High Utility Drift Detection in Quantitative Data Streams

Quang-Huy Duong[a], Heri Ramampiaro[a], Kjetil Nørvåg[a], Philippe Fournier-Viger[b], Thu-Lan Dam[a]

[a] *Department of Computer Science, Norwegian University of Science and Technology, Trondheim, Norway*
[b] *School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), Shenzhen, China*

Abstract

This paper presents an efficient algorithm for detecting changes (drifts) in the utility distributions of patterns, named High Utility Drift Detection in Transactional Data Stream (HUDD-TDS). The algorithm is specifically suitable for quantitative data streams, where each item has a unit profit, and non-binary purchase quantities are allowed. We propose a method that enables the HUDD-TDS algorithm to be used in an online setting to detect drifts. An important property of HUDD-TDS is that it can quickly adapt to changes in streams, while considering older transactions to be less important than new ones. Furthermore, the proposed method applies statistical testing based on Hoeffding bound with Bonferroni correction in order to ensure that only significant changes are reported to the user. This test allows identifying a change (drift) if the difference between current and the previous time window is significant in terms of utility distribution. In this work, we focus on both local and global utility drifts. A local utility drift is a drift in the utility distribution of a single pattern, whereas a global utility drift is a change in the utilities of all high utility itemsets. In order to be able to compute the similarity of different high utility itemsets to detect drifts, we propose a new distance measure function. The results of our experiments on both real world and synthetic datasets show the feasibility and efficiency of the proposed HUDD-TDS algorithm.

Keywords: High Utility Pattern Mining, Data Stream, Drift Detection, Change Detection

1. Introduction

Frequent Itemset Mining (FIM) is a fundamental research topic in data mining [1]. The task of FIM is to discover all itemsets in a transactional database so that the frequency of the itemsets is no less than a user-specified minimum support threshold. FIM has attracted a lot of attention from researchers and it has been applied in many applications [2, 3]. However, in FIM, the unit profits of items are not considered, and the purchase quantities are assumed to be binary in each transaction. This assumption often does not hold in real life. To address the limitation of these studies, the FIM problem has been generalized as the problem of High Utility Itemset Mining (HUIM) [4]. The goal of HUIM is to discover patterns that generate a high profit in static customer transaction databases. The key differences between HUIM and FIM are that each item has a unit profit, and non-binary purchase quantities are allowed. Several studies on HUIM have been conducted [5, 6], but most of existing approaches are suitable for pattern discovery in a static database. With more and more data, including customer transactions, being generated in streams, HUIM must also support pattern discovery in dynamic databases.

As with general data streams, streaming transactional data is generally infinite and changes continuously. This combined with the high data generation speed makes mining of a stream of transactional data to discover patterns more challenging than mining a static database. Thus, developing efficient methods and algorithms for analyzing transaction streams is an important research problem [7, 8]. Nevertheless, most studies on this topic, including [9, 10], have focused on adapting traditional data mining techniques to streams and

Email addresses: huydqyb@gmail.com (Quang-Huy Duong), heri@idi.ntnu.no (Heri Ramampiaro), noervaag@ntnu.no (Kjetil Nørvåg), philfv8@yahoo.com (Philippe Fournier-Viger), lanfict@gmail.com (Thu-Lan Dam)

improving their efficiency to deal with streaming data. Note, however, that the underlying distribution of data objects in a stream generally changes over time [11], thus making such approaches unsuitable. At the same time, detecting changes, called *concept drifts*, is crucial because it allows to discover the latest trends in a stream. A concept drift mainly refers to a significant decrease or increase in the distribution of data objects in a data stream with respect to a given measure [12].

In recent years, incremental and online learning have attracted the attention of many researchers to detect changes due to their numerous real-life applications, including market basket analysis, image processing, outlier detection, and climate monitoring [13, 14]. An important challenge of analyzing data streams is that trends may emerge, or remain steady over time, and that the streams often contain noise. In other words, to allow decision-makers to quickly react to changes, it is necessary to design efficient algorithms that can detect and monitor these changes in real-time. Nevertheless, although monitoring changes in data streams is widely recognized as important, most existing algorithms have mainly focused on discovering frequent patterns with changing frequencies, rather than considering changes in terms of other meaningful measures, such as the profit generated by the sale of items. To the best of our knowledge, only few approaches have been proposed to detect changes in the utility (profit) distribution of itemsets, where transactions are treated as streaming data. Monitoring such fluctuations in profit is necessary and important in many real-life applications including online retail stores and monitoring stock exchanges.

The work presented in this paper is motivated by the need to address the limitations due to the lack of approaches that fully study the issues with concept drifts in high utility itemsets in data streams. We propose an efficient algorithm called HUDD-TDS (High Utility Drift Detection in Transactional Data Streams), with which we introduce several novel ideas to detect drifts efficiently. We propose a new distance measure function to measure the similarity of different high utility itemsets. In order to quickly adapt to changes in streams, the HUDD-TDS considers weighting factor of older transactions and utilizes statistical testing based on Hoeffding bound with Bonferroni correction. The proposed method detects changes in the utility distribution of patterns in quantitative data streams, which include both changes in the utility distribution of single itemsets and changes in the structure of utility distribution of itemsets.

Overall, the main contributions of this work are as follows:

1. We introduce the task of detecting both local and global drifts by considering the utility measure and the recency of transactions in streams, defined as follows:
   - A *local utility drift* is a change in the utility distribution of an itemset (e.g., the utility of an itemset has recently considerably increased or decreased).
   - A *global utility drift* is a change in the total utility distribution of all itemsets (e.g., the sales of products in a retail store have globally considerably increased or decreased).
2. We propose an efficient algorithm for the drift detection task in 1). The proposed algorithm relies on probability theory and statistical testing to identify changes in the utility distribution of itemsets in a quantitative data stream. Moreover, our approach takes into account the evolving behavior of the streams, and we employ a fading function to identify recent trends. Such a fading function is specifically useful in weighing the importance of transactions according to their age.
3. We introduce a new distance measure function called *Dmo* to compare high utility itemsets for detecting drifts. Although *Dmo* is based on the cosine similarity, which is a standard measure for calculating the similarity between vectors, it is more general in that *Dmo* not only considers the difference of vectors in terms of orientation (i.e., vector angles) but also magnitude. As discussed in this paper, this is necessary to address our problem.
4. We conduct an extensive experiment to evaluate the proposed method HUDD-TDS, showing the feasibility, effectiveness, and efficiency of our HUDD-TDS algorithm.

The remainder of this paper is organized as follows. Section 2 briefly reviews the related work. Section 3 defines the problem of drift detection and introduces necessary preliminaries. Section 4 presents the proposed approach for detecting changes in the utility distribution of itemsets in a quantitative transaction data stream. Section 5 presents results from an extensive experimental evaluation to evaluate the performance of the proposed algorithm. Finally, Section 6 concludes the paper and outlines our plans for future work.

2. Related work

Detecting concept drifts is an important research problem that has applications in many domains such as flow prediction in industrial systems [15] and information filtering [16]. Numerous approaches have been proposed to detect changes in the distribution of data objects in data streams. Techniques for drift detection [17] are generally based on one of the following approaches: sequential analysis [18], statistical process control [19], comparison of two consecutive time windows [20], and contextual approaches [16]. The Hoeffding's Inequality has been used to design several approaches for determining the upper bounds for drift detection. Such upper bounds have been used in algorithms such as the Fast Hoeffding Drift Detection Method for Evolving Data Streams (FHDDM) [21], Hoeffding Adaptive Tree (HAT) [22], and the HAT+DDM+ADWIN [23] algorithm which extends ADaptive sliding WINdow (ADWIN) algorithm [24] and the Drift Detection Method (DDM) [18]. ADWIN is one of the most popular change detection, and it uses sliding windows to maintain the distribution and detect changes, whilst the DDM uses an online learning model to control the online error-rate and detect changes. Frías-Blanco et al. [25] proposed online and non-parametric drift detection methods using several bounds based on Hoeffding's Inequality. The algorithm can detect concept drifts based on the movements of distribution averages in streaming data. The algorithm uses counters to maintain information for detecting drifts. The time complexity of this approach is constant ($\mathcal{O}(1)$ for processing each data point in the stream).

In HUIM, several algorithms have been proposed for discovering high utility itemsets [4, 5, 26, 27] in static databases. Early HUIM algorithms adopt a two-phase approach to discover patterns in customer transaction databases. For example, Two-Phase [5], IHUP [26] and UPGrowth [28] are two-phase algorithms. Although a two-phase approach is useful and guarantees completeness in mining of high utility itemsets, a drawback is that the two-phase approach generates a huge amount of candidates, requiring a significant amount of memory to maintain the candidates. This greatly degrades the performance of the two-phase algorithms, thus making them unsuitable for streaming data. To address this issue, recent HUIM algorithms have adopted a one-phase approach using the utility-list structure. Liu et al. [29] first introduced and utilized this structure in the HUI-Miner algorithm. The utility-list structure can be used to mine high utility itemsets in a single phase, i.e., without maintaining candidates in memory. The utility of itemsets can be directly calculated using their utility-lists without scanning the database again. The simplicity of the utility-list structure has led to the development of numerous utility-list-based algorithms [29, 30, 31], which generally outperform other algorithms. To discover high-utility patterns in data streams, some algorithms have been proposed [10, 32, 33]. These studies generally extend traditional HUIM methods to increase their efficiency in a streaming context. Nevertheless, they are not designed to detect changes or drifts.

As mentioned earlier, customer transactions in retail stores can be seen as a stream of data, because customers continuously purchase products in the stores. This also means that the data is not static and is often impossible to store in memory due to its large volume. Moreover, in a streaming context, the distribution of data and the transactional behavior of customers can change and evolve over time. To the best of our knowledge, no work has been proposed to detect drifts for high utility itemset mining in streams of quantitative transactions, while considering the importance of utility over time. On the other hand, in traditional frequent itemset mining, several algorithms have been proposed to identify concept drifts in data streams [34, 35]. Ng et al. [34] proposed a test paradigm named Algorithm for change detection (ACD) for detecting changes in transactional data streams by considering the support of itemsets for reservoir sampling. ACD evaluates drifts by performing reservoir sampling and applying three statistical tests. The ACD method employs a bound based on Hoeffding's Inequality to determine the number of transactions to be kept in each reservoir. ACD selects transactions to fill its reservoir using a distance measure that is a function of the support of single items. A major limitation of this approach is that high frequency items have more chance of being sampled. However, in terms of utility (profit), high frequency items are often low utility itemsets in real-life customer transactions. Thus, this approach may find many patterns that are frequent but do not necessarily yield a high profit. Recently, Koh [35] proposed the CD-TDS algorithm, which considers two types of changes in transactional data streams for frequent pattern mining. A local drift, is a change in the frequency of single items, whilst a global change indicates a major change in the set of discovered frequent itemsets. The CD-TDS method uses a graph to represent the relationships between

items in transactions, and the Levenshtein distance to calculate the similarity between sets of frequent itemsets found at different times. A limitation of CD-TDS is that it detects local drifts for single items, and not for itemsets. CD-TDS consider itemsets to detect global drifts but do not provide information about which itemsets contribute the most to these global drifts. Moreover, CD-TDS considers drifts in terms of pattern frequencies, but it does not take into account changes in terms of utility. In addition, another limitation of the CD-TDS algorithm is that it treats all transactions as equally important. However, in real-life data streams, the most recent transactions are generally the most important transactions, as they provide information about recent trends.

In conclusion, there is an important need for an efficient method for online detection of utility drifts of high utility itemsets in a stream of quantitative customer transactions, which considers changes in both the utility of a high utility itemset (local drift) and the utility distribution of high utility itemsets (global drift), while also considering the recency of transactions. In the next sections, we formalize the problem and present the proposed method in details.

3. Preliminaries and problem definition

This section introduces preliminaries related to high utility itemset mining and drift detection. In the following paragraphs, we present some basic definitions that we will use in this paper. The notations used in this paper are summarized in Table 1.

Let there be a set of items $I = \{i_1, i_2, \ldots, i_m\}$ representing products sold in a retail store. For each item $i_j \in I$, the external utility of i_j is a positive number representing its unit profit (or more generally, its relative importance to the user). The external utility of an item i_j is denoted as $p(i_j)$. Let there be an infinite sequence of increasing positive integers $1 \leq x1 < x2 < x3 \ldots$. A streaming quantitative transactional database D is an infinite sequence of transactions $D = \{T_{x1}, T_{x2}, T_{x3}, \ldots\}$, where for each transaction $T_d \in D$, the relationship $T_d \in I$ holds. Moreover, for each transaction $T_d \in D$, d is a unique integer that is said to be the TID (Transaction IDentifier) of T_d, and represents its observation time. Thus, for two transactions T_a and T_b, if $a < b$, this indicates that transaction T_a has occurred before T_b. Assume the stream D does not contain two transactions with the same observation time[1]. Consider two observation times a and b, a data window W_{ab} is the finite sub-sequence of D containing all transactions from the observation time a to the observation time b. Formally, $W_{ab} = \{T_{y1}, T_{y2}, \ldots, T_{yn}\}$, for all integers $y1$, $y2$, $\ldots$, yn such that $a \leq y1 < y2 < \cdots < yn \leq b$, and T_{y1}, T_{y2}, $\ldots$, T_{yn} appear in D. The internal utility of an item i_j in a transaction T_d is denoted as $q(i_j, T_d)$. It is a positive number representing the purchase quantity of item i_j in T_d. A set of items $X = \{i_1, i_2, \ldots, i_l\} \subseteq I$ containing l items is said to be an itemset of length l, or alternatively, an l-itemset. For example, Figure 1 shows the first four transactions of a streaming quantitative transactional data stream, which will be used as running example. In this stream, the set of items I in D is $\{a, b, c, d, e, g\}$. The external utilities of these items are respectively 5, 2, 1, 2, 3, and 1.

Definition 1 (Utility of an item in a transaction)**.** Let there be an item i and a transaction T_d such that $i \in T_d$. The utility of i in T_d is the product of the internal utility (purchase quantity) of item i in T_d by the external utility (unit profit) of i, that is $u(i, T_d) = q(i, T_d) \times p(i)$.

Definition 2 (Utility of an itemset in a transaction)**.** For an itemset X and a transaction T_d ($X \subseteq T_d$), the utility of X in T_d is a positive number defined as $u(X, T_d) = \sum_{i \in X} u(i, T_d)$.

Definition 3 (Transaction utility and total utility)**.** The utility of a transaction T_d is the sum of the utilities of items appearing in that transaction, that is $TU(T_d) = u(T_d, T_d)$. The total utility of a window W in a stream D is the sum of the utilities of all transactions in W, that is $TUD(W) = \sum_{T_d \in W} TU(T_d)$.

[1] If two transactions are simultaneous, a total order on these transactions can be obtained by incrementing the observation time of one of those transactions by a small value.

TID	Transaction	Transaction utility
1	(a,1), (c,1), (d,1)	8
2	(a,2), (c,6), (e,2), (g,5)	27
3	(b,4), (c,3), (d,3), (e,1)	20
4	(b,2), (c,3), (e,2), (g,2)	15

Item	a	b	c	d	e	g
External utility	5	2	1	2	3	1

Figure 1: Four transactions of a quantitative transactional stream (top) and the corresponding external utilities of items (bottom).

Definition 4 (Utility and relative utility of an itemset)**.** Let there be a window W in a stream D and an itemset X. The utility of X in W is defined as $u(X,W) = \sum_{X \subseteq T_d \wedge T_d \in W} u(X,T_d)$. The relative utility of X in W is defined as $ru(X,W) = u(X,W)/TUD(W)$.

Definition 5 (Low utility itemset and high utility itemset)**.** Let the minimum utility threshold (abbreviated as *minutil*) be a positive number specified by the user such that $0 < minutil$. Consider an itemset X. It is said to be a *high utility itemset* (HUI) in W if its utility is no less than *minutil*, $u(X,W) \geq minutil$. Otherwise, X is said to be a *low utility itemset* in W.

Example 1. Consider the stream of Figure 1. The utility of itemset c in transaction T_1 is $u(c,T_1) = 1 \times 1 = 1$. The utility of itemset ac in T_1 is $u(ac,T_1) = u(a,T_1) + u(c,T_1) = 1 \times 5 + 1 \times 1 = 5 + 1 = 6$. The utility of transaction T_1 is $TU(T_1) = u(a,T_1) + u(c,T_1) + u(d,T_1) = 5 + 1 + 2 = 8$. Consider the window $W = \{T_1; T_2\}$ and that *minutil* is set to 22. The set of high utility itemsets in that window W is {*ac*:22, *ace*:22}, where the number beside each itemset indicates its utility.

Detecting drifts [17] in a stream can be done based on the following definitions. Generally, let there be a stream S that is a sequence of values $\{u_1; u_2; \ldots ; u_t; u_{t+1}; \ldots ; u_n\}$. Let μ_1 and μ_2 respectively denote the population means of two samples of instances U_1 and U_2, where $U_1 = \{u_1; u_2; \ldots; u_t\}$ and $U_2 = \{u_{t+1}; \ldots; u_n\}$. Detecting drifts using a statistical test can be done by comparing two hypotheses. The null hypothesis is that the population means of the two samples have the same distribution, that is H_0: $\mu_1 = \mu_2$. The alternative hypothesis H_1 is that $\mu_1 \neq \mu_2$. A statistical test is then applied to determine if the null hypothesis holds for a significance level α. The rule to accept the H_1 hypothesis is $Pr(|\mu_1 - \mu_2| \geq \varepsilon) \geq \alpha$, where ε is a user-defined positive number. A drift is said to occur at an observation time t if the population of the sample at time t is significantly different from that of the preceding observation time. In that case, t is said to be a drift point.

The problem of detecting drifts is to find the observation times where there are significant differences in the data distribution for a given measure (e.g. the support) with respect to the preceding observation times. Although several papers, e.g., [25, 35] have proposed approaches for drift detection, none have considered detecting drifts in the utility distribution of patterns including the utility distribution of items in patterns as well as the utility distribution of patterns in the whole set of patterns. However, the utility is a more useful measure compared to the support measure as it measures the profit generated by patterns, rather than simply measuring the number of transactions containing the items without considering their purchase quantities.

To allow discovering more useful patterns and drifts, this paper adapts the concept of drift to the utility measure to propose the problem of drift detection for high utility itemset mining in an evolving data stream. It consists of finding all observation times where there are significant differences in the utility distribution of itemsets, while also considering the recency of transactions. Finding such drifts provides information that allows decision-makers to quickly react to changes in customer behavior that influence profitability.

Table 1: Table of notations

Symbols	Description	Symbols	Description
D	Quantitative database for transaction stream	T_d	A specific transaction in D having transaction IDentifier d
S	Sequence of transactions	I	Set of items in the database D
X	An itemset	$p(i)$	External utility of single item i
$q(i, T_d)$	Internal utility of an item i in a transaction T_d	W_{ab}	A data window containing transactions from the observation time a to the observation time b
$u(X, T_d)$	Utility of itemset X in transaction T_d	$TUD(W)$	Total utility of window W in a stream D
$u(X, W)$	Utility of itemset X in window W	μ	Population mean
H	Hypothesis	u_t	Value in the stream at the observation time t
λ	Decay factor	$d^{\lambda}(t)$	Decay function of factor λ and time t
$U(T_i, t, \lambda)$	Utility of transaction T_i in the stream at observation time t	$U_S(X, T_i, t, \lambda)$	Utility of itemset X in transaction T_i at the observation time t
$HS_W^{n,\lambda}$	Set of all high utility itemsets in W of stream at observation time n	$\overline{U}$	Average of the sequence variables U_i
$\|W\|$	Cardinality: The number of members in W	α	Confidence level of hypothesis
ε	Error value	σ^2	Variance of utility in a window
Pr	Probability	E	Expectation value of variable
$\overrightarrow{Root_S}$	*Root* vector of stream S	$\overrightarrow{X}$	Vector of itemset X
$\|\overrightarrow{X}\|$	Euclidean norm, or Euclidean length, or magnitude of vector $\overrightarrow{X}$	$S_{cos}(\overrightarrow{x}, \overrightarrow{y})$	Cosine similarity between $\overrightarrow{x}$ and $\overrightarrow{y}$
$S_{mo}(\overrightarrow{x}, \overrightarrow{y})$	Similarity between $\overrightarrow{x}$ and $\overrightarrow{y}$	$D_{mo}(\overrightarrow{x}, \overrightarrow{y})$	Distance between $\overrightarrow{x}$ and $\overrightarrow{y}$
DIS_{HS}	Sum of the movements of high utility itemsets in the set HS to the root vector		

4. High utility drift detection algorithm in transactional data stream

This section introduces the High Utility Drift Detection in Transactional Data Stream (HUDD-TDS) algorithm to detect changes in the utility distribution of high utility itemsets in a stream of quantitative customer transactions. HUDD-TDS can detect both local and global changes by considering the utility measure, the recency of transactions, and the correlative conjunction of the utility distributions of itemsets in streams. It gives most importance to the most recent transactions since in practice, i.e., customer transaction data streams, recent trends are considered as the most valuable. In particular, the proposed approach uses a fading function to assign a weight to each transaction that is inversely proportional to its age. The next subsections describes the proposed approach in details.

4.1. A fading function for high utility itemset mining in a stream

An important characteristic of customer transaction data streams is that trends found in a stream change with time, as the behaviors of customers vary. In a data stream, data points arrive at a high speed. As previously explained, the importance of data points (transactions) can be viewed as inversely proportional to their age. Hence, a transaction that occurred a long time ago (before the current observation time) should be considered as less important than a recent transaction. To model the varying importance of transactions with respect to the observation time, a decay (fading) function is used in the proposed algorithm. At the time of observation, the utility values in a transaction that occurred at a time t are multiplied by a decay

factor calculated by a decay function $d^{\lambda}(t)$. The calculated decay factor is a value $d^{\lambda}(t) \in [0,1]$. The decay function $d^{\lambda}(t)$ is a user-defined function that is inversely proportional to the elapsed time. The decay function takes a positive constant λ as parameter, called the decay constant, which let the user indicate how fast the importance of transactions should decrease with respect to time.

Definition 6 (Utility of a transaction in a stream)**.** Let $S = (T_{x1}, T_{x2}, \ldots, T_{xn})$ be a sequence of transactions, where the notation T_i denotes the transaction that occurred at time i. At the time of observation t, the utility of a transaction T_i in the stream S is denoted as $U(T_i, t, \lambda)$ and defined as: $U(T_i, t, \lambda) = U(T_i, i) * d^{\lambda}(\triangle T)$, where $\triangle T = t - i \geq 0$ and $U(T_i, i)$ is the utility of transaction T_i at time i, it is equal to $TU(T_i)$.

Definition 7 (A fading function to consider the recency of transactions)**.** Let $S = (T_{x1}, T_{x2}, \ldots, T_{xn})$ be a sequence of transactions, and $d^{\lambda}(T)$ be the user-defined decay function. The utility of an itemset X in a transaction T_i at the observation time $t \geq i$ is denoted as $U_S(X, T_i, t, \lambda)$, and defined as $U_S(X, T_i, t, \lambda) = U_S(X, T_i, i, \lambda) \times d^{\lambda}(\triangle T) = u(X, T_i) \times d^{\lambda}(\triangle T)$, where $U_S(X, T_i, i, \lambda)$ is the utility of itemset X in T_i at observation time i, and $u(X, T_i)$ is the utility of X in T_i as defined in traditional high utility mining (without applying the decay function).

Example 2. Consider the stream database of Figure 1. Suppose that the decay function is $d^{\lambda}(\triangle T) = 2^{\frac{-\triangle T}{2}}$. The utility of itemset ac in T_1 at observation time $t = 2$ is $U_S(ac, T_1, 2, \lambda) = 6 \times 2^{\frac{-1}{2}} = 4.24$. The utility of itemset ac in T_1 at time $t = 3$ is $U_S(ac, T_1, 3, \lambda) = 6 \times 2^{\frac{-2}{2}} = 3$. Consider the window $W = \{T_1; T_2\}$, and that *minutil* is set to 22. The set of high utility itemsets in W when considering transaction recency is $\{ace{:}22\}$. The itemset ac is not high utility because its utility is $U_S(ac, T_1, 2, \lambda) + U_S(ac, T_2, 2, \lambda) = 4.24 + 16 = 20.24 <$ *minutil*.

There are two important differences between mining high utility itemsets in a stream and in a static transaction database. First, not all data points from a stream can be stored and kept in memory due to the limited amount of memory of a computer and the infinite nature of a stream. For this reason, data points can only be read once. Second, an important issue is that the utility of a transaction should decrease according to a decay function that is inversely proportional to its age. To consider the decay function when mining high utility itemsets in a quantitative transactional data stream, we redefine the problem of HUIM as follows. Let $W = (T_{y1}, T_{y2}, \ldots, T_{ym})$ be a window containing m transactions at the observation time ym of a sequence of transactions S. Let there be a threshold value θ defined by the user. An itemset X is a high utility itemset in W if the sum of its utilities in W is not less than θ, that is: $\sum_{T_i \in W}(U_S(X, T_i, ym, \lambda)) \geq \theta$. In the following, the notation $HS_W^{n,\lambda}$ is used to denote the set of all high utility itemsets found in a window W of a transactional data stream at observation time n. Formally, $HS_W^{n,\lambda} = \{X, \sum_{T_i \in W}(U_S(X, T_i, n, \lambda)) \geq \theta\}$.

4.2. A Hoeffding bound to assess the significance of drifts

Having explained how we apply fading, this section proposes a bound to detect utility drifts. The proposed bound is based on the Hoeffding Inequality [36] from the probability theory. This inequality has been used in various studies to analyze data streams [21, 24, 25, 35]. Given some independent random variables, the Hoeffding Inequality provides an upper bound on the probability that their sum deviates from its expected value. In this work, the Hoeffding Inequality is used as the basis for assessing if changes are statically significant in a flow of independent random transactions arriving in a data stream. If the probability of a predefined condition is greater than a user-specified threshold, the proposed approach considers that there is a change in the data. The Hoeffding's Inequality theorem states as follows [36].

Theorem 1 (Hoeffding's Inequality)**.** Let $U_1; U_2; \ldots ; U_n$ be independent random variables bounded by the interval [0, 1], that is $0 \leq U_i \leq 1$, where $i \in \{1; \ldots; n\}$. Let $\overline{U}$ denote the average of the random variables, that is $\overline{U} = \frac{1}{n}\sum_{i=1}^{n}(U_i)$. We have:

$$Pr(\overline{U} - E[\overline{U}] \geq \varepsilon) \leq e^{-2n\varepsilon^2}, \tag{1}$$

where E[X] is the expected value of X.

The inequality states that the probability that the estimation and true values differ by more than ε is bounded by $e^{-2n\varepsilon^2}$. Symmetrically, the inequality is also valid for the other side of the difference: $Pr(-\overline{U} + E[\overline{U}] \geq \varepsilon) \leq e^{-2n\varepsilon^2}$. As a result, a two-sided variant of the Inequality is obtained:

$$Pr(|\overline{U} - E[\overline{U}]| \geq \varepsilon) \leq 2e^{-2n\varepsilon^2} \tag{2}$$

This Inequality is true if all variables are bounded by the [0, 1] interval. More generally, if a variable U_i is bounded by an interval $[x_i, y_i]$, the Hoeffding's Inequality is generalized as follows:

$$Pr(|\overline{U} - E[\overline{U}]| \geq \varepsilon) \leq 2e^{\frac{-2n^2\varepsilon^2}{\sum_{i=1}^{n}(y_i - x_i)^2}} \tag{3}$$

The Hoeffding's Inequality (Theorem 1) can be used to assess the significance of changes in a stream of values, based on the following proposition.

Proposition 1. Let $U_1; U_2; \ldots; U_n$ be independent random variables bounded by the [0, 1] interval. These variables can be split into two windows using an index m as splitting point. This results in two windows, $W_1 = \{U_1; \ldots; U_m\}$ and $W_2 = \{U_{m+1}; \ldots; U_n\}$, such that $1 \leq m < n$. Then, for an error $\varepsilon > 0$, the following inequality holds:

$$Pr(\overline{U} - \overline{V} - (E[\overline{U}] - E[\overline{V}]) \geq \varepsilon) \leq e^{\frac{-2\varepsilon^2|W_1|.|W_2|}{|W_1|+|W_2|}}, \tag{4}$$

where $|W_1|$ and $|W_2|$ are the size of W_1 and W_2, respectively. Moreover, $\overline{U} = \frac{1}{|W_1|}\sum_{i=1}^{m}(U_i)$ and $\overline{V} = \frac{1}{|W_2|}\sum_{i=m+1}^{n}(U_i)$.

If the two-sided variant of the inequality is considered:

$$Pr(|(\overline{U} - E[\overline{U}]) - (\overline{V} - E[\overline{V}])| \geq \varepsilon) \leq 2e^{\frac{-2\varepsilon^2|W_1|.|W_2|}{|W_1|+|W_2|}} \tag{5}$$

Based on proposition 1, consider a significant confidence level α (probability of making an error), that controls the maximum false positive rate. The error ε can be estimated with respect to α as follows.

$$\begin{aligned} \alpha &= 2e^{\frac{-2\varepsilon^2|W_1|.|W_2|}{|W_1|+|W_2|}} \Rightarrow \frac{2\varepsilon^2|W_1|.|W_2|}{|W_1|+|W_2|} = ln[\frac{2}{\alpha}] \\ \Rightarrow \varepsilon &= \sqrt{\frac{|W_1|+|W_2|}{2|W_1|.|W_2|}ln[\frac{2}{\alpha}]} \end{aligned} \tag{6}$$

To assess the significance of changes in a stream of values, we use Equation 6 to obtain the cut point value ε_α based on the predefined α threshold. An observation time m is said to be a *distribution change point* if there is a significant difference between the averages of values in the windows W_1 and W_2, that is the difference is not less than ε_α. Furthermore, if that condition holds, we also say that there is a change and that the windows W_1 and W_2 are different. This can be expressed as a statistical test with bounding probability of errors. Let the null hypothesis be H_0: $(\overline{U} - E[\overline{U}]) = (\overline{V} - E[\overline{V}])$, stating that the distributions of the two windows are equal (assuming independent random variables). The H_0 hypothesis is compared with the alternative hypothesis H_1: $(\overline{U} - E[\overline{U}]) \neq (\overline{V} - E[\overline{V}])$ with the rule $|(\overline{U} - E[\overline{U}]) - (\overline{V} - E[\overline{V}])| \geq \varepsilon$ to reject H_0. HDDM [25] proposed a minor improvement of the Hoeffding's Inequality by using two counters instead of three counters for maintaining the left, right and the total mean at the cut point. The proposed method inherits this improvement to detect a global drift. This improvement is derived from the Hoeffding's Inequality as follows.

Proposition 2. Let $U_1; U_2; ...; U_n$ be independent random variables bounded by the [0, 1] interval. These variables can be split into two windows at an index m to obtain $W_1 = \{U_1; \ldots; U_m\}$ and $W_2 = \{U_{m+1}; \ldots; U_n\}$, such that $1 \leq m < n$. Then, for an error $\varepsilon > 0$, the following inequality is obtained:

$$Pr(\overline{U} - \overline{V} - (E[\overline{U}] - E[\overline{V}]) \geq \varepsilon) \leq e^{\frac{-2nm\varepsilon^2}{(n-m)}}, \tag{7}$$

where $\overline{U} = \frac{1}{m}\sum_{i=1}^{m}(U_i)$ and $\overline{V} = \frac{1}{n}\sum_{i=1}^{n}(U_i)$.

The estimated error ε with respect to α is:

$$\begin{aligned} \alpha &= 2e^{\frac{2nm\varepsilon^2}{(n-m)}} \Rightarrow \frac{-2nm\varepsilon^2}{(n-m)} = ln[\frac{2}{\alpha}] \\ \Rightarrow \varepsilon &= \sqrt{\frac{n-m}{2nm} ln[\frac{2}{\alpha}]} \end{aligned} \tag{8}$$

4.3. Two mechanisms to detect utility changes

This subsection presents the proposed approach to detect changes in the distribution of high utility itemsets in a stream of customer transactions. In general, change detection consists of detecting each observation time where there is a significant difference in the distribution of the data. In this paper, the object of change analysis is the sets of high utility itemsets mined from consecutive windows of transactions in a stream. As mentioned above, two kinds of changes are considered in this paper: local and global utility changes. A local utility change is a drift in the utility distribution of a high utility itemset. A global utility change is a drift in the overall utility distribution of all high utility itemsets. The next paragraphs describe mechanisms to detect these two types of changes.

4.3.1. Local Utility Change Detection

Let $HS^{x,\lambda}_{W_1}$ and $HS^{y,\lambda}_{W_2}$ be two sets of high utility itemsets mined at two different observation times in a transactional data stream. Local utility change detection consists of comparing the utility of each high utility itemset X in $HS^{x,\lambda}_{W_1}$ and $HS^{y,\lambda}_{W_2}$. For an itemset X, if there is a significant difference in terms of utility for the two observation times, it is called a local drift of X at the change point from $HS^{x,\lambda}_{W_1}$ to $HS^{y,\lambda}_{W_2}$. The theoretical background of our method is probability theory, statistical testing and the Hoeffding's Inequality. To detect a drift, existing methods such as CD-TDS [35] and ADWIN [24] apply the Bonferroni correction with the Hoeffding's Inequality. The reason is that the Bonferroni correction prevents an increase of the probability of incorrectly rejecting the null hypothesis when testing multiple hypotheses.

In this paper, the proposed approach applies the Bonferroni correction with the Hoeffding's Inequality to detect local drifts for high utility itemsets in two consecutive windows. For the two sets of high utility itemsets $HS^{x,\lambda}_{W_1}$ and $HS^{y,\lambda}_{W_2}$, let n_1 and n_2 be the number of sampled transactions from the stream that were used to obtain the sets $HS^{x,\lambda}_{W_1}$ and $HS^{y,\lambda}_{W_2}$ at their respective observation times. For the sake of brevity, the notation HS_1 and HS_2 will be used to refer to these sets in the following. There is a local change for an itemset X from HS_1 to HS_2 if its utility distribution difference in HS_1 and HS_2 is not less than a cut value:

$$\begin{aligned} \varepsilon_\alpha &= \sqrt{\frac{2(n_1+n_2)}{n_1 n_2}\sigma^2 ln[\frac{2ln(n_1+n_2)}{\alpha}]} \\ &\quad + \frac{2(n_1+n_2)}{3n_1n_2} ln[\frac{2(n_1+n_2)}{\alpha}] \\ &= \sqrt{2m\sigma^2 ln[\frac{2ln(n)}{\alpha}]} + \frac{2m}{3} ln[\frac{2ln(n)}{\alpha}], \end{aligned} \tag{9}$$

where $n = n_1 + n_2$, $m = n_1^{-1} + n_2^{-1}$, α is a user-defined confidence level, and σ^2 is the observed variance of the utility of the itemset in the window W formed by joining W_1 and W_2.

The variance σ^2 is defined and computed as the sum of the squared distances of each sample in the distribution from the mean, divided by the number of samples in the distribution. In the proposed algorithm, an efficient way of calculating the standard deviation for a set of numbers is proposed and as follows.

$$\sigma^2 = \frac{1}{N}\sum X^2 - (\frac{\sum X}{N})^2 \tag{10}$$

4.3.2. Global Utility Change Detection

Global utility change detection aims at detecting changes by comparing the utility distributions of high utility itemsets mined in two consecutive windows. For each itemset X, the utility of X is a function of the utility distributions of the items that it contains. Let $I = \{i_1, i_2, \ldots, i_n\}$ be the set of all items that appear in a customer transaction data stream. A high utility k-itemset X can be represented as $(\{i_{j1} : u_{j1}\}, \{i_{j2} : u_{j2}\}, \ldots, \{i_{jk} : u_{jk}\})$, where the notation i_{jx} represents an item in X, and the number beside each item indicates the utility contributed by that item to the utility of X. Because of differences in the utility distributions of items in high utility itemsets, this paper proposes a custom distance measure to efficiently detect drifts for high utility itemsets. The distance of each itemset to a reference point is calculated. Then, the distance of a set of high utility itemsets to the reference point is calculated as the sum of the distances of high utility itemsets to that point. The proposed measure is inspired by the observation that an itemset X with its utility can be presented as a vector, and that the distance between two itemsets can thus be calculated using vector distance measures. Formally, the distance between high utility itemsets is computed based on the following definitions.

Definition 8 (Root vector)**.** Let $S = (T_1, T_2, \ldots, T_n, \ldots)$ be a sequence of transactions, and $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n items in S, such that $i_j \prec i_{j+1}$ with $1 \leq j < n$ and $\prec$ be any total order on items from I. The Root vector in S is denoted as $\overrightarrow{Root_S} = \overrightarrow{i_1 i_2 \ldots i_n} = \{1, 1, \ldots, 1\}$. It is a vector described with n properties, and the value of each property is equal to 1 unit.

Definition 9 (Vector of a high utility itemset)**.** Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of items that appear in a set of transactions. Moreover, let there be a high utility k-itemset $X = \{i_{j1} i_{j2} \ldots i_{jk}\}$ such that the utility distribution of its items is $(\{i_{j1} : u_{j1}\}, \{i_{j2} : u_{j2}\}, \ldots, \{i_{jk} : u_{jk}\})$. The vector of the high utility itemset X is denoted as $\overrightarrow{X}$ and defined as $\overrightarrow{X} = \{U_1, U_2, \ldots, U_n\}$, where

$$U_j = \begin{cases} 0, & \text{if } i_j \notin X \\ u(i_j), & \text{otherwise.} \end{cases} \tag{11}$$

Distance. The first step for calculating the distance in the proposed approach is to calculate the similarity between two vectors using the cosine similarity. Consider two vectors for some high utility itemsets X and Y, defined as $\overrightarrow{X} = \{X_1, X_2, \ldots, X_n\}$, $\overrightarrow{Y} = \{Y_1, Y_2, \ldots, Y_n\}$, respectively. The cosine similarity between $\overrightarrow{X}$ and $\overrightarrow{Y}$ is:

$$\begin{aligned} S_{cos}(\overrightarrow{X}, \overrightarrow{Y}) &= \frac{\overrightarrow{X}.\overrightarrow{Y}}{\|\overrightarrow{X}\|.\|\overrightarrow{Y}\|} \\ &= \frac{\sum_{i=1}^{n}(X_i.Y_i)}{\sqrt{\sum_{i=1}^{n} {X_i}^2}\sqrt{\sum_{i=1}^{n} {Y_i}^2}} \end{aligned} \tag{12}$$

The cosine similarity is a standard measure for calculating the similarity between vectors. However, a drawback of this measure is that it only considers the difference in orientation of two vectors, while ignoring their difference in terms of magnitude [37]. For example, a cosine similarity of 1 indicates that two vectors have the same orientation. But these vectors may or may not have the same magnitude. Meanwhile, magnitude of an itemset vector represents its utility. To consider not only the difference in terms of orientation but also in terms of magnitude, the proposed approach calculates the similarity between two vectors $\overrightarrow{X}$ and $\overrightarrow{Y}$ using an improved similarity measure denoted as $S_{mo}(\overrightarrow{X}, \overrightarrow{Y})$, and defined as follows.

$$S_{mo}(\overrightarrow{X}, \overrightarrow{Y}) = S_{cos}(\overrightarrow{X}, \overrightarrow{Y}).(1 - \frac{abs(\|\overrightarrow{X}\| - \|\overrightarrow{Y}\|)}{max(\|\overrightarrow{X}\|, \|\overrightarrow{Y}\|)} \tag{13}$$

Similarly to the cosine similarity, the proposed S_{mo} similarity measure assigns a value of 1 to two vectors having the same orientation and magnitude. However, in the proposed approach, the distance between

vectors must be calculated rather than the similarity. Thus, based on S_{mo}, a distance measure D_{mo} is defined as: $D_{mo}(\overrightarrow{X}, \overrightarrow{Y}) = 1 - S_{mo}(\overrightarrow{X}, \overrightarrow{Y})$. In the following, this measure is called the *movement* between two vectors. To check if there is a global drift between two sets of high utility itemsets HS_1 and HS_2, the proposed approach computes the sum of the movements of high utility itemsets in the two sets.

$$DIS_{HS} = \sum_{X \subseteq HS} D_{mo}(\overrightarrow{X}, \overrightarrow{Root}) \tag{14}$$

Example 3. Consider the stream of Figure 1 and the decay function $d^{\lambda}(\triangle T) = 2^{\frac{-\triangle T}{2}}$. Furthermore, suppose that *minutil* is set to 22 and that the window size is set to 2. Consider the windows $W_1 = \{T_1; T_2\}$ and $W_2 = \{T_3; T_4\}$. By taking transaction recency into account, the set of high utility itemsets in W_1 is $\{ace{:}22\}$. The vector of the itemset *ace* is $\overrightarrow{ace} = \{10, 0, 6, 0, 6, 0\}$. The set of high utility itemsets in W_2 is $\{bce{:}22.9\}$. The utility of itemset *bce* is computed as the sum of its utilities in T_3 and T_4, that is $13 + 7 \times \sqrt{2} = 22.9$. The vector of the itemset *bce* is $\overrightarrow{bce} = \{0, 9.66, 5.12, 0, 8.12, 0\}$. The distance of the vector of itemset *ace* to the *root* vector is computed as $D_{mo}(\overrightarrow{ace}, \overrightarrow{root}) = 1 - S_{mo}(\overrightarrow{ace}, \overrightarrow{root}) = 1 - 0.128 = 0.872$, where $S_{mo}(\overrightarrow{ace}, \overrightarrow{root})$ is computed by Equation 13. In a similar way, the distance of the vector of itemset *bce* to the *root* vector is computed as $D_{mo}(\overrightarrow{bce}, \overrightarrow{root}) = 1 - 0.1234 = 0.8766$.

The *total distribution* is the sum of the distances of all high utility itemset vectors to the root vector. After that, the algorithm uses a statistical test (the A-test [25]) with the designed Hoeffding bound to determine if a global drift occurred at the current observation time. Note that it is also possible to detect drifts that represent increasing or decreasing trends by using the one-side or two-side variant of the Hoeffding's Inequality. We also consider increasing or decreasing trends and report it in Evaluation section of this paper.

4.4. The change detection algorithm

In the proposed approach, high utility itemsets are obtained at different observation times using windows on the stream. A set of high utility itemsets found at a given time is eventually replaced by a newer set, as time passes. Another important characteristic of a data stream is that data points arrive at a very high speed. Thus, an algorithm would be inefficient if it checks for changes for each new data point in the stream. The solution to this problem is to let the user set a parameter that determines the frequency of checks. On one hand, frequently checking for changes decreases the efficiency of the algorithm. On the other hand, rarely checking for changes results in higher efficiency but increases the risk of missing drift points as the windows may be too large. In the proposed method HUDD-TDS (High Utility Drift Detection in Transactional Data Stream), the checking frequency is a time interval length, and it is set by the user. The detailed pseudocode of the proposed method is presented in Algorithm 1.

The Algorithm 1 takes a stream of quantitative customer transactions as input. When a new transaction T_i is read from the stream, the transaction is temporarily stored in the limited amount of available memory (line 6). If the memory is full then the oldest transaction(s) are removed to free space for new transaction(s) T_i (line 4). To detect drift, the user must indicate the time interval length at which the algorithm should check for drifts (line 7). When the drift detection algorithm is called, it applies a procedure named HUI-Discovery (Algorithm 2) to mine all high utility itemsets in the window ending at the current observation time (line 8). Then, the similarity and distance of itemsets and the global distance are calculated (line 9). Thereafter, global and local checks are performed (lines 10 and 14) to detect significant changes in the utility distribution.

The HUI-Discovery procedure (Algorithm 2) mines high utility itemsets in a stream. The procedure first creates the window W ending at the current observation time t (line 3). Then, for each transaction in the window W, the procedure multiplies the utilities by the decay factor calculated by the fading function. This decreases the utility of items in the transaction as a function of its age. Then, the HUI-Discovery procedure applies a traditional HUI mining algorithm to extract each HUI in the current window (lines 12 to 15). For each itemset found (line 12), the utility distribution of each item is calculated to construct the itemset's vector and then calculate its distance to the root vector (line 13).

Algorithm 1: HUDD-TDS: High Utility Drift Detection in a Utility Data Stream

Input: Stream of transactions with utility.
Output: Utility changes in the stream.

Init() Initialize all variables: *minutil*, *interval*, *window size*, *confidence*, etc.
for *(each transaction* T_i *arriving on a stream* $T_{x1}; T_{x2}; \ldots ; T_{xi}; \ldots)$ **do**
  if *(Memory is full)* **then**
    Remove the oldest transactions from memory
  end
  Add T_i to the limited memory
  if *(Observation time i % check_interval = 0)* **then**
    checkpoints[*i*].*HUIs* ⟵ **HUI-Discovery**(i) using utility-list based mining method
    checkpoints[*i*].*distance* = sum of the distance of each itemset X in *checkpoints*[*i*].*HUIs*
    if *(IsGlobalDrift(checkpoints))* **then**
      Update last index where a drift is detected
      Output global drift
    end
    if *(IsLocalDrift(checkpoints))* **then**
      Output local drift
    end
  end
end

Algorithm 2: HUI-Discovery: Mine all high utility itemsets

Input: Observation time t and transactions.
Output: Set of high utility itemsets.

Init() Initialize all variables
HUIsSet ← ϕ
Scan transactions in memory to create a window W of a predefined length, ending at observation time t
for *(each transaction* T *in* W*)* **do**
  if *(is fading)* **then**
    T.utility = *T.utility* × *decay* function d^{λ}
    for *(each item it in* T*)* **do**
      update the utility of *it* in the transaction by multiplying it with the *decay* function d^{λ}
    end
  end
end
while *(Found high utility itemset X)* **do**
  $X.distance = Distance(\overrightarrow{X}, \overrightarrow{Root})$
  HUIsSet.add(X)
end
Return *HUIsSet*

The IsGlobalDrift procedure (Algorithm 3) detects global changes in a stream of values. It is based on probability theory, statistical testing, and the Hoeffding's Inequality. Lines 6-8 detect a cut point for an increasing trend of values, while lines 9-11 detect a cut point for a decreasing trend of values. If a change in the data distribution rejects the null hypothesis H_0 at line 12, a drift is said to occur at the cut point, and it is reported to the user (line 13).

Algorithm 3: IsGlobalDrift: Check if there is a global change in a utility stream

Input: List of distance values at checkpoints starting from the last change index: d_1; d_2; ... ; d_n.
Output: State with trend.

$\overline{U}_{drift}$: average statistic computed from d_1; d_2; ... ; d_{drift}
$\overline{V}$: average statistic computed from d_1; d_2; ... ; d_n
$\varepsilon_{\overline{U}_{drift}}$, $\varepsilon_{\overline{V}}$: error bounds by Hoeffding's Inequality
for *(each d_i in the list of distance values)* **do**
    update $\overline{U}_{drift}$, $\overline{V}$, $\varepsilon_{\overline{U}_{drift}}$, $\varepsilon_{\overline{V}}$
    if *($\overline{U}_{drift}$ + $\varepsilon_{\overline{U}_{drift}}$ $\geq$ $\overline{V}$ + $\varepsilon_{\overline{V}}$)* **then**
        `// This is an increasing trend`
        Update cut point: $\overline{U}_{drift} = \overline{V}$ and $\varepsilon_{\overline{U}_{drift}} = \varepsilon_{\overline{V}}$
    end
    if *($\overline{U}_{drift}$ - $\varepsilon_{\overline{U}_{drift}}$ $\leq$ $\overline{V}$ - $\varepsilon_{\overline{V}}$)* **then**
        `// This is a decreasing trend`
        Update cut point: $\overline{U}_{drift} = \overline{V}$ and $\varepsilon_{\overline{U}_{drift}} = \varepsilon_{\overline{V}}$
    end
    if *(The rule to reject hypothesis H_0, $|\overline{U}_{drift}$ - $\overline{V}| \geq \varepsilon$ as Equation 8)* **then**
        Output drift with trend (increasing or decreasing) & return drift
    else
        Output Stable State
    end
end

Algorithm 4: IsLocalDrift: Check if there is a local change in a utility stream

Input: List of checkpoints.
Output: Drift state with itemset.

for *(each check point cp in list* checkpoints*)* **do**
    for *(each itemset X in cp.HUIs)* **do**
        split *checkpoints* into two different observations $HS^{n,\lambda}_{W_1}$ and $HS^{m,\lambda}_{W_2}$ at *cp*
        calculate the variance of X, σ^2 by equation 10
        calculate the epsilon cut point, ε_α by equation 9 with Bonferroni correction
        if *(The rule to reject the hypothesis H_0, $|\overline{X}_1$ - $\overline{X_2}| \geq \varepsilon_\alpha$)* **then**
            Output local drift of itemset X
        end
    end
end

The IsLocalDrift procedure (Algorithm 4) is an algorithm to detect local changes in the utilities of itemsets in a stream. Similar to the IsGlobalDrift procedure, IsLocalDrift is also based on probability theory, statistical testing, and the Hoeffding's Inequality. At line 5, the procedure calculates the epsilon cut point with Bonferroni correction to prevent increasing of incorrectly rejecting a null hypothesis when multiple hypotheses are tested.

Complexity. The proposed method uses a time interval length m to select checkpoints at which drift detection is performed. Thus, the time complexity of the proposed approach is $\mathcal{O}(\frac{n}{m})$, where n is the space size of the stream. The HUI-Discovery algorithm is implemented using a traditional utility-list based algorithm for mining high utilty itemsets. In the worst case, the time complexity of a utility-list based algorithm is $\mathcal{O}(2^{|I^*|})$, where I^* is the set of remaining items in the processing window which have

transaction weighted utilities no less than the threshold value. The HUI-Discovery procedure requires only to scan each window twice. It employs an efficient structure *EUCS* and an improved utility list construction method [30] with complexity of $\mathcal{O}(|W|)$, where $|W|$ is the window size. The IsGlobalDrift and IsLocalDrift procedures have $\mathcal{O}(1)$ space and time complexity at each checkpoint.

5. Evaluation

This section presents an experimental evaluation of the performance of our approach. In order to show the generality, feasibility, and applicability of our approach, we performed the evaluation both on synthetic and real-world datasets. The experiments were carried out on a computer running the Windows 10 operating system, having a 64 bit Intel i7 2.6 GHz processor, and 16 GB of RAM. The algorithms were implemented in Java. In all the experiments, the exponential decay function $2^{-\frac{T-t}{|W|}}$ was used, excepts experiments in subsection 5.2.3 studying influence of the decay function. The window size was set to the half-life of the decay function (the time needed for the decay function to decrease a utility value by half). The interval parameter is used to monitor changes at each checkpoint. If its value is large, the number of checkpoints is small and the overlap between windows is small. This, in turn, means that the true positive and accuracy values are high, but it may miss some changes. Moreover, the global confidence level controls the error rate. When the value is high, the number of changes will increase with a high error rate. Setting the interval and confidence level follows the approach proposed in the literature [36, 38, 39] and is application-dependent. The utility threshold influences the number of itemsets, and is also dataset-dependent. If the threshold is set to a small value, the number of high utility itemsets may reach millions. If the threshold is large, few high utility itemsets are obtained. Here, the threshold has been set empirically.

In the following evaluations, we employed several various settings to show the feasibility of our method. In addition, we carried out experiments to evaluate the effects of different parameter values.

5.1. Experiments on real datasets

For the real-world experiments, we used the datasets named Chainstore, Accidents, and Kosarak. These datasets are standard benchmark datasets for utility mining, which were obtained from the SPMF open-source data mining library website[2]. The Chainstore dataset contains real internal and external utilities. For the two other datasets, the internal and external utilities have been generated using a Gaussian distribution in the [1,10] and [1,5] interval, respectively.

Chainstore contains customer transactions from a retail store. The dataset was transformed from the NU-Mine Bench software, and is provided on the SPMF website. Chainstore contains 1,112,949 transactions, with an average transaction length of 7.26 items, and 46,086 distinct items. For this dataset, the checkpoint *interval*, utility *threshold*, and global *confidence* level have been set to 10,000, 600,000, and 0.99, respectively.

Accidents is a traffic accident dataset, often used as a benchmark dataset. It contains 340,183 transactions, 468 distinct items, and having an average transaction length of 33.8 itemsets. For experiments carried out on this dataset, the checkpoint *interval* has been set to 10,000. *window size*, utility *threshold* and global *confidence* level have been initially set to 10,000, 900,000, and 0.99, respectively. These values have then been changed to 15,000, 1,300,000, and 0.8 in the second experiment.

Kosarak is a click-stream dataset of an online Hungarian news portal. It contains 990,000 transactions with 41,270 items, where transactions contain 8.1 items on average. For the experiments on this dataset, the parameters were set to: *threshold* = 200,000, *time interval* = 20,000, and *window size* = 50,000.

Tables 2 - 4 show the results of drift detection on the Chainstore, Accidents, and Kosarak datasets for various parameter values. In these tables, the columns indicate the window size, the number of check points, the number of increasing drifts that has been detected (denoted as Rise), and the number of decreasing drifts that has been detected (denoted as Fall), respectively. The results show that number of drift points vary slightly as parameters are changed. The number of detected drifts also varies slightly when we change the confidence level α, the size of the window, and the utility threshold to monitor the drifts. For the

[2]http://www.philippe-fournier-viger.com/spmf/

Table 2: Drift detection in Chainstore

Window size	Check points	Rise	Fall
40,000	108	35	36
50,000	107	38	35

Table 3: Drift detection in Accidents

Window size	Check points	Rise	Fall
10,000	34	18	15
15,000	33	14	18

real world datasets, there is no ground truth to evaluate the detected concept drifts. Therefore, there is no baseline for testing true positive, false negative and delay detection. In this evaluation, the number of drift points is reported, as well as the corresponding trend (increasing or decreasing). Figures 2 - 5 show visualizations of the utility distribution movements for the three datasets. It can be observed that the utility distribution underlying the data changes gradually and continuously varies for the Chainstore dataset, as new transactions arrive. When the window size is increased, the number of high utility itemsets and the total utility distribution in each window increases. However, the number of states in the Chainstore stream and the utility have only a slight change. Hence, the number of hits (change report) is stable. For the Accidents datasets, more abrupt changes occurred. The dataset containing the largest percentage of stable states is the Kosarak dataset. On this dataset, we varied the confidence level, which is a probability that influences the prediction failure rate. The results in Table 4 show the impact of varying the confidence level. As we can observe in this table, when the confidence value increases, the number of hits increases in both rise and fall states.

5.2. Experiments on synthetic datasets

Because of the lack of appropriate datasets with ground truth, we also performed experiments on synthetic datasets including evaluations influence of parameters, performance of the proposed methods, and comparison to other drift detectors. We used the java open source data mining library SPMF to generated three synthetic datasets, namely StreamT, StreamX, and StreamY. Each dataset contains 50k transactions. These datasets were concatenated multiple times to create a long stream. The reason for using different synthetic datasets and concatenating to form a stream is that the transition points of the resulting stream are known, and can thus be used to evaluate the ability to perform drift detection. Characteristics of the three synthetic datasets are as follows:

StreamT: This dataset contains 50k transactions and 10 distinct items. The average transaction length is 5.5 items. Internal and external utility values were generated in the [1,10] and [1,5] intervals, respectively.

StreamX: This dataset has 50k transactions. The number of distinct items is 15 and the average transaction length is 7.98 items. The [1,15] and [1,10] intervals were used to generate the internal and external utility values of items, respectively, using a Gaussian distribution.

StreamY: This dataset also contains 50k transactions. The average transaction length is 8.02 items, and 20 distinct items are used in this dataset. The internal and external values are generated with the same interval as in *StreamX*.

The stream considered in the following experiments is a concatenation of these three synthetic datasets, obtained by repeating the following pattern 25 times: *StreamT* + *StreamT* + *StreamX* + *StreamY*. This

Table 4: Drift detection in Kosarak

α	Check points	Rise	Fall
0.9	47	7	6
0.5	47	4	5

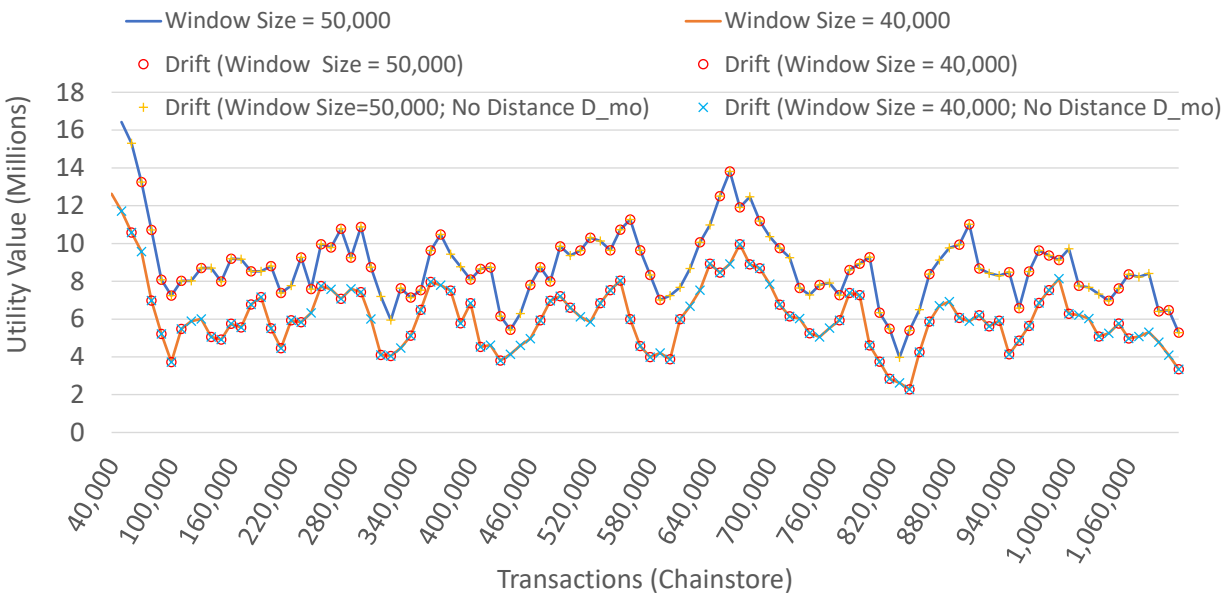


Figure 2: Utility distribution on Chainstore.

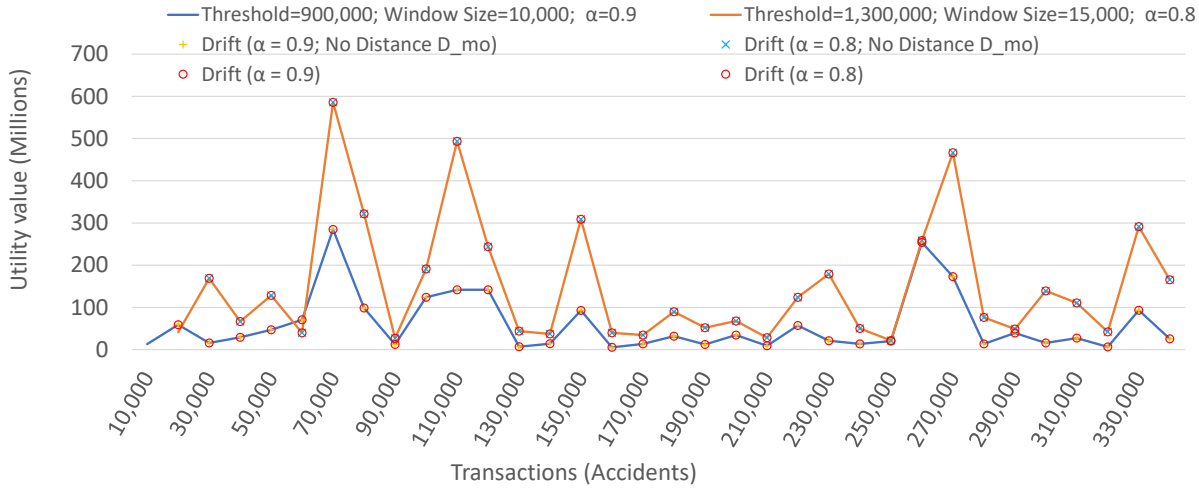


Figure 3: Utility distribution on Accidents.

results in a data stream containing 5 million transactions.

5.2.1. Influence of the confidence level

We carried out an experiment on the synthetic datasets to evaluate the influence of difference confidence level values. The utility threshold was set to 1,500,000. Both the window size and check point interval were set to 50,000 transactions to ensure that a checkpoint was located at every dataset transition. The confidence level was varied from 0.55 to 1.0.

Table 5 shows results when the confidence level α is varied. The result shows that the designed drift detector has a high true positive rate, and low false positive and false negative rates. In particular, when the confidence level α is increased, the accuracy of the approach increases. Generally, for the 1,000 tests performed on the synthetic data streams (a test is performed every 50k transactions), the true positive, false positive and false negative rates, and the accuracy are 86.5%, 0.0%, 27.7%, and 89.6%, respectively. This

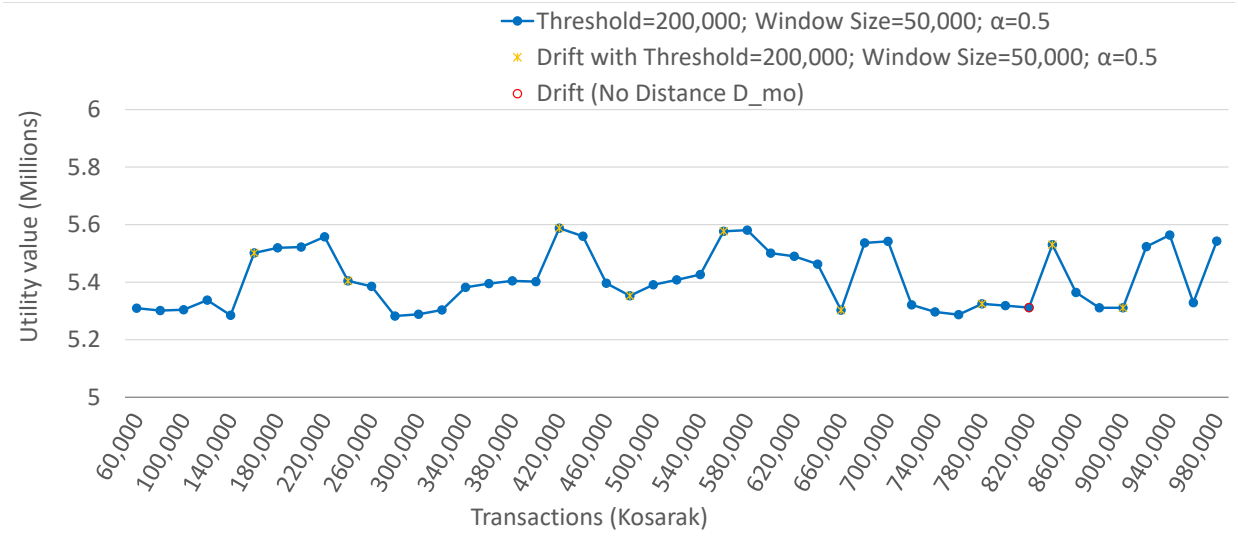


Figure 4: Utility distribution on Kosarak and drift points with $\alpha = 0.5$.

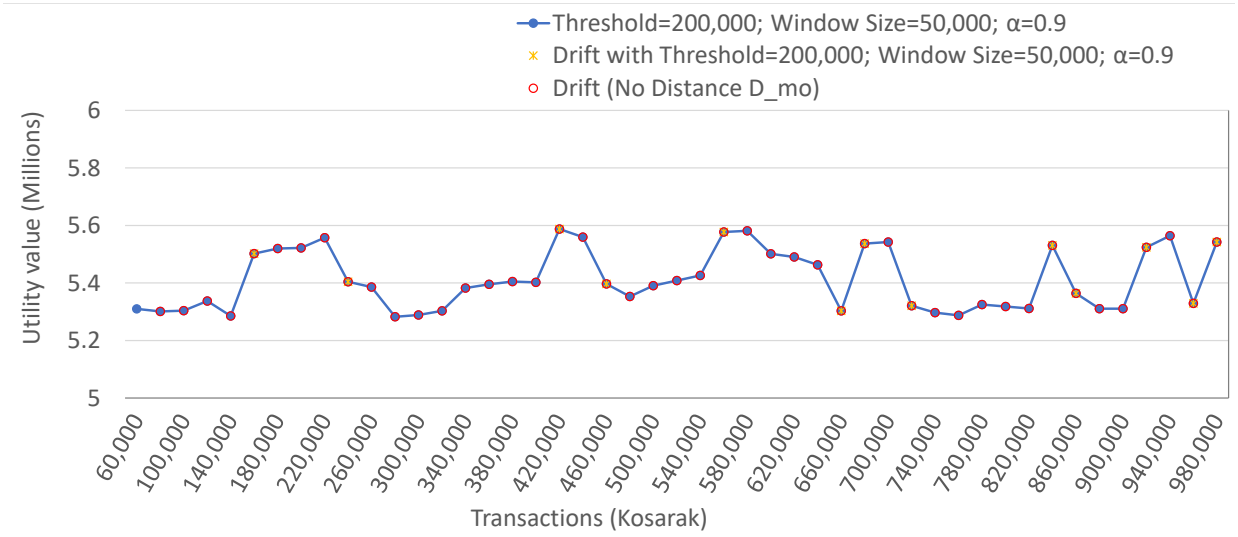


Figure 5: Utility distribution on Kosarak and drift points with $\alpha = 0.9$.

result can be explained as follows. When the confidence level is high, at each checkpoint, the detector checks the utility distribution of two successive windows and it reports concept drifts more accurately than for lower confidence levels where the detector probes concept drifts more tightly. In terms of runtime, the proposed approach is very fast, running in less than three seconds for processing windows of 50k transactions. Hence, the results from this experiment show that the detector can be used in an online setting to detect drifts, and that it can quickly adapt itself to changes.

5.2.2. Influence of the observation times

In the preceding subsection, an experiment was performed where the transition points were known, and where the proposed algorithm was applied exactly at these transition points, while varying the confidence level α. This section describes a follow-up experiment where the proposed algorithm is not applied exactly at the transition points to see the influence of the observation times. Instead, the checkpoint are gradually moved away from the transition points in the data stream. For this experiment, the confidence level was set

Table 5: Detection on synthetic datasets

α	FP rate(%)	TP rate(%)	FN rate(%)	Rise	Fall	Time Avg (s)
0.55	0.0	66.2	49.01	25	24	2.773
0.60	0.0	66.2	49.01	25	24	2.725
0.65	0.0	66.2	49.01	25	24	2.721
0.70	0.0	66.2	49.01	25	24	2.709
0.75	0.0	100.0	0.0	25	49	2.721
0.80	0.0	100.0	0.0	25	49	2.690
0.85	0.0	100.0	0.0	25	49	2.570
0.90	0.0	100.0	0.0	25	49	2.583
0.95	0.0	100.0	0.0	25	49	2.580
1.00	0.0	100.0	0.0	25	49	2.563
Summary (tests)				Accuracy rate (%)		Time Avg (s)
1000	0.0%	86.5%	27.7%	89.6%		2.664

to 0.9. The observation times where the change detection algorithm applying was shifted 8 times forward by 25 transactions. As a result, windows considered by the algorithm may contain transactions from two datasets. After each shift, the proposed algorithm was executed 100 times for 100 checkpoints.

Table 6: Detection on synthetic datasets with shift point

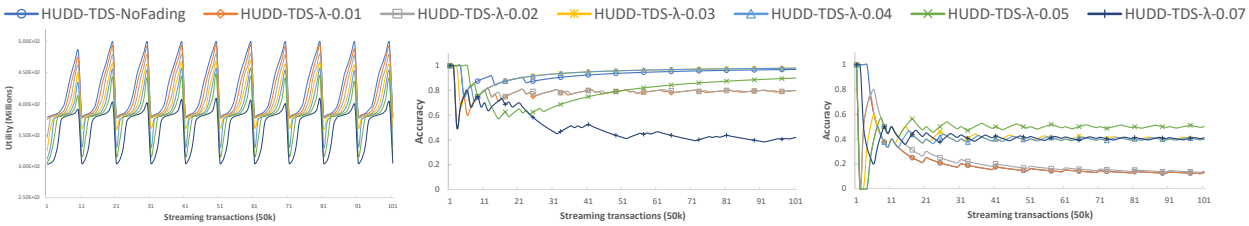
Shift	FP rate(%)	TP rate(%)	FN rate(%)	Rise	Fall	Time Avg (s)
25	0.0	100.0	0.0	25	49	2.744
50	0.0	100.0	0.0	25	49	2.707
75	0.0	100.0	0.0	25	49	2.733
100	0.0	100.0	0.0	25	49	2.768
125	0.0	100.0	0.0	25	49	2.748
150	0.0	100.0	0.0	25	49	2.771
175	0.0	100.0	0.0	25	49	2.726
200	0.0	100.0	0.0	25	49	2.754
Summary (tests)				Accuracy rate (%)		Time Avg (s)
800	0.0%	100.0%	0.0%	100.0%		2.744

Table 6 shows the results of this experiment, where checkpoints are shifted away from the real transition points of datasets. It is observed that when the shift is increased from 25 to 200 transactions, the detector can detect changes in the utility distribution without making any mistakes. The false positive and false negative rates of the proposed method are in that case equal to zero. The proposed drift detector has high true positive rate and accuracy. In a stream, data evolves and changes over time. Detecting changes within an acceptable delay is crucial. As shown in our experiments, when the number of shifted transactions was non-zero, our detector was able to detect exactly all changes in the utility distribution of the datasets under an acceptable delay, which was defined as 200 transactions for these experiments.

5.2.3. Influence of the decay function

We generated a synthetic stream containing 101 instances. Each instance in the stream has 50k transactions. Thus the stream has 5.05 million transactions. We use the dataset StreamT as our first seed. Each instance in the stream was seeded from its previous instance as follows. We randomly sampled a 10% of transactions in the seed. We added noise to these samples by increasing utility value to 5%. In the stream,

the instances at positions of 11, 21, 31, ... and so on were reset seeded from the StreamT. The utility threshold was set to 1,200,000. In the previous experiments, the window size was used as the half-life of the decay function. To evaluate the influence of the decay function, we multiplied the decay function with a constant λ. The value of λ was set to 0.01, 0.02, 0.03, 0.04, 0.05, and 0.07, respectively. We recorded experiment results of the proposed algorithm without using decay fading and with using different decay values as described above. Figure 6a is the utility distribution of high utility itemsets in the stream. Figures 6b - 6c show the accuracy of the proposed algorithm with the confidence level value set to 0.7 and 0.8 respectively. The results show that the accuracy is gradually stable, and it is affected by decay value. From Figure 6a, we can observe that the utility changes slightly when we vary the decay factor. If λ's value is high, the importance of old transactions quickly decreases. Therefore, the weights of past transactions have a minor influence on the utility of itemsets, and the total utility distribution changes more smoothly than for small λ values. When α=0.7, the best accuracy of the proposed detector was obtained with λ=0.03 or λ=0.04. While the best accuracy of the proposed detector was obtained with λ=0.05 if α=0.8. The value of decay function is application-specific and can be chosen either by using a heuristic method.



(a) Utility obtained (b) Accuracy with confidence α=0.7 (c) Accuracy with confidence α=0.8

Figure 6: Influence of the decay function.

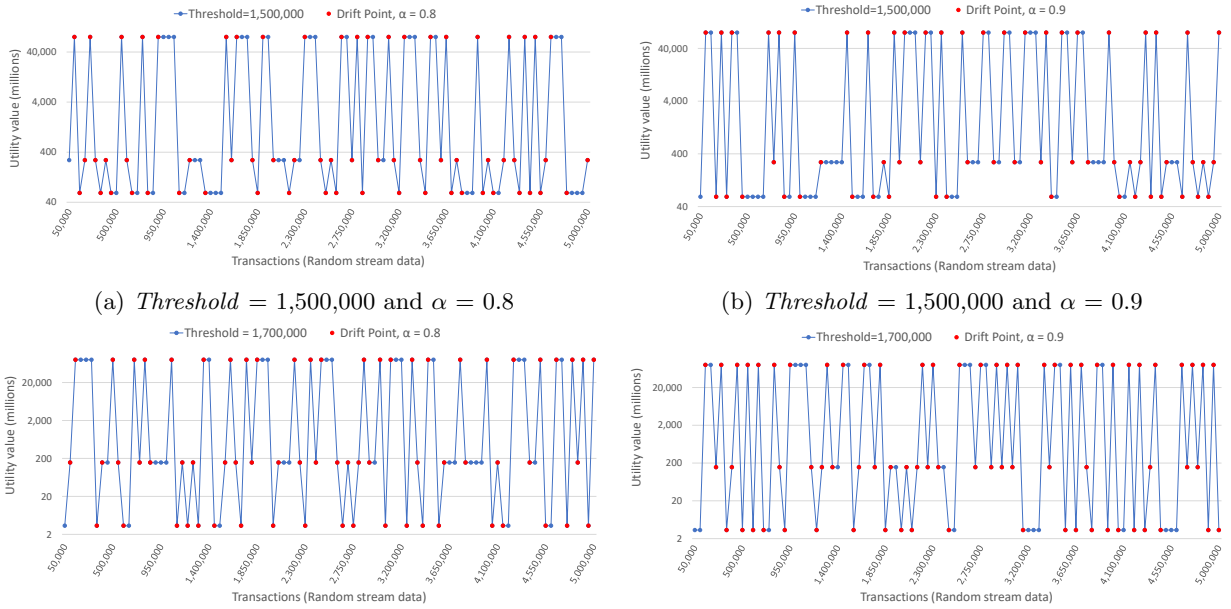
5.2.4. Evaluation on a random stream

The synthetic data stream used in the previous experiments is a repeated concatenation of the following sequence of datasets: $StreamT + StreamT + StreamX + StreamY$. To more extensively evaluate the performance of the designed method, an additional experiment was performed where we concatenated several datasets in different ways to generate a synthetic stream. In this experiment, streams are generated by combining a sequence of datasets formed as $Dataset_1 - Dataset_2 - \cdots - Dataset_k$, where $Dataset_i$ $(1 \leq i \leq k)$ is randomly selected among $StreamT$, $StreamX$ and $StreamY$. Each stream includes 100 random instances of datasets and has 5 million transactions. Four random data streams were generated. The designed algorithm was run with a confidence level set to 0.8 and 0.9, and a minimum utility threshold set to 1,500,000 and 1,700,000.

Figures 7a - 7d show drift points detected by our detector for the four random data streams. In these figures, a red point indicates an observation time where a change in the utility distribution of high utility itemsets was found. At each checkpoint, the proposed algorithm examines the movement in utility distributions in the window consisting of the transactions since the last estimated change point. If the movement is significantly different, the algorithm marks the checkpoint as a drift point. That checkpoint is then remembered as the last estimated change point. Results have shown that the proposed algorithm can detect exactly all the drift points at the checkpoints, where there is a significant change at the transitions between successive windows.

5.2.5. Evaluation of the drift detector for classification

This subsection presents experiments to evaluate our proposed detector in terms of detection delay, true positive (TP), true negative (TN), false negative (FN), and accuracy with a base incremental classifier applied on a stream. We used Natïve Bayes (NB) and Hoeffding Tree (HT) classifiers as base learners



(a) *Threshold* = 1,500,000 and $\alpha = 0.8$

(b) *Threshold* = 1,500,000 and $\alpha = 0.9$

(c) *Threshold* = 1,700,000 and $\alpha = 0.8$

(d) *Threshold* = 1,700,000 and $\alpha = 0.9$

Figure 7: Drift points on Random Streams.

because they are usually used as benchmark classifiers in the literature [24, 25]. Moreover, we compared the results with several state-of-the-art drift detectors, namely EDDM [40], ECCD [41], SeqDrift2 [42], and RDDM [39]. All experiments were performed using the MOA framework [38] with parameters set to the default values for all the compared algorithms, as recommended in the original papers. Note that our drift detector uses a statistical test [25] as global detector. However, the original detector tests to reject the null hypothesis with an error bound ε of the streaming values from the [0; 1] interval. The proposed detector considers the error bound ε on a real value interval of the streaming data. For this evaluation, we used three widely-used synthetic data streams, namely Mixed, Sine, and Circles [21, 25]. Each stream dataset contains 100,000 instances.

The characteristics of these datasets are as follows [18]:

- *Mixed*: This dataset contains four attributes, including two Boolean attributes (v, w) and two numeric attributes (x, y) in the [0, 1] interval. If two of three conditions are satisfied: $v, w, y < 0.5+0.3sin(3\pi x)$, the instance is classified as positive. The Mixed dataset contains abrupt concept drifts. Drifts occur at every 20,000 instances with a transition length $\xi = 50$.

- *Sine1*: There are two attributes x and y that are uniformly distributed in the [0, 1] interval. If all points are below the piecewise function $y = sin(x)$, they are classified as positive. The classification is reversed after a change. The Sine1 dataset contains abrupt concept drifts. Drifts occur at every 20,000 instances with a transition length $\xi = 50$.

- *Circles*: This dataset uses four circles to simulate drift concepts. The radius of the circles are 0.15, 0.2, 0.25, and 0.3, respectively. Each instance has two numeric attributes (x, y) on the [0, 1] interval. If an instance is inside the circles, it is classified as positive. The Circles dataset contains gradual concepts drifts. Drifts occur at every 25,000 instances with a transition length $\xi = 500$.

In streaming data, to evaluate the measures of concept drift detector, such as true positive, true negative, false negative, and accuracy, the *acceptable delay* length metric [21, 43] is often adopted. Given a threshold Δ, if a detector can detect a change within a delay Δ from the true change point, it is considered as a true positive. Mixed and Sine1 contain abrupt concept drifts, while Circles contains gradual drifts. Therefore, in

Table 7: Results with Natïve Bayes(NB) and Hoeffding Tree(HT) classifiers

		Algorithms	Delay	TP	FP	FN	Accuracy	Rank
Mixed dataset	NB	HUDD-TDS	81.75 ± 18.71	3.97 ± 0.17	1.61 ± 1.34	0.03 ± 0.17	**83.26 ± 0.09**	**1**
		RDDM	104.97 ± 12.12	3.99 ± 0.1	1.86 ± 1.66	0.01 ± 0.1	83.24 ± 0.09	2
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.39 ± 0.79	0.0 ± 0.0	82.91 ± 0.08	3
		ECCD	38.87 ± 24.65	3.81 ± 0.42	142.29 ± 7.90	0.19 ± 0.42	81.00 ± 0.15	4
		EDDM	247.47 ± 8.65	0.11 ± 0.31	20.22 ± 7.70	3.89 ± 0.31	80.30 ± 2.33	5
	HT	HUDD-TDS	68.20 ± 15.44	4.0 ± 0.0	3.41 ± 2.09	0.0 ± 0.0	**83.25 ± 0.14**	**1**
		RDDM	106.68 ± 11.32	3.99 ± 0.1	3.49 ± 2.48	0.01 ± 0.1	83.17 ± 0.12	2
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.98 ± 1.21	0.0 ± 0.0	82.91 ± 0.11	3
		ECCD	39.76 ± 26.08	3.79 ± 0.46	138.34 ± 7.95	0.21 ± 0.46	80.95 ± 0.15	4
		EDDM	248.46 ± 7.73	0.05 ± 0.22	21.51 ± 7.74	3.95 ± 0.22	80.65 ± 0.82	5
Sine1 Dataset	NB	HUDD-TDS	85.68 ± 24.47	3.96 ± 0.20	1.04 ± 1.06	0.04 ± 0.20	85.94 ± 0.25	2
		RDDM	89.73 ± 16.54	3.99 ± 0.10	3.93 ± 2.92	0.01 ± 0.1	**85.98 ± 0.27**	**1**
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.26 ± 0.0	0.0 ± 0.58	85.60 ± 0.25	3
		ECCD	33.30 ± 23.22	3.85 ± 0.39	153 ± 8.34	0.15 ± 0.39	84.38 ± 0.14	4
		EDDM	234.28 ± 22.33	0.57 ± 0.64	33.53 ± 11.56	3.43 ± 0.64	83.44 ± 2.88	5
	HT	HUDD-TDS	58.96 ± 13.04	4.0 ± 0.0	2.25 ± 1.56	0.0 ± 0.0	**86.93 ± 0.17**	**1**
		RDDM	93.54 ± 7.82	4.0 ± 0.0	4.72 ± 3.59	0.0 ± 0.0	86.79 ± 0.19	2
		SeqDrift2	200.0 ± 0.0	4.0 ± 0.0	4.83 ± 1.16	0.0	86.53 ± 0.15	3
		ECCD	36.58 ± 25.54	3.8 ± 0.43	153.78 ± 7.67	0.2 ± 0.43	84.28 ± 0.14	5
		EDDM	243.83 ± 22.33	0.22 ± 0.64	33.77 ± 11.56	3.78 ± 0.64	84.71 ± 2.88	4
Circles Dataset	NB	HUDD-TDS	311.18 ± 109.16	2.9 ± 0.3	1.01 ± 0.93	0.1 ± 0.3	84.09 ± 0.12	2
		RDDM	406.50 ± 69.75	2.99 ± 0.1	2.15 ± 1.95	0.01 ± 0.1	84.05 ± 0.12	3
		SeqDrift2	276.67 ± 91.56	2.92 ± 0.27	2.49 ± 0.98	0.08 ± 0.27	**84.13 ± 0.14**	**1**
		ECCD	194.64 ± 158.13	2.84 ± 0.37	174.53 ± 7.62	0.16 ± 0.37	83.18 ± 0.11	4
		EDDM	938.27 ± 107.14	0.35 ± 0.5	31.09 ± 18.23	2.65 ± 0.5	83.12 ± 0.4	5
	HT	HUDD-TDS	242.60 ± 147.48	2.56 ± 0.52	4.71 ± 1.67	0.44 ± 0.52	85.97 ± 0.23	3
		RDDM	293.80 ± 38.72	2.98 ± 0.14	0.79 ± 1.26	0.02 ± 0.14	86.46 ± 0.16	2
		SeqDrift2	202.67 ± 16.19	3.0 ± 0.0	3.08 ± 0.91	0.0 ± 0.0	**86.47 ± 0.14**	**1**
		ECCD	186.40 ± 151.67	2.86 ± 0.35	175.16 ± 8.39	0.14 ± 0.35	83.21 ± 0.12	5
		EDDM	987.75 ± 54.64	0.06 ± 0.24	24.45 ± 14.57	2.94 ± 0.24	84.89 ± 0.29	4

Algorithms	HUDD-TDS	RDDM	SeqDrift2	ECCD	EDDM
Average Rank	**1.67**	2.0	2.33	4.33	4.67

this experiment, smaller values of acceptable delay are set for Mixed and Sine1 than for Circles. Specifically, the *acceptable delay* is set to 250 on the Mixed and Sine1 datasets, and 1,000 on the Circles dataset.

Table 7 shows the average and standard deviation of classification results for the proposed detector, EDDM, ECCD, SeqDrift2, and RDDM running on 100 samples of datasets. We can observe that on the Mixed and Sine1 datasets, ECCD has the shortest detection delay, with high true positive (TP) rates with both Natïve Bayes (NB) and Hoeffding Tree (HT) classifiers. This is because ECCD uses a window containing a small number of instances. However, the false positive (FP) rates are also very high, resulting in a low accuracy. For the Sine1 dataset with NB learner, RDDM discards old instances from the stream and is the most accurate. However, the difference between RDDM and HUDD-TDS is small, and the FP rate of RDDM is higher than the FP rate of HUDD-TDS. In almost all other cases, the proposed detector has the best accuracy and very good flow rates of detection delay, TP, FP, and false negative (FN). The reason for this is that for the Mixed and Sine1 datasets, changes are abrupt, and the proposed detector is an online and non-parametric detector. It estimates precisely the distribution of the stream. Therefore, it can quickly detect points lying out of the distribution boundary. For gradual concept drifts such as in the

Circles dataset, SeqDrift2 has the best performance, either using NB or HT learners. The reason is that SeqDrift2 uses a block of 200 instances for reservoir sampling. The block is helpful for the Circles dataset since it contains gradual concepts drifts with a transition length of 500. Nevertheless, FP rate of SeqDrift2 is still high. In most cases, ECCD and EDDM are the worst detectors.

To summaraze, the average ranks of the algorithms are shown in Table 7. This shows that the proposed detector is ranked first among the five compared detectors.

5.3. Evaluation of runtime performance

This section presents the experimental results of the proposed method in terms of runtime consumption on both real-world datasets and synthetic datasets. Figures 8 - 10 show the runtime of the proposed approach on the real-world datasets including Chainstore, Accidents, and Kosarak. The average runtime is very small, and it is just around a second for performing drift detection and processing windows of 50k transactions at each checkpoint. The average runtime are respectively around 156(ms), 2.5(s), and 494(ms) on the Chainstore, Accidents, and Kosarak datasets.

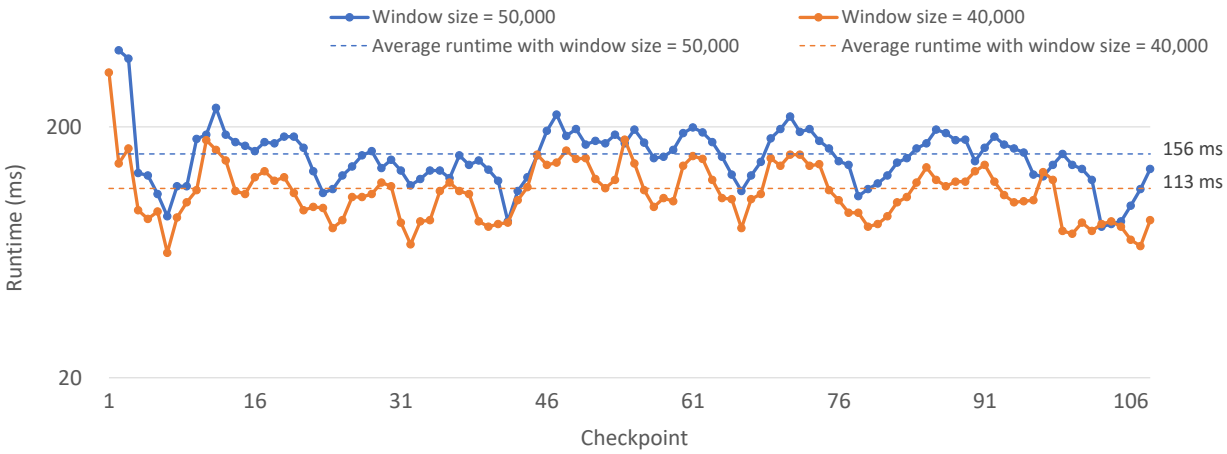


Figure 8: Runtime on Chainstore.

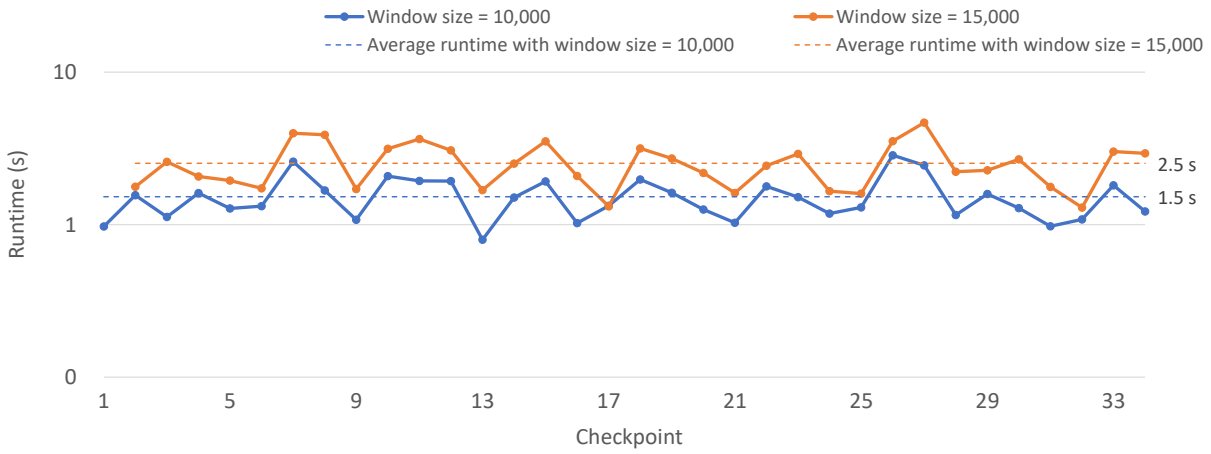


Figure 9: Runtime on Accidents.

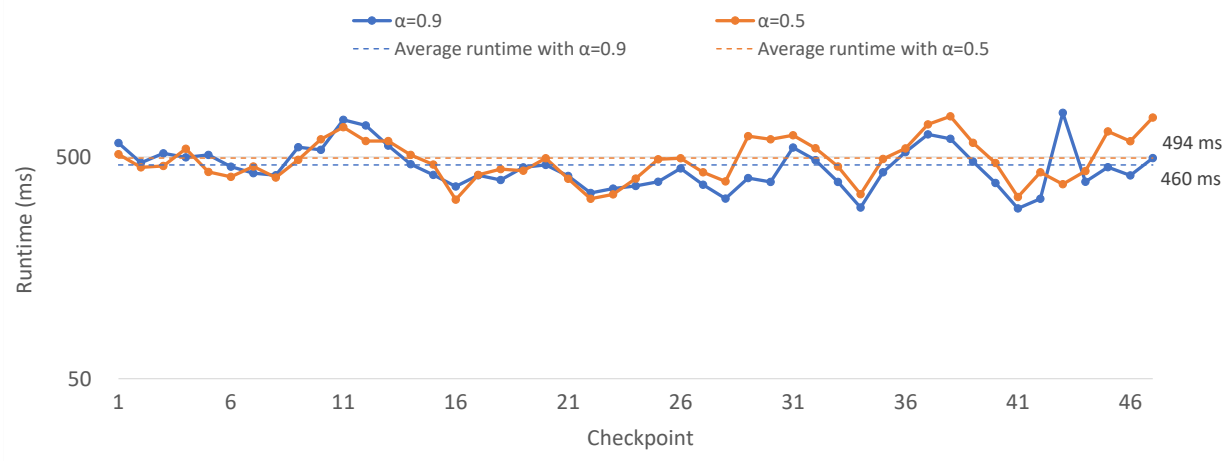
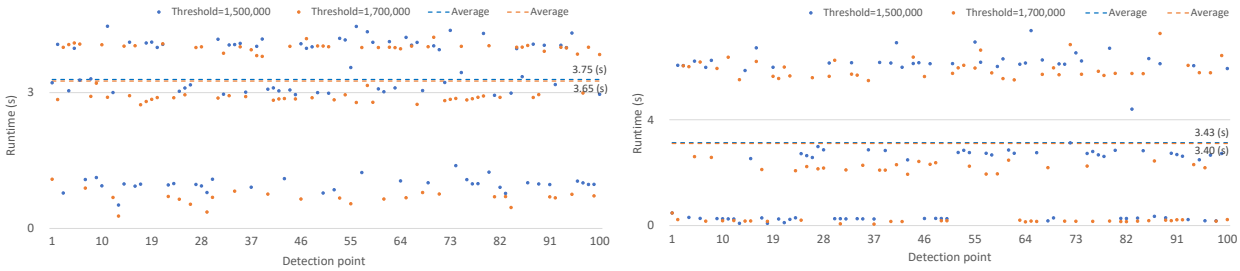


Figure 10: Runtime on Kosarak.

On the four random data streams, the proposed approach is quite fast. As presented in Figures 11a - 11b, it takes around three seconds in average for performing detection in a window of 50k transactions. Figure 12 shows average runtime performing detection on random streams while varying window size. At first, window size was set to 50,000 and then was increased 10 times. Each time we increased the size of window by 10,000 transactions. We record the average runtime consumption while utility threshold values are set to 1,500,000 and 1,700,000 respectively, along with and without using the D_{mo} distance. The result shows that the average runtime consumption is small, and it is linear when the size of window increases.



(a) $\alpha = 0.8$ (b) $\alpha = 0.9$

Figure 11: Runtime on Random Streams.

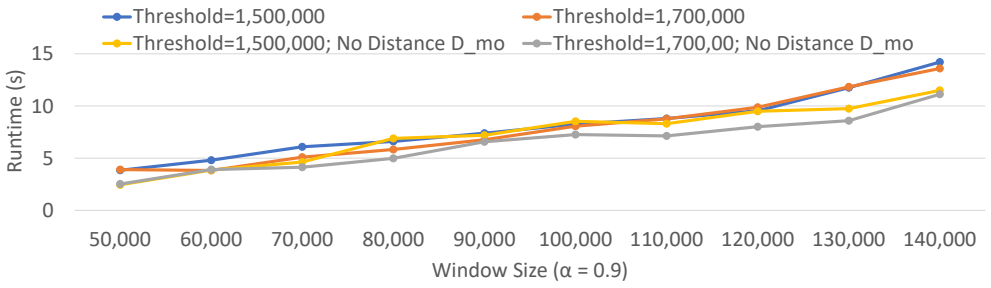


Figure 12: Average Runtime on Random Streams.

In general, our empirical experiments have shown that running time of the proposed algorithm is very small on both real-world datasets and synthetic datasets under various parameter settings. Furthermore, our detector is online and non-parametric, which makes it suitable for online drift detection from a stream, where concept drifts are short and the method must quickly detect changes with high true positive, high accuracy, and low detection delay.

6. Conclusion

In this paper we presented an algorithm named High Utility Drift Detection in Transactional Data Stream (HUDD-TDS) to detect changes in the utility distributions of itemsets in a stream of quantitative customer transactions. The algorithm utilizes a fading function to quickly adapt to changes in a data stream by placing less importance on older transactions than the recent ones. To ensure that only significant changes are reported to the user, we proposed an approach that uses statistical testing based on the Hoeffding's Inequality with Bonferroni correction. The main advantage with our algorithm is that it can detect both local and global utility drifts, i.e., drifts in the utility distribution of single patterns and of multiple high utility patterns, respectively, in addition to discovering both increasing and decreasing trends. To detect global utility drifts, we proposed a new distance measure, which generalizes the cosine similarity, by also taking the distance between pairs of high utility itemsets into account. To evaluate our approach, we carried out extensive experiments both on real and synthetic datasets. The results from the experiments showed the efficiency of the proposed method. In particular, it can identify both types of changes in the utility distributions of high utility itemsets in real-time, yielding a high true positive rate, while keeping both false positive and false negative at a lowest possible rate.

For future work, we will explore the detection of patterns with different weights in streams, online detection, and also develop techniques to automatically set the parameters for specific applications. Moreover, we plan to extend the proposed method to detect changes using closed and maximal high utility itemsets.

Acknowledgements

This research was funded by the Norwegian University of Science and Technology (NTNU) through the MUSED project.

References

[1] R. Agrawal, R. Srikant, Fast Algorithms for Mining Association Rules , VLDB (1994) 487–499.
[2] W. Yao-Te, A. J. T. Lee, Mining Web navigation patterns with a path traversal graph, Expert Systems with Applications 38 (6) (2011) 7112–22.
[3] A. Deepak, D. Fernandez-Baca, S. Tirthapura, M. J. Sanderson, M. M. McMahon, EvoMiner: frequent subtree mining in phylogenetic databases, Knowledge and Information Systems 41 (3) (2014) 559–590.
[4] H. Yao, H. J. Hamilton, C. J. Butz, A foundational approach to mining itemset utilities from databases, in: Proceedings of the 4th Siam International Conference on Data Mining, 2004, pp. 482–486.
[5] Y. Liu, W.-k. Liao, A. Choudhary, A Two-phase Algorithm for Fast Discovery of High Utility Itemsets, in: Proceedings of the 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, 2005, pp. 689–695.
[6] P. Fournier-Viger, J. C.-W. Lin, Q.-H. Duong, T.-L. Dam, FHM+: Faster High-Utility Itemset Mining Using Length Upper-Bound Reduction, in: Trends in Applied Knowledge-Based Systems and Data Science, 2016, pp. 115–127.
[7] J. Cheng, Y. Ke, W. Ng, A survey on algorithms for mining frequent itemsets over data streams, Knowledge and Information Systems 16 (1) (2008) 1–27.
[8] N. Manerikar, T. Palpanas, Frequent items in streaming data: An experimental evaluation of the state-of-the-art, Data & Knowledge Engineering 68 (4) (2009) 415 – 430.
[9] H.-F. Li, H.-Y. Huang, S.-Y. Lee, Fast and memory efficient mining of high-utility itemsets from data streams: with and without negative item profits, Knowledge and Information Systems 28 (3) (2011) 495–522.
[10] S.-J. Yen, Y.-S. Lee, An Efficient Approach for Mining High Utility Itemsets Over Data Streams, in: Proceedings of Data Science and Big Data: An Environment of Computational Intelligence, 2017, pp. 141–159.
[11] A. Balzanella, R. Verde, Summarizing and Detecting Structural Drifts from Multiple Data Streams, in: Proceedings of Classification and Data Mining, 2013, pp. 105–112.
[12] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Computing Surveys 46 (4) (2014) 44:1–44:37.

[13] G. Ditzler, R. Polikar, Incremental Learning of Concept Drift from Streaming Imbalanced Data, IEEE Transactions on Knowledge and Data Engineering 25 (10) (2013) 2283–2301.
[14] D. Marron, J. Read, A. Bifet, T. Abdessalem, E. Ayguade, J. Herrero, Echo State Hoeffding Tree Learning, in: Proceedings of the 8th Asian Conference on Machine Learning, Vol. 63, 2016, pp. 382–397.
[15] M. Pechenizkiy, J. Bakker, I. Žliobaitė, A. Ivannikov, T. Kärkkäinen, Online Mass Flow Prediction in CFB Boilers with Explicit Detection of Sudden Concept Drift, SIGKDD Explorations Newsletter 11 (2) (2010) 109–116.
[16] R. Klinkenberg, Learning Drifting Concepts: Example Selection vs. Example Weighting, Intelligent Data Analysis 8 (3) (2004) 281–300.
[17] J. a. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A Survey on Concept Drift Adaptation, ACM Computing Surveys 46 (4) (2014) 44:1–44:37.
[18] J. Gama, P. Medas, G. Castillo, P. P. Rodrigues, Learning with Drift Detection, in: Advances in Artificial Intelligence - SBIA, 2004, pp. 286–295.
[19] A. Bouchachia, Fuzzy classification in dynamic environments, Soft Computing 15 (5) (2011) 1009–1022.
[20] I. Adä, M. R. Berthold, EVE: a framework for event detection, Evolving Systems 4 (1) (2013) 61–70.
[21] A. Pesaranghader, H. L. Viktor, Fast Hoeffding Drift Detection Method for Evolving Data Streams, in: Proceedings of the 2016 ECML Machine Learning and Knowledge Discovery in Databases Conference, 2016, pp. 96–111.
[22] A. Bifet, R. Gavaldà, Adaptive Learning from Evolving Data Streams, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 249–260.
[23] U. Adhikari, T. Morris, S. Pan, Applying Hoeffding Adaptive Trees for Real-Time Cyber-Power Event and Intrusion Classification, IEEE Transactions on Smart Grid PP (99) (2017) 1–12.
[24] A. Bifet, R. Gavaldà, Learning from Time-Changing Data with Adaptive Windowing, in: Proceedings of the 2007 SIAM International Conference on Data Mining, pp. 443–448.
[25] I. Frías-Blanco, J. del Campo-Ávila, G. Ramos-Jiménez, R. Morales-Bueno, A. Ortiz-Díaz, Y. Caballero-Mota, Online and Non-Parametric Drift Detection Methods Based on Hoeffding's Bounds, IEEE Transactions on Knowledge and Data Engineering 27 (3) (2015) 810–823.
[26] C. Ahmed, S. Tanbeer, B. S. Jeong, Y. K. Lee, Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases, IEEE Transactions on Knowledge and Data Engineering 21 (12) (2009) 1708–1721.
[27] Q.-H. Duong, P. Fournier-Viger, H. Ramampiaro, K. Nørvåg, T.-L. Dam, Efficient high utility itemset mining using buffered utility-lists, Applied Intelligence PP (2017) 1–19.
[28] V. Tseng, B.-E. Shie, C.-W. Wu, P. Yu, Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases, IEEE Transactions on Knowledge and Data Engineering 25 (8) (2013) 1772–1786.
[29] M. Liu, J. Qu, Mining High Utility Itemsets Without Candidate Generation, in: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, 2012, pp. 55–64.
[30] Q.-H. Duong, B. Liao, P. Fournier-Viger, T.-L. Dam, An Efficient Algorithm for Mining the Top-k High Utility Itemsets, Using Novel Threshold Raising and Pruning Strategies, Knowledge-Based Systems 104 (C) (2016) 106–122.
[31] T.-L. Dam, K. Li, P. Fournier-Viger, Q.-H. Duong, An efficient algorithm for mining top-k on-shelf high utility itemsets, Knowledge and Information Systems (2017) 1–35.
[32] H. F. Li, H. Y. Huang, Y. C. Chen, Y. J. Liu, S. Y. Lee, Fast and Memory Efficient Mining of High Utility Itemsets in Data Streams, in: 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 881–886.
[33] S. Dawar, V. Sharma, V. Goyal, Mining top-k high-utility itemsets from a data stream under sliding window model, Applied Intelligence 47 (4) (2017) 1240–1255.
[34] W. Ng, M. Dash, A Test Paradigm for Detecting Changes in Transactional Data Streams, in: Proceedings of the 13th International Conference on Database Systems for Advanced Applications, 2008, pp. 204–219.
[35] Y. S. Koh, CD-TDS: Change detection in transactional data streams for frequent pattern mining, in: Proceedings of the 2016 International Joint Conference on Neural Networks, 2016, pp. 1554–1561.
[36] W. Hoeffding, Probability Inequalities for Sums of Bounded Random Variables, Journal of the American Statistical Association 58 (301) (1963) 13–30.
[37] A. Heidarian, M. J. Dinneen, A Hybrid Geometric Approach for Measuring Similarity Level Among Documents and Document Clustering, in: 2016 IEEE Second International Conference on Big Data Computing Service and Applications (BigDataService), 2016, pp. 142–151.
[38] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: Massive Online Analysis, The Journal of Machine Learning Research 11 (2010) 1601–1604.
[39] R. S. Barros, D. R. Cabral, P. M. Gonalves, S. G. Santos, RDDM: Reactive drift detection method, Expert Systems with Applications 90 (Supplement C) (2017) 344 – 355.
[40] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldá, R. Morales-Bueno, Early drift detection method, in: The 4th International Workshop on Knowledge Discovery from Data Streams, 2006.
[41] G. J. Ross, N. M. Adams, D. K. Tasoulis, D. J. Hand, Exponentially weighted moving average charts for detecting concept drift, Pattern Recognition Letters 33 (2) (2012) 191 – 198.
[42] R. Pears, S. Sakthithasan, Y. S. Koh, Detecting concept change in dynamic data streams, Machine Learning 97 (3) (2014) 259–293.
[43] A. Pesaranghader, H. Viktor, E. Paquet, McDiarmid Drift Detection Methods for Evolving Data Streams, CoRR abs/1710.02030. arXiv:1710.02030.