Event Detection in Social Streams

Charu C. Aggarwal* Karthik Subbian†

Abstract

Social networks generate a large amount of text content over time because of continuous interaction between participants. The mining of such *social streams* is more challenging than traditional text streams, because of the presence of both text content and implicit network structure within the stream. The problem of event detection is also closely related to clustering, because the events can only be inferred from *aggregate* trend changes in the stream. In this paper, we will study the two related problems of clustering and event detection in social streams. We will study both the supervised and unsupervised case for the event detection problem. We present experimental results illustrating the effectiveness of incorporating network structure in event discovery over purely content-based methods.

1 Introduction

In this paper, we will study the problem of clustering and event detection in social streams. Much of the text data in social scenarios arises in the context of streaming applications, in which the text arrives as a continuous and massive stream of text segments [2]. Such applications present a special challenge for mining algorithms, because of the fact that it is often necessary to *process the data in a single pass* and one cannot store all the data on disk for re-processing.

The online event detection problem is closely related to that of *topic detection and tracking* [4, 5, 7, 14, 17, 22, 23, 24]. This problem is also closely related to stream clustering, and attempts to determine new *topical trends in the text stream* and their significant evolution. The idea is that important and newsworthy events in real life are often captured in the form of *temporal bursts of closely related messages* in a social stream. The problem can be proposed in both the supervised and unsupervised scenarios. In the *unsupervised case*, it is assumed that no training data is available in order to direct the event detection process for the stream. In the *supervised case*, prior data about events is available in order to guide the event detection process.

In this paper, we will study the problems of clustering and event detection in *social streams* in which each text message is associated with at least a pair of actors in the social network. We use the term social network rather loosely, as it could refer to the messages in an online chat messenger service, or it could even refer to an email network in which messages are sent between pairs of actors. A number of interesting issues arise in such social networks, because they are dynamic, and are associated with *network structure* in the stream. Specifically, each actor is a node in the social network, and each message sent in the social network is the text content associated with an edge in the social network. Clearly, multiple messages can be sent between the same pair of actors over time. In this case, we would like to use *the topical content of the documents, their temporal distribution, and the graphical structure of the dynamic network of interactions in order to detect interesting events and their evolution*. Clearly, messages which are sent between a tightly knit group of actors may be more indicative of a particular event of social interest, than a set of messages which are more diffusely related from a structural point of view. Such messages are structurally well connected, when the social network is viewed as a graph, with the edges corresponding to the messages sent between entities. This is related to the problem of *community detection* [3], in which we try to find structurally connected regions of the social network. At the same time, the *content and topics* of the documents should also play a strong role in the event detection process. Thus, both network locality and structure need to be leveraged in a *dynamic streaming scenario* for the event detection process.

Therefore, the key challenges for event detection in social streams are as follows: **(a)** The ability to use both the content and the (graphical) structure of the interactions for event detection. **(b)** The ability to use temporal information in the event detection process. For example, a new trend of closely related text documents from a structural and content point of view, which have not been encountered earlier may correspond to a new event in the stream. **(c)** The ability to handle *very large and massive* volumes of text documents under the *one-pass constraint of streaming scenarios*.

This paper is organized as follows. The remainder of this section presents related work. In section 2, we present the model for event mining in social streams. The algorithms for supervised and unsupervised event detection are presented in section 3. In section 4, we present the experimental results. Section 5 presents the conclusions and summary.

*IBM T. J. Watson Research Center, Hawthorne, NY, USA, charu@us.ibm.com

†Department of Computer Science & Engineering, University of Minnesota, Twin Cities, MN, USA, karthik@umn.edu

1.1 Related Work The problem of determining events in streams is closely related to the problem of stream clustering [2]. This method has been studied extensively by the text mining community in the context of the topic detection and tracking problem [4, 5, 7, 14, 17, 22, 23, 24]. This is also related to the problem of clustering and topic modeling [6, 22, 13, 16, 21, 27] in dynamic text streams.

However, in social networks, there is a rich amount of structure available in determining the key events in the network. For example, an event corresponding to *Mideast Unrest* may often correspond to text streams exchanged between members who are closely linked to one another based on geographical proximity. While the use of linkages in order to determine clusters and patterns has been widely studied by the social networking community [8, 9, 11, 25], these methods are typically designed for static networks. Some clustering methods have recently also been designed for dynamic networks [8, 9], though they do not use the content of the underlying network for the mining process. On the other hand, some recent methods for pattern discovery in networks use both content and structure [25, 28], though these methods are not defined for the problem of event detection in the *temporal scenario*. A method in [15] is designed to measure the diffusion and spread characteristics of known and popular events, but is not designed for *new* event *discovery*. In this paper, we design a method which can use the content, structural and temporal information in a holistic way in order to detect relevant clusters and events in social streams.

2 Event Mining in Social Streams: The Model

In this section, we introduce the notations and definitions and the model for event mining in social streams. Such social streams are assumed to consist of *content-based interactions between structurally connected entities* in the data. Therefore, we will propose a number of notations and definitions for this purpose. We assume that the structure of the social network is denoted by the graph $G = (N, A)$. The node set is denoted by N and edge set is denoted by A. The social stream corresponds to the interactions between the different actors in the node N, and each interaction is an edge drawn from the linkage set A. Therefore, the graph G provides information about the universe of interactions in the social network. We next define the concept of a social stream which is overlaid on this social network structure.

DEFINITION 1. (SOCIAL STREAM) *A social stream is a continuous and temporal sequence of objects $S_1 \ldots S_r \ldots$, such that each object S_i corresponds to a content-based interaction between social entities, and contains explicit content information and linkage information between entities as follows:*

- *The object S_i contains a text document T_i which corresponds to the content of the interaction of an entity in the social network with one or more other entities.*
- *The object S_i contains the origination node $q_i \in N$ which is the sender of the message T_i to other nodes.*
- *The object S_i contains a* set of one or more *receiver nodes $R_i \subseteq N$, which correspond to all recipients of the message T_i from node q_i. Thus, the message T_i is sent from the origination node q_i to each node $r \in R_i$. It is assumed that each edge (q_i, r) belongs to the set A.*

Thus, the object S_i is represented by the tuple (q_i, R_i, T_i).

We note that the above definition of a social stream captures a number of different natural scenarios in different kinds of social networks:

- In the *Twitter* social network, the document T_i is the tweet content, the node q_i is the tweeting actor, and the set R_i is the recipient set.
- Email and chat interaction networks may also be considered social networks with an exactly similar interpretation to the above.
- In many social networks, a posting on the wall of one actor to another corresponds to an edge with the document T_i corresponding to the content of the posting.

Such a social stream typically contains rich information about the trends which may lead to changes both in the content and the structural locality of the network in which the interactions may occur. We begin with describing an unsupervised technique for event detection, which continuously characterizes the incoming interactions in the form of clusters, and leverages them in order to report events in the data stream. We formally define the social stream clustering problem.

DEFINITION 2. (SOCIAL STREAM CLUSTERING) *A social stream $S_1 \ldots S_r \ldots$ is continuously partitioned into k* current *clusters $\mathcal{C}_1 \ldots \mathcal{C}_k$, such that:*

- *Each object S_i belongs to at most one of the current clusters $\mathcal{C}_r$.*
- *The objects are assigned to the different clusters with the use of a similarity function which captures both the content of the interchanged messages, and the dynamic social network structure implied by the different messages.*

We will use both the content and linkage information in order to create the clusters. Since the clusters are created dynamically, they may change considerably over time, as the stream evolves, and new points are added to the clusters. Furthermore, in some cases, an incoming object may be different enough from the current clusters. In that case, it

may be put into a cluster of its own, and one of the current clusters may be removed from the set $\mathcal{C}_1 \ldots \mathcal{C}_k$. Such an event may be an interesting one, especially if the newly created cluster starts a new pattern of activity in which more stream objects are subsequently added. At the same time, in some cases, the events may not be entirely new, but may correspond to significant changes in the patterns of the arriving objects in terms of their relative distribution to clusters. Therefore, we define two kinds of events, which are referred to as *novel events* and *evolution events* in order to describe these different scenarios.

DEFINITION 3. (NOVEL EVENT) *The arrival of a data point S_i is said to be a novel event if it is placed as a single point within a newly created cluster $\mathcal{C}_i$.*

We denote the creation time for cluster $\mathcal{C}_i$ by $t(\mathcal{C}_i)$. The event in this case is the story or topic underlying the data point S_i and not the data point itself. We will discuss the process for creation and maintenance of a cluster in a social stream slightly later. Next, we define the concept of an *evolution event*. An evolution event is defined with respect to specific time horizon and represents a change in the relative activity for that particular cluster. We first define the concept of fractional cluster presence.

DEFINITION 4. (FRACTIONAL CLUSTER PRESENCE) *The fractional cluster presence for cluster $\mathcal{C}_i$ in the time period (t_1, t_2) is the fraction of records from the social stream arriving during time period (t_1, t_2), which belong to the cluster $\mathcal{C}_i$. This fractional presence is denoted by $F(t_1, t_2, \mathcal{C}_i)$.*

The occurrence of a new event typically affects the relative presence of the data points in the different clusters, or it may result in a novel event. For example, the *Mideast Unrest* event may result in either the creation of a new cluster or the significant addition of new data points to the clusters most closely related to this topic. This is because it is often possible for previously existing clusters to match closely with a sudden burst of objects related to a particular topic. This sudden burst is characterized by a change in fractional presence of data points in clusters. Formally, we define such an event as an *Evolution Event*. In order to determine such evolution events, we determine the higher rate at which data points points have arrived to this cluster in the previous time window of length H, as compared to the that even before it. A parameter α is used in order to measure this evolution rate.

DEFINITION 5. (EVOLUTION EVENT) *An evolution event over horizon H at current time t_c is said to have occurred at threshold α for cluster $\mathcal{C}_i$, if the ratio of the relative presence of points in cluster $\mathcal{C}_i$ over the horizon $(t_c - H, t_c)$ to that before time $t_c - H$ is greater than the threshold α. In other words, we have:*

$$\frac{F(t_c - H, t_c, \mathcal{C}_i)}{F(t(\mathcal{C}_i), t_c - H, \mathcal{C}_i)} \geq \alpha \tag{2.1}$$

Furthermore, it is assumed that $t_c - 2 \cdot H \geq t(\mathcal{C}_i)$.

We assume that the value of $t_c - 2 \cdot H$ is larger than the cluster creation time $t(\mathcal{C}_i)$ in order to define the afore-mentioned evolution ratio in a stable way. This ensures that at least H units of time are used in the computation of the denominator of this ratio.

Algorithm *SocialStreamClustering*(NumClusters: k);
begin
  Initialize clusters $\mathcal{C}_1 \ldots \mathcal{C}_r$ to null;
  Initialize i, μ, σ, M_0, M_1, M_2 to 0;
  repeat
    $i = i + 1$;
    Receive next social stream object S_i;
    for each cluster $\mathcal{C}_l$ compute $Sim(S_i, \mathcal{C}_l)$;
    Let r be the index of cluster $\mathcal{C}_r$ with largest similarity to S_i;
    if $(Sim(S_i, \mathcal{C}_r) < \mu - 3 \cdot \sigma)$ **then** replace most stale
      cluster with a new cluster containing the single point S_i;
    else add S_i to $\mathcal{C}_r$ and update statistics $\psi_r(\mathcal{C}_r)$ of cluster $\mathcal{C}_r$;
    Update M_0, M_1, M_2 additively;
    $\mu = M_1/M_0$;
    $\sigma = \sqrt{M_2/M_0 - \mu^2}$;
  until(*end_of_stream*);
end

Figure 1: Social Stream Clustering for Event Detection

3 Social Stream Clustering

The design of an effective online clustering algorithm is the key to the event detection process. Therefore, we will first focus on the problem of online clustering. Then, we will discuss how to leverage this clustering for event detection. We assume that the clustering algorithm uses the number of clusters k as input, and maintains the structural and content information in the underlying clusters in the form of node and word frequencies in the cluster. The clusters are denoted by $\mathcal{C}_1 \ldots \mathcal{C}_k$. We assume that the set of nodes associated with the cluster $\mathcal{C}_i$ is denoted by V_i, and the set of words associated with it is denoted by W_i. The set V_i is referred to as the node summary, whereas the set W_i is referred to as the word summary. As we will see later, this summary characterization can be used for assignment of incoming social stream objects to clusters. The set V_i contains the nodes $j_{i1}, j_{i2} \ldots j_{is}$, together with node frequencies denoted by $\nu_{i1} \ldots \nu_{is}$. The word set W_i contains the word identifiers $l_{i1}, l_{i2}, \ldots l_{is}$ together with word frequencies denoted by $\phi_{i1}, \phi_{i2} \ldots \phi_{is}$. One challenge with maintaining node summaries in large social networks is that the number of nodes can be extremely large (in the hundreds of millions), as a result of which the summary may be too large to use

efficiently, especially in the online context. Later, we will discuss how to use sketch-based methods in order to compute the similarities more effectively. First, we will discuss a more straightforward method for online maintenance of the clusters with the direct use of the cluster-summary. Now, we will formally define the concept of cluster-summary.

DEFINITION 6. *The cluster-summary $\psi_i(\mathcal{C}_i)$ of cluster $\mathcal{C}_i$ is defined as follows:*

- *It contains the node-summary, which is a set of nodes $V_i = \{j_{i1}, j_{i2} \ldots j_{is_i}\}$ together with their frequencies $\eta_i = \nu_{i1} \ldots \nu_{is_i}$. The node set V_i is assumed to contain s_i nodes.*
- *It contains the content-summary, which is a set of word identifiers $W_i = \{l_{i1}, l_{i2}, \ldots l_{iu_i}\}$ together with their corresponding word frequencies $\Phi_i = \phi_{i1}, \phi_{i2} \ldots \phi_{iu_i}$. The content-summary W_i is assumed to contain u_i words.*

The overall summary is $\psi_i(\mathcal{C}_i) = (V_i, \eta_i, W_i, \Phi_i)$.

We design an online partition-based clustering methodology, in which a set of clusters $\mathcal{C}_1 \ldots \mathcal{C}_k$ are maintained together with their cluster summaries $\psi_1(\mathcal{C}_1) \ldots \psi_k(\mathcal{C}_k)$. As new social stream objects arrive, the clusters are continuously updated. At the same time, the changes in the underlying clusters are continuously tracked and used in order to raise alarms for new events. First, we will discuss how the clusters are maintained. We will discuss the process of constructing event alarms from cluster statistics slightly later.

In each iteration, we compute the similarity of the incoming social stream object S_i to each cluster summary $\psi_i(\mathcal{C}_i)$. The details of the similarity computation will be provided later. The incoming stream object is then assigned to its closest cluster, unless the closest similarity value is significantly lower than that attained for the stream objects encountered so far. In order to determine if the closest similarity value is significantly lower than that attained by previous stream objects, we maintain the mean μ and standard deviation σ of all closest similarity values of incoming stream objects to cluster summaries. The similarity value is said to be significantly below the threshold, if it is less than $\mu - 3 \cdot \sigma$. We will describe the process of dynamically maintaining μ and σ at a later stage of the paper.

If the similarity value of the closest cluster is above the threshold of $\mu - 3 \cdot \sigma$, then we can assign the incoming stream object S_i to its closest cluster centroid. Once the stream object S_i is assigned to its closest cluster centroid $\mathcal{C}_r$, we update the corresponding cluster summary $\psi_r(\mathcal{C}_r)$. Specifically, any new nodes in S_i, which are not already included in V_r are added to V_r, and the frequency of the nodes of S_i which are included in V_r are incremented by 1. We note that the nodes in S_i correspond to both the source node q_i and the destination nodes R_i. In other words, the set $R_i \cup \{q_i\}$ is used to update the set V_r and its member frequencies. The same approach is applied to the words in W_r with the use of the words in the social stream object S_i. The only difference in this case is that the frequencies of the words are not incremented only by 1, but by their frequency of presence in the underlying document. On the other hand, if the similarity of S_i to $\mathcal{C}_r$ is greater than $\mu - 3 \cdot \sigma$, then we create a singleton cluster containing only the object S_i and corresponding cluster summary statistics. This cluster replaces the most stale cluster from the current collection $\mathcal{C}_1 \ldots \mathcal{C}_k$. The most stale cluster is defined as the one which was updated the least recently. In the event that a null cluster exists (which has never been updated), it is automatically considered the most stale cluster. Any ties are broken randomly.

It remains to explain how the similarity of the stream object S_i is to computed to the cluster $\mathcal{C}_r$. In order to compute the similarity, we need to compute both the structural $SimS(S_i, \mathcal{C}_r))$ and the content components $SimC(S_i, \mathcal{C}_r)$ of the similarity value. The content-components of the similarity is straightforward, and is simply the tf-idf based [18] similarity between the content T_i (belonging to social stream object S_i) and the content W_r. Next, we discuss how to compute the structural similarity between the nodes V_r and the nodes $R_i \cup \{q_i\}$ in the social stream. Let $B(S_i) = (b_1, b_2, \ldots b_{s_r})$ be the bit-vector representation of $R_i \cup \{q_i\}$, which has a bit for each node in V_r, and in the same order as the frequency vector $\eta = (\nu_{r1}, \nu_{r2}, \ldots \nu_{rs_r})$ of V_r. The bit value is 1, if the corresponding node is included in $R_i \cup \{q_i\}$ and otherwise it is 0. The structural similarity between the object S_i and the frequency-weighted node set of cluster $\mathcal{C}_r$ is defined as follows:

$$SimS(S_i, \mathcal{C}_r) = \frac{\sum_{t=1}^{s_r} b_t \cdot \nu_{rt}}{\sqrt{||R_i \cup \{q_i\}||} \cdot (\sum_{t=1}^{s_r} \nu_{rt})} \tag{3.2}$$

Note that we are using the L_1-norm of the node-frequency vector in the denominator as opposed to the normal use of the L_2-norm in order to penalize the creation of clusters which are too large. This will result in more balanced clusters.

The overall similarity $Sim(S_i, \mathcal{C}_r)$ is computed as a linear combination of the structural and content-based similarity values.

$$Sim(S_i, \mathcal{C}_r) = \lambda \cdot SimS(S_i, \mathcal{C}_r) + (1 - \lambda) \cdot SimC(S_i, \mathcal{C}_r) \tag{3.3}$$

The parameter λ is the balancing parameter, and lies in the range $(0, 1)$. This parameter is specified by the user.

It remains to explain how we maintain the mean and standard deviation of the (closest) similarity values of the incoming objects to the clusters. For this purpose, we maintain the zeroth, first and second order moments M_0, M_1 and M_2 of the closest similarity values continuously.

These values can be easily maintained in the stream scenario, because they can be *additively* maintained over the data stream. The mean μ and standard deviation σ can be expressed in terms of these moments as follows:

$$\mu = M_1/M_0, \quad \sigma = \sqrt{M_2/M_0 - (M_1/M_0)^2}$$

The overall algorithm for cluster maintenance is illustrated in Figure 1. The main challenge of this algorithm is that the node based statistics can be rather large and the corresponding similarity computations cumbersome. For example, the number of nodes in V_r may be of the order of tens of millions and this can make the algorithm extremely slow. Therefore, we will design a sketch-based technique in order to speed up the computations.

3.1 Sketch-based Speedup In this section, we discuss the sketch-based technique for maintaining node statistics. Sketch based techniques [10] are a natural method for compressing the counting information in the underlying data so that the broad characteristics of the dominant counts can be maintained in a space-efficient way. In this paper, we will apply the count-min sketch [10] for maintaining node counts in the underlying clusters. In the count-min sketch, a hashing approach is utilized in order to keep track of the node counts in the underlying data stream. We use $w = \lceil \ln(1/\delta) \rceil$ pairwise independent hash functions, each of which map onto uniformly random integers in the range $h = [0, e/\epsilon]$, where e is the base of the natural logarithm. The data structure itself consists of a two dimensional array with $w \cdot h$ cells with a length of h and width of w. Each hash function corresponds to one of w 1-dimensional arrays with h cells each. In standard applications of the count-min sketch, the hash functions are used in order to update the counts of the different cells in this 2-dimensional data structure. For example, consider a 1-dimensional data stream with elements drawn from a massive set of domain values. For example, in our application, this domain of values corresponds to the different node identifiers in the social network. When a new element of the data stream is received, we apply each of the w hash functions to map onto a number in $[0 \ldots h-1]$. The count of each of the set of w cells is incremented by 1. In order to *estimate* the count of an item, we determine the set of w cells to which each of the w hash-functions map, and compute the minimum value among all these cells. Let c_t be the true value of the count being estimated. We note that the estimated count is at least equal to c_t, since we are dealing with non-negative counts only, and there may be an over-estimation because of collisions among hash cells. As it turns out, a probabilistic upper bound to the estimate may also be determined. It has been shown in [10], that for a data stream with T arrivals, the estimate is at most $c_t + \epsilon \cdot T$ with probability at least $1 - \delta$.

In order to use the count-min sketch for improving the node-count estimation process, we maintain a sketch table for each cluster in the data. The sketch table is used for the purpose of maintaining the frequency counts of the nodes in the incoming data stream. Specifically, the sketch table for the jth cluster is denoted by U_j. If desired, we can use the same set of w hash functions for the different clusters. The main condition is that the set of w hash functions should be independent of one another. For each incoming object S_i, we update the sketch table for the cluster to which it is assigned on the basis of the similarity measure. We apply the w different hash functions to the (string representation of the identifier of the) nodes in $R_i \cup \{q_i\}$, and add 1 to the counts of the corresponding cells. Thus, for the incoming object R_i, we need to apply each of hash functions to the different $|R_i| + 1$ different nodes, and update the corresponding cells. This corresponds to an application of $(|R_i| + 1) \cdot w$ hash function instantiations and corresponding cell updates.

The sketch-based structure can also be used to effectively *estimate* the similarity value $SimS(S_i, \mathcal{C}_r)$. We note that this similarity computation needs to be performed for each cluster $\mathcal{C}_r$, and its corresponding sketch table U_r in order to determine the closest cluster to the incoming object based on the composite similarity measure. The denominator of $SimS(S_i, \mathcal{C}_r)$ can be exactly estimated, because the object S_i is known, and therefore the value of $\sqrt{|R_i \cup \{q_i\}|}$ can also be known exactly. The value of $\sum_{t=1}^{s_r} \nu_{rt}$ can also be known exactly, because it is simply equal to the sum of all the values in the sketch table cells in U_r for any one of the w hash functions. Thus, this value may be obtained exactly by summing up the h cells for any particular[1] one of the hash functions. On the other hand, the numerator needs to estimated approximately. Note that the numerator is essentially defined by the sum of the estimated values of the frequencies of the nodes included in $R_i \cup \{q_i\}$.

The frequency of each such node can be estimated in a manner discussed in [10]. Specifically, for each node included in $R_i \cup \{q_i\}$, the corresponding cluster-specific frequency of the node can be obtained by applying the hash function to the identifier of each node. We note that the value of the corresponding hash cell will always be an overestimate because of collisions between the different node identifiers to the same hash cell. Therefore, the minimum of these values across the w different hash functions will also be an overestimate, though it will be much tighter and more robust because of the use of different hash functions. We sum up these estimated frequency values over the different nodes in $R_i \cup \{q_i\}$. This is essentially the estimate of the numerator. This estimation of the numerator can be used in conjunction with our exact knowledge about the different denominator values in order to create an estimate of $SimS(S_i, \mathcal{C}_r)$. Let $EstSimS(S_i, \mathcal{C}_r)$ represent the *estimated* similarity of S_i to

[1] The sum of the h cells would be the same, no matter which hash function is picked.

$\mathcal{C}_r$ with the use of the sketch-based approach. Then, we can show the following result:

LEMMA 3.1. *If a sketch-table with length h and width w is used, then for some small value $\epsilon > \sqrt{|R_i|+1}/h$, the estimated value of the similarity $EstSimS(S_i, \mathcal{C}_r)$ is bounded to the following range with probability at least $1 - \left(\frac{\sqrt{|R_i|+1}}{h \cdot \epsilon}\right)^w$:*

(3.4)
$$SimS(S_i, \mathcal{C}_r) \leq EstSimS(S_i, \mathcal{C}_r) \leq SimS(S_i, \mathcal{C}_r) + \epsilon$$

Proof. As discussed earlier in Equation 3.2, the structural similarity is given by the following equation:

$$SimS(S_i, \mathcal{C}_r) = \frac{\sum_{t=1}^{s_r} b_t \cdot \nu_{rt}}{\sqrt{||R_i \cup \{q_i\}||} \cdot (\sum_{t=1}^{s_r} \nu_{rt})} \tag{3.5}$$

We note that the numerator is (over)-estimated approximately with the use of the sketch-based process, whereas the denominator can be known exactly. It is evident that $SimS(S_i, \mathcal{C}_r) \leq EstSimS(S_i, \mathcal{C}_r)$ because of the over-estimation of the numerator. It remains to show that $EstSimS(S_i, \mathcal{C}_r) \leq SimS(S_i, \mathcal{C}_r) + \epsilon$ with probability at least $1 - \left(\frac{\sqrt{|R_i|+1}}{h \cdot \epsilon}\right)^w$.

Let $SimSN(S_i, \mathcal{C}_r)$ and $EstSimSN(S_i, \mathcal{C}_r)$ be the estimation of the numerator with the sketch-based approach. Then, since the denominator can be computed exactly, we have:

$$EstSimS(S_i, \mathcal{C}_r) = \frac{EstSimSN(S_i, \mathcal{C}_r)}{\sqrt{|R_i|+1} \cdot (\sum_{t=1}^{s_r} \nu_{rt})} \tag{3.6}$$

Furthermore, we have:

$$SimS(S_i, \mathcal{C}_r) = \frac{SimSN(S_i, \mathcal{C}_r)}{\sqrt{|R_i|+1} \cdot (\sum_{t=1}^{s_r} \nu_{rt})} \tag{3.7}$$

Therefore, in order to prove the result in the lemma we need to prove bounds on the approximation in the numerator. Specifically, we **need to prove that** the following holds true with probability at least $1 - \left(\frac{\sqrt{|R_i|+1}}{h \cdot \epsilon}\right)^w$.

(3.8)
$$EstSimSN(S_i, \mathcal{C}_r) \leq SimSN(S_i, \mathcal{C}_r) + \epsilon \cdot \sqrt{|R_i|+1} \cdot (\sum_{t=1}^{s_r} \nu_{rt})$$

Let us abbreviate one of the terms on the right hand side of the above equation by $B = \epsilon \cdot \sqrt{|R_i|+1} \cdot (\sum_{t=1}^{s_r} \nu_{rt})$. In other words, we need to show the above bound on the probability for the condition that:

$$EstSimSN(S_i, \mathcal{C}_r) - SimSN(S_i, \mathcal{C}_r) \leq B \tag{3.9}$$

The expected value of the estimation of $SimSN(S_i, \mathcal{C}_r) - SimSN(S_i, \mathcal{C}_r)$ *with the use of any particular hash function* is $(|R_i|+1) \cdot (\sum_{t=1}^{s_r} \nu_{rt})/h$. This is because, we need to sum up the errors over $|R_i|+1$ different frequency estimations, and the expected number of collisions for any of these cell-based estimations is $(\sum_{t=1}^{s_r} \nu_{rt})/h$. Then, we can use the Markov inequality to show that the probability that the condition in Equation 3.9 is violated *with the use of* 1 *hash function* with probability at most $\frac{(|R_i|+1) \cdot (\sum_{t=1}^{s_r} \nu_{rt})}{B \cdot h}$. We can generalize the probability of this to w independent hash functions to at most $\left(\frac{(|R_i|+1) \cdot (\sum_{t=1}^{s_r} \nu_{rt})}{B \cdot h}\right)^w$. Therefore, the condition in Equation 3.9 holds with probability at least $1 - \left(\frac{(|R_i|+1) \cdot (\sum_{t=1}^{s_r} \nu_{rt})}{B \cdot h}\right)^w$. By substituting the value of B in the above equation, we get the desired result.

The above result suggests that the value of the similarity can be estimated quite accurately with the use of modest memory requirements of a sketch table. For example, consider a tweet with $|R| \approx 100$, and a similarity estimation bound of $\epsilon = 0.001$. If we use a sketch table with $h = 200,000$ and $w = 5$ (typical values), it will require a storage of only 1 million cells, which is in the megabyte order. The similarity estimate lies within $\epsilon = 0.001$ of the true value with probability at least $1-(1/20)^5 > 1-10^{-6}$. In practice, since these theoretical bounds are quite loose, the estimates are much better.

3.2 Event Detection with Clustering The clustering method discussed above can be used directly in order to perform the event detection. As discussed before, the event detection algorithm uses a time horizon H as the input which is used for the event detection process. In order to perform the event detection, we monitor the ratio $\frac{F(t_c-H, t_c, \mathcal{C}_i)}{F(t(\mathcal{C}_i), t_c-H, \mathcal{C}_i)}$ for each cluster $\mathcal{C}_i$ continuously over time, and trigger an alarm whenever this ratio exceeds the threshold of α. This suggests a significant change in the underlying social stream, which is detected by a significant change in the ratios of stream objects being assigned to the different clusters.

3.2.1 Supervised Event Detection The afore-mentioned approach discusses the case of *unsupervised* event detection in which we only try to find significant events in the social network without any particular regard to their nature. In practice, one may want to detect *known events* which have been encountered earlier in the stream. This is the case of *supervised event detection*. In this case, we assume that we have access to the past history of the stream in which the event $\mathcal{E}$ has been known to have occurred. In addition, we have information about at least a subset of the social stream tweets which are relevant to this particular event. This is the ground truth which can be leveraged for more accurate event detection.

In order to perform supervised event detection, we need to make some changes to the clustering portion of the

algorithm as well. One major change is that we do not allow replacement of old clusters or creation of new clusters when a new incoming point does not naturally fit in any cluster. Rather, it is always assigned to its closest cluster, with any ties broken randomly. This is done in order to provide stability to the clustering characteristics, and is essential for characterizing the events in a consistent way over time with respect to the clusters in the underlying data.

The *relative distribution of event-specific stream objects* to clusters is used as a signature which is specific to the event. These can be used in order to perform the detection in real time. The assumption in the supervised case is that the training data about the social stream objects which are related to the event are available in the historical training data. The signature of the event $\mathcal{E}$ is defined as follows.

DEFINITION 7. (EVENT SIGNATURE) *The event signature of a social stream is a k-dimensional vector $V(\mathcal{E})$ containing the (average) relative distribution of event-specific stream objects to clusters. In other words, the ith component of $V(\mathcal{E})$ is the fraction of event-specific (training) stream objects which are assigned cluster i.*

Clearly, the event signature provides a useful characterization of the relative topical distribution during an event of significance. For example, during a period of mideast unrest (the event $\mathcal{E}$), some clusters are likely to be much more active than others, and this can be captured in the vector $V(\mathcal{E})$, as long as ground truth is available to do so. The event signatures can be compared to *horizon signatures*, which are essentially defined in the same way as the event signatures, except that they are defined over the more recent time horizon $(t_c - H, t_c)$ of length H.

DEFINITION 8. (HORIZON SIGNATURE) *The horizon signature over the last time period $(t_c - H, t_c)$ is a k-dimensional vector containing the relative distribution of social stream objects to clusters which have arrived in the period $(t_c - H, t_c)$.*

In order to perform the supervised event detection, we simply compute the dot product of the horizon signature with the known event signature (which was computed by the ground truth), and output an alarm level which is equal to this value. The tradeoff between false positives and false negatives is determined by the threshold chosen to decide when such an event has really occurred.

4 Experimental Results

In this section, we will study the clustering and event detection algorithms for effectiveness and efficiency. We used two real data sets to evaluate the effectiveness of our approach.

4.1 Data Sets Our algorithm was tested on the following two data sets:

Twitter Social Stream: This is a stream of tweets which was crawled from the *Twitter* social network. Each social object contains the network structure and content in a tweet. The stream contained a total of 1,628,779 tweets, which were distributed over a total of 59,192,401 nodes. The nodes include the sender and the receivers either in the form of direct mentions from senders or their followers in the case of broadcast messages. On the average, each stream object contained about 84 nodes per tweet.

Enron Email Stream: The well known Enron email data set[2] was converted to a stream with the use of the time stamp information in the emails. Each object contained the text of the email, and the network structure corresponding to the sender and receiver(s). In this sense, the network structure of an email is very similar to a tweet with a single sender and multiple receivers. The *Enron* email data stream contained a total of 517,432 emails. We eliminated emails that did not have valid sender and receiver email identifiers. We also filtered out the calender invites, duplicate emails, and the email history at the bottom of each email. The total emails after the filtering process were 349911, which were distributed over a total of 29,083 individuals. On the average, each email contained 3.62 receivers.

4.2 Evaluation Measures We tested both the clustering and event detection methods for effectiveness. For the case of efficiency, we tested only the clustering method, because the majority of the time for event detection was spent in the clustering process. In order to test the effectiveness, we used a number of class labels which were associated with the objects in the social stream. For the case of the *Twitter* stream, these class labels were the hash tags which were associated with the tweet. The most frequent hash tags in the stream often correspond to events in the stream such as *Uganda protests* or the *Japan Earthquake*, and represent meaningful characteristics of the underlying objects which ought to be clustered together. These hash tags also often represent the meaningful events in the stream. We note that not every tweet may contain hash tags, and therefore, the hash tags were only associated with a subset of the tweets. For the case of the *Enron* email stream, the class labels were defined by the most frequently occurring tokens in the subject line. These tokens were as follows:

meeting, agreement, gas, energy, power, report, update, request, conference, letter, deal, credit, california, trading, contract, project, presentation, houston, announcement

All emails which contained any of the above tokens in the subject line were tagged with the corresponding class label. Thus, for both data streams, a subset of the stream objects were tagged with a class label. Clearly, clusters of high quality would tend to put objects with similar tags in

[2] http://www.cs.cmu.edu/~enron/

the same cluster. Therefore, we computed the dominant class purity of each cluster. For each cluster, we determined the tag with the highest presence, and computed the fraction of the (tagged) cluster objects, which belonged to that label. This value was averaged over the different clusters in a weighted way, where the weight of a cluster was proportional to the number of (tagged) objects in it.

We also tested the efficiency of the social stream clustering process. In order to test efficiency, we computed the number of social stream objects processed per unit of time, and we presented this number for the different algorithms.

We also also tested the effectiveness of the event detection algorithm. For the case of the unsupervised algorithm, we will present a case study, which illustrates the interesting events found by the technique. This provides an intuitive idea about the effectiveness of the technique. For the supervised algorithm, we first need to generate the ground truth for the supervised algorithm. For this purpose, we used the hash tags in the *Twitter Stream* in order to generate a 0-1 bit stream corresponding to when the events actually occurred. We used the two hash tags corresponding to $\#japan$ and *#uganda* in order to generate the events in our *Twitter* data stream. Specifically, at each time stamp, we looked at the past window of length h and counted the number of occurrences of a particular hash tag in that window. If the number of occurrences of the hash tag was at least 3, then we generated a bit of 1 at that time stamp to indicate that the event has indeed occurred.

We note that our supervised event detection algorithm generates a *continuous* alarm level. It is possible to use a threshold t on this continuous alarm level in order to generate a 0-1 bit stream corresponding to the *algorithmic prediction* of when the event has occurred. By using different thresholds t on this alarm level, we can obtain different tradeoffs between precision and recall. Let $S_F(t)$ be the set of time stamps at which an alarm is generated with the use of threshold value of t on the real alarm level. Let S_G be the ground truth set of time stamps at which the event truly occurs. Then, $Precision(t)$ and $Recall(t)$ can be computed as follows:

$$Precision(t) = \frac{|S_F(t) \cap S_G|}{|S_F(t)|} \tag{4.10}$$

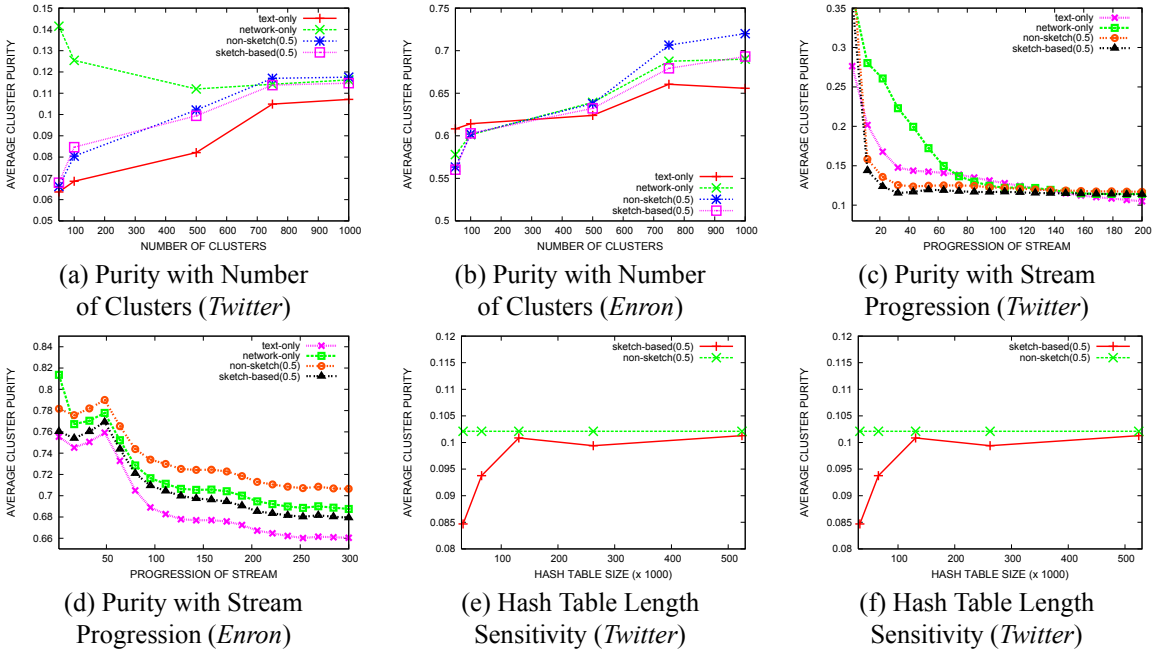
$$Recall(t) = \frac{|S_F(t) \cap S_G|}{|S_G|} \tag{4.11}$$

(4.12)

We presented the tradeoff between the precision and recall by varying the value of t in order to generate a plot between the two.

4.3 Algorithm Variations and Baseline We tested different algorithm settings to determine the effect of content and network structure on accuracy and efficiency. We note that by setting λ to the extreme values of 0 and 1, we can test how well the algorithm performs, when we use either the network only, or the text only for the clustering process. We also tested a combination scheme in which the value of λ was set to 0.5. This kind of scheme provides equal weight to the text and content in the clustering and event detection process. For the combination scheme, we tested it both with and without the sketch in order to estimate the effects of sketch use on the scheme in terms of accuracy and efficiency. We note that the text-only variation can be considered quite similar to the stream text-clustering approach discussed in [1]. Therefore, this variation also serves as a good baseline, because the use of pure text content is the only natural alternative for this problem at this juncture.

4.4 Effectiveness Results for Clustering First, we will present the effectiveness results of the clustering algorithm in terms of the cluster purity. We tested the effectiveness of the approach with increasing number of clusters. The results for the *Twitter* and *Enron* streams are illustrated in Figure 2(a) and (b) respectively. The sketch table length h was set to 262,213, whereas the sketch table width w was set to 2 for the *Twitter* social stream, and these values were set to 16,369 and 2 for the *Enron* data stream. In each case, we have illustrated the number of clusters on the X-axis, and the cluster purity on the Y-axis. A very interesting trend was observed for both the data sets in terms of the relative performance of the different algorithms. In all cases, the algorithm which used only text performed the worst among all the algorithms. The trends between the purely network-based approach and combined approach were dependent upon the level of granularity at which the clustering was performed. For both data streams, we found that when a small number of clusters were used, the network-based approach was superior. When a larger number of clusters were used, the combination methods outperformed the purely network-based approach. This is because the network locality information is more than sufficient to effectively partition the clusters, when the granularity is relatively coarse. In such cases, the addition of text does not improve the quality of the underlying clusters, and can even be detrimental to clustering quality. However, when the number of clusters increases, the combination approach tends to perform better, because the granularity of the clusters is much higher, and therefore more attributes are required to distinguish between the different clusters. This is particularly evident in the case of the *Enron* data stream, in which the gap between the combination-approach and purely network-based approach is rather large. In all cases, we found that the use of sketches lead to some loss of accuracy. However, this loss of accuracy is not very significant, especially when we consider the fact that the sketch-based approach was significantly faster. It is also important to note that the pure text-based approach (which is



(a) Purity with Number of Clusters (*Twitter*)

(b) Purity with Number of Clusters (*Enron*)

(c) Purity with Stream Progression (*Twitter*)

(d) Purity with Stream Progression (*Enron*)

(e) Hash Table Length Sensitivity (*Twitter*)

(f) Hash Table Length Sensitivity (*Twitter*)

Figure 2: Effectiveness Results for Clustering

our baseline) performed the worst in all scenarios. These results also seem to suggest that the network information in the social stream provides much more powerful information for clustering as compared to the text-information. This is not particularly surprising for sources such as the *Twitter* data stream in which the text is rather noisy and often contains non-standard acronyms or other text which are hard to use in a meaningful way. However, we found it somewhat interesting and surprising that these trends were also true for a source such as the *Enron* data stream, in which the text was relatively clean and was usually quite informative for the underlying class labels.

We have also illustrated the cluster purity with progression of the stream. This provides a *dynamic idea* of how the clustering process performed during the progression of the stream itself. The results for the *Twitter* and *Enron* data streams are illustrated in Figures 2(c) and (d) respectively. The number of clusters was fixed to 750. For the *Twitter* data stream the sketch table length was fixed to 262,213 and for *Enron* it was set to 16,369. The sketch table width was fixed to 2 in both cases. The X-axis illustrates the progression of the stream for every 1000 objects processed, and the Y-axis illustrates the cluster purity in the last time window of 1 hour. The relative trends between the different methods are similar to our earlier observations, though the main observation is that the cluster purities generally reduce with progression of the stream. The reason for this is that the number of class labels in the stream generally increase over the progression of the stream, as new classes, tags and events are encountered. As a result, the cluster purity also generally reduces with stream progression.

Finally, we also tested the sensitivity of the approach with sketch-table length and width. The sensitivity results with increasing sketch-table length for the *Twitter* data stream is illustrated in Figures 2(e). The sketch table length is illustrated on the X-axis, whereas the cluster purity is illustrated on the Y-axis. The number of clusters was set to 500 in this case and sketch table width was set to 2. We have also shown the results for the method which does not use the sketch in the same figure in order to provide a baseline for the relative effectiveness of the incorporation of the sketch structure. It is clear that the cluster-purity increases with increasing sketch-table length. This is because a larger sketch-table length reduces the number of collisions in the sketch table, and it therefore improves the overall accuracy. We also note that when the sketch-table length is increased sufficiently, the accuracy of the approach based on the sketch-table approaches that of the method which does not use the sketch table. In such cases, the collisions reduce sufficiently to the point that there of reduction in accuracy because of sketch=table use.

We also tested the sensitivity of the approach with increasing sketch-table width. The sensitivity results with increasing sketch-table width for the *Twitter* data stream is illustrated in Figures 2(f). The number of clusters was set to 500 and sketch-table length was set to 262,213. While the effectiveness results improve with increasing sketch-table width, it is evident that the purity results are not quite as

sensitive to sketch-table width, as they were to the sketch-table length. This is because an increase in the number of hash functions provides additional robustness, but it does not drastically reduce the number of collisions between the different items.

4.5 Efficiency Results for Clustering We also tested the efficiency of the clustering approach with increasing number of clusters. The parameter settings were the same as those of the effectiveness results. The efficiency results for the *Twitter* and *Enron* data streams are illustrated in Figures 3(a) and (b) respectively. The X-axis denotes the number of clusters, whereas the Y-axis denotes the number of stream objects processed every hour. It is evident that the network-based approach was much slower than the text-based approach. This is because of the large number of distinct nodes which need to be processed by a text-based approach. However, the sketch-based approach is significantly faster in both data streams, because of the fact that the number of operations in the similarity computation are reduced by the sketch representation. Another interesting observation is that the processing rate did not necessarily reduce with increasing number of clusters. Much of this was also because the a larger number of clusters resulted in faster similarity computations between incoming objects and the underlying clusters. This is because a larger number of clusters resulted in sparser clusters with fewer number of objects in each cluster.

We also tested the efficiency with stream progression and present the results in Figure 3(c) and (d) respectively. The stream progression is illustrated on the X-axis, and the processing rate is illustrated on the Y-axis. It is evident that the processing rate reduces with stream progression for all the different methods. This is because the clusters contain a larger number of objects with progression of the stream. This increases the complexity of the similarity computations because the number of attributes in each cluster (in terms of the number of text words or nodes) increases as well. However, this slow-down levels out after a certain point as the number of attributes in each cluster stabilizes.

We also tested the sensitivity of the approach with hash-table length and width. The sensitivity results with hash-table length and width for the *Twitter* data streams are illustrated in Figures 3(e), and (f). It is evident from the figures that the running time is not very sensitive to the hash table length, and most of the variations are really random variations. On the other hand, the hash table width affects the number of hash functions which need to be computed. Therefore, the running time generally reduces with increasing hash table width. These results seem to suggest that a greater hash table width is not particularly useful because it does not affect the purity very much, but it affects the efficiency substantially. Therefore additional memory should be used for increasing hash table length rather than width.

4.6 Unsupervised Event Detection Case Study In this section, we provided a case study of the unsupervised event detection problem. Both evolutionary and novel events were detected with the use of this approach. Typically, such events were associated with a particular news, country, language or mood.

The first event was related to the Japan nuclear crisis. In particular, the relevant segment of the social stream corresponds to the case where the Japanese Prime Minister requested the Chubu electric company to shut down the Hamaoka nuclear plant. This event generated considerable chatter in the social stream, in which the underlying actors were discussing the positives and negatives of this directive. The corresponding portion of the social stream contained structural nodes which were geographically biased towards Japan, and generally created clusters of their own on the basis of the underlying network and content information. The frequent text content in these clusters included the following words:

nuclear, hamaoka, plant, concerns, dilemma, gaswat, hairline, halt, shut, heavy, japan, neglect, operation (1.152)

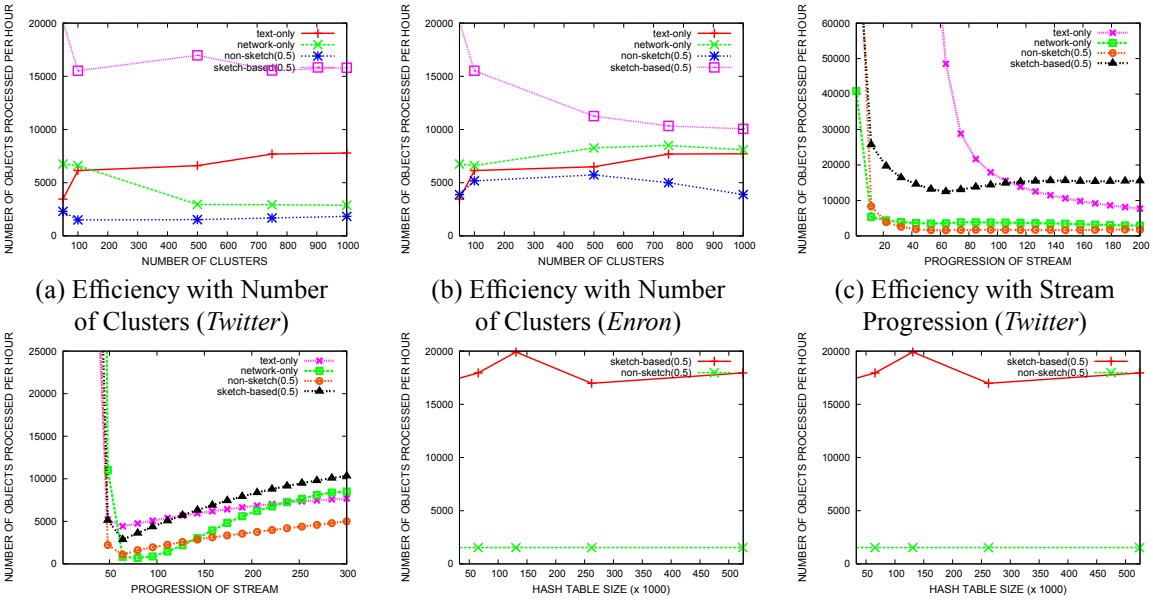
One interesting aspect of our algorithm was that it was able to detect events, for which the content was in a foreign language. The reason for this is that the event detection algorithm does not use any methods which are specific to English, and the use of network structure is blind to the use of a specific language. For example, the minister for finance in Indonesia issued an order on May 9, 2011 to buy 7% of the shares in PT Newmont Nusa Tenggara company. This triggered discussion threads in twitter which were captured because of their related content and network structure. The frequent text-content which was associated in the cluster most related to this event was the following:

keputusan, menkeu, beli, newmont, saham, wapres, didukung, challenge, minskade, metro, jak, kimiad, menos (0.9964)

We note that the entire text content in this event consists of foreign language words.

Our event detection algorithm was also capable of connecting related events. This is because the actors and events are often overlapping in such cases. This overlap could result in the placement of closely related events in the same cluster. An example of this were the protests on May 9, 2011 on the Gay bill death penalty in Uganda. Many of these same actors were also discussing issues related to the Kentucky anti-gay marriage law, even though the two events were geographically quite disparate. As a result, these two related events were reported as a single composite event, since they reflected issues which were discussed in relation to one another by the participants. The corresponding content words were as follows:

protest, power, sign, identity, much, please, political, citizens, petition,



(a) Efficiency with Number of Clusters (*Twitter*)

(b) Efficiency with Number of Clusters (*Enron*)

(c) Efficiency with Stream Progression (*Twitter*)

(d) Efficiency with Stream Progression (*Enron*)

(e) Hash Table Length Sensitivity (*Twitter*)

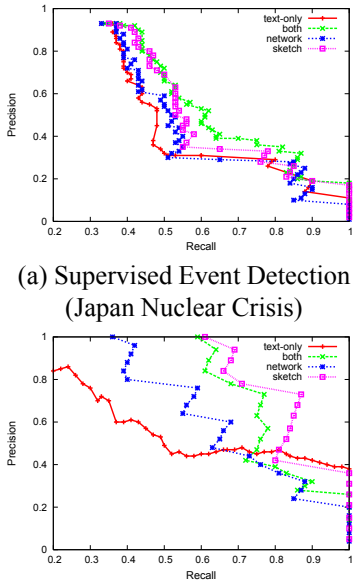
(f) Hash Table Length Sensitivity (*Twitter*)

Figure 3: Efficiency Results for Clustering

stop, kill, gays, bill, kentucky, progress, support (0.6490)

Thus, our unsupervised event detection approach was capable of discovering interesting and novel events in the underlying social stream. Such inference can be very useful in diagnosing important social moods which are related to real events.

4.7 Supervised Event Detection We also tested the supervised event detection method on the *Twitter* stream. To create the supervised events, we set the horizon to 5 minutes and determined the periods in which events corresponding to the *Japan Nuclear Crisis* and *Uganda Protest* occurred in the data stream. In each case, we set the number of clusters to 750 in order to detect the underlying events. We illustrated the tradeoff between the precision and recall on Figures 4(a) and (b) respectively. The recall is illustrated on the X-axis, whereas the precision is illustrated on the Y-axis. In each case, we have implemented the event detection algorithm with different variations of the algorithm. It is clear that the use of only text did not provide as accurate an event detection than all the methods which used network structure in the event detection process. In particular, the method which used both the network structure and the text content provided the most accurate results. The use of sketches degraded the accuracy to some extent, but the approach was still more accurate than that with the use of pure text. It is also evident that the absolute values of precision and recall were fairly high. For example, in the case of *Japan Nuclear event*, a precision of 0.525 was obtained at a recall point of approximately 0.62,



(a) Supervised Event Detection (Japan Nuclear Crisis)

(b) Supervised Event Detection (Uganda Protests)

Figure 4: Supervised Event Detection

when both network and text were used. On the other hand, the scheme which used only either text or network achieved a precision of about 0.3 at a recall point of around 0.6.

In the case of *Uganda Protests*, the differences between the different methods were even sharper. Both combination methods were extremely accurate, and maintained a precision of 1 at a recall point of around 0.6. This essentially means that the top reported alarms (in terms of alarm magnitude) were all correct up to the point where 60% of the correct alarm points were detected. On the other hand, the text-based baseline provided a precision of only about 0.5 at the same recall point. This means that about 50% of the reported alarms were incorrect at the same recall point. The purely network-based method was also not significantly better in this case. Thus, the use of a combination of network and text greatly improved the accuracy of the event detection algorithm. These results seem to suggest that our approach of combining network and text content for clustering and event detection can provide useful and accurate results in a wide variety of scenarios.

5 Conclusions and Summary

In this paper, we proposed new methods for clustering and event-detection in social streams. We show that the use of content- and network-stream based clustering and event detection has a number of fundamental advantages which cannot be easily handled by pure text-based methods, which are the current state-of-the-art. Our results suggest that social streams can be used as a valuable resource to monitor and detect relevant and interesting events in the social stream. We present experimental results on a number of real social streams, which illustrate the effectiveness of our approach.

Acknowledgements

Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

References

[1] C. Aggarwal, and P. Yu, *A Framework for Clustering Massive Text and Categorical Data Streams*, SDM, (2006).

[2] C. Aggarwal, *Data Streams: Models and Algorithms*, Springer, (2007).

[3] C. Aggarwal, *Social Network Data Analytics*, Springer, (2011).

[4] J. Allan, R. Papka, and V. Lavrenko, *On-line new event detection and tracking*, SIGIR Conf., (1998).

[5] J. Allan, V. Lavrenko, and H. Jin, *First story detection in tdt is hard*, CIKM Conf., (2000).

[6] D. Blei, and J. Lafferty, *Dynamic topic models*, ICML Conf., (2006).

[7] T. Brants, F. Chen, and A. Farahat, *A system for new event detection*, SIGIR Conf., (2003).

[8] D. Chakrabarti, R. Kumar, and A. Tomkins, *Evolutionary clustering*, KDD Conf., (2006).

[9] Y. Chi, X. Song, D. Zhou, K. Hino, and B. Tseng, *Evolutionary spectral clustering by incorporating temporal smoothness*, KDD Conf., (2007).

[10] G. Cormode and S. Muthukrishnan, *An Improved Data-Stream Summary: The Count-min Sketch and its Applications*, Jour. of Algorithms, 55(1), (2005).

[11] A. Clauset, M. Newman, and C. Moore, *Finding community structure in very large networks*, Phys. Rev. E, (2004).

[12] D. Cutting, D. Karger, J. Pedersen, and J. Tukey, *Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections*, SIGIR Conf., (1992).

[13] Q. He, K. Chang, E.-P. Lim, and J. Zhang, *Bursty feature representation for clustering text streams*, SDM Conf., (2007).

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

[15] C. Lin, B. Zhao, Q. Mei, and J. Han, *PET: a statistical model for popular events tracking in social communities*, KDD Conf., (2010).

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

[17] Q. Mei, and C.-X. Zhai, *Discovering Evolutionary Theme Patterns from Text - An Exploration of Temporal Text Mining*, KDD Conf., (2005).

[18] G. Salton, *An Introduction to Modern Information Retrieval*, Mc. Graw Hill, (1983).

[19] H. Sayyadi, M. Hurst, and A. Maykov, *Event Detection in Social Streams*, AAAI Conf., (2009).

[20] H. Schutze, and C. Silverstein, *Projections for Efficient Document Clustering*, SIGIR Conf., (1997).

[21] A. Surendran, and S. Sra, *Incremental Aspect Models for Mining Document Streams*, PKDD, (2006).

[22] X. Wang, C.-X. Zhai, X. Hu, and R. Sproat, *Mining Correlated Bursty Topic Patterns from Correlated Text Streams*, KDD Conf., (2007).

[23] Y. Yang, T. Pierce, and J. Carbonell, *A study on retrospective and on-line event detection*, SIGIR Conf., (1998).

[24] Y.Yang, J. Carbonell, and C. Jin, *Topic-conditioned Novelty Detection*, KDD Conf., (2002).

[25] T. Yang, R. Jin, Y. Chi, and S. Zhu, *Combining link and content for community detection: a discriminative approach*, KDD Conf., (2009).

[26] H. Kwak, C. Lee, H. Park, and S. Moon, *What is Twitter, a social network or a news media?*, WWW Conf., (2010).

[27] S. Zhong, *Efficient Streaming Text Clustering*, Neural Networks, Vol. 18, Issue 5-6, (2005).

[28] Y. Zhou, H. Cheng, and J. X. Yu, *Graph clustering based on structural/attribute similarities*, Proc. VLDB Conf., (2009).

[29] `http://projects.ldc.upenn.edu/TDT`