Streaming Spectral Clustering

Shinjae Yoo
Computational Science Center
Brookhaven National Laboratory
Upton, New York 11973-5000
Email: shinjae@gmail.com

Hao Huang
Machine Learning Laboratory
General Electric Global Research
San Ramon, CA 94583
Email: haohuanghw@gmail.com

Shiva Prasad Kasiviswanathan
Samsung Research America
Mountain View, CA 94043
Email: kasivisw@gmail.com

***Abstract*—Clustering is a classical data mining task used for discovering interrelated pattern of similarities in the data. In many modern day domains, data is getting continuously generated as a stream. For scalability reasons, clustering the points in a data stream requires designing single pass, limited memory *streaming clustering* algorithms. However, the performance of the known streaming clustering algorithms, that typically use K-means (or its variants) on the original feature space, tend to suffer when the feature space is high-dimensional. To overcome this problem, we propose a streaming spectral clustering algorithm. Our algorithm maintains an approximation of the normalized Laplacian of the data stream over time and efficiently updates the changing eigenvectors of this Laplacian in a streaming fashion. It requires just one pass over the data, consumes limited memory, and is stable to the ordering of the data stream. We provide a theoretical analysis of our streaming spectral clustering algorithm and our experimental results show that while gaining in scalability, its performance compares well with other known batch/streaming clustering approaches.**

I. Introduction

Clustering is an important unsupervised learning technique and usually it is among the very first steps used in modern data analysis. The essential characteristic for any good scalable clustering algorithm is the ability to handle huge volumes of data in a high dimensional feature space. Most modern high dimensional data such as documents, images, and multimedia from the web naturally arrives in a streaming fashion. However, detecting clusters from such a large volume and high-dimensional data stream is also a challenging problem due to the following reasons: 1) the data stream could be infinite, so that any off-line algorithm that attempts to store the entire stream for analysis will eventually run out of memory, 2) the clusters evolve dynamically over time due to *concept-drift*, so that old clusters may be merged and new clusters may be created, 3) there is a curse of dimensionality for many popular clustering approaches, and 4) for various online applications, it is important to obtain the clusters in (near) real-time.

In this paper, we investigate clustering in a streaming setting. Although there is a considerable previous literature on streaming clustering algorithms [47], [42], [30] and most of these and other streaming clustering algorithms do effectively handle large volume data stream, their performance tend to degrade in presence of high-dimensional data [4]. To overcome this problem, we propose a streaming adaptation of the spectral clustering algorithm. Spectral clustering has gained immense popularity in the last decade in the data mining community because of its ability to discover embedded structure in the data (a.k.a. manifolds). In its most popular form, the spectral clustering algorithm involves two steps: first, the eigenvectors of the normalized Laplacian constructed using a kernel function are used to embed the dataset, and second, the K-means clustering algorithm is applied to the embedded dataset [40], [32].

The first challenge for adapting spectral clustering to a streaming setting lies in the construction of the normalized Laplacian. Constructing the exact normalized Laplacian is inherently a batch operation as it requires access to the whole data and storage space that is quadratic in the length of the stream. Therefore, the traditional Laplacian construction techniques fail to adapt to a streaming setting. In this paper, we focus on the two most popular kernels, cosine and Gaussian, and for both these kernels, we propose a streaming Laplacian approximation technique.

The next challenge lies in updating the spectral embedding of the Laplacian efficiently and effectively so that we could process data stream in near real-time. For this we utilize recent ideas from matrix sketching[1] to maintain a low-rank approximation of the entire observed data at every time step, and this approximation is continually updated as new data arrives. For matrix sketching, we adapt a recent algorithm (called *Frequent Directions*) proposed by Liberty [27]. Using this low-rank approximation, at every time step, we align the learned spectral embedding to have the same basis between every two adjacent time steps, so that it would not be sensitive to concept-drift. Given the embedding of the data stream, we then apply a streaming K-means algorithm [41] to handle the creation and consolidation of the clusters.

We present a theoretical analysis of our algorithm and show that under certain realistic assumptions, our constructed streaming embedding is a good approximation to the embedding created by an expensive batch technique. To the best of our knowledge, ours is the first streaming spectral clustering algorithm operating in a *true* data streaming setting (i.e., with one pass over any data sample). Our proposed streaming spectral clustering algorithm is effective and efficient in the following ways:

(a) It is space and time efficient while requiring only one pass over the data with the limited memory footprint. Let m be the data dimension. If at time t, n_t points arrive in the data stream, our algorithm requires just $O(m\ell + mn_t)$ space and $O(\max\{mn_t\ell, m\ell^2\})$ time, where the sketch matrix is of dimension $m \times \ell$. In practice, it suffices to set ℓ much

[1] Informally, a sketch of a matrix Z is another matrix Z' that is of smaller size than Z, but still approximates it well [27].

smaller compared to m. Therefore, both the time and space requirements are almost linear in the input size (as the input at time t is an $m \times n_t$ matrix).

(b) It easily adapts to unseen patterns on the data stream. At every time step t, it provides an updated spectral embedding of the whole stream that has arrived till time t. Such embedding can then be used to provide a cluster list showing the assignments of all the data points in the stream.

Empirical studies show that our approach is effective and efficient in terms of both space and time, compared to other popular batch/streaming clustering algorithms on various datasets from text, image, network, and collaborative filtering domains.

II. Background

In this section, we will briefly review the basic ideas behind spectral clustering and streaming K-means approaches. We start off by formally defining our notation. We denote $[n] = 1 : n$. Vectors are always in column-wise fashion and are denoted by boldface letters. For a vector $\mathbf{v}$, $\mathbf{v}^\top$ denotes its transpose and $\|\mathbf{v}\|$ denotes its Euclidean norm. For a vector $(a_1, \ldots, a_m) \in \mathbf{R}^m$, $diag(a_1, \ldots, a_m) \in \mathbf{R}^{m \times m}$ denotes a diagonal matrix with $a_1, \ldots, a_m$ as its diagonal entries. Let $\mathbb{I}_m$ denote an identity matrix of dimension $m \times m$. For a matrix $Z \in \mathbf{R}^{m \times n}$, row_sum$(Z) \in \mathbf{R}^m$ is a vector with i entry equaling the sum of entries in the ith row of Z. We use $z_{i,j}$ to denote its (i,j)th element of Z. Spectral norm is defined as $\|Z\| = \sup\{\|Z\mathbf{v}\| : \|\mathbf{v}\| = 1\}$. We also use entry-wise norms denoted by $\|Z\|_p$, where $p = 2$ gives (Frobenius norm) $\|Z\|_F^2 = \sum_{ij} z_{i,j}^2$, and $p = \infty$ gives $\|Z\|_\infty = \max_{i,j} |z_{i,j}|$. Given a set of matrices, $Z_1, \ldots, Z_t$, we use the notation $Z_{[t]}$ to denote the matrix obtained by horizontally concatenating $Z_1, \ldots, Z_t$, i.e., $Z_{[t]} = [Z_1, \ldots Z_t]$.

We use $\textsc{Svd}(Z)$ to denote the singular value decomposition of Z, i.e., $\textsc{Svd}(Z) = U\Sigma V^\top$. Here U is an $m \times m$ orthogonal matrix, Σ is an $m \times n$ diagonal matrix, and V is an $n \times n$ orthogonal matrix. The diagonal entries of Σ, where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m$ (given $m \leq n$), are known as the singular values of Z. We follow the common convention to list the singular values in non-increasing order. For a symmetric matrix $S \in \mathbf{R}^{m \times m}$, we use $\textsc{Eig}(S)$ to denote its eigenvalue decomposition, i.e., $U\Lambda U^\top = \textsc{Eig}(S)$. Here U is an $m \times m$ orthogonal matrix and Λ is an $m \times m$ diagonal matrix whose (real) entries are $\lambda_1, \ldots, \lambda_m$ are known as the eigenvalues of S (again listed in non-increasing order).

The best rank-k approximation (in both the spectral and Frobenius norm sense) to a matrix $Z \in \mathbf{R}^{m \times n}$ is $Z_{(k)} = \sum_{i=1}^k \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$, where σ_i $(i \leq k)$ are the top-k singular values of Z, with associated left and right singular vectors $\mathbf{u}_i \in \mathbf{R}^m$ and $\mathbf{v}_i \in \mathbf{R}^n$, respectively. We use $\textsc{Svd}_k(Z)$ to denote the truncated singular value decomposition of $Z_{(k)}$, i.e., $Z_{(k)} = \textsc{Svd}_k(Z) = U_{(k)}\Sigma_{(k)}V_{(k)}^\top$. Here $\Sigma_{(k)} = diag(\sigma_1, \ldots, \sigma_k) \in \mathbf{R}^{k \times k}$, $U_{(k)} = [\mathbf{u}_1, \ldots, \mathbf{u}_k] \in \mathbf{R}^{m \times k}$, and $V_{(k)} = [\mathbf{v}_1, \ldots, \mathbf{v}_k] \in \mathbf{R}^{n \times k}$. The following well-known theorem bounds the approximation error of the best rank-k approximation.

Theorem 2.1: [Golub *et al.* [20]] Let $Z \in \mathbf{R}^{m \times n}$ with $n > m$, and let $\sigma_1 \geq \cdots \geq \sigma_m$ be the singular values of Z. Let $\textsc{Svd}_k(Z) = U_k\Sigma_k V_k^\top$. Then

$$\min_{rank(X) \leq k} \|Z - X\|_F = \|Z - U_{(k)}\Sigma_{(k)}V_{(k)}^\top\|_F = \sqrt{\sum_{j=k+1}^{m} \sigma_{k+1}^2}.$$

A. Spectral Clustering

Algorithm 1: SpectralClustering [32]

Input: Input data $Y = [\mathbf{y}_1, \ldots, \mathbf{y}_n] \in R^{m \times n}$ and $k \in \mathbf{R}$ (number of clusters)
Output: Cluster assignments of n instances

1 Construct the kernel affinity matrix $W \in R^{n \times n}$, e.g.,
a) $w_{i,j} \leftarrow \exp\left(\frac{-\|\mathbf{y}_i - \mathbf{y}_j\|^2}{2\sigma^2}\right)$ (Gaussian kernel), or
b) $w_{i,j} \leftarrow \frac{\langle \mathbf{y}_i, \mathbf{y}_j \rangle}{\|\mathbf{y}_i\| \|\mathbf{y}_j\|}$ (Cosine kernel)
2 Degree matrix $D \leftarrow diag(d(\mathbf{y}_1), \ldots, d(\mathbf{y}_n))$ where the degree of a point $\mathbf{y}_i$, $d(\mathbf{y}_i) = \sum_{j=1}^n w_{ij}$
3 Normalized Laplacian $L_{sym} \leftarrow I - D^{-1/2}WD^{-1/2}$
4 $P\Lambda P^\top \leftarrow \textsc{Eig}_k(L_{sym})$
5 $V \leftarrow$ normalized P with unit L_2 row norms
6 Cluster the rows of V into k clusters (using K-means)

Algorithm SpectralClustering outlines a very popular approach for clustering based on embedding the data set using a kernel function, and utilizing the top eigenvectors of the normalized Laplacian to discover the underlying clusters. Spectral clustering has strong connections to the graph partitioning problem, in that eigenspaces are used to solve relaxed forms of the balanced graph partitioning problem [32]. Another aspect of spectral clustering is that it can capture the manifold structure of data, which is difficult or impossible to achieve by K-means style algorithms. Algorithm SpectralClustering first builds the affinity or similarity matrix (we show the most popular two kernels, cosine kernel and Gaussian kernel but it is not limited to these two). Then we construct a Laplacian. In Algorithm SpectralClustering, we present the most popular symmetric normalized Laplacian (L_{sym}). Other popular choices for Laplacian include the unnormalized Laplacian $L_{un} = D - W$ and a random walk normalized one $L_{rw} = I - D^{-1}W$. Once we compute a Laplacian, we do rank-k eigenvalue decomposition to get the spectral embedding restricted to the top-k eigenvalues. A row normalization step is used before one of the simple clustering algorithms (such as K-means) is applied on V to identify the clusters.

B. Streaming K-means

In this section, we discuss a recent fast streaming Euclidean K-means clustering algorithm of Shindler *et al.* [41] (presented in Algorithm StrmKMeans). As Spectral Clustering uses K-means as its final clustering step, in our proposed approach we apply this streaming K-means algorithm on the constructed manifolds. If the number of input data points is n and the goal is to output k clusters, the algorithm maintains a set of micro-clusters or facilities, denoted as $\mathcal{C}$, which are consolidated to form k clusters at the end. Algorithm StrmKMStep, which forms the key part of Algorithm StrmKMeans, details how to add a new datapoint $\mathbf{x}$ into the current facility set $\mathcal{C}$. With probability δ/f where δ is the square of the minimum

Euclidean distance between $\mathbf{x}$ and any facility in $\mathcal{C}$ and $f = 1/(k(1+\log n))$ (noted as facility cost), a new facility is initialized with just $\mathbf{x}$ in it. With remaining probability, $\mathbf{x}$ is assigned to the closest existing facility. The algorithm tries to ensure that there are no more than $\rho = \Theta(k \log n)$ facilities. If the number of facilities reaches ρ, then the algorithm reorganizes (merges) facilities to obtain a smaller number of facilities and correspondingly updates the facility centers. A final "ball K-means" step (which is reminiscent to Lloyd-like re-clustering) is performed to obtain the cluster centers. Shindler *et al.* [41] theoretically analyzed the performance of Algorithm StrmKMeans and also provided experimental support for its performance. However, since Algorithm StrmKMeans operates in the input dimension, it would suffer from the *curse of dimensionality* on a high-dimensional dataset. However, this is not an issue in our streaming spectral clustering approach (presented in the next section), as the Streaming K-means algorithm operates on a lower-dimensional manifold.

Algorithm 2: StrmKMeans (Restated from [41])

Input: Data stream $\mathcal{S}$, ρ is the maximum number of facilities, k is the number of clusters, β is a scalar.
Output: the facility set $\mathcal{C}$ and the datapoint assignment

Initialize $f = 1/(k(1+\log n))$ and an empty set $\mathcal{C}$
while *stream $\mathcal{S}$ not finished* **do**
  Read the next point $\mathbf{x}$ from the stream
  $[\mathcal{C}, f] \leftarrow$ StrmKMStep $(\mathbf{x}, \mathcal{C}, \rho, \beta, f)$
end
Run batch K-means on weighted points $\mathcal{C}$
Perform ball K-means [9] on the resulting clusters to obtain the final cluster centers $\mathcal{C}$

Algorithm 3: StrmKMStep (Restated from [41])

Input: datapoint $\mathbf{x}$, current facility set $\mathcal{C}$, ρ is the maximum number of facilities, β is a scalar, f is the current facility cost
Output: new facility cost f, new facility set $\mathcal{C}$ and datapoint assignment

Measure $\delta = \min_{\mathbf{y} \in \mathcal{C}} \|\mathbf{x} - \mathbf{y}\|^2$ (if $\mathcal{C}$ empty then $\delta = f$)
Let r be a uniform random number between 0 and 1
if $r \leq \delta / f$ **then**
  Set $\mathcal{C} \leftarrow \mathcal{C} \cup \{\mathbf{x}\}$
else
  Assign $\mathbf{x}$ to its closest facility in $\mathcal{C}$
end
while $|\mathcal{C}| > \rho$ **do**
  Set $f \leftarrow \beta f$
  Move each $\mathbf{z} \in \mathcal{C}$ to the center-of-mass-of the points assigned to that cluster
  Let $w_{\mathbf{z}}$ be the number of points assigned to $\mathbf{z} \in \mathcal{C}$
  $\hat{\mathcal{C}} \leftarrow$ the first facility from $\mathcal{C}$
  for *each* $\mathbf{z} \in \mathcal{C}$ **do**
    Let $\delta = \min_{\mathbf{y} \in \hat{\mathcal{C}}} \|\mathbf{z} - \mathbf{y}\|^2$
    if *probability* δ / f *event occurs* **then**
      Set $\hat{\mathcal{C}} \leftarrow \hat{\mathcal{C}} \cup \{\mathbf{y}\}$
    else
      Assign $\mathbf{z}$ to its closest facility in $\hat{\mathcal{C}}$
    end
  end
  Set $\mathcal{C} \leftarrow \hat{\mathcal{C}}$
end

III. Streaming Spectral Clustering Algorithm

There are two major difficulties that have to be overcome while designing algorithms for spectral clustering under a streaming environment: 1) The first challenge is how to efficiently construct a normalized Laplacian matrix over a stream (the first three steps of Algorithm SpectralClustering). The Laplacian construction is inherently non-streaming task because the affinity matrix W requires access to the whole dataset, and so do the degree matrix D and the Laplacian L. Therefore, we need novel ideas to approximate the Laplacian matrix in a streaming setting. 2) Given the Laplacian matrix, the second challenge is in constructing the streaming manifold V, which is hard because the concept-drift (topic changes) in the stream causes these embeddings to vary a lot over time. Therefore, we need novel techniques for robust streaming embedding construction that is both efficient and capable to adapt to the inherent concept-drift.

Before introducing our proposed algorithm, we start by describing our streaming setup. We assume that the data arrives in streams and each datapoint has a timestamp that indicates when it arrives. Let $\mathcal{S} = \{Y_t \in \mathbf{R}^{m \times n_t},\ t = 1, 2, \dots\}$ denote a sequence of streaming data matrices, where Y_t represents the data points arriving at time t. Here m is the size of the feature space, and $n_t \geq 1$ is the number of data points arriving at time t.[2] Let $Y_{[t]} = [Y_1, \dots, Y_t] \in \mathbf{R}^{m \times n_{[t]}}$ denote the stream of all data points arriving till time t.

In the remainder of this section, we present and analyze our streaming spectral clustering approach, outlined in Algorithm StrmSC. We start by describing the various building blocks for our algorithm.

A. Normalized Laplacian Construction

Affinity Matrix. The dimensions of the affinity matrix grow as we continually observe data from the stream, however, we do not need to construct the affinity matrix explicitly for spectral clustering. Here we focus on two popular kernel affinity constructions.

For the case of cosine kernel, at time t, given the stream $Y_{[t]}$ with each column (point) having unit L_2-norm, the affinity matrix $W_{cos} = Y_{[t]}^\top Y_{[t]}$. Let $\text{Svd}(Y_{[t]}) = U_c \Sigma_c V_c^\top$. We can get the eigenvalue decomposition without constructing affinity matrix because:

$$\text{Eig}(W_{cos}) = \text{Eig}(Y_{[t]}^\top Y_{[t]}) = V_c \Sigma_c^2 V_c^\top.$$

In other words, as long as we can do a streaming singular value decomposition of $Y_{[t]}$, we can get the spectral embedding V_c required for spectral clustering.

[2] Many prior streaming algorithms assume that only one point arrives at every time step, i.e., $n_t = 1$. By allowing n_t to be ≥ 1, we allow more flexibility in our setup.

Algorithm 4: StrmSC

Input: Data stream $\mathcal{S}$, $\ell \in \mathbf{R}$, $\kappa\,(\leq \ell) \in \mathbf{R}$, $\rho \in \mathbf{R}$, $k \in \mathbf{R}$, and $\beta \in \mathbf{R}$
Output: Cluster assignments of all instances in $\mathcal{S}$

Initialize $f = 1/(k(1+\log n))$ and an empty set $\mathcal{C}$
$\mathbf{s}_0 \leftarrow$ all zeros vector $\in \mathbf{R}^m$
$B_0 \leftarrow$ all zeros matrix $\in \mathbf{R}^{m\times \ell}$
$\tilde{U}_0 \leftarrow \tilde{U}_1$ (to skip the first rotation)
while *stream $\mathcal{S}$ not finished* **do**
  let $Y_t \in \mathbf{R}^{m\times n_t}$ be the batch with timestamp t in stream $\mathcal{S}$
  $[B_t, \tilde{U}_{t_{(\kappa)}}, \breve{V}_t, \mathbf{s}_t] \leftarrow$ StrmEmb $(Y_t, B_{t-1}, \mathbf{s}_{t-1}, \kappa)$
  $\hat{V}_t \leftarrow$ normalized $\breve{V}_t$ with unit L_2 row norms
  $R_t \leftarrow \tilde{U}^\top_{t-1_{(\ell)}} \tilde{U}_{t_{(\ell)}}$
  Replace each $\mathbf{z} \in \mathcal{C}$ by $R_t\mathbf{z}$
  for *each column* $\mathbf{x}$ *in* $\hat{V}_t$ **do**
    $[\mathcal{C}, f] \leftarrow$ StrmKMStep $(\mathbf{x}, \mathcal{C}, \rho, \beta, f)$
  end
end
Run batch K-means on weighted points $\mathcal{C}$ to form k clusters

Algorithm 5: StrmEmb

Input: $Y_t \in \mathbf{R}^{m\times n_t}$, $B_{t-1} \in \mathbf{R}^{m\times \ell}$, $\mathbf{s}_{t-1} \in \mathbf{R}^m$, and $\kappa \in \mathbf{R}$
Output: B_t, $\tilde{U}_{t_{(\ell)}}$, $\breve{V}_t$, $\mathbf{s}_t$

$\mathbf{s}_t \leftarrow \mathbf{s}_{t-1} + \text{row\_sum}(Y_t)$
$\mathbf{c}_t \leftarrow \mathbf{s}_t/\|\mathbf{s}_t\|$
$\tilde{D}_t \leftarrow diag(\langle \mathbf{y}_1, \mathbf{c}_t\rangle, \ldots, \langle \mathbf{y}_{n_t}, \mathbf{c}_t\rangle)$ where $Y_t = [\mathbf{y}_1, \ldots, \mathbf{y}_{n_t}]$
$\tilde{Y}_t \leftarrow Y_t \tilde{D}_t^{-1/2}$
$C_t \leftarrow [B_{t-1}, \tilde{Y}_t]$
$\tilde{U}_{t_{(\ell)}} \tilde{\Sigma}_{t_{(\ell)}} \tilde{V}^\top_{t_{(\ell)}} \leftarrow \text{SVD}_\ell(C_t)$
$\breve{\Sigma}_t \leftarrow diag\left(\sqrt{\tilde{\sigma}^2_{t_1} - \tilde{\sigma}^2_{t_\ell}}, \ldots, \sqrt{\tilde{\sigma}^2_{t_{\ell-1}} - \tilde{\sigma}^2_{t_\ell}}, 0\right)$ where $\tilde{\Sigma}_{t_{(\ell)}} = diag(\tilde{\sigma}_{t_1}, \ldots, \tilde{\sigma}_{t_\ell})$
$B_t \leftarrow \tilde{U}_{t_{(\ell)}} \breve{\Sigma}_{t_{(\ell)}}$
$\tilde{U}_{t_{(\kappa)}} \leftarrow [\mathbf{u}_1, \ldots, \mathbf{u}_\kappa]$ where $\tilde{U}_{t_{(\ell)}} = [\mathbf{u}_1, \ldots, \mathbf{u}_\ell]$
$\tilde{\Sigma}_{t_{(\kappa)}} \leftarrow diag(\tilde{\sigma}_{t_1}, \ldots, \tilde{\sigma}_{t_\kappa})$
$\breve{V}_t^\top \leftarrow \tilde{\Sigma}^{-1}_{t_{(\kappa)}} \tilde{U}_{t_{(\kappa)}} \tilde{Y}_t$

For the case of Gaussian kernel, given the stream $Y_{[t]}$, the (i,j)th entry of the affinity matrix W_{gau} equals $\exp(-\|\mathbf{y}_i - \mathbf{y}_j\|^2/2\sigma^2)$. Therefore the construction of W_{gau} is a little more complicated since the kernel function is not linear. However, we can still approximate by first linearizing the kernel using the random Fourier basis projection of Rahimi and Recht [38]. The idea is to explicitly map the data to a Euclidean inner product space using a randomized feature map $h : \mathbf{R}^m \to \mathbf{R}^d$ such that the kernel evaluation can be approximated by the Euclidean inner product between the transformed pair. Concretely, for any two data points $\mathbf{x}, \mathbf{y} \in \mathbf{R}^m$, the Gaussian kernel can be expressed as the expectation of $h(\mathbf{x})^\top h(\mathbf{y})$, where $h(\mathbf{x}) = \cos(\omega^\top \mathbf{x} + \mathbf{b})$, $\omega \in \mathbf{R}^{m\times d}$ is drawn from an appropriately scaled Gaussian distribution, $\mathbf{b} \in \mathbf{R}^d$ is drawn from a uniform distribution on $[0, \pi)$, and the cosine function is applied entry-wise. According to the analysis in [38], as the number of samples d increases, the error of this random Fourier bases approximation goes to zero. The above projection can be formulated as the following steps:

1) Draw d i.i.d. samples $\omega(1), \ldots, \omega(d)$ from $p(\omega)$ where $\omega \sim \frac{1}{\sigma^2}\mathcal{N}(0,1))$ and $p(\cdot)$ is the fast Fourier transform;
2) Draw d i.i.d. samples (offsets) $\mathbf{b}_1, \ldots, \mathbf{b}_d$ from uniform distribution on $[0, \pi]$;
3) Compute the projected data where $h(\mathbf{x}) = \cos(\omega^\top \mathbf{x} + \mathbf{b})$;

Since each point can be projected independently of the others, it is ideal for streaming applications. The idea is to replace each point $\mathbf{y}$ in the stream with its projection $h(\mathbf{y})$. Let $H_{[t]} = [h(\mathbf{y}_1), \ldots, h(\mathbf{y}_{n_{[t]}})] \in \mathbf{R}^{d\times n_{[t]}}$ with $\text{SVD}(H_{[t]}) = U_g \Sigma_g V_g^\top$,

$$\text{EIG}(W_{gau}) \approx \text{EIG}(H_{[t]}^\top H_{[t]}) = V_g \Sigma_g^2 V_g^\top.$$

Therefore, Gaussian kernel could be thought as applying cosine kernel (ignoring the normalization) on H. In the following, we assume that each stream datapoint has been transformed using the above projection operation, and therefore, can focus only on the cosine kernel case.

Degree Matrix (Steps 1-3 in Algorithm StrmEmb). The next issue is that as we observe more data from the stream, the diagonal degree matrix is increasing not only in its size but also in the values of its entries (because the affinity matrix W is non-negative). We overcome this problem using a clever trick. At time t, the degree of a datapoint $\mathbf{y}$ in the stream $Y_{[t]}$ for the cosine kernel can be computed as follows:

$$d_t(\mathbf{y}) = \sum_{\mathbf{z}\in Y_{[t]}} \mathbf{y}^\top \mathbf{z} = \mathbf{y}^\top \sum_{\mathbf{z}\in Y_{[t]}} \mathbf{z} = \mathbf{y}^\top \mathbf{s}_t \tag{1}$$

where $\mathbf{s}_t = \sum_{\mathbf{z}\in Y_{[t]}} \mathbf{z} = \text{row\_sum}(Y_{[t]})$ is the dataset sum vector at time t. In other words, without constructing affinity matrix, we can compute the degree matrix as long as we know the whole dataset sum vector. In streaming setting, given $\mathbf{s}_t$, we can compute the degree of the data points in Y_t in the stream $Y_{[t]} = [Y_1, \ldots, Y_t]$. However, as we observe more and more data points, the norm of $\mathbf{s}_t$ keeps increasing, and thus at time t, we have to recalculate the degrees of all data points that have arrived before t. To overcome this, we propose to use the dataset centroid $\mathbf{c}_t = \mathbf{s}_t/\|\mathbf{s}_t\|$ for the degree approximation as follows. Define,

$$\tilde{d}_t(\mathbf{y}) = \frac{\mathbf{y}^\top \mathbf{s}_t}{\|\mathbf{s}_t\|} = \mathbf{y}^\top \mathbf{c}_t. \tag{2}$$

Note that in a streaming setup, it is reasonable to assume that the dataset centroid $\mathbf{c}_t = \mathbf{s}_t/\|\mathbf{s}_t\|$ (for sufficiently large t) is *stable* (i.e., $\mathbf{c}_t$ changes slowly with time) because most topic distributions would have been observed. The following lemma bounds the centroid shift between any two consecutive time steps.

Lemma 3.1: Let $\mathbf{y} \in \mathbf{R}^m$ be a unit vector. The shift in the above defined normalized degree of $\mathbf{y}$ between time steps t and $t+1$ satisfies:

$$\tilde{d}_{t+1}(\mathbf{y}) - \tilde{d}_t(\mathbf{y}) \leq \sqrt{n_{t+1}/(n_{[t+1]} - n_{t+1})}$$

where n_{t+1} is the number of points arriving at time $t+1$ and $n_{[t+1]}$ is the total number of points observed till time $t+1$.

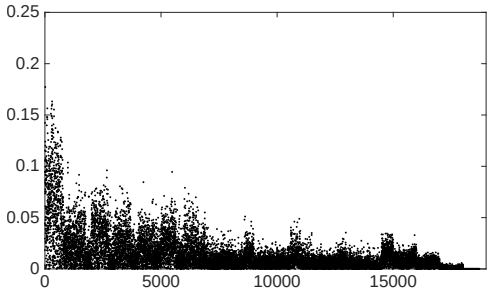
Proof:

$$\begin{aligned}\tilde{d}_{t+1}(\mathbf{y}) - \tilde{d}_t(\mathbf{y}) &= \mathbf{y}^\top \mathbf{c}_{t+1} - \mathbf{y}^\top \mathbf{c}_t \\ &\leq \|\mathbf{y}\| \|\mathbf{c}_{t+1} - \mathbf{c}_t\| \\ &= \left\| \frac{\mathbf{s}_{t+1}}{\|\mathbf{s}_{t+1}\|} - \frac{\mathbf{s}_t}{\|\mathbf{s}_t\|} \right\| \\ &\leq \frac{\|\mathbf{s}_{t+1} - \mathbf{s}_t\|}{\|\mathbf{s}_t\|} \\ &= \sqrt{\frac{n_{t+1}}{n_{[t+1]} - n_{t+1}}}\end{aligned}$$

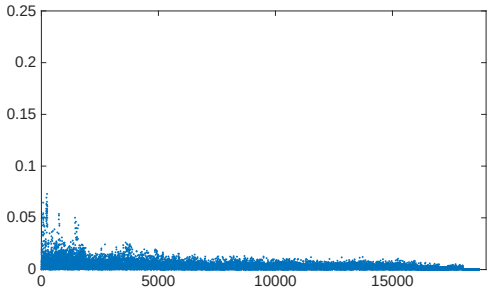
Here we used that $\|\mathbf{y}\| = 1$. The second inequality uses the fact that the entries in $\mathbf{s}_t$ are ≥ 0 and are entry-wise dominated by $\mathbf{s}_{t+1}$. ■

The above lemma shows that the normalized degree of a point in a stream does not vary much, once sufficiently enough data has been observed, as typically $n_{[t]} \gg n_t$. This suggests that we could set the normalized degree of a datapoint $\mathbf{y}$ arriving at time t as $\tilde{d}_t(\mathbf{y})$ and not adjust it in the subsequent time steps. From arguments above, we know that $\tilde{d}_t(\mathbf{y})$ continues to remain a good approximation to the actual normalized degree of $\mathbf{y}$ in the stream even as new data arrives. We use this normalized degree approximation in the remainder of this paper.

We now test the quality of this approximation in practice. Figure 1 shows the degree approximation error for two different stream orders (topic or timestamp) on 20 newsgroup (20NG) dataset (for the dataset details, please refer to Section V). The batchsize (n_t) is set to 1000. Figure 1(a) shows the result on a stream ordered by topic where all documents (points) belonging to same topic (cluster) are together. This is the most difficult case to approximate because whenever a new topic is introduced, it will create a significant concept-drift. In spite of observing clear topic changes in the stream, the degree approximation error decreases as we observe more and more documents. The final average true normalized degree is 0.1388 but the average approximated degree is 0.1441 and on average the degree approximation error is 0.0053 ($< 4\%$). Figure 1(b) shows the more realistic case, where the documents are ordered by their true timestamps. In this case the degree approximation error is much smaller, on average 0.0009 ($< 2\%$). There are still a few spikes in the beginning of the stream but the error decreases over time.



(a) Ordered by topic



(b) Ordered by timestamp

Fig. 1. The degree approximation error plot for each document on 20 newsgroup (20NG) dataset. Y-axis is the sum of absolute error between approximated degree and true normalized degree using the whole dataset. Figure 1(a) shows the stream data ordered by the topic, which is the most difficult case to approximate and Figure 1(b) shows the stream data ordered by the timestamp, which is the more realistic case.

For Gaussian kernel to overcome these issues, by using the random Fourier projection described earlier, we can use the same degree approximation on the projected space.

Normalized Laplacian (Step 4 in Algorithm StrmEmb). After constructing the affinity matrix and the degree normalization matrix, we have to construct the eigenvectors of the symmetric normalized Laplacian. Here we claim that without constructing the Laplacian explicitly, we can determine the eigenvectors of L_{sym}. In a batch setting, given the affinity matrix $W_{cos} = Y^\top Y$ and the corresponding degree matrix D, we can construct a symmetric normalized graph Laplacian as follows:

$$L_{sym} = I - W_{sym} = I - D^{-1/2} W_{cos} D^{-1/2}$$

There is an equivalence between the eigenvectors of L_{sym} and W_{sym} as we establish in the following lemma.

Lemma 3.2: The ith smallest eigenvalue of L_{sym} is $(1 - \lambda_i)$ where λ_i is the ith largest eigenvalue of W_{sym}.

Proof: From the definition of eigendecomposition, we know that:

$$L_{sym}\mathbf{v} = \lambda \mathbf{v}.$$

We can rearrange the terms into the following form:

$$(L_{sym} - \lambda I)\mathbf{v} = \mathbf{0}$$

If we replace W_{sym} with L_{sym}, then we have the following relationship between those two:

$$\begin{aligned}(L_{sym} - \lambda I)\mathbf{v} &= \mathbf{0} \\ (I - W_{sym} - \lambda I)\mathbf{v} &= \mathbf{0} \\ (-W_{sym} + I - \lambda I)\mathbf{v} &= \mathbf{0} \\ (W_{sym} - I + \lambda I)\mathbf{v} &= \mathbf{0} \\ (W_{sym} - (1 - \lambda) I)\mathbf{v} &= \mathbf{0} \\ W_{sym}\mathbf{v} &= (1 - \lambda)\mathbf{v}\end{aligned}$$

Since the eigenvalues of W_{sym} are always ≥ 0 and because of the normalization, the eigenvalues lie from 1 to 0, the eigenvalues of L_{sym} range from 0 to 1.[3] ■

We use the first k significant eigenvectors of W_{sym}, which are the k least significant eigenvectors of L_{sym}. Since only eigenvectors are needed for spectral clustering, therefore, the

[3] In practice, the eigenvalues range from -1 to 1. If that happens, we can simply flip the sign of eigenvectors and eigenvalues.

spectral embedding of a normalized Laplacian L_{sym}, can be easily obtained from a SVD on $YD^{-1/2}$, *without constructing the Laplacian explicitly*.

However, in a streaming setting, constructing the exact degree matrix is impossible (without storing all the observed points), therefore we use the streaming degree approximation idea from Equation (2). Let $Y_{[t]} = [Y_1, \ldots, Y_t]$ denote the original input stream. Instead of $Y_{[t]}$, we focus on a modified stream:

$$\tilde{Y}_{[t]} = [\tilde{Y}_1, \ldots, \tilde{Y}_t] \text{ where } \tilde{Y}_i = Y_i \tilde{D}_i^{-1/2} \text{ for } i \in [t],$$

where $\tilde{D}_i$ is the degree approximation for the data points in Y_i. Note that $\tilde{Y}_{[t]}$ can be obtained from $Y_{[t]}$ in a streaming fashion.

B. Streaming Manifold Construction

Manifold Construction (Steps 5-11 in Algorithm StrmEmb). The main remaining challenge now lies in efficiently constructing the spectral embedding of the stream. Given $\tilde{Y}_{[t]}$, the streaming manifold construction is presented in Algorithm StrmEmb Steps 5-11.

We use a matrix sketching based approach for constructing the spectral embedding of $\tilde{Y}_{[t]}$ in a streaming fashion. In his recent paper [27], Liberty introduced an elegant algorithm (called *Frequent Directions*) for matrix sketching. The *Frequent Directions* algorithm operates in a streaming model and constructs a sketch matrix using a (surprisingly) simple idea of "shrinking" a few orthogonal vectors. In the original *Frequent Directions* algorithm setting [27], the input is a matrix $Z \in \mathbf{R}^{p \times d}$. In each step, one row of Z is processed by the algorithm, and the algorithm iteratively updates a matrix $Q \in \mathbf{R}^{\ell \times d}$ ($\ell \ll p$) such that for any unit vector $\mathbf{x} \in \mathbf{R}^d$, $\|Z\mathbf{x}\|^2 - \|Q\mathbf{x}\|^2 \leq 2\|Z\|_F^2/\ell$. Here parameter ℓ adjusts the trade-off between the size of the sketch matrix and the approximation error bounds (larger ℓ increases the computational and storage requirements of the algorithm while giving better results). Recently, Ghashami and Philips [18], reanalyzed the *Frequent Directions* algorithm, to show that it provides relative error bounds for low-rank matrix approximation.

Our approach for constructing the spectral embedding of the stream $\tilde{Y}_{[t]}$ (outlined between Steps 5-9 in Algorithm StrmEmb) is based on extending the *Frequent Directions* algorithm of [27] to a more general setting where in every time step, we add $n_t \gg 1$ new columns.[4] As in *Frequent Directions*, our algorithm requires just one pass over the data stream. In Algorithm StrmEmb, B_t is the maintained sketch of the stream $\tilde{Y}_{[t]}$, and is updated at every time step as new data arrives. The parameter ℓ (as mentioned earlier) defines the size of the sketch matrix and parameter $\kappa (\leq \ell)$ defines the dimension of the embedded data. We discuss the setting of both these parameters later.

Embedding (Re)alignment (Step 9 and 10 in Algorithm StrmSC). Although we have addressed the challenges for constructing the embedding $\breve{V}_t$ for each stream batch in Algorithm StrmEmb, the low rank space basis at time t, $\tilde{U}_{t_{(\kappa)}}$, tend to change over time due to concept-drift. Therefore, we cannot simply use the found embedding $\breve{V}_t$ in clustering. At every time step it is necessary to realign the past facility or micro cluster centroids to the new basis (Steps 9-10 in Algorithm StrmSC). For this, we construct the alignment matrix, R_t, as follows:

$$R_t = \tilde{U}_{t-1_{(\kappa)}}^\top \tilde{U}_{t_{(\kappa)}}.$$

Then we rotate previously found each facility $\mathbf{z}$ by $R_t\mathbf{z}$.

C. Computational Complexity

At any time t, the running time of the Algorithm StrmEmb is $O(\max\{mn_t\ell, m\ell^2\})$ by using power-iteration or rank-revealing QR decomposition for SVD [20]. The space complexity is $O(mn_t + m\ell)$. For a batch spectral clustering algorithm, that at time t uses all the data till time t, the space complexity is $O(mn_{[t]})$ and time complexity is $O(mn_{[t]}k)$. One notices that Algorithm StrmSC is much more efficient than its batch counterpart, as $n_{[t]}$ grows very rapidly.

D. Theoretical Analysis

We now prove the effectiveness of Algorithm StrmEmb, by showing that under reasonable assumptions, the spectral embedding constructed by Algorithm StrmEmb is close to the embedding constructed by utilizing the whole data stream. Due to space constraints, we omit detailed proofs here.

At time t in Algorithm StrmEmb, consider the rank-κ approximation of $C_t = [B_{t-1}, \tilde{Y}_t]$. Let

$$\textsc{Svd}_\kappa(C_t) = \tilde{U}_{t_{(\kappa)}} \tilde{\Sigma}_{t_{(\kappa)}} \tilde{V}_{t_{(\kappa)}}^\top.$$

Algorithm StrmEmb constructs an embedding $\breve{V}_t$ of $\tilde{Y}_t$ (assuming $\tilde{\Sigma}_{t_{(\kappa)}}^{-1}$ exists) defined as,

$$\breve{V}_t^\top = \tilde{\Sigma}_{t_{(\kappa)}}^{-1} \tilde{U}_{t_{(\kappa)}} \tilde{Y}_t.$$

Now B_{t-1} is the sketch of $\tilde{Y}_{[t-1]}$. Consider the rank-κ approximation of $\tilde{Y}_{[t]} = [\tilde{Y}_{[t-1]}, \tilde{Y}_t]$. Let

$$\textsc{Svd}_\kappa(\tilde{Y}_{[t]}) = U_{[t]_{(\kappa)}} \Sigma_{[t]_{(\kappa)}} V_{[t]_{(\kappa)}}^\top.$$

We compare the embedding $\breve{V}_t$ constructed by the algorithm to the embedding V_t constructed using the actual stream $\tilde{Y}_{[t]}$, where V_t is defined as (assuming $\Sigma_{[t]_{(\kappa)}}^{-1}$ exists)

$$V_t^\top = \Sigma_{[t]_{(\kappa)}}^{-1} U_{[t]_{(\kappa)}}^\top \tilde{Y}_t.$$

We claim $\breve{V}_t$ is a good approximation of V_t.

$$\begin{aligned} \|V_t - \breve{V}_t\|_F &= \|V_t^\top - \breve{V}_t^\top\|_F \\ &= \|\Sigma_{[t]_{(\kappa)}}^{-1} U_{[t]_{(\kappa)}}^\top \tilde{Y}_t - \tilde{\Sigma}_{t_{(\kappa)}}^{-1} \tilde{U}_{t_{(\kappa)}}^\top \tilde{Y}_t\|_F \\ &\leq \|\Sigma_{[t]_{(\kappa)}}^{-1} U_{[t]_{(\kappa)}}^\top - \tilde{\Sigma}_{t_{(\kappa)}}^{-1} \tilde{U}_{t_{(\kappa)}}^\top\|_F \|\tilde{Y}_t\|_F. \end{aligned} \quad (3)$$

[4] A similar sketching based low-rank matrix approximation approach was recently used in an entirely different context of feature selection [23].

Let us concentrate on bounding $\|\Sigma^{-1}_{[t]_{(\kappa)}} U^\top_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}} \tilde{U}^\top_{t_{(\kappa)}}\|_F$.

$$\begin{aligned}
&\|\Sigma^{-1}_{[t]_{(\kappa)}} U^\top_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}} \tilde{U}^\top_{t_{(\kappa)}}\|_F \\
&= \|U_{[t]_{(\kappa)}} \Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{U}_{t_{(\kappa)}} \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_F \\
&= \|U^\top_{[t]_{(\kappa)}} \Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{U}_{t_{(\kappa)}} \Sigma^{-1}_{[t]_{(\kappa)}} + \tilde{U}_{t_{(\kappa)}} \Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{U}_{t_{(\kappa)}} \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_F \\
&\leq \|U_{[t]_{(\kappa)}} \Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{U}_{t_{(\kappa)}} \Sigma^{-1}_{[t]_{(\kappa)}}\|_F + \|\tilde{U}_{t_{(\kappa)}} \Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{U}_{t_{(\kappa)}} \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_F \\
&\leq \|U_{[t]_{(\kappa)}} - \tilde{U}_{t_{(\kappa)}}\|_F \|\Sigma^{-1}_{[t]_{(\kappa)}}\| + \|\Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_F \\
&\leq \|U_{[t]_{(\kappa)}} - \tilde{U}_{t_{(\kappa)}}\|_F \|\Sigma^{-1}_{[t]_{(\kappa)}}\| + \sqrt{\kappa} \|\Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t}\|_\infty. \qquad (4)
\end{aligned}$$

The second inequality uses the fact that the $\Sigma^{-1}_{[t]_{(\kappa)}}$ and $\tilde{\Sigma}^{-1}_{t_{(\kappa)}}$ are diagonal matrices, and the columns of $\tilde{U}_{t_{(\kappa)}}$ are orthonormal. For the last inequality we used the fact that since $\Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}$ is a diagonal matrix, its Frobenius norm is at most the square root of its dimension times the largest diagonal value.

So a bound on $\|V_t - \breve{V}_t\|_F$ follows from respective bounds on $\|U_{[t]_{(\kappa)}} - \tilde{U}_{t_{(\kappa)}}\|_F$ and $\|\Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty$. For the former, we adapt a recent analysis of Huang *et al.* [23], who analyzed similar bounds in a different context of using matrix sketching for feature selection.

Proposition 3.3 (Modified from [23]): Let λ_i denote the i^{th} eigenvalue of $\tilde{Y}_{[t]}\tilde{Y}^\top_{[t]}$. Additionally, assume $L = \min_{i \neq j} |\lambda_i - \lambda_j| > 0$ and ℓ satisfies

$$\ell = \Omega\left(\frac{\sqrt{m}\|\tilde{Y}_{[t]}\|^2 \|\tilde{Y}_{[t]} - \tilde{Y}_{[t]_{(\kappa)}}\|^2_F}{L^2}\right),$$

where $\tilde{Y}_{[t]_{(\kappa)}}$ is the rank-κ approximation of $\tilde{Y}_{[t]}$. Then

$$\|U_{t_{(\kappa)}} - \tilde{U}_{t_{(\kappa)}}\|_F \leq \frac{\sqrt{2}L}{\sqrt{L + 8\|\tilde{Y}_{[t]}\|^2}\sqrt[4]{L^2 + 16\|\tilde{Y}_{[t]}\|^4}}. \qquad (5)$$

Note that the right hand side of (5) decreases as L increases. In practice, we noticed that this is quite small. The assumption of $L > 0$ is commonly satisfied in practice, especially if m is reasonably smaller than the number of datapoints in $\tilde{Y}_{[t]}$. Also practically, we noticed that setting $\ell \approx \sqrt{m}$ suffices to get good results.

To bound $\|\Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty$, note that

$$\|\Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty \leq \|\Sigma^{-1}_{[t]_{(\kappa)}} - \breve{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty + \|\breve{\Sigma}^{-1}_{t_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty.$$

Let σ_{t_i}, $\tilde{\sigma}_{t_i}$, and $\breve{\sigma}_{t_i}$ be the ith singular value of $\tilde{Y}_{[t]}$, C_t, and B_t respectively. This implies that

$$\begin{aligned}
\Sigma^{-1}_{[t]_{(\kappa)}} &= \mathrm{d}iag(1/\sigma_{t_1}, \ldots, 1/\sigma_{t_\kappa}), \\
\tilde{\Sigma}^{-1}_{t_{(\kappa)}} &= \mathrm{d}iag(1/\tilde{\sigma}_{t_1}, \ldots, 1/\tilde{\sigma}_{t_\kappa}), \text{ and} \\
\breve{\Sigma}^{-1}_{t_{(\kappa)}} &= \mathrm{d}iag(1/\breve{\sigma}_{t_1}, \ldots, 1/\breve{\sigma}_{t_\kappa}).
\end{aligned}$$

The following proposition bounds $\|\Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty$ by bounding $\|\Sigma^{-1}_{[t]_{(\kappa)}} - \breve{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty$ and $\|\breve{\Sigma}^{-1}_{t_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty$ separately.

Proposition 3.4:

$$\|\Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty \leq \frac{\|\tilde{Y}_{[t]} - \tilde{Y}_{[t]_{(\kappa)}}\|_F}{\breve{\sigma}^2_{t_\kappa}} \sqrt{\frac{\kappa}{\ell - \kappa}} + \max_{i \in [\kappa]} \left| \frac{1}{\tilde{\sigma}_{t_i}} - \frac{1}{\sqrt{\tilde{\sigma}^2_{t_i} - \tilde{\sigma}^2_{t_\ell}}} \right|. \qquad (6)$$

Proof: Note that $\|\Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty$ can be bound as

$$\|\Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty \leq \|\Sigma^{-1}_{[t]_{(\kappa)}} - \breve{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty + \|\breve{\Sigma}^{-1}_{t_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty.$$

By construction in Algorithm StrmEmb,

$$\begin{aligned}
\|\breve{\Sigma}^{-1}_{t_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty &= \max_{i \in [\kappa]} \left| \frac{1}{\tilde{\sigma}_{t_i}} - \frac{1}{\breve{\sigma}_{t_i}} \right| \\
&= \max_{i \in [\kappa]} \left| \frac{1}{\tilde{\sigma}_{t_i}} - \frac{1}{\sqrt{\tilde{\sigma}^2_{t_i} - \tilde{\sigma}^2_{t_\ell}}} \right|.
\end{aligned}$$

To bound $\|\Sigma^{-1}_{[t]_{(\kappa)}} - \breve{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty$, we first establish that $\tilde{Y}_{[t]}Y^\top_{[t]} \succeq B_t B_t^\top$ following the analysis in [27]. This implies that for all $i \in [\kappa]$, $\sigma_{t_i} \geq \breve{\sigma}_{t_i}$. By Weyl's inequality [19], $\forall i \in [\kappa], \sigma^2_{t_i} - \breve{\sigma}^2_{t_i} \leq \|\tilde{Y}_{[t]}\tilde{Y}^\top_{[t]} - B_t B_t^\top\|$ (as $\sigma^2_{t_i}$ and $\breve{\sigma}^2_{t_i}$ are the eigenvalues of $\tilde{Y}_{[t]}\tilde{Y}^\top_{[t]}$ and $B_t B_t^\top$ respectively). Now

$$\begin{aligned}
\|\Sigma^{-1}_{[t]_{(\kappa)}} - \breve{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty &= \max_{i \in [\kappa]} \left| \frac{1}{\sigma_{t_i}} - \frac{1}{\breve{\sigma}_{t_i}} \right| \\
&\leq \frac{\max_{i \in [\kappa]} |\sigma_{t_i} - \breve{\sigma}_{t_i}|}{\breve{\sigma}^2_{t_\kappa}} \\
&\leq \frac{\|\tilde{Y}_{[t]}\tilde{Y}^\top_{[t]} - B_t B_t^\top\|}{\breve{\sigma}^2_{t_\kappa}}.
\end{aligned}$$

By using an analysis from [23], we can show that

$$\|\tilde{Y}_{[t]}\tilde{Y}^\top_{[t]} - B_t B_t^\top\| \leq \|\tilde{Y}_{[t]} - \tilde{Y}_{[t]_{(\kappa)}}\|_F \sqrt{\frac{\kappa}{\ell - \kappa}}.$$

Putting everything together, we get

$$\|\Sigma^{-1}_{[t]_{(\kappa)}} - \tilde{\Sigma}^{-1}_{t_{(\kappa)}}\|_\infty \leq \frac{\|\tilde{Y}_{[t]} - \tilde{Y}_{[t]_{(\kappa)}}\|_F}{\breve{\sigma}^2_{t_\kappa}} \sqrt{\frac{\kappa}{\ell - \kappa}} + \max_{i \in [\kappa]} \left| \frac{1}{\tilde{\sigma}_{t_i}} - \frac{1}{\sqrt{\tilde{\sigma}^2_{t_i} - \tilde{\sigma}^2_{t_\ell}}} \right|.$$

■

Note that for the right hand side of (6), both the terms decrease with increasing ℓ (for the second term, larger ℓ leads to larger $1/\sqrt{\tilde{\sigma}^2_{t_i} - \tilde{\sigma}^2_{t_\ell}}$).

Putting it all Together. The following theorem follows by combining Propositions 3.3 and 3.4 with (3) (4).

Theorem 3.5: With the notation defined above and under the conditions of Proposition 3.3, at every time t, the embedding $\breve{V}_t$ constructed by Algorithm StrmEmb satisfies,

$$\|V_t - \breve{V}_t\|_F \leq \|\tilde{Y}_t\|_F \Bigg(\frac{1}{\sigma_{t_\kappa}} \frac{\sqrt{2}L}{\sqrt{L + 8\|\tilde{Y}_{[t]}\|^2} \sqrt[4]{L^2 + 16\|\tilde{Y}_{[t]}\|^4}} + \frac{\kappa \|\tilde{Y}_{[t]} - \tilde{Y}_{[t]_{(\kappa)}}\|_F}{\breve{\sigma}^2_{t_\kappa} \sqrt{\ell - \kappa}} + \sqrt{\kappa} \max_{i \in [\kappa]} \left| \frac{1}{\tilde{\sigma}_{t_i}} - \frac{1}{\sqrt{\tilde{\sigma}^2_{t_i} - \tilde{\sigma}^2_{t_\ell}}} \right| \Bigg).$$

The above theorem shows that, under reasonable practical assumptions and setting of ℓ, at every time t, the embedding $\breve{V}_t$ constructed by the Algorithm StrmEmb is close to the embedding V_t constructed using the entire data stream $\tilde{Y}_{[t]}$.

IV. Discussions

We now justify the utility of our proposed Algorithm StrmSC by briefly comparing with a few relevant methods. For a detailed discussion about clustering on data streams, please refer to the surveys of [7], [17], [37], [2].

Streaming Clustering on Full Feature Space. The basic idea of streaming clustering is to create/update micro-clusters while assigning new coming data stream to them, and to finally use the summary statistics to produce the final macro-clusters. The assignment is usually based on neighborhood search on the full feature space (such as in the classical K-means/medians). Guha *et al.* [22], [21] presented a divide-and-conquer algorithm for stream clustering. O'Callaghan *et al.* [36] proposed a method based on local search. Charikar *et al.* [12] proposed a randomized algorithm for the K-median problem which guarantees a constant factor approximation to the objective function with one pass and small amount of storage. An efficient streaming text clustering was presented by Zhong *et al.* [50] based on constructing an online update for the spherical K-means.

Another class is based on accelerating the neighborhood search by using efficient data structures. Zhang *et al.* [49] built a hierarchical data structure to incrementally cluster the incoming data streams. Kranen *et al.* [24] used a compact and self-adaptive index structure for maintaining stream summaries. Ackermann *et al.* [1] constructed a small weighted sample of the data stream by using a new data structure called *coreset tree*. Shindler *et al.* [41] recently proposed an effective and efficient algorithm (Algorithm StrmKMeans) for stream clustering using ideas from online facility location literature. Their algorithm compares favorably to existing algorithms both theoretically and experimentally. Very recently, Liberty *et al.* [28] proposed an online K-means clustering algorithm with space and time requirement that is only poly-logarithmic in the length of the stream. All the above techniques operate on the entire feature space. However, when the dataset is high-dimensional, these techniques which rely on neighborhood search face stability issues and could perform poorly, this is commonly referred to as the *curse of dimensionality*. Comparably, our method uses a low-dimensional embedding derived from singular value decomposition of the data stream. Therefore, it has the similar stability and performance advantages that batch spectral clustering enjoys over the traditional K-means approach (and its variants) operated on the full feature space. We present an experimental comparison in Section V.

Streaming Clustering on Projected Space. There are quite a few works on clustering data streams after projection on a lower dimensional space. Aggarwal *et al.* [3], [5] proposed a projection-based clustering approach that maintains statistical information of micro-clusters of data streams, which are temporal extensions of cluster feature vectors. These ideas were further refined in [4]. Projected clustering has also been combined with the density-based algorithms [15], [8], [31], [11], [13], and with high-dimensional data [35]. The assumption of projected clustering is to have each cluster specific to a particular feature subset that optimizes a quality criterion for that cluster. However, the feature subsets could be quite different across different clusters and the selections are typically based on parameter settings, which leads to unstable performance. For example, the HDDStream algorithm in [35] tries to model clusters as areas of high density (relying on some unknown density-distribution). It summarizes both the data points and the dimensions where these points are grouped together and maintains these summaries as new points arrive over time. However, its performance relies on quite a few parameters specified by the user. Since these parameters directly affect the correlation threshold, consequently the clustering results are sensitive to the choice of the parameters, the relative ordering of the arrival points, and other noise issues. On the contrary, our proposed method is based on matrix sketching, which is a stable compressed statistical representation of the entire (observed) data stream. Therefore, the projection is robust to noise and ordering of the stream. We present an experimental comparison in Section V.

Evolutionary/Incremental Spectral Clustering. In a different input setting (from our problem definition), evolutionary/incremental spectral clustering [14], [33], [34] approaches have been proposed, where typically the assumption is that all or most of the points of a graph are already known and the edges are added incrementally. While these approaches work well on network datasets such as blogs, these methods need to maintain a n^2 graph at any time. It is hard to maintain this matrix with limited memory when the number of samples n is large which is usually the case in streaming setting. And to the best of our knowledge there is no known generalization here that operates on a data stream consisting of high-dimensional data points. Therefore, we do not include them in our experimental comparisons.

Efficient Static Clustering. Our proposed algorithm maintains and updates a set of cluster center candidates as new data stream comes. Many efficient approaches have been proposed [39], [46], [44], [6], [48], [29] that try to recover the sparse cluster centers of high-dimensional data. But all of them require access to the whole dataset, so are not suitable for a streaming setup. Therefore, we do not include them also in our following experimental comparisons.

V. Experimental Analysis

In this section, we experimentally test our proposed approach in terms of effectiveness and efficiency. All the experi-

TABLE I. STATISTICS OF THE EXPERIMENTAL DATASETS.

	Dataset	#instances	#features	#clusters
1	sector	9,619	55,197	105
2	ohscal	11,162	11,465	10
3	20NG	18,707	29,476	20
4	RCV1	193,844	47,236	103
5	MNIST	70,000	784	10
6	Tiny	1,000,000	3,072	75,062
7	SenVec (SensIT Vehicle)	98,528	100	3
8	jester	73,421	100	86

ments were run on an Intel(R) Xeon(R) CPU X5650 2.67GHz processor with 128GB memory.

A. Experiment Setup

Datasets. We conducted our experiments on four text datasets (sector, ohscal, 20Newsgroup and RCV1), two image datasets (MNIST and Tiny), one sensor network dataset (SenVec), and one collaborative filtering rating dataset (jester), all summarized in Table I. For the first four text datasets we use the cosine kernel affinity matrix, and Gaussian kernel affinity matrix is used for the remaining four. The four text datasets are L_2 normalized for each document.

The sector dataset consists of company web pages classified in a hierarchy of industry sectors. The ohscal dataset was from the OHSUMED collection which contains documents from medical categories. 20NG (20Newsgroup) is a balanced dataset that covers 20 news topics. RCV1 dataset contains an archive of manually categorized newswire stories made available by Reuters [26]. We used a subset of RCV1 dataset, only choosing those documents with exactly one label. MNIST dataset has 10 classes of handwritten digits and 784 image features. All the above datasets can be found either in [10] or [16]. Tiny is a large web-image collection for non-parametric object and scene recognition (downloaded from [45]). From the 80 million tiny dataset images, we created a dataset of million images consisting of $60,000$ labeled images that cover 100 classes (from [25]) and other randomly sampled images. The evaluation here is restricted to the labeled $60,000$ images. We used the Tiny images with all the 3072 raw features, which are 32×32 color images in 3 color channels (RGB), to demonstrate the clustering performance on images with a high-dimensional feature vector. SenVec (SensIT Vehicle) dataset contains distributed sensor network data for vehicle classification. Jester dataset contains anonymous ratings data that are used to recommend jokes.

Baselines. We test two versions of our proposed algorithm: **SSC-1**, which uses Algorithm StrmEmb to construct the low-dimensional embedding of the stream but we perform the final K-means clustering after collecting the low-dimensional embedding of the whole stream; and **SSC-2**, which is the same to Algorithm StrmSC (a streaming K-means clustering is used at every time step). Results from SSC-1 demonstrate the quality of the spectral embedding, whereas the SSC-2 results represent the full streaming spectral clustering.

Five other algorithms are chosen for comparison: classical K-means (referred to as **KM**), classical spectral clustering algorithm[5] (referred to as **NJW**), **BIRCH** [49], streaming K-means[6] (referred to as **SKM**), and **HDDStr** [35]. The first two algorithms are the most widely used solution for batch clustering. BIRCH builds a hierarchical data structure to cluster the incoming data streams incrementally. SKM as discussed earlier is a recently proposed fast and accurate streaming K-means clustering method. HDDStr is an efficient projected streaming clustering method over high-dimensional data.

Evaluation Metrics and Parameter Settings. We use two popular evaluation metrics: Normalized Mutual Information (**NMI**) and **Purity**. To show the stability of each algorithm, we also report the standard deviation of these two metrics for each algorithm and each dataset.

The number of clusters k in the dataset is assumed to be known *a priori* as in other works [32], [41], [35]. We set the n_t as 1000 (stability test of n_t will be shown in Section V-D), and the dimension of low-dimensional spectral embedding in Algorithm StrmSC (κ) is set to a multiple of k. The size of matrix sketch ℓ is set as $\sqrt{m}$ as analyzed in Section III-D. For the random feature maps using Gaussian kernel, we set bandwidth-scale as 5000 and the projected dimension (d) as 2000. The maximum number of micro-clusters (facilities) ρ in Algorithm StrmKMeans is set to $k \log n$, where n is the number of data samples (assumed to be known *a priori* as in [41]). For BIRCH, SKM, and HDDStr, we follow the suggestions in [49], [41], [35] for their respective parameter settings. For all the compared algorithms, we performed WCSS K-means (minimizing within-cluster sum of squares, with 30 inner loops and 30 outer loops) to obtain stable cluster assignments.

B. General Performance Comparison

Table II shows the general performance of the compared algorithms. For each dataset, we randomly shuffle the data stream order 30 times and report the average NMI (top) and Purity (bottom). We can make the following observations from these results:

(1) Compared with NJW, our proposed SSC-1 reaches over 92% NMI and over 99% purity on average. This confirms that our streaming spectral embedding approximation approaches the quality of the computationally expensive batch version. SSC-1 has a very similar result stability (measures using the standard deviation) as that of NJW, and on average this is much better than KM.

(2) SSC-1 outperforms those algorithms that operate on the full feature space (KM, BIRCH, and SKM) on the high dimensional datasets (the four text datasets and Tiny image dataset).

(3) Not surprisingly, all the streaming algorithms perform worse than the batch K-means (KM). BIRCH, SKM, and HDDStr only get on average around 30% NMI and 60% purity compared with KM. But our proposed SSC-2, which operates in a true streaming setting, attains on average 58% NMI and 97% purity of KM, which is far more effective than the other streaming competitors ($\approx$ two times in NMI and 1.6 times in purity). This demonstrates the effectiveness of SSC-2 over other streaming competitors (BIRCH, SKM, and HDDStr).

[5]For spectral clustering, we use Algorithm SpectralClustering.

[6]For streaming K-means, we use Algorithm StrmKMeans.

(4) The streaming algorithms are also more sensitive and unstable than the batch algorithms (NJW, KM, etc.) under the change of data order. However, by utilizing low-dimensional embeddings, our proposed SSC-2 algorithm achieves stability that is comparable to KM. Compared with either of BIRCH, SKM, or HDDStr, SSC-2 is on average more than three times stable in NMI and about ten times more stable in purity.

C. Order Sensitivity

It is well known that streaming algorithms are sensitive to the ordering of data, or concept drift. To test the performance of our approach in such scenarios, we sorted the data stream by the following two additional conditions, time and topical order. The chronological order reflects the realistic scenario, whereas the topical order is for simulating the most difficult condition for degree approximation because of the big concept drifts between clusters. For the latter, in addition to grouping points belonging to a cluster together, we also try to guarantee that similar topics are next to each other (e.g., in the 20NG dataset, the set of documents from the comp.sys.ibm.pc.hardware topic and comp.sys.mac.hardware topic are placed next to each other).

Figure 2 shows the stability results on the chronological input stream order. For the sector dataset, only a few clusters appear in the very beginning, therefore, the earlier performance is better than the latter. Overall, all the three datasets show quite stable performances.

Figure 3 shows the input stream ordered by topics. Surprisingly, with the sorted topics, the SSC results are even higher than the average performance on Table II. We conjecture that although there are drastic topic changes, the data stream is probably much better represented by the sketch because documents belonging to the same topics are together. This leads to richer spectral embeddings that can capture more information from the stream. In general, both figures show that the results are quite stable for both our proposed SSC-1 and SSC-2 algorithms, and they outperform the other three streaming algorithms both with chronological and topical data ordering.

D. Batch Size Sensitivity

SSC-1 and SSC-2 tests across different batch sizes (n_t) are shown in Figure 4. The experiment setting is the same as those in Table II but with batch sizes in 500, 800, 1000, 1200, 1500, 1800 and 2000. The performance indicates the stable behavior of SSC-1 and SSC-2 with different batch sizes.

E. Scalability Comparison

Figure 5 shows the scalability comparison between the various clustering algorithms on the Tiny dataset (with up-sampling and down-sampling). All the streaming algorithms required just a few minutes to cluster more than a million datapoints. Especially, we observed that our two versions of SSC compare favorably well with other efficient streaming algorithms, while having better effectiveness (Table II). Compared with KM and NJW, our proposed algorithms show superior scalability, while having comparable effectiveness.

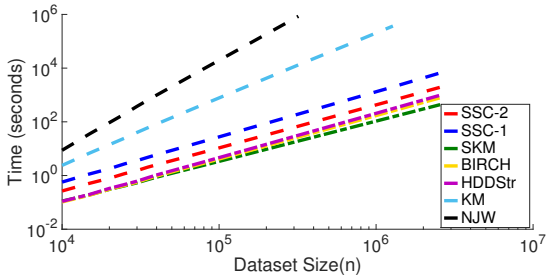


Fig. 5. Scalability experiments. KM and NJW does not scale to large datasets (failing because of their extremely high memory overhead).

VI. Conclusion

We presented a novel approach for streaming spectral clustering. Our algorithm builds a concise approximation to the Laplacian and adapts ideas from matrix sketching to efficiently construct a low-dimensional spectral embedding of the stream. Our algorithm requires only one-pass over the data, utilizes limited storage, adapts well to concept drift and operates in near-real time. The experimental results show that our proposed method can outperform the effectiveness of popular streaming clustering algorithms on high-dimensional data streams while simultaneously achieving good scalability.

References

[1] M. R Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler. Streamkm++: A clustering algorithm for data streams. *ACM JEA*, 2012.

[2] C. C. Aggarwal. A survey of stream clustering algorithms., 2013.

[3] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *VLDB*, 2003.

[4] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for projected clustering of high dimensional data streams. In *VLDB*, 2004.

[5] C. C. Aggarwal and P. S. Yu. A framework for clustering massive text and categorical data streams. In *SIAM SDM*, 2006.

[6] M. Azizyan, A. Singh, and L. Wasserman. Minimax theory for high-dimensional gaussian mixtures with sparse mean separation. In *NIPS*, 2013.

[7] D. Barbará. Requirements for clustering data streams. *ACM SIGKDD Explorations Newsletter*, 2002.

[8] C. Bohm, K. Railing, H. P. Kriegel, and P. Kroger. Density connected clustering with local subspace preferences. In *IEEE ICDM*, 2004.

[9] Vladimir Braverman, Adam Meyerson, Rafail Ostrovsky, Alan Roytman, Michael Shindler, and Brian Tagiku. Streaming k-means on well-clusterable data. In *SODA*, pages 26–40, 2011.

[10] D. Cai. Matlab codes and datasets for feature learning. http://www.cad.zju.edu.cn/home/dengcai/Data/data.html.

[11] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *SIAM SDM*, 2006.

[12] M. Charikar, L. O'Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *ACM STOC*, 2003.

[13] Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *ACM SIGKDD*, 2007.

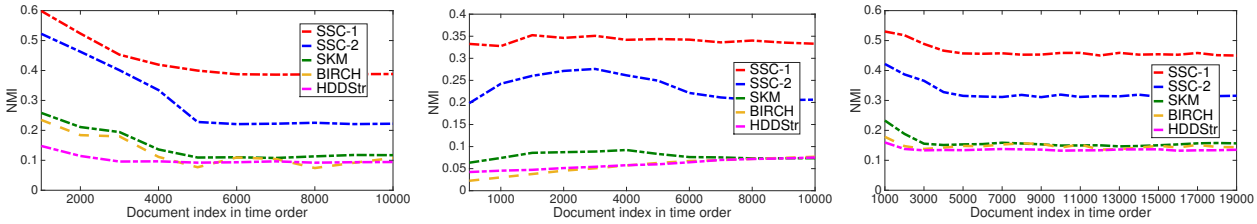
[14] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng. Evolutionary spectral clustering by incorporating temporal smoothness. In *ACM SIGKDD*, 2007.

[15] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *ACM SIGKDD*, 1996.

[16] R. E. Fan and C. J. Lin. Libsvm data: Classification, regression, and multi-label. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/.

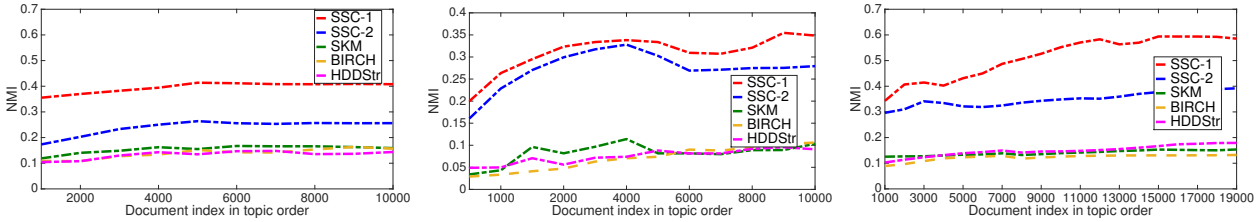
TABLE II. PERFORMANCE COMPARISON IN NMI (TOP) AND PURITY (BOTTOM). THE NUMBERS IN PARENTHESES ARE STANDARD DEVIATION FOR THE RESPECTIVE METRIC. WE COULDN'T RUN NJW ON THE RCV1 AND THE TINY DATASETS DUE TO RUNNING OUT OF MEMORY. THE NUMBERS FOR NJW ON THE RCV1 DATASET ARE FROM [43].

	NJW	SSC-1	KM	BIRCH	SKM	HDDStr	SSC-2
sector	0.4039(0.003)	0.3961(0.003)	0.3554(0.010)	0.0872(0.013)	0.0921(0.011)	0.0821(0.037)	0.2361(0.008)
	0.9794(0.001)	0.9790(0.001)	0.9576(0.005)	0.5372(0.006)	0.6023(0.006)	0.6117(0.019)	0.8616(0.002)
ohscal	0.3408(0.018)	0.3217(0.016)	0.3041(0.022)	0.0876(0.048)	0.1060(0.027)	0.0781(0.052)	0.2273(0.018)
	0.8897(0.004)	0.8833(0.005)	0.8299(0.007)	0.1902(0.174)	0.2405(0.058)	0.3193(0.181)	0.8584(0.009)
20NG	0.4836(0.008)	0.4699(0.009)	0.4686(0.016)	0.1342(0.037)	0.1601(0.041)	0.1232(0.067)	0.3512(0.010)
	0.9643(0.002)	0.9532(0.004)	0.8976(0.007)	0.6723(0.046)	0.6402(0.048)	0.7011(0.081)	0.9412(0.002)
RCV1	[43] 0.2875(0.001)	0.2733(0.003)	0.2577(0.009)	0.1332(0.014)	0.1222(0.012)	0.1065(0.037)	0.1733(0.005)
	—(—)	0.9432(0.001)	0.8721(0.006)	0.5011(0.009)	0.5274(0.009)	0.5567(0.014)	0.7657(0.003)
MNIST	0.4965(0.011)	0.3792(0.004)	0.4873(0.022)	0.1272(0.053)	0.1289(0.079)	0.1431(0.102)	0.1953(0.003)
	0.8868(0.003)	0.8788(0.001)	0.8809(0.006)	0.4292(0.197)	0.4059(0.206)	0.4566(0.258)	0.8587(0.002)
jester	0.5099(0.003)	0.4723(0.005)	0.5322(0.002)	0.1482(0.003)	0.1501(0.002)	0.1438(0.006)	0.2601(0.012)
	0.9444(0.001)	0.9442(0.001)	0.9446(0.001)	0.7428(0.009)	0.7311(0.009)	0.6857(0.019)	0.9296(0.007)
SenVec	0.3007(0.001)	0.2810(0.001)	0.2831(0.016)	0.0989(0.071)	0.0975(0.067)	0.0864(0.097)	0.1221(0.021)
	0.6769(0.001)	0.6921(0.001)	0.6944(0.006)	0.4639(0.118)	0.4606(0.102)	0.5367(0.143)	0.6261(0.062)
Tiny	—(—)	0.2001(0.001)	0.1744(0.001)	0.0817(0.003)	0.0721(0.005)	0.0538(0.012)	0.1703(0.001)
	—(—)	0.9569(0.002)	0.8793(0.003)	0.7011(0.003)	0.7314(0.003)	0.6593(0.009)	0.9182(0.001)



(a) sector (b) ohscal (c) 20NG

Fig. 2. Concept drift across time. Both of our SSC-1 and SSC-2 outperform the other three streaming methods.



(a) sector (b) ohscal (c) 20NG

Fig. 3. Concept drift across sorted topics. Both of our SSC-1 and SSC-2 outperform the other three streaming methods.

[17] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *ACM Sigmod Record*, 2005.

[18] M. Ghashami and J. M. Phillips. Relative errors for deterministic low-rank matrix approximations. In *SODA*, pages 707–717, 2014.

[19] G. H. Golub, M. Heath, and G. Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

[20] G. H. Golub and C. F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[21] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE TKDE*, 2003.

[22] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Foundations of computer science*. IEEE, 2000.

[23] H. Huang, S. Yoo, and S. Kasiviswanathan. Unsupervised feature selection on data streams. In *CIKM*, 2015.

[24] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. The clustree: indexing micro-clusters for anytime stream mining. *Knowledge and information systems*, 2011.

[25] A. Krizhevsky, V. Nair, and G. Hinton. The cifar-100 dataset. http://www.cs.toronto.edu/~kriz/cifar.html.

[26] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *ACM SIGKDD Explorations Newsletter*, 5:361–397, 2004.

[27] E. Liberty. Simple and deterministic matrix sketching. In *ACM SIGKDD*, pages 581–588, 2013.

[28] E. Liberty, R. Sriharsha, and M. Sviridenko. An algorithm for online k-means clustering. *arXiv preprint*, 2014.

[29] J. Liu, C. Wang, M. Danilevsky, and J. Han. Large-scale spectral clustering on graphs. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1486–1492. AAAI Press, 2013.

[30] C. Maung and H. Schweitzer. Pass-efficient unsupervised feature selection. In *NIPS*, 2013.

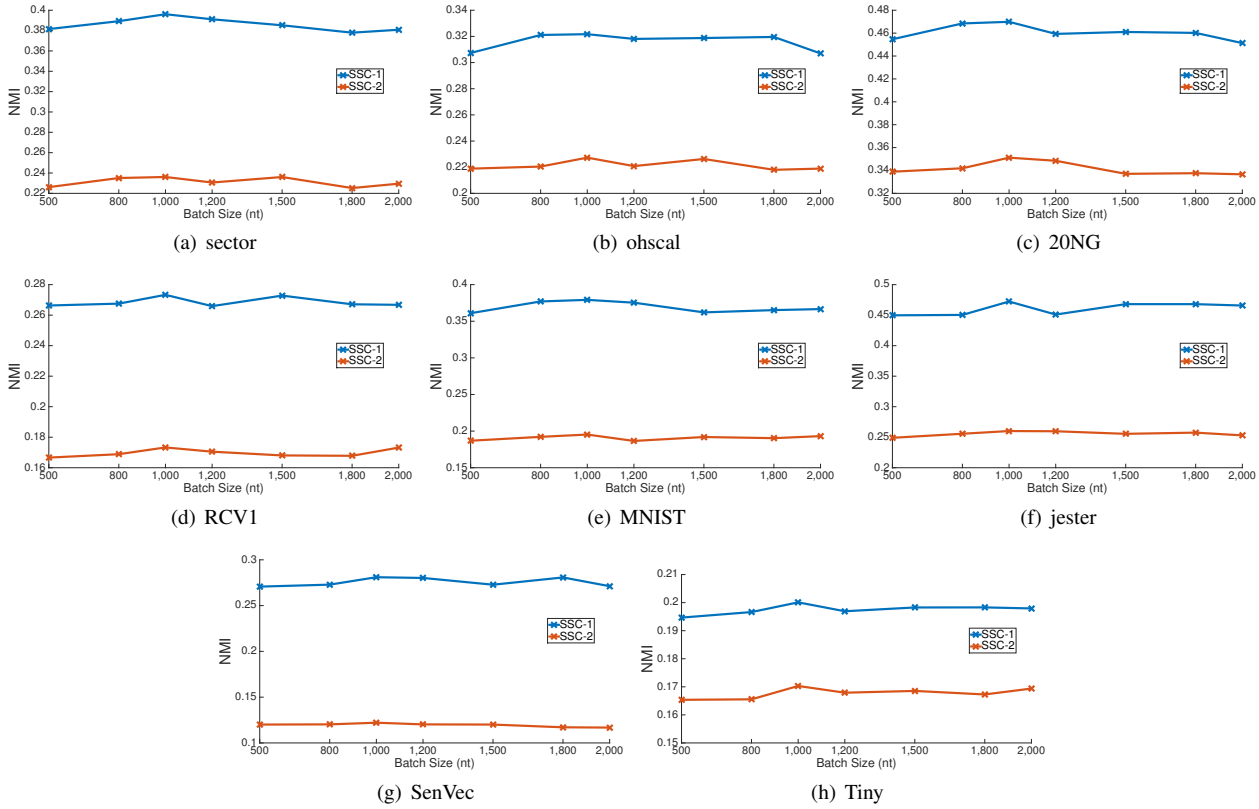
[31] O. Nasraoui and C. Rojas. Robust clustering for tracking noisy evolving data streams. In *SDM*, 2006.

[32] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856, 2002.

[33] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang. Incremental spectral clustering with application to monitoring of evolving blog communities. In *SIAM SDM*, 2007.

[34] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang. Incremental spectral clustering by efficiently updating the eigen-system. *Pattern Recognition*, 43(1):113–127, 2010.

[35] I. Ntoutsi, A. Zimek, T. Palpanas, P. Kröger, and H. P. Kriegel. Density-



(a) sector (b) ohscal (c) 20NG

(d) RCV1 (e) MNIST (f) jester

(g) SenVec (h) Tiny

Fig. 4. Stability of our approaches with different batch sizes. Both SSC-1 and SSC-2 show stable performance with changing batch sizes.

based projected clustering over high dimensional data streams. In *SIAM SDM*, 2012.

[36] L. O'callaghan, A. Meyerson, R. Motwani, N. Mishra, and S. Guha. Streaming-data algorithms for high-quality clustering. In *IEEE ICDE*, 2002.

[37] M. E. Orlowska, X. Sun, and X. Li. Can exclusive clustering on streaming data be achieved? *ACM SIGKDD Explorations Newsletter*, 2006.

[38] A. Rahimi and B. Recht. Random features for large-scale kernel machines. *NIPS*, 2007.

[39] D. Sculley. Web-scale k-means clustering. In *ACM WWW*, 2010.

[40] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE TPAMI*, 22(8):888–905, 2000.

[41] M. Shindler, A. Wong, and A. W. Meyerson. Fast and accurate k-means for large datasets. In *NIPS*, 2011.

[42] Q. Song, J. Ni, and G. Wang. A fast clustering-based feature subset selection algorithm for high-dimensional data. *IEEE TKDE*, 2013.

[43] Y. Song, W. Chen, H. Bai, C. Jin, and E. Y. Chang. Parallel spectral clustering. *Machine Learning and Knowledge Discovery in Databases*, 2008.

[44] W. Sun, J. Wang, Y. Fang, et al. Regularized k-means clustering of high-dimensional data and its asymptotic consistency. *Electronic Journal of Statistics*, 2012.

[45] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE TPAMI*, 2008.

[46] J. Wang, J. Wang, Q. Ke, G. Zeng, and S. Li. Fast approximate k-means via cluster closures. In *IEEE CVPR*, 2012.

[47] H. Yang, M. R. Lyu, and I. King. Efficient online learning for multitask feature selection. *ACM TKDD*, 2013.

[48] J. Yi, L. Zhang, J. Wang, R. Jin, and A. Jain. A single-pass algorithm for efficiently recovering sparse cluster centers of high-dimensional data. In *ICML*, 2014.

[49] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *ACM SIGMOD*, 1996.

[50] S. Zhong. Efficient streaming text clustering. *Neural Networks*, 18(5):790–798, 2005.