. When less time is available, it is likely that more tracks still contain the information of previous concepts. Nevertheless, the **FG-CDCStream** is able to detect concept drift fast, while maintaining a low false detection rate.

The effect of concept drift injection magnitudes on the κ^+ performance measure throughout the stream is shown in Fig. 6. The particular stream represented was injected with seven changes from low to high change magnitudes. At shorter distances, change points are less defined and the system has a more difficulty to recover, especially from higher magnitude change. This is due to less reliable change detection, as discussed above and therefore less representative models.

Finally, we turn our attention to the case when increasing levels of noise were injected into streams with multiple abrupt drifts. The averaged results and

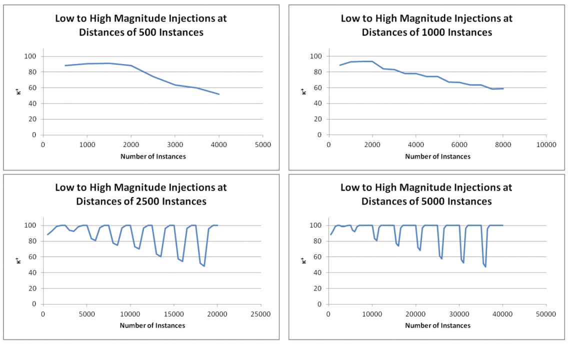

Fig. 6. FG-CDCStream κ^+ performance on LED data stream with varying distances between change injection points.

corresponding trends may be observed in Fig. 7. The results indicate that **FG-CDCStream** is robust to noise, especially when multiple changes are present and the distances between those changes are large. It follows that, the level of acceptable noise would depend on the particular application. In general, though, since a value greater than 15% noise in every attribute is a rather high noise ratio to be occurring in real data, it is likely that **FG-CDCStream** would perform satisfactorily on most streams containing abrupt drifts.

4.2 Discussion

The above-mentioned experimental results confirm that the **FG-CDCStream** algorithm leads to improved abrupt change detection and adaptation.

Improving Abrupt Change Detection. FG-CDCStream was designed with the intention of retaining the appealing qualities of **CDCStream** while improving change detection elements where opportunities exist. It essentially uses an ensemble of change detectors, each containing slightly shifted information, that vote on whether or not a change has occurred. Only the change detectors in close proximity in the forward direction to the detector that first flags change submits a vote. This is appropriate, due to the temporal nature of the data and the desire to detect a change quickly. The voting system decreases the chances of completely losing information due to aggregation. This not only increases the probability of detecting the change (reducing MDR) and detecting it quickly (reducing MTD), but provides another advantage of decreasing the probability of false detections (increasing MTFA). Since a vote is required to confirm a change, slight variations in distribution that would flag a false change in **CDCStream** would, in **FG-CDCStream**, be outvoted and therefore not flag a false change. This further increases the algorithm's robustness. In summary, **FG-CDCStream** overcomes the issues that **CDCStream** has with regard to the batch scenario and dilution due to aggregation, improving its change detection capabilities.

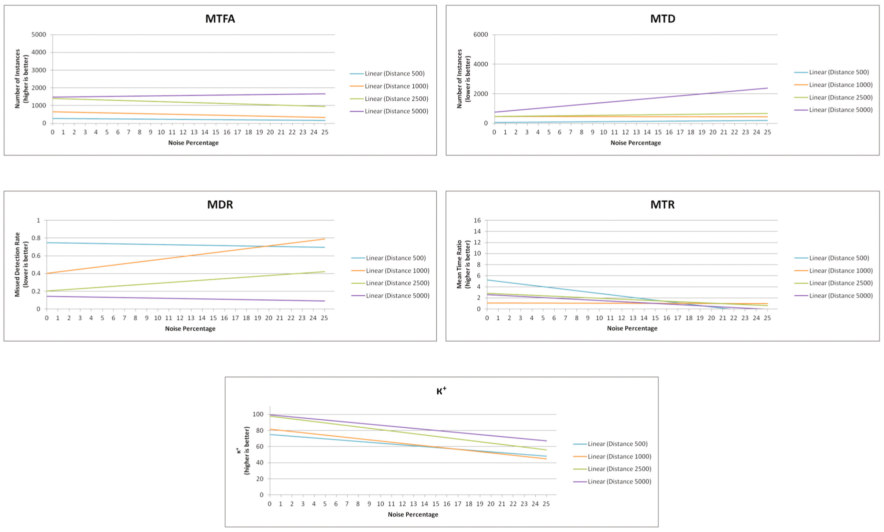

Fig. 7. A graphical comparison of linear trends of the averaged performance statistics of FG-CDCStream on six data sets containing a multiple abrupt changes at varying distances from one another over increasing noise levels.

Improving Adaptation. A major limitation of **CDCStream** is its potential for a very unrepresentative replacement classifier during adaptation. This is caused by the termination requirement of a warning period being only a change period regardless of how closely or distantly that change period occurs. This requirement potentially results in a background classifier that is non-representative of the new distribution following a change detection. **FG-CDCStream** reconciles this by permitting a warning period to terminate if it is followed by either a static period or a change. In **FG-CDCStream**, if a warning period is followed by a static period, the background classifier initiated by the warning is ignored rather than continuing to build until the next change occurs. The only background classifier that may be used as an actual replacement classifier in **FG-CDCStream** is one that is initiated by a warning immediately prior (i.e. there is no static period in between) a detected change. If, by chance, a change occurs without a preceding warning, the replacement classifier is created from the most recent batch only. This follows, since a change occurring absent of any warning is likely to be abrupt, and therefore a short historical window is appropriate.

5 Conclusion

This paper introduces the **FG-CDCStream** algorithm, an unsupervised and context-based change detection algorithm for streaming categorical data. This algorithm provides users with more precise information than that of learning-based methods about the changing distributions in categorical data streams and provides context-based change detection capabilities for categorical data, previously undocumented in machine learning research. The experimental results show that **FG-CDCStream** method is able to detect abrupt drift fast, while maintaining both lower false alarm rates and lower detection misses.

This research would benefit from further exploration of the effects of different types of data streams on algorithm performance. For instance, a comprehensive study on real data might provide further insight into algorithm behaviour. Additional research on no-change data streams would also be beneficial. Further, studying how attribute cardinality effects the algorithm would be useful as well.

References

1. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. J. Mach. Learn. Res. **11**, 1601–1604 (2010)
2. Bifet, A., Read, J., Pfahringer, B., Holmes, G., Žliobaitė, I.: CD-MOA: change detection framework for massive online analysis. In: Tucker, A., Höppner, F., Siebes, A., Swift, S. (eds.) IDA 2013. LNCS, vol. 8207, pp. 92–103. Springer, Heidelberg (2013). doi:10.1007/978-3-642-41398-8_9
3. Bifet, A., Read, J., Žliobaitė, I., Pfahringer, B., Holmes, G.: Pitfalls in benchmarking data stream classification and how to avoid them. In: Blockeel, H., Kersting, K., Nijssen, S., Železný, F. (eds.) ECML PKDD 2013. LNCS, vol. 8188, pp. 465–479. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40988-2_30
4. Boriah, S., Chandola, V., Kumar, V.: Similarity measures for categorical data: a comparative evaluation. In Proceedings of the 2008 SIAM International Conference on Data Mining, pp. 243–254 (2008)
5. Cao, F., Zhexue Huang, J., Liang, J.: Trend analysis of categorical data streams with a concept change method. Inf. Sci. **276**, 160–173 (2014)
6. Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. **46**(4), 1–37 (2014)
7. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. ACM SIGKDD Explor. **11**(1), 10–18 (2009)
8. Ienco, D., Pensa, R.G., Meo, R.L.: From context to distance: learning dissimilarity for categorical data clustering. ACM Trans. Knowl. Discov. Data **6**(1), 1–25 (2012)
9. Ienco, D., Bifet, A., Pfahringer, B., Poncelet, P.: Change detection in categorical evolving data streams. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC 2014), pp. 274–279 (2014)
10. Yu, L., Liu, H.: Feature selection for high-dimensional data: a fast correlation-based filter solution. In: Proceedings of Twentieth International Conference on Machine Learning, vol. 2, pp. 856–863 (2003)