Content-Based Crowd Retrieval on the Real-Time Web

Krishna Y. Kamath
Texas A&M University
College Station, TX 77843
kykamath@cs.tamu.edu

James Caverlee
Texas A&M University
College Station, TX 77843
caverlee@cse.tamu.edu

ABSTRACT

In this paper, we propose and evaluate a novel content-driven crowd discovery algorithm that can efficiently identify newly-formed communities of users from the real-time web. Short-lived crowds reflect the real-time interests of their constituents and provide a foundation for user-focused web monitoring. Three of the salient features of the algorithm are its: (i) prefix-tree based locality-sensitive hashing approach for discovering crowds from high-volume rapidly-evolving social media; (ii) efficient user profile updating for incorporating new user activities and fading older ones; and (iii) key dimension identification, so that crowd detection can be focused on the most active portions of the real-time web. Through extensive experimental study, we find significantly more efficient crowd discovery as compared to both a k-means clustering-based approach and a MapReduce-based implementation, while maintaining high-quality crowds as compared to an offline approach. Additionally, we find that expert crowds tend to be "stickier" and last longer in comparison to crowds of typical users.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*Information networks*

General Terms

Algorithms, Experimentation

Keywords

clustering, social media, community detection, real-time web

1. INTRODUCTION

The real-time web has grown at an astonishing rate in the past several years. As one example, Twitter has rapidly grown from handling 5,000 tweets per day in 2007 to 50 million tweets per day in 2010 to 140 million per tweets per day in 2011. During the recent run-up and immediate aftermath of President Obama's announcement about Osama Bin Laden, Twitter boasted a peak of 5,000 tweets per second (corresponding to 432 million tweets per day) and a sustained average rate of 3,000 tweets per second over several hours (corresponding to 259 million tweets per day).[1] At an order of magnitude higher, Facebook reported in 2009 that it was handling around 1 billion chat messages per day,[2] and there is widespread evidence of massive growth in web-based commenting systems (like on Reddit, Digg, and NYTimes) and other real-time "social awareness streams".

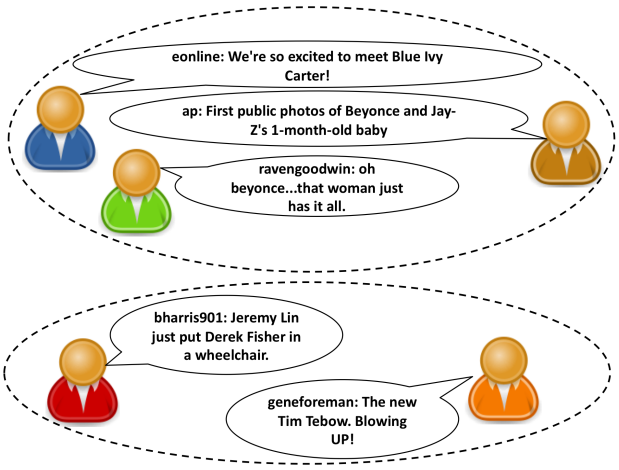
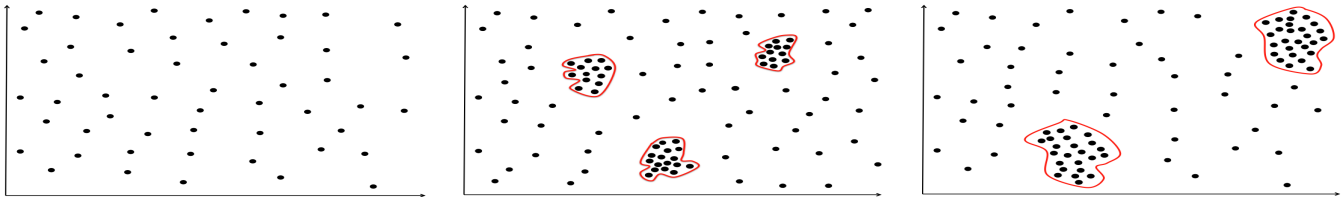


Figure 1: Examples of content based crowds.

While long-lived communities have been one of the key organizing principles of the Web, the real-time web supports the near instantaneous formation of ad-hoc communities linked by the real-time interests of their constituents. These "crowds" range from groups of loosely-connected Twitter users responding to a live presidential address, to users sharing pictures about a chemical fire at a nearby refinery, to flash mobs congregating at particular locations (and revealing this location via services like Foursquare), and so on. For example, Figure 1 shows example of two content based crowds, one discussing the public release of Jay-Z and Beyonce's baby pictures with 3 users (eonline, ap, ravengoodwin), and another crowd about NY Knicks vs LA Lakers basket ball game with 2 users (bharris901, geneforeman).

[1] http://blog.twitter.com/2011/03/numbers.html
[2] http://www.facebook.com/note.php?note_id=91351698919

CIKM'12, October 29–November 2, 2012, Maui, HI, USA.



(a) t_n (b) t_{n+1} (c) t_{n+2}

Figure 2: Example of user vectors in 2-dimensional space showing the evolution of users during three time intervals. Crowds are shown using a red boundary.

In contrast to traditional long-lived communities, these crowds are dynamically formed and potentially short-lived, often with only implicit signals of their formation within the massive scale of the real-time web. While much research has focused on community mining from large-scale systems [10, 21, 25, 26], of event detection on the Web and in social media [7, 8, 17, 24], and topic detection and tracking over streaming news items [3, 4, 12], there is a great opportunity to explore the near real-time extraction of crowds in the critical moments of their initial formation. Efficiently identifying these crowds of related users as they form and as they evolve can benefit many domains including epidemiological and disease control experts searching for evidence of new outbreaks and the reaction of the public to new vaccines, municipalities interested in responding to local events (like the recent Vancouver riots), finance experts monitoring stock price jumps or crashes, political scientists tracking chatter about presidential debates, as well as average users interested in local crowds (what restaurants are hot?) and crowds associated with particular topics (e.g., sports, movies).

In this paper we formalize the problem of crowd discovery over rapidly evolving social media and provide solutions for efficiently identifying crowds. Although we focus on text-based social media streams popularized by Twitter and related services, the discussion and techniques are designed for generic application to other temporally ordered social media resources. Concretely, this paper makes the following contributions:

- We present an efficient algorithm for identifying clusters of related users (*crowds*) from the real-time web using a prefix-tree based locality hashing approach.
- We describe an efficient method for updating user profiles in rapidly evolving social media as users post new messages.
- We show how to focus crowd detection via key dimension identification, so that crowd detection can be focused on the most active portions of the real-time web and so resources are not wasted.
- We evaluate the performance of the proposed crowd discovery algorithm over two Twitter datasets and we find the proposed approach is significantly faster than alternative approaches while maintaining high crowd quality.

2. RELATED WORK

In addition to the works cited in the introduction, there have been many efforts aimed at detecting cluster structure in text-based collections [19, 9, 5]. But, these approaches, however, are typically not designed for high-volume incrementally updated domains as on the real-time web. Alternatively, there is a large body of stream-oriented clustering work for finding correlations in streaming data. For example, StatStream [28] clusters evolving time series data using the Discrete Fourier Transform. Both [3] and [11] explore two-stage approaches for finding clusters in low dimensional data (unlike the case of text clustering, which typically is very high-dimensional due to the number of tokens observed). Clustering over text streams has been studied in [2, 18, 14]. These efforts have focused on the clustering of independent text elements (e.g., new messages), whereas our focus is on finding groups of related users by their sequences of related posts to the real-time web.

The solution approach in this paper relies on locality-sensitive hashing (LSH) for finding nearest-neighbors as a primitive for crowd detection. Nearest-neighbor and approximate nearest-neighbor search in a high-dimensional vector space is a difficult problem that Indyk and Motwani [15, 13] approach through the use of a family of randomized hash functions that generate similar vector signatures if the vectors are closer to each other in the high-dimensional space. In [6], Charikar constructed the LSH function for cosine similarity, which supports fast similarity between two high-dimensional vectors by reducing them to bit-arrays of much smaller dimensions. This result has been used in several problems, including efficient noun clustering [23, 20, 22]. In previous work, we studied crowd detection based on user communication, without regard for the content of the messages as we do here [16].

3. CROWD DISCOVERY: OVERVIEW AND SOLUTION APPROACH

Let $U = \{u_1, u_2, \ldots, u_i \ldots\}$ be a (potentially) unbounded set of users posting messages to a real-time web stream such as Twitter or Facebook. Each user may contribute an arbitrary number of messages, where the messages are ordered in a non-decreasing fashion using the time-stamp values of the messages. We say that a crowd $C = \{u_i, u_2 \ldots u_l\}$, at a given time, is defined as a subset of users that are close to each other at that time, where closeness is measured using a similarity function $sim(u_i, u_j)$. For example Figure 2, shows a simple scenario where users are mapped into

a 2-dimensional space (say, by using TF-IDF weights of the words in the messages). In the initial figure at time t_n, users are sparsely distributed in the space and there are no clear crowds. As users generate more messages, we see in the following two intervals the formation of several tight clusters of users ("crowds"). Intuitively, these crowds correspond to collections of users who are posting messages about similar topics (e.g., the Super Bowl on one day and Presidential elections the next day).

Given a user similarity measure $sim(\vec{u_i}, \vec{u_k})$ and a user similarity threshold ϵ, we formulate crowd detection as an operation that preserves the following two properties:

Property 1: Every user in a crowd has at least one other user in the same crowd, such that the similarity between them is at least ϵ. That is, $\forall\ u_i \in C\ \ \exists\ u_k : u_k \in C, u_i \neq u_k\ and\ sim(\vec{u_i}, \vec{u_k}) \geq \epsilon$

Property 2: Every user in a crowd has no other user outside the crowd, such that the similarity between them is at least ϵ. $\forall\ u_i \in C\ \neg\exists\ u_k : u_k \in S \setminus C\ and\ sim(\vec{u_i}, \vec{u_k}) \geq \epsilon$

These two properties ensure that (i) all users within a crowd are more similar to users within the crowd than outside of the crowd; and that (ii) there does not exist any user outside of a crowd who is similar to users within a crowd.

By viewing crowd detection in this way, we can avoid memory-intensive approaches that require maintaining the overall cluster structure (which may be unreasonable for high-volume text); instead, we can formulate the crowd detection problem using nearest-neighbor search as a primitive, as illustrated in Algorithm 1. That is, for every new message posted to the real-time web, we determine the user nearest to the user posting the new message. If the similarity between the user posting the new message and the nearest user is at least ϵ, we add the user to the crowd to which this nearest user belongs, if he is not already in it. If the similarity does not exceed ϵ, we create a new crowd for the given user. K_t is the set of all current crowds at time t. While such an approach may allow long chains of users (where the first user in a crowd is quite distant from the last user), it has the compelling advantage of efficiency.

Algorithm 1 Crowd Discovery

```
for (u, d, t) ∈ I do
    Determine the user nearest to u, u_n, and the crowd
    u_n belongs to C_n.
    if sim(u, u_n) ≥ ε then
        if u is not in crowd C_n then
            Add u to C_n
        end if
    else
        Create a new crowd C with a single user u and
        add it to K_t.
    end if
end for
```

Towards efficiently discovering crowds from the real-time web, we make note of the following three challenges:

- **Efficient User Profile Updating**: Compared to traditional document clustering, in which documents themselves are static and the goal is to find clusters of related documents, crowd discovery seeks to find clusters of similar users in which users are constantly changing (by posting new messages, changing areas of interest, and so on). Hence, the first challenge is to develop an appropriate representation for users that reflects their current interests accurately and can be easily updated every time they generate a new message.

- **Efficient Crowd Assignment**: The second challenge is to determine an efficient method to determine nearest neighbor for crowd assignment. To find nearest neighbors there are several possible methods (including linear search) and several space partitioning data structures (e.g., k-d trees). However, due to the scale of real-time web updates, such methods may incur a high overhead. Hence, we propose a prefix-tree based locality sensitive hashing method that supports $O(1)$ lookup of a user's nearest neighbor, leading to efficient crowd assignment.

- **Identifying Key Dimensions**: Even with a reasonable method for updating user profiles and assigning users to crowds, the real-time web is constantly growing due to the insertion of new phrases, hashtags, and other artifacts of user-contributed content. Figure 3 shows the number of unique tokens encountered over two 10-day Twitter samples (described more fully in Section 4.1), leading to a linear growth in the dimensions for representing users. Hence, the third challenge is to develop a method to identify important dimensions, so that crowd detection can be focused on the most active portions of the real-time web and so resources are not wasted.

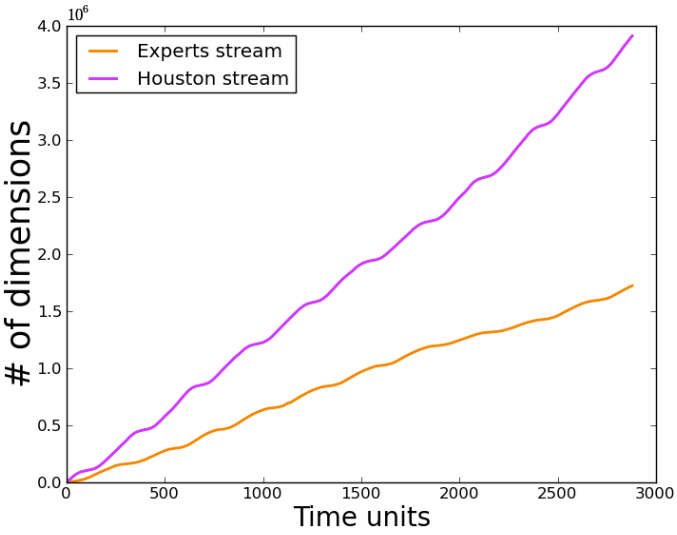


Figure 3: Linear growth of dimensions

In the following, we approach each of these three challenges in turn, before turning to an experimental evaluation.

3.1 Efficient User Profile Updating

In this section, we first develop a vector representation for users that decays temporally, so that users are assigned to

crowds that reflect their current interests and then we show how to efficiently update these user profiles as new messages are generated.

3.1.1 Vector Representation with Fading Memories

Adopting a vector space model for users, let $\vec{u_i}$ be the vector representation for user u_i, where the elements of the vector correspond to tokens parsed from u_i's messages. There are many domain-dependent choices for parsing messages, including language-dependent parsers, entity extraction, stemming, and so forth; for simplicity, we adopt a simple unigram parser that treats all strings separated by whitespace as valid tokens. Since the number of unique tokens corresponding to dimensions are not known in advance, we represent each user profile vector using an infinite co-ordinate space F^{∞} [1]. Under this model, a user u_i at time t is represented as:

$$\vec{u_i}^t = (V_{i1}^t, V_{i2}^t, \ldots, V_{im}^t, \ldots)$$

where the User Vector Dimension (UVD) value V_{im}^t, is the value for u_i in the m^{th} dimension at time t. Let x_{im}^t be the number of times u_i generates m at time t, and $X_{im}^{t_l} = \{x_{im}^1, x_{im}^2 \ldots x_{im}^{t_l}\}$ be the set all occurrences of m generated by u_i until t_l, then $V_{im}^{t_l}$ is defined as:

$$V_{im}^{t_l} = \sum_{x_{im}^t \in X_{im}^{t_l}} \mathcal{F}(x_{im}, t, t_l) = \sum_{x_{im}^t \in X_{im}^{t_l}} x_{im}^t \qquad (1)$$

where $\mathcal{F}$ is a function of x_{im}, t and t_l and is called the UVD function.

In this way, a user is represented as the sum of his entire message history. However, since crowds are designed to reflect users with a similar current interest, such an approach may favor crowds of users who are similar in the long-term. For example, we may identify crowds of students, of entertainers, and of politicians, but miss cross-cutting crowds that are drawn together by their current situation (e.g., emergency-oriented crowds reacting to a local earthquake). An alternate approach is to construct user profile vectors using the latest messages only. While such an approach has the advantage of being memory-less (and so, old messages may be dropped with no penalty), grouping users based only on their most recent messages may result in high crowd fluctuation since crowd assignments may vary with each new message.

To balance these two extremes, we propose to adopt a representation that fades user vectors such that recently used dimensions have higher values and older dimensions have lower values. To decay user vectors, we design another UVD function $\mathcal{D}$, which decreases the score of inactive dimensions and increases the score of active dimensions in user vectors. The function $\mathcal{D}$, re-calculates scores for $x_{im}^{t_o}$ at time t_n, as shown:

$$\mathcal{D}(x_{im}, t_o, t_n) = \lambda_u^{t_n - t_o} x_{im}^{t_o} \qquad (2)$$

where $\lambda_u \in [0, 1]$ is a constant know as the *user dimension score decay rate*. Hence, we can re-write $V_{im}^{t_l}$ as:

$$V_{im}^{t_l} = \sum_{x_{im}^t \in X_{im}^{t_l}} \mathcal{D}(x_{im}, t, t_l) = \sum_{x_{im}^t \in X_{im}^{t_l}} \lambda_u^{t_l - t} x_{im}^t \qquad (3)$$

Note that when $\lambda_u = 1$, the value of $V_{im}^{t_l}$ is same as that calculated using $\mathcal{F}$ as the UVD function.

3.1.2 Efficient Updates

To calculate $V_{im}^{t_l}$ using (3), we have to maintain the entire set $X_{im}^{t_l}$. In the context of the real-time web, this can be inefficient since it requires maintaining $X_{im}^{t_l}$ for all users and all dimensions and since the calculation of $V_{im}^{t_l}$ would be $O(|X_{im}^{t_l}|)$. To solve this problem we prove a proposition that will help us calculate the value of $V_{im}^{t_l}$ efficiently in $O(1)$ time without requiring us to maintain the set $X_{im}^{t_l}$.

PROPOSITION 3.1. *If t_{n-k} is the latest time when u_i generated a message with dimension m until t_n, then the value of the dimension at time t_n, is given by:*

$$V_{im}^{t_n} = \lambda_u^{(t_n - t_{n-k})} V_{im}^{t_{n-k}} + x_{im}^{t_n}$$

where, $V_{im}^{t_{n-k}}$ and $V_{im}^{t_n}$ are the values of dimension m for u_i at time t_{n-k} and t_n respectively.

PROOF. Let $X_{im}^{t_{n-k}}$ be the set all occurrences of dimension m in the messages generated by u_i up to time t_{n-k}. Then, using (3) we get:

$$V_{im}^{t_{n-k}} = \sum_{x_{im}^t \in X_{im}^{t_{n-k}}} \lambda_u^{t_{n-k} - t} x_{im}^t \qquad (4)$$

Using (2), $\forall x_{im}^t \in X_{im}^{t_{n-k}}$ we can write:

$$\mathcal{D}(x_{im}, t, t_n) = \lambda_u^{t_n - t} x_{im}^t = \lambda_u^{(t_n - t_{n-k}) + (t_{n-k} - t)} x_{im}^t \qquad (5)$$

$$\mathcal{D}(x_{im}, t, t_n) = \lambda_u^{(t_n - t_{n-k})} \lambda_u^{(t_{n-k} - t)} x_{im}^t \qquad (6)$$

where t is the time-stamp of every occurrence of m in messages generated by u_i.

Using (3) again, we write,

$$V_{im}^{t_n} = \sum_{x_{im}^t \in X_{im}^{t_n}} \mathcal{D}(x_{im}, t, t_n)$$

$$= \sum_{x_{im}^t \in X_{im}^{t_{n-k}}} \mathcal{D}(x_{im}, t, t_n) + \sum_{n'=n-k+1}^{n} \mathcal{D}(x_{im}, t_{n'}, t_n)$$

Using (4) and (6) we can now write

$$V_{im}^{t_n} = \lambda_u^{(t_n - t_{n-k})} V_{im}^{t_{n-k}} + \sum_{n'=n-k+1}^{n} \mathcal{D}(x_{im}, t_{n'}, t_n)$$

Since u_i did not generate any messages with dimension m after t_{n-k} until t_n, $\forall\ n' \in [n-k+1\ ..\ n-1]$, we have:

$$\mathcal{D}(x_{im}, t_{n'}, t_n) = \lambda_u^{t_n - t_{n'}} x_{im}^{t_{n'}} = \lambda_u^{t_n - t_{n'}} .0 = 0$$

Hence,

$$V_{im}^{t_n} = \lambda_u^{(t_n - t_{n-k})} V_{im}^{t_{n-k}} + \mathcal{D}(x_{im}, t_n, t_n)$$

$$V_{im}^{t_n} = \lambda_u^{(t_n - t_{n-k})} V_{im}^{t_{n-k}} + x_{im}^{t_n}$$

Note that, by definition $x_{im}^{t_n} \neq 0$ if u_i generates a message with m, else it is 0. This proves the proposition. □

In brief, we have described an approach to represent users in high-dimensional vector space that reflects their current interests and we have shown how to update this user profile efficiently upon the arrival of each new user message.

3.2 Efficient Crowd Assignment

Given the user profile developed in the previous section, we now turn to the challenge of assigning users to crowds as outlined in Algorithm 1. This is the core step in crowd detection and is, in essence, a nearest-neighbor problem. To find nearest neighbors there are several possible methods. The simplest algorithm to determine nearest neighbor is through linear $O(n)$ search, which is not efficient due to the large number of users on the real-time web. Alternatively, we can use efficient space-partitioning methods like k-d trees, which have a complexity of $O(\log n)$. Here, we propose a specialized variation of the randomized approach to discover nearest neighbors by using locality sensitive hashing (LSH). In this specialized version, we use an additional prefix tree data structure to support $O(1)$ lookup of a user's nearest neighbor, at a cost of requiring $O(n)$ to look up the user's next nearest neighbor. But by constructing crowd detection as a requiring only user's single nearest neighbor (recall the two properties at the beginning of this section), we can support efficient crowd detection over the real-time web.

3.2.1 Similarity using Locality-Sensitive Hashing

We first describe a function to calculate the similarity between two vectors using LSH and then describe how we can use this similarity function to determine nearest neighbors efficiently using a prefix tree. Since users are represented as vectors, we can use a metric like cosine similarity to determine the nearest neighbor. But, as described in [15], determining nearest neighbors using cosine similarity is inefficient in high dimensions. Hence, we calculate the approximate cosine distance between two vectors using the approach proposed by Charikar [6].

In [6], the author proposed using LSH functions generated using random hyperplanes to calculate approximate cosine distance. Consider a set of vectors in the collection R^m. Let $\vec{r}$ be a m-dimensional random vector, such that each dimension in it is drawn from a 1-dimensional gaussian distribution with mean 0 and variance 1. Then the hashing function $h_{\vec{r}}$ corresponding to $\vec{r}$ is:

$$h_{\vec{r}}(\vec{v}) = \begin{cases} 1 & \text{if } \vec{r}.\vec{v} \geq 0 \\ 0 & \text{Otherwise} \end{cases}$$

Now, if we have a set $R = \{\vec{r_1}, \vec{r_2} \ldots \vec{r_{|R|}}\}$ of such m-dimensional random vectors, then for a vector $\vec{v}$, we can generate its signature $\bar{v} = (h_{\vec{r_1}}(\vec{v}), h_{\vec{r_2}}(\vec{v}), \ldots h_{\vec{r_{|R|}}}(\vec{v}))$. Given two user vectors $\vec{u_i}$ and $\vec{u_j}$, the approximate cosine similarity between them is given as:

$$sim(\vec{u_i}, \vec{u_j}) = cos(\theta(\vec{u_i}, \vec{u_j})) = cos((1 - Pr[\bar{u}_i = \bar{u}_j])\ \pi) \tag{7}$$

So, the closer the signatures, the greater is the cosine similarity, and the more dissimilar the signatures, the lesser is their cosine similarity. This equation measures approximate cosine distance, and accuracy of this approximation can be improved by using a longer signature, i.e, a larger R.

3.2.2 Nearest Neighbor using Prefix Tree

We now describe the procedure to find the nearest user u_n for a user u, from whom we can determine the nearest crowd C_n. We determine u_n using a set of permutation functions $P = \{\pi_1, \pi_2, \ldots, \pi_{|P|}\}$, where each permutation function is of the form:

$$\pi(x) = (ax + b) mod\ p$$

where, p is a prime number and a, b are chosen randomly.

Let $\mathcal{P}$ be a collection of $|P|$ prefix trees, where every prefix tree corresponds to a permutation function $\pi \in P$. Now, to add a vector $\vec{v}$ to $\mathcal{P}$, first its signature $\bar{v}$ is determined, and then the signature is inserted into every prefix tree in $\mathcal{P}$ after permuting it using the corresponding permutation function. So for a given vector, $|P|$ permutations of its signature are stored in $\mathcal{P}$. Every time we observe a new user vector it is added to $\mathcal{P}$. Similarly, every time we modify a user vector, we remove its old signature from all the prefix trees in $\mathcal{P}$ and add the new one.

To determine the crowd nearest to $\vec{u}$ in $\mathcal{P}$, we first calculate its signature $\bar{u}$. Then for every prefix tree in $\mathcal{P}$, we permute this signature using the corresponding permutation function and find the nearest signature in the prefix tree, by iterating through the tree one level at a time starting from the root. After doing this step we end up with $|P|$ signatures, of which the crowd corresponding to the signature with smallest hamming distance is picked as the nearest neighbor of $\vec{u}$. As a result, we see that using a prefix tree in combination with LSH, we can design an efficient algorithm to assign users to crowds.

3.3 Identifying Key Dimensions

The final challenge is a consideration of the purpose of the crowd monitoring application in the selection of the key dimensions for representing user vectors. For example, if the crowd detection system is intended for topic-focused crowd detection (e.g., identify all "earthquake" related crowds, find all crowds related to "politics"), then the user vectors could be weighted toward these key dimensions (e.g., as in a scheme for weighting the dimensions corresponding to the tokens "obama", "debate", "republican" as more important dimensions than non-politics dimensions). Potential solutions include pre-seeding the crowd detection system with expert-labeled keywords or in identifying high value terms by their *inverse document frequency (IDF)*, which weights key terms by their relative rarity across all documents.

In this paper, we propose to select as key dimensions those that reflect the general consensus of the real-time web. That is, we seek to identify tokens that are globally popular at a particular time for biasing the crowd detection toward these tokens. In this way, crowds are defined both by users who have posted similar messages recently (as described in the previous section) and by reflecting topics of great importance to the overall system.

Concretely, our goal is to select from all dimensions the most m significant dimensions. As the real-time web evolves the list of top-m dimensions can then be updated frequently to remove old dimensions and add new ones. Hence, we require a metric to score the dimensions observed so far. To score the dimensions observed in the stream, we use an approach similar to the one used in scoring dimension score for a user vector in Section 3.1.

Let y_d^t be the number of times a dimension d appeared in the stream at time t. Then the score for $y_d^{t_o}$ at time t_n,

Algorithm 2 Crowd Discovery

Create R: Create the set $R = \{\vec{r_1}, \vec{r_1} \dots r_{\vec{|R|}}\}$ of random Gaussian vectors such that $|R| << m$.
Initialize $\mathcal{P}$: Create the set of permutation functions $P = \{\pi_1, \pi_2, \dots, \pi_{|P|}\}$, where each permutation function is defined using a prime number p and values a, b chosen randomly. Initialize $\mathcal{P}$ as a collection of $|P|$ prefix trees and assign a unique permutation function from P to every prefix tree in $\mathcal{P}$.
for $(u, d, t) \in I$ **do**
  Update $\vec{u}$: Update the user vector $\vec{u}$ using (d, t) as described in Section 3.1. Generate new signature for $\vec{u}$ and add or replace it in $\mathcal{P}$.
  Generate $\bar{u}$: Generate the $|R|$-bit signature for $\vec{u}$, $\bar{u}$ using R.
  Step 1: Determine u_n **and** C_n: Get the user nearest to u, u_n and the crowd u_n belongs to $C_n \in K_t$.
  if $sim(\vec{u}, \vec{u_n}) \geq \epsilon$ **then**
    if u is not in crowd C_n **then**
      Step 2: Add u **to** C_n: Add u to crowd C_n.
    end if
  else
    Step 3: Create C: Create a new crowd C with a single user u and add it to K_t.
  end if
end for

$t_n \geq t_o$, is given by a function $\mathcal{E}$, defined as:

$$\mathcal{E}(y_d, t_o, t_n) = \lambda_d^{t_n - t_o} y_d^{t_o} \tag{8}$$

where $\lambda_d \in [0, 1]$ is a constant know as the *dimension score decay rate*.

Since a dimension can be observed several times in a stream, the score for a dimension d at time t, W_d^t, is calculated as shown in Proposition 3.2

PROPOSITION 3.2. *If t_{n-k} is the latest time when dimension d was observed on the stream until t_n, then the dimension score for the dimension at time t_n, is given by:*

$$W_d^{t_n} = \lambda_d^{(t_n - t_{n-k})} W_d^{t_{n-k}} + y_d^{t_n}$$

where, $W_d^{t_{n-k}}$ and $W_d^{t_n}$ are the dimension scores at time t_{n-k} and t_n respectively.

PROOF. The proof for this is similar to the proof of Proposition 3.1. □

Hence, we can identify dimensions that reflect the consensus of the current activity of the real-time web, so that crowd detection can be focused on the most active portions of the real-time web and so resources are not wasted.

3.4 Putting it All Together

Taken together, the high-level crowd discovery algorithm described in Algorithm 1 and the three methods developed – efficient user profile updating, efficient crowd assignment, and identifying key dimensions – give us the crowd discovery algorithm in Algorithm 2.

4. EXPERIMENTS

In this section, we report a series of experiments to study crowd discovery. We evaluate the running time performance of the proposed crowd discovery algorithm with other algorithms for crowd discovery. We define metrics to measure quality of crowds discovered and using these metrics we evaluate the quality of crowds discovered by several crowd discovery algorithms. We study the factors impacting the performance of the proposed algorithm, and finally we analyze the properties of crowds discovered over two Twitter datasets.

4.1 Dataset

To simulate a Twitter stream, we selected a set of Twitter users and crawled their tweets using Twitter API. The users in this set are labeled using 4 classes – technology, entertainment, politics and sports. To collect this labeled dataset we used the snowball sampling approach. This approach is as follows:

- First, for every class we selected a set of 5 Twitter users, called seed users, that belong to this class and 5 key words that describe the class. For example, for the class sports, a seed user was "espn" and a keyword was "sports".
- We then used Twitter API to select all Twitter lists that contain a seed user, such that the list's name contains a class specific key word. For example if "sports_news" and "news" are Twitter lists that contain "espn", then we select "sports_news" but not "news", since the former has the keyword "sports" in its name.
- We then extracted a set of new users from the lists selected in previous step and crawled their lists like before.

Following these steps resulted in a "snowball" or chain of crawling actions, which we stopped once we observed sufficient users. At the end of this crawl, we were left with a set of users, lists they belong to, with lists labeled with the class they belong to. Using this information, for each domain we selected around 1,200 top users and used their tweets to simulate a labelled Twitter stream, resulting in about 1.6 million tweets for 30 days. A similar approach for sampling class specific Twitter data is described in [27]. In addition to this dataset (which we shall call the Experts dataset), we collected a location-based dataset of users tweeting from the Houston region who were selected through random sampling. A 30-day sampling of this stream had about about 15 million tweets from about 107 thousand users. We use the Experts dataset for all of our experiments, except for the experiments in Section 4.7.

4.2 Setup

We compare the crowd discovery algorithm (*CDA*) proposed in this paper with four alternatives: k-means clustering (*k-means*), a Map-Reduce implementation of k-means clustering (*MR k-means*), a deterministic batched version of the CDA approach (*Iterative-CDA*) – in which we iterate through all the pairs of user vectors to find the best crowds possible, and a Map-Reduce implementation of Iterative-CDA (*MR-CDA*).

For user vector processing, we set the following parameters: number of dimensions $m = 199,999$, user dimension score decay rate $\lambda_u = 0.75$ and dimension score decay rate $\lambda_d = 0.75$. For efficeint crowd assignment, we set signature length $|R| = 23$, number of permutation functions $|P| = 13$ and $\epsilon = 0.005$.

In initial experiments, we varied the choice of k for k-means, finding in many cases that k-means identified many singleton crowds. For the experiments reported here, we set the number of clusters as $k = 0.95\times$number of items to cluster.

4.3 Running Time Analysis

To evaluate the running time performance of the proposed approach, we perform two experiments: (i) we use tweet sets of varying sizes as input to all the algorithms and determine the time taken by them to discover crowds; (ii) we measure the tweet processing rate of the algorithms. For these experiments we use a 30 day sample of Experts stream.

Running Time with Clustering Algorithms: The plot in Figure 4(a), shows the running times for the two k-means clustering algorithms and CDA to discover crowds on data collection of varying sizes. The running times graph is a log-log graph, hence there are orders of magnitude difference between the running times of the algorithms. We see that the time required to discover crowds using the proposed algorithm is significantly lesser than that required by the clustering algorithms. As the size of the message collection increases, both the clustering algorithms become slower. This behavior is expected in case of iterative k-means, because of the extra iterations required by the algorithm, but was not expected in the Map-Reduce version. Generally, the Map-Reduced running time increases at a much slower rate, but is still lesser than that of the iterative version. We believe the worsening performance is because of the large value of k. Larger k results is passing of greater number of centroids to a map job which slows down the algorithm. Hence, either of these algorithms are not efficient to discover crowds.

Running Time with CDA Algorithms: We now run similar experiments with the other crowd discovery algorithms. As in the case of the clustering algorithms, we see that CDA, in Figure 4(b), performs much better than the batched CDA algorithms. The Iterative-CDA performs the worst while the MR-CDA performs better after about 10^4 messages. The bad performance of MR-CDA on initial message sets can be attributed the time spent by the MR cluster in setting up the job and passing messages between various workers.

Message Processing Rate with CDA Algorithms: To compare the rate at which the algorithms process messages as they arrive, we note the number of messages that the algorithms have processed at equally spaced time intervals. This comparison is shown in Figure 4(c). As expected, we observe that the number of messages processed by the proposed algorithm is more than that for the other CDA algorithms. This result supports the result we observed with running time Figure 4(b). Similar results were observed for k-means clustering as well but are omitted due to the space constraint.

4.4 Crowds Quality Analysis

We now evaluate the quality of crowds discovered using the proposed crowd discovery approach. We know the class to which users in our Twitter stream belong, hence, to evaluate crowd quality we can compare the crowds discovered to this "ground truth". While we do not expect all users belonging to a particular class (e.g., "sports") to form a single large crowd, we do expect that crowds that form will tend to be composed of users belonging to these classes. We use the same 30 day sample of the stream that we used in Section 4.3. Like before, the experiments are run with the same value for parameter m. We next describe evaluation metrics that we use to measure quality of crowds and then present performance of CDA against k-means clustering algorithms and deterministic CDAs.

Quality metrics: Consider the set of crowds $K = \{C_1, C_2, \ldots, C_n\}$ for users in set U and a set of classes $\Omega = \{\omega_1, \omega_2, \ldots, \omega_w\}$ to which users in U belong. To measure the quality of crowds generated using crowd discovery algorithms we use the following metrics.

Purity: To compute purity, we assign crowd to the domain which is most frequent in it, and then the accuracy of this assignment is measured by calculating the ratio of correctly assigned users.

$$purity(K, \Omega) = \frac{1}{|U|} \sum_n \max_w |C_i \cap \omega_j|$$

NMI: Purity gives a good understanding of quality. But, it is susceptible because high purity can be achieved when there are large number of crowds, which we expect in crowd discovery problem. Hence, to deal with this issue, we use a secondary information theory based quality metric called Normalized Mutual Information (NMI). It is defined as:

$$NMI(K, \Omega) = \frac{I(K, \Omega)}{[H(K), H(\Omega)]/2}$$

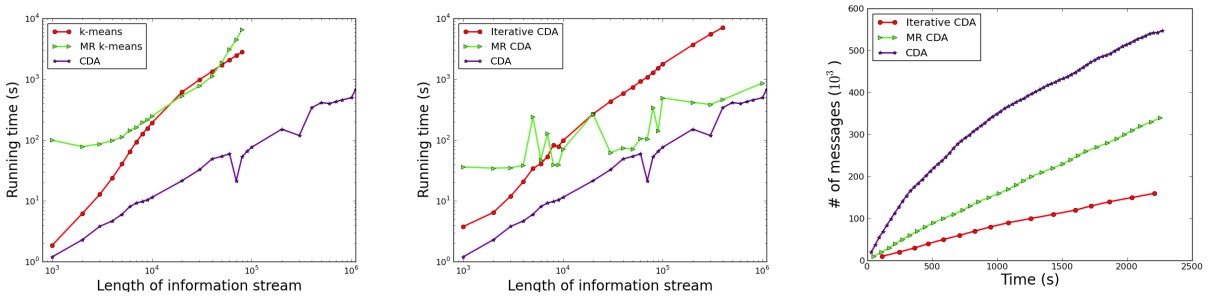
$$I(K, \Omega) = \sum_n \sum_w \frac{|C_n \cap \omega_w|}{|U|} \log \frac{|U||C_n \cap \omega_w|}{|C_n||\omega_w|}$$

$$H(K) = -\sum_n \frac{|C_n|}{|U|} \log \frac{|C_n|}{|U|}$$

where, $I(K, \Omega)$ is mutual information and H entropy.

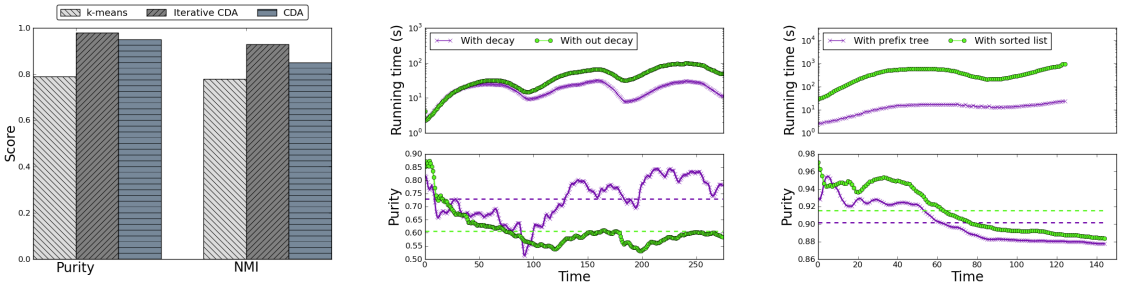
Comparison with Clustering Algorithms: The comparison between quality of crowds discovered using the Iterative k-means and that discovered using CDA is shown Figure 5(a). We see that despite the significant improvements in running time, the crowds discovered by the CDA are still of high quality. We also notice, for all the metrics, the quality of crowds generated using CDA is better than the quality of crowds generated using a clustering algorithm. The relatively poor performance of the clustering algorithm can be attributed to the difficulties in estimating the number of clusters k.

Comparison with CDA Algorithms: The comparison between quality of crowds discovered using the Iterative-CDA and that discovered using CDA is shown in Figure 5(a). We see that crowds discovered by Iterative-CDA are always better than that discovered using CDA. The lower values for



(a) Comparison with k-means (b) Comparison with CDA (c) Message processing rate comparison of CDAs

Figure 4: Comparing the efficiency of crowd discovery (CDA) versus alternatives



(a) Quality of crowd discovery (b) User vector representation (c) Crowd assig. with prefix trees.

Figure 5: Crowd algorithm analysis

these metrics is expected in case of CDA, as it is a randomized and an approximate algorithm whereas Iterative-CDA is an deterministic algorithm.

4.5 Impact of User Vector Representation

In Section 3.1, we described the method to exponentially decay user vectors to help us discover temporally relevant crowds. We evaluate the effectiveness of this approach by analyzing the performance of CDA when the user vectors are exponentially decayed and when they are not. To evaluate the performance of the algorithm without decay, we set $\lambda_u = 1.0$. The difference in quality of the crowds generated by the algorithm using these two approaches is shown in Figure 5(b).

The top plot of Figure 5(b), shows the running time of the algorithms for this experiment. We observe that, thought the running times for the algorithms is almost the same initially, the difference between them increases with time. This is because, as time increases, the algorithm that decays user vector and uses techniques to score dimensions, has the ability to remove dimensions when they become stale. This feature is not possible when the algorithm is run without decay.

As shown in the bottom plot of Figure 5(b), the quality of crowds discovered using exponential decay is much better than the crowds discovered without decay. When user vectors are not decayed, old dimensions are not removed from it, resulting in crowds being discovered which contain users from different domains. This results in lower quality crowds.

4.6 Impact of Prefix Trees

We next analyze the impact of using prefix trees on efficient crowd assignment. An alternative approach described in [23] suggests representing $\mathcal{P}$ as a collection of sorted lists of signatures rather than prefix trees. Such a structure is robust in the sense that signatures are sorted and hence nearest neighbor can be found faster than linear search, but has the downside that determining the nearest neighbor and adding a new vector takes $O(\log n)$ time, considering $|P|$ is constant. To characterize the impact of the prefix-tree based locality-sensitive hashing approach, we run CDA both with prefix trees and with sorted lists. The results are shown in Figure 5(c).

The top plot shows the running time and the bottom plot shows the quality of crowds discovered. We see that by using prefix trees, we can discover crowds at speeds several times the speed using sorted lists. As mentioned before, the improved speed efficiency is because of the constant time required to retrieve crowds in case of prefix tree instead of $O(\log n)$ as in case of sorted lists.

The quality of crowds generated varies initially when the number of crowds in the prefix tree is small because of randomization involved in determining the nearest neighbor. This variance is overcome as the number of crowds in the prefix tree increases and the mean quality of crowds discovered remains almost the same. After sometime, once we have observed sufficient crowds, we observe that the crowds quality is almost same while using both prefix tree and sorted lists.

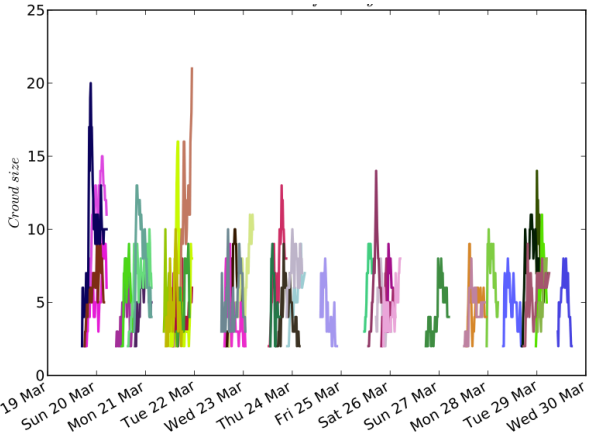


Figure 6: Example of crowds related to Libya

4.7 Comparing Crowds

Finally, we explore the impact of the kind of users on crowd formation. We compare the crowd size distribution, followed by the lifespan distribution of the crowds. Then we plot these two properties towards understanding crowding behaviors in these two datasets.

The distribution of crowd sizes is shown in Figure 7(a). We see that the Houston dataset tends to have larger crowds in comparison to the Experts dataset. These larger crowds may be attributed to the fact that the Houston dataset has relatively more users in comparison to the Experts dataset, and hence more users talking about a particular event resulting in the formation of larger crowds. To understand these dynamics better, we show the lifespan of these crowds in Figure 7(b). The lifespan distribution shows that expert crowds, despite being smaller, are mostly longer lasting than the larger crowds discovered in Houston. Based on further analysis, we find that the experts stream is more *sticky* – that is, crowds in the experts stream added new users over time and decayed more slowly.

We attribute this finding to the crowd formation properties of the Experts dataset, whereby crowds are initiated by users who are popular within a particular domain and hence tend to tweet similar things more often. This shared interests among users forms crowds that discuss chains of events resulting in longer lifespans. While users in the Houston dataset, which is made up of users with relatively varied interests, form crowds that last only as long as the event they are discussing is popular. Continuing this avenue of investigation, we plot crowd size versus life span in Figure 7(c). If the crowds in the Experts dataset are really sticky, as we expect, this should be observed across all the crowds of different sizes, i.e, only larger crowds should not have contributed in making the life span distribution in Figure 7(b) appear the way it does. We observe that irrespective of crowd size, expert crowds always seem to have a higher lifespan than Houston crowds. This clearly shows the way users in expert crowds are tweeting and the content of their tweets is making them stick together longer than Houston crowds. In addition to this observation, we also see that the stickiness of the crowds increases with crowd size. This is observed both for the experts and Houston crowds.

We also find that events that last for a long time have more number of crowds that are spread across the event's duration. An example of such a long term event is the revolution in Libya, and crowds related to this appear throughout the experiment duration following a daily pattern based on users activity, as shown in Figure 6.

5. CONCLUSION

We have seen how the proposed content-driven crowd discovery algorithm can efficiently identify newly-formed communities of users from the real-time web. The approach leverages optimizations to locality-sensitive hashing via prefix trees, incorporates efficient user profile updating, and identifies key dimensions for supporting crowd detection. In our continuing work we are interested to understand the dynamics if crowds formation and their evolution. We are also interested in analyzing the impact geography and culture has on crowd formations.

6. ACKNOWLEDGMENTS

This work was supported in part by NSF grant IIS-1149383, DARPA grant N66001-10-1-4044 and a Google Research Award. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsors.

7. REFERENCES

[1] Infinite coordinate space. wikipedia: Examples of vector spaces, 2011.

[2] C. C. Aggarwal. A framework for clustering massive text and categorical data streams. In *In: Proc. SIAM conference on Data Mining*, pages 477–481, 2006.

[3] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '2003, pages 81–92. VLDB Endowment, 2003.

[4] J. Allan. *Introduction to topic detection and tracking*, pages 1–16. Kluwer Academic Publishers, Norwell, MA, USA, 2002.

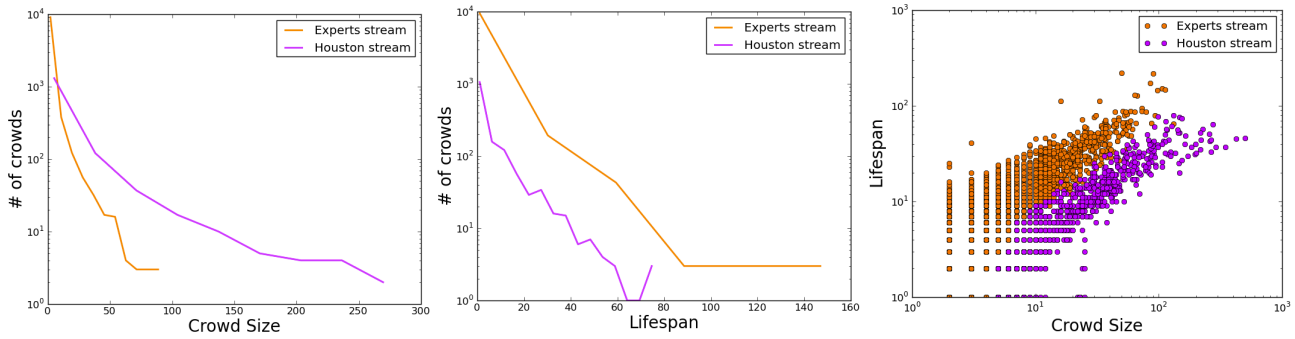
[5] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[6] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, STOC '02, pages 380–388, New York, NY, USA, 2002. ACM.

[7] S. Chen, H. Wang, S. Zhou, and P. S. Yu. Stop chasing trends: Discovering high order models in evolving data. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 923–932, Washington, DC, USA, 2008. IEEE Computer Society.

[8] H. Choi and H. Varian. Predicting the present with google trends. Technical report, Google, 2009.

[9] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/gather: a cluster-based approach to browsing large document collections. In *Proceedings*



(a) Crowd size distribution (b) Lifespan distribution (c) Crowd size Vs Lifespan

Figure 7: Comparing crowds discovered across the two datasets

of the 15th ACM SIGIR, SIGIR '92, pages 318–329, New York, NY, USA, 1992. ACM.

[10] I. Dhillon, Y. Guan, and B. Kulis. A fast kernel-based multilevel algorithm for graph clustering. In *KDD '05*, pages 629–634, New York, NY, USA, 2005. ACM.

[11] W. Fan, Y. Koyanagi, K. Asakura, and T. Watanabe. Clustering over evolving data streams based on online recent-biased approximation. chapter Knowledge Acquisition: Approaches, Algorithms and Applications, pages 12–26. Springer-Verlag, Berlin, Heidelberg, 2009.

[12] M. Franz, T. Ward, J. S. McCarley, and W.-J. Zhu. Unsupervised and supervised clustering for topic tracking. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '01, pages 310–317, New York, NY, USA, 2001. ACM.

[13] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[14] L. Gong, J. Zeng, and S. Zhang. Text stream clustering algorithm based on adaptive feature selection. *Expert Syst. Appl.*, 38:1393–1399, March 2011.

[15] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[16] K. Y. Kamath and J. Caverlee. Transient crowd discovery on the real-time social web. In *Proceedings of the fourth ACM international conference on Web search and data mining*, WSDM '11, pages 585–594, New York, NY, USA, 2011. ACM.

[17] C. X. Lin, B. Zhao, Q. Mei, and J. Han. PET: A statistical model for popular events tracking in social communities. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 929–938, New York, NY, USA, 2010. ACM.

[18] Y.-B. Liu, J.-R. Cai, J. Yin, and A. W.-C. Fu. Clustering text data streams. *Journal of Computer Science and Technology*, 23(1):112–128, 2008.

[19] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[20] F. Moerchen, K. Brinker, and C. Neubauer. Any-time clustering of high frequency news streams. 2007.

[21] M. E. J. Newman. Fast algorithm for detecting community structure in networks, September 2003.

[22] S. Petrović, M. Osborne, and V. Lavrenko. Streaming first story detection with application to twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 181–189, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[23] D. Ravichandran, P. Pantel, and E. Hovy. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 622–629, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.

[24] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *WWW '10*, pages 851–860, New York, NY, USA, 2010. ACM.

[25] L. Tang and H. Liu. *Community Detection and Mining in Social Media*. Morgan and Claypool, 2010.

[26] X. Wang, L. Tang, H. Gao, and H. Liu. Discovering overlapping groups in social media. In *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM)*, pages 569–578, Washington, DC, USA, 2010. IEEE Computer Society.

[27] S. Wu, J. M. Hofman, D. J. Watts, and W. A. Mason. Who says what to whom on twitter. *WWW 2011*, 2011.

[28] Y. Zhu and D. Shasha. Statstream: statistical monitoring of thousands of data streams in real time. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 358–369. VLDB Endowment, 2002.