Model-based Clustering of Short Text Streams

Jianhua Yin
School of Computer Science and Technology, Shandong University
jhyin@sdu.edu.cn

Daren Chao
School of Computer Science and Technology, Shandong University
drchao@mail.sdu.edu.cn

Zhongkun Liu
School of Computer Science and Technology, Shandong University
zkliu@mail.sdu.edu.cn

Wei Zhang*
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University
zhangwei.thu2011@gmail.com

Xiaohui Yu
School of Computer Science and Technology, Shandong University
xhy@sdu.edu.cn

Jianyong Wang
Department of Computer Science and Technology, Tsinghua University
jianyong@tsinghua.edu.cn

ABSTRACT

Short text stream clustering has become an increasingly important problem due to the explosive growth of short text in diverse social medias. In this paper, we propose a model-based short text stream clustering algorithm (MStream) which can deal with the concept drift problem and sparsity problem naturally. The MStream algorithm can achieve state-of-the-art performance with only one pass of the stream, and can have even better performance when we allow multiple iterations of each batch. We further propose an improved algorithm of MStream with forgetting rules called MStreamF, which can efficiently delete outdated documents by deleting clusters of outdated batches. Our extensive experimental study shows that MStream and MStreamF can achieve better performance than three baselines on several real datasets.

CCS CONCEPTS

• **Information systems** → **Data stream mining**; **Clustering**;

KEYWORDS

Text Stream Clustering; Mixture Model; Dirichlet Process

ACM Reference Format:
Jianhua Yin, Daren Chao, Zhongkun Liu, Wei Zhang, Xiaohui Yu, and Jianyong Wang. 2018. Model-based Clustering of Short Text Streams. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3219819.3220094

1 INTRODUCTION

Short text streams like microblog posts are popular on the Internet and often form clusters around real life events or stories. The task of clustering short text streams is to group documents into clusters as they arrive in a temporal sequence, which has many applications such as search result diversification, event detection and tracking, and text summarization [1, 24]. The short text stream clustering problem has the following challenges: (1) The sparsity of short text. (2) The documents arrive continuously, causing that storing all documents and iterating multiple times like static text clustering methods are impossible. (3) The topics of the text stream may continuously evolve over time, so we need to detect new clusters and remove outdated clusters automatically.

In general, the problem of clustering short text streams has two schemes: the one pass scheme and the batch scheme. The one pass scheme assumes that the streaming documents come one by one, we can process each document only one time [3, 10, 23, 32, 33]. The batch scheme assumes that the streaming documents come in batch, we can process the documents in each batch multiple times [4, 5, 8, 18]. In real applications, the batch scheme maybe more reasonable, because we can preprocess and cluster short text stream in parallel, which means we can preprocess a batch of documents and then send them to the stream clustering algorithm. In the batch scheme, we can iterate the documents of current batch multiple times, and discard them when a new batch arrives. The performance of the stream clustering algorithm can improve apparently when we allow multiple iterations of each batch.

In this paper, we propose two model-based short text stream clustering algorithms that can work well on both of the above two schemes. We first propose a short text stream clustering algorithm based on the Dirichlet process multinomial mixture (DPMM) model, called MStream. The MStream algorithm has one pass clustering process and update clustering process of each batch. Experiments show that MStream can achieve state-of-the-art performance with only one pass of the stream, and can have even better performance when we allow multiple iterations of each batch. Rather than assuming a document is distributed over topics, we assume each document is associated with only one topic (cluster). In this way, the MStream algorithm can cope with the sparsity problem of short text. Moreover, we do not need a similarity threshold to determine when to assign a document to a new cluster, because we can compute the probability of a document choosing a new cluster directly from the DPMM model following [31]. In this way, the MStream algorithm can deal with the concept drift problem naturally.

As the number of clusters increases with more documents coming in, the space and time complexity of MStream will grow too large if we do not delete outdated clusters. Besides, we usually are

*Corresponding author.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220094

interested in the information of a specific period of time, rather than the whole stream. However, it is difficult and inefficient to detect and delete outdated clusters, because we cannot store documents of all clusters in memory. We propose an improved algorithm of MStream with forgetting rules called MStreamF, which can delete outdated documents by deleting clusters of outdated batches. In MStreamF, we still only store the documents of current batch, while we can memorize the clusters of several batches. Because the clusters of a batch contain all requisite information for deleting the documents of a batch, and memorizing the clusters of a batch needs not much memory. When the documents of a batch are outdated, we can delete these documents from the current cluster feature (CF) vectors with a subtraction operation effectively. The MStreamF algorithm can also store the clusters of each batch on disk for later offline analysis of the text stream.

The contributions of this paper are summarized as follows.

- We propose a model-based clustering algorithm for short text streams (MStream), which can deal with the concept drift problem naturally. The MStream algorithm can achieve state-of-the-art performance with only one pass of the stream, and can have even better performance when we allow multiple iterations of each batch.
- We propose an improved algorithm of MStream with forgetting rules called MStreamF, which can efficiently delete outdated documents by deleting clusters of outdated batches.
- We conduct extensive experimental study on several real datasets, which validates both the effectiveness and efficiency of our methods. The source code and datasets are available at https://github.com/jackyin12/MStream.

2 RELATED WORK

General surveys on stream data clustering can be found in [1, 19, 20, 24]. Text stream clustering methods can be categorized into the following two categories: similarity-based stream clustering and model-based stream clustering.

2.1 Similarity-based Stream Clustering

Similarity-based text stream clustering methods mostly use the vector space model to represent documents and choose similarity metric like cosine similarity to measure the similarity between documents or clusters.

CluStream [3] is one of the most classic stream clustering methods, which consists of an online micro-clustering component and an offline macro-clustering component. CluStream uses the pyramidal time frame to store historical micro-clusters at different moments for later analysis. DenStream [10] combines micro-clustering with a density-estimation process for stream clustering, which can form data clusters of any shape and handle outliers. Yoo et al. [32] presented a streaming spectral clustering method which maintains an approximation of the normalized Laplacian of the data stream over time and efficiently updates the changing eigenvector of the Laplacian in a streaming fashion.

An efficient stream text clustering algorithm was presented by Zhong et al. [33] using an online update for cluster centriods based on the well-known Winner-Take-All competitive learning. Aggarwal and Yu [2] presented a condensation based method for text and categorical data stream clustering which summarizes the stream into fine grained cluster droplets. Shou et al. [23] presented a continuous summarization prototype called Sumblr for tweet streams. Sumblr has a tweet stream clustering module which compresses tweets into tweet feature vectors (TCVs) and maintains them in an online fashion. Kalogeratos et al. [17] presented an approach using term burst information for text stream clustering. This approach takes the advantage of the fact that most of the important documents of a topic are published during the period in which the "main" topic terms are bursty.

The limitation of similarity-based text stream clustering methods is that they need to choose a similarity threshold manually to determine whether a document is assigned to a new cluster or not. Our proposed MStream method computes the probability of a document belonging to existing clusters and a new cluster, and assign a document to an existing cluster or a new cluster according to these probabilities. In this way, MStream can detect new clusters more naturally and deal with the concept drift problem.

2.2 Model-based Stream Clustering

Model-based text stream clustering methods assume documents are generated by a mixture model, and then use techniques like Gibbs Sampling [14] and Sequential Monte Carlo [11] to estimate the parameters of the mixture model, so as to obtain the clustering results.

Many models that extend Latent Dirichlet Allocation (LDA) [9] have been proposed for modeling text streams, such as dynamic topic model (DTM) [8], topic over time model (TOT) [27], dynamic mixture model (DMM) [29], topic tracking model (TTM) [15], temporal LDA (TM-LDA) [28], streaming LDA (ST-LDA) [5], and Dirichlet-Hawkes topic model [12]. These methods assume that the content of documents are rich enough to infer per-document multinomial distributions for each topic. This assumption does not hold for short text, which results these methods cannot achieve good performance on short text streams. Liang et al. [18] presented the dynamic clustering topic model (DCT) which is based-on the Dirichlet multinomial mixture model [21]. DCT can handle short text by assigning a single topic to each short document and using the distributions inferred at certain time as priors for the inference of next batch.

Most of the above models assume the number of clusters as a fixed number, which means they cannot cope with the concept drift problem of text stream clustering efficiently. Ahmed and Xing [4] presented the temporal Dirichlet process mixture model (TDPM) for evolutionary clustering that automatically increases the number of clusters with the data. However, TDPM is an offline framework and needs the whole sequences of text stream. We propose text stream clustering methods that can work well on both the one pass scheme and the batch scheme. Besides, our methods can handle both short text challenge and concept drift challenge.

3 BACKGROUND

In this section, we give a brief introduction of Dirichlet process and the Dirichlet process multinomial mixture (DPMM) model.

Dirichlet Process (DP) [26] is one member of nonparametric stochastic processes, which is often used in Bayesian nonparametric

modeling of data. It is a distribution over distributions, i.e. each distribution drawn from a Dirichlet process is itself a distribution. A Dirichlet process, denoted by $DP(\alpha, G_0)$, is parameterized by a base measure G_0, and a concentration parameter α. We donate $G \sim DP(\alpha, G_0)$ as a draw from a Dirichlet process distributed over a given parameter space Φ, and we can draw samples from G since it is also a distribution.

Polya urn scheme [7] describes a process that draws a sequence of samples $\phi_1, \phi_2, \ldots$ from distribution G, which can be summarized as follows:

$$\phi_n|\phi_{1:n-1} \sim \frac{\alpha}{\alpha + n - 1} + \frac{\sum_{k=1}^{n-1} \delta(\phi_n - \phi_k)}{\alpha + n - 1} \tag{1}$$

Here, $\delta(x) = 1$ if $x = 0$ and $\delta(x) = 0$ otherwise. In the beginning there are no balls in the urn, and we pick a color drawn from the base measure G_0, i.e. draw $\phi_1 \sim G_0$, and drop a ball with that color into the urn. In subsequent steps, we will either drop a ball of new color into the urn with probability $\frac{\alpha G_0}{\alpha+n-1}$, or drop a ball of a color already in the urn into the urn with probability $\frac{n-1}{\alpha+n-1}$.

Since the draws of distribution G are repeated, suppose n draws of ϕ_i can take on $K < n$ distinct values. This defines a partition of the n draws into K clusters. The distribution over such partitions is a **Chinese restaurant process (CRP)** [13]. Imagine a restaurant with infinite number of tables, each table can seat infinite number of customers. The first customer sits at the first table. Later, the n-th customer either chooses an already occupied table k with probability $\frac{n_k}{\alpha+n-1}$, or chooses a new table with probability $\frac{\alpha}{\alpha+n-1}$. Here, n_k is the number of customers of table k. The Chinese restaurant process illustrates the cluster property of Dirichlet process, i.e., a new customer has higher probability of choosing a table with more customers, and thus only a limited number of tables will be occupied although the restaurant has infinite number of tables. This is also called the richer gets richer phenomenon where tables with more customers tend to attract more customers.

When we regard tables as clusters and customers as documents, the Chinese restaurant process turns to be a simple text stream clustering method, which only considers the size of the clusters when clustering a new document. While documents of each cluster are more important information to consider, and a document should have higher probability of choosing a cluster with more similar documents. By using Dirichlet process at the top of a mixture model, we can obtain the Dirichlet process mixture model for nonparametric clustering [6]. Yin and Wang [31] presented a collapsed Gibbs sampling algorithm for the Dirichlet process multinomial mixture (DPMM) model for static text clustering. As a further study, we propose a short text stream clustering algorithm based on the DPMM model.

The Polya urn scheme and Chinese restaurant process refer to draws from distribution G, while the **stick-breaking construction** shows the property of G explicitly:

$$G(\phi) = \sum_{k=1}^{\infty} \theta_k \delta(\phi - \phi_k), \qquad where\ \phi_k \sim G_0 \tag{2}$$

Here, $\delta(x) = 1$ if $x = 0$ and $\delta(x) = 0$ otherwise. The mixture weights $\theta = \{\theta_k\}_{k=1}^{\infty}$ can be constructed by $\theta \sim GEM(\gamma)$ [26].

The stick-breaking construction is significantly straightforward and can lead to novel inference techniques for Dirichlet process mixture models. We use the stick-breaking construction for the generative process of the Dirichlet process multinomial mixture (DPMM) model as follows:

$$\begin{aligned}
&\theta|\gamma \sim GEM(\gamma) \\
&\phi_k|\beta \sim Dir(\beta) \qquad k = 1, ..., \infty \\
&z_d|\theta \sim Mult(\theta) \qquad d = 1, ..., \infty \\
&d|z_d, \{\phi_k\}_{k=1}^{\infty} \sim p(d|\phi_{z_d})
\end{aligned}$$

The probability of document d generated by cluster z is defined as follows:

$$p(d|\phi_z) = \prod_{w \in d} Mult(w|\phi_z) \tag{3}$$

Here, we make the Naive Bayes assumption: the words in a document are generated independently when the document's cluster assignment z is known. We also assume that the probability of a word is independent of its position within the document.

4 APPROACH

In this section, we first introduce the representation of documents and clusters used in our methods. Then we propose the MStream algorithm which can work well in both the one pass scheme and batch scheme. We further propose an improved algorithm of MStream with forgetting rules called MStreamF, which can efficiently delete outdated documents.

4.1 Representation

Similarity-based clustering methods represent documents with the vector space model (VSM) [22], and use term frequency-inverse document frequency (TF-IDF) to measure the weights of words in the vector, which essentially assumes a document as a point in a high-dimentional space whose dimension is the length of the dictionary. Differently, we represent a document with only its words and the corresponding frequency in the document, which is more concise and straightforward. The essential difference is that VSM-based methods assume that a document is generated by the Gaussian distribution, while we assume that a document is generated by the multinomial distribution.

Similarity-based clustering methods like K-means [16] represent a cluster as the mean of its document vectors. In contrast, we represent a cluster with the cluster feature (CF) vector, which essentially is a big document combined with its documents. The definition of the CF vector of a cluster is defined as follows.

Definition 1. The cluster feature (CF) vector for a cluster z is defined as a tuple $\{\vec{n}_z, m_z, n_z\}$, where

- $\vec{n}_z$ contains a list of word frequencies in cluster z.
- m_z is the number of documents in cluster z.
- n_z is the number of words in cluster z.

The cluster feature (CF) vector presents important addible and deletable properties, as described next.

- **Addible Property.** A document d can be efficiently added to cluster z by updating its CF vector as follows.

$$\begin{aligned}
&n_z^w = n_z^w + N_d^w \qquad \forall w \in d \\
&m_z = m_z + 1 \\
&n_z = n_z + N_d
\end{aligned}$$

- **Deletable Property.** A document d can be efficiently deleted from cluster z by updating its CF vector as follows.

$$n_z^w = n_z^w - N_d^w \qquad \forall w \in d$$
$$m_z = m_z - 1$$
$$n_z = n_z - N_d$$

Here, N_d^w and N_d are the number of occurrences of word w in document d and the total number of words in document d, respectively, and $N_d = \sum_{w \in d} N_d^w$. Besides, n_z^w is the number of frequency of word w in cluster z. The complexity of adding a document to a cluster and deleting a document from a cluster are both $O(\bar{L})$, where $\bar{L}$ (often less than 10^2 for short text) is the average length of the documents. The addible and deletable properties of CF vectors are useful in the stream text clustering algorithm we will propose in the next section.

4.2 The MStream Algorithm

In this section, we propose a model-based clustering algorithm for short text streams called MStream, which has one pass clustering process and update clustering process of each batch. From the experimental study part, we can see MStream can achieve state-of-the-art performance with only one pass clustering process. The update clustering process can further improve the clustering performance when we allow multiple iterations of each batch.

One of the most important considerations of clustering text streams is to define the relationship of documents and clusters. Similarity-based stream clustering methods [1, 24] use metrics like cosine similarity to define the similarity between a document and a cluster. To deal with the concept drift problem, these methods usually choose a similarity threshold. When clustering a document, if its similarity between the closest cluster is larger than the threshold, it is assigned to this closest cluster. Otherwise, the document is assigned to a new cluster. However, it is hard to manually choose a proper similarity threshold in real applications.

Differently, we assume the documents are generated by the Dirichlet Process Multinational Mixture (DPMM) model introduced in Section 3. Yin and Wang [31] presented a collapsed Gibbs sampling algorithm for the DPMM model for static text clustering. As a further study, we propose a short text stream clustering algorithm based on the DPMM model. From the DPMM model, we can derive the probability of document d choosing an existing cluster z as follows:

$$p(z_d = z|\vec{z}_{\neg d}, \vec{d}, \alpha, \beta) \propto \frac{m_{z,\neg d}}{D - 1 + \alpha D} \frac{\prod_{w \in d} \prod_{j=1}^{N_d^w} (n_{z,\neg d}^w + \beta + j - 1)}{\prod_{i=1}^{N_d} (n_{z,\neg d} + V\beta + i - 1)} \tag{4}$$

Here, $\neg d$ means document d was removed from its current cluster feature (CF) vector, which is useful for the update clustering process of MStream. For a new coming document, $\neg d$ does not influence the CF vectors. Different from static text clustering [31], D is the number of current recorded documents and V is the size of the vocabulary of current recorded documents.

We can also derive the probability of document d choosing a new cluster as follows:

$$p(z_d = K + 1|\vec{z}_{\neg d}, \vec{d}, \alpha, \beta) \propto \frac{\alpha D}{D - 1 + \alpha D} \frac{\prod_{w \in d} \prod_{j=1}^{N_d^w} (\beta + j - 1)}{\prod_{i=1}^{N_d} (V\beta + i - 1)} \tag{5}$$

Different from static text clustering [31], we set $\gamma = \alpha D$, because the hyper parameter γ for generating mixture weights $\theta \sim GEM(\gamma)$ should be dynamic for text stream clustering. Here, αD is the pseudo number of documents in the new cluster, and β is the pseudo number of occurrences of each word in the new cluster.

The first part of Equation 4 and Equation 5 means that the probability of a document choosing a cluster is proportional to the number of documents in the cluster. A new document has higher probability of choosing a cluster with more documents, and thus only a limited number of clusters will be created although the number of clusters can be infinite. This is also known as the richer gets richer phenomenon where large clusters tend to attract more documents.

The second part of Equation 4 and Equation 5 actually defines the similarity between the document and the cluster. It is a product of N_d parts that correspond to the N_d words in document d. For each word w in document d, the corresponding part measures the the frequency of word w in cluster z. When a cluster has more documents that share same words with document d, the second part will be larger, and document d will be more likely to choose that cluster.

The detail of the MStream algorithm is shown in Algorithm 1. The MStream algorithm has one pass clustering process and update clustering process of each batch.

The **one pass clustering process of MStream** can be used to deal with the one pass scheme of text streams which assumes that the streaming documents come one by one, and we can process each document only one time. For the first document, it will choose a new cluster. The cluster feature (CF) vector of this new created cluster will be initialized with the first document. Later, a new coming document will choose one of the existing clusters or a new cluster according to corresponding probability computed with Equation 4 and Equation 5. When a new cluster is chosen, we create a new cluster to store the corresponding document. Otherwise, we add the corresponding document into the chosen existing cluster with the addible property.

The **update clustering process of MStream** can be used to deal with the batch scheme of text streams which assumes that the streaming documents come in batch, and we can process the document in each batch multiple times. When a new batch of documents comes, we first use the above one pass clustering process of MStream to obtain an initial clustering result with one iteration of the batch. Then we use the update clustering process of MStream to update the clustering results. For each document, we first delete it from its current cluster with the deletable property. Then, we reassign the document to a cluster according to the probability of the document belonging to each of the K existing clusters and a new cluster computed with Equation 4 and Equation 5. For the last iteration, we reassign each document to the cluster with the highest probability.

Algorithm 1: MStream($\vec{d}_t$)

Input : Documents $\vec{d}_t$ of batch t.
Output: Cluster assignments $\vec{z}_t$ of batch t.
begin
  //One pass clustering process
  for *d = 1 to* $|\vec{d}_t|$ **do**
    Compute the probability of document d choosing each of the K existing clusters and a new cluster.
    Sample cluster index z for document d according to the above $K+1$ probabilities.
    if $z == K+1$ **then**
      //A new cluster is chosen
      $K = K+1$
      Initialize m_K, n_K, and n_K^w as zero
    $m_z = m_z + 1$ and $n_z = n_z + N_d$
    for *each word* $w \in d$ **do**
      $n_z^w = n_z^w + N_d^w$
  //Update clustering process
  for *iter = 2 to I* **do**
    for *d = 1 to* $|\vec{d}_t|$ **do**
      Record the current cluster of d: $z = z_d$
      $m_z = m_z - 1$ and $n_z = n_z - N_d$
      for *each word* $w \in d$ **do**
        $n_z^w = n_z^w - N_d^w$
      Compute the probability of document d choosing each of the K existing clusters and a new cluster.
      if $iter < I$ **then**
        Sample cluster index z for document d according to the above $K+1$ probabilities.
      else
        Choose cluster index z for document d with the highest probability.
      if $z == K+1$ **then**
        //A new cluster is chosen
        $K = K+1$
        Initialize m_K, n_K, and n_K^w as zero
      $m_z = m_z + 1$ and $n_z = n_z + N_d$
      for *each word* $w \in d$ **do**
        $n_z^w = n_z^w + N_d^w$

The MStream algorithm always records the current K cluster feature (CF) vectors and the documents of the current batch. Then the space complexity of the MStream algorithm is $O(KV + M\bar{L})$, where K is the number of clusters, V is the size of the vocabulary, M is the number of documents in each batch, and $\bar{L}$ is the average length of the documents. The time complexity of computing the probability of document belonging to a cluster is linear to the average length of the documents $\bar{L}$. The time complexity of MStream with only one pass clustering process is $O(KN\bar{L})$, where N is the total number of documents in the stream. The time complexity of MStream with both one pass clustering process and update clustering process is $O(IKN\bar{L})$, where I is the number of iterations of each batch. As shown in Section 5.4, the update clustering process can apparently improve the performance of MStream, and MStream can achieve good performance with only two iterations of each batch.

4.3 The MStreamF Algorithm

As the number of clusters increases with more documents coming in, the space and time complexity of MStream will grow too large if we do not delete outdated clusters. Besides, we usually are interested in the information of a specific period of time, rather than the whole stream. Many data stream clustering methods [3, 10, 23] try to detect and delete outdated clusters regularly. However, it is difficult and inefficient to detect outdated clusters, because we cannot store documents of all clusters in memory, especially when some clusters grow big. Another choice is to directly delete outdated documents, which is more simple and efficient. However, we only store the documents of the current batch, and the documents of past batches are discarded. The intuition is that a cluster can be seen as a big document combined with its documents, and we can store the clusters of not outdated batches. When the documents of batch b are outdated, we can delete documents of batch b from the current cluster feature (CF) vectors with the following subtraction operation.

Definition 2. Subtraction Operation For each cluster z of batch b, the subtraction operation of its documents from the current CF vectors is given by

$$\vec{n}_z = \vec{n}_z - \vec{n}_{b,z}$$
$$m_z = m_z - m_{b,z}$$
$$n_z = n_z - n_{b,z}$$

Here $\{\vec{n}_z, m_z, n_z\}$ is the current CF vector of cluster z, and $\{\vec{n}_{b,z}, m_{b,z}, n_{b,z}\}$ is the CF vector of cluster z of batch b.

Based on the above intuition, we propose an improved algorithm of MStream with forgetting rules, called MStreamF. The detail of MStreamF is shown in Algorithm 2. MStreamF has a parameter B_s which means the number of batches we store, and the current CF vectors record documents of the latest B_s batches. When the number of batches stored is larger than B_s, we should delete the CF vectors of the oldest batch with the subtraction operation before clustering documents of a new batch with the MStream algorithm.

As the arriving of more batches of documents, we can detect new clusters and deleting documents of old batches. Some clusters turn into empty because their documents were outdated and deleted in this process. When a cluster gets empty, the probability of later documents choosing this cluster is zero, which means no document will choose this cluster anymore. For the convince of further analysis, we do not reuse cluster IDs of past clusters.

Many data stream clustering methods [3, 23] store the clustering results at particular moments of the stream on disk for further analysis. Because they delete outdated clusters rather than outdated documents in the stream clustering period. As we delete outdated documents by deleting CF vectors of outdated batch, we cannot obtain the clustering results of a period by the clustering results of its start and end moments. We can store the cluster feature (CF) vectors of each batch on disk and obtain the clustering results of a specific period. This is a balance of the efficiency of online clustering and offline analysis.

Algorithm 2: MStreamF

Input : Short text streams $\vec{d}_1, \vec{d}_2, \ldots, \vec{d}_t, \ldots$
Number of stored batches B_s
Output: Cluster assignments of each batch $\vec{z}_1, \vec{z}_2, \ldots, \vec{z}_t, \ldots$
begin
  t = 0 // t records the ordinal number of batches.
  while *!stream.end()* **do**
    t = t + 1
    $\vec{d}_t$ = stream.next()
    //If the number of stored batches is larger than B_s, we delete the oldest batch from current CF vectors.
    if $t > B_s + 1$ **then**
      $b = t - B_s - 1$
      for *each cluster* $z \in$ *batch* b **do**
        $\vec{n}_z = \vec{n}_z - \vec{n}_{b,z}$
        $m_z = m_z - m_{b,z}$
        $n_z = n_z - n_{b,z}$
      Remove the CF vectors of the oldest batch b.
    //Initialize CF vectors of bath t with current CF vectors.
    $\vec{n}_{t,z} = \vec{n}_z$
    $m_{t,z} = m_z$
    $n_{t,z} = n_z$
    //Clustering documents of batch t with MStream.
    MStream($\vec{d}_t$)
    //Compute CF vectors of batch t
    for *each cluster* $z \in batcht$ **do**
      $\vec{n}_{t,z} = \vec{n}_z - \vec{n}_{t,z}$
      $m_{t,z} = m_z - m_{t,z}$
      $n_{t,z} = n_z - n_{t,z}$

The MStreamF algorithm always records the current K cluster feature (CF) vectors, the clusters of B_s stored batches, and the documents of the current batch. Then the space complexity of MStreamF is $O(B_s\bar{K}V + M\bar{L})$, where B_s is the number of stored batches, $\bar{K}$ is the average number of the clusters of each batch, V is the size of the vocabulary, M is the number of documents in each batch, and $\bar{L}$ is the average length of the documents. The time complexity of MStreamF is $O(B\bar{K}V + IKN\bar{L})$, where B is the number of deleted batches, I is the number of iterations of each batch. We should note that MStreamF can achieve good and stable performance with few iterations. Besides, the number of current clusters K of MStreamF can be smaller than that of MStream, because of the forgetting rules of MStreamF.

5 EXPERIMENTAL STUDY

5.1 Experimental Setup

5.1.1 Datasets. We use two real short text datasets and two variants of them in the experimental study:

- **Tweets**. This dataset consists of 30,322 tweets that are highly relevant to 269 queries in the TREC 2011-2015 microblog track [1] . The average length of the documents in this dataset is 7.97.

[1] http://trec.nist.gov/data/microblog.html

Dataset	D	K	V	Avg Len
Tweets & Tweets-T	30,322	269	12,301	7.97
News & News-T	11,109	152	8,110	6.23

Table 1: Statistics of the text datasets (D: Number of documents, K: Number of clusters, V: Vocabulary size, Avg Len: Average length of the documents)

- **News**. This dataset was first used in [30], which consists of 11,109 news titles belonging to 152 clusters. The average length of the documents in this dataset is 6.23.
- **Tweets-T and News-T**. In the real world, the situation occurs frequently where topics appear only on a certain time window and disappear after that. So we sort Tweets and News by topics to obtain two new datasets. We further divide them into 16 equal parts and shuffle each part separately.

The preprocessing step includes converting all letters into lowercase, removing stop words, and stemming. Table 1 shows the statistics of these text datasets after preprocessing We can see that the average length indicates the two datasets is suitable for short streaming clustering.

5.1.2 Evaluation Metric. The Normalized Mutual Information (NMI) [25] is widely used to evaluate the quality of the clustering results. NMI measures the amount of statistical information shared by the random variables representing the cluster assignments and the ground truth groups of the documents. Normalized Mutual Information (NMI) is formally defined as follows :

$$NMI = \frac{\sum_{c,k} n_{c,k} \log(\frac{N \cdot n_{c,k}}{n_c \cdot n_k})}{\sqrt{(\sum_c n_c \log \frac{n_c}{N})(\sum_k n_k \log \frac{n_k}{N})}} \tag{6}$$

where n_c is the number of documents in class c, n_k is the number of documents in cluster k, $n_{c,k}$ is the number of documents in class c as well as in cluster k, and N is the number of documents in the dataset. When the clustering results perfectly match the ground truth classes, the NMI value will be one. While when the clustering results are randomly generated, the NMI value will be close to zero.

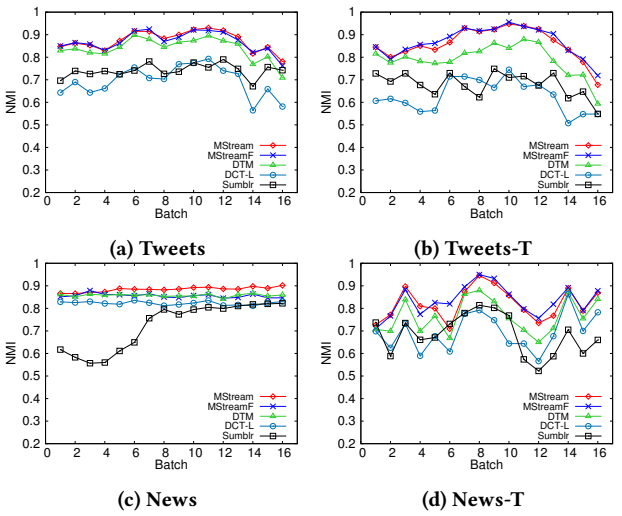
5.1.3 Methods for Comparison. We compare MStream and MStreamF with the following state-of-the-art algorithms:

- **DTM**. Dynamic topic models [8] are generative models that can be used to analyze the evolution of (unobserved) topics of a collection of documents over time. This family of models is an extension to Latent Dirichlet Allocation (LDA) that can handle sequential documents.
- **DCT-L**. Dynamic clustering topic model (DCT) [18] enables tracking the time-varying distributions of topics over documents and words over topics. Long-term dependency DCT (DCT-L) can capture long-term trends in topics.
- **Sumblr**. Sumblr [23] proposes an online tweet stream clustering algorithm which is able to efficiently cluster the tweets and maintain compact cluster statistics, with only one pass of the stream.

Without specification, we use MStream to deal with Tweets and News datasets and MStreamF to deal with Tweets-T and News-T datasets. For MStream and MStreamF, we set $\alpha = 0.03$, $\beta = 0.03$, and the number of iterations to 10, and the maximum number of stored batches is set to one for MStreamF. We set α to 0.01 for DTM,

	Tweets	Tweets-T	News	News-T
MStream	**.844** ± .002	.882 ± .004	**.834** ± .004	.850 ± .004
MStreamF	.823 ± .005	**.923** ± .003	.797 ± .003	**.873** ± .003
DTM	.801 ± .002	.802 ± .001	.793 ± .002	.806 ± .002
DCT-L	.697 ± .002	.669 ± .005	.733 ± .002	.744 ± .004
Sumblr	.689 ± .001	.695 ± .003	.575 ± .005	.720 ± .002

Table 2: NMI results of different methods.



(a) Tweets **(b) Tweets-T**

(c) News **(d) News-T**

Figure 1: NMI results of different methods on each batch.

initial α and β at 1 and 0.1 respectively for DCT-L and β to 0.02 for Sumblr. For DTM, DCT-L and Sumblr, in which the number of topics should be specified before the experiment, we set it at 300 and 170 for Tweets and News datasets respectively. Without specification, we set the maximum number of iterations of each batch to 10 and run 10 independent trials for each method.

5.2 Comparison with Existing Methods

In this part, we compare the performance of MStream and MStreamF with DTM [8], DCT-L [18] and Sumblr [23]. For each algorithm, we report the mean and standard deviation of the NMI of the overall results in Table 2 and the NMI of the results for each 16 batches in Figure 1. To make a comparison between MStream and MStreamF, we report the number of stored clusters when dealing with each batch in Figure 2.

From Table 2, we can see that MStream and MStreamF always achieve the highest performance compared with the other three clustering methods on all the datasets. Meanwhile, the standard deviations of the 10 independent trials of MStream and MStreamF are not really big which means they have high consistency. By comparing MStream and MStreamF, we find that the former performs better on the normal datasets and the latter performs better on the datasets organized by topics, which proves our forgetting strategy works well.

From Figure 1, we can see that the NMI in each batch is fluctuant, indicating that there are many different cases, which means that the batches in each dataset are representative. Generally speaking, MStream and MStreamF perform better in each batch of each dataset, showing that both of them can adapt to many situations.

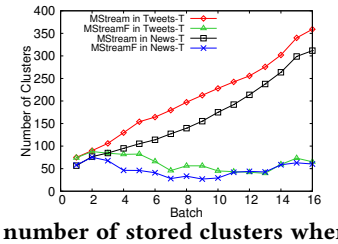
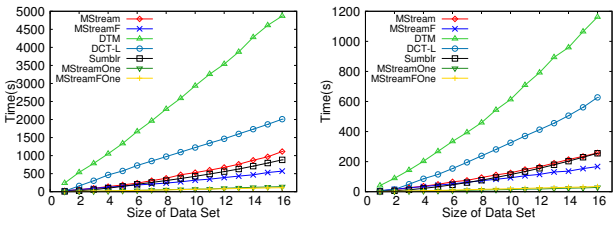


Figure 2: The number of stored clusters when dealing with each batch.



(a) Speed on Tweets **(b) Speed on News**

Figure 3: Speed of different methods.

Figure 2 shows the number of stored clusters when dealing with each batch found by MStream and MStreamF. From Figure 2, we can see that MStream and MStreamF get the same number of clusters on the first and next batches. But in dealing with the subsequent batches, MStreamF will stored not too many clusters by forgetting batches existing for too long. MStream, on the contrary, gets more and more clusters over time. Therefore, for Tweets-T and News-T, MStreamF will not only perform better, but run faster and take up less storage space.

5.3 Speed of the Algorithms

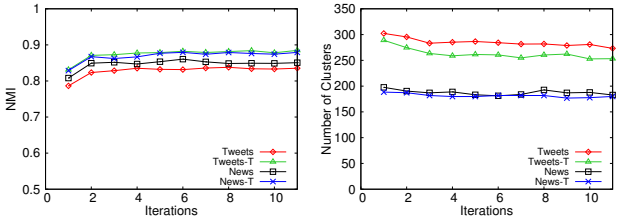
In this part, we try to investigate the speed of MStream, MStreamF and other methods. MStream, MStreamF and Sumblr were implemented in Python. DTM and DCT were implemented in C++. All algorithms were run on a Linux server with Intel Xeon X5690 3.47GHz CPU and 94GB memory. We set the number of iterations of each batch to 10 for all methods (except Sumblr) to make a fair comparison. Sumblr has only one pass of each batch. MStreamOne and MStreamFOne are the versions of MStream and MStreamF with only the one pass clustering process.

Figure 3 shows the speed of different methods with different size of datasets. We can see that the speed of these methods are approximately linear to the size of datasets, MStreamOne and MStreamFOne are apparently faster than other methods. We can also see that MStreamF is faster than MStream, the reason is that MStreamF keeps less clusters than MStream by deleting outdated batches.

5.4 Influence of the Number of Iterations

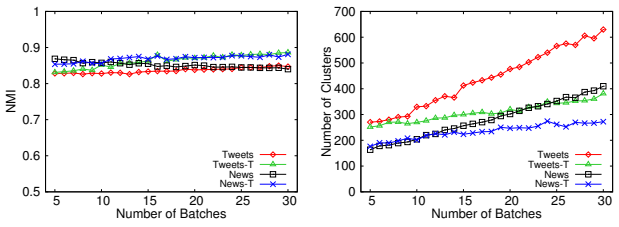
In this part, we try to investigate the influence of the number of iterations of each batch to the performance and the number of clusters found by MStream and MStreamF. We use MStream to deal with Tweets and News datasets and MStreamF to deal with Tweets-T and News-T datasets.

Figure 4a shows the performance with different number of iterations of each batch. From Figure 4a, we can see that MStream and MStreamF can achieve state-of-the-art performance with only



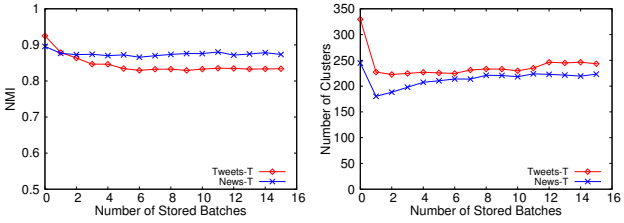
(a) Peformance (b) Number of clusters

Figure 4: Influence of the number of iterations of each batch.



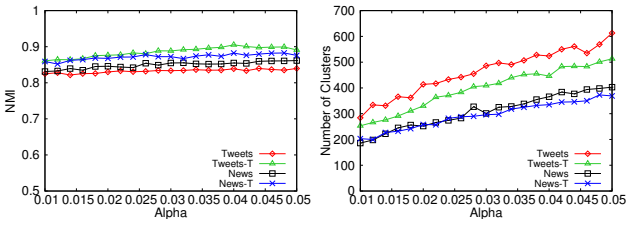
(a) Peformance (b) Number of clusters

Figure 5: Influence of the number of batches.



(a) Performance (b) Number of clusters

Figure 6: Influence of the most number of stored batches.



(a) Performance (b) Number of clusters

Figure 7: Influence of Alpha.

the one pass clustering process of the stream, which is the first iteration of each batch. Besides, the update clustering process can improve the performance quickly and apparently, experimentally the second iteration of each batch. We can see that MStream and MStreamF get stable performance within about 2 iterations. This shows that the two methods are both fast to converge.

Figure 4b shows the number of clusters found by MStream and MStreamF with different number of iterations of each batch. From Figure 4b, we can see that the number of clusters drops with more iterations of each batch, this means the update clustering process can amend the result of the one pass clustering process. We also can see that, after about three iterations, the number of clusters found on each datasets is near the true number of clusters in the datasets.

5.5 Influence of the Number of Batches

In this part, we try to investigate the influence of the number of batches to the performance and the number of clusters found by MStream and MStreamF. We use MStream to deal with Tweets and News datasets and MStreamF to deal with Tweets-T and News-T datasets. The number of batches ranges from 5 to 30, which means the size of batches ranges from 1000 to 6000 on Tweets and from 370 to 2200 on News respectively.

Figure 5a shows the performance with different number of batches. From Figure 5a, we can see that MStream and MStreamF can handle different situations, where the size of batches varies from hundred to thousand magnitudes, using constant parameters and resulting good performance. We can also see that, in most cases, the performance grows with the number of batches, which means this method performs better on smaller size of batches.

Figure 5b shows the number of clusters found by MStream and MStreamF with different number of batches. From Figure 5b, we can see that the number of clusters grows with the number of batches, which means the number of clusters increases as the size of batches becomes smaller. But for datasets arranged by topics, this trend is relatively flat.

5.6 Influence of the Maximum Number of Stored Batches

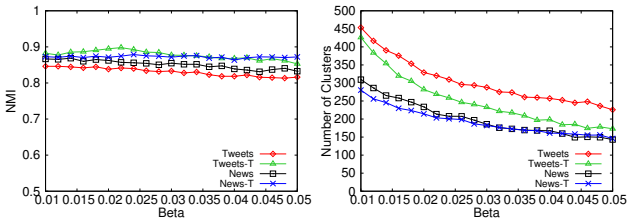
In this part, we try to investigate the influence of the number of stored batches to the number of clusters and performance found by MStreamF on Tweets-T and News-T. The number of batches is 16. The number of stored batches ranges from 0 to 15.

Figure 6 shows the number of clusters and performance with different number of stored batches. From Figure 6a, we can see that NMI drops as the number of stored batches increases. Due to the way of construction of Tweets-T and News-T, most topics only appear in a specific batch, so MStreamF will perform better if it stores less batches. But from Figure 6b, we can see that the number of clusters found by MStreamF is large when no batch is stored and suddenly drops at one, then increases as the number of stored batches increases. So storing zero batch is not a good solution. A compromise to deal with this situation is storing one batch to get NMI high enough and find not too many clusters.

5.7 Influence of Alpha

In this part, we try to investigate the influence of α to the performance and the number of clusters found by MStream and MStreamF. We use MStream to deal with Tweets and News datasets and MStreamF to deal with Tweets-T and News-T datasets. The range of α is from 0.01 to 0.05.

Figure 7a shows the performance of MStream and MStreamF with different values of α. From Figure 7a, we can see MStream and MStreamF can achieve stable performance with different α on these datasets. Figure 7b shows the number of clusters found by MStream and MStreamF with different values of α. An observation is that the number of clusters found by MStream and MStreamF increases with α. The reason is that α is the pseudo number of documents



(a) Performance **(b) Number of clusters**

Figure 8: Influence of Beta.

in each cluster, and the probability of the document choosing the potential cluster grows with α.

5.8 Influence of Beta

In this part, we try to investigate the influence of β to the peformance and the number of clusters found by MStream and MStreamF. We use MStream to deal with Tweets and News datasets and MStreamF to deal with Tweets-T and News-T datasets. The range of β is from 0.01 to 0.05.

Figure 8a shows the performance of MStream and MStreamF with different values of β. From Figure 8a, we can see MStream and MStreamF can achieve stable performance with different β on these datasets. Figure 8b shows the number of clusters found by MStream and MStreamF with different values of β. An observation is that the number of clusters found drops when β gets larger. The reason is that β is the pseudo frequency of each word in each cluster, and the probability of a document choosing a cluster is less sensitive to the similarity between the documents and the clusters when β gets larger. As a result, "richer gets richer" property makes MStream and MStreamF get fewer clusters.

6 CONCLUSION

In this paper, we first propose a short text stream clustering algorithm based on the Dirichlet process multinomial mixture model, call MStream, which can deal with the concept drift problem and sparsity problem naturally. The MStream algorithm can achieve state-of-the-art performance with only one pass of the stream, and can have even better performance when we allow multiple iterations of each batch. We propose an improved algorithm of MStream with forgetting rules called MStreamF, which can efficiently delete outdated documents by deleting clusters of outdated batches. Our extensive experimental study shows that MStream and MStreamF can achieve better performance than three baselines on real datasets. As future work we intent to use the proposed methods to improve the performance of other related applications such as search result diversification, event detection and tracking, and text summarization in the context of short text streams.

ACKNOWLEDGMENTS

This work was supported in part by National Basic Research 973 Program of China under Grant No. 2015CB352502 and 2014CB340505, National Natural Science Foundation of China under Grant No. 61702190, 61532010, and 61521002.

REFERENCES

[1] Charu C Aggarwal. 2013. A Survey of Stream Clustering Algorithms. (2013).
[2] Charu C Aggarwal and S Yu Philip. 2010. On clustering massive text and categorical data streams. *Knowledge and information systems* 24, 2 (2010), 171–196.
[3] Charu C Aggarwal, S Yu Philip, Jiawei Han, and Jianyong Wang. 2003. -A Framework for Clustering Evolving Data Streams. In *VLDB*. Elsevier, 81–92.
[4] Amr Ahmed and Eric Xing. 2008. Dynamic non-parametric mixture models and the recurrent chinese restaurant process: with applications to evolutionary clustering. In *SDM*. SIAM, 219–230.
[5] Hesam Amoualian, Marianne Clausel, Eric Gaussier, and Massih-Reza Amini. 2016. Streaming-lda: A copula-based approach to modeling topic dependencies in document streams. In *SIGKDD*. ACM, 695–704.
[6] Charles E Antoniak. 1974. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *The annals of statistics* (1974), 1152–1174.
[7] David Blackwell and James B MacQueen. 1973. Ferguson distributions via Pólya urn schemes. *The annals of statistics* (1973), 353–355.
[8] David M Blei and John D Lafferty. 2006. Dynamic topic models. In *ICML*. ACM, 113–120.
[9] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* (2003). http://dl.acm.org/citation.cfm?id=944919.944937
[10] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. 2006. Density-based clustering over an evolving data stream with noise. In *SDM*. SIAM, 328–339.
[11] Arnaud Doucet, Nando De Freitas, and Neil Gordon. 2001. An introduction to sequential Monte Carlo methods. In *Sequential Monte Carlo methods in practice*. Springer, 3–14.
[12] Nan Du, Mehrdad Farajtabar, Amr Ahmed, Alexander J Smola, and Le Song. 2015. Dirichlet-hawkes processes with applications to clustering continuous-time document streams. In *SIGKDD*. ACM, 219–228.
[13] Thomas S Ferguson. 1973. A Bayesian analysis of some nonparametric problems. *The annals of statistics* (1973), 209–230.
[14] Hemant Ishwaran and Lancelot F James. 2001. Gibbs sampling methods for stick-breaking priors. *J. Amer. Statist. Assoc.* 96, 453 (2001), 161–173.
[15] Tomoharu Iwata, Shinji Watanabe, Takeshi Yamada, and Naonori Ueda. 2009. Topic Tracking Model for Analyzing Consumer Purchase Behavior.. In *IJCAI*, Vol. 9. 1427–1432.
[16] Anil K. Jain. 2010. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters* 31, 8 (2010), 651–666.
[17] Argyris Kalogeratos, Panagiotis Zagorisios, and Aristidis Likas. 2016. Improving text stream clustering using term burstiness and co-burstiness. In *SETN*. ACM, 16.
[18] Shangsong Liang, Emine Yilmaz, and Evangelos Kanoulas. 2016. Dynamic clustering of streaming short documents. In *SIGKDD*. ACM, 995–1004.
[19] Alireza Rezaei Mahdiraji. 2009. Clustering data stream: A survey of algorithms. *International Journal of Knowledge-based and Intelligent Engineering Systems* 13, 2 (2009), 39–44.
[20] Hai-Long Nguyen, Yew-Kwong Woon, and Wee-Keong Ng. 2015. A survey on data stream clustering and classification. *Knowledge and information systems* 45, 3 (2015), 535–569.
[21] Kamal Nigam, Andrew McCallum, Sebastian Thrun, and Tom M. Mitchell. 2000. Text Classification from Labeled and Unlabeled Documents using EM. *Machine Learning* 39, 2/3 (2000), 103–134.
[22] Gerard Salton, A. Wong, and C. S. Yang. 1975. A Vector Space Model for Automatic Indexing. *Commun. ACM* 18, 11 (1975), 613–620.
[23] Lidan Shou, Zhenhua Wang, Ke Chen, and Gang Chen. 2013. Sumblr: continuous summarization of evolving tweet streams. In *SIGIR*. ACM, 533–542.
[24] Jonathan A Silva, Elaine R Faria, Rodrigo C Barros, Eduardo R Hruschka, Andre CPLF De Carvalho, and João Gama. 2013. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)* 46, 1 (2013), 13.
[25] Alexander Strehl and Joydeep Ghosh. 2003. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research* 3 (2003), 583–617.
[26] Yee Whye Teh. 2010. Dirichlet process. In *Encyclopedia of machine learning*. Springer, 280–287.
[27] Xuerui Wang and Andrew McCallum. 2006. Topics over time: a non-Markov continuous-time model of topical trends. In *SIGKDD*. ACM, 424–433.
[28] Yu Wang, Eugene Agichtein, and Michele Benzi. 2012. TM-LDA: efficient online modeling of latent topic transitions in social media. In *SIGKDD*. ACM, 123–131.
[29] Xing Wei, Jimeng Sun, and Xuerui Wang. 2007. Dynamic Mixture Models for Multiple Time-Series.. In *IJCAI*, Vol. 7. 2909–2914.
[30] Jianhua Yin and Jianyong Wang. 2014. A dirichlet multinomial mixture model-based approach for short text clustering. In *SIGKDD*. ACM, 233–242.
[31] Jianhua Yin and Jianyong Wang. 2016. A model-based approach for text clustering with outlier detection. In *ICDE*. IEEE, 625–636.
[32] Shinjae Yoo, Hao Huang, and Shiva Prasad Kasiviswanathan. 2016. Streaming spectral clustering. In *ICDE*. IEEE, 637–648.
[33] Shi Zhong. 2005. Efficient streaming text clustering. *Neural Networks* 18, 5-6 (2005), 790–798.