Stream Clustering: Efficient Kernel-based Approximation using Importance Sampling

Radha Chitta, Rong Jin and Anil K. Jain
Department of Computer Science and Engineering,
Michigan State University,
East Lansing, MI 48824, USA
chittara@msu.edu, rongjin@cse.msu.edu, jain@cse.msu.edu

***Abstract*—Stream clustering methods, which group continuous, temporally ordered dynamic data instances, have been used in a number of applications such as stock market analysis, network analysis, and cosmological analysis. Most of the popular stream clustering algorithms are linear in nature, i.e. they assume that the data is linearly separable in the input space and use measures such as the Euclidean distance to define the inter-point similarity. Though these linear clustering algorithms are efficient, they do no achieve acceptable cluster quality on real-world data. Kernel-based clustering algorithms, which use non-linear similarity measures, yield better cluster quality, but are unsuitable for clustering data streams due to their high running time and memory complexity. We propose an efficient kernel-based clustering algorithm, called the *Approximate Stream Kernel k-means*, which uses importance sampling to sample a subset of the data stream, and clusters the entire stream based on each data point's similarity to the sampled data points in real-time. Every time a data point is sampled, the kernel matrix representing the similarity between the sampled points is updated, and projected to a low dimensional space spanned by its top eigenvectors. The data points are clustered in this low-dimensional space using the *k*-means algorithm. Thus, the Approximate Stream Kernel *k*-means algorithm performs clustering in linear time using kernel-based similarity. We show that only a small number of points need to be sampled from the data stream, and the resulting approximation error is well-bounded. Using several large benchmark data sets to simulate data streams, we demonstrate that the proposed algorithm achieves a significant speedup over other kernel-based clustering algorithms with minimal loss in cluster quality. We also demonstrate the practical applicability of the proposed algorithm by using it to efficiently find trending topics in the Twitter stream data.**

***Keywords*—Stream clustering, Kernel clustering, k-means, Twitter stream**

I. Introduction

In many applications related to stock trading, social networks, and communication networks, large amounts of data are generated continuously at an extremely rapid rate. For example, about 1 terabyte of trade information is generated during each trading session in the New York Stock Exchange. Over $100,000$ tweets are published every 60 seconds by millions of users on Twitter[1]. This data needs to be analyzed in real-time to gain useful insights and make important decisions.

Clustering is an important exploratory technique for grouping and learning about data. Many efficient algorithms have been developed for clustering large data sets [1], [2], [3]. However, stream data introduces some additional challenges to clustering:

(i) As the data is generated continuously and may be unbounded, it is not possible to store all the data in memory. Each data point can be accessed at most once.

(ii) The data is non-stationary, i.e. the characteristics of the stream data can change over time. This necessitates the cluster model (number of clusters, cluster representatives, cluster size and shape, etc.) to evolve dynamically.

Batch clustering algorithms such as *k*-means and kernel *k*-means [4], assume that (i) the entire data to be clustered is available at the time of clustering, and (ii) the input data is drawn from a mixture of known distributions. For these reasons, batch clustering algorithms cannot be directly used to cluster stream data. Stream clustering algorithms generally consist of two stages: (i) an online phase, where the stream data is summarized into prototypes as it arrives, and (ii) an offline phase where these prototypes are used to obtain the clusters. The set of prototypes are dynamically updated to account for the evolution of the clusters in the stream data.

Many stream clustering algorithms assume that the data is linearly separable in the input space and use measures such as the Euclidean distance to define the inter-point similarity. While these "linear" algorithms are efficient, they are not able to identify complex non-linearly separable clusters in real data sets as accurately as kernel-based clustering algorithms, which use non-linear pairwise similarity measures. However, kernel-based clustering algorithms such as kernel *k*-means and spectral clustering have at least quadratic running time complexity, and are ill-suited to data streams [5]. The kernel-based stream clustering algorithms, currently published in the literature, have high running time complexity, cannot perform real-time clustering, and usually require the selection of a large number of parameters (e.g. thresholds on the inter-cluster distance [6]), which are difficult to tune.

In this article, we propose a variant of the kernel *k*-means algorithm, called *approximate stream kernel k-means*, which samples the data points as they arrive, with probability proportional to their "importance" in the stream, measured in terms of the statistical leverage scores [7]. An approximate kernel matrix is constructed, using the sampled points, and used to find the cluster centers. Clustering is performed in a low-dimensional space spanned by the top eigenvectors of the approximate kernel matrix. The running time complexity of the proposed algorithm is linear in N, the number of data points in the stream. We show that only a small subset of points needs to be sampled and stored in memory. As only the sampled points are used to perform clustering, the proposed algorithm is very efficient. We demonstrate empirically using several benchmark data sets that the proposed algorithm can cluster stream data sets at speeds up to 8 MBps with as few as $1,000$ sampled data points.

Unlike other kernel-based stream clustering algorithms,

[1]http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf

Approach	Examples
CF-Trees	Stream, Single-pass *k*-means [8]
Microcluster trees	CluStream, HPStream [8]
Coresets	StreamKM++ [9]
Grids	D-Stream, ODAC [8]
Approximate clustering	Streaming *k*-means approximation [10], Fast streaming *k*-means [11]
Kernel-based	Incremental spectral clustering [12], Adaptive non-linear clustering [6], Streaming kernel *k*-means [13]

TABLE I: Major published approaches to stream clustering

our algorithm performs real-time clustering, and automatically determines the novelty of the data points based on their importance, thereby eliminating the need for tuning complex parameters. In addition, it maintains the long-term history of the data, resulting in cluster quality similar to that of batch kernel *k*-means, while also allowing the decay and re-birth of clusters.

The rest of the article is organized as follows: Section II first describes the kernel *k*-means algorithm which forms the basis of the proposed algorithm, and then discusses some of the linear and kernel-based stream clustering algorithms published in the literature. In Sections III and IV, we describe the proposed algorithm and analyze its complexity. We finally present our empirical analysis in Section V and conclude our study in Section VI.

II. Background

In this section, we first outline the batch kernel *k*-means algorithm, and then describe some of the related work on stream clustering.

A. Kernel k-means

Kernel *k*-means is a non-linear extension of the popular *k*-means algorithm. The key principle behind kernel *k*-means is to project the data to a high-dimensional Reproducing Kernel Hilbert space (RKHS) $\mathcal{H}_\kappa$, using a non-linear function $\phi(\cdot)$, and execute *k*-means on the projected data. Given an input data set $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, $\mathbf{x}_i \in \Re^d$, to be clustered into C clusters, a user-defined non-linear similarity function $\kappa(\cdot,\cdot) : \Re^d \times \Re^d \mapsto \Re$, where $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ is used to define the similarity between data points. The C clusters are obtained by minimizing the sum-of-squared-errors in $\mathcal{H}_\kappa$:

$$\min_{U \in \{0,1\}^{C\times N}} \max_{\{c_k(\cdot)\in\mathcal{H}_\kappa\}_{k=1}^C} \sum_{k=1}^{C}\sum_{i=1}^{N} U_{ki}\, ||c_k(\cdot) - \kappa(\mathbf{x}_i,\cdot)||^2_{\mathcal{H}_\kappa}, \quad (1)$$

where $||\cdot||_{\mathcal{H}_\kappa}$ is the functional norm for $\mathcal{H}_\kappa$, $c_k(\cdot)$ represents the k^{th} cluster center in the RKHS, and U represents the $C \times N$ cluster membership matrix, where $U_{ki} = 1$ if $\mathbf{x}_i$ belongs to the k^{th} cluster and 0 otherwise. The optimization problem (1) can be relaxed to the following trace maximization problem [14]:

$$\max_{U \in \{0,1\}^{C\times N}} \text{tr}(\widetilde{U} K \widetilde{U}^\top), \quad (2)$$

where K is the $N \times N$ pairwise similarity matrix, defined by $K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$, and $\widetilde{U} = \text{diag}\,(U\mathbf{1})^{-1/2}\, U$. The number of data points N in the stream can be unbounded, so it is infeasible to compute and store the full $N \times N$ kernel matrix. Due to this reason, it is prohibitive to execute kernel *k*-means on stream data.

B. Stream Clustering

Most stream clustering algorithms summarize the data stream using special data structures, and output a set of cluster representatives. These algorithms differ in the data structures used to summarize the data; common data structures are trees, coresets, and grids (See Table I). Algorithms such as Stream first cluster fixed-size segments of the data, and then cluster the prototypes from each of these clusters to obtain the final clusters. Cluster feature trees [15] and micro-cluster trees were employed to summarize the data in algorithms like single-pass *k*-means, CluStream and HPStream. The StreamKM++ algorithm summarizes the data stream into a set of coresets (a weighted subset of points that approximate the input data), and organizes them into a coreset tree. The clusters are obtained by grouping the coresets in the root node of the coreset tree. Grid-based algorithms such as DStream and DGClust partition the d-dimensional feature space into grid cells, each cell representing a cluster. Approximate clustering algorithms such as streaming *k*-means first choose a subset of the points from the stream, ensuring that the selected points are as distant from each other as possible, and then execute *k*-means on the data subset.

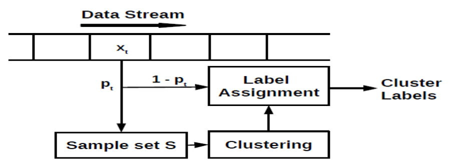


Fig. 1: Schema of the proposed approximate stream kernel *k*-means algorithm

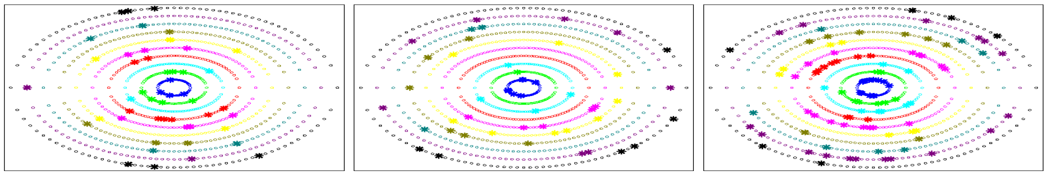
To the best of our knowledge, based on published literature, very few attempts have been made to use non-linear similarity measures for clustering data streams. The incremental spectral clustering algorithm updates the graph Laplacian and its eigenvectors incrementally with each new edge. The clusters are obtained by executing *k*-means on the updated eigenvectors. The adaptive non-linear clustering algorithm partitions the data into segments which are separated from each other by novel data points, identifies the representative segments, and executes spectral clustering on the kernel matrix obtained from the means of the representative segments. Novelty of a data point is determined based on its distance from the mean of the current segment. The streaming kernel *k*-means algorithm (sKKM) divides the data into windows of fixed time-steps, and clusters the data points in every two consecutive windows using weighted kernel *k*-means.

The proposed approximate stream kernel *k*-means algorithm offers the following advantages over the existing algorithms:

- (i) The proposed algorithm uses kernel-based similarity measures, resulting in higher cluster quality than linear clustering algorithms.
- (ii) Clustering is performed using the similarity of the input points with a relatively small number of sampled points. Therefore, the running time complexity of the proposed algorithm is $O(N)$.
- (iii) Unlike the adaptive non-linear clustering algorithm, the novel points are determined automatically using importance sampling.
- (iv) Unlike the streaming kernel *k*-means algorithm, the proposed method uses the complete history of the stream data to compute the clusters, thereby achieving higher cluster quality.
- (v) The proposed method assigns the cluster labels in real-time, and allows the decay and re-birth of clusters, unlike most stream clustering algorithms like Stream, StreamKM++ and single-pass *k*-means.

III. Approximate Stream Kernel *k*-means

Given a stream data set $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots,\}$, $\mathbf{x}_t \in \Re^d$, the objective of the approximate stream kernel *k*-means algorithm (Algorithm 1) is to cluster the data points in real-time using the kernel function $\kappa(\cdot,\cdot) : \Re^d \times \Re^d \mapsto \Re$, that defines the similarity between data points. The key idea behind the



(a) (b) (c)

Fig. 2: Illustration of importance sampling on a two-dimensional synthetic data set containing $1,000$ points along 10 concentric circles (100 points in each cluster). All data points are represented by "o" and the sampled points are represented by "*". Figure (a) shows 50 points sampled using importance sampling, and Figures (b) and (c) show 50 and 100 points selected using Bernoulli sampling, respectively. All the 10 clusters are well-represented by just 50 points sampled using importance sampling. On the other hand, 50 points sampled using Bernoulli distribution are not adequate to represent these 10 clusters (Cluster 4 in red has no representatives). At least 100 points are needed to represent all the clusters.

proposed algorithm is to incrementally sample a subset of the stream data set, and construct a kernel matrix using the sampled points. This approximate kernel matrix is used to obtain the cluster centers. The cluster labels are assigned to the unsampled data points using their kernel similarity with the sampled points. A high level overview of the proposed clustering framework is presented in Figure 1. Our framework consists of three primary components, working in tandem: (i) importance sampling, (ii) clustering, and (iii) cluster label assignment. The sampling component samples the points from the stream and constructs the approximate kernel matrix. The clustering and label assignment components update the clusters and the number of clusters dynamically, and assign cluster labels to all the data points in the stream.

We describe each of these components in the following sections:

A. *Sampling*

One of the obstacles to using kernel k-means for clustering stream data is that it requires the computation of the $N \times N$ kernel matrix. It is infeasible to compute the full kernel matrix for stream data because N is potentially unbounded. The proposed algorithm alleviates this issue by incrementally sampling a subset of the points from the stream and using only this subset to construct the kernel matrix. We maintain a buffer S in memory to store the sampled points; the number of points s in S is constrained by the user-defined parameters m and M ($m \leq s \leq M$). Let K_{t-1} represent the kernel matrix at time $(t-1)$, with $K_1 = 1$. When a data point $\mathbf{x}_t$ arrives at time t, we update the kernel matrix as

$$K_t = \begin{cases} \begin{bmatrix} K_{t-1} & \varphi^\top \\ \varphi & \kappa(\mathbf{x}_t, \mathbf{x}_t) \end{bmatrix} & \text{with probability } p_t, \\ K_{t-1} & \text{with probability } 1 - p_t, \end{cases} \tag{3}$$

where $K_{t-1} = [\kappa(\mathbf{x}_i, \mathbf{x}_j)], \mathbf{x}_i, \mathbf{x}_j \in S$, and $\varphi = (\kappa(\mathbf{x}_t, \mathbf{x}_1), \ldots, \kappa(\mathbf{x}_t, \mathbf{x}_s))^\top$.

The simplest method for sampling data point $\mathbf{x}_t$ is to perform independent Bernoulli trials, i.e. $\mathbf{x}_t$ is stored in S with probability $p_t = \frac{1}{2}$. However, Bernoulli sampling results in a large kernel approximation error and requires a large number of points to be stored in memory[2].

To alleviate this issue, we perform importance sampling instead of Bernoulli sampling. The sampling probability p_t for each point $\mathbf{x}_t$ is based on its "importance", defined in terms of the statistical leverage scores [7]. Let the kernel matrix K_t at time t be decomposed as $K_t \simeq V_C \Sigma_C V_C^\top$, where C represents the number of active clusters[3] at time t, $\Sigma_C = \text{diag}(\lambda_1, \ldots, \lambda_C)$ contains the highest C eigenvalues of K_t and $V_C = (\mathbf{v}_1, \ldots, \mathbf{v}_C)$ contains the corresponding eigenvectors. The probability of adding point $\mathbf{x}_t$ to S is defined by

$$p_t = \frac{1}{C} \left|\left| V_C^{(t)} \right|\right|_2^2, \tag{4}$$

where $V_C^{(j)}$ is the j^{th} row of V_C. Statistical leverage scores measure the influence of each data point in the approximation of the kernel matrix [16]. The subset of data corresponding to the largest statistical leverage values are the most informative, and can represent the distribution of the entire data. By using importance sampling, we obtain a good approximation of the true kernel by sampling just a fraction of the data set (about $s = \Omega(C \ln C)$ samples [17]). Figures 2(a)-(c) illustrate the advantage of importance sampling over Bernoulli sampling on a two-dimensional data set containing $1,000$ points from 10 clusters. Each true cluster is a concentric circle with varying radius, as shown in Figure 2(a). Figure 2(a) also shows 50 points sampled using importance sampling. We observe that all the 10 clusters are adequately represented by the 50 sampled points. Figure 2(b) shows that 50 points sampled from the data using Bernoulli sampling do not represent all the clusters, as the probability of sampling data points from all the clusters is low. All the clusters are represented only when 100 points are sampled, as shown in Figure 2(c).

B. *Clustering*

Let s be the number of points in the buffer S and C be the number of active clusters[3] at time t. After the kernel matrix K_t is constructed in accordance with (3), the data points in S can be partitioned into C clusters by solving the kernel k-means problem

$$\max_{U \in \{0,1\}^{C \times s}} \text{tr}(\widetilde{U} K_t \widetilde{U}^\top). \tag{5}$$

The running time complexity of this step would be $O(s^2)$. We further reduce this complexity by constraining the cluster centers to a smaller subspace, spanning the top C eigenvectors of the kernel matrix K_t, along the lines of the spectral clustering algorithm. We pose the clustering problem as the following optimization problem:

$$\min_{U \in \{0,1\}^{C \times s}} \max_{\{c_k(\cdot) \in \mathcal{H}_a\}_{k=1}^C} \sum_{k=1}^{C} \sum_{i=1}^{s} \frac{U_{ki}}{s} \left|\left| c_k(\cdot) - \kappa(\mathbf{x}_i, \cdot) \right|\right|_{\mathcal{H}_\kappa}^2, \tag{6}$$

where $\mathcal{H}_a = \text{span}(\mathbf{v}_1, \ldots, \mathbf{v}_C)$. The cluster centers can be expressed as linear combinations of the eigenvectors of the kernel matrix:

$$c_k(\cdot) = \sum_{i=1}^{s} \sum_{j=1}^{C} \frac{U_{ki}}{N_k} \sqrt{\lambda_j} v_{ij} = \frac{\mathbf{u}_k}{N_k} V_C \Sigma_C^{1/2}, \; k \in [C], \tag{7}$$

where N_k is the number of points in the k^{th} cluster, and $\mathbf{u}_k = (U_{k1}, \ldots, U_{ks})$.

By substituting (7) in (6), we obtain the following trace maximization problem:

$$\max_{U \in \{0,1\}^{C \times s}} \text{tr}(\widetilde{U} V_C \Sigma_C V_C^\top \widetilde{U}^\top). \tag{8}$$

The above problem can be solved efficiently by executing k-means on the matrix $V_C \Sigma_C^{1/2}$. The following lemma shows

[2]We demonstrate this using a synthetic data set in Figure 2 and using four large benchmark data sets in Section V.

[3]We refer to the set of clusters that the data points in the buffer S belong to at time t as the set of active clusters.

that the error incurred due to the approximation (6) is well-bounded, provided that the tail of the eigenspectrum is fast decaying, which is true for most real data sets:

Lemma 1: Let E and E_a represent the optimal clustering errors in (5) and (8), respectively. We have

$$|E - E_a| \leq \sum_{i=C+1}^{s} \lambda_i.$$

The proof of this lemma is omitted due to space constraints.

We note that the eigenvalues and eigenvectors do not need to be re-computed for clustering, as they were already computed while calculating the leverage scores. This eliminates the need for computing and storing the kernel matrix K_t, as only its top eigenvalues and the corresponding eigenvectors are required for both sampling and clustering. Starting with $V_C = \mathbf{1}$ and $\Sigma_C = 1$, we can update the eigensystem incrementally as the data points arrive. Efficient methods to update the eigenvectors and eigenvalues incrementally are discussed in Section IV.

C. Label Assignment

Data points are assigned cluster labels using the cluster centers obtained from the sampled data points, and the active clusters are updated using a fading cluster mechanism, similar to that used by the adaptive non-linear clustering algorithm [6]. Each cluster k is associated with a timestamp t_k representing the last time a data point was assigned the k^{th} cluster label, and a recency value defined by a monotonic function

$$f_k(t) = \exp\left(-\gamma\left(t - t_k\right)\right), \tag{9}$$

where γ is a user-defined parameter, representing the decay rate of a cluster [18]. A data point $\mathbf{x}_t$ is added to cluster k^* if

$$k^* = \arg\min_{k \in [C]} ||c_k(\cdot) - g_t(\cdot)||^2_{\mathcal{H}_\kappa} \text{ and } f_{k^*}(t) > \eta, \tag{10}$$

where $c_k(\cdot)$ is the cluster center given by (7), $g_t(\cdot)$ is the projection of $\kappa(\mathbf{x}_t, \cdot)$ into the subspace spanned by the eigenvectors V_C, and η is a user-defined lifetime threshold which determines how long a cluster remains active. If the recency $f_{k^*}(t)$ of the closest cluster k^* is less than η, then a new cluster is created with the data point $\mathbf{x}_t$. After the cluster assignment is made, the timestamp and recency of the assigned cluster are updated. Clusters whose recency is less than η (called stale clusters) are deleted and the data points in the buffer that belong to these stale clusters are removed from the buffer.

IV. Implementation and Complexity

The two major operations in the proposed algorithm are: computing the leverage scores, and clustering the top C eigenvectors of the approximate kernel matrix using k-means. Both the operations require the eigenvalues and eigenvectors of the kernel matrix. Let s be the number of points in the sample set S at time t. Eigendecomposition of an $s \times s$ kernel matrix K_t takes $O\left(s^3\right)$ time, if performed naively. However, we can update the eigensystem incrementally using the fast rank-one update mechanism proposed in [19]. Given the eigendecomposition, $K_t = V\Sigma V^\top$, and vector $\varphi \in \Re^s$, this method finds the eigendecomposition of $\left(K_t + \varphi\varphi^\top\right)$ as

$$K_t + \varphi\varphi^\top = \begin{bmatrix} V & \frac{p}{||p||} \end{bmatrix} \Sigma' \begin{bmatrix} V & \frac{p}{||p||} \end{bmatrix}^\top \tag{11}$$

where $p = \left(I - VV^\top\right)\varphi$ is the component of K_t that is orthogonal to V, and Σ' contains the dominant eigenvalues of the sparse matrix

$$\begin{bmatrix} \Sigma & V^\top\varphi \\ \varphi^\top V & ||p|| \end{bmatrix}.$$

Algorithm 1 Approximate Stream Kernel k-means

```
1: Input:
     • $\mathcal{D} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$: the data stream to be clustered
     • $\kappa(\cdot,\cdot) : \Re^d \times \Re^d \mapsto \Re$: kernel function
     • $C$: the initial number of clusters
     • $m$: the initial number of points in the buffer ($m > C$)
     • $M$: maximum number of points allowed in the buffer ($m < M$)
     • $\gamma$: cluster decay rate
     • $\eta$: cluster lifetime threshold
2: Output: Cluster labels for the data points in the stream
3: Initialize $S = \{\mathbf{x}_1\}$, $V_C = \mathbf{1}$ and $\Sigma_C = \kappa(\mathbf{x}_1, \mathbf{x}_1)$.
4: for $t = 1, 2, \ldots, m$ do
5:    Set $S = S \cup \{\mathbf{x}_t\}$.
6:    Update the eigenvalues $\Sigma_C$ and eigenvectors $V_C$ using (11).
7: end for
8: Cluster the data points in $S$ into $C$ clusters by executing $k$-means on
   $V_C {\Sigma_C}^{1/2}$.
9: Set the last update time $t_k = t, k \in [C]$.
10: Evaluate the recency function $f_k(t), k \in [C]$ according to (9).
11: for $t = m+1, m+2, \ldots$ do
12:    Calculate the probability $p_t$ using (4) and set $S = S \cup \{\mathbf{x}_t\}$ with
       probability $p_t$.
13:    If $\mathbf{x}_t$ was added to $S$ in Step 12, update the eigenvalues $\Sigma_C$ and
       eigenvectors $V_C$ using (11), and recluster the points in $S$ by executing
       $k$-means on $V_C {\Sigma_C}^{1/2}$, otherwise find the cluster $k^*$ whose center is
       closest to $\mathbf{x}_t$.
14:    If $f_{k^*}(t) > \eta$, assign $\mathbf{x}_t$ to $k^*$, otherwise create a new cluster with
       $\mathbf{x}_t$ and set $C = C + 1$.
15:    Find clusters whose recency $f_k(t) \leq \eta, k \in [C]$, and remove these
       stale clusters. Set $C = C - c$, where $c$ is the number of stale clusters.
16:    If $\mathbf{card}(S) >= M$, find index $q = \arg\min_l \left|\left|V_C^{(l)}\right|\right|_2^2$ and remove
       data point $\mathbf{x}_q$ from $S$.
17: end for
```

This operation, repeated every time a new data point is input to the system, can be performed in $O(sC + C^3)$ time.

Clustering is performed every time a point is added to the sample set S, which takes $O(sC^2l)$ time, where l is the number of iterations required to reach convergence. In order to reduce the running time, we can employ a *lazy reclustering* approach, by which we perform the clustering after every T data point additions. To further enhance the efficiency of the algorithm, the data points can be processed in batches of size B.

In summary, the time taken by the proposed approximate stream kernel k-means algorithm to cluster a data set of size N is $O\left(NCM + NC^3 + M^2C^2l\right) \sim O\left(NCM\right)$, when $\max(C, M, l) \ll N$. This contrasts with the $O(N^2)$ running time complexity of typical kernel-based clustering.

V. Experimental Results

In this section, we demonstrate that the proposed approximate stream kernel k-means algorithm is more efficient and accurate than the state-of-the-art stream clustering algorithms.

We first use four benchmark data sets (CIFAR-10, Forest Cover Type, Imagenet, and Network Intrusion) to simulate stream data and study the affect of varying the parameters of the algorithm, and the order of the data, on the performance of the proposed algorithm. We use the true class labels of these data sets to evaluate the performance of our algorithm in comparison with the baseline algorithms. We then use the proposed algorithm to cluster the Twitter data stream containing tweets related to programming languages. We find the trend in programming languages over time using the proposed algorithm and compare these topic trends with the true topic trends.

A. Data sets

We demonstrate the effectiveness of the proposed algorithm using the following data sets:

- **CIFAR-10 [20]:** The CIFAR-10 image data set contains $60,000$ unique 32×32 color images from 10 classes. The images are represented by 384 GIST features. We use this medium-sized data set to compare the cluster quality of the proposed algorithm with that of batch kernel k-means.

- **Forest Cover Type [21]:** This data set contains $581,012$ data points, each representing the attributes of a 30×30 square meter cell of the forest floor in the United States. The data, represented using 54 features, belongs to 7 classes, each class representing a different forest cover type.
- **Imagenet [22]:** The Imagenet data set contains about 14 million images organized into a concept-based "synset" hierarchy. We downloaded $1,262,102$ images from 34 classes, and represented them using 900 bag-of-words features, obtained from the SIFT descriptors.
- **Network Intrusion [23]:** The Network Intrusion data set contains $4,897,988$ 50-dimensional data points from 10 classes, representing the TCP dump data from seven weeks of a local-area network traffic.

B. Baselines

We compared the performance of the proposed algorithm with two recent stream clustering algorithms, StreamKM++[9] and sKKM [13], which have been shown to perform better than the other stream clustering algorithms. We show that the proposed approximate stream kernel *k*-means is more effective than these algorithms. Unlike the StreamKM++ algorithm, the proposed algorithm can perform real-time label assignment, and unlike sKKM, it maintains the long-term history of the data, resulting in higher cluster quality. We also compared the performance of the proposed algorithm with (i) the batch *k*-means algorithm to show that our algorithm achieves higher cluster quality, and (ii) the batch kernel *k*-means algorithm to evaluate the loss in the cluster quality. We could execute the kernel *k*-means algorithm only on the medium-sized CIFAR-10 data set. It is prohibitive to compute the full kernel matrix for the remaining data sets, so we executed kernel *k*-means on a $50,000$-sized randomly selected subset of the data, and assigned the remaining points to the closest cluster centers. This gives us an approximation of the time taken to execute kernel *k*-means on the full data set. We finally evaluated the performance of the proposed approximate stream kernel *k*-means algorithm when each data point is sampled with probability $1/2$, and show that importance sampling plays a significant role in reducing the memory requirements and enhancing the cluster quality.

C. Algorithm Parameters

We used the universal RBF kernel for the proposed algorithm and the other kernel-based baseline algorithms on all the data sets. We tuned the kernel width using grid search in the range $[0, 1]$ to obtain best performance. For the proposed approximate stream kernel *k*-means algorithm, we varied the initial sample size from $m = 1,000$ to $m = 5,000$ in multiples of $1,000$, and the maximum buffer size from $M = 5,000$ to $M = 20,000$ in multiples of $5,000$, to constrain the memory used to 4 GB. We employed the lazy reclustering approach with T set to 50 and processed the data in batches of size $B = 10,000$. We set the cluster decay factor $\gamma = 0.5$ as suggested in [6], and varied the lifetime threshold η as $\eta = \exp(-\gamma\tau)$, where $\tau = \{1, 2, \ldots, 5\}$. The coreset size and chunk size parameters for the StreamKM++ and sKKM algorithms were varied from $1,000$ to $5,000$. The initial number of clusters C was set equal to the true number of classes in the data set, for all the algorithms.

We obtained the code for the StreamKM++ algorithm from the authors[4], and implemented the other algorithms in MATLAB[5]. We executed each algorithm 10 times on a 2.8 GHz processor with the memory constrained to 4 GB for the stream clustering algorithms, and to 20 GB for the batch clustering algorithms. We present the mean and variance of the time taken for clustering (in milliseconds) and the cluster quality, measured in terms of the Normalized Mutual Information (NMI), over these 10 runs. Different permutations of the data set were input to the clustering algorithms in each run.

[4]The code for StreamKM++ is available at http://tinyurl.com/streamKM.

[5]The code for the proposed approximate stream kernel *k*-means algorithm is available at http://tinyurl.com/approxStreamKKmeans.

M	Running time			NMI		
	5,000	**10,000**	**15,000**	**5,000**	**10,000**	**15,000**
CIFAR-10	9.34 ($\pm$0.76)	8.50 ($\pm$3.33)	9.57 ($\pm$2.79)	6.22 ($\pm$0.27)	8.07 ($\pm$2.73)	15.49 ($\pm$0.18)
Forest Cover Type	7.07 ($\pm$0.27)	24.17 ($\pm$6.69)	40.65 ($\pm$12.81)	0.56 ($\pm$0.07)	0.72 ($\pm$0.05)	12.19 ($\pm$0.02)
Imagenet	10.57 ($\pm$2.62)	18.77 ($\pm$4.85)	48.15 ($\pm$18.18)	1.58 ($\pm$1.27)	1.73 ($\pm$1.62)	6.55 ($\pm$1.19)
Network Intrusion	12.09 ($\pm$2.57)	27.15 ($\pm$7.07)	43.05 ($\pm$15.31)	13.71 ($\pm$0.01)	13.86 ($\pm$0.40)	13.75 ($\pm$0.30)

TABLE III: Effect of the maximum buffer size M on the running time (in milliseconds) and NMI (in %) of the proposed approximate stream kernel *k*-means algorithm.

τ	**1**	**2**	**3**	**4**	**5**
CIFAR-10	7.48 ($\pm$1.24)	9.28 ($\pm$1.03)	8.33 ($\pm$1.53)	8.54 ($\pm$1.66)	9.08 ($\pm$1.12)
Forest Cover Type	58.55 ($\pm$21.57)	42.80 ($\pm$17.26)	48.78 ($\pm$20.72)	40.09 ($\pm$13.81)	41.88 ($\pm$15.90)
Imagenet	57.91 ($\pm$22.20)	60.25 ($\pm$24.43)	55.77 ($\pm$26.20)	57.24 ($\pm$24.57)	54.98 ($\pm$31.10)
Network Intrusion	161.89 ($\pm$0.69)	164.61 ($\pm$0.70)	165.18 ($\pm$0.71)	162.36 ($\pm$0.68)	163.05 ($\pm$0.64)

TABLE IV: Effect of the cluster lifetime threshold $\eta = \exp(-\gamma\tau)$ on the running time (in milliseconds) of the proposed approximate stream kernel *k*-means algorithm.

D. Results

Clustering efficiency and quality: Clustering time for our algorithm is computed as the average time taken to assign a label to each data point. For the baseline algorithms, we computed this time by dividing the total time taken to cluster the data set by the number of points in the data set. Table II compares the running time and NMI of the proposed algorithm with the baseline algorithms when $m = 5,000$, $M = 20,000$ and $\tau = 1$. As expected, the proposed algorithm was faster than batch kernel *k*-means algorithms and its approximation

τ	**1**	**2**	**3**	**4**	**5**
CIFAR-10	15.49 ($\pm$0.39)	15.55 ($\pm$0.23)	15.41 ($\pm$0.33)	15.45 ($\pm$0.23)	15.50 ($\pm$0.25)
Forest Cover Type	14.27 ($\pm$2.13)	12.10 ($\pm$0.03)	12.11 ($\pm$0.03)	12.10 ($\pm$0.03)	12.10 ($\pm$0.03)
Imagenet	7.04 ($\pm$1.24)	7.04 ($\pm$1.24)	6.95 ($\pm$1.14)	6.95 ($\pm$1.14)	7.76 ($\pm$1.54)
Network Intrusion	14.32 ($\pm$0.10)	13.65 ($\pm$0.06)	13.65 ($\pm$0.06)	13.65 ($\pm$0.06)	13.66 ($\pm$0.06)

TABLE V: Effect of the cluster lifetime threshold $\eta = \exp(-\gamma\tau)$ on the NMI (in %) of the proposed approximate stream kernel *k*-means algorithm.

Data set	**Importance sampling**			**Bernoulli sampling**		
	Running time (ms)	**NMI (%)**	**Number of points sampled**	**Running time (ms)**	**NMI (%)**	**Number of points sampled**
CIFAR-10	7.48 ($\pm$1.24)	15.49 ($\pm$0.39)	5,434 ($\pm2,093$)	2091.50 ($\pm$47.34)	11.33 ($\pm$4.9)	31,483 ($\pm$717)
Forest Cover Type	58.55 ($\pm$21.57)	14.27 ($\pm$2.13)	16,561 ($\pm3,710$)	1257.03 ($\pm$39.33)	3.93 ($\pm$0.7)	407,220 ($\pm5,807$)
Imagenet	57.91 ($\pm$22.20)	7.04 ($\pm$1.24)	14,735 ($\pm1,790$)	3002.45 ($\pm$77.97)	4.97 ($\pm$0.19)	389,177 ($\pm11,325$)
Network Intrusion	161.89 ($\pm$0.69)	14.32 ($\pm$0.10)	14,886 ($\pm2,627$)	923.16 ($\pm$40.41)	6.50 ($\pm$0.15)	1,711,101 ($\pm44,866$)

TABLE VI: Comparison of the performance of the approximate stream kernel *k*-means with importance sampling and Bernoulli sampling.

Data set	**Running time**					**NMI**				
	Approximate stream kernel *k*-means (proposed)	**StreamKM++**	**sKKM**	**Kernel *k*-means (batch)**	***k*-means (batch)**	**Approximate stream kernel *k*-means (proposed)**	**StreamKM++**	**sKKM**	**Kernel *k*-means (batch)**	***k*-means (batch)**
CIFAR-10	7.48 (±1.24)	9.22 (±0.74)	16.86 (±0.79)	12.09 (±0.12)	2.65 (±1.26)	15.40 (±0.39)	8.55 (±0.35)	4.98 (±0.28)	16.98 (±0.02)	10.18 (±0.13)
Forest Cover Type	58.55 (±0.22)	3.82 (±0.32)	54.35 (±1.19)	8.13 (±0.87)	0.05 (±0.02)	14.27 (±2.13)	5.28 (±1.90)	1.70 (±0.37)	7.51 (±1.90)	9.53 (±0.01)
Imagenet	57.91 (±22.20)	18.82 (±0.77)	29.78 (±4.11)	168.32 (±34.52)	32.73 (±9.85)	7.04 (±1.24)	9.66 (±0.22)	8.41 (±0.26)	10.40 (±0.19)	10.01 (±0.01)
Network Intrusion	57.91 (±22.20)	2.00 (±0.06)	42.12 (±0.09)	695.69 (±29.87)	1.82 (±0.57)	14.32 (±0.10)	7.02 (±0.44)	10.32 (±0.64)	7.14 (±1.01)	9.51 (±0.001)

TABLE II: Running time (in milliseconds) and NMI (in %) of the clustering algorithms. It is not feasible to execute kernel *k*-means on the Forest Cover Type, Imagenet, and Network Intrusion data sets due to their large size. The running time of kernel *k*-means on these data sets is obtained by executing kernel *k*-means on a randomly chosen subset of $50,000$ data points to find the cluster centers, and assigning the remaining points to the closest cluster center.

(described in Section V-B) on most of the data sets, but took longer than the *k*-means algorithm because our algorithm has to compute the kernel similarity and its top eigenvectors unlike the *k*-means algorithm. The NMI achieved by our algorithm is higher than that of *k*-means because of the use of non-linear similarity measures. On the CIFAR-10 data set, the batch kernel *k*-means achieved an NMI value of 16.9%. The proposed algorithm achieved comparable NMI values (15.4%).

Compared to the StreamKM++ algorithm, the proposed algorithm achieved better cluster quality although it took longer to assign cluster labels to the points. Our algorithm offers the advantage that the cluster labels can be obtained in real-time, unlike the StreamKM++ algorithm which needs to process all the data points before assigning the cluster labels. For instance, the proposed algorithm was able to cluster about $2,700$ images from the CIFAR-10 data set per second, which is equivalent to a speed of about 8 MBps. On the remaining three data sets, the clustering speed ranged from 30 KBps to 700 KBps. Our algorithm also outperformed the sKKM clustering algorithm in terms of cluster quality. While the sKKM algorithm was slower than the proposed algorithm on the CIFAR-10 data set, it's speed was at par with the proposed algorithm on the remaining data sets. The StreamKM++ algorithm obtains clusters from coresets which summarize *all* the points in the data set. The sKKM algorithm relies on the information from only two time steps and discards most of the historical information. The proposed approximate stream kernel *k*-means algorithm finds the middle ground by retaining potentially useful data points using importance sampling and discarding the rest of the data points. This is reflected in the NMI values achieved by the algorithms.

Figure 3 shows how the NMI values of the proposed algorithm fall due to the accumulation of the kernel approximation error over time. We observe that the reduction in NMI was slow and stabilized over time for all the data sets. The error accumulation can be further minimized by clustering the points in the buffer more frequently (as discussed in Section IV), although this would increase the running time. The user can trade-off between the efficiency and accuracy by tuning the parameters of the algorithm.

Parameter sensitivity: The proposed approximate stream kernel *k*-means algorithm relies on five parameters: initial sample size m, maximum buffer size M, initial number of clusters C, cluster decay rate γ and cluster lifetime threshold η. We study the influence of these parameters on the algorithm's performance and present heuristics to set the parameter values:

- **Initial sample size m:** The time taken by the proposed algorithm to cluster each data point $\mathbf{x}_t$ is influenced by the number of points in the buffer S at time t, because the size of the eigenvector matrix V_C increases proportionally. The buffer size at time t, in turn, depends on the first m data points $\{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ input to the system. More data points were sampled from the stream and added to S, if the initial sample did not contain a sufficient number of representative points. On the CIFAR-10 data set, the number of additional points sampled reduced from $6,087$ to $4,434$ as the initial sample size m was increased from $1,000$ to $5,000$. Similar trends were observed for the remaining data sets as well. Figure 4 compares the running time of the proposed algorithm with the StreamKM++ and sKKM algorithms as the parameter m is varied. As m was increased, the time taken for clustering by the baseline algorithms also increased. As expected, the proposed algorithm took slightly longer than the StreamKM++ and sKKM algorithms for most data sets, especially when m was large. However, the NMI values achieved by the proposed algorithm were much higher than those achieved by the baseline algorithms, as shown in Figure 5. Our algorithm's cluster quality improved significantly as m increased, while there was minimal improvement in the cluster quality of the StreamKM++ algorithm. This improvement in accuracy compensates for the higher running time of the proposed algorithm. These results indicate that the initial sample, determined by the order of the data, plays a crucial role in the performance of the proposed algorithm. The variance in the NMI tends to reduce as m increases, again indicating that the order of the data is important.
- **Maximum buffer size M:** The maximum buffer size M does not affect the efficiency of the proposed algorithm, provided that $M \sim 2m$ and the initial sample is representative of the stream (See Table III). If M is small, data points need to be removed from the buffer S to accommodate for the newly sampled data points, which results in an increased running time. For instance, when $M = 5,000$, about $2,500$ points were removed from S, whereas no points needed to be removed when $M = 20,000$, resulting in a 2 millisecond reduction of the clustering time per data point. The NMI values increased as M increased because a larger number of representative data points could be stored in the buffer.
- **Cluster decay rate γ, lifetime threshold η and number of clusters C:** The final number of clusters at the end of clustering depends on the ordering of the data set, and the cluster decay and lifetime parameters γ and η. For instance, when the points in the CIFAR-10 data set were input in their true order (i.e. all images from class i were input before all images from class j ($i < j$)) for $C = 5$, $\gamma = 0.5$ and $\eta = \exp(-\gamma) = 0.61$, 10 clusters were found. On the other hand, when the data was permuted randomly and clustered, there was no increase in the number of clusters, because no clusters became stale. The number of clusters increased more rapidly when γ and η were set to lower values because the clusters became stale faster. This also influenced the clustering time minimally. The effect of the parameter η on the running time is recorded in Table IV. The NMI is better for lower values

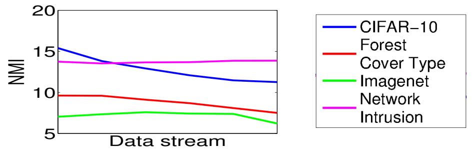


Fig. 3: Change in the NMI (in %) over time for the four data sets.

of γ and η as shown in Table V. These parameters can be selected using prior knowledge of how fast the clusters evolve and change.

Sampling techniques: Table VI shows how the performance of the proposed algorithm on the four data sets changes, when importance sampling is replaced by Bernoulli sampling, where each data point is sampled with probability $1/2$, and no limit is placed on the size of the sample set. We record, for each sampling procedure, the running time in milliseconds, the NMI values and the average number of points stored in memory after all the data points have been clustered. For importance sampling, we set $m = 5,000$. We observe that the number of points sampled using Bernoulli sampling was much higher than that using importance sampling. For instance, about $31,483$ points were sampled from the CIFAR-10 data set when Bernoulli sampling was employed, whereas only about $5,434$ points were sampled using importance sampling. In addition, the cluster quality of Bernoulli sampling was much lower than that of importance sampling. This is because the kernel approximation error is much higher when the data is sampled with equal probability. The running time was also higher when compared to the proposed algorithm with importance sampling due to the large number of sampled points.

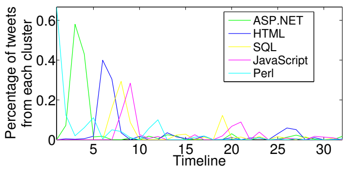
E. Application to Twitter Stream Clustering

Twitter[6] is a popular microblogging social network for sharing information over the web. Users post short messages (called *tweets*), limited to 140 characters, which include user-mentions, links, and emoticons in addition to plain text. Tweets are also often annotated with *hashtags* that denote keywords related to the tweets. A large body of work on topic detection, event detection, hashtag recommendation, and sentiment analysis has been performed on the Twitter data. Clustering has been used to find trending topics in Twitter posts, find user communities based on the similarity of their posts, and automatically annotate tweets with hashtags [24], [25]. In order to demonstrate the practical applicability of the proposed approximate stream kernel k-means algorithm, we used it to cluster the Twitter data, and find the most-tweeted-about technologies over a period of time. We used the Twitter streaming search API to obtain over a billion tweets generated during the month of January 2015, using the following 20 popular keywords as hashtag search queries: Python, Perl, C#, Java, Ruby, C++, JavaScript, VBScript, Scala, Objective C, PHP, SQL, Postgresql, GO, Julia, Erlang, HTML, XML, Swift, and ASP.NET. We filtered out the non-English tweets, removed the hashtags and eliminated stop words to obtain a vocabulary containing $8,042$ terms. We used the corresponding tf-idf (term frequency-inverse document frequency) features and the timestamp of the tweets as features for calculating the kernel

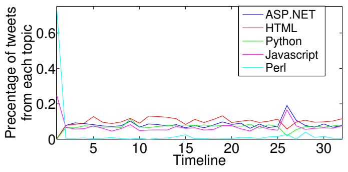
$$\kappa(\mathbf{x}_a, \mathbf{x}_b) = \lambda \exp\left(-\|ts_a - ts_b\|^2\right) + (1-\lambda)\frac{f_a^\top f_b}{\|f_a\|\|f_b\|},$$

where ts_a and f_a denote the timestamp and the tf-idf features of a tweet represented by data point $\mathbf{x}_a$. The first term in the kernel function ensures that two tweets which were generated in the same time period are likely to be assigned to the same cluster, and the second term ensures that two tweets with similar vocabulary are grouped together. We gave equal importance to both the timestamp and the tf-idf features by setting

[6]www.twitter.com



(a) Cluster trends



(b) True trends of the topics

Fig. 7: Trending clusters in Twitter. The horizontal axis represents the timeline in days and the vertical axis represents the percentage ratio of the number of tweets in the cluster to the total number of tweets obtained on the day. Figure (a) shows the trends obtained using the proposed approximate stream kernel k-means algorithm and Figure (b) shows the true trends.

$\lambda = 0.5$. We set the parameters $m = 5,000$, $M = 10,000$, $C = 20$, $\gamma = 0.5$, $\eta = \exp(-\gamma) = 0.6$ and $B = 10,000$. Our algorithm assigned a cluster label to each tweet in about 200 milliseconds. Treating the hashtags as the ground truth labels[7], we obtained an average cluster quality of 61% in terms of NMI. On the other hand, the StreamKM++ algorithm took about 83 milliseconds per tweet and achieved an NMI value of 40%, and the sKKM algorithm took about 2 seconds per tweet and achieved an NMI value of 53%. Figures 6(a) and 6(b) show some sample tweets from the ASP.NET and HTML clusters, respectively. We observed that, by giving equal importance to the timestamp of the tweet, and the words in the tweet, we obtained clusters containing tweets that have both temporal proximity and vocabulary similarity. Retweets were always assigned to the same cluster as the original tweet. For example, both the tweets about *sticky headers* were assigned to the HTML cluster, as seen in Figure 6(b). More recent tweets rather than old tweets were stored in the memory. Figure 7(a) shows the trends of the top five clusters over the month. This coincides well with the true trend of the top topics shown in Figure 7(b). We found that the order of popularity of the topic clusters was ASP.NET, HTML, SQL, JavaScript, Perl, C++, Postgresql, Python, GO, PHP, Swift, Scala, Java, Ruby, C#, XML, Erlang, Julia, Objective C and VBScript; while the true order of topic popularity was ASP.NET, HTML, Python, JavaScript, Perl, Java, PHP, Ruby, SQL, C++, Swift, C#, Scala, Postgresql, XML, Erlang, Julia, GO, Objective C, and VBScript.

VI. Conclusions

We have proposed an efficient and effective real-time stream clustering algorithm called the approximate stream kernel k-means. Experimental results on benchmark data sets show that the proposed algorithm offers a good trade-off between clustering efficiency and cluster quality (in terms of NMI). Further, unlike some state-of-the-art kernel-based stream clustering algorithms, the proposed algorithm can control the decay and birth of clusters, thereby dynamically controlling the final number of clusters. The key to the efficiency of the proposed algorithm is sampling the stream data based on their importance, defined in terms of the statistical leverage scores. This allows us to maintain the long-term history of the stream data and also limit the memory required to store the data. We demonstrated empirically that the proposed algorithm can cluster fast streams such as the Twitter stream with limited

[7]Although hashtags are prone to error, they are the best indicators of the topic of a tweet. They have been used as topic labels in many other studies [24].

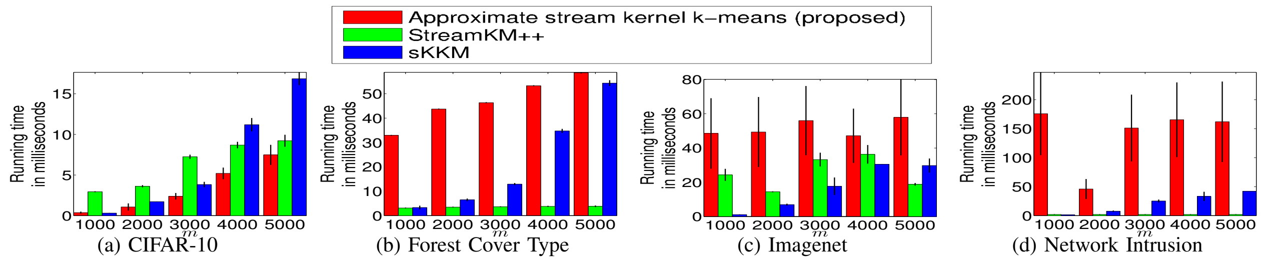


Fig. 4: Effect of the initial sample size m on the running time (in milliseconds) of the proposed approximate stream kernel k-means algorithm. m represents the initial sample set size, the coreset size and the chunk size for the approximate stream kernel k-means, StreamKM++ and sKKM algorithms, respectively.

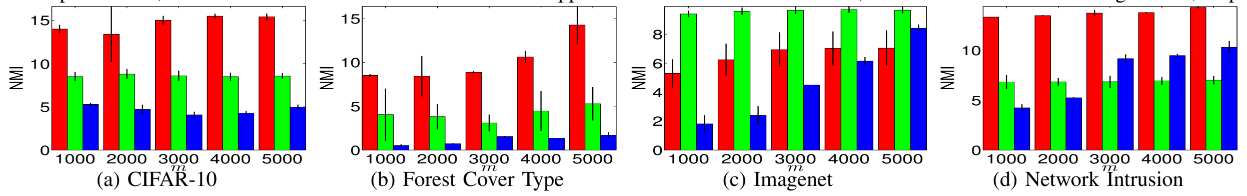
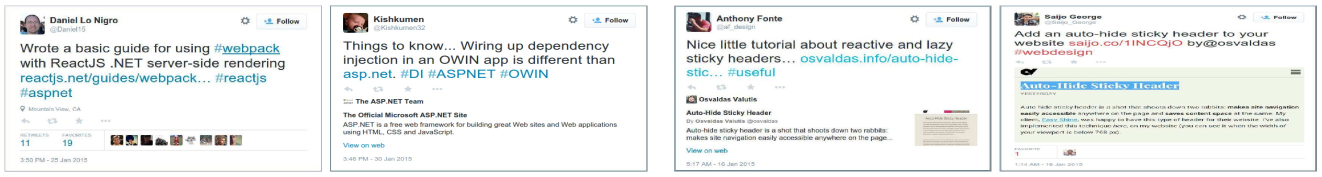


Fig. 5: Effect of the initial sample size m on the NMI (in %) of the proposed approximate stream kernel k-means algorithm. m represents the initial sample set size, the coreset size and the chunk size for the approximate stream kernel k-means, StreamKM++ and sKKM algorithms, respectively.



Daniel Lo Nigro
@Daniel15
Follow

Wrote a basic guide for using #webpack with ReactJS .NET server-side rendering reactjs.net/guides/webpack... #reactjs #aspnet

Mountain View, CA

RETWEETS 11 FAVORITES 19

3:50 PM - 25 Jan 2015

Kishkumen
@Kishkumen32
Follow

Things to know... Wiring up dependency injection in an OWIN app is different than asp.net. #DI #ASPNET #OWIN

The ASP.NET Team

The Official Microsoft ASP.NET Site
ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.

View on web

3:46 PM - 30 Jan 2015

(a)

Anthony Fonte
@af_design
Follow

Nice little tutorial about reactive and lazy sticky headers... osvaldas.info/auto-hide-stic... #useful

Osvaldas Valutis

Auto-Hide Sticky Header
By Osvaldas Valutis @osvaldas
Auto-hide sticky header is a shot that shoots down two rabbits: makes site navigation easily accessible anywhere on the page...

View on web

5:17 AM - 16 Jan 2015

Saijo George
@Saijo_George
Follow

Add an auto-hide sticky header to your website saijo.co/1INCQjO by@osvaldas #webdesign

Auto-Hide Sticky Header
YESTERDAY

Auto hide sticky header is a shot that shoots down two rabbits: **makes site navigation easily accessible** anywhere on the page and **saves content space** at the same. My client, Easy Shine, was happy to have this type of header for their website. I've also implemented this technique *here*, on my website (you can see it when the width of your viewport is below 768 px).

FAVORITE 1

1:14 AM - 16 Jan 2015

(b)

Fig. 6: Sample tweets from the (a) ASP.NET and (b) HTML clusters. Tweets which are proximal in time and have similar keywords belong to the same cluster.

memory, and achieve higher cluster quality than the current stream clustering algorithms. By using a parallelized scheme for updating the eigensystem and defining the kernel function appropriately, our algorithm can be easily extended to cluster parallel data streams and time series data. This is a potential direction for future work.

References

[1] R. Chitta, R. Jin, T. C. Havens, and A. K. Jain, "Approximate kernel k-means: Solution to large scale kernel clustering," in *Proceedings of the International Conference on Knowledge Discovery and Data mining*, 2011, pp. 895–903.

[2] T. Liu, C. Rosenberg, and H. Rowley, "Clustering billions of images with large scale nearest neighbor search," in *Proceedings of the IEEE Workshop on Applications of Computer Vision*, 2007, pp. 28–33.

[3] D. Judd, P. K. McKinley, and A. K. Jain, "Large-scale parallel data clustering," in *Proceedings of the International Conference on Pattern Recognition*, vol. 4, 1996, pp. 488–493.

[4] M. Girolami, "Mercer kernel-based clustering in feature space," *IEEE Transactions on Neural Networks*, vol. 13, no. 3, pp. 780–784, 2002.

[5] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta, "A survey of kernel and spectral methods for clustering," *Pattern Recognition*, vol. 41, no. 1, pp. 176–190, 2008.

[6] A. Jain, Z. Zhang, and E. Y. Chang, "Adaptive non-linear clustering in data streams," in *Proceedings of the International Conference on Information and Knowledge Management*, 2006, pp. 122–131.

[7] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff, "Fast approximation of matrix coherence and statistical leverage," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 3475–3506, 2012.

[8] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. de Carvalho, and J. Gama, "Data stream clustering: A survey," *ACM Computing Surveys*, vol. 46, no. 1, pp. 1–37, 2013.

[9] M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, and C. Sohler, "StreamKM++: A clustering algorithm for data streams," *Journal of Experimental Algorithmics*, vol. 17, no. 2, pp. 1–30, 2012.

[10] N. Ailon, R. Jaiswal, and C. Monteleoni, "Streaming k-means approximation," in *Proceedings of the Conference on Neural Information Processing Systems*, 2009, pp. 10–18.

[11] M. Shindler, A. Wong, and A. W. Meyerson, "Fast and accurate k-means for large datasets," in *Proceedings of the Conference on Neural Information Processing Systems*, 2011, pp. 2375–2383.

[12] H. Ning, W. Xu, Y. Chi, Y. Gong, and T. S. Huang, "Incremental spectral clustering by efficiently updating the eigen-system," *Pattern Recognition*, vol. 43, no. 1, pp. 113–127, 2010.

[13] T. C. Havens, "Approximation of kernel k-means for streaming data," in *Proceedings of the International Conference on Pattern Recognition*, 2012, pp. 509–512.

[14] H. Zha, X. He, C. Ding, M. Gu, and H. D. Simon, "Spectral relaxation for k-means clustering," in *Proceedings of the Conference on Neural Information Processing Systems*, 2001, pp. 1057–1064.

[15] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases," *ACM SIGMOD Record*, vol. 25, no. 2, pp. 103–114, 1996.

[16] S. Chatterjee and A. S. Hadi, "Influential observations, high leverage points, and outliers in linear regression," *Statistical Science*, vol. 1, no. 3, pp. 379–393, 1986.

[17] A. Gittens and M. W. Mahoney, "Revisiting the nystrom method for improved large-scale machine learning," *arXiv preprint arXiv:1303.1849*, 2013.

[18] C. C. Aggarwal, *Data streams: Models and algorithms*. Springer Science & Business Media, 2007, vol. 31.

[19] M. Brand, "Fast low-rank modifications of the thin singular value decomposition," *Linear Algebra and its Applications*, vol. 415, no. 1, pp. 20–30, 2006.

[20] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Department of Computer Science, University of Toronto, Tech. Rep., 2009.

[21] J. Blackard and D. Dean, "Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables," *Computers and Electronics in Agriculture*, vol. 24, no. 3, pp. 131–152, 1999.

[22] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[23] S. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. Chan, "Cost-based modeling for fraud and intrusion detection: Results from the JAM project," in *Proceedings of the DARPA Information Survivability Conference and Exposition*, vol. 2, 2000, pp. 130–144.

[24] F. Godin, V. Slavkovikj, W. De Neve, B. Schrauwen, and R. Van de Walle, "Using topic models for twitter hashtag recommendation," in *Proceedings of the International Conference on World Wide Web Companion*, 2013, pp. 593–596.

[25] G. Petkos, S. Papadopoulos, and Y. Kompatsiaris, "Two-level message clustering for topic detection in twitter." in *Proceedings of the SNOW Data Challenge*, 2014, pp. 49–56.