Chapter 9

MINING TEXT STREAMS

Charu C. Aggarwal
IBM T. J. Watson Research Center
Hawthorne, NY 10532, USA
charu@us.ibm.com

Abstract The large amount of text data which are continuously produced over time in a variety of large scale applications such as social networks results in massive streams of data. Typically massive text streams are created by very large scale interactions of individuals, or by structured creations of particular kinds of content by dedicated organizations. An example in the latter category would be the massive text streams created by news-wire services. Such text streams provide unprecedented challenges to data mining algorithms from an efficiency perspective. In this chapter, we review text stream mining algorithms for a wide variety of problems in data mining such as clustering, classification and topic modeling. We also discuss a number of future challenges in this area of research.

Keywords: Text Mining, Data Streams

1. Introduction

Text streams have become ubiquitous in recent years because of a wide variety of applications in social networks, news collection, and other forms of activity which result in the continuous creation of massive streams. Some specific examples of applications which create text streams are as follows:

- In social networks, users continuously communicate with one another with the use of text messages. This results in massive volumes of text streams which can be leveraged for a variety of mining and search purposes in the social network. This is because the text

C.C. Aggarwal and C.X. Zhai (eds.), *Mining Text Data*, DOI 10.1007/978-1-4614-3223-4_9,

messages are reflective of user interests. A similar observation applies to chat and email networks.

- Many news aggregator services [1] may receive large volumes of news articles continuously over time. Such articles are often longer and more well structured than the kinds of messages which are seen in social, chat, or email networks.

- Many web crawlers may collect a large volume of documents from networks in a small time frame. In many cases, such documents are restricted to those which have been modified in a small time frame. This naturally results in a stream of modified documents.

Text streams create a huge challenge from the perspective of a wide variety of mining applications, because of the massive volume of the data which must be processed in online fashion. Data streams have been studied extensively in recent years not just in the text domain, but also in the context of a wide variety of multi-dimensional applications such as numerical and categorical data. A detailed discussion of mining algorithms for stream data may be found in [1]. While many of the techniques proposed for multi-dimensional data [1] can be generalized to text data at the high level, the details can be quite different because of the very different format and lack of structure of text data. Since stream mining techniques are generally dependent upon summarization, it follows that methods for online summarization need to be designed which work well for the unstructured nature of text data.

In the case of multi-dimensional and time-series data, such summarization often takes the form of methods such as histograms, wavelets, and sketches which can be used in order to create a structured summary of the underlying data [1]. However, the unstructured nature of text makes the use of such summaries quite challenging. While sketches have been used to some effect in the text domain [23], it has generally been difficult to generalize wavelet and histogram methods to the text domain. As we will see later in this section, the summarization methods designed for text streaming problems may vary a lot, and may often need to be tailored to the problem at hand. One of the goals of this chapter is to provide a broad spectrum of the different methods which are used for text mining, which can provide an overview of the tools which can be most effectively used for the text stream scenario. We will also present the future challenges and research directions associated with text stream mining.

[1] An example would be the *Google News* service.

This chapter is organized as follows. In section 2, we will present a variety of the well known algorithms for clustering text streams. This includes popular methods for topic detection and tracking in text stream. This is because the process of event detection is closely related to the clustering problem. The methods for classification of text streams are reviewed in section 3. Section 4 presents methods for evolution analysis of text stream. The conclusions and summary are presented in section 5.

2. Clustering Text Streams

The problem of clustering text streams has been widely studied in the context of numerical data [1, 4, 12]. Other popular methods which have been studied in the machine learning literature include the COBWEB and CLASSIT methods [20, 25]. The COBWEB algorithm assumes nominal attributes, whereas the CLASSIT algorithm assumes real-valued attributes. Many of these methods [4, 12] are extensions of the k-means method as extended to the stream scenario. This trend has also been applied to the case of text streams. One of the earliest methods for streaming text clustering is discussed in [54]. This technique is referred to as the *Online Spherical k-Means Algorithm (OSKM)*, which reflects the broad approach used by the methodology. This techniques divides up the incoming stream into small segments, each of which can be processed effectively in main memory. A set of k-means iterations are applied to each such data segment in order to cluster them. The advantage of using a segment-wise approach for clustering is that since each segment can be held in main memory, we can process each data point multiple times as long as it is held in main memory. In addition, the centroids from the previous segment are used in the next iteration for clustering purposes. A decay factor is introduced in order to age-out the old documents, so that the new documents are considered more important from a clustering perspective. This approach has been shown to be extremely effective in clustering massive text streams in [54].

The method in [54] is designed as a flat clustering algorithm, in which there is a single level to the clustering process. In many applications, it is useful to design hierarchical clustering algorithms in which different levels of the clustering can be explored. In the context of text data, this implies that different levels of topics and subtopics can be explored with the use of a hierarchical clustering process. A distributional modeling method for hierarchical clustering of streaming documents has been proposed in [37]. The method extends the COBWEB and CLASSIT algorithms [20, 25] to the case of text data. The work in [37] studies the

different kinds of distributional assumptions of words in documents. We note that these distributional assumptions are required to adapt these algorithms to the case of text data. The approach essentially changes the distributional assumption so that the method can work effectively for text data.

A different method for clustering massive text and categorical data streams is discussed in [3]. The method discussed in [3] uses an approach which examines the relationship between outliers, emerging trends, and clusters in the underlying data. Old clusters may become inactive, and eventually get replaced by new clusters. Similarly, when newly arriving data points do not naturally fit in any particular cluster, these need to be initially classified as outliers. However, as time progresses, these new points may create a distinctive pattern of activity which can be recognized as a new cluster. The temporal locality of the data stream is manifested by these new clusters. For example, the first web page belonging to a particular category in a crawl may be recognized as an outlier, but may later form a cluster of documents of its own. On the other hand, the new outliers may not necessarily result in the formation of new clusters. Such outliers are true short-term abnormalities in the data since they do not result in the emergence of sustainable patterns. The approach discussed in [3] recognizes new clusters by first recognizing them as outliers.

This approach works with the use of a summarization methodology, which is motivated by the micro-clustering approach proposed in [4]. While the concept of micro-clustering was designed for numerical data, it can also be extended to the case of text and categorical data streams. This methodology essentially creates summaries from the data points which are used in order to estimate the assignment of incoming data points to clusters. The concept of micro-clusters is generalized to that of *condensed droplets* in [3].

In order to ensure greater importance of more recent data, a time-sensitive weightage is assigned to each data point. It is assumed that each data point has a time-dependent weight defined by the function $f(t)$. The function $f(t)$ is also referred to as the *fading function*. The fading function $f(t)$ is a non-monotonic decreasing function which decays uniformly with time t. In order to formalize this concept, we will define the *half-life* of a point in the data stream.

DEFINITION 9.1 *The half life t_0 of a point is defined as the time at which $f(t_0) = (1/2)f(0)$.*

Conceptually, the aim of defining a half life is to quantify the rate of decay of the importance of each data point in the stream clustering

process. The *decay-rate* is defined as the inverse of the half life of the data stream. We denote the decay rate by $\lambda = 1/t_0$. We denote the weight function of each point in the data stream by $f(t) = 2^{-\lambda \cdot t}$. From the perspective of the clustering process, the weight of each data point is $f(t)$. It is easy to see that this decay function creates a half life of $1/\lambda$. It is also evident that by changing the value of λ, it is possible to change the rate at which the importance of the historical information in the data stream decays. The higher the value of λ, the lower the importance of the historical information compared to more recent data. For more stable data streams, it is desirable to pick a smaller value of λ, whereas for rapidly evolving data streams, it is desirable to pick a larger value of λ.

When a cluster is created during the streaming process by a newly arriving data point, it is allowed to remain as a trend-setting outlier for at least one half-life. During that period, if at least one more data point arrives, then the cluster becomes an active and mature cluster. On the other hand, if no new points arrive during a half-life, then the trend-setting outlier is recognized as a true anomaly in the data stream. At this point, this anomaly is removed from the list of current clusters. We refer to the process of removal as *cluster death*. Thus, a new cluster containing one data point dies when the (weighted) number of points in the cluster is 0.5. The same criterion is used to define the death of mature clusters. A necessary condition for this criterion to be met is that the inactivity period in the cluster has exceeded the half life $1/\lambda$. The greater the number of points in the cluster, the greater the level by which the inactivity period would need to exceed its half life in order to meet the criterion. This is a natural solution, since it is intuitively desirable to have stronger requirements (a longer inactivity period) for the death of a cluster containing a larger number of points.

Next, we describe the process of creation of a condensed-droplet from the underlying text stream. An important point to remember is that a text data set can be treated as a *sparse numeric* data set. This is because most documents contain only a small fraction of the vocabulary with non-zero frequency. For a cluster of documents $\mathcal{C}$ at time t, we denote the corresponding condensed droplet by $\mathcal{D}(t, \mathcal{C})$.

DEFINITION 9.2 *A cluster droplet $\mathcal{D}(t, \mathcal{C})$ for a set of text data points $\mathcal{C}$ at time t is defined to as a tuple $(\overline{DF2}, \overline{DF1}, n, w(t), l)$. Each tuple component is defined as follows:*

- *The vector $\overline{DF2}$ contains $3 \cdot wb \cdot (wb-1)/2$ entries. Here wb is the number of distinct words in the cluster $\mathcal{C}$. For each pair of dimensions, we maintain a list of the pairs of word ids with non-*

zero counts. We also maintained the sum of the weighted counts for such word pairs.

- *The vector $\overline{DF1}$ contains $2 \cdot wb$ entries. We maintain the identities of the words with non-zero counts. In addition, we maintain the sum of the weighted counts for each word occurring in the cluster.*
- *The entry n contains the number of data points in the cluster.*
- *The entry $w(t)$ contains the sum of the weights of the data points at time t. We note that the value $w(t)$ is a function of the time t and decays with time unless new data points are added to the droplet $\mathcal{D}(t)$.*
- *The entry l contains the time stamp of the last time that a data point was added to the cluster.*

The concept of cluster droplet has some interesting properties that will be useful during the maintenance process. These properties relate to the additivity and decay behavior of the cluster droplet.

OBSERVATION 2.1 *Consider the cluster droplets $\mathcal{D}(t, \mathcal{C}_1) = (\overline{DF2_1}, \overline{DF1_1}, n_1, w(t)_1, l_1)$ and $\mathcal{D}(t, \mathcal{C}_2) = (\overline{DF2_2}, \overline{DF1_2}, n_2, w(t)_2, l_2)$. Then the cluster droplet $\mathcal{D}(t, \mathcal{C}_1 \cup \mathcal{C}_2)$ is defined by the tuple $(\overline{DF2_1} + \overline{DF2_2}, \overline{DF1_1} + \overline{DF1_2}, n_1 + n_2, w(t)_1 + w(t)_2, max\{l_1, l_2\})$.*

The cluster droplet for the union of two clusters is the sum of individual entries. The only exception is the last entry which is the maxima of the two last-update times. We note that the additivity property provides considerable convenience for data stream processing since the entries can be updated efficiently using simple additive operations.

The second observation relates to the rate of decay of the condensed droplets. Since the weights of each data point decay with the passage of time, the corresponding entries also decay at the same rate. Correspondingly, we make the following observation:

OBSERVATION 2.2 *Consider the cluster droplet $\mathcal{D}(t, \mathcal{C}) = (\overline{DF2}, \overline{DF1}, n, w(t), l)$. Then the entries of of the same cluster droplet $\mathcal{C}$ at a time $t' > t$ are given by $\mathcal{D}(t', \mathcal{C}) = (\overline{DF2} \cdot 2^{-\lambda \cdot (t'-t)}, \overline{DF1} \cdot 2^{-\lambda \cdot (t'-t)}, n, w(t) \cdot 2^{-\lambda \cdot (t'-t)}, l)$.*

The second observation is critical in regulating the rate at which the cluster droplets are updated during the clustering process. Since all cluster droplets decay at essentially the same rate (unless new data points are added), it follows that it is not necessary to update the decay statistics

at each time stamp. Rather, each cluster droplet can be updated lazily, whenever new data points are added to it.

The overall algorithm proceeds as follows. At the beginning of algorithmic execution, we start with an empty set of clusters. As new data points arrive, unit clusters containing individual data points are created. Once a maximum number k of such clusters have been created, we can begin the process of online cluster maintenance. Thus, we initially start off with a trivial set of k clusters. These clusters are updated over time with the arrival of new data points.

When a new data point $\overline{X}$ arrives, its similarity to each cluster droplet is computed. In the case of text data sets, the cosine similarity measure [17, 40] between $\overline{DF1}$ and $\overline{X}$ is used. The similarity value $S(\overline{X}, \mathcal{C}_j)$ is computed from the incoming document $\overline{X}$ to every cluster $\mathcal{C}_j$. The cluster with the maximum value of $S(\overline{X}, \mathcal{C}_j)$ is chosen as the relevant cluster for data insertion. Let us assume that this cluster is $\mathcal{C}_{mindex}$. We use a threshold denoted by $thresh$ in order to determine whether the incoming data point is an outlier. If the value of $S(\overline{X}, \mathcal{C}_{mindex})$ is larger than the threshold $thresh$, then the point $\overline{X}$ is assigned to the cluster $\mathcal{C}_{mindex}$. Otherwise, we check if some inactive cluster exists in the current set of cluster droplets. If no such inactive cluster exists, then the data point $\overline{X}$ is added to $\mathcal{C}_{mindex}$. On the other hand, when an inactive cluster does exist, a new cluster is created containing the solitary data point $\overline{X}$. This newly created cluster replaces the inactive cluster. We note that this new cluster is a potential true outlier or the beginning of a new trend of data points. Further understanding of this new cluster may only be obtained with the progress of the data stream.

In the event that $\overline{X}$ is inserted into the cluster $\mathcal{C}_{mindex}$, we need to perform two steps:

- We update the statistics to reflect the decay of the data points at the current moment in time. This updating is performed using the computation discussed in Observation 2.2. Thus, the relevant updates are performed in a "lazy" fashion. In other words, the statistics for a cluster do not decay, until a new point is added to it. Let the corresponding time stamp of the moment of addition be t. The last update time l is available from the cluster droplet statistics. We multiply the entries in the vectors $\overline{DC2}$, $\overline{DC1}$ and $w(t)$ by the factor $2^{-\lambda \cdot (t-l)}$ in order to update the corresponding statistics. We note that the lazy update mechanism results in stale decay characteristics for most of the clusters. This does not however affect the afore-discussed computation of the similarity measures.

- In the second step, we add the statistics for each newly arriving data point to the statistics for $\mathcal{C}_{mindex}$ by using the computation discussed in Observation 2.2.

In the event that the newly arriving data point does not naturally fit in any of the cluster droplets and an inactive cluster does exist, then we replace the most inactive cluster by a new cluster containing the solitary data point $\overline{X}$. In particular, the replaced cluster is the least recently updated cluster among all inactive clusters. This process is continuously performed over the life of the data stream, as new documents arrive over time. The work in [3] also presents a variety of other applications of the stream clustering technique such as evolution and correlation analysis.

A different way of utilizing the temporal evolution of text documents in the clustering process is described in [26]. Specifically, the work in [26] uses *bursty features* as markers of new topic occurrences in the data stream. This is because the semantics of an up-and-coming topic are often reflected in the frequent presence of a few distinctive words in the text stream. A specific example illustrates the bursty features in two topics corresponding to the two topics of *"US Mid-Term Elections"* and *"Newt Gingrich resigns from House"* respectively. The corresponding bursty features [26] which occurred frequently in the newsstream during the period for these topics were as follows:

US Mid-term Elections (Nov. 3, 1998 burst):
election, voters, Gingrich, president, Newt, ...

Newt Gingrich resigns from house (Nov 6, 1998 burst):
House, Gingrich, Newt, president, Washington ...

It is evident that at a given period in time, the nature of relevant topics could lead to bursts in specific features of the data stream. Clearly, such features are extremely important from a clustering perspective. Therefore, the method discussed in [26] uses a new representation, which is referred to as the *bursty feature representation* for mining text streams. In this representation, a time-varying weight is associated with the features depending upon its burstiness. This also reflects the varying importance of the feature to the clustering process. Thus, it is important to remember that a particular document representation is dependent upon the particular instant in time at which it is constructed.

Another issue which is handled effectively in this approach is an implicit reduction in dimensionality of the underlying collection. Text is inherently a high dimensional data domain, and the pre-selection of some of the features on the basis of their burstiness can be a natural way to

reduce the dimensionality of document representation. This can help in both the effectiveness and efficiency of the underlying algorithm.

The first step in the process is to identify the bursty features in the data stream. In order to achieve this goal, the approach uses Kleinberg's 2-state finite automaton model [27]. Once these features have been identified, the bursty features are associated with weights which depend upon their level of burstiness. Subsequently, a bursty feature representation is defined in order to reflect the underlying weight of the feature. Both the identification and the weight of the bursty feature are dependent upon its underlying frequency. A standard k-means approach is applied to the new representation in order to construct the clustering. It was shown in [26] that the approach of using burstiness improves the cluster quality. Once criticism of the work in [26] is that it is mostly focussed on the issue of improving effectiveness with the use of temporal characteristics of the data stream, and does not address the issue of efficient clustering of the underlying data stream.

In general, it is evident that feature extraction is important for all clustering algorithms. While the work in [26] focusses on using temporal characteristics of the stream for feature extraction, the work in [32] focusses on using *phrase extraction* for effective feature selection. This work is also related to the concept of topic-modeling, which will be discussed somewhat later. This is because the different topics in a collection can be related to the clusters in a collection. The work in [32] uses topic-modeling techniques for clustering. The core idea in the work of [32] is that individual words are not very effective for a clustering algorithm because they miss the context in which the word is used. For example, the word "star" may either refer to a celestial body or to an entertainer. On the other hand, when the phrase "fixed star" is used, it becomes evident that the word "star" refers to a celestial body. The phrases which are extracted from the collection are also referred to as *topic signatures*.

The use of such phrasal clarification for improving the quality of the clustering is referred to as *semantic smoothing* because it reduces the noise which is associated with semantic ambiguity. Therefore, a key part of the approach is to extract phrases from the underlying data stream. After phrase extraction, the training process determines a translation probability of the phrase to terms in the vocabulary. For example, the word "planet" may have high probability of association with the phrase "fixed star", because both refer to celestial bodies. Therefore, for a given document, a rational probability count may also be assigned to all terms. For each document, it is assumed that all terms in it are generated either by a topic-signature model, or a background collection model.

The approach in [32] works by modeling the soft probability $p(w|C_j)$ for word w and cluster C_j. The probability $p(w|C_j)$ is modeled as a linear combination of two factors; (a) A maximum likelihood model which computes the probabilities of generating specific words for each cluster (b) An indirect (translated) word-membership probability which first determines the maximum likelihood probability for each topic-signature, and then multiplying with the conditional probability of each word, given the topic-signature. We note that we can use $p(w|C_j)$ in order to estimate $p(d|C_j)$ by using the product of the constituent words in the document. For this purpose, we use the frequency $f(w, d)$ of word w in document d.

$$p(d|C_j) = \prod_{w \in d} p(w|C_j)^{f(w,d)} \tag{9.1}$$

We note that in the static case, it is also possible to add a background model in order to improve the robustness of the estimation process. This is however not possible in a data stream because of the fact that the background collection model may require multiple passes in order to build effectively. The work in [32] maintains these probabilities in online fashion with the use of a *cluster profile*, that weights the probabilities with the use of a fading function. We note that the concept of cluster profile is analogous to the concept of condensed droplet introduced in [3]. The key algorithm (denoted by OCTS) is to maintain a dynamic set of clusters into which documents are progressively assigned with the use of similarity computations. It has been shown in [32] how the cluster profile can be used in order to efficiently compute $p(d|C_j)$ for each incoming document. This value is then used in order to determine the similarity of the documents to the different clusters. This is used in order to assign the documents to their closest cluster. We note that the methods in [3, 32] share a number of similarities in terms of (a) maintenance of cluster profiles, and (b) use of cluster profiles (or condensed droplets) to compute similarity and assignment of documents to most similar clusters. (c) The rules used to decide when a new singleton cluster should be created, or one of the older clusters should be replaced.

The main difference between the two algorithms is the technique which is used in order to compute cluster similarity. The OCTS algorithm uses the probabilistic computation $p(d|C_j)$ to compute cluster similarity, which takes the phrasal information into account during the computation process. One observation about OCTS is that it may allow for very similar clusters to co-exist in the current set. This reduces the space available for distinct cluster profiles. A second algorithm called OCTSM is also proposed in [32], which allows for merging of very similar clusters. Before each assignment, it checks whether pairs of similar clusters can

be merged on the basis of similarity. If this is the case, then we allow the merging of the similar clusters and their corresponding cluster profiles. Detailed experimental results on the different clustering algorithms and their effectiveness are presented in [32].

Another method [42] uses a combination of a spectral partitioning and probabilistic modeling method for novelty detection and topic tracking. This approach uses a HITS-like spectral technique within a probabilistic framework. The probabilistic part is an unsupervised boosting method, which is closer to semi-parametric maximum likelihood methods.

A closely related area to clustering is that of topic modeling, which is a problem closely related to that of clustering. In the problem of topic modeling, we perform a *soft* clustering of the data in which each document has a membership probability to one of a universe of topics rather than a deterministic membership. A variety of mixture modeling techniques can be used in order to determine the topics from the underlying data. Recently, the method has also been extended to the *dynamic* case which is helpful for topic modeling of text streams [10]. A closely related topic is that of topic *detection* and *tracking*, which is discussed below.

Recently, a variety of methods for maintaining topic models in a streaming scenario have been proposed in [49]. The work evaluates a number of different methods for adapting topic models to the streaming scenario. These include methods such as Gibbs sampling and variational inference. In addition, a method is also proposed, which is based on text classification. The latter has the advantage of requiring only a single matrix multiplication, and is therefore much more efficient. A method called *SparseLDA* is proposed, which is an effective method for evaluating Gibbs sampling distributions. The results in [49] show that this method is 20 times faster than traditional LDA.

In some applications, it is desirable to have clusters of approximately balanced size, in which a particular cluster is not significantly larger than the others. A competitive online learning for determining such balanced clusters have been proposed in [9]. The essential approach in [9] is to design a model in which it is harder for new data points to join larger clusters. This is achieved by penalizing the imbalance into the objective function criterion. It was shown in [9] that such an approach can determine well balanced clusters.

2.1 Topic Detection and Tracking in Text Streams

A closely related problem to clustering is that of *topic detection and tracking* [5, 11, 24, 46, 47, 51]. In this problem, we create unsupervised

clusters from a text stream, and then determine the sets of clusters which match real events. These real events may correspond to documents which are identified by a human. Since the problem of online topic detection is closely related to that of clustering, we will discuss this problem as a subsection of our broader discussion on clustering, though not all methods for topic detection use clustering techniques. In this subsection, we will discuss all the methods for topic detection, whether they use clustering or not.

The earliest work on topic detection and tracking was performed in [5, 47]. The work arose out of a DARPA initiative [55] which formally defined this problem and proposed the initial set of algorithms for the task. An interesting technique in [47] designs methods which can be used for both retrospective and online topic tracking and detection. In retrospective event detection, we create groups from a corpus of documents (or stories), and each group corresponds to an event. The online version is applicable to the case of data streams, and in this case, we process documents sequentially in order to determine whether an incoming document corresponds to a new event. An online clustering algorithm can also be used in order to track the different events in the data in the form of clusters. For each incoming document $\overline{X}$ we compute its similarity to the last m documents $\overline{Y_{t-1}} \ldots \overline{Y_{t-m}}$. The score for the incoming document $\overline{X}$ is computed as follows:

$$score(\overline{X}) = \max_{i \in \{t-m \ldots t-1\}} (1 - sim(\overline{X}, \overline{Y_i})) \tag{9.2}$$

A document is considered novel, when its score is above a pre-defined threshold. In addition, a decay-weighted version is designed in which the weight of documents depends upon its recency. The idea here is that a document is considered to be a new event when the last occurrence of a similar document did not occur recently "enough". In this case, the corresponding score is designed as follows:

$$score(\overline{X}) = \max_{i \in \{t-m \ldots t-1\}} (1 - \frac{(m + i - t + 1)}{m} \cdot sim(\overline{X}, \overline{Y_i})) \tag{9.3}$$

We note that each $\overline{Y_i}$ need not necessarily represent a single document, but may also represent a cluster or a larger grouping. We also note that the method for detecting a new event can be combined with a cluster tracking task. This is because the determination of a new event is indicative of a new event (or singleton cluster) in the data.

The work in [5] addressed the problem of new event detection by examining the relationship of a current document to the previous documents in the data. The key idea is to use feature extraction and selection techniques in order to design a query representation of the document

content. We determine the query's initial threshold by comparing the document content with the query, and set it as the triggering threshold. Then, we determine if the document triggers any queries from the previous documents in the collection. If no queries are triggered, then the document is deemed to be a new event. Otherwise, this document is not a new event.

One general observation about the online topic detection and tracking problem [6, 48] is that this problem is quite hard in general, and the performance of first event detection can degrade under a variety of scenarios. In order to improve the effectiveness of first event detection, the work in [48] proposes to use the training data of old events in order to learn useful statistics for the prediction of new events. The broad approach in [48] contains the following components:

- The documents are classified into broad topics, each of which consists of multiple events.
- Named entities are identified, optimizing their weight relative to normal words for each topic, and computing a stopword list for each topic.
- Measuring the novelty of a new document conditioned on the system-predicted topic for that document.

Clearly such an approach has the tradeoff that it requires prior knowledge about the collection in the form of training data, but provides better accuracy. More details on the approach may be found in [48].

A method proposed in [11] is quite similar to that proposed in [47], except that it proposes a number of improvements in how the tf–idf model is incrementally maintained for computation of similarity. For example, a *source-specific* tf–idf model is maintained in which the statistics are specific to each news source. Similarly, the approach normalizes for the fact that documents which come from the same source tend to have a higher similarity because of the vocabulary conventions which are often used in the similarity computation. In order to achieve this goal, the approach computes the average similarity between the documents from a particular pair of sources and subtracts this average value while computing the similarity between a pair of documents for the purpose of new event detection. A number of other techniques for improving the quality of event detection (such as using inverse event frequencies) are discussed in [11]. A method for online topic detection and tracking is presented in [51] as an application of stream clustering. This work uses a probabilistic LDA model in order to create an online model for estimating the growing number of clusters. The general approach in this work is

similar to the concepts already proposed in [47]. The main novelty of the work is the design of an online approach for probabilistic clustering.

Another method for fast and parameter-free bursty event detection is proposed in [24]. This approach focusses on finding bursty features which characterize the presence of an event. In order to achieve this goal, the technique in [24] proposes a feature-pivot clustering, which groups features on the basis of bursty co-occurrence. The approach is designed to be parameter-free, which gives it an advantage in a number of scenarios.

The problem of event detection has also been studied with the use of *keyword graphs* in [39]. The work in [39] builds a keywords graph from a text stream in which a node corresponds to a keyword, and an edge is added to the keyword graphs when a pair of words occur together in the document. Events are characterized as communities of keywords in this network. This broad approach is used in the context of a *window-based technique* for the case of social streams.

In the context of social network streams, a natural question arises, as to whether one can use any of the *social dimensions* of the underlying stream in order to improve the underlying event detection process. Such an approach has been proposed in [53], in which events are determined by combining text clustering, social network structure clustering, and temporal segmentation. In addition, a multi-dimensional visualization tool is provided, which discusses ways of visualizing the relationships between the events along the three dimensions. In this case, an event is defined as a set of text documents which are semantically coherent, and are structurally well connected both in terms of social network structure and time. These three different characteristics are used in the following ways:

- First, the content is used in order to create a hierarchy of topics from the social text stream.

- While the topical patterns are useful for distinguishing content, the temporal segmentation is used in order to distinguish different events. The assumption is that the communication between different parties happen during a short contiguous time period.

- Since it is assumed that the events occur between connected entities, we use the connectivity between the different events in order to further segment the events. An edge between a pair of nodes corresponds to a communication between the social entities. Multiple edges are allowed between pairs of nodes. This structure is used in order to determine the event-dependent communities.

Another method [36] has been proposed in the context of the *Twitter* social network data set. In this paper, a locality sensitive hashing method (LSH) is used in order to keep track of the different documents. The idea in LSH [14] is to use a hashing scheme in which the probability of hashing two documents into the same bucket is proportional to the cosine of the angle between the two documents. For a given query document, we check all the documents in the same bucket, and then perform the similarity calculation explicitly with all documents in the same bucket. The closest document is returned as the nearest neighbor. We note that the problem of first-story detection can be considered an application of repeated nearest neighbor querying in which an incoming document is compared to the previously seen documents from the data stream. While LSH can be used directly in conjunction with a nearest neighbor search for first story detection, such an approach typically leads to poor results. This is because LSH works effectively only if the true nearest neighbor is close to the query point. Otherwise, such an approach is unable to find the true nearest neighbor. Therefore, the approach in [36] uses the strategy of using LSH only for declaring a document to be sufficiently new on an aggregate basis. For such cases, the document is compared against a fixed number of the most recent documents. In the event that the corresponding distance is above a given threshold, we can declare the underlying story as novel. The main advantage of this technique over many of the aforementioned techniques is that of *speed*, which is especially important, when dealing with social streams of very high volume. This speed is achieved because of the use of the LSH technique, though there is some loss in accuracy because of the approximation process. This technique was compared [36] against the *UMass system* [7], and it was found that more than an order of magnitude improvement in speed was obtained with only a minor loss in accuracy.

Most of the work in text stream mining and topic detection is performed in the context of a *single news stream*. In many cases, we have multiple text streams [44], which are indexed by the same set of time points (called coordinated text streams). For example, when a major event happens, all the news articles published by different agencies in different languages cover the same event in that period. This is referred to as a *correlated bursty topic pattern* in the different news article streams. In some cases, when the correlated streams are multi-lingual, they may even have completely different vocabulary. The discovery of bursty topic patterns can determine the interesting events and associations between different streams. Such an approach can also determine

interesting local and stream-specific patterns by factoring out the global noise which is common to the different streams.

In order to achieve this goal, the technique in [44] aligns the text samples from different streams on the basis of the shared time-stamps. This is used in order to discover topics from multiple streams simultaneously with a single probabilistic mixture model. The approach of constructing independent topic models from different streams is that the topic models from the different streams may not match each other very well, or at least, create a challenge in matching the different topic models, if it is done at the end of the process of model construction. Therefore, it is important to make the mixture models for the different streams communicate with one another during the modeling process, so that a single mixture model is constructed across the different streams. In order to achieve this goal, the stream samples at a given point are merged together, while keeping the stream identities. In order to achieve this, we align the topic models from different streams while keeping their identities. While the topic models from different streams are separate, the global mixture model is designed for the overall text stream sample. Such a mixture model is coordinated, because it would emphasize topics which tend to occur across multiple streams. Once the coordinated mixture model is obtained, the topical models for the different streams can be extracted by fitting the mixture model to the different streams. As the topic models for the different streams ate aligned with one another, we can obtained a correlated bursty topic pattern, when the corresponding topic is bursty during the same period. A key aspect of this approach is that it does not require the different streams to share vocabulary. Rather it is assumed the topics involved in a correlated bursty topic pattern tend to have the same distribution across streams, and this can be used in order to match topics from different streams. More details on the approach may be found in [44].

3. Classification of Text Streams

The problem of classification of data streams has been widely studied by the data mining community in the context of different kinds of data [1, 2, 43, 50]. Many of the methods for classifying numerical data can be easily extended to the case of text data, just as the numerical clustering method in [4] has been extended to a text clustering method in [3]. In particular, the classification method of [2] can be extended to the text domain, by adapting the concept of numerical micro-clusters to condensed droplets as discussed in [3]. With the use of the condensed droplet data structure, it is possible to extend all the methods discussed

in [2] to the text stream scenario. Similarly, the core idea in [43] uses an ensemble based approach on chunks of the data stream. This broad approach is essentially a *meta-algorithm* which is not too dependent upon the specifics of the underlying data format. Therefore, the broad method can also be easily extended to the case of text streams.

The problem of text stream classification arises in the context of a number of different IR tasks. The most important of these is *news filtering* [30], in which it is desirable to automatically assign incoming documents to pre-defined categories. A second application is that of email spam filtering [8], in which it is desirable to determine whether incoming email messages are spam or not. We note that the problem of text stream classification can arise in two different contexts, depending upon whether the training or the test data arrives in the form of a stream:

- In the first case, the training data may be available for batch learning, but the test data may arrive in the form of a stream.
- In the second case, both the training and the test data may arrive in the form of a stream. The patterns in the training data may continuously change over time, as a result of which the models need to be updated dynamically.

The first scenario is usually easy to handle, because most classifier models are compact and classify individual test instances efficiently. On the other hand, in the second scenario, the training model needs to be constantly updated in order to account for changes in the patterns of the underlying training data. The easiest approach to such a problem is to incorporate temporal decay factors into model construction algorithms, so as to age out the old data. This ensures that the new (and more timely data) is weighted more significantly in the classification model. An interesting technique along this direction has been proposed in [38], in which a temporal weighting factor is introduced in order to modify the classification algorithms. Specifically, the approach has been applied to the Naive Bayes, Rocchio, and k-nearest neighbor classification algorithms. It has been shown that the incorporation of temporal weighting factors is useful in improving the classification accuracy, when the underlying data is evolving over time.

A number of methods have also been designed specifically for the case of text streams. In particular, the method discussed in [23] studies methods for classifying text streams in which the classification model may evolve over time. This problem has been studied extensively in the literature in the context of multi-dimensional data streams [2, 43]. For example, in a spam filtering application, a user may generally delete

the spam emails for a particular topic, such as those corresponding to political advertisements. However, in a particular period such as the presidential elections, the user may be interested in the emails for that topic, it may not be appropriate to continue to classify that email as spam.

The work in [23] looks at the particular problem of classification in the context of user-interests. In this problem, the label of a document is considered either *interesting* or *non-interesting*. In order to achieve this goal, the work in [23] maintains the interesting and non-interesting topics in a text stream together with the evolution of the theme of the interesting topics. A document collection is classified into multiple topics, each of which is labeled either interesting or non-interesting at a given point. A concept refers to the main theme of interesting topics. A concept drift refers to the fact that the main theme of the interesting topic has changed.

The main goals of the work are to maximize the accuracy of classification and minimize the cost of re-classification. In order to achieve this goal, the method in [23] designs methods for detecting both *concept drift* as well as *model adaptation*. The former refers to the change in the theme of user-interests, whereas the latter refers to the detection of brand new concepts. In order to detect concept drifts, the method in [23] measures the classification error-rates in the data stream in terms of true and false positives. When the stream evolves, these error rates will increase, if the change in the concepts are not detection. In order to determine the change in concepts, techniques from statistical quality control are used, in which we determine the mean μ and standard deviation σ of the error rates, and determine whether this error rate remains within a particular tolerance which is $[\mu - k \cdot \sigma, \mu + k \cdot \sigma]$. Here the tolerance is regulated by the parameter k. When the error rate changes, we determine when the concepts should be dropped or included. In addition, the drift rate is measured in order to determine the rate at which the concepts should be changed for classification purposes. In addition, methods for dynamic construction and removal of sketches are discussed in [23].

Another related work is that of one-class classification of text streams [52], in which only training data for the positive class is available, but there is no training data available for the negative class. This is quite common in many real applications in which it easy to find representative documents for a particular topic, but it is hard to find the representative documents in order to model the background collection. The method works by designing an ensemble of classifiers in which some of the classifiers corresponds to a recent model, whereas others correspond

to a long-term model. This is done in order to incorporate the fact that the classification should be performed with a combination of short-term and long-term trends.

A rule-based technique, which can learn classifiers incrementally from data streams is the *sleeping-experts systems* [15, 21]. One characteristic of this rule-based system is that it uses the position of the words in generating the classification rules. Typically, the rules correspond to sets of words which are placed close together in a given text document. These sets of words are related to a class label. For a given test document, it is determined whether these sets of words occur in the document, and are used for classification. This system essentially learns a set of rules (or sleeping experts), which can be updated incrementally by the system. While the technique was proposed prior to the advent of data stream technology, its online nature ensures that it can be effectively used for the stream scenario.

One of the classes of methods which can be easily adapted to stream classification is the broad category of *neural networks* [41, 45]. This is because neural networks are essentially designed as a classification model with a network of perceptrons and corresponding weights associated with the term-class pairs. Such an incremental update process can be naturally adapted to the streaming context. These weights are incrementally learned as new examples arrive. The first neural network methods for online learning were proposed in [41, 45]. In these methods, the classifier starts off by setting all the weights in the neural network to the same value. The incoming training example is classified with the neural network. In the event that the result of the classification process is correct, then the weights are not modified. On the other hand, if the classification is incorrect, then the weights for the terms are either increased or decreased depending upon which class the training example belongs to. Specifically, if class to which the training example belongs is a positive instance, the weights of the corresponding terms (in the training document) are increased by α. Otherwise, the weights of these terms are reduced by α. The value of α is also known as the *learning rate*. Many other variations are possible in terms of how the weights may be modified. For example, the method in [18] uses a multiplicative update rule, in which two multiplicative constants $\alpha_1 > 1$ and $\alpha_2 < 1$ are used for the classification process. The weights are multiplied by α_1, when the example belongs to the positive class, and is multiplied by α_2 otherwise. Another variation [31] also allows the modification of weights, when the classification process is correct. A number of other online neural network methods for text classification (together with background on the topic) may be found in [16, 22, 34, 35]. A Bayesian method for

classification of text streams is discussed in [13]. The method in [13] constructs a Bayesian model of the text which can be used for online classification. The key components of this approach are the design of a Bayesian online perceptron and a Bayesian online Gaussian process, which can be used effectively for online learning.

4. Evolution Analysis in Text Streams

A key problem in the case of text is to determine evolutionary patterns in temporal text streams. An early survey on the topic of evolution analysis in text streams may be found in [28]. Such evolutionary patterns can be very useful in a wide variety of applications, such as summarizing events in news articles and revealing research trends in the scientific literature. For example, an event may have a life cycle in the underlying theme patterns such as the beginning, duration, and end of a particular event. Similarly, the evolution of a particular topic in the research literature may have a life-cycle in terms of how the different topics affect one another. This problem was defined in [33], and contains three main parts: (a) Discovering the themes in text; (b) creating an evolution graph of themes; and (c) studying the life cycle of themes.

A *theme* is defined as a semantically related set of words, with a corresponding probability distribution, which coherently represents a particular topic or sub-topic. This corresponds to a model, which is represented by θ. The *span* of such a theme represents the starting and end point of the corresponding theme in terms of the time-interval (s, t). Thus, the theme span is denoted by the triple $\gamma = (\theta, s, t)$. As time passes, a particular theme γ_1 may perform a transition into another theme γ_2. A theme γ_1 is said to have evolved into another theme γ_2, if there is sufficient similarity between their corresponding spans. It is possible to create a *theme evolution graph* $G = (N, A)$, in which each node in N corresponds to a theme, and each edge in A corresponds to a transition between two themes. The overall approach requires three steps:

- In the first step, we segment the text stream into a number of possibly overlapping sub-collections with fixed or variable time spans. This is done in an application-specific way.
- The salient themes are determined from each collection with the use of a probabilistic mixture model. A standard mixture model technique [19] was used for this purpose.
- Finally, all the evolution transitions are determined from these theme patterns. This is done with the use of the KL-divergence measure in order to compute the evolution distance between two

themes. In the event that the similarity is above a given threshold, the transition is considered valid.

In addition, a method is proposed in [33] for analyzing the entire theme life cycle by measuring the strength of the theme over different periods. A method based on HMM is proposed to measure the theme-shifts over the entire period as well.

The problem of tracking new topics, ideas, and memes across the Web has been studied in [29]. This problem is related to that of the topic detection and tracking discussed earlier. However, the rate of evolution in the web and blog scenario is significantly greater than the models which have been discussed in earlier work. In the context of the web and social networks, the content spreads widely and then fades over time scales on the order of days. The work is [29] develops a framework for tracking short, distinctive phrases that persistently appear in online text over time. In addition, scalable algorithms were proposed for clustering textual variants of such phrases. In addition, the approach is able to perform local analysis for specific threads. This includes the determination of peak intensity and the rise and decay in the intensity of specific threads. The relationship between the news cycle and blogs is examined in terms of how events propagate from one to the other, and the corresponding time-lag for the propagation process.

5. Conclusions

This chapter studies the problem of mining text streams. The challenge in the case of text stream arises because of its temporal and dynamic nature in which the patterns and trends of the stream may vary rapidly over time. The determination of the changes in the underlying patterns and trends is very useful in the context of a wide variety of applications. A variety of problems in text stream mining are examined such as clustering, classification, evolution analysis, and event detection. In addition, we studied the applications of some of these techniques in the context of new applications such as social networks. A lot of interesting avenues for research remain in the context of social media analytics, and the use of social dimensions in order to enhance text stream mining. In particular, the incorporation of network structure into the mining of social streams such as *Twitter* remains a relatively unexplored area, which can be a fruitful avenue for future research.

References

[1] C. C. Aggarwal. Data Streams: Models and Algorithms, *Springer*, 2007.

[2] C. C. Aggarwal, J. Han, J. Wang, P. Yu. On Demand Classification of Data Streams, *KDD Conference*, 2004.

[3] C. C. Aggarwal, P. S. Yu. A Framework for Clustering Massive Text and Categorical Data Streams, *SIAM Conference on Data Mining*, 2006.

[4] C. C. Aggarwal, J. Han. J. Wang, P. Yu. A Framework for Clustering Evolving Data Streams, *VLDB Conference*, 2003.

[5] J. Allan, R. Papka, V. Lavrenko. On-line new event detection and tracking. *ACM SIGIR Conference*, 1998.

[6] J. Allan, V. Lavrenko, H. Jin. First story detection in tdt is hard. *ACM CIKM Conference*, 2000.

[7] J. Allan, V. Lavrenko, D. Malin, R. Swan. Detections, bounds and timelines: Umass and tdt3, *Proceedings of the Topic Detection and Tracking Workshop*, 2000.

[8] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, C. D. Spyropoulos. An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages. *Proceedings of the ACM SIGIR Conference*, 2000.

[9] A. Banerjee, J. Ghosh. Competitive learning mechanisms for scalable, balanced and incremental clustering of streaming texts, *NIPS Conference*, 2003.

[10] D. Blei, J. Lafferty. Dynamic topic models. *ICML Conference*, 2006.

[11] T. Brants, F. Chen, A. Farahat. A system for new event detection. *ACM SIGIR Conference*, 2003.

[12] L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, S. Guha. Streaming-Data Algorithms for High-Quality Clustering. *ICDE Conference*, 2002.

[13] K. Chai, H. Ng, H. Chiu. Bayesian Online Classifiers for Text Classification and Filtering, *ACM SIGIR Conference*, 2002.

[14] M. Charikar. Similarity Estimation Techniques from Rounding Algorithms, *STOC Conference*, 2002.

[15] W. Cohen, Y. Singer. Context-sensitive learning methods for text categorization. *ACM Transactions on Information Systems*, 17(2), pp. 141–173, 1999.

[16] K. Crammer, Y. Singer. A New Family of Online Algorithms for category ranking, *ACM SIGIR Conference*, 2002.

[17] D. Cutting, D. Karger, J. Pedersen, J. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. *Proceedings of the SIGIR*, 1992.

[18] I. Dagan, Y. Karov, D. Roth. Mistake-driven learning in text categorization. *Conference Empirical Methods in Natural Language Processing*, 1997.

[19] A. P. Dempster, N. M. Laird, D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of Royal Statistical Society* 39: pp. 1–38, 1977.

[20] D. Fisher. Knowledge Acquisition via incremental conceptual clustering. *Machine Learning*, 2: pp. 139–172, 1987.

[21] Y. Freund, R. Schapire, Y. Singer, M. Warmuth. Using and combining predictors that specialize. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pp. 334–343, 1997.

[22] Y. Freund, R. Schapire. Large Margin Classification using the perceptron Algorithm, *COLT*, 1998.

[23] G. P. C. Fung, J. X. Yu, H. Lu. Classifying text streams in the presence of concept drifts. *PAKDD Conference*, 2004.

[24] G. P. C. Fung, J. X. Yu, P. Yu, H. Lu. Parameter Free Bursty Events Detection in Text Streams, *VLDB Conference*, 2005.

[25] J. H. Gennari, P. Langley, D. Fisher. Models of incremental concept formation. *Journal of Artificial Intelligence*, 40: pp. 11–61, 1989.

[26] Q. He, K. Chang, E.-P. Lim, J. Zhang. Bursty feature representation for clustering text streams. *SDM Conference*, 2007.

[27] J. Kleinberg, Bursty and hierarchical structure in streams, *ACM KDD Conference*, pp. 91–101, 2002.

[28] A. Kontostathis, L. Galitsky, W. M. Pottenger, S. Roy, D. J. Phelps. A survey of emerging trend detection in textual data mining. *Survey of Text Mining*, pp. 185–224, 2003.

[29] J. Leskovec, L. Backstrom, J. Kleinberg. Meme Tracking and the Dynamics of the News Cycle, *KDD Conference*, 2009.

[30] D. Lewis. The TREC-4 filtering track: description and analysis. *Proceedings of TREC-4, 4th Text Retrieval Conference*, pp. 165–180, 1995.

[31] D. Lewis, R. E. Schapire, J. P. Callan, R. Papka. Training algorithms for linear text classifiers. *ACM SIGIR Conference*, 1996.

[32] Y.-B. Liu, J.-R. Cai, J. Yin, A. W.-C. Fu. Clustering Text Data Streams, *Journal of Computer Science and Technology*, Vol. 23(1), pp. 112–128, 2008.

[33] Q. Mei, C.-X. Zhai. Discovering Evolutionary Theme Patterns from Text- An Exploration of Temporal Text Mining, *ACM KDD Conference*, 2005.

[34] H. T. Ng, W. B. Goh, K. L. Low. Feature selection, perceptron learning, and a usability case study for text categorization. *SIGIR Conference*, 1997.

[35] F. Rosenblatt. The perceptron: A probabilistic model for information and storage organization in the brain, *Psychological Review*, 65: pp. 386–407, 1958.

[36] S. Petrovic, M. Osborne, V. Lavrenko. Streaming First Story Detection with Application to Twitter. *Proceedings of the ACL Conference*, pp. 181–189, 2010.

[37] N. Sahoo, J. Callan, R. Krishnan, G. Duncan, R. Padman. Incremental Hierarchical Clustering of Text Documents, *ACM CIKM Conference*, 2006.

[38] T. Salles, L. Rocha, G. Pappa, G. Mourao, W. Meira Jr., M. Goncalves. Temporally-aware algorithms for document classification. *ACM SIGIR Conference*, 2010.

[39] H. Sayyadi, M. Hurst, A. Maykov. Event Detection in Social Streams, *AAAI*, 2009.

[40] H. Schutze, C. Silverstein. Projections for Efficient Document Clustering, *ACM SIGIR Conference*, 1997.

[41] H. Schutze, D. Hull, J. Pedersen. A comparison of classifiers and document representations for the routing problem. *ACM SIGIR Conference*, 1995.

[42] A. Surendran, S. Sra. Incremental Aspect Models for Mining Document Streams. *PKDD Conference*, 2006.

[43] H. Wang, W. Fan, P. Yu, J. Han, Mining Concept-Drifting Data Streams with Ensemble Classifiers, *KDD Conference*, 2003.

[44] X. Wang, C.-X. Zhai, X. Hu, R. Sproat. Mining Correlated Bursty Topic Patterns from Correlated Text Streams, *ACM KDD Conference*, 2007.

[45] E. Wiener, J. O. Pedersen, A. S. Weigend. A Neural Network Approach to Topic Spotting. *SDAIR*, pp. 317–332, 1995.

[46] Y. Yang, J. Carbonell, R. Brown, T. Pierce, B. T. Archibald, X. Liu. Learning approaches for detecting and tracking news events. *IEEE Intelligent Systems*, 14(4):32–43, 1999.

[47] Y. Yang, T. Pierce, J. Carbonell. A study on retrospective and on-line event detection. *ACM SIGIR Conference*, 1998.

[48] Y.Yang, J. Carbonell, C. Jin. Topic-conditioned Novelty Detection. *ACM KDD Conference*, 2002.

[49] L. Yao, D. Mimno, A. McCallum. Efficient methods for topic model inference on streaming document collections, *ACM KDD Conference*, 2009.

[50] K. L. Yu, W. Lam. A new on-line learning algorithm for adaptive text filtering. *ACM CIKM Conference*, 1998.

[51] J. Zhang, Z. Ghahramani, Y. Yang. A probabilistic model for on-line document clustering with application to novelty detection. In *Saul L., Weiss Y., Bottou L. (eds) Advances in Neural Information Processing Letters*, 17, 2005.

[52] Y. Zhang, X. Li, M. Orlowska. One Class Classification of Text Streams with Concept Drift, *ICDMW Workshop*, 2008.

[53] Q. Zhao, P. Mitra. Event Detection and Visualization for Social Text Streams, *ICWSM*, 2007.

[54] S. Zhong. Efficient Streaming Text Clustering. *Neural Networks*, Volume 18, Issue 5-6, 2005.

[55] http://projects.ldc.upenn.edu/TDT/