

# Efficient Streaming Text Clustering

Shi Zhong

Department of Computer Science and Engineering

Florida Atlantic University, Boca Raton, FL 33431

Email: zhong@cse.fau.edu

**Abstract**—Clustering data streams has been a new research topic, recently emerged from many real data mining applications, and has attracted a lot of research attention. However, there is little work on clustering high-dimensional streaming text data. This paper combines an efficient online spherical k-means (OSKM) algorithm with an existing scalable clustering strategy to achieve fast and adaptive clustering of text streams. The OSKM algorithm modifies the spherical k-means (SPKM) algorithm, using online update (for cluster centroids) based on the well-known Winner-Take-All competitive learning. It has been shown to be as efficient as SPKM, but much superior in clustering quality. The scalable clustering strategy was previously developed to deal with very large data bases that cannot fit into a limited memory and that are too expensive to read/scan multiple times. Using the strategy, one keeps only sufficient statistics for history data to retain (part of) the contribution of history data and to accommodate the limited memory. To make the proposed clustering algorithm adaptive to data streams, we introduce a forgetting factor that applies exponential decay to the importance of history data. The older a set of text documents, the less weight they carry. Our experimental results demonstrate the efficiency of the proposed algorithm and reveal an intuitive and an interesting fact for clustering text streams—one needs to forget to be adaptive.

## I. INTRODUCTION

In many real data mining applications, data comes in as a continuous stream and presents several challenges to traditional static data mining algorithms. Application examples include topic detection from a news stream, intrusion detection from continuous network traffic, object recognition from video sequences, etc. Challenges lie in several aspects: high algorithm efficiency is required in real time; huge data volume that cannot be kept in memory all at once; multiple scans from secondary storage is not desirable since it causes intolerable delays; and mining algorithms need to be adaptive since data patterns change over time.

Many recent research works tried to address these challenges, yet with different focuses. In general, they can be divided into two categories—*scalable* methods and *adaptive* methods—based on the focus. Works in the first category (Bradley et al., 1998; Farnstrom et al., 2000; O’Callaghan et al., 2002; Guha et al., 2003) aim to capture overall data characteristics over a long time period despite the constraint that at any time only a small subset of data can reside in the memory for a limited time frame for once. Adaptive methods

(Hulten et al., 2001; Agrawal et al., 2003; Agrawal et al., 2004; Kifer et al., 2004), on the other hand, focus on following changes in the data characteristics over time. Therefore, the quality of such mining algorithms is usually measured locally at a set of stream points.

Like most of the aforementioned works, we focus on clustering techniques in the paper. More specifically, we investigate a streaming text clustering method that emphasizes on adaptivity.

Text clustering has become an increasingly important technique for unsupervised document organization, automatic topic extraction, and fast information retrieval or filtering. Not surprisingly, there has been a vast literature on text clustering. However, there has been little work on clustering streaming text data such as news streams. Banerjee and Ghosh (2004) extended a frequency-sensitive spherical k-means algorithm to cluster text streams. Their work focused on extracting a set of clusters (of balanced size) over a long period. That is, their method effectively belongs to the *scalable* category and they measured clustering performance using all documents in a stream. Furthermore, their experimental setting on streaming clustering is flawed in that they repeated a set of documents multiple times to create a stream, thus effectively reading each text document into memory multiple times.

In this paper, we extend our previous work on efficient online spherical k-means (Zhong, 2005) to cluster text streams. The extension is essentially a combination of the online spherical k-means (OSKM) algorithm with scalable clustering techniques (Farnstrom et al., 2000). This combination leverages on the efficiency and effectiveness of OSKM, which has been demonstrated in Zhong (2005), as well as the ability of *scalable* methods to process huge data streams. But there is an improvement on the *scalable* part. Similar to the methods proposed in Agrawal et al. (2003), we focus on the adaptivity of our streaming algorithm by introducing to the scalable approach a decay factor that exponentially reduce the contribution of history data. The decay factor essentially impose a forgetting mechanism on the clustering process, which is commonly used in adaptive neural networks (Widrow & Lehr, 1990).

In this paper, we design experiments to investigate the effect of the forgetting mechanism, and results show that one needs to forget to adapt. That is, clustering quality at a series of stream points is high when the decay parameter is small (or history is quickly forgotten). This indicates that the original scalable approach (Farnstrom et al., 2000) cannot be applied

directly to streaming data clustering without any change.

The rest of this paper is organized as follows. We review the OSKM algorithm (Zhong, 2005) in the next section and scalable clustering techniques (Bradley et al., 1998; Farnstrom et al., 2000) in Section III. The proposed streaming OSKM algorithm is presented in Section IV and experimental results are shown in Section V. Finally, Section VI concludes this paper.

For notation in this paper, we use bold face variables (e.g.,  $\mathbf{x}$ ,  $\mu$ ) to represent vectors, upper case calligraphic alphabets (e.g.,  $\mathcal{X}$ ,  $\mathcal{Y}$ ) to represent sets,  $\|\cdot\|$  to denote the  $L_2$  norm, and  $|\cdot|$  to denote the number of elements in a set.

## II. ONLINE SPHERICAL K-MEANS

### A. Spherical K-means (SPKM)

For high-dimensional data such as text documents (represented as Term Frequency-Inverse Document Frequency vectors), cosine similarity has been shown to be a superior measure to Euclidean distance (Strehl et al., 2000). The implication is that the direction of a document vector is more important than the magnitude. This leads to a unit-vector representation, i.e., each document vector is normalized to be of unit length ( $\|\mathbf{x}\| = 1$ ). Let  $\{\mu_1, \dots, \mu_K\}$  be a set of unit-length centroid vectors, the spherical k-means algorithm (Dhillon & Modha, 2001) aims to maximize the average cosine similarity objective

$$L = \sum_{\mathbf{x}} \mathbf{x}^T \mu_{k(\mathbf{x})}, \quad (1)$$

where  $k(\mathbf{x}) = \arg \max_k \mathbf{x}^T \mu_k$ . The batch version of SPKM again iterates between a data assignment step and a centroid estimation step (Zhong & Ghosh, 2003). It can also be derived from mixture of von Mises-Fisher distributions (Banerjee & Ghosh, 2004). The main difference from standard k-means (MacQueen, 1967) is that the re-estimated mean vectors  $\{\mu\}$  need to be normalized to unit-length.

### B. Online Spherical K-means (OSKM)

The ‘‘Winner-Take-All’’ mechanism is one of the simplest competitive learning strategies (Ahalt et al., 1990) and allows only one winner per input. For each input, only the winning neuron learns (i.e., gets updated). When applied to clustering (with cluster centroids as cells), WTA leads to online k-means clustering. Soft competitive learning methods (Kohonen, 1990; Martinetz et al., 1993) have been employed to adjust multiple neurons per input, with the rate of adjustment inversely proportional to the distance (or some function of the distance) between each neuron and the given input.

We use WTA learning in online spherical k-means clustering to maximize the objective (1). Given a data point  $\mathbf{x}$ , we incrementally update the closest cluster center  $\mu_{k(\mathbf{x})}$  as:

$$\mu_{k(\mathbf{x})}^{(new)} = \frac{\mu_{k(\mathbf{x})} + \eta \frac{\partial L_{\mathbf{x}}}{\partial \mu_{k(\mathbf{x})}}}{Z} = \frac{\mu_{k(\mathbf{x})} + \eta \mathbf{x}}{Z}, \quad (2)$$

where  $\eta$  is a learning rate,  $L_{\mathbf{x}} = \mathbf{x}^T \mu_{k(\mathbf{x})}$ , and  $Z$  is a normalization term to keep the updated  $\mu$  on the unit hypersphere. The derivative term  $\frac{\partial L_{\mathbf{x}}}{\partial \mu_{k(\mathbf{x})}}$  is basically the gradient of the  $\mathbf{x}$ -related part of the objective function with respect to  $\mu_{k(\mathbf{x})}$ . Thus this is essentially an incremental gradient ascent algorithm.

**Algorithm:** online spherical k-means (OSKM)  
**Input:** A set of  $N$  unit-length data vectors  $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  in  $\mathbb{R}^d$ , number of clusters  $K$ , and number of batch iterations  $M$ .  
**Output:** A partition of the data vectors given by the cluster identity vector  $\mathcal{Y} = \{y_1, \dots, y_N\}$ ,  $y_n \in \{1, \dots, K\}$ .  
**Steps:**

1. Initialization: initialize the unit-length cluster centroid vectors  $\{\mu_1, \dots, \mu_K\}$ ,  $i = 0$ ;
2. For  $m = 1$  to  $M$   
For  $n = 1$  to  $N$ 
  - (a) For each data vector  $\mathbf{x}_n$ , find the closest centroid  $y_n = \arg \max_k \mathbf{x}_n^T \mu_k$ ;
  - (b) Estimate each centroid as  $\mu_{y_n} \leftarrow \frac{\mu_{y_n} + \eta^{(i)} \mathbf{x}_n}{\|\mu_{y_n} + \eta^{(i)} \mathbf{x}_n\|}$ ;
  - (c)  $i \leftarrow i + 1$ ;

Fig. 1. Online spherical k-means algorithm.

In the OSKM algorithm (Fig. 1), we use an exponentially decreasing learning rate  $\eta^{(i)} = \eta_0(\eta_f/\eta_0)^{\frac{i}{NM}}$ , where  $i$  is the online iteration index ( $0 \leq i \leq NM$ ) and  $M$  is the number of batch iterations. The exponential learning rate schedule has been shown to work better than the flat learning rate (Fritzke, 1995; Zhong, 2005), due to an annealing effect introduced by the gradual decrease of  $\eta$ . There are other learning rate schedules that guarantee each input contributes equally (Wang, 1997). But unequal contributions may not be an issue in our case since we randomize the order of inputs across multiple batch iterations.

### C. Efficient Implementation of OSKM

This section describes several implementation details of the online spherical k-means algorithm that help improve the efficiency and avoid empty clusters.

*Deferred normalization:* It is easy to verify that the computational bottleneck for OSKM is with the centroid normalization step (Step 2(b) in Fig. 1). Since each document vector  $\mathbf{x}$  is usually very sparse, the complexity of addition  $\mu + \eta \mathbf{x}$  is linear in the number of nonzeros in  $\mathbf{x}$ , much lower than  $d$ —the dimensionality of  $\mathbf{x}$ . The complexity of the normalization, however, is  $O(d)$  since the centroid vector  $\mu$  is usually not sparse. Thus the total complexity for estimating  $\mu$ 's is  $O(MNd)$ . This is very costly since  $d$  is the size of word vocabulary, often even larger than  $N$  for many text data sets. In contrast, one can easily verify that, *with a sparse matrix representation*, the total complexity for the data assignment step is just  $O(MKN_{nz})$ , where  $N_{nz}$  is the total number of nonzeros in the whole term-document matrix and usually  $N_{nz} \ll Nd$ . The number of clusters  $K$  is often set *a priori*.

One way of removing this computational bottleneck is to defer the normalization. We keep a record for every  $\mu$ 's  $L_2$  norm (and the square  $s(\mu) = \|\mu\|^2$ ) and do not normalize

any  $\mu$  until its norm gets too large (e.g., close to overflow). As a result, we change the cosine similarity calculation in the assignment step to  $\frac{\mathbf{x}_j \cdot \mu}{\|\mu\|}$  and the centroid estimation step to the following sub-steps:

$$\begin{cases} s(\mu) \leftarrow s(\mu) - \sum_{j \in J} \mu(j)^2, \\ \mu(j) \leftarrow \mu(j) + \eta \|\mu\| \mathbf{x}_j, \forall j \in J, \\ s(\mu) \leftarrow s(\mu) + \sum_{j \in J} \mu(j)^2, \\ \|\mu\| \leftarrow \sqrt{s(\mu)}, \end{cases}$$

where  $J$  is the set of indices for nonzero entries in  $\mathbf{x}$ ,  $\mu(j)$  is the  $j$ -th dimension of  $\mu$ , and  $s(\mu) = \|\mu\|^2$ . Note all these sub-steps together only takes time linear in the number of nonzeros in  $\mathbf{x}$  since only those dimensions in  $\mu$  get updated. The total complexity for computing  $\mu$ 's now becomes  $O(MN_{nz})$ , which is a huge improvement over  $O(MNd)$ . The total time complexity for OSKM is now  $O(MKN_{nz})$ .

*Avoiding Empty Clusters:* It is well-known that the standard k-means algorithm, either batch version or online version, tends to generate empty clusters (due to the greed optimization strategy). The situation deteriorates further as number of clusters and data dimensionality grows. Here we use a very simple heuristic: At the end of each batch iteration (i.e., each time after going through all data points), we keep a list of up to  $K$  data points that are most distant from their cluster centroids, and a list of empty clusters. The centroid of each empty cluster will then be replaced by a data point (from the list constructed above) that is most remote to its center (and the data point will then be removed from the list). We can always achieve zero empty clusters with this strategy.

*Sampling:* Let us consider the gradually decreasing learning rate schedule. Initially, the learning rate is high and the centroids will be moving around a lot and get spread out globally in the data space. As the rate decreases, each mean gets refined and starts adapting to the local cluster structure. We conjecture that many initial moves are redundant and a small random sample would be enough to achieve the same spreading-out effect. Therefore, we propose the following sampling strategy: at the  $m$ -th batch iteration (suppose we go through all data points  $M$  times), we randomly sample  $\frac{mN}{M}$  data points to do the online learning for  $\mu$ 's. Our experiments show that this strategy can double the clustering speed while not damaging the quality of clustering results.

### III. SCALABLE CLUSTERING

A commonly used scalable clustering strategy (Bradley et al., 1998; Farnstrom et al., 2000) was initially proposed to address the challenge of very large data bases, but later was adopted by streaming clustering methods as well (Guha et al., 2003; Agrawal et al., 2003). The basic idea is to go through a large database (or data set) segment by segment. The segment size depends on available buffer capacity. Each segment is read into the buffer (only once), grouped with some clustering algorithm (usually k-means), and then either compressed or discarded before next segment.

In the first proposal, Bradley et al. (1998) divide the data points in a segment (after clustering) into three different

categories: points to be kept intact in the buffer; points to be compressed into sufficient statistics; and points to be discarded. Sufficient statistics include sum, sum-of-squares, and number of points in a compressed group. In later clustering process, each data point kept intact is just like a new point (carrying a weight of 1), but each compressed group carries a weight equal to the number of points compressed. One disadvantage of this method is that the criteria used to divide data points contain parameters that need to be pre-set.

Farnstrom et al. (2000) show that using a simple compression method works equally well as the more complicated dividing (and discarding/compressing/keeping) method. In the simple approach, after clustering each segment (together with summarized history), each cluster is compressed into one history vector (with the weight being the total number of data points in the cluster), which serves as a summarized history for the next round of clustering. Figure 2 shows the pseudocode for this approach (using standard k-means).

**Algorithm:** simple scalable k-means  
**Input:** A set of  $N$  data vectors  $\mathcal{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$  in  $\mathbb{R}^d$  and the number of clusters  $K$ . Segment length  $S$  (in terms of number of data vectors).  
**Output:** A set of  $K$  cluster centroid vectors  $\{\mu_1, \dots, \mu_K\}$ .  
**Steps:**

1. Initialization: initialize the  $K$  history vectors  $\{\mathbf{c}_1, \dots, \mathbf{c}_K\}$  as empty and corresponding weights  $\{w_1, \dots, w_k\}$  as zeros, and the centroid vectors  $\{\mu_1, \dots, \mu_K\}$  randomly;
2. Until data vectors are exhausted
  - (a) Read in  $S$  data vectors, each with a weight of 1;
  - (b) Run k-means on the  $S$  data vectors and  $K$  history vectors, and during each iteration calculate  $\mu_k$  using weighted average;
  - (c) For each cluster  $k$ , update  $w_k$  as the sum of the weights of all data vectors and history vectors in the cluster, and set  $\mathbf{c}_k = \mu_k$ .

Fig. 2. Simple scalable k-means algorithm.

The above scalable clustering ideas can be directly applied to clustering data streams (e.g., Guha et al., 2003) due to the similarity between streaming clustering and scalable clustering. However, streaming clustering has an important difference—it desires an adaptive algorithm that can capture varying cluster distributions over time. This leads to a modification of existing scalable clustering techniques, since, intuitively, simply carrying all the history (in a summarized form) in the clustering process slows down the learning of new clusters. This can partially explain the superiority of the algorithm proposed in Agrawal et al. (2003) to that of (Guha et al., 2003). The former modifies the scalable clustering idea to take into account time information and reduce the weights for history vectors generated in the past.

In the next section, we propose the streaming OSKM algorithm, a combination of the modified scalable clustering idea with efficient online spherical k-means, for clustering text streams.

#### IV. STREAMING OSKM

Even though text documents are assumed to come in as a continuous stream, our streaming OSKM algorithm does not process the stream continuously. Instead, we adopt the same strategy used in scalable clustering—divide a stream into segments and process them segment by segment. This can be viewed as a semi-continuous process, which is a reasonable approach in practice since we usually do not expect to see a change in cluster distributions after only a few new documents.

Processing in segments also has the advantage of visiting a document multiple times while it is in the memory for a limited time frame, which makes running OSKM (usually needs multiple iterations through data) possible. The size of a segment may be determined by available buffer size or application requirements (e.g., how often we want clusters to be updated). In this paper, we experiment on a set of different segment lengths and explore the effect.

**Algorithm:** streaming OSKM

**Input:** A stream of unit-length data vectors  $\mathcal{X} = \{\mathbf{x}_n\}_{n=1,2,\dots}$  in  $\mathbb{R}^d$  and the number of clusters  $K$ . A segment length of  $S$ . Number of iterations  $M$  for OSKM. A decay factor  $0 < \gamma < 1$ .  
**Output:** A set of  $K$  cluster centroid vectors  $\{\mu_1, \dots, \mu_K\}$ .

**Steps:**

1. Initialization: initialize the  $K$  history vectors  $\{\mathbf{c}_1, \dots, \mathbf{c}_K\}$  as empty and corresponding weights  $\{w_1, \dots, w_k\}$  as zeros, the cluster centroid vectors  $\{\mu_1, \dots, \mu_K\}$  randomly, and the segment index  $t = 0$ ;
2. Reading in next  $S$  new data vectors,  $t = t + 1$ ,
  - (a) Set the weight of each new data vector as 1;
  - (b) Run OSKM on the  $S$  data vectors and  $K$  history vectors:  
 Set  $i = 0$   
 For  $m = 1$  to  $M$ 
    - (i) For each data vector  $\mathbf{x}_n$  ( $n = 1, \dots, S$ ), set  
 $y_n = \arg \max_k \mathbf{x}_n^T \mu_k$ ,  
 $\mu_{y_n}^{(new)} \leftarrow \frac{\mu_{y_n} + \eta^{(i)} \mathbf{x}_n}{\|\mu_{y_n} + \eta^{(i)} \mathbf{x}_n\|}$ ,  
 and  $i = i + 1$ ;
    - (ii) For each history vector  $\mathbf{c}_k$  ( $k = 1, \dots, K$ ), set  
 $y = \arg \max_k \mathbf{c}_k^T \mu_k$ ,  
 $\mu_y^{(new)} \leftarrow \frac{\mu_y + \eta^{(i)} w_k \mathbf{c}_k}{\|\mu_y + \eta^{(i)} w_k \mathbf{c}_k\|}$ ,  
 and  $i = i + 1$ ;
  - (c) For each cluster  $k$ , let  $N_k$  be the number of data vectors grouped into the cluster, and  $W_k$  the sum of the weights of all history vectors grouped into the cluster. Set  $\mathbf{c}_k \leftarrow \mu_k$  and  $w_k \leftarrow \gamma(N_k + W_k)$ ;
3. Repeat Step 2 until end of stream or stopped by user.

Fig. 3. Streaming online spherical k-means algorithm.

Our algorithm is shown in Fig. 3, in which  $\eta^{(i)} = \eta_0(\eta_f/\eta_0)^{\frac{i}{M}}$ . There are two worth-mentioning features with our streaming text clustering algorithm. The first is the efficiency leveraged from the OSKM algorithm. To fit OSKM in the streaming algorithm, One only needs to modify it slightly to handle weighted data vectors. This is straightforward given the local objective function that streaming OSKM maximizes

for segment  $t$

$$L^{(t)} = \sum_{\mathbf{x} \in \mathcal{X}^{(t)}} \mathbf{x}^T \mu_{k(\mathbf{x})} + \sum_{\mathbf{c} \in \mathcal{C}^{(t-1)}} w \mathbf{c}^T \mu_{k(\mathbf{c})}, \quad (3)$$

where  $\mathcal{X}^{(t)}$  is the set of documents in segment  $t$ ,  $\mathcal{C}^{(t-1)}$  the set of history vectors updated after previous segment,  $w$  the weight associated with  $\mathbf{c}$ , and  $k(\mathbf{c}) = \arg \max_k \mathbf{c}^T \mu_k$ .

The second feature is the decay factor  $\gamma$  introduced, which causes past history to be forgotten at an exponential rate. Let  $N_k^{(t)}$  be the number of data vectors grouped into cluster  $k$  and  $w_k^{(t)}$  be the weight of history vector  $\mathbf{c}_k$  when clustering segment  $t$ . Assume that every time  $\mathbf{c}_k$  is grouped into cluster  $k$ .<sup>1</sup> Now the weight of history vector  $\mathbf{c}_k$  after segment  $t$  is

$$\begin{aligned} w_k^{(t+1)} &= \gamma(N_k^{(t)} + w_k^{(t)}) \\ &= \gamma(N_k^{(t)} + \gamma(N_k^{(t-1)} + w_k^{(t-1)})) \\ &= \dots \\ &= \sum_{l=1}^t \gamma^l N_k^{(t+1-l)}. \end{aligned}$$

Since  $\gamma$  is usually a number smaller than 1, the contribution of a segment  $t_1$  to clustering a future segment  $t_2$  decreases exponentially (with a weight of  $\gamma^{t_2-t_1}$ ). In our experiments, we shall test several different decay values (from 0 to 1).

#### V. EXPERIMENTAL RESULTS

In this section, we first introduce the data sets and clustering evaluation criteria used in our experiments and then present two sets of experiments, with the first to demonstrate the performance of OSKM and the second to study streaming OSKM.

##### A. Text Datasets

We used the 20-newsgroups data<sup>2</sup> and a number of data sets from the CLUTO toolkit<sup>3</sup> (Karypis, 2002). These data sets provide a good representation of different characteristics as shown in Table I. All data sets are very sparse, as shown by the (very) low values for the fraction of nonzero entries in the term-document matrix.

TABLE I  
SUMMARY OF TEXT DATA SETS (FOR EACH DATA SET,  $N$  IS THE TOTAL NUMBER OF DOCUMENTS,  $d$  THE TOTAL NUMBER OF WORDS,  $K$  THE NUMBER OF CLASSES, AND  $N_{nz}/(Nd)$  THE FRACTION OF NONZEROS IN THE TERM-DOCUMENT MATRIX)

Data	$N$	$d$	$K$	$\frac{N_{nz}}{Nd}$
<i>NG20</i>	19949	43586	20	0.0018
<i>classic</i>	7094	41681	4	0.0008
<i>ohscal</i>	11162	11465	10	0.0053
<i>klb</i>	2340	21839	6	0.0068

<sup>1</sup>This assumption of course does not hold but makes the following analysis simpler. And it is easy to see that the conclusion from the following analysis does not change even when the assumption is not true.

<sup>2</sup><http://kdd.ics.uci.edu/databases/20newsgroups/20newsgroups.html>.

<sup>3</sup><http://www.cs.umn.edu/~karypis/CLUTO/files/datasets.tar.gz>.

The *NG20* data set is a collection of 20,000 messages, collected from 20 different usenet newsgroups, 1,000 messages from each. We preprocessed the raw data set using the Bow toolkit (McCallum, 1996), including chopping off headers and removing stop words as well as words that occur in less than three documents. In the resulting data set, there are 19,949 documents in a 43,586-dimensional space (with empty documents removed).

The other data sets are associated with the CLUTO toolkit and have already been preprocessed (Zhao & Karypis, 2001) and we further removed those words that appear in two or fewer documents.

### B. Clustering Evaluation

Since the category labels of the benchmark documents are known, we chose to measure clustering quality using the mutual information  $I(Y; \hat{Y})$  between a r.v.  $Y$ , governing the cluster labels, and a r.v.  $\hat{Y}$ , governing the class labels, which has been shown to be a superior measure to others (Strehl & Ghosh, 2002). We shall follow the definition of normalized mutual information (*NMI*) using geometrical mean,  $NMI = \frac{I(Y; \hat{Y})}{\sqrt{H(Y) \cdot H(\hat{Y})}}$ . The *NMI* value is 1 when clustering results perfectly match the external category labels and close to 0 for a random partitioning.

Clustering efficiency is measured by run time.

### C. Results on OSKM

We compare the following algorithms: **SPKM**—batch spherical k-means; **OSKM**—online spherical k-means with the exponentially decreasing learning rate schedule (with  $\eta_0 = 1.0, \eta_f = 0.01$ ), efficient implementation tricks are used except sampling; **OSKM-s**—OSKM plus the sampling trick in Section II-C; **CLUTO**—We picked the state-of-the-art graph-based clustering algorithm, CLUTO (Karypis, 2002), as the leading representative of this class of similarity-based approaches, for a comparison. We used the *vcluster* program with default setting in the CLUTO package. The default objective function is equivalent to the average cosine similarity objective in (1).

For SPKM and OSKM, we used a common initialization strategy—randomly perturbing global mean to get  $K$  similar centroid vectors. We use a maximum number of iterations  $M = 20$  and a relative convergence criterion of  $1e^{-6}$ . Note that no relative criterion is needed for OSKM. Each experiment is run ten times, the *NMI* values (averages and standard deviations), and run time results are reported. The run time results were measured on a PC with Intel Pentium 4 3GHz CPU and 1GB memory, running Windows XP.

Table II shows the *NMI* results. Boldface highlights the best result in each column. We have performed paired *t*-tests (not shown here due to space limit) to compare different algorithms in the table. Both OSKM and OSKM-s are significantly better than SPKM on all four data sets. OSKM-s is comparable to OSKM on all but the *classic* data set. OSKM-s is significantly better than CLUTO on all four data sets.

TABLE II  
NORMALIZED MUTUAL INFORMATION (*NMI*) RESULTS

	<i>NG20</i>	<i>ohscal</i>	<i>classic</i>	<i>k1b</i>
$K$	20	10	4	6
SPKM	.54 ± .02	.42 ± .03	.55 ± .04	.56 ± .05
OSKM	<b>.59 ± .01</b>	.44 ± .01	<b>.63 ± .03</b>	<b>.66 ± .03</b>
OSKM-s	<b>.59 ± .01</b>	<b>.45 ± .01</b>	.59 ± .03	.65 ± .03
CLUTO	.58 ± .01	.44 ± .00	.55 ± .01	.62 ± .03

Table III shows the average run time results in seconds. OSKM-s is the fastest for all four data sets. It reduces the run time of OSKM by half, but performs just as well. Except for *NG20* data set, OSKM is faster than CLUTO. Here we fixed the value of  $K$  to be the same as the true number of classes in each data set, but different  $K$  values do not change our conclusion (results not shown due to space limit).

TABLE III  
RUN TIME RESULTS (IN SECONDS)

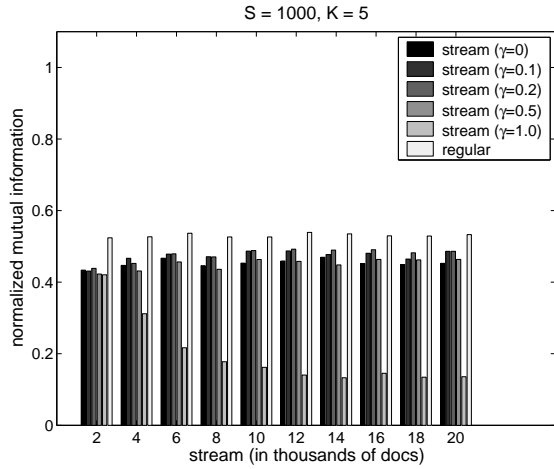
	<i>NG20</i>	<i>ohscal</i>	<i>classic</i>	<i>k1b</i>
$K$	20	10	4	6
SPKM	19.87	3.5	0.34	0.96
OSKM	21.88	3.45	0.78	1.05
OSKM-s	<b>11.38</b>	<b>1.8</b>	<b>0.27</b>	<b>0.58</b>
CLUTO	18.35	5.5	1.16	1.56

### D. Results on Stream Data

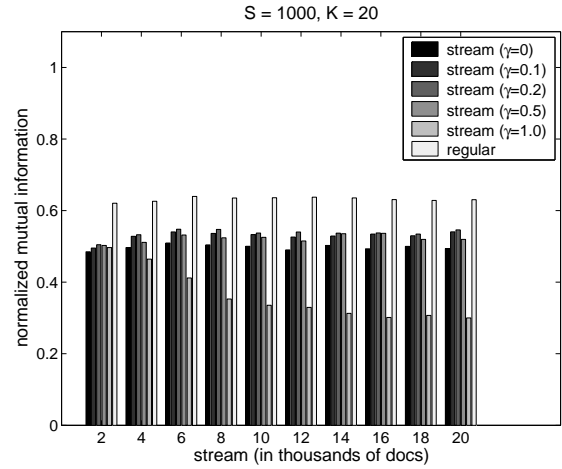
We used only the largest data set described above (*NG20*) to conduct our streaming clustering experiments. To study the adaptivity of streaming OSKM with different  $\gamma$  values, we manipulate the *NG20* data set to get two different types of streams (of 19,949 documents). The first one, *NG20-random*, is a random reordering of the original data set, thus minimizing changes in cluster distributions over the whole stream. The second one, *NG20-drift*, is created in such a way that the stream starts with 5 classes (randomly mixed) and slowly drifts into another 5 classes, then another 5 classes, and then final 5 classes. So there are four big chunks, each dominated by a different set of five classes. There are overlaps between neighboring chunks but no overlaps between non-neighboring chunks, which means there are at most 10 clusters in any one segment of the stream.

Experimental setting for the OSKM part of the streaming OSKM algorithm is the same as that in the previous section. Other parameters such as segment length  $S$ , number of clusters  $K$ , and decay factor  $\gamma$ , are varied so that we can study the effect of each of them (see Fig. 4 and 5).

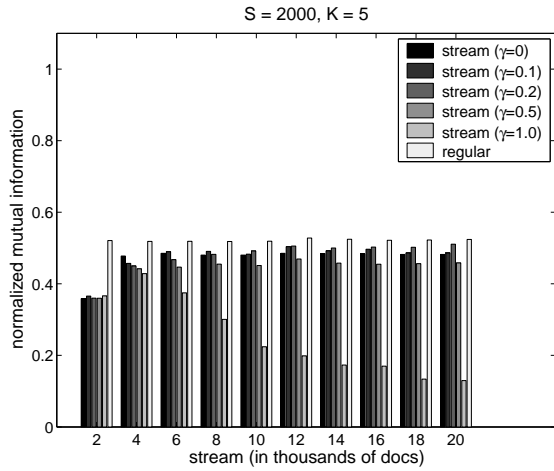
Fig. 4 show the *NMI* results averaged over 10 runs of each algorithm (each run on a different randomly generated *NG20-random* data set). Fig. 5 show average *NMI* results on *NG20-drift* data sets. The regular OSKM algorithm is run on the whole stream though the presented *NMI* results are measured locally at different stream points. Running regular OSKM over the whole stream is not possible in real streaming clustering problems, but presented here for comparison.



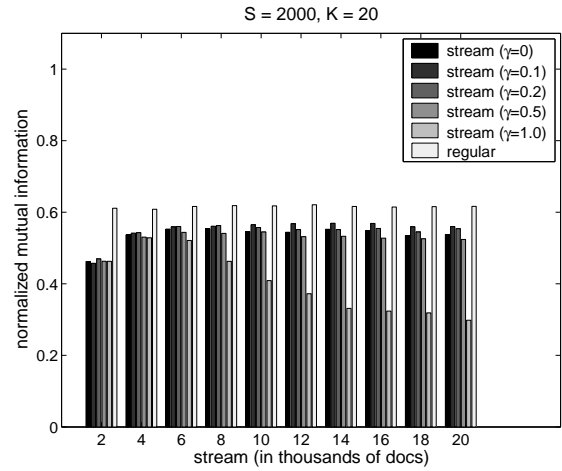
(a)



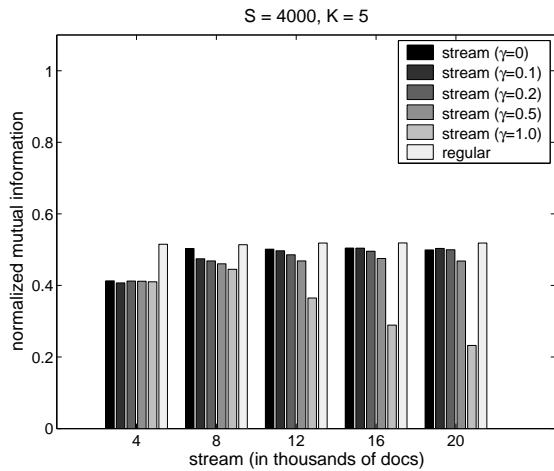
(d)



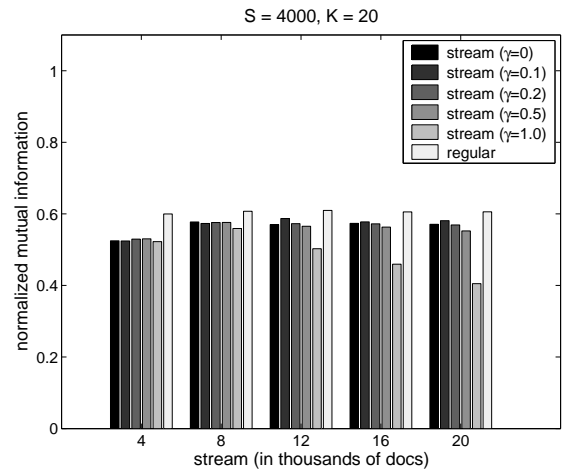
(b)



(e)

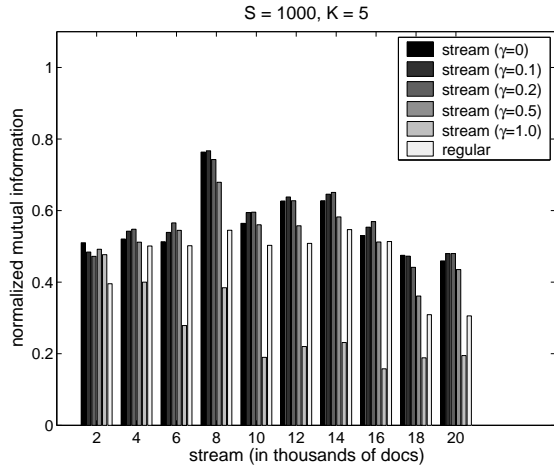


(c)

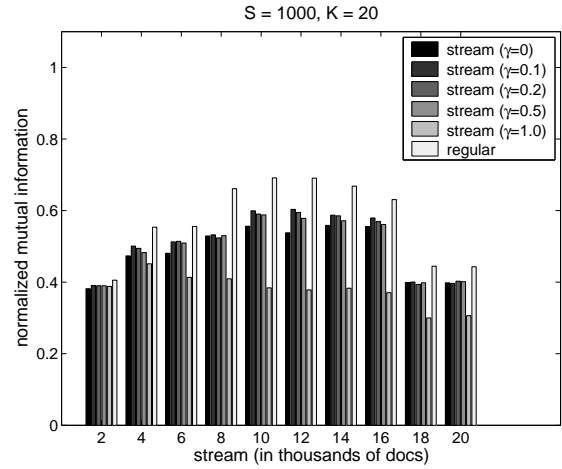


(f)

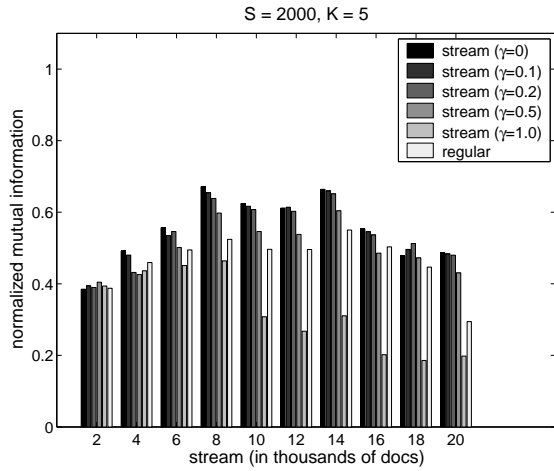
Fig. 4. NMI Results for streaming OSKM and regular OSKM on *NG20-random* data sets, with number of clusters  $K = 5$  on the left and  $K = 20$  on the right. The NMI value for each stream point  $n_t$  is measured using  $S$  documents (from  $n_t - S + 1$  to  $n_t$ ).



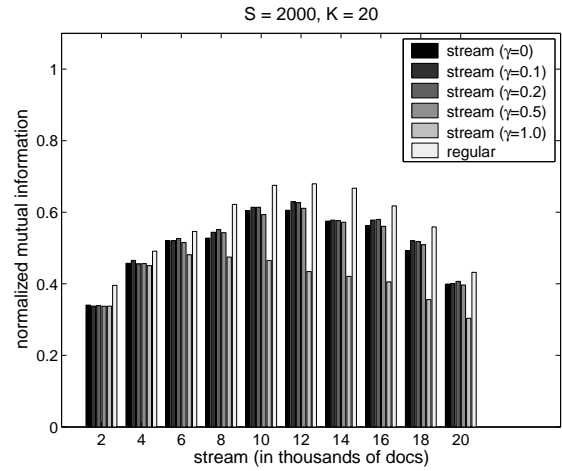
(a)



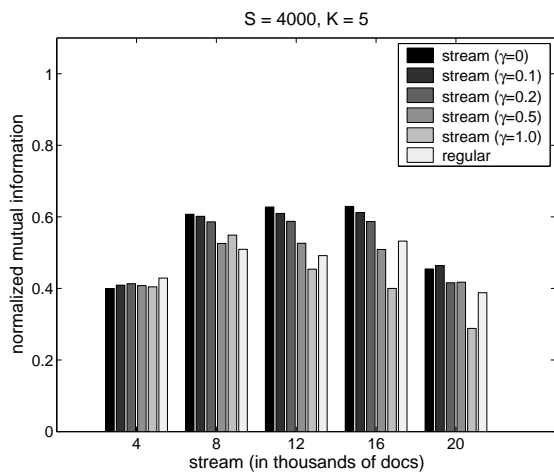
(d)



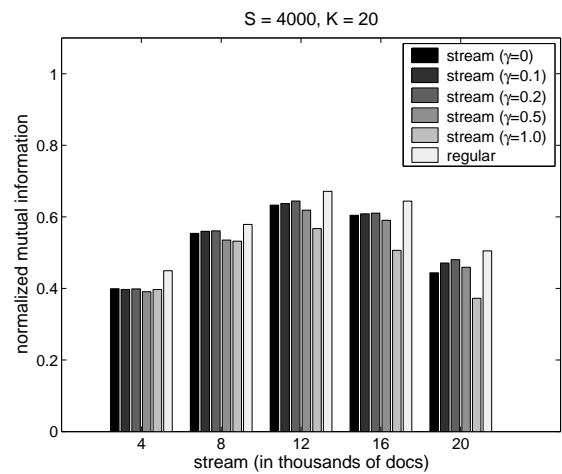
(b)



(e)



(c)



(f)

Fig. 5. NMI Results for streaming OSKM and regular OSKM on *NG20-drift* data sets, with number of clusters  $K = 5$  on the left and  $K = 20$  on the right. The NMI value for each stream point  $n_t$  is measured using  $S$  documents (from  $n_t - S + 1$  to  $n_t$ ).

Several observations can be made from the results. First of all, streaming OSKM with  $\gamma = 1.0$  is the worst-performing one in all situations (except at the beginning of each stream), indicating that carrying all history information hinders adaptive learning over a stream (especially when further into a stream, i.e., when more history gets accumulated). In contrast, a  $\gamma$  value of 0.5 or lower leads to much better performance. In many situations (except when  $S$  is large and  $K$  is small),  $\gamma = 0.1$  or  $\gamma = 0.2$  leads to the highest NMI values among all streaming OSKM versions. This suggests that carrying a little history often helps OSKM better capture the clusters in current segment. It is intuitive that this benefit shall diminish as segment size grows and number of clusters is small, as seen in Fig. 4(c) and Fig. 5(c). Overall, we think these results suggest a general intuitive fact in streaming data clustering—one needs to forget to adapt to new data.

On *NG20-random* data sets, the NMI values are relatively flat for all algorithms (except streaming OSKM with  $\gamma = 1.0$ ), and increase as  $K$  goes from 5 to 20. The regular OSKM always delivers better NMI values. On *NG20-drift* data sets, the NMI values for each algorithm is not flat any more because of changing characteristics over time. When  $K = 5$ , streaming OSKM (with  $\gamma < 0.5$ ) outperforms regular OSKM at most stream points (except for the beginning of each stream), showing good adaptivity. When  $K = 20$ , regular OSKM performs better since it uses the whole stream thus have information about all 20 classes.

For the results in this section, we used OSKM in the streaming algorithm and observed that, over the whole stream, run time for streaming OSKM varied between one to two times of the run time of regular OSKM. According to our time results, we can at least process about 500 documents per second. In practice, OSKM-s can be used to further improve speed.

## VI. CONCLUSION

In this paper, we have investigated an efficient online spherical k-means algorithm and constructed an efficient streaming text clustering techniques by combining it with existing scalable clustering algorithms. The online spherical k-means algorithm, which employs the “Winner-Take-All” competitive learning technique and an exponentially decreasing learning rate schedule, outperforms other approaches compared in this paper. We have shown that the proposed streaming OSKM algorithm can nicely capture changing cluster distributions over time. We have also investigated, through designed experiments, the effect of several parameters, especially a decay factor, associated with the streaming OSKM technique. Our results show that a small decay factor, which means forgetting history quickly, leads to much better clustering results than a decay factor of 1 (which means keeping all summarized history).

## REFERENCES

Agrawal, C. C., Han, J., Wang, J., & Yu, P. S. (2003). A framework for clustering evolving data streams. *Proc. 29th Int. Conf. Very Large Data Bases* (pp. 81–92). Berlin, Germany.

Agrawal, C. C., Han, J., Wang, J., & Yu, P. S. (2004). A framework for projected clustering of high dimensional data streams. *Proc. 30th Int. Conf. Very Large Data Bases* (pp. 852–863). Toronto, Canada.

Ahalt, S. C., Krishnamurthy, A. K., Chen, P., & Melton, D. E. (1990). Competitive learning algorithms for vector quantization. *Neural Networks*, 3, 277–290.

Banerjee, A., & Ghosh, J. (2004). Frequency-sensitive competitive learning for scalable balanced clustering on high-dimensional hyperspheres. *IEEE Trans. Neural Networks*, 15, 702–719.

Bradley, P. S., Fayyad, U. M., & Reina, C. (1998). Scaling clustering algorithms to large databases. *Proc. 4th ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining* (pp. 9–15).

Dhillon, I. S., & Modha, D. S. (2001). Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42, 143–175.

Farnstrom, F., Lewis, J., & Elkan, C. (2000). Scalability for clustering algorithms revisited. *SIGKDD Explorations*, 2, 51–57.

Fritzke, B. (1995). Incremental learning of local linear mappings. *Int. Conf. Artificial Neural Networks* (pp. 217–222).

Guha, S., Meyerson, A., Mishra, N., Motwani, R., & O’Callaghan, L. (2003). Clustering data streams: Theory and practice. *IEEE Trans. Knowledge Data Eng.*, 15, 515–528.

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. *Proc. 7th ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining* (pp. 97–106). San Francisco, CA.

Karypis, G. (2002). *CLUTO - a clustering toolkit*. Dept. of Computer Science, University of Minnesota. <http://www-users.cs.umn.edu/~karypis/cluto/>.

Kifer, D., Ben-David, S., & Gehrke, J. (2004). Detecting changes in data streams. *Proc. 30th Int. Conf. Very Large Data Bases* (pp. 180–191). Toronto, Canada.

Kohonen, T. (1990). The Self-Organizing Map. *Proc. IEEE*, 78, 1464–1480.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Proc. 5th Berkeley Symp. Math. Statistics and Probability* (pp. 281–297).

Martinetz, T. M., Berkovich, S. G., & Schulten, K. J. (1993). “Neural-Gas” network for vector quantization and its application to time-series prediction. *IEEE Trans. Neural Networks*, 4, 558–569.

McCallum, A. K. (1996). Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. Available at <http://www.cs.cmu.edu/~mccallum/bow>.

O’Callaghan, L., Mishra, N., Meyerson, A., Guha, S., & Motwani, R. (2002). Streaming-data algorithms for high-quality clustering. *Proc. 18th Int. Conf. Data Engineering* (pp. 685–696). San Jose, CA.

Strehl, A., & Ghosh, J. (2002). Cluster ensembles — a knowledge reuse framework for combining partitions. *Journal of Machine Learning Research*, 3, 583–617.

Strehl, A., Ghosh, J., & Mooney, R. J. (2000). Impact of similarity measures on web-page clustering. *AAAI Workshop on AI for Web Search* (pp. 58–64).

Wang, L. P. (1997). On competitive learning. *IEEE Trans. Neural Networks*, 8, 1214–1217.

Widrow, B., & Lehr, M. A. (1990). 30 years of adaptive neural networks: Perceptron, madaline and backpropagation. *Proc. IEEE*, 78, 1415–1442.

Zhao, Y., & Karypis, G. (2001). *Criterion functions for document clustering: experiments and analysis* (Technical report #01-40). Department of Computer Science, University of Minnesota.

Zhong, S. (2005). Efficient online spherical k-means clustering. *Proc. IEEE Int. Joint Conf. Neural Networks*. July 31 - August 4, 2005. Montreal, Canada.

Zhong, S., & Ghosh, J. (2003). A unified framework for model-based clustering. *Journal of Machine Learning Research*, 4, 1001–1037.