pcStream: A Stream Clustering Algorithm for Dynamically Detecting and Managing Temporal Contexts

Yisroel Mirsky(✉), Bracha Shapira, Lior Rokach, and Yuval Elovici

Department of Information Systems Engineering, Ben Gurion University, Be'er Sheva, Israel
{yisroel,liorrk}@post.bgu.ac.il, {bshapira,yuval}@bgu.ac.il

Abstract. The clustering of unbounded data-streams is a difficult problem since the observed instances cannot be stored for future clustering decisions. Moreover, the probability distribution of streams tends to change over time, making it challenging to differentiate between a concept-drift and an anomaly. Although many excellent data-stream clustering algorithms have been proposed in the past, they are not suitable for capturing the temporal contexts of an entity.

In this paper, we propose pcStream; a novel data-stream clustering algorithm for dynamically detecting and managing sequential temporal contexts. pcStream takes into account the properties of sensor-fused data-streams in order to accurately infer the present concept, and dynamically detect new contexts as they occur. Moreover, the algorithm is capable of detecting point anomalies and can operate with high velocity data-streams. Lastly, we show in our evaluation that pcStream outperforms state-of-the-art stream clustering algorithms in detecting real world contexts from sensor-fused datasets. We also show how pcStream can be used as an analysis tool for contextual sensor streams.

Keywords: Stream clustering · Concept detection · Concept drift · Context-awareness

1 Introduction

Context, in the scope of machine learning, can be described as any information that helps explain an entity's behavior [9]. Context-awareness is the idea of constantly tracking an entity's context over time for some application [18]. For example, an application of context-awareness is the task of data-leakage prevention for smartphones. In this instance, the tracked context is the locomotion of the user (e.g. walking or running) and the behavior of interest is the outgoing emails. By tracking the context, a machine learning algorithm can infer that it is unlikely for an email to be sent while the user is running.

Modern technology can generate vast amounts of sensor data continuously. Even a singular entity, such as a smartphone, can generate a potentially endless amount of data from its sensors. A sensor-stream can be viewed as a sequence of attribute vectors in geometric space [24]. From these sensor-streams it is possible to obtain a context-awareness of the entity [5, 20]. One method is to define an ontology, or a rule set, for each known context—as was done in [22]. However, defining contexts for a sensor-stream is impractical because the definition of contexts may change over time and previously unseen contexts may appear later on. For instance, the definition of a

T. Cao et al. (Eds.): PAKDD 2015, Part II, LNAI 9078, pp. 119–133, 2015.
DOI: 10.1007/978-3-319-18032-8_10

person's home may change when he/she moves, and the sensory definition of a user as he/she is walking will change as he/she gets older.

A concept is an underlying distribution observed from a data-stream which is stable for a period of time [11]. Concepts may reoccur, or evolve over time [17, 28]. In many cases, an entity's context is linked to underlying concepts found in its stream. Therefore, these "hidden contexts" of the entity can be implicitly extracted from the stream itself [13, 14, 26]. Since the hidden contexts have distinct distributions, stream clustering can be performed to detect them in an unsupervised manner.

Clustering a stream is challenging since memory is limited and the stream is potentially boundless. Although many data-stream clustering algorithms exist, they are not suitable for clustering contextual sensor-streams because overlapping clusters cannot be detected. This is because: **1)** *The temporal relation of the arriving points in the clustering decision is not considered (the data-flow is clustered as a sporadic mixture of classes),* **2)** *The clusters' correlated distributions are generally not considered.*

To exemplify the importance of these aspects, consider the case of performing clustering on a smartphone's accelerometer for the application of activity recognition. The objective is to capture and distinguish the underlying contexts found in the stream. Illustrated in Fig. 1 is a possible sequence of points captured from an arbitrary sensor. Here, the captured points form three distinct overlapping distributions (contexts) in a sequential manner.

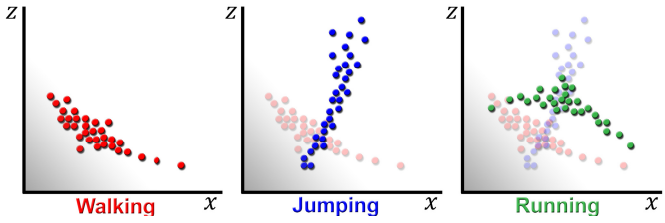

Fig. 1. An illustration of possible sensor values captured as a smartphone user walks (*left*) then repeatedly jumps (*middle*) and then runs (*right*). Here the clusters form distinct correlated distributions which overlap in geometric space.

Since other stream clustering algorithms cannot cope with these types of streams, we propose a different approach to this type of clustering problem. Concretely, when a stream exhibits a certain context, all instances during that time period should be assigned (i.e. clustered) to that context. In other words, the data-stream should be partitioned ad hoc according to the inherent contexts, while detecting reoccurring contexts and accounting for concept drift.

In this paper we present pcStream; a stream clustering algorithm for dynamically detecting and managing temporal contexts. The name "**pc**Stream" is attributed to the **p**rincipal **c**omponents of the distributions in the data-stream which are used to dynamically detect and compare contexts (discussed later in further detail).

The paper's theoretical contributions are: **1)** *a novel method for partitioning data streams considering both temporal and spatial domains during the clustering decision process,* **2)** *a novel method for summarizing (modeling) clusters found in streams (as correlated distributions).*

The algorithm's practical contribution is: *an effective method for detecting and analyzing hidden contexts in a stream, while accounting for context drift.*

The remainder of the paper is organized as follows: In Section 2, we review related work. In Section 3, the notations and problem definition are presented. In Section 4, the core pcStream algorithm and its components are presented. In Section 5, the pcStream algorithm is evaluated as an unsupervised context detection algorithm in comparison to state-of-the-art stream clustering algorithms, and in Section 6 we present our conclusion.

2 Related Work

As opposed to regular clustering algorithms, data-stream clustering algorithms must summarize the data seen in order to preserve memory. CluStream [1] accomplishes this by summarizing the observations into micro-clusters using a tuple of three components (called CF) which describes the micro-cluster's centroid, radius and diameter, which can be updated incrementally. DenStream [8] is a density-based stream clustering algorithm. It uses the CF form to determine whether a group of micro-clusters are a legitimate cluster or a collection of outliers. D-Stream [10] also performs density-based stream clustering, but across a grid.

In contrast to the aforementioned algorithms, pcStream summarizes its clusters with the mean and principal components (vectors of highest variance) of the cluster's last observations (discussed in Section 4). Moreover, none of these algorithms consider the temporal relation between arriving points while making clustering decisions. This makes it difficult to discern between two overlapping concepts and a concept drift. Lastly, they do not cluster a stream as if it were an entity transitioning between concepts. Tracking the stream from this perspective assists in the detection of outliers and new contexts.

In order to assign points to clusters, pcStream uses the Soft Independent Modelling by Class Analogy method (SIMCA) [27] to calculate similarity scores. SIMCA, popular in the domain of chemometrics, is a statistical method for the supervised classification of instances. The classification is “soft” in that it offers fuzzy classifications. Concretely, new instances may be classified as members of one or more classes, or even an outlier, based on their Mahalanobis distance from each of the class's distributions. Only the subspace which describes most of the distribution's variance is retained for this calculation. SIMCA performs well on classes which have distinctly different correlated distributions in multidimensional space [19].

As far as we know, SIMCA has not been used on unbounded streams, nor has it been used as an unsupervised clustering method. Moreover, we have not seen any work where SIMCA has been used to dynamically detect new classes (in our case contexts). Lastly, in contrast to SIMCA, we leave the statistical threshold open to help detect contexts of different categories (discussed later in Section 4).

3 Notation and Problem Definition

In this section we define the notation and basic concepts used in this paper. We also provide a formal problem definition. A full summary of this paper’s notations can be found in Table 1.

Definition 1. Let a *context space* be defined as the geometrical space $\mathbb{R}^n$, where n is the number of attributes which define the stream. For instance, one dimension may be the y-axis

readings of a smartphone's accelerometer, while another may be the beats per second (bps) of the smartphone's user. This definition is similar to Context Space Theory (CST), formally proposed in [21].

Definition 2. Let a *stream* S be defined as an unbounded sequence of data objects having the form of points in $\mathbb{R}^n$, and let $\vec{x}_i \equiv [x_{1,i}, x_{2,i}, \dots, x_{n,i}]$ be the i-th point in the sequence. S can also be viewed as a matrix having n columns and an unbounded number of rows, where row i represents the values sampled at time tick i. We use the notation t to denote the current time tick and the notation $\vec{x}_t$ to refer to the most recent point received from S. Let $f_{a,S}$ be the arrival rate of the row vectors in S measured in Hz.

Definition 3. We define a *high velocity stream* as a *stream* which has an arrival rate that is faster than the stream clustering algorithm's processing rate of new arrivals (f_p). More formally, when $f_{a,S} > f_p$ then S is called a *high velocity stream.*

Definition 4. Let c be a *context* (i.e. concept) defined as a cluster of sequential points having a correlated distribution in $\mathbb{R}^n$, in which S exists within for at least t_{min} time ticks at a time. The distribution of c is generally stationary, but may change gradually over time as it is subjected to concept drift [11]. For instance, with the accelerometer data of a user's smartphone, the *context* which captures the action of jumping may change as the user gets older or sicker. We use the notation c_t to refer to the current context of S.

Definition 5. We define a *contextual stream* to be a *stream* that captures the temporal contexts of a real-world entity. More formally, S is a *contextual stream* if S travels among a finite number of distinct contexts, staying at each for at least d time ticks per visit. The property of revisiting certain distributions is known as a reoccurring drift or reoccurring concepts [17, 28].

Let $\boldsymbol{C}$ be the finite collection of known contexts in which S has been found, such that $c_i \in \boldsymbol{C}$ is the i-th discovered *context*. Let $|\boldsymbol{C}|$ denote the number of known contexts.

It is important to note that $\boldsymbol{C}$ does not necessarily form a distinct partition of $\mathbb{R}^n$. As mentioned earlier, *contexts* are fuzzy by nature and it is possible that two identical points $\vec{x}_a$ and $\vec{x}_b$ belong to two distinctly different contexts c_i and c_j.

For the duration of this paper we will only consider *contextual streams*.

Definition 6. We define a *context category* as all contexts from a *contextual stream* that have the same t_{min}, rate of concept drift, and distinction between their distributions.

Problem Definition. Given the *contextual stream* S, a target *context category* and a limited memory space, dynamically detect the finite number of *contexts* exhibited by S, determine the *current context* (c_t) to some degree of certainty, and provide a fuzzy membership score for $\vec{x}_t$ at any time.

4 Principal Component Stream Clustering

4.1 The Context Model

Since we define *contexts* as correlated distributions in $\mathbb{R}^n$, we model the *contexts* using principal component analysis (PCA) [16]. PCA captures the relationship of the correlation between the dimensions of a collection of observations stored in the $m \times n$

matrix X, where m is the number of observations. The result of performing PCA on X are two $n \times n$ matrices; the diagonal matrix V (the Eigen-values) and the orthonormal matrix P (the Eigen-vectors, a.k.a. *principal components*). The Eigen-vectors $\vec{p}_1, \vec{p}_2, \ldots, \vec{p}_n$ form a basis in $\mathbb{R}^n$ centered on X and oriented according to the correlation of X (see Fig. 2). The Eigen-values $\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2$ are the variances of the data in the direction of their respective Eigen-vectors. The Eigen-values of V are sorted from highest to lowest variance and the respective Eigen-vectors in P are ordered accordingly. In other words, from the mean of the collection X, $\vec{p}_1$ is the direction of highest variance in the data (with σ_1^2).

We define the contribution of component $\vec{p}_i$ as the percent of total variance it describes for the collection X. More formally,

$$cont_X(\vec{p}_i) = \frac{\sigma_i^2}{\sum_{j=1}^{n} \sigma_j^2} \tag{1}$$

PCA has been widely used to reduce the dimensionality of a dataset while preserving the information it holds [23]. This is accomplished by projecting observations onto the top *principal components* (PCs). Typically, most of a collection's variance is captured by just a few PCs. By retaining only these PCs, we effectively summarize the distribution and focus our future calculations on the dimensions of interest.

Let ρ be the target percent of variance to be retained. We define $k \in \mathbb{N}$ to be the fewest, most influential PCs in which their cumulative sum of contributions surpasses ρ. Stated otherwise, $\arg\min_k \{\sum_{i=1}^{k} cont_X(\vec{p}_i) \geq \rho\}$. We denote the k associated with *context* c_i as k_{c_i}.

We model the i-th discoverd *context* as the tuple $c_i \equiv \langle M_i, \mu_i, A_i \rangle$, where M_i is a $m \times n$ for the last m observations assigned to c_i, μ_i is the mean of the observations in M_i, and $A_i = \left[\frac{\vec{p}_1}{\sigma_1}, \frac{\vec{p}_2}{\sigma_2}, \ldots, \frac{\vec{p}_{k_{c_i}}}{\sigma_{k_{c_i}}}\right]$ is a $n \times k_{c_i}$ transformation matrix.

A_i is essentially a truncated version of P with its Eigen-vectors normalized to their standard deviations (SD). A_i can be calculated by first performing PCA on M_i, to get P_i and V_i, and then by calculating $A_i = Q_i \Lambda_i$ where Λ_i is a diagonal matrix of the top k_{c_i} largest SDs (obtained from V_i), and Q_i is the column-wise truncation of P_i so that it only includes the first k_{c_i} columns. The significance of the transformation matrix A_i will be detailed later in the paper.

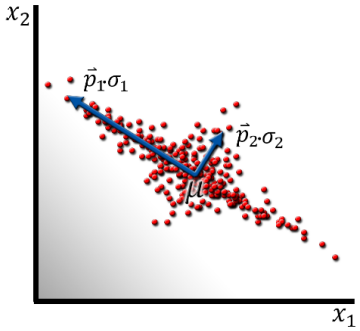

Fig. 2. A visualization of the principal components ($\vec{p}_1, \vec{p}_2$) scaled to their respective standard deviations (σ_1, σ_2) and centered on the distribution's mean (μ)

Matrix M_i acts as a windowed memory for c_i by discarding the m^{th} oldest observation when a new one is added. Windowing over a stream is an implicit method for dealing with concept drift [3, 24].

4.2 Merging Models

There are cases where the parameters of pcStream will create too many models for the memory space of the host system. Concretely, in the worst case scenario ($k_{c_i} = n$) the maximum memory required for context model c_i is $\theta(n^2)$ for matrix A_i and $\theta(mn)$ for matrix M_i. In total, the memory space of pcStream would be $\theta\big(|\boldsymbol{C}|(n^2 + mn)\big)$.

To enforce a memory limit, we propose that if a new *context* is discovered though the memory limit has been reached, then two models will be merged into one to make space for the new model. To select which models to merge, we will follow the method used in CluStream where the cluster with the oldest average timestamp (among its observations) is merged with its closest neighbor [1]. Let the merge operation be defined as

$$merge\big(c_i, c_j\big) = c_l \tag{2}$$

where c_l is a context model generated from merging the models c_i and c_j. In essence, c_l is created by generating M_l from M_i and M_j, and then calculating μ_l and A_l from M_l. Since we store the last m observations of each model's history, (2) should take into account the temporal history of the observations.

There are at least two ways of accomplishing this. One way is to evenly interleave the first m observations of M_i and M_j into M_l. This will ensure that at least half of each context's information will be preserved. Another way is to merge the memories in order of timestamp. Doing so will place emphasis on retaining the most recent knowledge. The complexity of performing either method is $O(m)$. The selection of which method to use is dependent on the application of pcStream. For example, anomaly detection versus situational awareness.

4.3 Similarity Scores

When a new observation $\vec{x}_t$ arrives, we must determine to what degree it belongs to each known context $c_i \in \boldsymbol{C}$. As mentioned earlier, contexts are fuzzy in their membership, and the point $\vec{x}_t$ can belong to multiple contexts at once. Therefore, we must compute the similarity score of the point in question to each known context.

We calculate the similarity score the same way the SIMCA method does. Therefore the point's statistical similarity to a distribution is calculated as the Mahalanobis distance using only the top k PCs of that distribution. Equivocally, they produce this score by first zero-meaning the point to that distribution, then transforming it onto the distribution's top k PCs, and then by computing the resulting point's magnitude [19]. More formally, the similarity of point $\vec{x}$ to the distribution of c_i is

$$d_{c_i}(\vec{x}) = \|(\vec{x} - \mu_i)A_i\| \tag{3}$$

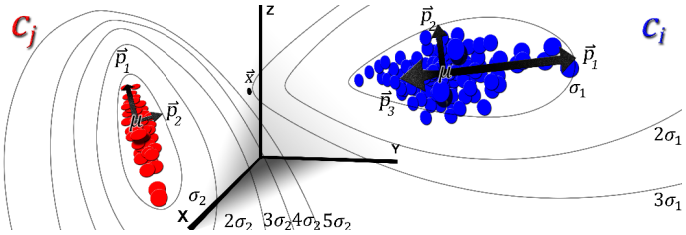

Fig. 3. An illustration of how the ellipsoids of the Mahalanobis distance affect the score of a point ($\vec{x}$) calculated in the perspective of the contexts c_i and c_j. Here $d_{c_i}(\vec{x}) < d_{c_j}(\vec{x})$.

Intuitively, the contours of performing Mahalanobis distance can be seen as ellipsoid shapes extending from the distribution, according to the variance in each direction. To strengthen the concept further, Fig. 3 illustrates an example where two contexts (c_i and c_j) are compared in similarity to point $\vec{x}$. From the perspective of the top k_{c_i} and k_{c_j} normalized PCs of each respective context, their similarity scores differ (even though the Euclidean distances between $\vec{x}$ and both μ_i and μ_j are equivalent). Furthermore, it is possible that the $k_{c_i} > k_{c_j}$ depending on ρ and the correlation of the distribution.

Let $\varphi \in [0, \infty)$ be defined as the similarity threshold. We say that a point $\vec{x}$ is not similar to *context* c_i if $d_{c_i}(\vec{x}) > \varphi$. Let $d_{\boldsymbol{C}}(\vec{x}) = \left[d_{c_1}(\vec{x}), d_{c_2}(\vec{x}), \ldots, d_{c_{|\boldsymbol{C}|}}(\vec{x})\right]$ be defined as the score vector (fuzzy membership) of $\vec{x}$ to each of the models in $\boldsymbol{C}$. We say that $\vec{x}$ does not fit any known context if all elements in $d_{\boldsymbol{C}}(\vec{x})$ are greater φ. Moreover, we say that $\vec{x}$ is most similar to *context* c_i if the smallest element in $d_{\boldsymbol{C}}(\vec{x})$ is $d_{c_i}(\vec{x})$.

4.4 Detection of New Contexts

A critical part of the pcStream algorithm is to detect when a previously unseen context has appeared. From Definition 4, contexts are assumed to have rather stationary distributions. Therefore, a new context is detected when the data distribution of S no longer fits the contexts in $\boldsymbol{C}$ for a consistent t_{min} time ticks. At this point, we say that these t_{min} observations constitute a new context, and are then modeled for future use.

To track this behavior, we introduce a new concept called a "drift buffer." Let the drift buffer be called D and have a length of t_{min}. Should D ever be filled without interruption (i.e. should $D = \{\vec{x}_{t-t_{min}-1}, \ldots, \vec{x}_t\}$) then the content of D is emptied to create a new context model, and is set as current context c_t. However, in the case of a partial drift (i.e. D did not get filled, yet $\vec{x}_t$ fits some context in $\boldsymbol{C}$) we assume that S is experiencing a wider boundary of c_t, therefore we empty D into c_t.

Table 1. Summary of theory and algorithm notations

Theory Notations		Algorithm Notations	
Nota.	**Description**	**Nota.**	**Description**
n	The number of attributes in the stream	$\vec{p}_i, \sigma_i$	The i-th principal component of collection X and its corresponding variance
$\mathbb{R}^n$	Context Space: The geometric space in which the stream exists, defined by the number of sensors	$cont_X(\vec{p}_i)$	The percentage of total variance component $\vec{p}_i$ describes in collection X
S	Stream: an $\infty \times n$ matrix representing an unbounded sequence of row vectors	ρ	The target percent of total variance to preserve from each distribution in $\boldsymbol{C}$
t	The row index to an arbitrary time tick in S	k_{c_i}	The number of the fewest top principal components of c_i needed to obtain at least ρ of the variance in the distribution of c_i
$\vec{x}_t$	The samples receives by S at time tick t, represented as the t-th row in R: $[x_{1,t}, x_{2,t}, \dots, x_{n,t}]$	$d_{c_i}(\vec{x})$	The distance in standard deviations from point $\vec{x}$ to the distribution of c_i measured using the top k principal components of that distribution
$\vec{x}_c$	The current (last received) vector from S	m	Model Memory Parameter: the maximum number of observations a model c_i may store
$f_{a,S}$	The arrival rate of data objects to S, measured in Hz	M_i	Model Memory: a $m \times n$ matrix of the last m observations assigned to context c_i
f_p	The respective algorithm's processing rate, measured in Hz	μ_i	The Mean of a Context: the mean of the observations in M_i
c_i	Context: The i-th discovered context from S. c_i is a correlated distribution in $\mathbb{R}^n$ modeled as the tuple $c_i \equiv \langle M_i, \mu_i, A_i \rangle$	A_i	The Context Transformation Matrix: a $n \times k_{c_i}$ matrix which translates zero-meaned points to context c_i's distribution by using the top k_{c_i} st.d.-normalized principal components of M_i
$\boldsymbol{C}$	The ordered collection of all known contexts (c_i) exhibited by S. The size of $\boldsymbol{C}$ is denoted as $\lvert\boldsymbol{C}\rvert$	φ	The Sensitivity Threshold: the distance $d_{c_i}(\vec{x}_t)$ after which $\vec{x}_t$ is not considered to be similar to c_i
t_{min}	The minimum context drift size: the number of time ticks S must stay in a new data distribution (distinct from all those in $\boldsymbol{C}$) to be considered a new context	D	Drift Buffer: a buffer with at most t_{min} consecutive points that do not fit the contexts in $\boldsymbol{C}$

4.5 The Core pcStream Algorithm

The basic approach of the core pcStream algorithm is to follow the *stream*'s data distribution. Fuzzy membership scores are available anytime by calculating the statistical similarities between a point and each known context. As long as the arriving points stay within a distribution of a known context, we assign them to that context. The moment the *stream*'s distribution does not fit any known context, we define a new one. Each of the concepts has a window of memory to allow for concept drifts. Should the allocated memory space be filled, then one of two methods for merging context models is performed. Point anomalies are detected as short-term drifts away from all known contexts. Lastly, different context categories are to be detected by adjusting the algorithm's parameters accordingly. The parameters for pcStream are: the sensitivity threshold φ, the context drift size t_{min}, the model memory size m, and the percent of variance to retain in projections ρ. The pseudo-code for pcStream can be found in Algorithm 1.

On lines 1-3, pcStream is initialized by creating the initial collection $\boldsymbol{C}$ with context c_1, and then by setting the current context (c_t) accordingly. The function $init(S, t_{min}, m, \rho)$ runs the function $CreateModel(X, m, \rho)$ on the first t_{min} points of S. The function $CreateModel(X, m, \rho)$ returns a new *context model* c by using the collection of observations X and target total variance retention percentage ρ. Remember that the memory of a *context model* M is a window (FIFO buffer) with a maximum length of m (forgetting the oldest observations). Optionally, an initial set of models for $\boldsymbol{C}$ can be made from a set of observations pre-classified as known contexts of S (e.g. a collection of points that capture

running and another that captures walking). From this point on, pcStream enters its running state (lines 4-5).

On lines 4.1-4.3, point $\vec{x}_c$ arrives and $\vec{x}_c$'s similarity score is calculated for all known contexts in $\boldsymbol{C}$. Stored in i is the index to the model in $\boldsymbol{C}$ to whom $\vec{x}_c$ is most similar. Reminder, the index of $\boldsymbol{C}$ is chronological by order of discovery.

On line 4.4 we determine whether $\vec{x}_c$ fits any of the contexts in $\boldsymbol{C}$. If it does, then we proceed to lines 4.4.1-4.4.3 where we update the model of best fit (c_i) with instance $\vec{x}_c$, and update c_t accordingly. Since this breaks any consistent drift (between contexts) we empty the drift buffer D into c_i as well (line 4.4.1). The function $UpdateModel(c_i, X)$ re-computes the tuple c_i from $\boldsymbol{C}$ after adding the observation(s) X to the FIFO memory M_i.

If the check on line 4.4 indicates that $\vec{x}_t$ does not any context in $\boldsymbol{C}$, then we add $\vec{x}_t$ to the drift buffer D, and subsequently check if D is full. If D has reached capacity (t_{min}) then unseen *context* has been discovered. In which case, D is then emptied and formed into a new *context model* (c), which is added to $\boldsymbol{C}$ and set as c_t. The function $AddModel(c, \boldsymbol{C})$ adds c to $\boldsymbol{C}$ as $c_{|\boldsymbol{C}|+1}$. If the additional model is too much for the memory space allocated to pcStream, then the function $merge(c_i, c_j)$ is used to free one space for c (in $\boldsymbol{C}$) by merging the average oldest context model c_i with its nearest context model c_j.

Online Algorithm 1: pcStream $\{S\}$
Input Parameters $\{\varphi, t_{min}, m, \rho\}$
Anytime Outputs: $\{c_t, d_{\boldsymbol{C}}(\vec{x}_t)\}$
1. $\boldsymbol{C} \leftarrow init(S, t_{min}, m, \rho)$
2. $c_t \leftarrow c_1$
3. $D \leftarrow \emptyset$
4. *loop*
   - 4.1. $\vec{x}_c \leftarrow next(S)$
   - 4.2. $scores \leftarrow d_{\boldsymbol{C}}(\vec{x}_c)$
   - 4.3. $i \leftarrow IndexOfMin(scores)$
   - 4.4. *if* $scores(i) < \varphi$
     - 4.4.1. $UpdateModel(c_i,\ Dump(D))$
     - 4.4.2. $UpdateModel(c_i, \vec{x}_c)$
     - 4.4.3. $c_t \leftarrow c_i$
   - 4.5. *else*
     - 4.5.1. $Insert(\vec{x}_c, D)$
     - 4.5.2. *if* $length(D) == t_{min}$
       - 4.5.2.1. $c \leftarrow CreateModel(Dump(D), m, \rho)$
       - 4.5.2.2. $AddModel(c, \boldsymbol{C})$
       - 4.5.2.3. $c_t \leftarrow c$
5. *end loop*

4.6 The Algorithm's Computational Cost

In dealing with a streaming algorithm, it is important to understand pcStream's processing rate (f_p) as it relates to the *stream*'s arrival rate (f_a). From a complexity standpoint, there are two calculations that occur on the arrival of $\vec{x}_c$. The first is the calculation of all similarity scores $d_{\boldsymbol{C}}(\vec{x}_c)$ (line 4.2) and the second is a PCA calculation (performed in either $CreateModel$ or $UpdateModel$). The complexity of calculating the similarity scores is $|\boldsymbol{C}|O(nk)$. Typically $k \ll n$ since most of a model's variance is captured on a few PCs

(we found k to range from 1 to 3 per context in our evaluations) making the complexity $O(n)$.

The complexity of performing the PCA calculation using a naïve method is $O(mn^2)$ [12]. It may seem that such a high exponential complexity is unsuitable for a streaming environment since f_p must keep up with f_a. However, for many applications (such as context awareness on a smartphone), n will be in the order of tens and m will typically be in the scale of about a thousand. Therefore, the time it takes to perform the PCA is acceptable in the sense of practicality.

Together, the two calculations that occur at each time tick have a rather linear complexity. Using a dataset with an n of 3, the average f_p for **pcStream**, D-Stream, DenStream, and CluStream was **0.4**, 2.9, 44.6, 240.1 milliseconds respectively (R-Studio on a single core of an Intel i5 processor). Moreover, with an n of 561, pcStream's f_p was 2ms. Therefore, pcStream is a practical stream clustering algorithm.

For applications where n and / or m are very large, or where S is a *high velocity stream*, we offer an addition to the core pcStream algorithm (described above) to help f_p maintain a speed at least as fast as f_a. The addition entails decoupling the online calculations (similarity scoring) from the offline calculations (model upkeep) by placing model *update*, *create*, and *merge* procedures into a priority queue (the operation with the highest priority is *merge*, followed by *create* and then *update*). Whenever a point is assigned to context c_i, it is added to the waiting list for c_i's update operation in the queue (if not already in the queue then the operation is added). Lastly, when it is c_i's update operation's turn, the operation's entire waiting list is added chronologically to M_i and $UpdateModel(c_i, X)$ is performed. This operation essentially performs a mini-batch update on the context model.

This addition makes f_p dependent on the score calculations (negligible in respect to the PCA operations) and therefore enables pcStream to be scaled to many more applications. Furthermore, it is possible to parallelize the offline computations over a multiple threads or networked clusters.

4.7 The Detection of Different Context Categories

Each selection of the parameters φ, t_{min}, and m changes pcStream's perspective on S. This essentially causes pcStream to focus on all contexts belonging to a single *context category*, where φ is the degree of distinction between contexts, and m is the rate of concept drift (see Definition 6).

Concretely, a *small* φ will give pcStream the perspective for indistinct contexts (i.e. small nuances) while a *large* φ will give the perspective for ones that are more unique. Similarly, a *small* t_{min} gives the perspective for short term-contexts as opposed to more long-term ones, and a *small* m, gives the perspective for sudden concept drifts as opposed to more gradual. Table 2 summarizes the impact that the pcStream's parameters have on the perspective *context category*.

Intuitively, multiple *context categories* are in a sensor stream at any given time. For example, at a given moment, a smartphone's accelerometer can capture the context whether a user is "*awake or asleep*", "*running or walking*", and "*running to catch the bus,* or *running for sports*". Therefore, should one be interested in different context categories at the same time, multiple instances of pcStream should be run in parallel with the respective settings.

5 Evaluation

5.1 Datasets and Test Platform

The pcStream algorithm was run in MATLAB, and the evaluation of the various other data-stream clustering algorithms were run in R (a software environment for statistical computing). The packages we used in R were stream and streamMOA [6, 7]. The streamMOA package provided an interface to algorithms implemented for the MOA (Massive Online Analysis) framework for data stream mining [7].

We used three datasets; each one for evaluating a different aspect. The first was the human activity recognition dataset (HAR) [2] for evaluating pcStream in large dimensional spaces. The second was KDD'99 network intrusion dataset [4], selected for testing the effect that mini-batch updates have on *high velocity streams*. Lastly, the third was the HearO smartphone sensor dataset [25] selected for evaluating pcStream's ability to detect different *context categories*. The HearO dataset is a sensor-fused dataset obtained from smartphones. What makes it unique is its explicit context labels provided by the smartphone user at various times over several months. A summarized description of these datasets can be found in Table 3.

Table 2. The effect pcStream's parameter selection has on detecting contexts

		Buffer Size (t_{min})	
		Small	Large
Threshold (φ)	Small	*Short-term* indistinct contexts	*Long-term* indistinct contexts
	Large	*Short-term* distinct contexts	*Long-term* distinct contexts

Table 3. A description of the three datasets used for pcStream's evaluation

Dataset	n	*Examples of* n	*# of rows*	*Context groups*	*# labels*	*Examples of labels*
HAR	561	Accl. (x,y,z), FFT…	347	Motion Activity	6	sitting, walking, going upstairs
HearO	5	Accl. correlation (xy,xz,yz), device temperature, battery level.	1764	High-level	4	at home, at work, on break
				Low-level	9	hungry, interested, shopping
				Phone Plugged In	2	yes, no
KDD'99	38	src_bytes, dst_bytes, serror_rate	494,020	Network Attacks	23	buffer overflow, rootkit, teardrop

5.2 Clustering Performance

In the domain of stream clustering, it is common to evaluate the clustering quality by measuring the sum of square distances (SSQ) of every point in a cluster to its cluster's median. However, this metric assumes that clusters do not overlap, and is therefore not suitable for our clustering problem. Therefore, we use the Adjusted Rand Index (ARI) [15].

The ARI is a measure of similarity between two data clustering assignments regardless of their spatial qualities. In our case, we measure an algorithm's performance (of detecting contexts in a dataset) by calculating the ARI between the algorithm's clustering assignment and the dataset's context labels.

The following was performed to evaluate an algorithm's clustering performance with regard to detecting the hidden contexts of streams. First, the dataset was clustered as a stream. Afterward, the ARI of the resulting cluster assignments was calculated using the

context labels provided by the dataset. Finally, the clustering performance was recorded as the highest ARI achieved across all possible input parameters of the respective algorithm (achieved with a moderate brute-force search). This was done for each algorithm on both HAR and HearO datasets (note that the HearO dataset has three separate categories of context labels). The algorithm's clustering performances on hidden contexts can be found in Fig. 4. The "Window" is a k-median window algorithm outlined in [3].

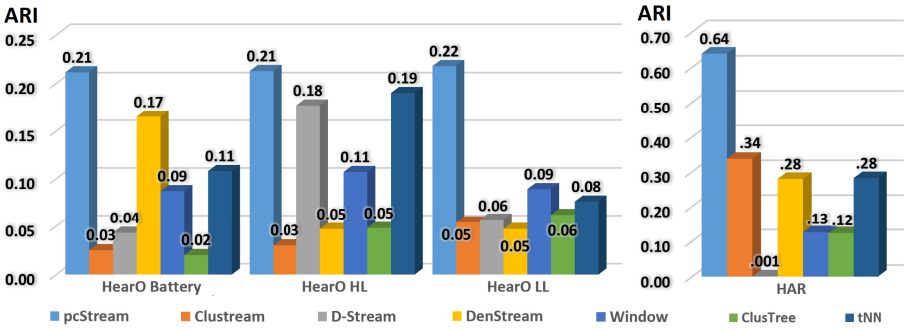

Fig. 4. A comparison of various stream clustering algorithms' best performances against pcStream

Lastly, we tested pcStream's clustering performance in the case of a *high velocity stream*. To measure the impact, we simulated various arrival-rates (f_a) over the KDD'99 dataset. In Fig. 5, there is a performance plot which shows how the clustering is affected as the stream's velocity increases. Noisy spikes are caused because we set a small model memory (m=500), and the duration of most attacks in the dataset last in the order of thousands.

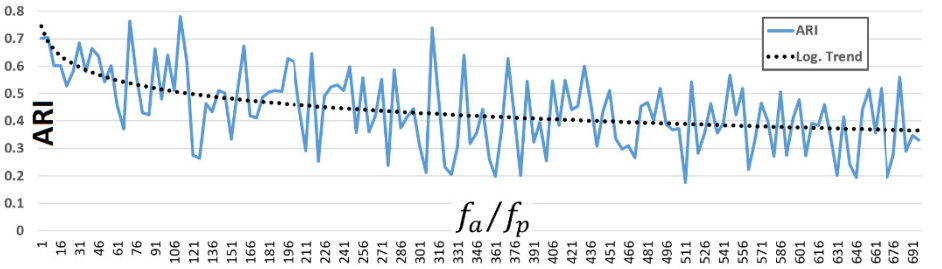

Fig. 5. The performance of pcStream when using mini-batch model updates. Each point is a full pass over the KDD dataset where the on the x-axis indicates the arrival rate of instances per mini-batch update.

5.3 Context Categories

One application of pcStream is to use it as a way to analyze a *contextual stream*. In particular, pcStream can be used to see what hidden contexts are in a stream for some *context category*. Therefore, the following was done in order to analyze how well pcStream detects different *context categories*: First pcStream was run many times on

the HearO dataset, each time targeting a different *context category* by using a different φ and t_{min} (for simplicity we set m to be very large for all trials). Then, for each of the three context groups (Plugged-In, High-level, and Low-Level), a plot was made where each coordinate's color in the plot indicates the ARI achieved when using the respective parameters (see Fig. 6).

It can be seen in Fig. 6 that each plot describes the context categories of the contexts in each respective group. For instance, these high-level contexts are generally long-term and semi-distinctive, while the low-level contexts are briefer and even more indistinct. Moreover, the context of the phone being plugged in or not is a distinct and short-term context. Note that there are multiple "hot" regions in each plot of Fig. 6. This is because each group has multiple contexts, each of which belong to a different context category.

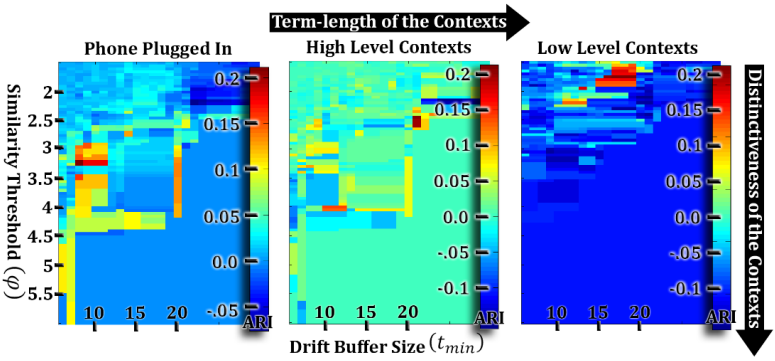

Fig. 6. ARI plots of pcStream for each HearO context group. A coordinate is a parameter selection and its color indicates the level of ARI achieved. The black arrows indicate the general trend of *context categories* based on Table 2. The labels of each context group form regions of "best parameter selections" correlated to the label's *context category*.

6 Conclusion

In this paper we have presented a stream clustering algorithm that effectively and dynamically detects temporal contexts in sensor streams. In addition, we have provided mechanisms which account for gradual concept drifts, reoccurring concepts, and clusters that overlap in geometrical space. Moreover, we have provided a mechanism for dealing with *high velocity streams*.

In our evaluations, we have determined that pcStream performs much better than other data-stream clustering algorithms on sensor-fused datasets. We have demonstrated that pcStream is capable of detecting different *context categories* from the same dataset, and that pcStream is a useful tool for context analysis of sensor-streams. Although the focus of this paper was on sensor-fused data-streams and the application of context-awareness, pcStream is applicable to any data-stream with sequential temporal clusters that have unique correlated distributions.

Acknowledgments. This research was supported by the Ministry of Science and Technology, Israel.

References

1. Aggarwal, C.C., et al.: A framework for clustering evolving data streams. In: Proceedings of the 29th International Conference on Very Large Data Bases vol. 29, pp. 81–92. VLDB Endowment (2003)
2. Anguita, D., Ghio, A., Oneto, L., Parra, X., Reyes-Ortiz, J.L.: Human Activity Recognition on Smartphones Using a Multiclass Hardware-Friendly Support Vector Machine. In: Bravo, J., Hervás, R., Rodríguez, M. (eds.) IWAAL 2012. LNCS, vol. 7657, pp. 216–223. Springer, Heidelberg (2012)
3. Babcock, B., et al.: Maintaining variance and k-medians over data stream windows. In: Proceedings of the Twenty-Second ACM Symposium On Principles Of Database Systems, pp. 234–243. ACM (2003)
4. Bache, K., Lichman, M.: UCI Machine Learning Repository (2013). http://archive.ics.uci.edu/ml
5. Baldauf, M., et al.: A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing. **2**(4), 263–277 (2007)
6. Bolanos, M., et al.: Introduction to stream: An extensible Framework for Data Stream Clustering Research with R
7. Bolanos, M., et al.: streamMOA: Interface to Algorithms from MOA for stream
8. Cao, F., et al.: Density-based clustering over an evolving data stream with noise. In: SDM, pp. 326–337 SIAM (2006)
9. Chandola, V., et al.: Anomaly Detection: A Survey. ACM Comput. Surv. **41**(3), 1–58 (2009)
10. Chen, Y., Tu, L.: Density-based clustering for real-time stream data. In: Proceedings of the 13th ACM SIGKDD International Conference On Knowledge Discovery And Data Mining, pp. 133–142. ACM (2007)
11. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with Drift Detection. In: Bazzan, A.L., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171, pp. 286–295. Springer, Heidelberg (2004)
12. Ge, Z., Song, Z.: Multivariate Statistical Process Control: Process Monitoring Methods and Applications. Springer (2012)
13. Gomes, J.B., et al.: CALDS: context-aware learning from data streams. In: Proceedings of the First International Workshop on Novel Data Stream Pattern Mining Techniques, pp. 16–24. ACM, Washington, D.C. (2010)
14. Harries, M.B., et al.: Extracting hidden context. Machine learning. **32**(2), 101–126 (1998)
15. Hubert, L., Arabie, P.: Comparing partitions. Journal of classification. **2**(1), 193–218 (1985)
16. Jolliffe, I.: Principal Component Analysis. Encyclopedia of Statistics in Behavioral Science. John Wiley & Sons, Ltd (2005)
17. Katakis, I., et al.: Tracking recurring contexts using ensemble classifiers: an application to email filtering. Knowledge and Information Systems. **22**(3), 371–391 (2010)
18. Liu, W., et al.: A survey on context awareness. In: 2011 International Conference on Computer Science and Service System (CSSS), pp. 144–147. IEEE (2011)
19. Maesschalck, R.D., et al.: The Mahalanobis distance. Chemometrics and Intelligent Laboratory Systems. **50**(1), 1–18 (2000)
20. Makris, P., et al.: A Survey on Context-Aware Mobile and Wireless Networking: On Networking and Computing Environments' Integration. Communications Surveys & Tutorials, IEEE. **15**(1), 362–386 (2013)

21. Padovitz, A., et al.: Towards a theory of context spaces. In: 2004, Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, pp. 38–42. IEEE (2004)
22. Riboni, D., Bettini, C.: COSAR: hybrid reasoning for context-aware activity recognition. Personal and Ubiquitous Computing. **15**(3), 271–289 (2011)
23. Shlens, J.: A tutorial on principal component analysis. arXiv preprint arXiv:1404.1100 (2014)
24. Silva, J.A., et al.: Data Stream Clustering: A Survey. ACM Comput. Surv. **46**(1), 1–31 (2013)
25. Unger, M., et al.: Contexto: lessons learned from mobile context inference. In: ACM 2014 International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication, pp. 175–178. ACM (2014)
26. Widmer, G.: Tracking context changes through meta-learning. Machine Learning. **27**(3), 259–286 (1997)
27. Wold, S., Sjostrom, M.: SIMCA: a method for analyzing chemical data in terms of similarity and analogy. Presented at the (1977)
28. Yang, Y., et al.: Mining in Anticipation for Concept Change: Proactive-Reactive Prediction in Data Streams. Data Mining and Knowledge Discovery. **13**(3), 261–289 (2006)