

# Real Time Principal Component Analysis

RANAK ROY CHOWDHURY, University of California San Diego, USA and Halicioğlu Data Science Institute, USA

MUHAMMAD ABDULLAH ADNAN, Bangladesh University of Engineering and Technology (BUET), Bangladesh

RAJESH K. GUPTA, University of California San Diego, USA and Halicioğlu Data Science Institute, USA

We propose a variant of Principal Component Analysis (PCA), that is suited for real-time applications. In the real-time version of the PCA problem, we maintain a window over the most recent data and project every incoming row of data into a lower dimensional subspace, which we generate as the output of the model. The goal is to reduce the reconstruction error of the output from the input and to retain major components pertaining to previous distributions of the data. We use the reconstruction error as the termination criteria to update the eigenspace as new data arrives. We then propose two variants of this algorithm that are progressively more time-efficient. To verify whether our proposed model can capture the essence of the changing distribution of large datasets in real-time, we have implemented the algorithms and compared performance against carefully designed simulations that change distributions of data sources over time in a controllable manner. Furthermore, we have demonstrated that proposed algorithms can capture the changing distributions of real-life datasets by running simulations on datasets from a variety of real-time applications e.g. localization, activity recognition, customer expenditure, etc. Results show that straightforward modifications to convert PCA to use a sliding window of data sets do not work because of the difficulties associated with determination of optimal window size. Instead, we propose algorithmic enhancements that rely upon spectral analysis to improve dimensionality reduction. Results show that our methods can successfully capture the changing distribution of data in a real-time scenario, thus enabling real-time PCA.

CCS Concepts: • **Information systems** → **Data analytics**; *Online analytical processing*; • **Mathematics of computing** → **Dimensionality reduction**; • **Computing methodologies** → **Principal component analysis**; • **Computer systems organization** → *Real-time systems*.

Additional Key Words and Phrases: Big Data, Real Time, Dimensionality Reduction, PCA.

## ACM Reference Format:

Ranak Roy Chowdhury, Muhammad Abdullah Adnan, and Rajesh K. Gupta. 2018. Real Time Principal Component Analysis. 1, 1 (May 2018), 36 pages. <https://doi.org/10.1145/1122445.1122456>

## 1 INTRODUCTION

### 1.1 Problem Context

Real time big data processing has far-reaching applications in the real world. For example, during times of a natural or a man-made catastrophe, social media platforms, like Facebook and Twitter, are flooded with posts and pictures pertaining

---

Authors' addresses: Ranak Roy Chowdhury, University of California San Diego, La Jolla, California, 92093, USA, [rchowdh@eng.ucsd.edu](mailto:rchowdh@eng.ucsd.edu); Muhammad Abdullah Adnan, Bangladesh University of Engineering and Technology (BUET), Dhaka, 1000, Bangladesh, [abdullah.adnan@gmail.com](mailto:abdullah.adnan@gmail.com); Rajesh K. Gupta, University of California San Diego, La Jolla, California, 92093, USA, [rgupta@ucsd.edu](mailto:rgupta@ucsd.edu).

---

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.

Manuscript submitted to ACM

to that event [72] [62] [53]. During such an event, there is a significant change in the pattern and distribution of words across posts. There may be valuable information implicitly coded within this shift in pattern. Identifying such change points and approximating the new distribution immediately from fresh data is vital to social media analysis and trend detection [60] [68] [67]. Unfortunately, even carefully engineered networked systems have to make assumptions about the nature of network traffic, data distribution that may not be valid in real-time episodic changes. Over-engineering for such situations may render the system slow and, therefore, the low latency proves to be costly during critical times [61][50]. We need means for data processing in a way that filters out the insignificant changes and takes into account the **latest shift** in the data distribution while **retaining such major changes** seen over time. Indeed, any system where time is of essence and decisions rely on real-time shift in the distribution of data requires fast data processing and on the fly computation. As the size of data to be processed in real-time increases, it makes task all the more difficult.

## 1.2 Approach and Contributions

We start with Principal Component Analysis (PCA) [33] [75] [42] to reduce data size while retaining as much of the underlying information as possible. PCA is a popular statistical procedure that is used for dimensionality reduction [42], factor analysis [26], correlation analysis [22], clustering [30], classification [55] and many more advanced scientific computing and research fields. Standard PCA is offline, that is, it assumes that the entire data set is present before computation begins. Recently, **Online PCA** [17] [45] have been proposed that compute principal components for streaming data with the goal to provide the best low dimensional representation of **all the data** seen so far which is not suitable for real-time applications where abrupt transitions in data distribution may occur. In this paper, we propose the concept of **Real-time PCA** that a) provides a low dimensional representation of the **current distribution** of the data using PCA, and b) **retains the major components** of the previous distributions seen so far.

We propose algorithms that operate on streaming high-dimensional input data. The window size and the target dimension must be specified as input parameters. A real-time system must not only provide the best approximation for the current distribution but also reflect the effects of major components of previous distributions on the current data. Therefore, the low-dimensional subspace representation given by our algorithms focuses on the current distribution besides retaining the major components pertaining to previously seen distributions. Hence, the low-dimensional output of our algorithms closely approximates its corresponding input as we show in the experimental verification section. The most obvious solution, as presented in **Real Time PCA using Block Offline PCA** algorithm, is to construct a window that slides past the input data and computes the standard offline PCA for each block<sup>1</sup>. However, it is an inelegant solution as it does redundant computation and focuses only on the current block, not the overall data in general. More refined techniques are needed to utilize the similarity within successive windows to reduce the computational burden and also to incorporate the global nature of data. The first algorithm described in this paper is **Real Time PCA with target dimension (RP)** that uses reconstruction error between the actual data block and its corresponding projected output to decide how much to modify the lower dimensional subspace, given by PCA. Next, we describe **Time-Efficient Real Time PCA with target dimension (RPT)** algorithm that chooses the most important directions among the existing and the newly computed directions to represent the subspace. Finally, a third algorithm **Robust Time-Efficient Real Time PCA with target dimension (RPTO)** updates either one direction or none at each iteration, which ensures any change that takes place in the representation of the lower dimensional subspace is gradual and not due to any random noise.

<sup>1</sup>We use block and window interchangeably throughout the paper.

We conduct extensive experiments using synthetic and real datasets from various real-time applications. We graphically show how the reconstruction error varies with the block size and target dimension for each algorithm. Moreover, we discuss the bandwidth capacity of our algorithms as compared to the speed at which the input is generated. The progressive algorithmic sophistication leading upto the **Algorithm 5, Robust Time-Efficient Real-Time PCA with target dimension (RPTO)** improves upon the other two algorithms in terms of error measurement and bandwidth performance. It is also the most time-efficient algorithm compared to the other two algorithms. We also experimentally validate that our algorithms are capable of capturing the changing distributions of real-time data in reduced dimension using spectral analysis. We verify whether the algorithms demonstrate their anticipated behavior whenever there is a change in data distribution. Using the Bhattacharyya Distance that measures the degree of similarity between two probability distributions, we show how well the distributions of the output generated by our algorithms correspond to the actual distributions of the input. Using this statistical measure, we also show that the statistical changes occurring across the input distributions are approximately similar to the ones occurring across the output distributions.

## 2 BACKGROUND

### 2.1 Principal Component Analysis (PCA)

PCA is a linear orthogonal transformation technique that transforms data to a new coordinate system such that the maximum variance by any projection of the data comes to lie on the first axis known as the first principal component, the second maximum variance on the second axis and so on. The ‘basis’ represents a set of orthogonal linearly independent vectors that form a coordinate system. Thus PCA seeks to optimize the dimension of input vectors with minimum reconstruction error. In the standard PCA problem[33][75], the input is a set of vectors  $[x_1, x_2, \dots, x_n] \in \mathbb{R}^d$  of dimension  $d$  and a target dimension of output vectors  $l$  ( $l < d$ ). The solution is a set of output vectors  $[y_1, y_2, \dots, y_n] \in \mathbb{R}^l$  of dimension  $l$ , such that the reconstruction error is reduced. The reconstruction error can be defined in two forms, the Frobenius norm:  $\|(X - \varphi Y)\|_F^2$  and the Spectral norm:  $\|(X - \varphi Y)\|_2^2$  where  $X$  is the input data,  $Y$  is the output data and  $\varphi \in \mathbb{R}^{d \times l}$  is the isometric projection matrix. The best optimum value of this reconstruction error can be obtained in standard offline PCA algorithm.

Principal components can thus be represented as the principal axis of a  $n$ -dimensional ellipsoid. The main objective of PCA is to fit the data to the  $n$ -dimensional ellipsoid. Small axis of the ellipsoid represents small variance along that axis and by ignoring that axis along with its corresponding principal component, we lose only a small amount of information that is not much significant.

### 2.2 PCA using Eigenvalue Decomposition

**2.2.1 Eigenvector and Eigenvalue.** A scalar  $\lambda$  is called an eigenvalue of the square  $n \times n$  matrix  $A$  if there is a nontrivial solution  $x$  of  $Ax = \lambda x$ . Such an  $x$  is called an eigenvector corresponding to the eigenvalue  $\lambda$ . In standard form:

$$(A - \lambda I)x = 0 \quad (1)$$

Here  $(A - \lambda I)$  is not invertible i.e.  $(A - \lambda I)$  is singular.

$$|A - \lambda I| = 0 \quad (2)$$

$$(\lambda - \lambda_1)(\lambda - \lambda_2)(\lambda - \lambda_3) \dots (\lambda - \lambda_n) = 0 \quad (3)$$

**2.2.2 Eigenvalue Decomposition (EVD).** Eigenvalue decomposition is the factorization of a matrix into a canonical form, whereby the matrix is represented in terms of its eigenvalues and eigenvectors. Only diagonalizable matrices can be factorized in this way. The eigenvalue decomposition applies to mappings from  $\mathbb{R}^n$  to itself, i.e. a linear operator  $A : \mathbb{R}^n \rightarrow \mathbb{R}^n$  described by a square matrix. An eigenvector  $e$  of  $A$  is a vector that is mapped to a scaled version of itself, i.e.,  $Ae = \lambda e$ , where  $\lambda$  is the corresponding eigenvalue. For a matrix  $A$  of rank  $r$ , we can group the  $r$  non-zero eigenvalues in an  $r \times r$  diagonal matrix  $\Lambda$  and their eigenvectors in a  $n \times r$  matrix  $E$ , and we have

$$AE = E\Lambda \quad (4)$$

Furthermore, if  $A$  is full rank ( $r = n$ ) then  $A$  can be factorized as

$$A = E\Lambda E^{-1} \quad (5)$$

which is a diagonalization similar to the Singular Value Decomposition (SVD). In fact, if and only if  $A$  is symmetric and positive definite (abbreviated SPD), we have that the SVD and the eigendecomposition coincides

$$A = USU^T = E\Lambda E^{-1} \quad (6)$$

with  $U = E$  and  $S = \Lambda$ .

**2.2.3 Computing PCA using EVD.** To find the axes of the ellipsoid, we must mean center the data around the origin by first subtracting the mean of each variable from the dataset. Then we can compute the covariance matrix of the data and calculate the eigenvalues and the corresponding eigenvectors of this covariance matrix. Next, we must orthogonalize the set of eigenvectors and normalize them to form unit vectors. Each of these mutually orthogonal, unit eigenvectors can be interpreted as an axis of the ellipsoid fitted to the data. The proportion of the total variance captured by each eigenvector can be calculated by dividing the eigenvalue corresponding to that eigenvector by the sum of all eigenvalues. Let,  $X_{n \times d}$  represent the centered input matrix of dimension  $n \times d$ , where there are  $n$  rows of data, each having dimension  $d$ . Then the covariance matrix is given by

$$A = \frac{XX^T}{n-1} \quad (7)$$

and from Equation (5), we know that  $A = E\Lambda E^{-1}$  via Eigenvalue Decomposition, where  $E$  is a matrix of eigenvectors (each column is an eigenvector) and  $\Lambda$  is a diagonal matrix with eigenvalues  $\lambda_j$  in decreasing order along the diagonal and  $j = 1, 2, \dots, d$ . The eigenvectors are called principal axes or principal direction of data. Projections of the data points on the principal axes are called principal components. The  $j$ -th principal component is given by  $j$ -th column of  $U$  and the coordinates of  $i$ -th data point in the new Principal Component space are given by the  $i$ -th row of  $XU$ .

## 2.3 PCA Using Singular Value Decomposition

The best way to solve this standard offline PCA problem is to center the input matrix  $X$  and perform the Singular Value Decomposition (SVD) [9] [41] on it.

**2.3.1 Singular Value Decomposition (SVD).** The singular value decomposition (SVD) factorizes a linear operator  $A : \mathbb{R}^n \rightarrow \mathbb{R}^m$  into three simpler linear operators:

- (1) Projection of  $z = V^T x$  into an  $r$ -dimensional space, where  $r$  is the rank of  $A$
- (2) Element-wise multiplication with  $r$  singular values, i.e.,  $z' = Sz$
- (3) Transformation of  $y = Uz'$  to the  $m$ -dimensional output space

Combining these statements,  $A$  can be re-written as

$$A = USV^T \quad (8)$$

where  $U$  is a  $m \times r$  orthonormal matrix spanning  $A$ 's column space,  $S$  is a  $r \times r$  diagonal matrix of singular values, and  $V$  is a  $n \times r$  orthonormal matrix spanning  $A$ 's row space.

**2.3.2 PCA Using SVD.** Let the SVD of  $X$  be  $USV^T$ , then the best isometric projection will be  $\varphi = U_l$ , where  $U_l \in \mathbb{R}^{d \times l}$  consists of  $l$  left singular vectors of  $U$  corresponding to the top  $l$  singular values of  $S$ . The corresponding output vector will be  $y_t = U_l^T x_t$ , where  $x_t$  is the input vector and  $y_t$  is the output vector at time  $t$ . As we need the whole input matrix  $X$  for generating the best projection plane  $\varphi$ , so for offline PCA we get the best optimal reconstruction error and the best isometric projection matrix.

## 2.4 Online PCA

We have discussed the standard offline PCA and how it finds the principal vectors and its components. It generates all the reduced dimension output vectors at the same time. It is a challenge to establish an online PCA algorithm that will simultaneously reduce the dimension of output vectors and the reconstruction error.

In the online version of the PCA problem, each of the input vectors  $x_t$  is given input to the system one by one and the system tries to reduce the reconstruction error with respect to the Frobenius norm,  $\|(X - Y\varphi^T)\|_F^2$  [17] or the Spectral Norm,  $\|(X - Y\varphi^T)\|_2^2$  [45] and generate the corresponding output vectors  $y_t$  before getting the next input. If  $l$  denotes the output dimension in online PCA, then  $l < d$ . As we cannot build the projection plane based on the whole input data matrix  $X$  for online PCA, clearly the reconstruction error for online PCA will not be as good as that for the standard offline PCA.

From the definition of Frobenius norm we know that,

$$\|A\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2 = \sum_{i=1}^R \sigma_i^2 \quad (9)$$

where  $m, n$  represent the row and column number of matrix  $A$ ,  $R$  represents the rank of matrix  $A$  and  $\sigma_i$  represents the  $i$ -th top singular value of  $A$ .

And the spectral norm is,

$$\|A\|_2^2 = \sigma_{\max}^2(A) \quad (10)$$

In the standard offline PCA, the best projection plane is not dependent on whether we consider the reconstruction error in Frobenius or Spectral perspective [34]. However, for online PCA, the projection plane varies based on our consideration of reconstruction error in Frobenius norm [17] or in Spectral norm form [45]. As we can not build the projection plane based on the whole input vectors  $X$  for online PCA and the selection of best principal components depends on the fixed error bound, so the projection plane varies based on Frobenius [17] or Spectral form [45] of reconstruction error. But our model is different from existing online PCA algorithms as we utilize the concept of a sliding window to choose the best projection plane to capture the distribution of current data in reduced dimension.

## 2.5 Bhattacharyya Distance (BD) Method

In order to check the statistical similarity between the input and the output data from PCA computation, a number of different statistical measures are available, such as Mahalanobis Distance [58], Kullback-Leibler (KL) divergence

metric [48], Bhattacharyya Distance (BD) [14], Hellinger Distance [63], Kolmogorov-Smirnov statistic [56], etc. BD is related to Bhattacharyya Coefficient (BC) which is a measure of the amount of overlap between two statistical samples or populations. In this paper, we choose Bhattacharyya Coefficient to measure the similarity of two probability distributions as it is symmetric and bounded. The coefficient can be used to determine the relative closeness of the two samples being considered. It is a better statistical measure than KL-Divergence because KL is unbounded and non-symmetric. Mahalanobis Distance is a particular case of Bhattacharyya Distance when the standard deviations of the two classes are the same. Mahalanobis Distance tends to zero when two classes have similar means but different standard deviations, whereas Bhattacharyya Distance grows depending on the difference between the standard deviations. Hence, Bhattacharyya Distance can effectively capture the difference between two classes with different standard deviations and is therefore a more reliable metric than Mahalanobis Distance.

For probability distributions  $p$  and  $q$  over the same domain  $X$ , Bhattacharyya Distance (BD) is defined as:

$$D_B(p, q) = -\ln(BC(p, q)) \quad (11)$$

where

$$BC(p, q) = \int \sqrt{p(x)q(x)} dx \quad (12)$$

is the Bhattacharyya Coefficient for continuous probability distributions.

$0 \leq BC \leq 1$  and  $0 \leq D_B \leq \infty$ . Bhattacharyya Distance,  $D_B$  does not obey the triangle inequality [44] and is unbounded. So in this paper, we choose Bhattacharyya Coefficient,  $BC$  to measure the similarity between two probability distributions as it is symmetric and bounded. Bhattacharyya Coefficient (BC) has the following properties [66]:

- $BC(p, q) = 1$  if and only if  $p = q$  and
- $BC(p, q) = 0$  if and only if  $p$  is orthogonal to  $q$
- $BC(p, q) = BC(q, p)$ , meaning  $BC$  is symmetric

Hence, a value of Bhattacharyya Coefficient,  $BC$  close to 1 indicates that the two distributions are statistically similar, whereas a value close to 0 indicates dissimilarity. We first calculate Bhattacharyya Distance,  $D_B$  and then compute Bhattacharyya Coefficient,  $BC$  from

$$BC = \frac{1}{e^{D_B}} \quad (13)$$

For multivariate normal distributions,  $p_i = \eta(\mu_i, \Sigma_i)$ ,  $D_B(p, q)$  is given by

$$D_B(p, q) = \frac{1}{8}(\mu_1 - \mu_2)^T \Sigma^{-1}(\mu_1 - \mu_2) + \frac{1}{2} \ln \left( \frac{\det \Sigma}{\sqrt{\det \Sigma_1 \det \Sigma_2}} \right) \quad (14)$$

where  $\mu_1, \Sigma_1, \mu_2$  and  $\Sigma_2$  are the means and covariances of  $p$  and  $q$  respectively and  $\Sigma = \frac{\Sigma_1 + \Sigma_2}{2}$ .

### 3 RELATED WORKS

The state-of-the-art dimensionality reduction methods can be classified into Feature Extraction (FE) [51][52][80] and Feature Selection (FS) [16][47][46][86] approaches. In general, FE approaches are more effective than the FS techniques [32][81][85] for real world dimensionality reduction problems [37][40][51][52]. Linear algorithms [36], such as Maximum Margin Criterion (MMC) [52], Principal Component Analysis (PCA) [43], and Linear Discriminant Analysis (LDA)[59][81], project the high dimensional data to a lower-dimensional space by linear transformations

based on some criteria. On the other hand, nonlinear algorithms[12], such as Locally Linear Embedding (LLE) [71], ISOMAP[78], and Laplacian Eigenmaps project the original data by nonlinear transformations. Linear techniques draw more interest and are used widely due to their efficiency in comparison to the non-linear ones.

Many variants of PCA algorithms have been proposed recently for multifarious real-life applications such as scalable PCA [35], sSketch [77], probabilistic PCA [79], sparse PCA [7], incremental PCA [8] [54], etc. Incremental PCA (IPCA) [8] [54] is a well-studied incremental learning algorithm that has been proposed to compute principal components without the covariance matrix [80] [64] [73]. The main difference among them is the incremental representation of the covariance matrix. The latest version of IPCA is called Candid Covariance-free Incremental Principal Component Analysis (CCIPCA) [83] which computes the principal components of samples incrementally without estimating the covariance matrix, for real-time applications. It has higher estimation accuracy and faster computing speed. But CCIPCA's estimation accuracy is limited by the number of samples, or in other words, the number of iterations. The principal components (PCs) of CCIPCA will converge to batch mode result when new data are added in training set continually. However, it does not guarantee an acceptable estimation error upper bound between two adjacent iterations. Therefore in learning tasks, where the training set is often changed frequently, the unwarranted error may lead to unreasonable PCs which may cause failure of the task.

Another line of interesting work is unsupervised window-based change detection based on comparing the distribution in a current stream window with a reference distribution [15] [49] [76] and updating the reference window with the current stream whenever a change is detected. [69] uses PCA to project the original multidimensional data stream into a lower dimensional space, which constitutes the reference window. Then the new incoming data is projected along the previously computed principal components (PCs) to build the test window. The test and the reference windows are then compared and a change score is assigned. If the change score is above a threshold, a change point is detected and the reference window is updated by the lower dimensional representation of the current window. Hence, in order to detect change points, it discretely identifies timesteps where the distribution change has occurred and updates the window **completely** at each change point. It captures the notion of “**rate of change**” at the current moment and will repeatedly detect changes. However in Real Time PCA, we do not want to lose information about **large** changes corresponding to previous distributions by discarding all the PCs of the reference window when a change has been detected. Therefore, Real Time PCA retains the **large PCs corresponding to previous distributions**, besides accounting for the **PCs of the current distribution**. This way we can capture the global trends in data besides capturing the major local trends and thereby model a better representation of the underlying distribution.

To the best of our knowledge, all previous work along this path have focused only on the current data to capture the notion of “**rate of change**” and to identify change points. This is the **first** paper that aims to provide a lower dimensional subspace representation for **streaming** data, that retains information about significant changes seen in **previous data** besides accommodating for the changes in **current data**. Besides, this is also the first paper to propose a model to give the eigenspace representation of streaming data whose distributions are changing over time.

## 4 MODEL FORMULATION

### 4.1 Problem Definition

In the real-time version of the PCA problem, at each timestep  $t$ , the input vector  $x_t$  of dimension  $d$ , is given input to the algorithm. The target dimension  $l$  and the window size  $k$  are specified as input parameters to the algorithm. As  $l$  denotes the dimension of the projected output in Real Time PCA, then  $l < d$  for dimensionality reduction. At timestep

$t$ , the input data  $x_t$  arrives and we build a block  $inpB_t$  with the latest  $k$  instances of data. Hence,  $inpB_t$  is of size  $k \times d$ . The goal of Real Time PCA is 1) to provide a low dimensional representation of the current distribution of the data using PCA, and 2) to retain the major components of the previous distributions seen so far. Therefore, we divide our problem formulation into the two following objectives:

- The first objective is to reduce the spectral norm of the reconstruction error with respect to  $inpB_t$  and generate the corresponding output vector  $y_t$  for timestep  $t$ ,  $t = 1, 2, \dots, n$ , before getting the next input  $x_{t+1}$ . This ensures that Real Time PCA provides a low dimensional representation of the current distribution of the data using PCA, which is our first objective. By current distribution, we are referring to the  $k$  instances of data within  $inpB_t$ , from timestep  $t - k + 1$  to timestep  $t$ . If  $U$  is the eigenspace representation of size  $d \times l$ , given by our algorithms, then the output projection of  $inpB_t$  is  $outB_t$ , where  $outB_t = inpB_t \times U$ . So  $outB_t$  is of size  $k \times l$ . The reconstruction of  $outB_t$  is given by  $outB_t \times U^T$ . Hence, the spectral norm of the reconstruction error is given by  $\|inpB_t - outB_t \times U^T\|_2^2$ . However,  $outB_t \times U^T$  can be simplified to  $inpB_t \times U \times U^T$  as  $outB_t = inpB_t \times U$ . Therefore, the expression for the spectral norm of the reconstruction error becomes  $\|inpB_t - inpB_t \times U \times U^T\|_2^2$ . For simplicity of notation, we can represent  $inpB_t$  by  $B_t$ . Therefore, the simplified expression for the spectral norm of the reconstruction error becomes  $\|B_t(I - UU^T)\|_2^2$ .
- The second objective is to retain the major components pertaining to previous distributions of the data seen thus far. So our second objective is to maintain a set of eigenvectors  $V$  such that  $V \subseteq U$  and the eigenvalues corresponding to the eigenvectors in  $V$  are all greater than the eigenvalues of the eigenvectors in  $U - V$ . For every data instance, we compute the set  $V$  from the set  $U$ . Hence,  $V$  represents the set of eigenvectors corresponding to the largest eigenvalues that we are retaining from previous data distributions. Therefore,  $0 \leq |V| \leq |U|$ . We know that  $|U| = l$ , because  $l$  is the target dimension and we are maintaining  $l$  eigenvectors in  $U$  at each timestep. However, if  $|V| < l$ , then the remaining  $l - |V|$  eigenvectors are computed from the reconstruction error of the current block, given by  $B_t(I - UU^T)$ . Therefore, the set  $V$  controls how much characteristics we retain from the previous distributions. In the following sections, we will use  $r$ , where  $r = |V|$ , to denote the number of eigenvectors we are retaining at each timestep  $t$ .

Hence the input and output of a desired real-time PCA algorithm follows:

- **Input:** Set of vectors  $X = [x_1, \dots, x_n]$  in  $\mathbb{R}^{n \times d}$ , a target dimension  $l$ ,  $l < d$  and a window size  $k$ .
- **Output:** Set of vectors  $Y = [y_1, \dots, y_n]$  in  $\mathbb{R}^{n \times l}$  that reduces the reconstruction error  $\|B_t(I - UU^T)\|_2^2$ , for  $t = 1, 2, \dots, n$ .

The explanation of the mathematical notation of the different data structures used for computation are as follows:

- $U \leftarrow$  eigenvector matrix, initialized as an all zeros matrix of size  $d \times l$ .
- $B \leftarrow$  block, initialized as an all zeros matrix of size  $k \times d$ .
- $\sigma_{arr} \leftarrow$  array of eigenvalues corresponding to eigenvectors in  $U$ , initialized as an all zeros array of size  $l$ .

All the algorithms formulated in this paper use a function named *constructWindow* to construct a window  $B_t$  by incorporating the input data  $x_t$  at timestep  $t$  into  $B_t$  and discarding the data which arrived at timestep  $t - k$  from  $B_t$ . While the concept of a block size is fairly simple, its impact on both storage performance and cost is profound. The block size  $k$  is usually set according to the application domain and previous experience. The user sets this parameter depending on whether to monitor the long/short term trends and the application's sensitivity against changes [4]. A small window size enables better detection of short-term trends and reduces the delay but may lead to a large number

of false positives. A large window size makes the algorithm more robust to short-term changes and thereby focuses on the global components but may miss out on recent small changes [4]. The choice of block size may also be restrained by factors other than the application, for example memory constraint, computational complexity constraint. If the memory in the RAM is limited, then the maximum block size will be governed by the availability of memory. Similarly, the computational complexity of a large block size will be high and the computational time might exceed the rate at which data arrives. In such a case, it is suggested to tune the block size in a way that permits the computation to be completed before the next data arrives.

#### 4.2 Real Time PCA using Online PCA with Spectral Bound (ROP)

We first examine whether Real Time PCA can be implemented using the existing architecture of online PCA. Algorithm 1 is the real-time version of the **Fixed Error: Conceptual Algorithm (FECA)** and **Fixed-Error : Space-Efficient Algorithm (FESEA)** proposed in [45]. The window size  $k$  and the reconstruction error  $\Delta$  are specified as input parameters. For data  $x_t$  at timestep  $t$ , we construct a block  $B_t$  with the latest  $k$  instances of data. We compute the spectral norm of the reconstruction error given by  $\|B_t(I - UU^T)\|_2^2$  in the **while** loop in line 5. If the spectral norm is greater than or equal to  $\Delta$ , we perform PCA on  $B_t(I - UU^T)$  using Eigenvalue Decomposition and append the eigenvector corresponding to the largest eigenvalue to  $U$ . We continue this process until the spectral norm of the reconstruction error  $\|B_t(I - UU^T)\|_2^2$  is less than  $\Delta$ . We repeat this procedure for input data  $x_t$  at each timestep  $t$ , for  $t = 1, 2, \dots, n$ .

Both online and real-time PCA compute PCA for streaming data. Although the algorithms presented in [45], specifically **FECA** and **FESEA**, appear to be quite similar to the ones presented here, there is a significant difference between the motives of the two. The goal of online PCA is to minimize the reconstruction error  $\|(X - Y\varphi^T)\|_F^2$  [17] or  $\|(X - Y\varphi^T)\|_2^2$  [45] with respect to  $X$ , which represents the entire dataset. In **FECA** of [45], we compute the pca on  $X(I - UU^T)$  where  $X$  represents the entire history of data seen till date. In **FESEA** of [45], the entire history of data seen till date is sketched into a block before the data is computed upon. The sketch summarizes the entire data in  $X$  into a smaller sized block. But in Real Time PCA, we are interested to provide the best approximation for the recent data stored in the current block  $B_t$ , not for the whole data or its summary constructed from the overall data seen so far. Therefore in Algorithm 1 (ROP), which is the real time counterpart of **FECA** and **FESEA**, we are computing the pca on  $B_t(I - UU^T)$ , where  $B_t$  holds only the latest  $k$  instances of data. Hence we are computing pca on most recent data from timestep  $t - k + 1$  to timestep  $t$ . Hence, the goal of real-time PCA is to reduce the reconstruction error  $\|(B_t - \varphi Y)\|_2^2$  with respect to  $B_t$  and not  $X$ . So real-time PCA aims to reduce the reconstruction error of the **current** block  $B_t$ , whereas online PCA tries to minimize the reconstruction error with respect to the **entire** dataset.

We now discuss why Algorithm 1 (ROP) is not suited for a real-time framework.

- Algorithm 1 (ROP) always **appends** eigenvectors to  $U$  but **never discards** them. Therefore, if  $\Delta$  is too small, dimensions will be added too frequently and may exceed input dimension after a couple of iterations. This proliferation of dimensions defeats the purpose of PCA.
- The reconstruction error  $\Delta$  must be specified as an input parameter for Algorithm 1 (ROP). But it is difficult to estimate a suitable value for this parameter as it varies across applications and require domain expertise. Besides, specification of  $\Delta$  might lead to a proliferation of dimensions, as mentioned in the point above. Therefore,  $\Delta$  cannot be user defined.

**Algorithm 1:** Real Time PCA using Online PCA with Spectral Bound (ROP)

---

**Input:**  $X, \Delta, k$   
**Output:**  $Y$

```

1  $U \leftarrow$  all zeros matrix
2  $B \leftarrow$  a window of size  $k \times d$ 
3 for  $x_t \in X$  do
4    $B_t = \text{constructWindow}(x_t, B_{t-1})$ 
5   while  $\|B_t(I - UU^T)\|_2^2 \geq \Delta$  do
6     Compute pca of  $B_t(I - UU^T)$  and append the top eigenvector to  $U$ 
7   end
8    $y_t = x_t U$ 
9 end

```

---

- Hence, we need to specify  $l$ , which is the target dimensionality, to ensure that the output dimension does not increase indefinitely. This ensures that dimensionality reduction takes place at all costs, no matter how random the underlying distribution is. But then, the  $d - l$  discarded dimensions will generate the reconstruction error. There is no known method to formulate this error. Hence, we need to come up with a different check for the condition of the **while** loop (line 5 in Algorithm 1): one that will shift the axes of the ellipsoid to a space that reflects the current nature of the distribution of data, within a reasonable number of iterations.

**4.3 Real Time PCA using Block Offline PCA (BPCA)**

To account for the limitations of Algorithm 1 (ROP), we propose the simplest naïve model of computing PCA of every block  $B_t$  from scratch, for  $t = 1, 2, \dots, n$ , in Algorithm 2 (BPCA). Here we focus only on our first objective of Real Time PCA, which is to reduce the spectral norm of the reconstruction error of the current block  $B_t$  to provide a low dimensional representation of the current distribution of the data. The window size  $k$  and the target dimension  $l$  are specified as input parameters instead of the reconstruction error  $\Delta$  as in Algorithm 1 (ROP). For data  $x_t$  at timestep  $t$ , we construct a block  $B_t$  with the latest  $k$  instances of data. We perform PCA on the reconstruction error, given by  $B_t(I - UU^T)$ , using Eigenvalue Decomposition.  $U$  is formed from the eigenvectors corresponding to the largest  $l$  eigenvalues. We conduct this computation once for each input data  $x_t$ , unlike in Algorithm 1 (ROP), where the number of times this computation is repeated for each input data depends on how often the spectral norm of the reconstruction error  $\|B_t(I - UU^T)\|_2^2$  is greater than or equal to the specified reconstruction error  $\Delta$ , i.e. the number of times the **while** loop iterates for each data.

This model is better than the previous one built in various aspects which are mentioned as follows:

- With the specification of target dimension  $l$ , it is now possible to specify that the top  $l$  eigenvectors must be used to reconstruct each block. Hence, unlike Algorithm 1 (ROP), dimensionality reduction is guaranteed to occur at all costs, if  $l < d$ .
- The specification of  $l$  circumvents the problems associated with the specification of a suitable value of  $\Delta$ . Although suitable specification of both  $\Delta$  and  $l$  requires domain expertise, the computational resources available may aid to set an upper limit for  $l$ . The memory constraints of the computing machines can be used to determine a suitable value for  $l$  because output data is of size  $n \times l$ . However, constraints on computational resources provide no

**Algorithm 2:** Real Time PCA using Block Offline PCA (BPCA)

---

**Input:**  $X, k, l$   
**Output:**  $Y$

```

1  $U \leftarrow$  all zeros matrix of size  $d \times l$ 
2  $B \leftarrow$  a window of size  $k \times d$ 
3 for  $x_t \in X$  do
4    $B_t = \text{constructWindow}(x_t, B_{t-1})$ 
5    $U \leftarrow$  eigenvectors corresponding to the top  $l$  eigenvalues computed from  $\text{pca of } B_t(I - UU^T)$ 
6    $y_t = x_t U$ 
7 end

```

---

clue regarding what could be a suitable value for  $\Delta$ . Hence, it is easier to specify  $l$  than to specify  $\Delta$ , if domain knowledge is limited.

- **FECA** [45] maintains the entire history of data  $X_{1:t}$  upto timestep  $t$  but Algorithm 2 (BPCA) stores the most recent  $k$  data rows. Hence, Algorithm 2 (BPCA) is less memory intensive than **FECA** [45].
- Both **FESEA** [45] and **Adaptive Error: Time Efficient Algorithm** [45] require covariance sketch of the data matrix to construct the block. But block construction is a simple insertion and deletion operation in Algorithm 2 (BPCA). Hence Algorithm 2 (BPCA) is also less time intensive than both **FESEA** [45] and **Adaptive Error: Time Efficient Algorithm** [45]

We now highlight some of the major drawbacks of Algorithm 2 (BPCA). In BPCA, all the eigenvectors in  $U$  are computed from the reconstruction error of the current block  $B_t$ . This presents some problems mentioned below:

- The biggest drawback of this algorithm is that it completely ignored the second objective of Real Time PCA which is to **retain the major components** of the previous distributions seen so far. In Algorithm 2 (BPCA),  $U$  stores only the eigenvectors computed from the reconstruction error of the current block  $B_t$ . It **does not** retain any component corresponding to previous distributions. Therefore,  $r$ , which is the number of eigenvectors we retain at each timestep is 0 for Algorithm 2 (BPCA). Thus, the goal of Real Time PCA is partially compromised.
- We want to avoid any change in  $U$  when the underlying input distribution stays uniform. Window  $B$  is maintained as such that only one row of data varies at each iteration. Hence, even if the input distribution stays the **same**,  $U_t$  ( $U$  at timestep  $t$ ) will differ slightly from  $U_{t-1}$  because one row of data is being updated at every iteration. This adds to the instability of the system as  $U$  is continuously changing at each iteration regardless of the change in the underlying distribution. Therefore, we should focus on building a system that preserves as much of  $U$  as possible at each iteration, provided the underlying distribution stays the same.
- If we let  $U$  change at each iteration, the  $U$  matrix will encapsulate even the slightest change in distribution, that we may not be particularly interested about. Such minor changes may result from **noise** or **slight shifts** in the local trend, that we wish to ignore. Our sole aim of performing Real Time PCA is to capture the major underlying distributions that can be identified from the major components of the eigenvectors in  $U$ . Hence,  $U$  should be updated only when there has been a major change in the underlying distribution of the input. Therefore, the time when  $U$  is updated should be a useful indicator as to when there has been a change in the distribution of the input data. If we let  $U$  change at each iteration, we may not be able to discretely identify timesteps at which major changes are occurring.

- The optimal window size parameter  $k$  is **indeterminate**. It varies across applications and may even fluctuate over the course of time depending on the changing nature of the underlying distribution. Therefore, the algorithm should be robust to the input parameter  $k$ . If we compute PCA from scratch for every window at each iteration, the quality of the output becomes dependent on  $k$ . However, if we want to make our algorithm invariant to the window size  $k$ , we should look to retain only the most significant eigenvectors in  $U$ , which will be replaced only when there has been a major change in the distribution of data. Therefore,  $U$  will be less dependant on block size,  $k$ .

Online PCA [45] tries to minimize the reconstruction error of the entire dataset and Algorithm 2 (BPCA) tries to reduce the reconstruction error for the current window  $B_t$  only. In the next section, we look to strike a balance between the two and come up with proper Real Time PCA algorithms that comply with both the objectives of Real Time PCA.

## 5 PROPOSED ALGORITHMS

In this section, we propose algorithms which implement PCA in real-time using a sliding window model. These algorithms address both the objectives of Real Time PCA unlike Algorithm 2 (BPCA) which failed to satisfy the second objective of retaining the major components of previous data distributions. This created the need to come up with algorithms that address both the objectives of Real Time PCA. We discuss the motivation and intuition behind each algorithm and analyze their time and space complexity. We justify how these algorithms overcome the drawbacks of the previous algorithms.

For the purpose of formulating our optimization function, let us consider  $B_t$  to be the input data,  $U_t$  to be the eigenvector matrix and  $O_t$  to be the corresponding output data at timestep  $t$ , such that  $O_t = B_t U_t$ . The goal of Real Time PCA is to reduce the reconstruction error  $\|B_t - (B_t O_t^+) O_t\|_F^2$ . The exact solution to this problem can be found (offline) by a partial Singular Value Decomposition (SVD). However, while the exact minimizer of  $\|B_t - (B_t O_t^+) O_t\|_F^2$  is also the minimizer of  $\|B_t - (B_t O_t^+) O_t\|_2^2$ , the same cannot be said about their approximate solutions. To make this point clear, consider an input matrix  $B_t$  whose first  $p$  singular values are equal to 1 and the rest are equal to  $1/2$ . We denote by  $\sigma_i$ , the  $i$ 'th singular value of  $B_t$  sorted in descending magnitude order. For this matrix,

$$\min_{O_t} \|B_t - (B_t O_t^+) O_t\|_F^2 = \sum_{i=p+1}^d \sigma_i^2 = (d-p)/4 \quad (15)$$

On the other hand, for any matrix  $O_t$ ,

$$\|B_t - (B_t O_t^+) O_t\|_F^2 \leq \|B_t\|_F^2 = (d-p)/4 + p \quad (16)$$

Here, any solution  $O_t$  is  $1 + \epsilon$  approximation so long as,  $d \geq 5p/\epsilon$ , where  $\epsilon$  is a constant. This is not the case when considering the spectral norm. The optimal  $O_t$  perfectly captures the signal and

$$\min_{O_t} \|B_t - (B_t O_t^+) O_t\|_2^2 = \sigma_{p+1}^2 = 1/4 \quad (17)$$

In sharp contrast to the above, obtaining  $O_t$  such that  $\|B_t - (B_t O_t^+) O_t\|_2^2 \leq 1/4 + \epsilon$  is far from trivial. It does not hold for a random  $O_t$  and it does not hold for  $O_t = S^T B_t$  where  $S^T$  is random as in [74] and [21].

The key idea in all the following three algorithms are similar with slight changes introduced in each subsequent algorithm.  $U$ , at any time, holds the eigenvectors that best represents the low-dimensional subspace of the current distribution and also accounts for the major components pertaining to previous distributions. PCA is always computed on the reconstruction error, given by  $B_t(I - UU^T)$ . This gives the eigenvectors (directions along which the reconstruction

**Algorithm 3:** Real Time PCA with Target Dimension (RP)

---

**Input:**  $X, k, l$   
**Output:**  $Y$

```

1  $U \leftarrow$  all zeros matrix of size  $d \times l$ 
2  $B \leftarrow$  a window of size  $k \times d$ 
3  $\sigma_{arr} \leftarrow$  all zeros array of size  $l$ 
4 for  $x_t \in X$  do
5    $B_t = \text{constructWindow}(x_t, B_{t-1})$ 
6    $min \leftarrow$  minimum of  $\sigma_{arr}$ 
7    $[\sigma, u] \leftarrow$  top eigenvalue and corresponding eigenvector computed from pca of  $B_t(I - UU^T)$ 
8   while  $\sigma > min$  do
9     Replace vector from  $U$  by  $u$  and  $min$  in  $\sigma_{arr}$  by  $\sigma$ 
10     $min \leftarrow$  minimum of  $\sigma_{arr}$ 
11     $[\sigma, u] \leftarrow$  top eigenvalue and corresponding eigenvector computed from pca of  $B_t(I - UU^T)$ 
12  end
13   $y_t = x_t U$ 
14 end

```

---

error occurs) and the eigenvalues (magnitude of error in each direction).  $u$  is the eigenvector corresponding to the largest eigenvalue computed from pca of  $B_t(I - UU^T)$ . We then compare whether the eigenvalue corresponding to  $u$  is greater than the smallest eigenvalue corresponding to any of the eigenvector in  $U$ . If this is the case, then we replace the eigenvector corresponding to the smallest eigenvalue in  $U$  by  $u$ . This ensures that the major eigenvectors of the previous distributions are retained in  $U$  and the largest eigenvector given by the reconstruction error of the current block is also incorporated into  $U$ , only if this eigenvector is larger than any of the existing eigenvectors in  $U$ . This way we are complying with both the objectives of Real Time PCA which are to reduce the spectral norm of the reconstruction error of the current block  $B_t$  and to retain the major components of previously seen distributions, as stated in Section 4.1 *Problem Definition*.

The first Algorithm 3 simply uses a window  $B$ , that slides past the input data, trying to ensure that the largest eigenvalue given by the current reconstruction error  $B_t(I - UU^T)$  is not bigger than the smallest eigenvalue in current  $U$ . The second Algorithm 4 is similar but selects the  $l$  largest eigenvalues chosen from the union of the set of newly computed eigenvalues (computed from the current reconstruction error  $B_t(I - UU^T)$ ) and the existing eigenvalues. This is faster than Algorithm 1. The third Algorithm 5 is similar to Algorithm 3 (RP), except that it computes the eigenvectors corresponding to the current reconstruction error  $B_t(I - UU^T)$  and compares its largest eigenvalue with the smallest eigenvalue from the current set  $U$ , only **once** at each iteration. As we experimentally show, our third algorithm is time-efficient and also resistant to noisy samples.

### 5.1 Real Time PCA with Target Dimension (RP)

In Algorithm 3 (RP), at each iteration, we compute the top eigenvalue,  $\sigma$ , and its corresponding eigenvector,  $u$ , from the reconstruction error  $B_t(I - UU^T)$ .  $\sigma_{arr}$  holds eigenvalues corresponding to the eigenvectors currently in  $U$ .  $min$  is the minimum eigenvalue currently in  $\sigma_{arr}$ . If  $\sigma$  is greater than  $min$ , we update  $\sigma_{arr}$  and  $U$  by replacing  $min$  in  $\sigma_{arr}$  and its corresponding eigenvector in  $U$  by  $\sigma$  and  $u$ , respectively. We continue this process of comparison and replacement in a **while** loop (line 8 in Algorithm 3) until the largest eigenvalue  $\sigma$ , given by the reconstruction error of the current

**Algorithm 4:** Time-Efficient Real Time PCA with Target Dimension (RPT)

---

**Input:**  $X, k, l$   
**Output:**  $Y$

```

1  $U \leftarrow$  all zeros matrix of size  $d \times l$ 
2  $B \leftarrow$  a window of size  $k \times d$ 
3  $\sigma_{arr} \leftarrow$  all zeros array of size  $l$ 
4  $S_U \leftarrow$  all zeros matrix of size  $d \times d$ 
5  $S_\sigma \leftarrow$  all zeros array of size  $d$ 
6 for  $x_t \in X$  do
7    $B_t = \text{constructWindow}(x_t, B_{t-1})$ 
8    $S_\sigma, S_U \leftarrow$  set of  $d$  eigenvalues and their corresponding eigenvectors computed from pca of  $B_t(I - UU^T)$ 
9    $\sigma_{arr} \leftarrow$  top  $l$  eigenvalues from  $\sigma_{arr} \cup S_\sigma$ 
10   $U \leftarrow$  eigenvectors from  $U \cup S_U$  corresponding to the top  $l$  eigenvalues
11   $y_t = x_t U$ 
12 end

```

---

window  $B_t$ , is smaller than  $\min$ , the smallest eigenvalue in  $\sigma_{arr}$ . Therefore, by incorporating the eigenvector,  $u$ , which is the direction along which the largest reconstruction error occurs, we are trying to reduce the reconstruction error. The parameter  $r$  in this algorithm can be a minimum of 0, when no eigenvectors currently in  $U$  are retained and the **while** loop in Line 8 iterates at least  $l$  times to replace all the existing eigenvectors in  $U$  at any particular timestep  $t$ . The maximum value of parameter  $r$  can be  $l$  which occurs when all the eigenvectors in  $U$  are retained because the algorithm never executed the **while** loop at that particular timestep  $t$ . This occurs when the top eigenvalue  $\sigma$  computed in Line 7 is smaller than  $\min$ , which is the smallest eigenvalue in  $\sigma_{arr}$ .

It might be argued as to why eigenvectors with larger eigenvalues are considered more important than the ones with smaller eigenvalues. In fact change detection based on PCA in multidimensional data (not streams) was considered using PCs with small eigenvalues because they are more likely to be affected by a change [49]. However, in this scenario changes are assumed to occur in projected data on principal components with uniform distributions. But in reality, changes occur in the original features' values. It is difficult to estimate and compare PCs with small variances due to their sensitivity to sample size and density model parameters. Thus, discarding PCs with small variances reduces computational cost and reduces false positive rates. Besides, retaining the largest variances encodes more information about the underlying distribution and discards the small insignificant variations in the data. Hence, all our algorithms prioritize the **larger** PCs compared to the smaller PCs.

## 5.2 Time-Efficient Real Time PCA with Target Dimension (RPT)

In Algorithm 4 (RPT), at each iteration, we compute the set of  $d$  eigenvalues,  $S_\sigma$  and their corresponding eigenvectors,  $S_U$  from  $B_t(I - UU^T)$ . Then we choose the  $l$  largest eigenvalues from the  $l + d$  eigenvalues obtained from  $\sigma_{arr} \cup S_\sigma$  and their corresponding eigenvectors from  $U \cup S_U$ . We then assign the chosen  $l$  eigenvalues and their corresponding eigenvectors to  $\sigma_{arr}$  and  $U$ , respectively. We do this once at every iteration. The motive here is to retain the eigenvectors corresponding to the **largest** eigenvalues at all times because these are the most important directions that capture the change. By including all those eigenvectors, whose directions contain the largest reconstruction error, we are trying to reduce the reconstruction error. The parameter  $r$  can take a minimum value of 0 at any particular timestep  $t$ , which occurs when any of the  $l$  eigenvalues among the  $d$  eigenvalues computed from the reconstruction error of the

**Algorithm 5:** Robust Time-Efficient Real Time PCA with Target Dimension (RPTO)

---

**Input:**  $X, k, l$   
**Output:**  $Y$

```

1  $U \leftarrow$  all zeros matrix of size  $d \times l$ 
2  $B \leftarrow$  a window of size  $k \times d$ 
3  $\sigma_{arr} \leftarrow$  all zeros array of size  $l$ 
4 for  $x_t \in X$  do
5    $B_t = \text{constructWindow}(x_t, B_{t-1})$ 
6    $min \leftarrow$  minimum of  $\sigma_{arr}$ 
7    $[\sigma, u] \leftarrow$  top eigenvalue and corresponding eigenvector computed from pca of  $B_t(I - UU^T)$ 
8   if  $\sigma > min$  then
9     Replace vector from  $U$  by  $u$  and  $min$  in  $\sigma_{arr}$  by  $\sigma$ 
10  end
11   $y_t = x_t U$ 
12 end

```

---

current block,  $B_t(I - UU^T)$ , are all greater than the largest eigenvalue in  $\sigma_{arr}$ . As a result all the  $l$  eigenvalues and their corresponding eigenvectors in  $\sigma_{arr}$  and  $U$ , respectively, are replaced by the largest  $l$  eigenvalues and their corresponding eigenvectors from  $S_\sigma$  and  $S_U$ , respectively. The maximum value of  $r$  at any particular timestep  $t$  can be  $l$  which occurs when the  $l$  eigenvalues in  $\sigma_{arr}$  are all greater than the largest eigenvalue computed from the reconstruction error of the current block,  $B_t(I - UU^T)$ . Therefore, all the  $l$  eigenvalues and their corresponding eigenvectors in  $\sigma_{arr}$  and  $U$ , respectively, are retained.

Algorithm 4 (RPT) is **faster** than Algorithm 3 (RP) because of the absence of the **while** loop, which may be executed several times during each iteration in Algorithm 3 (RP). Besides, during each iteration of the **while** loop in Algorithm 3 (RP), of  $B_t(I - UU^T)$  needs to be calculated, which is a computational bottleneck.

### 5.3 Robust Time-Efficient Real Time PCA with Target Dimension (RPTO)

In Algorithm 5 (RPTO), at each iteration, we compute the top eigenvalue  $\sigma$  and its corresponding eigenvector  $u$  from the reconstruction error  $B_t(I - UU^T)$ .  $\sigma_{arr}$  holds eigenvalues corresponding to the eigenvectors currently in  $U$ .  $min$  is the minimum eigenvalue currently in  $\sigma_{arr}$ . If  $\sigma$  is greater than  $min$ , we update  $\sigma_{arr}$  and  $U$  by replacing  $min$  in  $\sigma_{arr}$  and its corresponding eigenvector in  $U$  by  $\sigma$  and  $u$ , respectively. Therefore, by incorporating the eigenvector,  $u$ , which is the direction along which the largest reconstruction error occurs, we are trying to reduce the reconstruction error. We do this once at every iteration. Therefore, the minimum value of  $r$  at any particular timestep  $t$  can be  $l - 1$ , which occurs when  $\sigma > min$ . Therefore,  $l - 1$  eigenvalues and their corresponding eigenvectors are retained from  $\sigma_{arr}$  and  $U$ , respectively and the one remaining eigenvalue and eigenvector comes from the largest eigenvalue,  $\sigma$  and its corresponding eigenvector,  $u$ , computed from the reconstruction error of the current block,  $B_t(I - UU^T)$ . The maximum value of  $r$  at any particular timestep  $t$  can be  $l$  which occurs when  $\sigma \leq min$ . Therefore, all the  $l$  eigenvalues in  $\sigma_{arr}$  and their corresponding eigenvectors in  $U$  are retained.

This algorithm is also **more time-efficient** as compared to Algorithm 3 (RP) because of the absence of the **while** loop. Besides, this algorithm is also **robust** to noise, as compared to Algorithm 3 (RP). In Algorithm 3 (RP), a random noise sample may replace a huge proportion of eigenvectors in  $U$  and this will change the eigenspace completely, which is undesirable. But Algorithm 5 (RPTO) updates at most one eigenvector from  $U$  at each iteration. Hence, even if it

Table 1. Values of Parameter  $r$  for each algorithm

| Algorithm | Value of Parameter $r$ |
|-----------|------------------------|
| BPCA      | $r = 0$                |
| RP        | $0 \leq r \leq l$      |
| RPT       | $0 \leq r \leq l$      |
| RPTO      | $l - 1 \leq r \leq l$  |

encounters a noise sample,  $U$  will not be drastically modified and the majority of the eigenvectors will still be the same. However, if it was **not** a noise sample but the **emergence** of a new trend, then there would be a series of successive samples over the next couple of iterations, belonging to the latest distribution. Hence, Algorithm 5 (RPTO) will update  $U$  over the next couple of iterations to capture the eigenspace of the new distribution. The key assumption here is, whenever a new trend emerges, there will be a series of successive samples generated from the new distribution. But if it is a noise, then there will be just a few such samples randomly scattered around the distributions. Noise samples are not generated from a single distribution over a series of successive iterations. Thus Algorithm 5 (RPTO) can capture the latest distribution while being robust to noise.

#### 5.4 Theoretical bounds on $r$

Table 1 summarizes the inequalities that  $r$  obey for each of the four algorithms that we proposed, namely Algorithm 2 (BPCA), 3 (RP), 4 (RPT), and 5 (RPTO). We have discussed each of the bounds following the algorithms in Section 4.3, 5.1, 5.2 and 5.3, respectively.

A suitable value of  $r$  may vary across applications and over time. Therefore, it might be difficult for a user to estimate a suitable value of  $r$  for a particular application, at any given timestep. Hence, we do not declare  $r$  to be a user defined parameter but allow our algorithms to decide on a suitable value and adapt itself over time depending on the nature of and the change in data distributions.

#### 5.5 Time and Space Complexity

In this section, we measure the time and space complexity of our proposed algorithms.

- **Time Complexity:** The time complexity of our algorithms are as follows:

- In Algorithm 3 (RP), the **while** loop may iterate an indefinite number of times for each data instance. Therefore, it is not possible to formulate an expression for its time complexity.

The time complexity of  $pca(X, l)$  computed via  $svd$ , for  $l$  largest eigenvalues, where  $X$  is a data matrix of dimension  $n \times d$ , is  $O(l^2d)$  [41]. We compute  $svd$  on  $B_t(I - UU^T)$ . It has a size of  $k \times d$ .

- In Line 8 of Algorithm 4 (RPT), we are computing all the  $d$  eigenvalues and their corresponding eigenvectors from the  $pca$  on  $B_t(I - UU^T)$ . For this line, the target dimension is  $d$ . Hence, the time complexity of this line is  $O(d^2d)$ , which is  $O(d^3)$ . The time complexity of the other lines is of order  $O(1)$ . Because the algorithm iterates over  $n$  instances of the  $n \times d$  sized dataset, so the total runtime for this algorithm is  $O(nd^3)$ .
- In Line 7 of Algorithm 5 (RPTO), we are computing only the top eigenvalue and its corresponding eigenvector from the  $pca$  on  $B_t(I - UU^T)$ . So the target dimension in this line is 1. Hence, the time complexity of this line is  $O(1^2d)$ , which is  $O(d)$ . The time complexity of the other lines is of order  $O(1)$ . Because the algorithm iterates over  $n$  instances of the  $n \times d$  sized dataset, so the total runtime for this algorithm is  $O(nd)$ .

Table 2. Time and Space Complexity

| Algorithm | Time Complexity | Space Complexity |
|-----------|-----------------|------------------|
| RP        | Indefinite      | $(n + d)l$       |
| RPT       | $nd^3$          | $(n + d)l$       |
| RPTO      | $nd$            | $(n + d)l$       |

- **Space Complexity:** The memory requirements of each of the data structure and the final output of our proposed algorithms are as follows:

- $U$  requires  $O(dl)$  memory.
- Window  $B$  requires  $O(kd)$  memory.
- $\sigma_{arr}$  requires  $O(l)$  memory.
- Output  $Y$  requires  $O(nl)$  memory, where  $n$  is the number of incoming data rows and  $l$  is the target dimension.
- Additionally, Algorithm 4 (RPT) requires  $O(l)$  memory to store the eigenvalues in  $S_\sigma$  and  $O(dl)$  memory to store the corresponding eigenvectors in  $S_U$ .

So the total space complexity of our proposed algorithms is  $(n + d)l$ .

Table 2 summarizes the time and space complexity of the three algorithms that we have proposed.

## 5.6 Justification of Our Proposed Algorithms

In this section, we discuss how our proposed algorithms combat the limitations of Algorithm 1 (ROP) and Algorithm 2 (BPCA).

Online PCA and Algorithm 1 (ROP) specifies error bound,  $\Delta$  as an input parameter. We already mentioned the problems associated with specifying  $\Delta$ , which led us to specify  $l$  as the alternative option instead. This engendered a further problem of formulating a suitable condition for the **while** loop without the specification of  $\Delta$ . Algorithm 2 (BPCA) did not encounter this problem despite specifying  $l$  instead of  $\Delta$ , because there was no **while** loop required. However, our proposed algorithms in this section bypassed this problem by using a different condition: compare the largest eigenvalue, computed from  $B_t(I - UU^T)$ , with the minimum eigenvalue stored in  $\sigma_{arr}$ . Therefore, we can safely claim that our proposed algorithms counter all the drawbacks of Algorithm 1 (ROP) like Algorithm 2 (BPCA) does, including the one that BPCA did not, i.e, the one we just mentioned.

We now focus on how our proposed algorithms overcome the limitations of Algorithm 2 (BPCA). Our proposed algorithms update  $U$  only when there has been a **major change** in the underlying distribution, unlike Algorithm 2 (BPCA) which modified  $U$  at every iteration. As a result of this modification, our proposed algorithms are:

- **Stable:** Algorithm 2 (BPCA) modified  $U$  at every iteration when new data arrived, even if there was **no change** in the underlying distribution. But our proposed algorithms update  $U$  only when there has been a **major change** in the underlying distribution. We identify such **major changes** when the **largest** eigenvalue computed from  $B_t(I - UU^T)$  **exceeds** the **minimum** eigenvalue, currently stored in  $\sigma_{arr}$ . Hence,  $U$  stays same when the underlying distribution stays relatively constant over time and changes only when there has been a **major change** in the underlying distribution. Therefore, the time when  $U$  is updated is a suitable indicator of when the underlying distribution has changed **significantly**. Infact, we will experimentally illustrate this feature of our algorithms through spectral diagrams in the following section.

- **Robust to noise and slight shifts in local trend:** Because  $U$  changed at each iteration in Algorithm 2 (BPCA), it also encapsulated changes in the underlying distribution that occurred due to noise or slight shifts in local trend that we wished to ignore. Our proposed algorithms do not let  $U$  change at every iteration and is therefore robust to insignificant changes as compared to Algorithm 2 (BPCA).
- **Conform to the goal of Real Time PCA:** As mentioned previously, the goal of Real Time PCA is to **retain the major components** of the previous distributions seen so far, besides providing a low dimensional representation of the **current distribution** of the data. But Algorithm 2 (BPCA) does not conform to our goal as it discards all the eigenvectors from  $U$  and populates it with the eigenvectors computed from the reconstruction error of the current block  $B_t$ . However, all our proposed algorithms smartly update  $U$  by comparing the largest eigenvalue computed from the reconstruction error of the current block  $B_t$  with the minimum eigenvalue stored in  $\sigma_{arr}$ , thereby ensuring that only the largest eigenvectors from the ones currently in  $U$  and the ones computed from the reconstruction error of the current block  $B_t$  are retained in  $U$ . Algorithm 5 (RPTO) will always retain all the eigenvectors corresponding to previous distributions, except the smallest eigenvector which maybe replaced at an iteration. Algorithm 3 (RP) and Algorithm 4 (RPT) will preserve some of the eigenvectors corresponding to previous distributions while replace others with the ones computed from the reconstruction error of the current block  $B_t$ . Only in the rare case where there has been a drastic change in the underlying distribution will these algorithms update all the eigenvectors in  $U$ . Hence, all our algorithms conform to our formulated goal of Real Time PCA by **retaining the major components** of the previous distributions seen so far, besides providing a low dimensional representation of the **current distribution** of the data.
- **Less reliant on the unstable window size parameter,  $k$ :** Algorithm 2 (BPCA) suffered from its dependency on the window size parameter  $k$ , which arises because the newly computed eigenvectors computed from the reconstruction error of the current block  $B_t$  replaces all the eigenvectors in  $U$ , i.e, those that were computed from the reconstruction error of  $B_{t-1}$ . However in Algorithm 3 (RP), the number of eigenvectors, replaced at each timestep, depends on the magnitude of the underlying change in distribution. Algorithm 4 (RPT) also retains some of the eigenvectors from previous iterations and incorporates others computed from the reconstruction error of the current block  $B_t$ , depending on their relative magnitudes. And Algorithm 5 (RPTO) replaces at most one eigenvector at a time. Therefore, unlike Algorithm 2 (BPCA), which updates all the eigenvectors in  $U$  completely at each iteration, the algorithms proposed in this section retain some eigenvectors corresponding to previous distributions. The number of eigenvectors retained and the number of eigenvectors replaced depend on the nature of the underlying change in the distribution. Therefore, our proposed algorithms are robust to the input block size  $k$  and works well when a suitable estimation of optimal block size  $k$  is difficult due to limited knowledge of the application domain.

## 6 EXPERIMENTAL EVALUATION

In this section, we first describe our experimental settings. Then we present a rigorous evaluation of the performance of our algorithms in terms of their bandwidth and error measure. We compare the results among the different algorithms graphically. Finally, we verify the correctness of our algorithms using Spectral Diagram and Bhattacharyya Coefficient.

### 6.1 Experimental Settings

We perform experiments on both synthetic and real datasets from various real-life applications. The description of the small and the big datasets used are provided in Table 3 and 4 respectively.  $n$ ,  $d$  and  $distr.$ , in both the tables refer to

Table 3. Small Data Sets Used

| Name   | Dataset Characteristics | Attribute Characteristics | $n$    | $d$ | #distr. | Change Pt.  | $k$ | $l$ |
|--|-------------------------|---------------------------|--------|-----|---------|---|-----|-----|
| Synthetic                                    | Multivariate            | Real                      | 20,000 | 20  | 10      | 2000 (periodic)   | 100 | 5   |
| Wholesale Customers (WC) [1]                 | Multivariate            | Integer                   | 440    | 6   | 5       | 197, 274, 294, 341  | 20  | 3   |
| Wireless Indoor Localization (WIL) [13] [70] | Multivariate            | Real                      | 2,000  | 7   | 4       | 500 (periodic)  | 30  | 3   |
| Wine [27]                                    | Multivariate            | Integer, Real             | 178    | 13  | 3       | 60, 131   | 20  | 6   |
| Iris [27]                                    | Multivariate            | Real                      | 150    | 4   | 3       | 51, 101   | 30  | 1   |
| Avila [25]                                   | Multivariate            | Real                      | 10430  | 10  | 12      | 4287, 4292, 4395, 4747, 5842, 7803, 8249, 8768, 9599, 9643, 10165 | 30  | 5   |

Table 4. Big Data Sets Used

| Name   | Dataset Characteristics             | Attribute Characteristics | $n$     | $d$  | #distr. | Change Pt.  | $k$ | $l$ |
|--|-------------------------------------|---------------------------|---------|------|---------|---|-----|-----|
| Isolet [27]  | Multivariate                        | Real                      | 7797    | 617  | 150     | 52 (periodic)   | 40  | 50  |
| S&P 500 stock [65]   | Multivariate                        | Real                      | 619040  | 5    | 500     | 1259 (periodic)   | 20  | 2   |
| Daily and Sports Activities (DSA) [6] [10] [5]                             | Multivariate, Time-Series           | Real                      | 9120    | 5625 | 152     | 60 (periodic)   | 15  | 20  |
| Activity Recognition from Single Chest-Mounted Accelerometer (ARSCMA) [20] | Univariate, Sequential, Time-Series | Real                      | 1926881 | 3    | 141     | 33678, 34606, 38489, 65349, 69596, 72787, 75836, 78753, ... | 30  | 2   |

the number of instances, dimensionality and number of distributions, respectively. For datasets with a single value for *ChangePoint*, there is a periodic change in data distribution after that many instances of data. For example in the Synthetic Dataset in Table 3, distribution changes after every 2000 samples. For datasets where *ChangePoint* is specified as a set of values, changes in data distribution occur at those particular instances. For example in the Wine Dataset in Table 3, data from a new distribution arrives at instance number 60 and 131. In the Activity Recognition from Single Chest-Mounted Accelerometer (ARSCMA) dataset in Table 4, there are 141 distributions with 140 non-periodic change points. As it was not possible to enlist all of them, we showed the first 8 change points.  $k$  and  $l$  represents the block size and the target dimension, respectively, which were declared as user-defined input parameters to the algorithms. Therefore, we randomly specify some value for  $k$  and  $l$  for each dataset. The value of  $k$  and  $l$  chosen for a certain dataset will stay consistent across each algorithm during experimentation.

**6.1.1 Synthetic Dataset.** The synthetic dataset is generated according to standard Gaussian distributions. An advantage of using simulated datasets is that we can control the time when a change in the distribution occurs. Therefore, we can see how our algorithms behave in those particular change points. The synthetic data generation procedure is illustrated in Algorithm 6.  $d$  refers to the input dimension,  $c$  is the number of distributions in the dataset and  $obs$  refers to the array of the number of samples to be drawn from each distribution. So  $obs_i$  refers to the number of samples to be drawn from the  $i$ -th distribution.  $\mu_i$  and  $\Sigma_i$  refer to the mean and covariance matrix that defines the  $i$ -th distribution. The dataset built is  $X$ . Our synthetic dataset has 20000 samples, 20 dimensions and 10 distributions, with 2000 samples in each distribution.

**Algorithm 6:** Synthetic data generation

---

**Input:**  $d, c, obs$   
**Output:**  $X$

```

1  $\mu \leftarrow$  all zeros matrix of size  $d$ 
2  $\Sigma \leftarrow$  all zeros array of size  $d \times d$ 
3 for  $i$  in  $c$  do
4   Generate  $\mu_i$  and  $\Sigma_i$  randomly
5    $Y_i \leftarrow$  Generate  $obs_i$  number of samples randomly from a Gaussian distribution defined by  $(\mu_i, \Sigma_i)$ 
6    $X.append(Y_i)$ 
7 end

```

---

**6.1.2 Real Datasets.** The description of each of the real datasets used are given in Table 3 and 4. They are standard benchmark datasets collected from UCI Machine Learning Repository and Kaggle. They have been widely used by students, educators, and researchers all over the world as a primary source of benchmark machine learning datasets (Wholesale Customers (WC) [11] [57], Wireless Indoor Localization (WIL) [70], Wine [3] [2], Iris [39] [31] [23], Avila [25] [24], Isolet [38] [28] [29], Daily and Sports Activities (DSA) [6] [10] [5], and Activity Recognition from Single Chest-Mounted Accelerometer (ARSCMA) [19] [18] [20]). We chose datasets from multifarious applications to show that our algorithms operate successfully in multiple domains. We selected a diverse set of data with vastly different dimensionality, number of distributions and number of instances. In our datasets, the number of instances range from 150 to 1926881, distributions range from 3 to 500 and dimensions range from 3 to 5625. The detailed description of each of the real datasets used are as follows:

- **Wholesale Customers (WC):** The dataset [1] refers to clients of a wholesale distributor. It includes the annual spending in monetary units (m.u.) on diverse product categories with different attributes in three different regions. The distribution switches among these three regions.
- **Wireless Indoor Localization (WIL):** The data [13] [70] is collected in indoor space by observing signal strengths of seven WiFi signals visible on a smartphone for indoor localization. Each attribute is wifi signal strength observed on smartphone. The decision variable is one of the four rooms and the distribution switches among one room to another after every 500 instances.
- **Wine:** This dataset [27] contains the quantities of 13 constituents found in three different types of wines grown in three cultivars. The distribution switches from the first, to the second and then to the third cultivar.
- **Iris:** The dataset [27] contains 3 classes of 50 instances each, where each class refers to a type of iris plant. The classes are Iris Setosa, Iris Versicolour, and Iris Virginica. The distribution switches from each class to another after every 50 instances.
- **Avila:** This dataset [25] has been extracted from 800 images of the 'Avila Bible', an XII century giant Latin copy of the Bible. Each class corresponds to a copyist. Data have been normalized by using the Z-normalization method and divided into two data sets: a training set containing 10430 samples, and a test set containing the 10437 samples. We use only the training set and the change points are deduced by cumulatively adding the number of instances belonging to each class distribution. The distribution switches from one copyist to another after every change point.
- **Isolet:** This dataset [27] was generated as follows: 150 subjects spoke the name of each letter of the alphabet twice. Hence, we have 52 training examples from each speaker. Hence, there should have been 7800 instances

in this dataset. However, 3 instances are missing because they may have been dropped due to difficulties in recording. There is a change in speaker and therefore, a change in distribution after every 52 instances.

- **S&P 500 stock:** This dataset [65] provides the historical stock prices (last 5 years) for all companies currently found on the S&P 500 index. The features available are 1) Open - price of the stock at market open (this is NYSE data so all in USD), 2) High - highest price reached in the day, 3) Low - lowest price reached in the day, 4) Close - price of the stock at market close and 5) Volume - number of shares traded. The distribution switches from one company to another after every 1259 instances.
- **Daily and Sports Activities (DSA):** The dataset [6] [10] [5] comprises of motion sensor data of 19 daily and sports activities, each performed by 8 subjects in their own style for 5 minutes. Five Xsens MTx units are used on the torso, arms, and legs. Therefore, there are  $19 \times 8 = 152$  distributions in this dataset.
- **Activity Recognition from Single Chest-Mounted Accelerometer (ARSCMA):** The dataset [20] collects data from a wearable accelerometer mounted on the chest. The dataset is intended for Activity Recognition research purposes. Uncalibrated Accelerometer Data are collected from 15 participants, each performing 7 activities. The distribution switches among these 7 activities for each participant and also from one participant to another.

## 6.2 Experimental Results

In this section, we illustrate the block reconstruction error, individual reconstruction error and the bandwidth of our algorithms. We compare the performances of only Algorithm 5 (RPTO) and Algorithm 2 (BPCA) on error metric because the behavior of Algorithm 3 (RP) and Algorithm 4 (RPT) is indefinite. Algorithm 3 (RP) is indefinite because the **while** loop in line 8 may iterate an unknown number of times for each data. Additionally, in Algorithm 4 (RPTO), we are choosing the  $l$  largest eigenvalues from the  $l + d$  eigenvalues obtained from  $\sigma_{arr} \cup S_\sigma$  and their corresponding eigenvectors from  $U \cup S_U$ . So the number of eigenvalues chosen from each of  $\sigma_{arr}$  and  $S_\sigma$  and similarly, the corresponding number of eigenvectors retained from each of  $U$  and  $S_U$ , at each iteration is also indefinite. Their performances are dependent upon the erratic nature of the number of eigenvectors they replace in  $U$  at each timestep. Hence, they do not show any general trend in error with increasing block size or target dimension.

We compare the bandwidth performances of Algorithm 3 (RP), Algorithm 4 (RPT) Algorithm 5 (RPTO) and Algorithm 2 (BPCA) in order to demonstrate the bandwidth supremacy of Algorithm 5 (RPTO) over the other three.

**6.2.1 Variation of Sum of Block Reconstruction Error w.r.t. Block Size,  $k$  and Target Dimension,  $l$ .** This section compares the performances of Algorithm 5 (RPTO) and Algorithm 2 (BPCA) in terms of their Sum of Block Reconstruction Error. The Sum of Block Reconstruction Error is given by  $\sum_{t=k}^n (I - U_t U_t^T) B_t$ , where  $t$  is the current timestep and  $k$  is the block size.  $t$  starts from  $k$  because initially it takes  $k$  timesteps to form a block. Once a block has formed at timestep  $k$ , the block persists through the course of the algorithm by discarding the data with the oldest timestep and incorporating the data at the current timestep. The block reconstruction error is what we are looking to reduce in every algorithm. We are using this optimization criteria on a block of data to capture the current distribution of data. It is to be noted that all errors are measured in a logarithmic (base-10) scale.

We can compare the performance of the algorithms either in terms of the Sum of Block Reconstruction Error or the Average Block Reconstruction Error because they are directly proportional to one another.  $n$  is the number of data instances present in a dataset. This is also equivalent to the number of timesteps at which a data instance arrives. So  $n$  is 20,000 for the synthetic dataset we are using in this experiment because there are 20,000 data instances in this

Table 5. Variation of Sum of Block Reconstruction Error w.r.t Block Size,  $k$  and Target Dimension,  $l$  for 5 small datasets

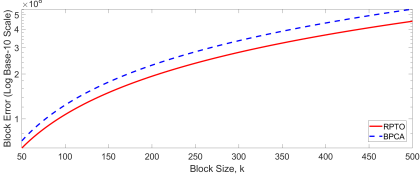
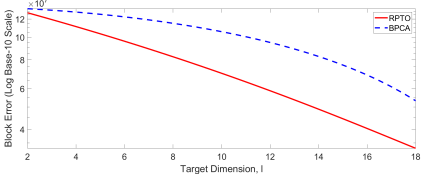
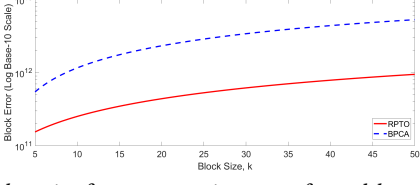
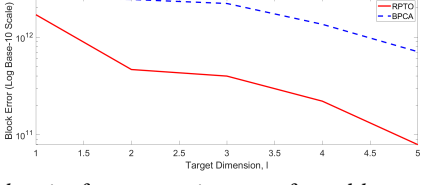
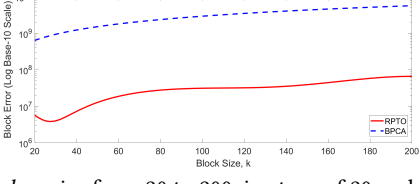
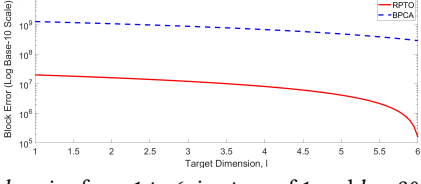
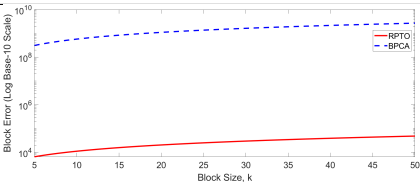
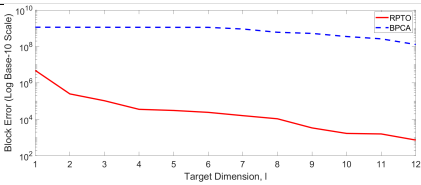
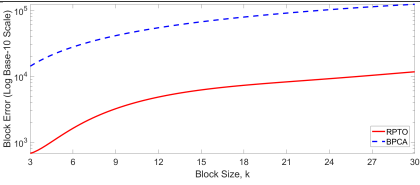
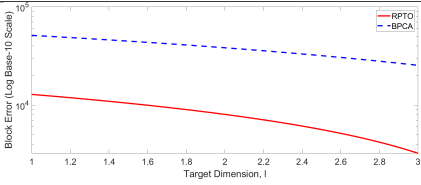
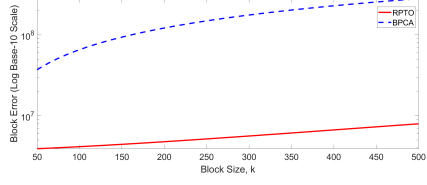
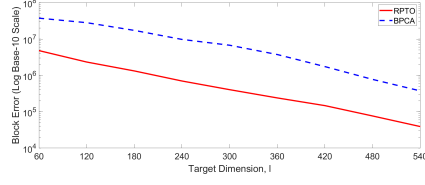
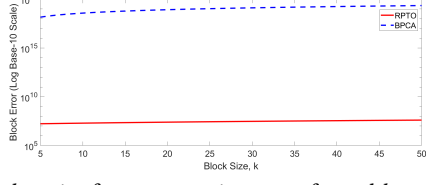
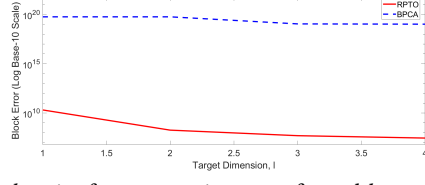
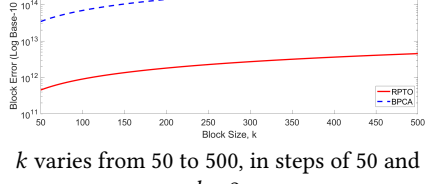
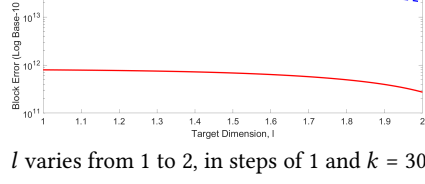
| Dataset                                      | Variation of Sum of Block Reconstruction Error w.r.t  |   |
|--|---|---|
|  | Block Size, $k$   | Target Dimension, $l$   |
| Synthetic                                    |  <p><math>k</math> varies from 50 to 500, in steps of 50 and <math>l = 5</math></p>  |  <p><math>l</math> varies from 2 to 18, in steps of 2 and <math>k = 100</math></p>  |
| Wholesale Customers (WC) [1]                 |  <p><math>k</math> varies from 5 to 50, in steps of 5 and <math>l = 3</math></p>     |  <p><math>l</math> varies from 1 to 5, in steps of 1 and <math>k = 20</math></p>    |
| Wireless Indoor Localization (WIL) [13] [70] |  <p><math>k</math> varies from 20 to 200, in steps of 20 and <math>l = 3</math></p> |  <p><math>l</math> varies from 1 to 6, in steps of 1 and <math>k = 30</math></p>   |
| Wine [27]                                    |  <p><math>k</math> varies from 5 to 50, in steps of 5 and <math>l = 6</math></p>   |  <p><math>l</math> varies from 1 to 12, in steps of 1 and <math>k = 20</math></p> |
| Iris [27]                                    |  <p><math>k</math> varies from 3 to 30, in steps of 3 and <math>l = 2</math></p>   |  <p><math>l</math> varies from 1 to 3, in steps of 1 and <math>k = 10</math></p>  |

Table 6. Variation of Sum of Block Reconstruction Error w.r.t Block Size,  $k$  and Target Dimension,  $l$  for 3 big datasets

| Dataset  | Variation of Sum of Block Reconstruction Error w.r.t   |  |
|--|--|--|
|  | Block Size, $k$  | Target Dimension, $l$  |
| Isolet [27]  |  <p><math>k</math> varies from 50 to 500, in steps of 50 and <math>l = 100</math></p> |  <p><math>l</math> varies from 60 to 540, in steps of 60 and <math>k = 50</math></p> |
| S&P 500 stock  |  <p><math>k</math> varies from 5 to 50, in steps of 5 and <math>l = 4</math></p>      |  <p><math>l</math> varies from 1 to 4, in steps of 1 and <math>k = 20</math></p>     |
| ARSCMA (first participant only out of 15 participants) [20] [70] |  <p><math>k</math> varies from 50 to 500, in steps of 50 and <math>l = 2</math></p>  |  <p><math>l</math> varies from 1 to 2, in steps of 1 and <math>k = 30</math></p>    |

dataset. When we conducted this experiment to compare the performance of several algorithms, we used the same synthetic dataset for each algorithm, which means that  $n$  is the same for each algorithm. Similarly, when we measured the reconstruction error of each algorithm for a particular value of block size, for example  $k = 100$ , then  $k$  was also kept at 100 for each algorithm.  $l$  was also kept the same for each algorithm. Average Block Reconstruction Error = Sum of Block Reconstruction Error / Number of blocks, where Number of blocks =  $(n - k + 1)$ . So with  $n$  constant for a particular dataset and  $k$  and  $l$  kept constant across each algorithm for a particular value of block size, Average Block Reconstruction Error  $\propto$  Sum of Block Reconstruction Error.

For both the small (Table 5) and big (Table 6) datasets and for both the sets of graphs (Block Size,  $k$ , and Target Dimension  $l$ ), we find that Algorithm 5 (RPTO) gives a lower Sum of Block Reconstruction Error than Algorithm 2 (BPCA). When the block size  $k$  increases, the Sum of Block Reconstruction Error increases as well. For block size  $k$  and number of input instances  $n$ , we are taking the sum over the reconstruction error of  $n - k + 1$  blocks through the course of the algorithm. As per our experiments  $k \ll n$ , therefore,  $n - k + 1 \approx n$ . So, if we are computing the sum of reconstruction error for  $n$  blocks, the larger the block size  $k$ , the greater the error will be. We also find that the Sum of Block Reconstruction Error decreases with increasing target dimension  $l$ . As we increase the target dimension  $l$ ,

the eigenvector matrix  $U$  captures a greater proportion of the underlying variance of the input data, and therefore the reconstruction error decreases.

Table 7. Variation of Sum of Individual Reconstruction Error w.r.t Block Size,  $k$  and Target Dimension,  $l$  for 5 small datasets

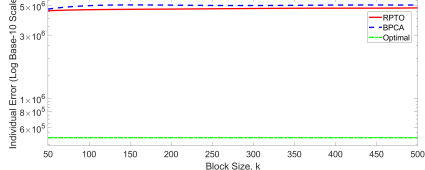
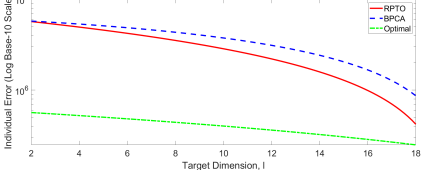
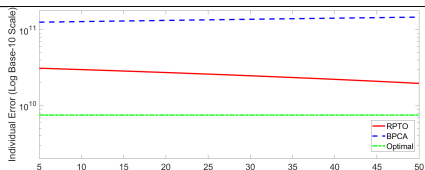
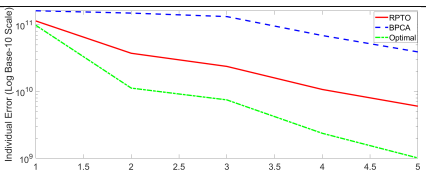
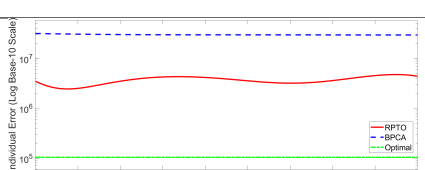
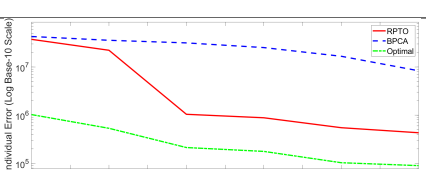
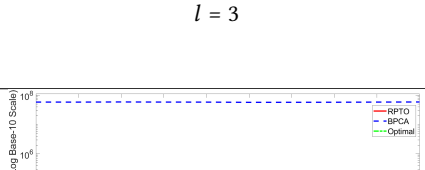
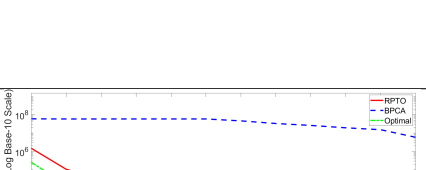
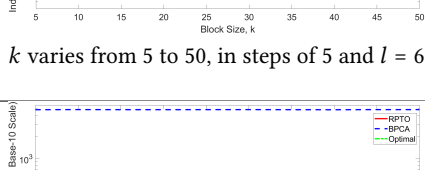
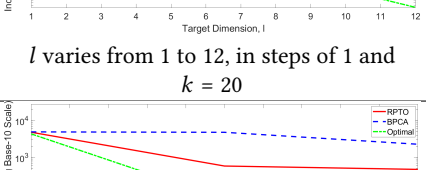
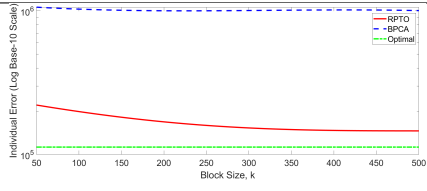
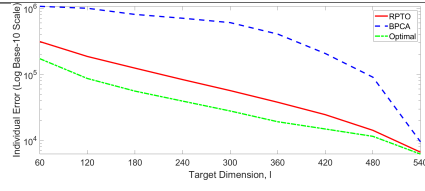
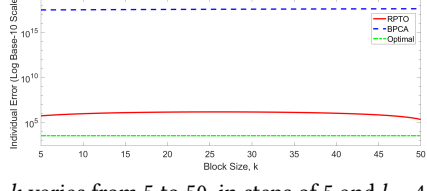
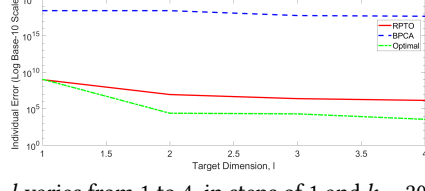
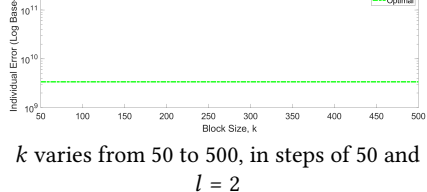
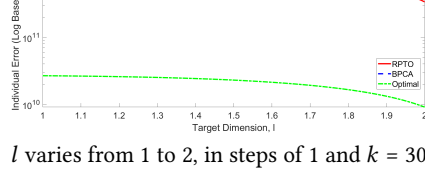
| Dataset                                      | Variation of Sum of Individual Reconstruction Error w.r.t   |   |
|--|---|---|
|  | Block Size, $k$   | Target Dimension, $l$   |
| Synthetic                                    |  <p><math>k</math> varies from 50 to 500, in steps of 50 and <math>l = 5</math></p>  |  <p><math>l</math> varies from 2 to 18, in steps of 2 and <math>k = 100</math></p>  |
| Wholesale Customers (WC) [1]                 |  <p><math>k</math> varies from 5 to 50, in steps of 5 and <math>l = 3</math></p>     |  <p><math>l</math> varies from 1 to 5, in steps of 1 and <math>k = 20</math></p>    |
| Wireless Indoor Localization (WIL) [13] [70] |  <p><math>k</math> varies from 20 to 200, in steps of 20 and <math>l = 3</math></p> |  <p><math>l</math> varies from 1 to 6, in steps of 1 and <math>k = 30</math></p>   |
| Wine [27]                                    |  <p><math>k</math> varies from 5 to 50, in steps of 5 and <math>l = 6</math></p>   |  <p><math>l</math> varies from 1 to 12, in steps of 1 and <math>k = 20</math></p> |
| Iris [27]                                    |  <p><math>k</math> varies from 3 to 30, in steps of 3 and <math>l = 2</math></p>   |  <p><math>l</math> varies from 1 to 3, in steps of 1 and <math>k = 10</math></p>  |

Table 8. Variation of Sum of Individual Reconstruction Error w.r.t Block Size,  $k$  and Target Dimension,  $l$  for 3 big datasets

| Dataset  | Variation of Sum of Individual Reconstruction Error w.r.t  |  |
|--|--|--|
|  | Block Size, $k$  | Target Dimension, $l$  |
| Isolet [27]  |  <p><math>k</math> varies from 50 to 500, in steps of 50 and <math>l = 100</math></p> |  <p><math>l</math> varies from 60 to 540, in steps of 60 and <math>k = 50</math></p> |
| S&P 500 stock  |  <p><math>k</math> varies from 5 to 50, in steps of 5 and <math>l = 4</math></p>      |  <p><math>l</math> varies from 1 to 4, in steps of 1 and <math>k = 20</math></p>     |
| ARSCMA (first participant only out of 15 participants) [20] [70] |  <p><math>k</math> varies from 50 to 500, in steps of 50 and <math>l = 2</math></p>  |  <p><math>l</math> varies from 1 to 2, in steps of 1 and <math>k = 30</math></p>    |

**6.2.2 Variation of Sum of Individual Reconstruction Error w.r.t. Block Size,  $k$  and Target Dimension,  $l$ .** We have shown the Sum of Block Reconstruction Error in the previous section because that is our optimization criteria. However, successive blocks will contain some common data, therefore, evaluating performance based on the sum of block reconstruction error may be redundant because then we are computing the error for each instance  $k$  times. Therefore, we felt the need to include graphs for Variation of Sum of Individual Reconstruction Error besides the graphs for Variation of Sum of Block Reconstruction Error w.r.t Block Size  $k$  and Target Dimension  $l$ . The Sum of Individual Reconstruction Error is given by  $\sum_{t=1}^n (I - U_l U_l^T) x_t$ , where  $x_t$  is the data instance arriving at timestep  $t$ . In this way, we are computing the reconstruction error for every input instance once. Moreover, the Sum of Individual Reconstruction Error is equal to the reconstruction error of the whole dataset. Offline PCA, using Singular Value Decomposition (SVD) optimizes the reconstruction error of the whole dataset, which is given by  $(I - UU^T)X$ , where  $X$  is the data matrix containing all the data instances. Therefore, in the graphs of Sum of Individual Reconstruction Error, we show how the reconstruction error of Algorithm 5 (RPTO) and Algorithm 2 (BPCA) compares to the minimum possible reconstruction error, given by the standard offline PCA which is computed via SVD. The green line which appears at the bottom of each graph in Table 7 and 8, and is labeled as optimal, refers to the reconstruction error given by the standard Offline PCA. It is to be noted

that all errors are measured in a logarithmic (base-10) scale. Offline PCA can compute the output data, given the input data and the target dimension. For the graphs of Variation of Sum of Individual Reconstruction Error w.r.t Block Size  $k$ , we measured the reconstruction error of Offline PCA, given by  $(I - UU^T)X$  with the same target dimension that was used for the corresponding graphs of Algorithm 5 (RPTO) and Algorithm 2 (BPCA). For the graphs of Variation of Sum of Individual Reconstruction Error w.r.t Target Dimension  $l$ , we computed the reconstruction error for each value of  $l$ .

We can compare the performance of the algorithms either in terms of the Sum of Individual Reconstruction Error or the Average Individual Reconstruction Error because they are directly proportional to one another.  $n$  is the number of data instances present in a dataset. This is also equivalent to the number of timesteps at which a data instance arrives. So  $n$  is 20,000 for the synthetic dataset we are using in this experiment because there are 20,000 data instances in this dataset. When we conducted this experiment to compare the performance of several algorithms, we used the same synthetic dataset for each algorithm, which means that  $n$  is the same for each algorithm. Similarly, when we measured the reconstruction error of each algorithm for a particular value of block size, for example  $k = 100$ , then  $k$  was also kept at 100 for each algorithm.  $l$  was also kept the same for each algorithm. Average Individual Reconstruction Error = Sum of Individual Reconstruction Error /  $n$ . So with  $n$  constant for a particular dataset and  $k$  and  $l$  kept constant across each algorithm for a particular value of block size, Average Individual Reconstruction Error  $\propto$  Sum of Individual Reconstruction Error.

For both the small (Table 7) and big (Table 8) datasets and for both the sets of graphs vs Block Size,  $k$ , and Target Dimension  $l$ , we find that Algorithm 5 (RPTO) gives a lower Sum of Individual Reconstruction Error than Algorithm 2 (BPCA). Offline PCA gives the optimal solution. An interesting point to note here is that the lines in the graphs of Variation of Sum of Individual Reconstruction Error w.r.t. Block Size,  $k$  are quite straight, which implies that the reconstruction error of the data stays relatively constant with changing block size,  $k$ . This implies that Algorithm 5 (RPTO), 2 (BPCA) and the optimal algorithm are all stable to the variation of the block size,  $k$ . We also find that the Sum of Individual Reconstruction Error decreases with increasing target dimension  $l$ . As we increase the target dimension  $l$ , the eigenvector matrix  $U$  captures a greater proportion of the underlying variance of the input data, and therefore the reconstruction error decreases.

**6.2.3 Bandwidth measure of our Algorithms.** Here, we compare the bandwidth performances of Algorithm 3 (RP), 4 (RPT), 5 (RPTO) and 2 (BPCA). In a Real Time application, the runtime efficiency for processing whole data sets is a trivial result. However, it is vital to show whether the algorithms can process input data items in the speed at which they are generated. For the synthetic dataset of 20000 instances and 20 dimensions, we have shown the *timeperiod*, *frequency* and the *bandwidth* required by our algorithms in Table 9. These experiments were conducted on a local PC with a core i3 1.80 GHz processor and a 4 GB RAM. As per [84], approximately 8344 tweets, 892 instagram photos, 3586 skype calls and 71993 google searches are generated every second. Therefore, when our algorithms are housed in clustered systems of servers, these algorithms will be fast enough to analyze such real-time big data.

### 6.3 Experimental Verification

In this section, we verify our claim that the algorithms implement PCA in real-time through spectral analysis and use Bhattacharyya Coefficient to illustrate how well our algorithms capture the changing distributions of data in real-time. The eighth and the ninth column of Table 3 and 4 show the input values of block size,  $k$  and target dimension,  $l$ , for each of the four algorithms.

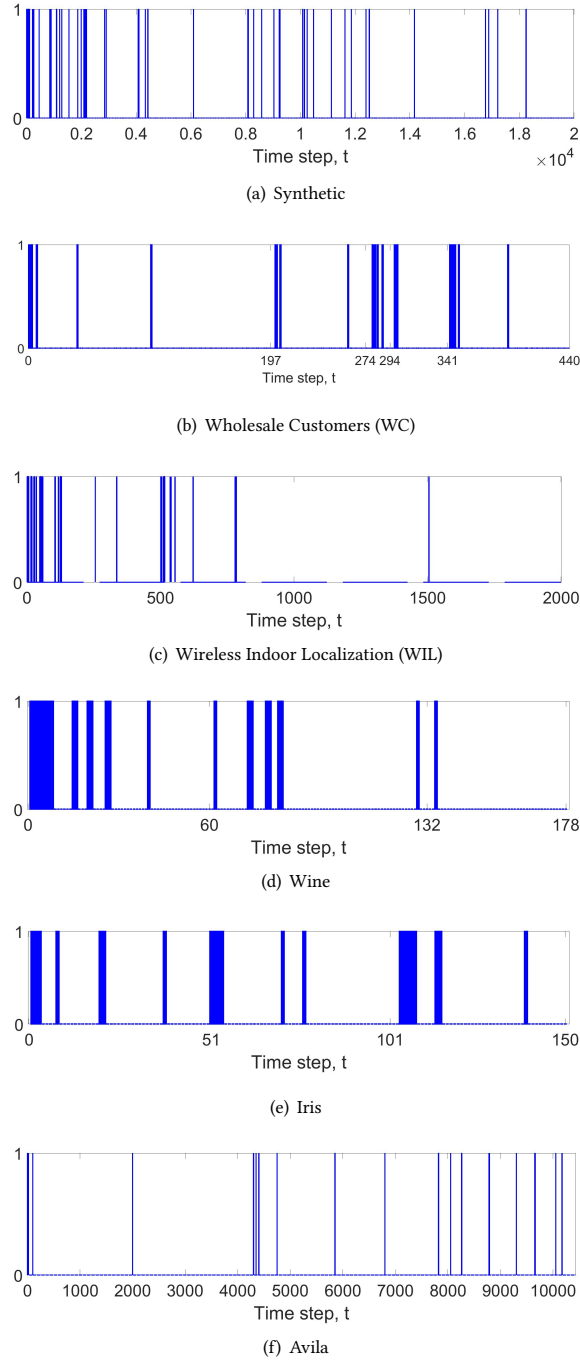
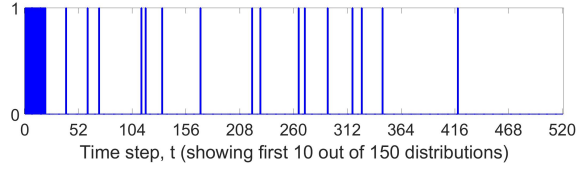


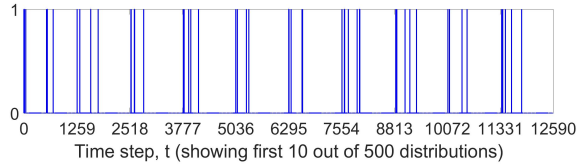
Fig. 1. Spectral Diagrams given by Algorithm 5 (RPTO) for small datasets, namely: (a) Synthetic (b) Wholesale Customers (WC) (c) Wireless Indoor Localization (WIL) (d) Wine (e) Iris and (f) Avila. The actual change points for each dataset are given in Table 3

Table 9. Time Period, Frequency and Bandwidth of our algorithms on Synthetic dataset

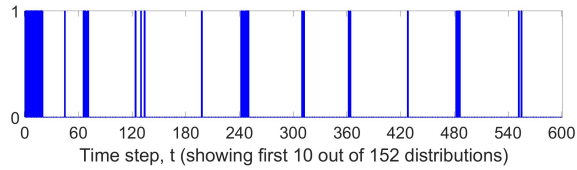
| Algorithm | Time Period, $p$ ,<br>in $ms$ | Frequency, $f = \frac{1}{p}$ ,<br>in $Hz$ | Bandwidth,<br>$b = 2 * f$ , in $Hz$ |
|-----------|-------------------------------|---|-------------------------------------|
| RP        | 3.22                          | 311                                       | 621                                 |
| RPT       | 2.34                          | 427                                       | 855                                 |
| RPTO      | 2.19                          | 457                                       | 913                                 |
| BPCA      | 2.28                          | 439                                       | 877                                 |



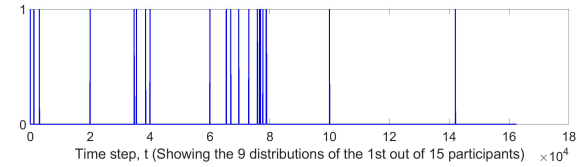
(a) Isolet



(b) S&amp;P 500 stock



(c) Daily and Sports Activities (DSA)



(d) Activity Recognition from Single Chest-Mounted Accelerometer (ARSCMA)

Fig. 2. Spectral Diagrams given by Algorithm 5 (RPTO) for big datasets, namely: (a) Isolet (b) S&P 500 stock (c) Daily and Sports Activities (DSA) and (d) Activity Recognition from Single Chest-Mounted Accelerometer (ARSCMA) (first participant only out of 15 participants). The expected change points for each dataset are given in Table 4

**6.3.1 Verification through Spectral Diagram.** In this section, we see how spectral diagrams help us to verify our claim for Real Time PCA. Whenever an eigenvector is added to  $U$ , we increment a counter that keeps track of the number of

eigenvectors added at each time step. The intuition is that whenever there is a change in the underlying data distribution, a large proportion of eigenvectors will be replaced at time steps immediately following the change in distribution. Hence, the occurrence of spikes should be a good indicator of when a change in distribution has occurred.

As evident from the spectral diagrams in Figure 1 and 2, the frequency of eigenvectors added to  $U$  is higher when distribution changes from one class to another. This suggests that our algorithms are updating  $U$  to account for the **latest** distribution. A cluster of spikes are generated **immediately** following the change in the input data distribution. This validates the claim for our proposed algorithms as **real-time**. We also find that this adaptation takes place within a relatively short time after the change in distribution has occurred, thus our algorithms have a low latency.

However, there are a few exceptions to the nature of these diagrams. Occasionally, we find that eigenvectors are added at timesteps when the distribution does not change. This may happen because some samples have been drawn from the tail ends of that distribution. Similarly, in some instances, we also find that no spikes are visible in the figures even after the distribution has changed. This might occur because the new distribution is very similar to the previous distribution. The actual time the algorithms need to recognize a change in input distributions is known as latency. Latency can be calculated from the difference between the timestep at which a spike occurs and the timestep at which the distribution actually changes.

For some datasets, like the Isolet dataset, S&P 500 stock data, and Daily and Sports Activities (DSA) dataset, the number of distributions is large. Hence, it was not possible to accommodate all those distributions in the spectral diagram. Therefore, we have shown the spectral diagram for the first 10 distributions. For the Activity Recognition from Single Chest-Mounted Accelerometer (ARSCMA) dataset, there were 15 participants, each of whom performed a series of 7 activities one after another. Due to space limitation, we have shown the spectral diagram for the first participant only.

**6.3.2 Verification through Comparison between Corresponding Distributions of Input and Output Data.** In this section, we use the statistical measure, Bhattacharyya Coefficient, to show similarity between the input and output data distributions. The intuition is, if the algorithms are updating  $U$  in accordance with the current data distribution, then the distribution of the projected output, generated from that  $U$ , will be statistically similar to the corresponding distribution of the input.

Bhattacharyya Coefficient (BC) [14] measures the similarity between two distributions of **equal** number of dimensions. But input data has  $d$  dimensions and output has  $l$  dimensions. So, we mean-standardize our input data for each distribution and then compute the PCA of the whole input to project it into a  $l$  dimensional subspace. With both the input and output having dimension  $l$ , we can now compute the  $BC$  between them.

We use  $BC$  to compute the statistical similarity between the corresponding distributions of the input and the output for each dataset. Figure 3(a) and 3(b) show the bar graphs of the Bhattacharyya Coefficient  $BC$  values, given by each algorithm, for each dataset. The height of the bar represents the mean and the height of the vertical line in the bar represent the standard deviation of the  $BC$  values across all the distributions in the dataset. We know that a  $BC$  value close to 1 indicates similarity between two probability distributions. As the values in the bar graphs show, all the mean  $BC$  values for all the algorithms are close to 1, which indicates that the distributions of the output, given by our algorithms, are statistically similar to the corresponding distributions of the input. This means that despite the changing distributions of the input, our algorithms are able to successfully update  $U$  in accordance with the current distribution. This is why the output distributions are statistically similar to the input. So our algorithms can capture the essence of **changing distributions** in **streaming** data over time and simultaneously **retain the major components**

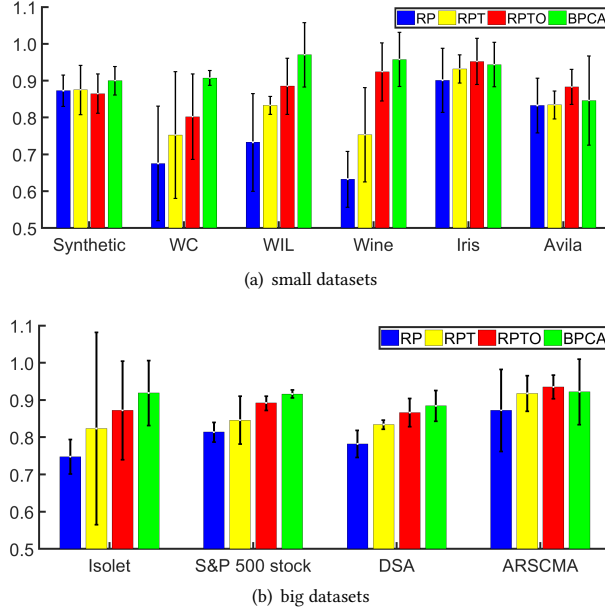


Fig. 3. Bar graph showing mean Bhattacharyya Coefficient (BC) values with standard deviation bars for (a) small datasets (b) big datasets

of previous distributions seen so far, thereby validating **real-time** PCA. Moreover, the values for the standard deviation of  $BC$  values are quite low for all the datasets. This signifies that the algorithms give consistently high  $BC$  values across all distributions. The  $BC$  values have a low variability across all distributions.

For some datasets, like Iris, Activity Recognition from Single Chest-Mounted Accelerometer (ARSCMA), and Avila, Algorithm 5 (RPTO) outperforms the rest by giving the highest  $BC$  value. For others, like Wholesale Customers (WC), Wireless Indoor Localization (WIL), Wine, Isolet, S&P 500 stock, and Daily and Sports Activities (DSA), Algorithm 5 (RPTO) gives a  $BC$  value higher than that of Algorithm 3 (RP) and Algorithm 4 (RPT) but slightly lower than that given by Algorithm 2 (BPCA). However, as illustrated in Table 5, 6, 7, and 8 Algorithm 5 (RPTO) gives a lower reconstruction error than Algorithm 2 (BPCA), with respect to both varying block size,  $k$  and varying target dimension,  $l$ . Therefore, even for those datasets, where Algorithm 5 (RPTO) gives a slightly lower  $BC$  value than Algorithm 2 (BPCA), it is preferable to use Algorithm 5 (RPTO) because it gives a lower reconstruction error than Algorithm 2 (BPCA). Moreover, Algorithm 5 (RPTO) gives the highest  $BC$  value when compared to Algorithm 3 (RP) and Algorithm 4 (RPTO) for all the datasets, except for the Synthetic Dataset, where it is marginally lower. Besides, as illustrated in Table 9, Algorithm 5 has a lower time period, therefore a higher bandwidth, than both Algorithm 3 (RP) and Algorithm 4 (RPT) for the Synthetic Dataset, with respect to both varying block size,  $k$  and varying target dimension,  $l$ . Therefore, taking both the  $BC$  values and the bandwidth into consideration, it is preferable to use Algorithm 5 (RPTO) ahead of Algorithm 3 (RP) and Algorithm 4 (RPT). Hence, Algorithm 5 (RPTO) is the **best** among all the algorithms that we proposed, considering all the performance parameters and the trade-offs involved.

Table 10. Mean Difference in Bhattacharyya Coefficient ( $BC$ ) values between the change in distribution of input and output data of the small datasets

| Dataset                            | RP       | RPT      | RPTO     | BPCA     |
|------------------------------------|----------|----------|----------|----------|
| Synthetic                          | 0.042308 | 0.072043 | 0.092350 | 0.110122 |
| Wholesale Customers (WC)           | 0.060192 | 0.043503 | 0.054558 | 0.168512 |
| Wireless Indoor Localization (WIL) | 0.752977 | 0.265083 | 0.052322 | 0.116313 |
| Wine                               | 0.234278 | 0.167334 | 0.135342 | 0.118274 |
| Iris                               | 0.625384 | 0.382571 | 0.152748 | 0.126386 |
| Avila                              | 0.238925 | 0.116958 | 0.084836 | 0.164894 |

Table 11. Mean Difference in Bhattacharyya Coefficient ( $BC$ ) values between the change in distribution of input and output data of the big datasets

| Dataset   | RP       | RPT      | RPTO     | BPCA     |
|---|----------|----------|----------|----------|
| Isolet  | 0.078323 | 0.093627 | 0.047355 | 0.082540 |
| S&P 500 stock   | 0.082631 | 0.042754 | 0.027954 | 0.091736 |
| Daily and Sports Activities (DSA)                                     | 0.077637 | 0.063381 | 0.041081 | 0.072618 |
| Activity Recognition from Single Chest-Mounted Accelerometer (ARSCMA) | 0.064828 | 0.104723 | 0.084874 | 0.097502 |

### 6.3.3 Verification through Comparison between the **Change** in Corresponding Distributions of Input and Output Data.

In the previous section, we showed that the distributions of the output data are statistically similar to those of the input data. In this section, we reinstate the validity of our algorithms by showing that the changes occurring across the distributions of the input are also statistically similar to the changes observed across the distributions of the output. The intuition is, if the input distributions are changing, then according to our model, the algorithms should update  $U$  in a way such that the changes occurring across the distributions of the input are reflected in the changes occurring across the distributions of the output.

We measure the statistical change between two successive distributions of the input data using Bhattacharyya Coefficient. Then we similarly compute the statistical change between the corresponding two successive distributions of the output data. We take the absolute value of the difference between the two  $BC$  scores. This value reflects how close the change occurring across the distributions of the output is to the change occurring across the corresponding distributions of the input. The lower the value, the closer the distribution change of the output is to that of the input. We repeat this procedure for all pairs of successive distributions in the data.

Table 10 and 11 show the results for all the datasets used. Algorithm 3 (RP) performs better than 2 (BPCA) for all datasets except Daily and Sports Activities (DSA), Avila, Iris, Wine, and Wireless Indoor Localization (WIL). Algorithm 4 (RPT) performs better than 2 (BPCA) for all datasets except Wireless Indoor Localization (WIL), Wine, Iris, Isolet, and Activity Recognition from Single Chest-Mounted Accelerometer (ARSCMA). Algorithm 5 (RPTO) performs better than 2 for all datasets except Wine and Iris. Although the performance of Algorithm 3 (RP), 4 (RPT) and 2 (BPCA) is inconsistent across some datasets, Algorithm 5 (RPTO) performs reasonably and consistently well for all datasets, giving values very close to zero. This indicates that the changes observed across the distributions of the output are approximately similar to the changes occurring across the distributions of the input.

In Section 6.3.2, the BC values mostly tend to increase from Algorithm 3 (RP) to Algorithm 4 (RPT) to Algorithm 5 (RPTO) to Algorithm 2 (BPCA) for both the small and the big datasets. However, for each of the algorithms, the BC values show a higher standard deviation for the small datasets than the big datasets. Moreover, the standard deviation of BC values between the different algorithms for each of the datasets is higher for the small datasets than the big datasets. In Section 6.3.3, the Mean Difference in Bhattacharyya Coefficient (BC) values between the change in the distribution of input and output data of the small datasets is larger than of the big datasets.

Algorithm 2 (BPCA) gives a higher BC value than Algorithm 5 (RPTO) in Section 6.3.2, indicating that its output distributions are reasonably similar to the corresponding input distributions. Therefore, Algorithm 2 (BPCA) should also be able to better capture the change across the input distributions than Algorithm 5 (RPTO), giving values closer to zero than Algorithm 5 (RPTO). However, it is interesting to note that Algorithm 5 (RPTO) outperforms Algorithm 2 (BPCA) by giving values closer to zero. The input data belongs to the same high-dimensional space throughout the whole time. However, the low-dimensional subspace of the output is continuously evolving due to the change in the eigenvector matrix  $U$ , every time an eigenvector is replaced. Since Algorithm 2 (BPCA) retains the eigenvectors computed from the reconstruction error of the current block  $B_t$  only, the low dimensional output reasonably approximates the input in terms of distributional similarity. Moreover, the output in this scenario corresponds to the current input distribution only. On the other hand, Algorithm 5 (RPTO) updates at most one eigenvector in  $U$  at every timestep. Hence, the output generated from this  $U$  retains some properties of previously encountered major distributions, and hence the change across the output data is more gradual. This explains why Algorithm 5 (RPTO) gives values closer to zero than Algorithm 2 (BPCA).

In the graphs of Variation of Sum of Block Reconstruction Error (Table 5 and 6) and the Variation of Sum of Individual Reconstruction Error (Table 7 and 8) w.r.t Block Size  $k$  and Target Dimension  $l$ , for 5 small and 3 big datasets, in Section 6.2.1 and 6.2.2, respectively, we find that Algorithm 5 (RPTO) gives a lower reconstruction error than Algorithm 2 (BPCA). We have stated reasons why Algorithm 1 (ROP) fails to perform Real Time PCA. We have also explained that Algorithm 5 (RPTO) is more noise resilient compared to Algorithm 3 (RP). Moreover, Algorithm 5 (RPTO) is more time-efficient than both Algorithm 3 (RP) and 4 (RPT), as shown in Table 2 (Time and Space Complexity) in Section 5.4 *Time and Space Complexity*. Algorithm 5 (RPTO) also gives the highest bandwidth compared to the other proposed algorithms as shown in Table 9 of Section 6.2.3 *Bandwidth measure of our Algorithms*. The spectral diagrams (Figure 1 and 2) also show that Algorithm 5 (RPTO) performs reasonably well for all the small and big, synthetic and real datasets. Besides, Algorithm 5 (RPTO) almost always has a higher mean Bhattacharyya Coefficient value than Algorithm 3 (RP) and 4 (RPT) for all the small and big datasets as shown in Figure 3 *Bar graph showing mean Bhattacharyya Coefficient (BC) values with standard deviation bars for (a) small datasets (b) big dataset* of Section 6.3.2 *Verification through Comparison between Corresponding Distributions of Input and Output Data*. Moreover, in Table 10 and 11 of Section 6.3.3 *Verification through Comparison between the **Change** in Corresponding Distributions of Input and Output Data*, we have shown that Algorithm 5 (RPTO) gives values closer to 0 than Algorithm 3 (RP), 4 (RPT) and 2 (BPCA). Therefore, taking all the parameters like Sum of Block Reconstruction Error, Sum of Individual Reconstruction Error, Time-Efficiency, Bandwidth, Spectral Diagram, mean Bhattacharyya Coefficient, and mean difference in Bhattacharyya Coefficient into consideration, we find that Algorithm 5 (RPTO) performs reasonably well across all the parameters compared to the other algorithms. Therefore, Algorithm 5 (RPTO) is the best algorithm that we propose to solve Real Time PCA.

## 7 CONCLUSION AND FUTURE WORK

This paper focuses on computing PCA on the fly in real-time and preserving the global nature of the data distribution in reduced dimension. To enable real time computation, we have proposed a novel approach to compute the eigenspace based on a sliding window model. We modified the existing architecture of online PCA to make it suitable for real-time in Algorithm 1 (ROP) and designed Algorithm 2 (BPCA), that is an overly simplistic approach to this problem. Next, we showed that these algorithms do not serve our intended purpose, which paved the way for Algorithm 3 (RP), 4 (RPT) and 5 (RPTO). We graphically show that Algorithm 5 (RPTO) surpasses other algorithms in terms of error measure and bandwidth performance. We use Spectral Diagram and Bhattacharyya Coefficient to establish that our algorithms **successfully** reduce data dimensionality in real-time in accordance with the **current** distribution while preserving the **major** components of the **previous** distributions.

This research opens up a number of interesting problems for follow up. A feasible problem is how to accurately estimate an initial value for the window size  $k$ , depending on past data and how to adjust it over the course of time with changing distribution. Another interesting problem would be to apply various methods of matrix sketching over the sliding window  $B_t$  [82], rather than to simply discard the data with the oldest timestamp and incorporate the new data. A further research conundrum might be to design the algorithm for a distributed system across multiple servers.

## 8 ACKNOWLEDGEMENTS

This research is partly supported by ICT Division Innovation Fund, Grant No. 56.00.0000.028.33.097.18-207 from Government of the Peoples' Republic of Bangladesh, Computing On Network Infrastructure for Pervasive Perception, Cognition and Action Research Center (CONIX) and Halicioğlu Data Science Institute's graduate prize fellowship award.

## REFERENCES

- [1] Nuno Abreu, Gonalo Costa, and Fernandes Marques. 2011. *Analise do perfil do cliente Recheio e desenvolvimento de um sistema promocional*. Ph.D. Dissertation.
- [2] Stefan Aeberhard, Danny Coomans, and Olivier de Vel. 1992. The classification performance of RDA. *Dept. of Computer Science and Dept. of Mathematics and Statistics, James Cook University of North Queensland, Tech. Rep* (1992), 92–01.
- [3] S Aeberhard, D Coomans, and O De Vel. 1992. Comparison of classifiers in high dimensional settings. *Dept. Math. Statist., James Cook Univ., North Queensland, Australia, Tech. Rep* 92, 02 (1992).
- [4] Charu C Aggarwal. 2003. A framework for diagnosing changes in evolving data streams. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 575–586.
- [5] Kerem Altun and Billur Barshan. 2010. Human activity recognition using inertial/magnetic sensor units. In *International workshop on human behavior understanding*. Springer, 38–51.
- [6] Kerem Altun, Billur Barshan, and Orkun Tunel. 2010. Comparative study on classifying human activities with miniature inertial and magnetic sensors. *Pattern Recognition* 43, 10 (2010), 3605–3620.
- [7] Cdric Archambeau and Francis R Bach. 2009. Sparse probabilistic projections. In *Advances in neural information processing systems*. 73–80.
- [8] Matej Artac, Matjaz Jogan, and Ales Leonardis. 2002. Incremental PCA for on-line visual learning and recognition. In *Pattern Recognition, 2002. Proceedings. 16th international conference on*, Vol. 3. IEEE, 781–784.
- [9] Kirk Baker. 2005. Singular value decomposition tutorial. *The Ohio State University* 24 (2005).
- [10] Billur Barshan and Murat Cihan Yksek. 2014. Recognizing daily and sports activities in two open source machine learning environments using body-worn sensor units. *Comput. J.* 57, 11 (2014), 1649–1667.
- [11] Jean-Patrick Baudry, Margarida Cardoso, Gilles Celeux, Maria Jos Amorim, and Ana Sousa Ferreira. 2012. Enhancing the selection of a model-based clustering with external qualitative variables. *arXiv preprint arXiv:1211.0437* (2012).
- [12] Mikhail Belkin and Partha Niyogi. 2003. Using manifold structure for partially labeled classification. In *Advances in neural information processing systems*. 953–960.
- [13] Rajen B Bhatt and M Gopal. 2008. FRCT: fuzzy-rough classification trees. *Pattern analysis and applications* 11, 1 (2008), 73–88.

- [14] Anil Bhattacharyya. 1943. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.* 35 (1943), 99–109.
- [15] Albert Bifet and Ricard Gavaldà. 2007. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*. SIAM, 443–448.
- [16] Avrim L Blum and Pat Langley. 1997. Selection of relevant features and examples in machine learning. *Artificial intelligence* 97, 1-2 (1997), 245–271.
- [17] Christos Boutsidis, Dan Garber, Zohar Karnin, and Edo Liberty. 2015. Online principal components analysis. *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, 887–901.
- [18] Pierluigi Casale, Oriol Pujol, and Petia Radeva. 2011. Human activity recognition from accelerometer data using a wearable device. In *Iberian Conference on Pattern Recognition and Image Analysis*. Springer, 289–296.
- [19] Pierluigi Casale, Oriol Pujol, and Petia Radeva. 2012. BeaStream-v0. 1: a new platform for Multi-Sensors Data Acquisition in Wearable Computing Applications. (2012).
- [20] Pierluigi Casale, Oriol Pujol, and Petia Radeva. 2012. Personalization and user verification in wearable systems using biometric walking patterns. *Personal and Ubiquitous Computing* 16, 5 (2012), 563–580.
- [21] Kenneth L Clarkson and David P Woodruff. 2009. Numerical linear algebra in the streaming model. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. ACM, 205–214.
- [22] Patricia Cohen, Stephen G West, and Leona S Aiken. 2014. *Applied multiple regression/correlation analysis for the behavioral sciences*. Psychology Press.
- [23] Belur V Dasarathy. 1980. Nosing around the neighborhood: A new system structure and classification rule for recognition in partially exposed environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 1 (1980), 67–71.
- [24] Claudio De Stefano, Francesco Fontanella, Marilena Maniaci, and Alessandra Scotto di Freca. 2011. A method for scribe distinction in medieval manuscripts using page layout features. In *International Conference on Image Analysis and Processing*. Springer, 393–402.
- [25] Claudio De Stefano, Marilena Maniaci, Francesco Fontanella, and A Scotto di Freca. 2018. Reliable writer identification in medieval manuscripts through page layout features: The “Avila” Bible case. *Engineering Applications of Artificial Intelligence* 72 (2018), 99–110.
- [26] Jamie DeCoster. 1998. Overview of factor analysis. (1998).
- [27] Dua Dheeru and Efi Karra Taniskidou. 2017. UCI Machine Learning Repository. <http://archive.ics.uci.edu/ml>
- [28] Thomas G Dietterich and Ghulum Bakiri. 1991. Error-correcting output codes: A general method for improving multiclass inductive learning programs. In *Association for the Advancement of Artificial Intelligence*. Citeseer, 572–577.
- [29] Thomas G Dietterich and Ghulum Bakiri. 1994. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research* 2 (1994), 263–286.
- [30] Chris Ding and Xiaofeng He. 2004. K-means clustering via principal component analysis. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 29.
- [31] Richard O Duda, Peter E Hart, and David G Stork. 1973. *Pattern classification and scene analysis*. Vol. 3. Wiley New York.
- [32] Richard O Duda, Peter E Hart, and David G Stork. 2012. *Pattern classification*. John Wiley & Sons.
- [33] George H Dunteman. 1989. *Principal components analysis*. Number 69. Sage.
- [34] Carl Eckart and Gale Young. 1936. The approximation of one matrix by another of lower rank. *Psychometrika* 1, 3 (1936), 211–218.
- [35] Tarek Elgamal, Maysam Yabandeh, Ashraf Aboulnaga, Waleed Mustafa, and Mohamed Hefeeda. 2015. sPCA: Scalable principal component analysis for big data on distributed platforms. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 79–91.
- [36] Brian S Everitt and Graham Dunn. 2001. *Applied multivariate data analysis*. Vol. 2. Wiley Online Library.
- [37] Weiguo Fan, Michael D Gordon, and Praveen Pathak. 2005. Effective profiling of consumer information retrieval needs: a unified framework and empirical comparison. *Decision Support Systems* 40, 2 (2005), 213–233.
- [38] Mark Fanty and Ronald Cole. 1991. Spoken letter recognition. In *Advances in Neural Information Processing Systems*. 220–226.
- [39] Ronald A Fisher. 1936. The use of multiple measurements in taxonomic problems. *Annals of eugenics* 7, 2 (1936), 179–188.
- [40] Rainer Hoch. 1994. Using IR techniques for text classification in document analysis. In *Special Interest Group on Information Retrieval (SIGIR’94)*. Springer, 31–40.
- [41] Michael Holmes, Alexander Gray, and Charles Isbell. 2007. Fast SVD for large-scale matrices. In *Workshop on Efficient Machine Learning at NIPS*, Vol. 58. 249–252.
- [42] Ian Jolliffe. 2011. *Principal component analysis*. Springer.
- [43] Ian T Jolliffe. 1990. Principal component analysis: a beginner’s guide—I. Introduction and application. *Weather* 45, 10 (1990), 375–382.
- [44] Thomas Kailath. 1967. The divergence and Bhattacharyya distance measures in signal selection. *IEEE transactions on communication technology* 15, 1 (1967), 52–60.
- [45] Zohar Karnin and Edo Liberty. 2015. Online pca with spectral bounds. In *Conference on Learning Theory*. 1129–1140.
- [46] Ron Kohavi and George H John. 1997. Wrappers for feature subset selection. *Artificial intelligence* 97, 1-2 (1997), 273–324.
- [47] Daphne Koller and Mehran Sahami. 1996. *Toward optimal feature selection*. Technical Report. Stanford InfoLab.
- [48] Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics* 22, 1 (1951), 79–86.
- [49] Ludmila I Kuncheva and William J Faithfull. 2014. PCA feature extraction for change detection in multidimensional unlabeled data. *IEEE transactions on neural networks and learning systems* 25, 1 (2014), 69–80.

- [50] Wenke Lee, Salvatore J Stolfo, and Kui W Mok. 1999. A data mining framework for building intrusion detection models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE, 120–132.
- [51] David D Lewis. 1992. Feature selection and feature extraction for text categorization. In *Proceedings of the workshop on Speech and Natural Language*. Association for Computational Linguistics, 212–217.
- [52] Haifeng Li, Tao Jiang, and Keshu Zhang. 2004. Efficient and robust feature extraction by maximum margin criterion. In *Advances in neural information processing systems*. 97–104.
- [53] Quanzhi Li, Armineh Nourbakhsh, Sameena Shah, and Xiaomo Liu. 2017. Real-time novel event detection from social media. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. IEEE, 1129–1139.
- [54] Yongmin Li, L-Q Xu, Jason Morphet, and Richard Jacobs. 2003. An integrated algorithm of incremental and robust pca. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, Vol. 1. IEEE, I–245.
- [55] Daw-Tung Lin. 2006. Facial expression classification using PCA and hierarchical radial basis function network. *Journal of information science and engineering* 22, 5 (2006), 1033–1046.
- [56] Raul HC Lopes. 2011. Kolmogorov-smirnov test. In *International encyclopedia of statistical science*. Springer, 718–720.
- [57] Moutinho Luiz and Huang Kun-huang. 2015. *Quantitative modelling in marketing and management*. World Scientific.
- [58] Prasanta Chandra Mahalanobis. 1936. On the generalized distance in statistics. National Institute of Science of India.
- [59] Aleix M Martínez and Avinash C Kak. 2001. Pca versus lda. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 2 (2001), 228–233.
- [60] Michael Mathioudakis and Nick Koudas. 2010. Twittermonitor: trend detection over the twitter stream. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 1155–1158.
- [61] Kevin Meagher, David Loiselle, and Rodger Koopman. 2012. Real time microgrid power analytics portal for mission critical power systems. US Patent 8,321,194.
- [62] Stuart E Middleton, Lee Middleton, and Stefano Modafferi. 2014. Real-time crisis mapping of natural disasters using social media. *IEEE Intelligent Systems* 29, 2 (2014), 9–17.
- [63] M Nikulin. 2001. Hellinger distance. Hazewinkel, Michiel, *Encyclopedia of Mathematics*. Springer, Berlin. doi 10 (2001), 1361684–1361686.
- [64] Erkki Oja and Juha Karhunen. 1985. On stochastic approximation of the eigenvectors and eigenvalues of the expectation of a random matrix. *Journal of mathematical analysis and applications* 106, 1 (1985), 69–84.
- [65] Cam Nugent (originator). [n. d.]. S&P 500 stock data. <https://www.kaggle.com/camnugent/sandp500>
- [66] G. Chaudhuri (originator). [n. d.]. Bhattacharyya distance. [https://www.encyclopediaofmath.org/index.php/Bhattacharyya\\_distance](https://www.encyclopediaofmath.org/index.php/Bhattacharyya_distance)
- [67] Nhathi Phan, Soon Ae Chun, Manasi Bhole, and James Geller. 2017. Enabling real-time drug abuse detection in tweets. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*. IEEE, 1510–1514.
- [68] Daniel Preotiu-Pietro, Sina Samangooei, Trevor Cohn, Nicholas Gibbins, and Mahesan Niranjan. 2012. Trendminer: An architecture for real time analysis of social media text. In *Sixth International Association for the Advancement of Artificial Intelligence Conference on Weblogs and Social Media*.
- [69] Abdulhakim A Qahtan, Basma Alharbi, Suojin Wang, and Xiangliang Zhang. 2015. A pca-based change detection framework for multidimensional data streams: Change detection in multidimensional data streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 935–944.
- [70] Jayant G Rohra, Boominathan Perumal, Swathi Jamjala Narayanan, Priya Thakur, and Rajen B Bhatt. 2017. User localization in an indoor environment using fuzzy hybrid of particle swarm optimization & gravitational search algorithm with neural networks. In *Proceedings of Sixth International Conference on Soft Computing for Problem Solving*. Springer, 286–295.
- [71] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.
- [72] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. 2010. Earthquake shakes Twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*. ACM, 851–860.
- [73] Terence D Sanger. 1989. Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural networks* 2, 6 (1989), 459–473.
- [74] Tamas Sarlos. 2006. Improved approximation algorithms for large matrices via random projections. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE, 143–152.
- [75] Lindsay I Smith. 2002. *A tutorial on principal components analysis*. Technical Report.
- [76] Jun-ichi Takeuchi and Kenji Yamanishi. 2006. A unifying framework for detecting outliers and change points from time series. *IEEE transactions on Knowledge and Data Engineering* 18, 4 (2006), 482–492.
- [77] Md Mehrab Tanjim and Muhammad Abdullah Adnan. 2018. sSketch: A Scalable Sketching Technique for PCA in the Cloud. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 574–582.
- [78] Joshua B Tenenbaum, Vin De Silva, and John C Langford. 2000. A global geometric framework for nonlinear dimensionality reduction. *science* 290, 5500 (2000), 2319–2323.
- [79] Michael E Tipping and Christopher M Bishop. 1999. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 61, 3 (1999), 611–622.
- [80] Satoshi Watanabe and Nikhil Pakvasa. 1973. Subspace method of pattern recognition. In *Proc. 1st. IJ CPR*. 25–32.
- [81] Andrew R Webb. 2003. *Statistical pattern recognition*. John Wiley & Sons.
- [82] Zhewei Wei, Xuancheng Liu, Feifei Li, Shuo Shang, Xiaoyong Du, and Ji-Rong Wen. 2016. Matrix sketching over sliding windows. In *Proceedings of the 2016 International Conference on Management of Data*. ACM, 1465–1480.

- [83] Juyang Weng, Yilu Zhang, and Wey-Shiuan Hwang. 2003. Candid covariance-free incremental principal component analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25, 8 (2003), 1034–1040.
- [84] Worldometers and 7 Billion World. [n. d.]. Internet Live Stats. <http://www.internetlivestats.com/one-second/#tweets-band>
- [85] Jun Yan, Ning Liu, Benyu Zhang, Shuicheng Yan, Zheng Chen, Qiansheng Cheng, Weiguo Fan, and Wei-Ying Ma. 2005. OCFS: optimal orthogonal centroid feature selection for text categorization. In *Proceedings of the 28th annual international ACM Special Interest Group on Information Retrieval (SIGIR) conference on Research and development in information retrieval*. ACM, 122–129.
- [86] Yiming Yang and Jan O Pedersen. 1997. A comparative study on feature selection in text categorization. In *Icml*, Vol. 97. 412–420.