Enhanced Situation Space Mining for Data Streams

Yisroel Mirsky
Ben-Gurion University
Beer-Sheva, Israel
yisroel@post.bgu.ac.il

Tal Halpern
Tel-Aviv University
Tel-Aviv, Israel
talhalpern10@gmail.com

Rishabh Upadhyay
Fr. Conceicao Rodrigues
College of Engineering
Mumbai, India
uhrishabh@gmail.com

Sivan Toledo
Tel-Aviv University
Tel-Aviv, Israel
stoledo@tau.ac.il

ABSTRACT

Data streams can capture the situation which an actor is experiencing. Knowledge of the present situation is highly beneficial for a wide range of applications. An algorithm called pcStream can be used to extract situations from a numerical data stream in an unsupervised manner. Although pcStream outperforms other stream clustering algorithms at this task, pcStream has two major flaws. The first is its complexity due to continuously performing principal component analysis (PCA). The second is its difficulty in detecting emerging situations whose distributions overlap in the same feature space.

In this paper we introduce pcStream2, a variant of pcStream which employs windowing and persistence in order to distinguish between emerging overlapping concepts. We also propose the use of incremental PCA (IPCA) to reduce the overall complexity and memory requirements of the algorithm. Although any IPCA algorithm can be used, we use a novel IPCA algorithm called Just-In-Time PCA which is better suited for processing streams. JIT-PCA makes intelligent 'short cuts' in order to reduce computations. We provide experimental results on real-world datasets that demonstrates how the proposed improvements make pcStream2 a more accurate and practical tool for situation space mining.

Keywords

Data stream; data mining; context space theory

1. INTRODUCTION

The ubiquity of sensors can be found in a wide range of systems. For example, the modern smartphone contains sensors that can be used to record the device's acceleration or even CPU utilization. These sensors can be used to implicitly infer an actor's situation.

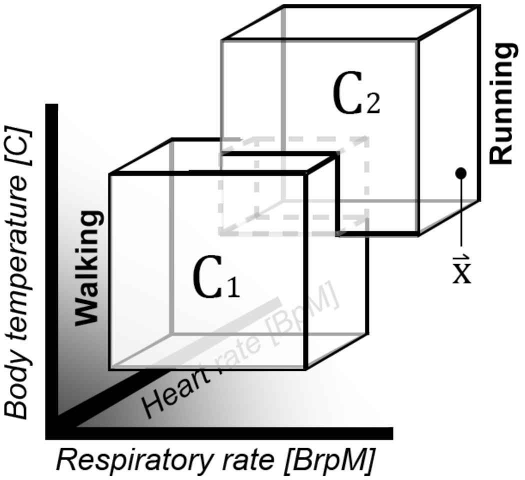

Figure 1: An illustration of a *context domain* for activity recognition consisting of two *situation spaces*: walking and running (c_1 and c_2).

For example, a stream of accelerometer x-y-z values from a smartphone can be used to infer whether a user is walking or running [1], and a stream of CPU/memory utilization values can be used to infer the malicious intent of some malware [2]. To infer an actor's situation from sensor values, one may apply Context Space Theory (CST).

CST is an approach of modeling an entity's context using geometric representations called *situation spaces*. CST was first introduced by Padovitz et al. in [3] and since has been used in a wide variety of context-aware applications [4, 5, 6]. CST defines a context to be a point in an n-dimensional space (referred to as the *context space*), where each axis captures a sensor's output, and each of the n sensors are relevant to capturing the situation of the actor. A *situation space* is a defined region in a context space. If a context is said to belong to this region, then it is said that the actor is experiencing the respective situation. A set of situation spaces for a particular context space is refereed to as a *context domain*. Figure 1 illustrates an example *context domain* where the point $\vec{x}$ is the actor's current context, defined by the values captured via the body temperature, respiratory rate, and heart rate sensors. There, the actor's current situation is c_2 (Running).

SAC 2017,April 03 - 07, 2017, Marrakech, Morocco
Copyright 2017 ACM 978-1-4503-4486-9/17/04…$15.00
http://dx.doi.org/10.1145/3019612.3019671

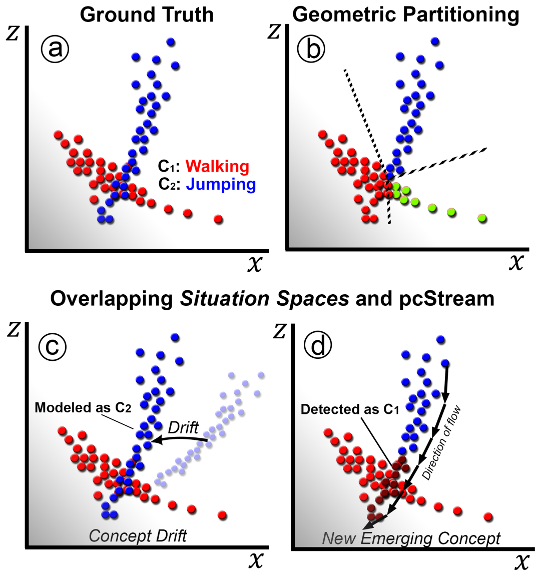

Figure 2: An illustration of the issues with detecting overlapping *situation spaces* from a data stream generated from smartphone accelerometer. Here the ground truth is activity recognition.

The main difficulty with CST is that an expert must define the *situation spaces* of a context space manually. Mining *situation spaces* from data streams is challenging because data streams are (1) potentially unbounded in length, and (2) subject to concept drift (gradual changes in the underlying distributions). Moreover, since *situation spaces* are inherently fuzzy (overlap), common stream clustering algorithms cannot be used. This is because these algorithms seek to find partitions of geometric space and thus do not respect the ground-truth (see Figure 2b).

In order to mine *situation spaces* from data streams, the authors in [7] proposed an algorithm called pcStream. pcStream considers both the temporal and spatial aspects of a data stream in order to dynamically detect, and extract *situation spaces* in an unsupervised manner. The authors showed there how pcStream can accurately mine and detect *situation spaces* and update them in accordance to concept drifts. In another work, the authors show how pcStream can be further extended to detect point, contextual, and collective anomalies in contextual data streams [8].

Briefly, pcStream operates by modeling *situation spaces* as Gaussian distributions while tracking the flow of the stream between known models. When an instance arrives from the stream, it is assigned to the closest model using Mahalanobis distance. Should there be no model close enough then the instance is added to the buffer B of length $t_{\min}$. The purpose of buffer B is to capture the instances that potentially belong to a new *situation space*. pcStream assumes that the actor behind the stream remains in each situation for at least $t_{\min}$ observations at a time. If an observation is ever assigned to an existing model, B is immediately emptied and used to update that model. However, should B ever reach capacity, a new *situation space* is modeled after the contents of B, and B is reset.

Although pcStream extracts *situation spaces* from data streams better than other stream clustering algorithms [7], it suffers from two major drawbacks:

Detecting new overlapping *situation spaces*. The implementation proposed in [7] can model overlapping distributions as a result of concept drift (Figure 2c). However, it cannot properly detect new situation spaces (emerging concepts) which overlap existing ones. This is because the algorithm assigns each instance to the closest model without considering that the instance may still belong to the current situation. Figure 2d illustrates this issue: while the stream is discovering c_2, it briefly cross over the space of c_1. The brevity of the overlap is an indication that the situation was always c_2, however pcStream will assign these overlapping instances to c_1 because c_1 has the smallest Malanobis distance to these overlapping points.

Algorithm Complexity. pcStream relies on using **p**rincipal **c**omponent analysis (PCA) to model each individual situation space. Whenever an instance is a assigned to a model, PCA must be performed to update that model. Since PCA has a complexity of $O(n^3)$, it is clear why pcStream suffers in performance when the stream has more than just a few dimensions.

In this paper we improve the authors' original algorithm (pcStream) and introduce pcStream2. We solve the issue of detecting new overlapping *situation spaces* by updating pcStream's core algorithm to follow a sliding window approach. From Empirical results, we found that pcStream2 is more accurate. Moreover, to address the complexity of performing PCA in pcStream, we propose the use of an incremental PCA (IPCA) algorithm, and describe how it can be incorporated into pcStream. To demonstrate the benefit of incorporating IPCA with pcStream, we use a new IPCA algorithm called *just-in-time PCA* (JIT-PCA). Unlike other IPCA algorithms, JIT-PCA minimizes the number of computations performed on a per-instance basis. Thus JIT-PCA not only reduces the memory footprint of pcStream, but also dramatically improves its runtime as a streaming algorithm. The source code to pcStream2 and JIT-PCA can be downloaded online.[1]

The remainder of the paper is structured as follows: In section 2 we present the relevant notations and the original pcStream algorithm. In section 3 we present our new algorithm pcStream2. In section 4 we present JIT-PCA and describe how to incorporate it into pcStream and pcStream2. In section 5 we present experimental results, and in section 6 we provide our conclusion.

2. PCSTREAM

In this section, we briefly review the basic pcStream algorithm as presented in [7].

2.1 Notations and Definitions

Let a *context space* be defined as the geometrical space $\mathbb{R}^n$, where n is the number of attributes which define the stream. Let a *stream* S be defined as an unbounded sequence of data objects having the form of points in $\mathbb{R}^n$, and let $x_i \equiv [x_{1,i}, x_{2,i}, ..., x_{n,i}]^T$ be the i-th point in the sequence. Let c be a *situation space* (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 it may change gradually over time as it is subjected to concept drift. We use the notation c_t to refer to the current situation of S. We define a *contextual stream* to be a *stream* that captures the temporal situations of a real world actor. More formally,

[1] https://github.com/ymirsky/pcStream.

S is a *contextual stream* if S travels among a finite number of distinct *situation spaces*, staying at each for at least $t_{\min}$ time ticks per visit. The property of revisiting certain distributions is known as a reoccurring drift or reoccurring concepts [9]. Finally, let $\mathbf{C}$ be the *context domain*: the finite collection of known situation spaces found in S, such that $c_i \in \mathbf{C}$ is the i-th discovered *situation space*. Let $|\mathbf{C}|$ denote the number of known situations.

2.2 The Situation Space Model

In [7] the authors define *situation spaces* as correlated distributions in $\mathbb{R}^n$. Thus they model the *situation spaces* using principal component analysis (PCA) [10]. PCA captures the relationship of the correlation between the dimensions of a collection of observations stored in the $n \times m$ 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 Eigenvalues) and the orthonormal matrix P (the Eigenvectors, a.k.a. *principal components*). The Eigenvectors $p_1, p_2, ..., p_n$ form a basis in $\mathbb{R}^n$ centered on X and oriented according to the correlation of X. The Eigenvalues $\sigma_1^2, \sigma_2^2, ..., \sigma_n^2$ are the variances of the data in the direction of their respective Eigenvectors. The eigenvalues of V are sorted from highest to lowest variance and the respective Eigenvectors in P are ordered accordingly. In other words, from the mean of the collection X, p_1 is the direction of highest variance in the data (with σ_1^2). We define the contribution of component p_i as the percent of total variance it describes for the collection X, such that $cont_X(p_i) = \frac{\sigma_i^2}{\sum_{j=1}^{n} \sigma_j^2}$. pcStream uses the hyperparameter ρ to set the percent of variance to be retained for each model. Let $k \in \mathbb{N}$ to be the fewest, most influential PCs in which their cumulative sum of contributions surpasses ρ. Stated otherwise as $argmin_k \{\sum_{i=1}^{k} cont_X(p_i) \geq \rho\}$. Let the k associated with *situation space* c_i be denoted as k_{c_i}.

pcSteam models the i-th discovered *situation space* as the tuple $c_i \equiv \langle M_i, \mu_i, A_i \rangle$, where M_i is a $n \times m$ matrix consisting of the last m observations assigned to c_i, μ_i is the mean of the observations in M_i, and $A_i = [p_1\sigma_1, p_2\sigma_2, ..., p_{k_{c_i}}\sigma_{k_{c_i}}]$ is a $n \times k_{c_i}$ transformation matrix which is used for measuring the Mahalanobis distance between x and the distribution of c_i by computing $d_{c_i}(x) = \|A_i(x - \mu_i)\|$. Finally, 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 [11, 12].

2.3 The pcStream Algorithm

The hyperparameters for pcStream are: the sensitivity threshold φ (the fuzziness of the detected situations), the minimum situation duration $t_{\min}$, the model memory size m, and the percent of variance to retain in the projections ρ. The pseudo-code for pcStream can be found in Algorithm 1.

In lines 1-3, pcStream is initialized by creating the initial collection $\mathbf{C}$ with *situation* c_1, and then by setting the current *situation* (c_t) accordingly. The function init(S,$t_{\min}$,m,ρ) runs the function CreateModel(X,m,ρ) on the first $t_{\min}$ points of S. The function CreateModel(X,m,ρ) returns a new model c by using the collection of observations X and target total variance retention percentage ρ. Remember that the memory of a model M is a window (FIFO buffer) with a maximum length of m (forgetting the oldest observations). Optionally, an initial set of models for $\mathbf{C}$ can be made from a set of observations pre-classified as known *situations* of S (e.g. a collection of points that capture running and another a collection of points that captures walking). From this point on, pcStream enters its running state (lines 4-18).

In lines 5-6, point x_t arrives and x_t's similarity score is calculated for all known *situations* in $\mathbf{C}$. Stored in i is the index to the model in $\mathbf{C}$ to whom x_t is most similar. Reminder, the index of $\mathbf{C}$ is chronological by order of discovery.

In line 8, it is determined whether x_t fits any of the *situations* in $\mathbf{C}$. If it does, then the model of best fit (c_i) is updated with instance x_t, and the contents of B. The reason B is emptied is because the outliers in B seen until now were not part of a new (unseen) *situation*, but rather a new boundary for the current *situation*. The function UpdateModel(c_i,X) re-computes the tuple c_i from $\mathbf{C}$ after adding the observation(s) X to the FIFO memory M_i.

If the check on line 8 indicates that x_t does not fit any *situation* in $\mathbf{C}$, then x_t is added to the buffer B, and subsequently check if B is full. If B has reached capacity ($t_{\min}$) then an unseen *situation* has been discovered. In this case, B is then emptied and formed into a new model (c), which is added to $\mathbf{C}$ and set as c_t. The function AddModel(c,$\mathbf{C}$) adds c to $\mathbf{C}$ as $c_{|\mathbf{C}|+1}$. If the additional model is too large for the memory space allocated to pcStream, the function merge(c_i,c_j) is used to free one space for c (in $\mathbf{C}$) by merging the average oldest model c_i with its nearest model c_j based on the euclidean distance between centroids. There are two methods for performing this merge: situation freshness (modeling a *situation space* from the m most recent observations between M_i and M_j) and situation preservation (modeling a *situation space* from the interleave between the top $m/2$ observations of M_i and M_j) [7].

Algorithm 1 pcStream$\{S\}$

```
    Input Parameters: $\{\varphi, t_{\min}, m, \rho\}$
    Anytime Outputs: $\{c_t, d_{\mathbf{C}}(\vec{x}_t)\}$
 1: $\mathbf{C} \leftarrow$ init($S$,$t_{\min}$,$m$,$\rho$)
 2: $c_t \leftarrow c_1$
 3: $B \leftarrow \emptyset$
 4: loop:
 5:   $\vec{x}_t \leftarrow$ next($S$)
 6:   $scores \leftarrow d_{\mathbf{C}}(\vec{x}_t)$
 7:   $i \leftarrow$ IndxMin($scores$)
 8:   if $scores(i) < \varphi$ then
 9:     UpdateModel($c_i$,$Dump(B)$)
10:     UpdateModel($c_i$,$\vec{x}_t$)
11:     $c_t \leftarrow c_i$
12:   else
13:     Insert($\vec{x}_t$,$B$)
14:     if length($B$)$==t_{\min}$ then
15:       $c \leftarrow$ CreateModel(Dump($B$),$m$,$\rho$)
16:       AddModel($c$,$\mathbf{C}$)
17:       $c_t \leftarrow c$
18: end loop
```

3. PCSTREAM2

pcStream can model overlapping *situation spaces*, and can handle the case when c_i gradually overlaps c_j due to concept drift. However, pcStream cannot detect when S is in c_i and then crosses c_j briefly. This is because of line 7 in Algorithm 1, where the algorithm naively assigns $\vec{x}_t$ to the closest model (c_i) without considering that the incursion into c_i's space may only be for a brief period of time (i.e., less than $t_{\min}$). This means that pcStream cannot detect new *situation spaces* that inherently overlap from the outset.

For this reason, we propose a new version of pcStream, called pcStream2, that solves this issue with two changes:

Persistence. Instead of immediately assigning each arriving instance to the closest model, we first check if $d_{c_i}(\vec{x}_t) < \varphi$, where i is the index to the current situation's model. If the condition holds then it means that S has not left c_i, thus we should not set some model other model c_j to be current *situation* even if $d_{c_j}(\vec{x}_t) < d_{c_i}(\vec{x}_t)$.

Windowing. In order to detect new *situation spaces* which overlap existing ones, we inspect the most recent $t_{\min}$ observations for a new emerging concept (method described below), a common approach used for concept drift detection [13]. According to our definition of a situation space in section 2.3, we can safely assume that any instances which are older than $t_{\min}$ do not belong to any potential emerging *situation space*, and thus can be used to update exiting models.

We implemented windowing by maintaining a FIFO buffer (B) with a maximum length of $t_{\min}$. When x_t arrives we push into B the tuple $u \equiv (x,d,i)$, where d is the malahanobis distance between x and its assigned model at index i. Later, if the tuple (x,d,i) is popped out of B then we assign x to model i. If B is full, then we check if B contains a new *situation space* with the function containsNewConcept(B). This function returns *true* if the first, last, and overall majority of tuples in B have a $d > \varphi$. In this case, B is then emptied into a new model.

The full pseudocode for pcStream2 can be found in Algorithm 2. The main loop of Algorithm 2 follows three steps: (1) **push** the new instance from S into B, (2) **process** the instance that may have been popped out of B because $|B| > t_{\min}$, and (3) **detect** if there is a new *situation space* in B.

We note that pcStream2 can be more efficient than pcStream. This is because in pcStream, $d_{\mathbf{C}}(\vec{x}_t)$, which has a complexity of $O(|\mathbf{C}|n^2)$, is computed for every observation that arrives from S (Algorithm 1 on line 6). However, in pcStream2, $d_{\mathbf{C}}(\vec{x}_t)$ is only computed if the stream has drifted away from the current *situation space* c_i (Algorithm 2 line 8).

4. PCSTREAM AND IPCA

The implementations of pcStream require the use of PCA. Thus pcStream must store observations in each model and continuously perform a rather expensive operation. These are undesirable aspects with regards to the domain of stream processing. Therefore, we propose the use of incremental PCA (IPCA) instead. Incorporating IPCA into pcStream has three main advantages:

Reduced Complexity. To perform a single PCA operation in pcStream, first the covariance matrix is computed $O(n^2m)$, and then its eigenvalue decomposition is extracted $O(n^3)$. Therefore the complexity of PCA is $O(n^2m+n^3)$ operations. In contrast, the worst case update time with incremental PCA is $O(nk^2)$, where k is the retained rank. Since pcStream relies on the size of a models memory (m), and because typically $k \ll m$, pcStream will benefit in the reduction of complexity.

Algorithm 2 pcStream2$\{S\}$

```
    Input Parameters: $\{\varphi, t_{\min}, m, \rho\}$
    Anytime Outputs: $\{c_i, d_{\mathbf{C}}(\vec{x}_t)\}$
 1: $\mathbf{C} \leftarrow$ init($S, t_{\min}, m, \rho$)
 2: $i \leftarrow 1$
 3: $B \leftarrow \emptyset$
 4: loop:
 5:   $\vec{x}_t \leftarrow$ next($S$)
 6:   if $d_{c_i}(x_t) \leq \varphi$ then                    ▷ Step 1: push
 7:       $u_{\text{in}} \leftarrow (\vec{x}_t, d_{c_i}(x_t), i)$
 8:   else
 9:       $scores \leftarrow d_{\mathbf{C}}(\vec{x}_t)$
10:       $i \leftarrow$ IndxMin($scores$)
11:       $u_{\text{in}} \leftarrow (\vec{x}_t, \text{Min}(scores), i)$
12:   $u_{\text{out}} \leftarrow$ PUSH($u_{\text{in}}, B$)
13:   if $u_{\text{out}} \neq \emptyset$ then                  ▷ Step 2: process
14:       UpdateModel($u_{\text{out}}$)
                                                   ▷ Step 3: detect
15:   if length($B$) == $t_{\min}$ & containsNewConcept($B$) then
16:       $c \leftarrow$ CreateModel($Dump(B), m, \rho$)
17:       AddModel($c, \mathbf{C}$)
18:       $i \leftarrow |\mathbf{C}|$
19: end loop
```

Reduced Memory Consumption. IPCA maintains summaries of the distribution seen thus far. Therefore, there is no need to store the last m observations in each model c, which reduces the memory requirements of pcStream significantly.

Damped Windowing. By maintaining a memory FIFO buffer M_i for each model i, pcStream adapts with concept drift by forgetting older instances. However, a more appropriate approach is to use a damped window [13] that weighs newer instances higher than older ones. Later in section 4.3 we propose using a version of IPCA call JIT-PCA which implicitly operates as a damped window during the incremental updates.

Three changes are required in order to incorporate IPCA into pcStream. First we now model the ith discovered *situation space* as the tuple $c_i \equiv \langle \mu_i, A_i, Q_i \rangle$, where Q_i is a set of variables and parameters used to incrementally update the matrices V and P, and then update the transformation matrix A (see section 2.2). Second, we must change the function init($S, t_{\min}, m, \rho$) by removing the parameter m and by making the function initialize an IPCA model. Finally, we must replace the function CreateModel(X, m, ρ) with IncrementModel(X, c_i) which uses the observations in X to increment the model in Q_i, returning an updated c_i.

4.1 From SVD to Incremental Approximate PCA

The singular-value decomposition (SVD) is well suited for construction of low-rank approximation of matrices, at least if memory and computational costs are ignored. Truncating the smallest $n-k$ singular triplets from the SVD of a matrix X produces the best rank-k approximation to X in both the 2-norm and the Frobenius norm. However, in streaming applications, the cost of the SVD (or even PCA) are prohibitive, so simplifications are widely used. The windowed PCA used in pcStream is one option; here we explore options with less memory.

A wide range of low-rank incremental approximate SVD and PCA algorithms can be derived from the update formula for the full SVD. Let X_t be the matrix consisting of the first t observed columns in the stream and let $X_t = U_t S_t V_t^T$ be its SVD. Let x_{t+1} be the $(t+1)$st columns (observations) of the stream. Denoting $x_{t+1}^{span} = U_t^T x_{t+1}$, $x_{t+1}^{orth} = \left(I - U_t U_t^T\right) x_{t+1} = x_{t+1} - U_t x_{t+1}^{span}$, $\delta = \|x_{t+1}^{orth}\|$, $d = x_{t+1}^{orth}/\delta$. One can easily verify that:

$$X_{t+1} = \begin{bmatrix} X_t & x_{t+1} \end{bmatrix} = \begin{bmatrix} U_t S_t V_t^T & x_{t+1} \end{bmatrix} = \begin{bmatrix} U_t & d \end{bmatrix} \begin{bmatrix} S_t & x_{t+1}^{span} \\ \bar{0} & \delta \end{bmatrix} \begin{bmatrix} V_t & \bar{0} \\ \bar{0} & 1 \end{bmatrix}^T \quad (1)$$

We denote the SVD of $Z = \begin{bmatrix} S_t & x_{t+1}^{span} \\ \bar{0} & \delta \end{bmatrix}$ by $Z = U_z S_Z V_Z^T$ and the SVD of X_{t+1} by $X_{t+1} = U_{t+1} S_{t+1} V_{t+1}^T$. We have

$$U_{t+1} = \begin{bmatrix} U_t & d \end{bmatrix} U_Z \quad (2)$$

$$S_{t+1} = S_Z \quad (3)$$

$$V_{t+1} = \begin{bmatrix} V_t & \bar{0} \\ \bar{0} & 1 \end{bmatrix} V_Z \quad (4)$$

The formulas above update the exact, full incremental SVD, which unless $\delta = 0$, results in a rank increase. To limit the computational and memory costs, researchers have proposed several rules to truncate the resulting decomposition to a fixed rank k. One simple way to truncate is to drop the smallest singular triplet (pair in PCA) after every update, if the rank goes higher than k.

Even if we truncate the PCA to rank k and even if $k \ll n$, the per-iteration cost of incremental PCA is high. If implanted naively (as above), the cost of an IPCA step is dominated by the $O\left(nk^2\right)$ cost of updating $U_{t+1} = \begin{bmatrix} U_t & d \end{bmatrix} U_Z$. Over the years, several lower-cost alternatives have been proposed, generally aiming to reduce the quadratic dependence on k to linear. One approach updates the SVD only every batch of k or so vectors, not after every vector [14]. This approach still has a quadratic cost per update but if the block size is at least k, then the amortized per-update cost is linear. Chahlaou et al. [15] proposed a QR representation with Q orthonormal and R upper triangle. Their algorithm takes advantage of efficient multiplication using Householder reflections, achieving an arithmetic cost of only $8nk$ operations per update. Brand [16] proposed a completely different approach: From a certain iteration t and for all remaining columns, set $x_{t+1} = x_{t+1}^{span}$. From this point on, the subspace that represents the column space of X_t does not change, but the singular vectors that span the space and the associated singular values might still change. To achieve maximum efficiency, Brand's algorithm keeps the rotation that defines U_t implicit (as a product of rotations), so this algorithm too relies on batching. The per-iteration cost of this method is dominated by the $2nk$ cost of the projection $U_t^T x_{t+1}$.

Comparing these approaches to the windowed PCA that was used in pcStream, we see that these incremental approximate methods can potentially deliver three benefits. First, they use almost all the memory to represent the basis U_t, whereas windowed PCA requires storing both the input vectors of the window and the singular vectors. To accurately represent a space of dimension k, the window often needs to be much larger, so the input vectors in the window dominate memory use and limit k and/or the number of cluster, and hence the accuracy of the representation. Second, because the window size m must be significantly larger than k, the cost of every PCA computation is $O(m^2 n)$, where as approximate incremental PCA methods can be implemented in $O(kn)$ operations per iteration; even if the windowed approach computes a PCA every k or every m columns, the incremental approaches are cheaper (even if they update the basis after every vector!). Third, the windowed approach completely truncates the history of the stream, whereas incremental PCA methods retain some representation of the history.

4.2 Engineering an Effective IPCA Algorithm

Incremental PCA/SVD algorithms that truncate the representation to maintain an $O(nk)$ memory bound are all heuristic, in the sense that none of them deliver the strong bounds of the full SVD. We also note that the strong guarantees of full PCA are not necessarily relevant to our application, in which the matrices are not given, but constructed dynamically in a way that depends on the PCA algorithm. That is, different PCA and IPCA algorithms generate different matrices for the clusters in our application, so pure linear-algebraic characterizations of them are not necessarily relevant.

Furthermore, we have observed that the algorithms in the literature often fail in fairly simple cases. Even the incremental algorithm with the strongest guarantees in the literature, FrequentDirections [17], only provides guarantees on the Frobenius norm of the error, not on its 2-norm. To address this difficulty, we have conducted extensive experiments with synthetic streams with the aim of finding the failure modes of the algorithms that have been proposed so far. Due to lack of space in this abstract, we omit the details. We used these observations to drive the design of a new IPCA algorithm that is more robust than existing proposals in the literature, and as efficient as the fastest of them, with a per-operation cost between $2nk$ and $8nk$. We stress that our algorithm, called JIT-PCA, is also a heuristic algorithm that may fail on some streams. But we have included features that do protect it from simple failure modes that we have observed in other algorithms.

4.3 JIT-PCA

JIT-PCA is a heuristic randomized incremental PCA algorithm. It combines building blocks from many previous proposals in the literature to produce an effective and fast algorithm. More specifically, we use the following building blocks and insights.

- We use the QR-based update formulas developed by Chahlaoui et al [15] to ensure that the arithmetic cost of every update step is bounded by $8nk$ operations.

- We exploit the observation that when $\delta = 0$, the representation can be updated at a cost of only $2nk$ operations [16], and we use it even when δ is small but not exactly zero.

- To ensure that the errors due to the previous technique (rounding small δ to zero) or due to the incremental process itself, does not accumulate, the decision to truncate is randomized, and when the random process decides not to truncate, it boosts x_{t+1}^{orth} in order to compensate for truncation decisions in other cases. This idea builds both on the many recent randomized sketching algorithms in linear algebra, which started in the seminal paper by Frieze et al. [18], and on the scaling technique of Liberty [17] (except that Liberty scales down the existing basis vectors and we boost the new direction).

Algorithm 3 JIT-PCA_init

```
    **Input Parameters:** $X_{initial}$          ▷ First block of instances
    **Outputs:**
    Matrices: $U_t, S_t, R_t, U_{rotate_t}$
    Integers/Booleans: $tMass, t, prevSpan, r$
    $\{U_t, S_t\} \leftarrow \mathrm{SVD}(X_{initial})$
    $R_t \leftarrow 1$
    $U_{rotate_t} \leftarrow I_k$
    $r \leftarrow 1$
    $prevSpan \leftarrow no$                     ▷ Previous Span
    $tMass \leftarrow \sum_{i=1}^{t} x_i^2$      ▷ Total Mass
```

As explained above, our aim in this algorithm is to achieve high performance through (1) the use of clever algebraic representations [15] and (2) exploitation of easy instances, which here corresponds to steps in which the subspace (almost) does not change, only its singular basis [16]. We also aim to avoid failure modes that we have observed in simpler algorithms; in this task we combine scaling [17] with randomization [18]. JIT-PCA tries to estimate the importance of the new vector to our approximation. It maintains a counter called $total\_mass = \sum_{i=1}^{t} x_i^2$ and defines $avgMass = total\_mass/t$. The probability to use the orthogonal part in the update based on its estimated importance is $min\left(\delta^2/avgMass, 1\right)$.

The second method is a relaxation method, intended to make sure that the update has time to stabilize on new directions and prevent it from "jumping" from one direction to another. Here JIT-PCA maintain a counter called r, which is set to 1 after each time the orthogonal part is used and incremented by 1 if its not. The final probability to use the orthogonal part is $prob\_orth = min\left(\delta^2/avgMass, 1\right) \cdot (1 - 1/(1+r))$. We then roll a dice and decide if the orthogonal part is to be used or not.

If JIT-PCA decides not to use the orthogonal part, it sets $\delta = 0$ and continues with an update similar to Brand's [16]. If the orthogonal part is to be used, based on the value of the smallest singular value in S_t it either keep δ untouched or give it a boost. In both cases the update method will then be similar to Chahlaoui et al [15]

In Algorithms 3 and 4 we present the pseudo-code for JIT-PCA's initialize and increment procedures respectively. In the pseudo-code we denote $\sigma_{min} = S_{k,k}$, and set $orth\_large \leftarrow true$ if $\delta > \sigma_{min}$, otherwise $false$. The $prevSpan$ variable is an indicator of the update method used at previous step. This is needed because JIT-PCA uses different structures for the different update types. *update* is an indicator of which type of update to perform, with / without the orthogonal part, and also indicates which type of manipulation to perform on δ. $\widetilde{\delta}$ can be 0 (if the chosen method is to only use the spanned part), untouched (if both *use_orth* and *orth_large* are *true*) or boosted by $\widetilde{\delta} = \min(\sigma, ratio \cdot \delta)$ where $ratio = \sqrt{(\|x_{t+1}\|^2 + \sigma^2)/\|x_{t+1}\|^2}$.

The new direction d, is calculated only if needed and under the *update_pca* function.

5. EXPERIMENTAL RESULTS

In this section we present an evaluation of pcStream2 and the incorporation of IPCA into pcStream with JIT-PCA. We measure the performance of the modifications in terms of clustering accuracy and runtime. Each experiment was conducted on a single core of a Xeon E5-2640 processor.

Algorithm 4 JIT-PCA_increment(x_{t+1})

```
    **Inputs & Outputs:**
    Matrices: $U_t, S_t, R_t, U_{rotate_t}$
    Integers/Booleans: $tMass, t, prevSpan, r$
    $\{tMass, avgMass, \|x_{t+1}\|\}$
                  $\leftarrow \mathrm{updateMass}(tMass, counter, x_{t+1})$
    $x_{t+1}^{span} \leftarrow \mathrm{genSpan}(U_t, U_{rotate_t}, prevSpan)$
    $\{\delta, \sigma_{min}\} \leftarrow \mathrm{preprocess}\left(x_{t+1}^{span}, \|x_{t+1}\|, S_t\right)$
    $\{use\_orth, orth\_large, r\}$
                              $\leftarrow \mathrm{draw}(avgMass, \delta, r, \sigma_{min})$
    $\{update, prevSpan\}$
             $\leftarrow \mathrm{decide}(use\_orth, orth\_large, prevSpan)$
    $\widetilde{\delta} \leftarrow \mathrm{manipulate}(update, \sigma_{min}, \delta, \|x_{t+1}\|)$
    $\{U_{t+1}, S_{t+1}, R_{t+1}, U_{rotate_{t+1}}\}$
    $\leftarrow \mathrm{incrementPCA}\left(x_{t+1}, x_{t+1}^{span}, \widetilde{\delta}, U_t, S_t, R_t, U_{rotate_t}, update\right)$
```

Four different data streams were used in the evaluations, each capturing a different *context domain*. The first dataset (HearO) was a smartphone sensor dataset [19]. An instance in the dataset contains a user's device motion and battery consumption along with a label indicating the user's context (provided explicitly by the user). This dataset was selected to evaluate the performance in detecting complex *situation spaces*. The second dataset (KDD) was the KDD'99 network intrusion dataset by the MIT Lincoln laboratory [20]. This dataset was selected primarily to evaluate runtime performance in the presence of many features. The third dataset (SherLock) [21] was a smartphone dataset that measures a large number of device sensors in the background while the user interacts with infected malicious applications. From this dataset we extracted a time-series trace of spyware. The time series contained three features (sampled once every 5 seconds) that captured the infected application's behavior. The fourth dataset (SCA), was an activity recognition dataset where a single chest-mounted accelerometer was used to capture a human's various activities [22]. This dataset was selected to validate if pcStream2 improves accuracy by detecting overlapping *situation spaces*. All four datasets were zscore-normalized before being processed. A summary of the datasets can be found in Table 1, and list of the parameters used on each dataset is available in Table 2. The model memory parameter (m) was set to 1000 for all experiments.

5.1 pcStream vs pcStream2

To compare the quality of a clustering assignment (i.e., which observation belongs to which *situation space*), we used the Adjusted Rand Index (ARI). 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 on a dataset by calculating the ARI between the algorithm's clustering assignment and the dataset's labels. When there is 1-to-1 match then the ARI score is 1.

Figure 3 presents the best ARI achieved by pcStream and pcStream2 for each of the datasets, where it can be seen that pcStream2 shows improvement in every case. As an example of parameter selection robustness, we present in Figure 4 the resulting ARI from each experiment (set of parameters) performed on the SherLock dataset. It can be seen that pcStream2 is noisier than pcStream. This is likely because of pcStream2's

Dataset	n	Features	# of Rows	Context Domain	# Labels	Labels
HearO	5	Acceleration correlation between the xy, xz, and yz axis, device temperature, and battery level.	1764	High-level Contexts	4	At the University, On my way to, At home, At work.
KDD'99	38	duration, src_bytes, dst_bytes, wrong_fragment, urgent, hot, num_failed_logins, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, …	50,000	Network Attacks	23	back,buffer_overflow,ftp_write,guess_passwd,imap,ipsweep,land,loadmodule,multihop,neptune,nmap,normal,perl,phf,pod,portsweep,rootkit,satan,smurf,spy,teardrop,warezclient,warezmaster.
SherLock	3	CPU utilization, Memory Resident Set Size (RSS), current app importance	357,087	Smartphone Spyware	2	Benign Session, Malicious Session
SCA	3	Acceleration on the x, y, and z axis	162,500	Motion Activity	7	Working at computer, Going updown stairs, Standing up then walking, Going updown stairs, Walking and talking with someone, Talking while standing.

Table 1: Summaries of the three datasets used for the evaluations.

Datasets	φ	t_{min}	ρ	Total
HearO	0.5:0.01:6.0	6:1:50	98%	24,795
KDD	0.5:0.1:6.0	40:10:250	98%	1,232
SherLock	0.5:0.1:6.0	6:1:40	98%	1,960
SCA	0.5:0.1:6.0	50:10:250	80%	1,176

Table 2: The parameters used in the evaluations over each dataset.

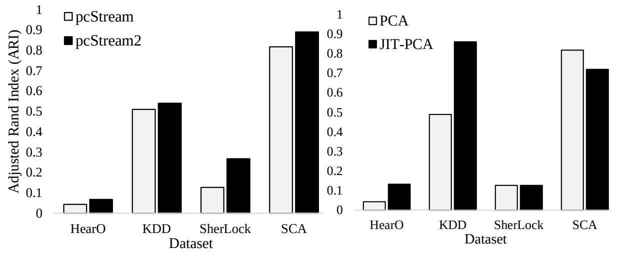

Figure 3: The best ARI achieved by pcStream and pcStream2 (left), pcStream with PCA and JIT-PCA (right), for each dataset.

persistence. However, the persistence in-turn made pcStream2 more successful in detecting the *situation spaces*.

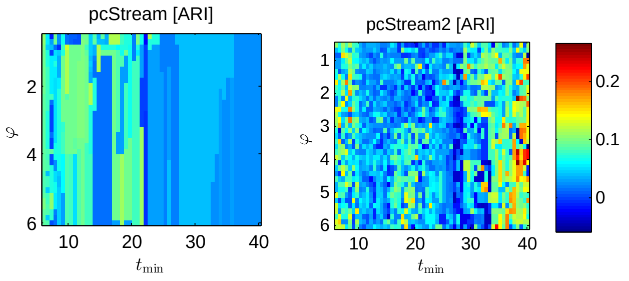

Figure 4: The resulting ARI for every parameter selection for both pcStream and pcStream2 over the SherLock dataset.

5.2 pcStream with IPCA

Figure 5 presents a full comparison between using PCA and JIT-PCA over the SCA dataset. It can be seen from the top row of Figure 5 that usage of IPCA does not significantly harm the robustness of pcStream's parameter selection. This means that a set of parameters used with pcStream will find the same *situation spaces* when used with pcStream2. Moreover, the bottom row shows that JIT-PCA improves runtime regardless of feature selection. This is an important observation because different sets of hyper-parameters extract different *situation spaces*, and not necessarily those reflected by dataset's labels. Figure 5 and 6 indicate that JIT-PCA can provide a speed increase five times greater than using regular PCA.

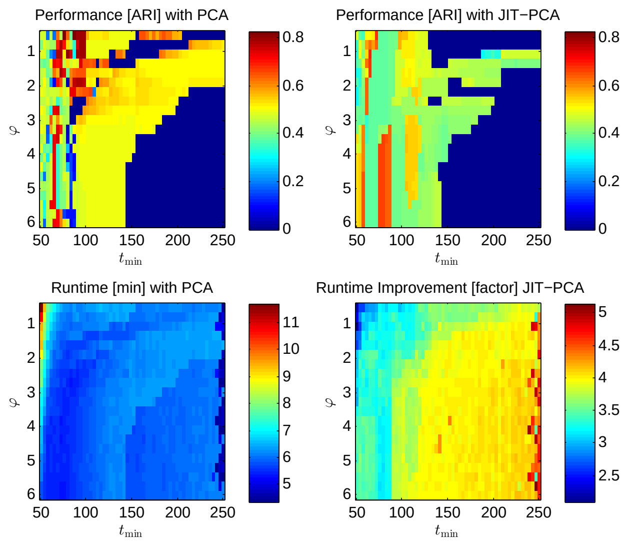

Figure 5: A comparison of accuracy (top row) and runtime (bottom row) when using PCA and IPCA with different pcStream parameters over the SCA dataset.

Figure 3 shows that a reasonable ARI can be obtained for each dataset with IPCA. In some cases IPCA outperforms regular PCA. This is because in the PCA version of pcStream, each model can only retain the last m instances, whereas with IPCA, the model retains the information from each observation longer.

Finally, using the KDD dataset and the best pcStream parameters, we measured the affect on runtime when adding more features (dimensions) to the stream. For each dimension size n, we repeated 100 experiments each time with a random subset of n features. Figure 7 demonstrates the advantage of using JIT-PCA as pcStream's IPCA with stream that have a large dimensionality.

6. CONCLUSION

In this paper we have shown two new methods for improving pcStream's efficiency and accuracy. Both are important aspects when dealing with online data stream mining. With pcStream2 it is possible to detect emerging situation spaces which overlap in the same feature space, and in conjunction with JIT-PCA, these results can be accomplish far more efficiently, and sometimes

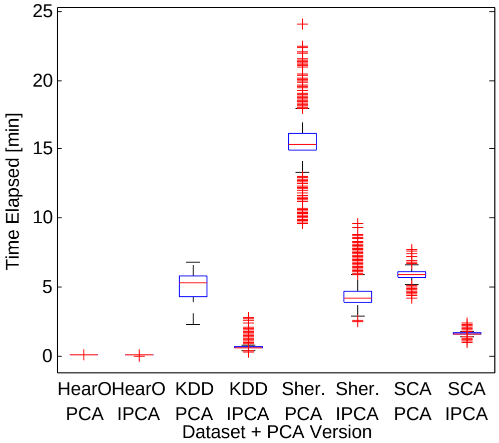

Figure 6: The runtimes of pcStream with PCA and JIT-PCA for each dataset.

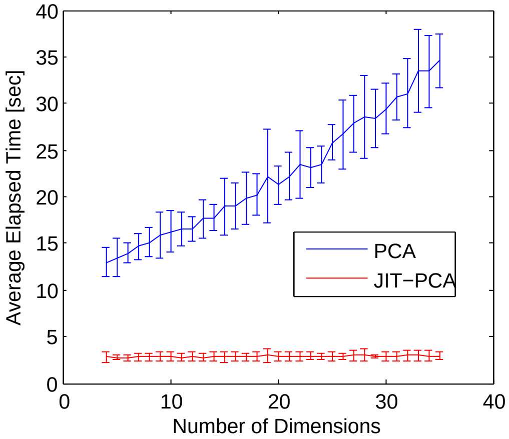

Figure 7: The affect the number of dimensions have on pcStream's runtime on the KDD dataset with PCA and JIT-PCA respectively. The bars represent the standard deviation.

more accurately. In the future we plan to evaluate pcStream2 in the application of anomaly detection [8]. We also plan to perform an in-depth comparison between using different IPCA algorithms with pcStream.

7. ACKNOWLEDGMENTS

This research was funded by the Israeli Ministry of Science, Technology and Space.

8. REFERENCES

[1] Pierluigi Casale, Oriol Pujol, and Petia Radeva. Human activity recognition from accelerometer data using a wearable device. In *Iberian Conference on Pattern Recognition and Image Analysis*, pages 289–296. Springer, 2011.

[2] Guillermo Suarez-Tangil, Juan E Tapiador, Pedro Peris-Lopez, and Arturo Ribagorda. Evolution, detection and analysis of malware for smart devices. *IEEE Communications Surveys & Tutorials*, 16(2):961–987, 2014.

[3] Amir Padovitz, Seng Wai Loke, and Arkady Zaslavsky. Towards a theory of context spaces. In *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on*, pages 38–42. IEEE, 2004.

[4] Julien Pauty, Paul Couderc, and Michel Banâtre. Using context to navigate through a photo collection. In *Proceedings of the 7th international conference on Human computer interaction with mobile devices & services*, pages 145–152. ACM, 2005.

[5] Pari Delir Haghighi, Arkady Zaslavsky, Shonali Krishnaswamy, and Mohamed Medhat Gaber. Mobile data mining for intelligent healthcare support. In *hicss*, pages 1–10. IEEE, 2009.

[6] Seungkeun Lee and Junghyun Lee. Dynamic context aware system for ubiquitous computing environment. In *Agent Computing and Multi-Agent Systems*, pages 409–419. Springer, 2006.

[7] Yisroel Mirsky, Bracha Shapira, Lior Rokach, and Yuval Elovici. pcstream: A stream clustering algorithm for dynamically detecting and managing temporal contexts. In *Advances in Knowledge Discovery and Data Mining*, pages 119–133. Springer, 2015.

[8] Yisroel Mirsky, Asaf Shabtai, Bracha Shapira, Yuval Elovici, and Lior Rokach. Anomaly detection for smartphone data streams. *Pervasive and Mobile Computing*, 2016.

[9] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106, 2004.

[10] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[11] Jonathan A. Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschka, André C. P. L. F. de Carvalho, and João Gama. Data stream clustering: A survey. *ACM Comput. Surv.*, 46(1):13:1–13:31, July 2013.

[12] Brain Babcock, Mayur Datar, Rajeev Motwani, and Liadan O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the Twenty-second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '03, pages 234–243, New York, NY, USA, 2003. ACM.

[13] Jonathan A Silva, Elaine R Faria, Rodrigo C Barros, Eduardo R Hruschka, André CPLF de Carvalho, and João Gama. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)*, 46(1):13, 2013.

[14] A Levey and Michael Lindenbaum. Sequential karhunen-loeve basis extraction and its application to images. *IEEE Transactions on Image processing*, 9(8):1371–1374, 2000.

[15] Younes Chahlaoutf, Kyle A Gallivant, and Paul Van Dooren. An incremental method for computing dominant singular spaces. *Computational information retrieval*, 106:53, 2001.

[16] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. *Linear algebra and its applications*, 415(1):20–30, 2006.

[17] Edo Liberty. Simple and deterministic matrix sketching. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 581–588. ACM, 2013.

[18] Alan Frieze, Ravi Kannan, and Santosh Vempala. Fast monte-carlo algorithms for finding low-rank approximations. *Journal of the ACM (JACM)*, 51(6):1025–1041, 2004.

[19] Moshe Unger, Ariel Bar, Bracha Shapira, Lior Rokach, and Ehud Gudes. Contexto: lessons learned from mobile context inference. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 175–178. ACM, 2014.

[20] Stephen D Bay, Dennis Kibler, Michael J Pazzani, and Padhraic Smyth. The uci kdd archive of large data sets for data mining research and experimentation. *ACM SIGKDD Explorations Newsletter*, 2(2):81–85, 2000.

[21] Yisroel Mirsky, Asaf Shabtai, Lior Rokach, Bracha Shapira, and Yuval Elovici. Sherlock vs moriarty: A smartphone dataset for cybersecurity research. In *Proceedings of the 2016 ACM workshop on Artificial intelligence and security*, pages 45–54. ACM, 2016.

[22] Pierluigi Casale, Oriol Pujol, and Petia Radeva. Personalization and user verification in wearable systems using biometric walking patterns. *Personal and Ubiquitous Computing*, 16(5):563–580, 2012.