CD-TDS: Change Detection in Transactional Data Streams for Frequent Pattern Mining

Yun Sing Koh
Department of Computer Science
The University of Auckland
Private Bag 92019, Auckland 1142, New Zealand
Email: ykoh@cs.auckland.ac.nz

***Abstract*—Online mining is a difficult task especially when such data streams evolve over time. Evolving data stream occurs when concepts drift or change completely, is becoming one of the core issues. A large portion of change detection research are carried out in the area of supervised learning, very little has been carried out for unlabeled data specifically in the area of transactional data streams. Overall when we monitor changes in transactional data we can consider two different types of changes: local and global change. Local changes are changes in distribution of the data, whereas global changes are data composition changes within the data stream. To detect changes in transactional data streams containing unlabeled data, we introduce a new technique called CD-TDS, that detects both these changes. Our change detector can identifies changes in relationships between items as data evolves with the progression of a stream. Crucially, detection of global drift enables us to better understand the dynamics in relationships that takes place over time. Experimental results using both real world and synthetic data show that the proposed approach is robust to noise and identifies structural changes with a high true positive rate while preserving a low false alarm rate.**

1. Introduction

A substantial amount of recent work has focused on continuous mining of data streams. One of the main challenges in data streams is the distribution and the composition of the data may change over time. These changes can affect the accuracy performance of mining. Data stream mining comprises of several classes of tasks including clustering, classification, frequent-pattern mining, and time-series analysis. Whilst frequent-pattern mining in data stream has become a popular topic in data mining research [1], [2], [3], including in areas such as web usage mining, network traffic analysis, intrusion detection, and bioinformatics. Most of the frequent stream mining techniques mainly concentrates on efficiently processing the data and rarely deals with detecting changes within the data stream. Most change detection for data stream works for in classification environment, whereby we monitor the error rate of a classifier by comparing the accuracy of the its prediction. This type of change is also known as concept drift. It is not ideal to use current change detection techniques meant for a predictive analytics, which monitors the accuracy of the model, for an unsupervised data stream and frequent pattern mining task.

As data stream is dynamic in nature, its data distribution may alter with time. As a result the frequencies of items may change continuously in an unforeseen manner. Consequently, the performance of the frequent pattern mining method can deteriorate and become unstable as time passes. For example consider a supermarket data, the concept of stream mining data is defined by the set of merchandise (commodity) sold by the store and the bought frequency of each commodity and the together-purchased frequency of each combination of multiple commodities. Most of the research only concentrate on changes in the frequency distribution of instances (item or pattern). In mining for patterns within unlabeled transactional data streams there exist multiple types of changes that is different to classical change detection using prediction models.

Here we coin two different change types within the data, global and local changes. A global change represents a data composition change with the stream whereby the type of commodity sold together changes over time. Here a change can be represented as there is an uptake in the sales of new products in the store. There is another type of change, local change, whereby there is a change in the frequency of the commodities sold but the data composition of the together-purchase items remains the same. For example there maybe an influx of sales of particular commodities due to special holiday seasons. Previous research use change detectors to monitor the changes in frequency which we termed as local change.

In frequent pattern mining task we are interested in both local and global changes. Both these type of changes are useful to decision makers. To detect these changes we propose a new technique called Change Detector for Transactional Data Streams (CD-TDS). This particular change detector monitors both local and global changes.

Our main contributions include:

- proposing two different types of changes local change and global change, and
- proposing a novel technique CD-TDS.

This paper is organized as follows. Section 2 provides a

978-1-5090-0620-5/16/$31.00

brief overview of the related research. Section 3 discusses the classification of different change types. Section 4 introduces the concepts that underlie our CD-TDS algorithm. Section 5 compares our CD-TDS to other algorithms by experimentation on datasets that have different characteristics, and Section 6 outlines the conclusions.

2. Related Work

In this section, we introduce existing work related to our research.

Concept Drift Detection The concept drift detection problem has a classic statistical interpretation: given a sample of data, does this sample represent a single homogeneous distribution or is there some point in the data (i.e the concept change point) at which the data distribution has undergone a significant shift from a statistical point of view? All concept change detection approaches in the literature formulate the problem from this viewpoint but the models and the algorithms used to solve this problem differ greatly in their detail. In data streams, the relations between the input data and their labels may change due to concept drift [4], [5].

The problem of drift detection and analysis can be more formally framed. Let $S_1 = (x_1, x_2, ..., x_m)$ and $S_2 = (x_{m+1}, ..., x_n)$ with $0 < m < n$ representing two samples of instances from a stream with population means μ_1 and μ_2 respectively. The drift detection problem can be expressed as testing the null hypothesis H_0 such that $\mu_1 = \mu_2$, *i.e.* the two samples are drawn from the same distribution against the alternate hypothesis H_1 that they arrive from different distributions with $\mu_1 \neq \mu_2$. In practice the underlying data distribution is unknown and a test statistic based on sample means needs to be constructed by the drift detector. If the null hypothesis is accepted incorrectly when a change has occurred then a false negative has taken place. On the other hand if the drift detector accepts H_1 when no change has occurred in the data distribution then a false positive has occurred. Since the population mean of the underlying distribution is unknown, sample means need to be used to perform the above hypothesis tests. The hypothesis tests can be restated as the following. We accept hypothesis H_1 whenever $Pr((|\hat{\mu_1} - \hat{\mu_2}|) \geq \epsilon) > \delta$, where δ lies in the interval $(0, 1)$ and is a parameter that controls the maximum allowable false positive rate, while ϵ is a function of δ and the test statistic used to model the difference between the sample means.

There are various drift detection methods in the area [6], [7], [8], [9]. Sebastiao and Gama [10] present a concise survey on drift detection methods. They point out that methods used fall into four basic categories: Statistical Process Control (SPC), Adaptive Windowing, Fixed Cumulative Windowing Schemes and other classic statistical change detection methods such as the Page Hinkley test. Gama et al. [11] adapted the SPC method for change detection and formulate an algorithm in a data stream context. Bifet et al. [12] proposed an adaptive windowing scheme called ADWIN that is based on the use of the Hoeffding bound to detect concept change. The ADWIN algorithm was shown to outperform the SPC approach and has the attractive property of providing rigorous guarantees on false positive and false negative rates. Recently SeqDrift proposed in [13] used a sequential hypothesis testing strategy, thus avoiding the need to re-examine previous cuts and resulting in greatly improved processing times.

Structural Drift Detection Current structural drift detection techniques look at the data evolution from a systematic point of view where each data stream is seen as part of a set and a change in its data reflects some structural drift only if there are impacts on the proximity to the other streams [14]. Current techniques look at detecting changes at a macro level using proximity measures based on global changes and ignore changes that takes place between different components in the system.

Streams present inherit dynamics in the flow of data that are usually not considered in the concept of usual data mining. The distribution generating the examples of each stream may change over time. Thus, new approaches are needed to consider this possibility of change and new methods have been proposed to deal with variable concept drift [11]. However, detecting concept drift as usually conceived for one variable is not the same as detecting concept drift on the clustering of the components within data streams [15]. Structural drift is a point in the stream of data where the structure of the clustering or clustering structure generated with previous data no longer holds since it no longer represents the new relations of proximity and dissimilarity between the streams [14].

Frequent Pattern Mining in Data Streams The following is a formal statement of frequent pattern mining for a transactional data stream. Let $I = i_1, i_2, \ldots, i_m$ be the set of all possible items. A set $P \subseteq I$ of items is called an itemset or pattern. In particular, an itemset with k items is called a k-itemset. In association rule mining, the main function of a unique transaction ID is to allow an itemset to occur more than once in a database. Every transaction contains a unique transaction ID tid. A transaction $t = (tid, P)$ is a tuple where P is an itemset. A transaction $t = (tid, P)$ is said to contain itemset X if $X \subseteq t$. A streaming dataset T is a s sequence of transactions, *i.e.* $T = \{t_1, t_2, t_3, \ldots\}$. The sequence is of indefinite length. At time j, the data window $W_{i,j}$ is a finite subsequence of transactions from an earlier time i to the present time j, *i.e.* $W_{i,j} = \{t_i, t_{i+1}, \ldots, t_j\}$ where $i \leq j$. Because smaller patterns may be subsets of larger patterns, any data window may contain numerous patterns. Let $Patt(W)$ be a subpatterns enumeration function which generates a multiset of patterns contained within a data window W. The support of a sub-pattern P, denoted by *supp*(P, W) in a data window W is the proportion of transaction in the window W containing P.

Given a user-specified minimum support threshold *minsupp* $\in [0, 1]$, P is called a frequent itemset or frequent pattern in T if supp$(P, W) \geq$ *minsupp*. The problem of mining frequent patterns is to find the complete set of frequent patterns in a transaction database with respect to a given support threshold. In mining frequent itemsets on data

streams, previous research mainly focus on adapting rather than detecting changes [cite]. Although maintaining an up-to-date model is important, it is also imperative to signal an alarm when change is detected. Users may be interested to understand the nature of the change so that they can take the appropriate action against such change in the shortest possible time.

Most of the methods allow us to find approximate frequent patterns within a guaranteed error bound or require an additional pruning threshold. Up until now only a few proposed approaches [1], [2], [3] were designed to find the exact set of recent frequent patterns from a data stream using a sliding window model.

3. Classification of Change Types in Transactional Data Streams

Most current change detection technique refers to changes as concept drifts. The literature has given a probabilistic definition of concept. We can define a concept as the prior class probabilities $P(Y)$ and class conditional probabilities $P(X|Y)$ [16]. As $P(Y)$ and $P(X|Y)$ uniquely determines the joint distribution $P(X,Y)$ and vice versa, this is equivalent to defining a concept as the joint distribution $P(X,Y)$]. We can use the definition for stream classification as $Concept = P(X,Y)$ proposed by Gama et al. [17].

Beyond classification learning in an unsupervised case such as transactional dataset, where there is no special class attribute $Concept = P(X)$. In the context of a data stream, we need to recognize that concepts may change over time. To this end we define the concept at a particular time t as $P_t(X)$ A change in concept occurs between times t and u when the distributions change, $P_t(X) \neq P_u(X)$.

In this section we will look at the different types of changes that can occur in a transactional data stream. We can characterize the changes based into different types. Figure 1 shows the different sub-classification found in changes for transactional data stream. Monitoring the distribution change or frequency change in a data stream can be consider as local changes. We consider it as local changes as we only monitor the distributions of the appearance of item changes individually without considering the changes of the connections between items. The second type of change is global change which occurs when the connections between the co-occurrence of items change. Both these changes can have changes that are both abrupt or gradual.

Consider a dataset that has five items A, B, C, D, E in a data stream, there are two different scenarios of changes in a data stream. In local change for a particular item A, as shown in Figure 2, we see that item A has a change in support whereby its support increases overtime however the co-occurrence of items with item A remains stable. In this particular case item A is connected to B, C, and D, across the time period when there was a distribution change in support for item A. We notice that the change is based local distribution change, and overall the structure of the graph has remained the same.

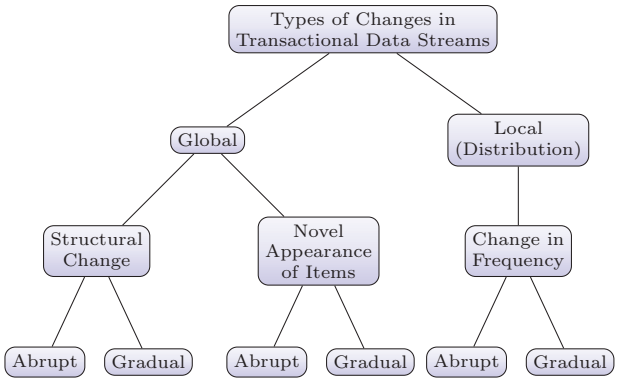

Figure 1. Roadmap of Different Changes in Transactional Dataset

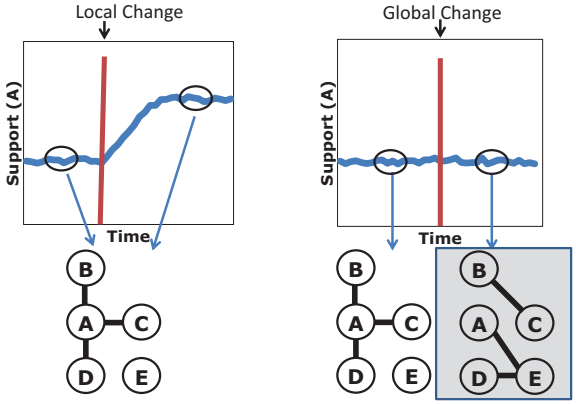

Figure 2. Example of Different Changes Types

The problem of **local change detection** and analysis can be more formally framed. Let $S_1 = (t_1, t_2, ..., t_m)$ and $S_2 = (t_{m+1}, ..., t_n)$ with $0 < m < n$ representing two samples of transactions from a stream. Given item $i \subset t_x \in S_1$ and $i \subset t_y \in S_2$, the support of $i \in S_1$ is $supp(i, S_1)$ and the support of $i \in S_2$ is $supp(i, S_2)$. The local change problem for item i can be expressed as testing the null hypothesis H_0 such that $supp(i, S_1) = supp(i, S_2)$, *i.e.* the two samples are drawn from the same distribution against the alternate hypothesis H_1 that they arrive from different distributions with $supp(i, S_1) \neq supp(i, S_2)$. We accept hypothesis H_1 whenever $Pr((|supp(i, S_1) - supp(i, S_2)|) \geq \epsilon) > \delta$, where δ lies in the interval $(0, 1)$ and is a parameter that controls the maximum allowable false positive rate, while ϵ is a function of δ and the test statistic used to model the difference between the sample means.

In global change, as shown in Figure 2, we notice that the co-occurrence of items has changed. Here we see that A co-occurred with B, C, and D but later is only connected to E. In this case the structure of the graph has changed tremendously based on the co-occurrence of items. In this

example we note that the support of item A remains stable. In reality we consider that there is a global change regardless of whether there is a fluctuation in the support of item A.

The problem of **global change detection** and analysis can be more formally framed. Let $S_1 = (t_1, t_2, ..., t_m)$ and $S_2 = (t_{m+1}, ..., t_n)$ with $0 < m < n$ representing two samples of transactions from a stream, and G_1 represents the connectivity graph built from S_1 and G_2 represents the connectivity graph built from S_2. There is a connection between two items if the items co-occur together. The global change problem can be expressed as testing the null hypothesis H_0 such that $G_1 \cong G_2$, *i.e.* the two samples have similar underlying connectivity structure against the alternate hypothesis H_1 that they arrive from a different connectivity model with $G_1 \ncong G_2$. We accept hypothesis H_1 whenever $sim(|G_1, G2|) > \gamma$, where sim is a similarity function that compares two graphs, and γ is a threshold parameter that controls the maximum allowable differences amongst the two graphs.

4. Change Detection for Transactional Data Stream

In this section we describe our algorithm for dynamically detecting drift for transactional data, make a formal claim about its performance, and derive an additional variation. The general framework of this technique can be divided into two parts: (i) local drift detection and (ii) global drift detection. We monitor both these changes simultaneously. Each of the changes provides a different type of information to the users. A local change indicates the number of occurrence of a scenario is changing without causing a change in connections amongst items. In a classification sense this can be likened to a virtual shift [16]. However in frequent pattern mining, this type of change is particularly interesting as it provides information on the increase of decrease of a set of items. Based on support and confidence framework, a particular pattern produced based on the set of items may become more interesting if they co-occurred more often. Whereas with a global change, the connections between the items have changed indicating that new patterns may have emerged and/or some old patterns may have disappeared. It is crucial for frequent pattern mining techniques for a data stream environment to detect these changes, as they provide further actionable knowledge for the users.

4.1. Local Change

The general idea of a local change is simple: whenever two sub-windows S_0 and S_1 from a stream S are sufficiently dissimilar a split operation is needed, whereby, the older sub-window, S_0, is dropped. At this point we note that there is structural drift within the stream and S_0 does not represent the current structure within the dataset. In this research we used the Hoeffding bound [18], which provides guarantee that a drift is signalled with probability at most δ (a user defined parameter): $Pr(|\mu S_0 - \mu S_1| \geq \epsilon) \leq \delta$ where μS_0 is the sample mean of the reference window of data, S_0, and μS_1 is the sample mean of the current window of data, S_1. The ϵ value is a function of δ parameter and is the test statistic used to model the difference between the sample mean of the two windows.

Essentially when the difference in sample means between the two windows is greater than the test statistic ϵ, a drift will be signaled. ϵ is given by the Hoeffding bound with Bonferroni correction as:

$$\epsilon_{localcut} = \sqrt{\frac{2}{n} \cdot \sigma_S^2 \cdot ln\frac{2}{\delta'}} + \frac{2}{3m} ln\frac{2}{\delta'}$$

where $m = \frac{1}{1/|S_0| + 1/|S_1|}$, $\delta' = \frac{\delta}{|S_0| + |S_1|}$ For local drift detection, we monitor the distribution change of each item in the stream using similar to that of ADWIN.

4.2. Global Change

To detect global change using graphs we represent the connections between items using a tree G as a representation of the stream S, so that the sub-windows S_0 and S_1 are represented by two trees G_0 and G_1. A tree structure is a normal form of representing the connections of items in transactional datasets and has been commonly used in previous research. To determine whether the sub-windows are sufficiently dissimilar we can use an appropriate statistical test for significant pairwise disordering of the tree structures from the sub-windows. The information needed is a significance level p, the lengths of the sub-windows, and their contents.

Measures of similarity between two trees have been well studied in combinatorial pattern matching. Various functions, such as edit distance, largest common subtree and smallest common super-tree have been proposed to measure similarity between trees. We chose the Levenshtein edit distance as it is most suited to our objective of comparing differences across two sub-windows in terms of the number of changes required to transform the first sub-window into the second.

We measure how similar two trees are based on the following assumption. Given two trees, T_{CURR} and T_{PREV} and a distance threshold ϵ, let $dist_T$ be a distance function defined on trees. If $dist_T\,(T_{CURR}, T_{PREV}) \leq \epsilon_{cut}$ then T_{CURR} and T_{PREV} are similar. Otherwise, if $dist_T\,(T_{CURR}, T_{PREV}) > \epsilon_{cut}$ then T_{CURR} and T_{PREV} are not similar and there is structural drift between T_{CURR} and T_{PREV}. Here we introduce a one-to-one sequencing method to transform trees to strings. We first traverse each tree in a pre-order sequence. The root-to-leaf paths, P_{PREV_i} and P_{CURR_j} generated from T_{PREV} and T_{CURR}.

The Levenshtein distance $e_L(x, x')$ between strings $x = x_1, \ldots x_t$ and $x' = x'_1 \ldots x'_v$ can be computed in $O(|x| \cdot |x'|)$ time using dynamic programming. Using the Levenshtein distance we formulate the distance function $dist_T$ as:

$$dist_T = \frac{1}{|P|} \sum_{i \in |P|} \frac{e_L(P_{PREV_i}, P_{CURR_i})^2}{|P_{PREV_i}| . |P_{CURR_i}|} \qquad (1)$$

where $|P|$ is the number of paths in T_{PREV}; $|P_{PREV_i}|$, $|P_{CURR_i}|$ are the sizes of paths P_{PREV_i} and P_{CURR_i} respectively. Equation 1 represents the percentage of changes from the previous tree to the current tree. A $dist_T$ of 0 indicates that T_{PREV} and T_{CURR} represent the same tree, whereas a $dist_T$ of 1 indicates that the two trees differ in every path.

We use rank correlation as a measure of the degree of similarity between rankings between the two trees. Rank correlation measures are often used as test statistics for the independence of two sorting "criteria" or of two continuous variables X and Y whose joint distribution is unspecified. Spearman's ρ correlation measures the distance between ranks. It has been widely used in different fields and is thus used in our research. Spearman's ρ states that:

$$\rho = 1 - \frac{6\sum_{i=1}^{n}(R_{i1} - R_{i2})^2}{n(n^2-1)} \tag{2}$$

Here $(R_{1h}, \ldots, R_{nh})$ is the set of ranks of a sample size of n according to the h-th sorting criterion $(h = 1, 2)$. Equation 2 shows that ρ is based on the squared differences of the pairs of corresponding ranks. Given a significance level α and a corresponding critical value P, when $\rho \geq P(\alpha)$ a significant difference between the two rankings is said to exist. To determine the ϵ_{cut} value, we substitute the critical P-value, δ as the boundary value in Equation 2.

$$\rho = 1 - \frac{6\sum_{i=1}^{n}(R_{i1} - R_{i2})^2}{n(n^2-1)} = \delta$$

$$\frac{\sum_{i=1}^{n}(R_{i1} - R_{i2})^2}{n^2} = \frac{(1-\delta)(n^2-1)}{6n} \tag{3}$$

In Equation 2 n reflects the number of tree paths, we use n^2 as an approximation to $(|P_{PREV_i}|.|P_{CURR_i}|)$. Also we note that $e_L(P_{PREV_i}, P_{CURR_i}) = (R_{i1} - R_{i2})^2$, giving:

$$dist_T = \frac{1}{|P|}\sum_{i=1}^{n}\frac{(R_{i1} - R_{i2})^2}{n^2}$$

At the critical value δ, the $dist_T$ is the upper boundary for the ϵ_{cut} threshold. Thus,

$$dist_T = \frac{1}{|P|}\sum_{i=1}^{n}\frac{(R_{i1} - R_{i2})^2}{n^2} = \epsilon_{cut} \tag{4}$$

Substituting Equation 4 into Equation 3, we can determine the ϵ_{cut} threshold below.

$$\epsilon_{globalcut} = \frac{(1-\delta)(n^2-1)}{6n \cdot |P|} \tag{5}$$

Pairing edit distance with Spearman's ρ correlation is the ideal choice, as we are interested in the homogeneity between the connections belonging to the two windows rather than a 1-1 comparison.

4.3. CD-TDS

In our proposed algorithm we store transactions from the data stream into trees. As with normal change detection we need to determine when to check for changes within the stream. If change detection checks are performed too often it becomes inefficient, whereas if we do not perform the checks sufficiently we may miss changes occurred within the stream. To efficiently determine the potential change points we adapted the exponential histogram [19] to obtain a compressed version of the detection window. We use the exponential histogram on our transactional dataset in the form of trees (shown in Algorithm 2). We build a time-based histogram in which every bucket represents a contiguous time interval and maintains transactions that arrive in that interval. In our algorithm each bucket is represented by a tree.

Our data structure is a variation on exponential histograms, a data structure that maintains an approximation of a set of transactions of length S. This adaptively re-size the window strategically. We used a capacity parameter to control the size of the window. Capacity is the number of elements that arrived in the bucket. We use a maximum capacity threshold M that is set according to the amount of memory available to determine the number of buckets. The choice of here M does not reflect any assumption about the time-scale of change. The transactions are checked at a geometric rate. Each time a new transaction arrives, we create a new bucket of capacity 1. Thereafter, we compress the rest of the buckets: When there are $M + 1$ buckets of size 2^i, we merge the two oldest ones (adding its capacity) into a bucket of size 2^{i+1} as shown in Algorithms 2 and 3. Consider the following example with a sliding window of 11 elements. We register it as:

In our example, suppose $M = 2$. If a new element arrives then we have:

There are 3 buckets of capacity 1, and so we compress the to older buckets T_{x_9} and $T_{x_{10}}$ together:

In practice each bucket size should approximate to 30 times the number of unique items.

In Algorithm 5 $supp(i, T_{CURR})$ represents support of item i in T_{CURR}, and $supp(i, T_{PREV})$ represents support of item i in T_{PREV}.

Algorithm 1 Adaptive Window

Initialize W as an empty list of buckets, each bucket is stored as a tree
for all $t > 0$ **do**
  INSERT_ELEMENT(e, W)
  split W into $T_{PREV}.T_{CURR}$
  if GLOBAL_DRIFT(T_{PREV}, T_{CURR}) or LOCAL_DRIFT(T_{PREV}, T_{CURR}) **then**
    drift $\leftarrow$ True
    Drop T_{PREV}
  end if
  output DriftAlarm
end for

Algorithm 2 INSERT_ELEMENT(Element e, List W)

create a new bucket b with content e and capacity 1 (each bucket is a tree)
$W \leftarrow W \cup b$
MERGE_TREES(W)

Algorithm 3 MERGE_TREES(List W)

Traverse the list of buckets in increasing order
while If there are more than M buckets of the same capacity **do**
  while merge buckets **do**
    MERGE_TREES(sublist of W not traversed)
  end while
end while

Algorithm 4 GLOBAL_DRIFT(T_{PREV}, T_{CURR})

if $dist_T(T_{PREV}, T_{CURR}) > \epsilon_{globalcut}$ **then**
  return Global Drift
end if

Algorithm 5 LOCAL_DRIFT(T_{PREV}, T_{CURR})

for all $i \in T_{CURR}$ **do**
  if $|supp(i, T_{PREV}) - supp(i, T_{CURR})| > \epsilon_{localcut}$ **then**
    return Local Drift
  end if
end for

5. Evaluation

In this section we evaluate our CD-TDS technique against some of the standard performance measures used in evaluating change detectors. The performance measures include true detection, false alarms and detection delay to evaluate the performance of CD-TDS. The true detection measures the capacity to detect and react to change; false alarms measures resilience to false alarms when there is no drift, that is, to not detect drift when there is no change in the stream; the delay in detection measures the number of transactions required to detect a change after the occurrence of a structural drift. The delay measure is $delay = x - c$, where x is the transaction at which drift is detected, c is the actual point of drift. We evaluate both our techniques using synthetic datasets on the performance measures mentioned above. We compare the performance of our technique and compared it against local change detector.

5.1. Synthetic Datasets

We generated data streams of length $L = 1,000,000$. Our synthetic data generator was based on IBM's synthetic generator [20]. We initially generated two pools of large or frequent itemsets. The itemsets in the first pool is different from the itemsets in the second pool based on two different characteristics: the items that were used in the two pools may be different, and the connections amongst the items that forms the itemsets may alter as well. For example, we have a data stream with two pools of itemsets the first pool may contain three itemsets $\{ABC\}$, $\{DEF\}$, and $\{GHI\}$, where as the second pool may contain three itemsets $\{ADG\}$, $\{BEI\}$, and $\{BYZ\}$. Note that the connection for items A, B, D, E, G has changed and items C and H in the first pool does not exist in the second pool while new items Y and Z are introduced in the second pool.

Items in the large itemset have significant connections amongst items. To keep the tree stable during the first $L - x$ transactions, we insert transactions generated from the combination of itemsets from the first pool of large itemsets and then we intertwine transactions already selected from the first pool with those from the second pool of large itemsets in a probabilistic fashion past the $L-x$ transactions. In our experiments we set x to 20,000.

By introducing new items in the second pool we are trying to measure local changes of items with the data stream. Whereas changing the items connections can be considered a global change. To evaluate the sensitivity of our technique to gradual and abrupt changes, we use the concept of slope value. This allows us to characterize the speed of drift. At each time step after $L-x$ we increase the probability selecting an item from the second pool by the slope value. A small slope value indicates a more gradual drift whereas a higher slope value indicates a more abrupt drift. The slope value can be a keened to the probability of observing an itemset from the second pool of large itemsets. The primary goal of these experiments is to measure the accuracy of our technique global and local changes based on the connection induced by the itemsets from the first and second pool of itemset.

5.2. True Positive and Delay

We investigate the performance of our technique and compared it to only detecting local changes which is how changes are previously detected. We ran two experiments. In the first experiment we altered both the connections between the items and the distribution of the items. In the second experiment we altered the connections between the items whilst keeping the distribution of items stable.

TABLE 1. TRUE POSITIVE AND DELAY EXPERIMENTS (100 TRIALS) ON DATASETS CONTAINING ITEMS WITH NETWORK CHANGES AND DISTRIBUTION CHANGES

	CD-TDS					Local Change				
Slope	RD	Delay	Std Delay	Time (s)	Std Time	RD	Delay	Std Delay	Time (s)	Std Time
0.0001	100	1028.08	540.10	6.78	0.31	100.00	2152.96	1392.86	5.99	0.24
0.0002	100	873.28	202.23	6.80	0.31	100.00	2122.00	1076.20	6.01	0.23
0.0003	100	832.00	106.21	6.83	0.30	100.00	1905.28	910.50	6.07	0.31
0.0004	100	832.00	44.06	6.72	0.29	100.00	2142.64	1080.94	6.32	0.43

TABLE 2. TRUE POSITIVE AND DELAY EXPERIMENTS (100 TRIALS) DATASETS CONTAINING ITEMS WITH NETWORK CHANGES

	CD-TDS					Local Change				
Slope	RD	Delay	Std Delay	Time (s)	Std Time	RD	Delay	Std Delay	Time (s)	Std Time
0.0001	100	2060.08	1108.57	7.59	0.43	84	2257.14	1890.24	6.50	0.22
0.0002	100	1595.68	724.17	7.55	0.50	89	3383.01	2616.68	6.46	0.21
0.0003	100	1141.60	536.24	7.50	0.42	93	4382.97	3103.25	6.44	0.21
0.0004	100	966.16	376.50	7.44	0.46	91	5016.70	3026.66	6.46	0.20

In the first experiment we vary both the statistical distribution of mean support and the network connections of items in the dataset. We used two pools of large items to form the datasets for our experiment. We force minimum overlap of items in both pools, this attributes to a statistical distribution and the network connections between each of the items to alter at the change point. The experiments were repeated 100 times. In Table 1, RD represents change detected. The results show that both CD-TDS and local change detector detects all the changes in the datasets for each of the parameters. The overall delay from CD-TDS is lower than the local change detector.

In the second experiment, changing the connections among items enabled us to contrast traditional change detection methods for structural change detection in an environment where statistical distribution changes are less obvious when compared to structural changes. We use two pools of large itemsets to form the datasets used in our experiments. In this experiment we reuse items from the first pool to populate the second pool. This was accomplished by selecting items from the first pool and reassigning them randomly to form large itemsets in the second pool. We then interleave transactions created from each of the pools at the change point based on a specified slope value. This allows us to maintain a similar distribution of items used in both pools whilst the connections between the items in the second pool would have changed. In Table 2, the results show that both CD-TDS detects all the changes in the datasets for each of the parameters but the local change detector misses some of the changes.

5.3. False Positive

We investigate the rate of false alarms of our technique. The cost of a false positive is directly proportional to the cost associated with the action taken when such a false alarm is reported. The false alarm evaluation was based on a stream of length $L = 1,000,000$. Table 3 shows the false alarm rate detected was consistently low, although CD-TDS has a marginally higher number of false alarm.

TABLE 3. FALSE ALARM RATE

	FP	Std Dev	Time (s)	Std Time
CD-TDS	2.38×10^{-2}	6.7×10^{-3}	6.38	0.31
Local Change	2.30×10^{-2}	6.7×10^{-3}	6.02	0.28

5.4. Results from Real World Datasets

We then tested our algorithm on four different real world datasets, namely Kosarak [21], Airlines [22] and BMS-POS [21]. These datasets were chosen as they had diverse characteristics. However as there were no ground truth provided, it is hard to gauge where the right change points are. Table 4 provides a summary of the results from the real world experiments. Overall each of the datasets produced a varying degrees of changes detected and frequent patterns.

TABLE 4. RESULTS FROM REAL WORLD DATASETS

Datasets	Change Detected	Time	Average Rule Per Change	Std Rule Per Change
Kosarak	244	70.37	9.03	0.18
Airlines	122	15.85	12.35	1.50
BMS-POS	176	13.81	18.09	15.07

To show the number of rules produced at each of the change points detected we illustrate the number of rules produced versus the transaction number (time). Interestingly we notice that there are different scenarios of changes and patterns detected.

First, we notice that both datasets Kosarak (in Figure 3) and Airlines (in Figure 4) produced a stable number of rules at each of the change points. Although the rules

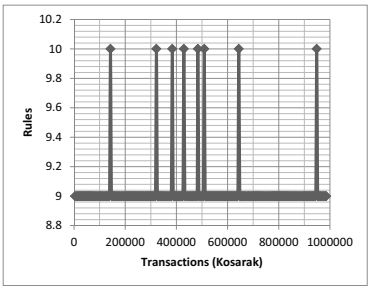

Figure 3. Kosarak

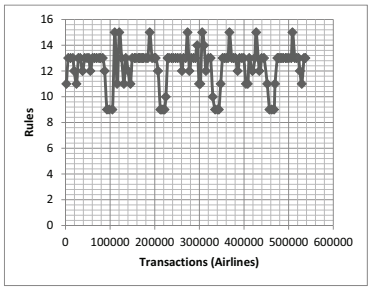

Figure 4. Airlines

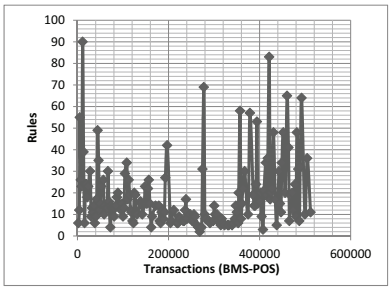

Figure 5. BMS-POS

generated were slightly different, but the underlying items remained fairly stable. This was mainly cause by the local change detector being triggered. Second we notice that the remaining dataset BMS-POS (in Figure 5) produced a more volatile number of rules at the different change points, this was triggered by global changes.

6. Conclusions

We proposed a novel technique to detect changes for an unsupervised environment called CD-TDS. We discuss the two different type of changes in transactional data stream. We tested on both synthetic and real datasets and showed that CD-TDS adapts its behaviour to both the structural characteristics of the problem and local changes in the data stream. We showed that our approach was effective at determining structural changes with a high true positive rate while maintaining a low false alarm rate even when faced with noise.

References

[1] C. K.-S. Leung, Q. I. Khan, Z. Li, and T. Hoque, "CanTree: a canonical-order tree for incremental frequent-pattern mining," *Knowl. Inf. Syst.*, vol. 11, pp. 287–311, April 2007.

[2] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Efficient frequent pattern mining over data streams," in *Proceedings of the 17th ACM conference on Information and knowledge management*, ser. CIKM '08. New York, NY, USA: ACM, 2008, pp. 1447–1448.

[3] Y. S. Koh, R. Pears, and G. Dobbie, "Extrapolation prefix tree for data stream mining using a landmark model," in *14th International Conference Data Warehousing and Knowledge Discovery*, 2012, pp. 340–351.

[4] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine learning*, vol. 23, no. 1, pp. 69–101, 1996.

[5] A. Tsymbal, "The problem of concept drift: definitions and related work," *Computer Science Department, Trinity College Dublin*, vol. 106, p. 2, 2004.

[6] R. Klinkenberg, "Learning drifting concepts: Example selection vs. example weighting," *Intelligent Data Analysis*, vol. 8, no. 3, pp. 281–300, 2004.

[7] B. Kurlej and M. Woźniak, *Learning curve in concept drift while using active learning paradigm*. Springer, 2011.

[8] G. R. Marrs, M. M. Black, and R. J. Hickey, "The use of time stamps in handling latency and concept drift in online learning," *Evolving Systems*, vol. 3, no. 4, pp. 203–220, 2012.

[9] A. Shaker and E. Lughofer, "Self-adaptive and local strategies for a smooth treatment of drifts in data streams," *Evolving Systems*, vol. 5, no. 4, pp. 239–257, 2014.

[10] R. Sebastiao and J. Gama, "A study on change detection methods," in *Proceedings of the 14th Portuguese Conference on Artificial Intelligence*, ser. EPIA 2009. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 353–364.

[11] J. Gama, P. Medas, G. Castillo, and P. P. Rodrigues, "Learning with drift detection," in *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence (Advances in Artificial Intelligence - SBIA 2004)*, ser. Lecture Notes in Computer Science, vol. 3171. Springer, 2004, pp. 286–295.

[12] A. Bifet and R. Gavaldà, "Learning from time-changing data with adaptive windowing," in *Proceedings of the Seventh SIAM International Conference on Data Mining*. SIAM, 2007.

[13] R. Pears, S. Sakthithasan, and Y. Koh, "Detecting concept change in dynamic data streams," *Machine Learning*, pp. 1–35, 2014.

[14] A. Balzanella and R. Verde, "Summarizing and detecting structural drifts from multiple data streams," in *Classification and Data Mining*, ser. Studies in Classification, Data Analysis, and Knowledge Organization, A. Giusti, G. Ritter, and M. Vichi, Eds. Springer Berlin Heidelberg, 2013, pp. 105–112.

[15] P. P. Rodrigues and J. Gama, "Online prediction of clustered streams," in *Proceedings of the Fourth International Workshop on Knowledge Discovery from Data Streams*, 2006, pp. 23–32.

[16] T. R. Hoens, R. Polikar, and N. V. Chawla, "Learning from streaming data with concept drift and imbalance: an overview," *Progress in Artificial Intelligence*, vol. 1, no. 1, pp. 89–101, 2012.

[17] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 44, 2014.

[18] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.

[19] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows: (extended abstract)," in *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '02, 2002, pp. 635–644.

[20] "IBM quest data mining project, Quest synthetic data generation code," http://www.almaden.ibm.com/cs/quest/syndata.html, 1996.

[21] B. Goethals and M. J. Zaki, "Frequent itemset mining implementations repository (FIMI)," http://fimi.ua.ac.be/, 2003.

[22] E. Ikonomovska, J. Gama, and S. Džeroski, "Learning model trees from evolving data streams," *Data mining and knowledge discovery*, vol. 23, no. 1, pp. 128–168, 2011.