Real-time Event Detection in Massive Streams

Saša Petrović

Doctor of Philosophy

Institute for Language, Cognition and Computation

School of Informatics

University of Edinburgh

2012

Abstract

New event detection, also known as first story detection (FSD), has become very popular in recent years. The task consists of finding previously unseen events from a stream of documents. Despite the apparent simplicity, FSD is very challenging and has applications anywhere where timely access to fresh information is crucial: from journalism to stock market trading, homeland security, or emergency response. With the rise of user generated content and citizen journalism we have entered an era of big and noisy data, yet traditional approaches for solving FSD are not designed to deal with this new type of data.

The amount of information that is being generated today exceeds by many orders of magnitude previously available datasets, making traditional approaches obsolete for modern event detection. In this thesis, we propose a modern approach to event detection that scales to unbounded streams of text, without sacrificing accuracy. This is a crucial property that enables us to detect events from large streams like Twitter, which none of the previous approaches were able to do.

One of the major problems in detecting new events is vocabulary mismatch, also known as lexical variation. This problem is characterized by different authors using different words to describe the same event, and it is inherent to human language. We show how to mitigate this problem in FSD by using paraphrases. Our approach that uses paraphrases achieves state-of-the-art results on the FSD task, while still maintaining efficiency and being able to process unbounded streams.

Another important property of user generated content is the high level of noise, and Twitter is no exception. This is another problem that traditional approaches were not designed to deal with, and here we investigate different methods of reducing the amount of noise. We show that by using information from Wikipedia, it is possible to significantly reduce the amount of spurious events detected in Twitter, while maintaining a very small latency in detection.

A question is often raised as to whether Twitter is at all useful, especially if one has access to a high-quality stream such as the newswire, or if it should be considered as sort of a poor man's newswire. In our comparison of these two streams we find that Twitter contains events not present in the newswire, and that it also breaks some events sooner, showing that it is useful for event detection, even in the presence of newswire.

Acknowledgements

First of all, I would like to thank my supervisor Miles Osborne for his great guidance and encouragement throughout my PhD. Without him, this thesis would just not be possible. A big thanks also goes to my second supervisor Victor Lavrenko for the many helpful discussions and the long hours we spent getting decade-old systems working again.

I would also like to thank the EPSRC-funded Cross project (EP/J020664/1) for providing the financial support for the work done in Chapter 7.

A number of people in ILCC (and formerly, ICCS) have made my time here very enjoyable, despite the weather. Special thanks to Moreno Coco, Desmond Elliott, Stella Frank, Diego Frassinelli, Tom Kwiatkowski, Mike Lewis, and Dave Matthews for all the beer, table tennis, and climbing. Most of all, thank you for your friendship.

Marina, thank you for all the support and understanding, I am lucky to have you. Thank you for being there.

Last, but not least, I would like to thank my mother. I cannot hope to list all the things for which I am grateful to you. Thank you for everything.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Saša Petrović*)

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

News only has value while it is still news. Finding out about a major earthquake two days after it happened, while still informative, is probably not very useful. While there is a continuum of allowed latency depending on the type of news, it should be fairly clear that the sooner we hear about the news, the better. Central to the notion of news is the concept of an *event* – something that happens at a specific time and place. It is not surprising that there is a lot of interest in reporting new events as soon as they happen.

Many new events happen in the world every day. Depending on the event's importance and scale, it may receive anywhere from none to thousands of follow-up stories reported in the newswire. Clearly, it would be beneficial to automatically organize this huge stream of documents produced by news agencies every day and identify only the new events, keeping in mind that the latency in reporting these events should be minimal. This is known as *new event detection* or *first story detection* (FSD) and previous studies have shown that it is a very challenging task (Allan et al., 2000a). In the last few years, this problem has only become harder due to the fact that news is no longer being reported only through newswire. We live in an age where user generated content and citizen journalism have become so mainstream that news often breaks first in other media and is only later picked up by newswire. Nowadays, these alternative sources of news usually come in the form of social media sites, e.g., blogs, forums, Twitter, pinboards, imageboards, etc. Of these, we focus primarily on Twitter because of its real-time nature – people normally tweet about events as they are happening, sometimes reporting them before mainstream media. More prominent examples of this include leaking of Osama bin Laden's death[1] or notifying of Michael Jackson's

[1] http://www.huffingtonpost.com/2011/05/02/osama-bin-laden-death-twitter-leak_n_856121.html.

death[2] an hour before any other news reported it. There are also cases where Twitter was the primary media to carry news – e.g., events surrounding the Arab Spring were primarily reported on Twitter and later picked up by newswire.[3]

Popularity of event detection in social media is probably best illustrated by examples. One example is the company *BreakingNews*, best known for its Twitter account *@BreakingNews*.[4] This team of dozen journalists and technologists "scour the planet for breaking news" in order to provide a real-time feed of breaking news. The company relies on almost 300 news organizations using specific hashtags to indicate breaking stories, as well as on the massive manual effort of their employees who look for new stories and remove duplicate ones. Another example is the startup *Storyful*,[5] whose team of approximately 20 journalists constantly read new tweets and report when they spot a new event. These examples show that event detection, especially event detection in social media, is a clear user-facing task that has generated a lot of interest (e.g., @BreakingNews has over 4.5 million followers on Twitter). Having a system that can automatically perform this task in real time would be very beneficial and would significantly reduce the human effort involved.

In this thesis, we present a modern approach to event detection that scales to massive volumes of data without sacrificing accuracy. Furthermore, we want to be able to exploit information coming from a variety of sources (e.g., newswire, Twitter, Wikipedia). Thus, the main challenges that we face in this work, in addition to those present in the original work on FSD, are scale, large amounts of irrelevant information, and combining Twitter with other streams.

The current scale at which documents are produced in social media streams exceeds by many orders of magnitude the scale at which documents are produced in newswire. More importantly, we cannot think of data as a finite set of documents any more. Rather, data today is an endless stream of documents, and modern algorithms should be designed for working in such an environment. Therefore, making sure that an approach can handle unlimited streams of data becomes a priority. While existing work in first story detection has largely ignored scalability and concentrated on improving accuracy, we will need to make sure that our approach scales well to high-volume unbounded streams both in terms of space and time.

[2] http://www.techradar.com/news/world-of-tech/internet/10-news-stories-that-broke-on-twitter-first-719532.

[3] http://www.huffingtonpost.com/raymond-schillinger/arab-spring-social-media_b_970165.html.

[4] http://twitter.com/BreakingNews.

[5] http://storyful.com/.

Traditional FSD approaches (Allan et al., 2000b; Yang et al., 1998) assume documents come from a professionally curated stream like newswire, where all the documents are about newsworthy events. When dealing with social media streams, this is no longer the case, and most of the documents in the stream will be spam or trivial chatter that should never be shown to the user. Therefore, a modern event detection approach should report as many new events, while at the same time trying to minimize the amount of irrelevant information that is shown to the user. Because this irrelevant information is something that the user is not interested in and distracts her from the actual information (signal), we will use the term *noise* to refer to it. This term is often used to denote unwanted, irrelevant information,[6] and we found it fitting to use it here to denote all the irrelevant information about teenage pop-stars and people's breakfasts.

News today is reported in different streams and may not even manifest as explicit posts (e.g., page view spikes in Wikipedia). Modern approaches should take advantage of this fact to reduce noise and latency in reporting events. In the Topic Detection and Tracking (TDT) project, all the different streams (newswire, radio, broadcast) have been merged into a single stream and processed as such. This approach made sense because all these streams discuss mostly news and are created by professionals for public consumption. On the other hand, simply combining Twitter and, say, newswire, is more difficult as Twitter is very different: content is generated by users, its main purpose is not reporting news, but connecting with friends, and there is a constraint on the length of documents. Therefore, it should be clear that we need a better way of combining information that comes from multiple different streams.

Based on the above discussion, we define four key goals that we want our event detection approach to satisfy:

1. Generality. This refers to the types of events that we can detect. As we shall see in Chapter 2, most existing event detection systems can only detect a certain type of events (e.g., earthquakes or celebrity-related events), but a modern system should capture all event types.

2. Scalability. As we just discussed, a modern approach to event detection should be scalable to unbounded, high-volume streams. Scalability here means that the system should not get slower over time, and that it can process all the incoming data as soon as it arrives.

[6] In the context of Twitter, the term has been used extensively in the literature, see Sankaranarayanan et al. (2009), O'Connor et al. (2010), or Genc et al. (2011) as an example.

3. Real-time processing. This property is orthogonal to scalability. By real-time processing we mean that the approach should be able to detect an event with minimal lag, i.e., as soon as possible after the event happens. It is possible for an approach to be scalable, but not real-time, e.g., if it can process all of the data but requires a day of data to accumulate before it can reliably detect events. We will see examples of such approaches in Chapter 2.

4. No supervision. Finally, we require our approach to be unsupervised in the sense that no labeled training data or interaction from the user is needed in order for the approach to work. While this property is not crucial for a modern approach, we restrict ourselves to unsupervised approaches for three main reasons: i) producing even small amounts of labeled training data for event detection is a tedious and expensive task, ii) any progress that is made with unsupervised approaches can likely be used to improve supervised approaches, while the reverse is not true, and iii) using supervised techniques for event detection is an entire thesis in itself as one has to deal with issues like the choice of algorithm, feature engineering, efficient training and re-training, etc.

We first deal with the problem of *scale* by developing a new method for scalable FSD. In its core, this method uses locality sensitive hashing (LSH), an approximate nearest neighbor technique, to dramatically reduce our search space. However, because of the specifics of the first story detection task, using LSH alone performs poorly. This is why we take advantage of the probabilistic bounds provided by LSH to determine if the provided result is good enough. When we deem that this is not the case, we perform a limited secondary search over the most recent documents in order to find a better solution. Using LSH together with the secondary search improves the results by 19% and reduces the variance in the results by an order of magnitude. Experiments on TDT5 (largest available collection of documents for FSD) show that our approach is over an order of magnitude faster than a state-of-the-art FSD system and scales to unbounded streams, while achieving the same accuracy.

One of the problems that plagues FSD, as well as other IR and NLP tasks, is the problem of *lexical variation*, sometimes also known as the vocabulary mismatch problem. Because different authors use different words to describe the same events, this often leads to non-novel stories appearing to be new because of the different vocabulary that was used. We use paraphrases as a way to alleviate this problem, making sure that we integrate them into our efficient approach so that we are still able to handle un-

bounded streams. Our experiments on TDT5 data show that this approach outperforms the previously best reported result on the FSD task by 19%, while still being efficient.

We next deal with the problem of *noise* in Twitter. Because most of the related work on event detection in Twitter relies on burstiness to detect true events and filter out noise, we take this approach as a starting point. As part of this work, we conduct the first experiments that show the tradeoff between latency in reporting the events and their quality. We find that a low threshold on burstiness can significantly improve results, but that aggressive thresholds, which are frequently used in the literature, hurt recall, while not significantly improving precision, except for high latencies.

Finally, we look at combining Twitter with other streams, namely Wikipedia and newswire. We use Wikipedia as an additional source of information to further reduce the amount of irrelevant events detected by our approach. While Wikipedia was used before for the purpose of event detection, we are the first to take time into account, and we show that doing so yields significantly better results than ignoring the time properties. Comparing Twitter with newswire, we show that most events reported in the newswire are also reported in Twitter, while the reverse is not true. We then conduct the first experiment to quantify to which extent Twitter breaks news before the newswire, and show that this happens in about 0.4% of cases. Our experiments with Twitter and newswire show that combining the two streams can lead to more events being detected, and with a lower latency than with either stream alone.

To summarize, the main contributions of this thesis are:

- **Creation of the first FSD corpus for Twitter.**
- **A novel algorithm for doing scalable first story detection.**
- **Integrating paraphrases into our efficient FSD system, which helps reduce the effect of vocabulary mismatch in this task.**
- **First evaluation of the tradeoff between latency and accuracy in FSD.**
- **A novel method for combining Twitter and Wikipedia that can significantly improve the quality of detected events.**
- **First work on using an external collection (newswire) to expand microblogs for the purpose of detection or first story detection.**

- **Investigation of the differences between Twitter and newswire, showing that Twitter is indeed a very useful resource for event detection, even if one has access to newswire.**

1.1 Organization of the Thesis

In this section we give a brief overview of the rest of this thesis.

Chapter 2 gives background information that is instrumental to understanding the rest of the thesis. We first describe the task of first story detection and the related tasks from the Topic Detection and Tracking project. We then provide background on the area of approximate nearest neighbors methods, which is the backbone of our scalable approach to event detection. Finally, we review the related work on event detection in social media, focusing on event detection in Twitter. We show that none of the current approaches fully satisfy the conditions for an event detection system laid out in this thesis.

Chapter 3 describes the corpus of tweets that we have created for the purposes of evaluating event detection systems. While there is an increasing amount of interest in detecting events from Twitter, the progress is hampered by the lack of a corpus that could be used to compare systems. We build the first such corpus, which enables us to perform evaluation of first story detection on Twitter, noting also that our corpus should be helpful to other researchers working on this and similar tasks. This work was previously reported in Petrović et al. (2012).

Chapter 4 presents our approach to scaling event detection to unbounded streams. Our approach based on locality sensitive hashing requires $O(1)$ processing time per document, and uses a bounded amount of space. In our experiments, we show that our approach achieves state-of-the-art results on the FSD task, while being over an order of magnitude faster than a state-of-the-art system (UMass). We explore different aspects of our approach in order to paint a complete picture of how to set the different parameters and what their effect is on performance. This chapter extends work previously published in Petrović et al. (2010).

Chapter 5 shows how paraphrasing can be used for query expansion yet retain efficiency. We use paraphrase information in order to improve the state-of-the-art results in the FSD task by reducing the effect of vocabulary mismatch. We show

that our approach which uses paraphrases improves the state-of-the-art results in FSD on the standard TDT5 corpus by 18.6% over the previously best result, while still remaining scalable. Together with parts of Chapter 3, this work was published in Petrović et al. (2012).

Chapter 6 deals with the noisy nature of Twitter. We first show that novelty alone, which is what the traditional TDT approaches used, performs poorly for event detection in Twitter. On the other hand, more recent work on event detection in social media uses a simple strategy based on burstiness of items in some time window. We combine these two approaches and show that this leads to significantly improved precision of detected events. We explore the tradeoff between the length of this time window (latency) and the accuracy of detected events and find that using a latency higher than ten minutes does not significantly improve results, while resulting in a system that can no longer be considered real-time. Parts of this chapter were published in Petrović et al. (2010).

Chapter 7 looks at the problem of combining evidence from other streams for the purpose of improving different aspects of our system. By combining Twitter and Wikipedia, we show that it is possible to achieve significantly better precision than the baseline approach based on burstiness, and with a much lower latency. We show that newswire articles can be used to expand tweets, producing a better clustering of tweets. We combine Twitter and newswire and show that Twitter is a useful resource for event detection, even if one has access to newswire. This is because Twitter contains information not present in the newswire, and reports some events with lower latency. Preliminary results from this chapter were published in Osborne et al. (2012).

Chapter 8 concludes the thesis by summarizing the contributions, and giving possible avenues for future work.

Chapter 2

Background

The work in this thesis draws upon several fields from computer science, such as information retrieval, natural language processing, and randomized algorithms. In this chapter we review the background from these fields necessary for understanding the rest of the thesis. Given the recent renewed interest in event detection, only in the context of social media, we also present a survey of the related work in this area and explain how our work sits with respect to these approaches.

2.1 Topic Detection and Tracking

Topic Detection and Tracking (TDT) (Allan, 2002) was a project started in 1998 with the overall goal of improving technologies related to event-based information organization tasks. The project was divided into five tasks: segmentation, tracking, detection (also called on-line clustering), first story detection (also called new event detection), and linking. In this thesis we focus mostly on the FSD task, but we also look at the implication of our work on some of the other tasks. In this section we give an overview of all five tasks and the current state-of-the-art approaches in first story detection that this work builds upon.

Because we use the terms *story*, *event*, and *topic* throughout the thesis, let us start by explaining these concepts as they were defined in the TDT project. Fiscus and Doddington (2002) state that a story is a *topically cohesive segment of news that includes two or more declarative independent clauses about a single event*. Furthermore, they define a topic as *a seminal event or activity, along with all directly related events and activities*. Finally, event is defined as *something that happens at a specific time and place along with all necessary preconditions and unavoidable consequences*. We will

see in Chapter 3 that some of these definitions will have to be adapted to account for the differences between tweets, that we use in this thesis, and newswire articles, which were used in TDT.

2.1.1 Subtasks

2.1.1.1 Segmentation

The input to a TDT system is originally meant to come in the form of automatically transcribed text. Because this text is not yet segmented into individual stories, the first task of the system is to segment this stream of text. More formally, the goal of the segmentation task is to *segment the source stream into its constituent stories, for all audio sources* (Fiscus and Doddington, 2002). In the context of TDT, a story is "a topically cohesive segment of news that includes two or more declarative independent clauses about a single event." Note that this definition of a story excludes commercials, which means that systems are not evaluated on how well they detect boundaries between commercials. In this thesis, we will assume that the stories are already segmented, and there will therefore be no need to develop a segmentation system. We can assume this because we are interested in event detection in social media, where the stories do not come from audio sources, but rather in the form of blog posts, comments, or tweets, all of which are naturally segmented at source.

2.1.1.2 Tracking

The goal of the tracking task is to detect stories that discuss a previously known (target) topic (Fiscus and Doddington, 2002). The target topic is given in the form of a set of stories that are known to be on topic. A tracking system is also given a part of the evaluation corpus for training and is required to find other on-topic stories in the rest of the corpus. Note that systems are trained on each topic independently; other topic's definitions or training stories cannot be used. Tracking is very closely linked to the first story detection task. In fact, a tracking system can be used to solve FSD, and any nearest-neighbor based FSD system could be used to solve tracking.

2.1.1.3 Detection

This task is concerned with detecting new, previously unseen topics. It is similar to the tracking task in that topics are defined by clustering stories that discuss the same

event, but it is different in that there is no training data or example stories available. This task is often also called on-line clustering because that is what a detection system basically does – every new story is assigned to an existing cluster, or a new cluster is started. Because of topic independence, evaluation in this task ignores stories which are on-topic for more than one topic.

2.1.1.4 First Story Detection

First story detection (FSD) is considered the most difficult of the five TDT tasks (Allan et al., 2000a). The aim of the task is to detect the very first story to discuss a previously unknown event, and in this way we can consider FSD as being a special case of detection. If we look at detection as the problem of on-line clustering, then FSD deals with the problem of when to start a new cluster.

2.1.1.5 Linking

Of all five TDT tasks, linking is the only one that does not have a direct application. The linking task is simply asking, given a pair of stories, whether they discuss the same topic. To avoid evaluating the systems on all $O(N^2)$ pairs, the story pairs are chosen by random sampling. The point of the task was to assess how good is the core component of a TDT system, the distance metric. Because linking is a completely artificial task and because it ignores one of the most important problems of dealing with social media – scale, we will not devote any more consideration to it in the remainder of the thesis.

2.1.2 Difference Between TDT and Other IR Tasks

As we mentioned before, the way the TDT task is defined is very different from most other IR tasks. The characteristics that make TDT special are:

1. Concept of relevance. As noted in Lavrenko (2004), probably the most important feature that sets TDT apart from other tasks is the way that relevance is defined, or rather the fact that it is *not* defined. Because TDT revolves around events, the term relevance is never used. Rather, terms like *on-topic* or *off-topic* stories replace relevant or non-relevant documents. Note that in TDT there is no concept of a query as it usually exists in tasks like ad-hoc retrieval. In TDT, the user's information need is given implicitly, either as a set of seed on-topic stories (like in tracking), a set of stories that we do *not* want to detect (first story detection), or even as an unobserved variable (detection and linking).

2. Intended users. Unlike in other retrieval scenarios (e.g., web search) where the typical user is a casual user who does not spend a lot of time formulating his request and is satisfied with a small number of relevant results, the typical TDT user is envisioned to be a professional analyst, like an intelligence analyst or a journalist. This user wants to find all the stories on an event, and is thus much more recall oriented than other users.

3. Completely online processing. Not only do all the statistics about the data have to be computed online, but all the algorithms have to be online (incremental) as well. For example, this means that a large class of clustering algorithms cannot be used for the detection task. For the FSD task, this means that the decision about whether a story is about something new has to be made right away. Allowing a certain latency (called *deferral* in the context of TDT) in making this decision results in better accuracy, but at the cost of being late in making the decision. This tradeoff between latency and performance has not been explored much in the context of TDT, but it presents an interesting problem in the social media domain where the high volume of posts (follow-ups and/or new stories) means that waiting only a few minutes before making the decision could significantly improve performance.

4. Heterogeneous streams of data and multi-linguality. Finally, TDT systems are required to deal with data from very different sources. The TDT corpora include stories from newswire (e.g., Associated Press, New York Times), radio broadcasts (Voice of America), and TV news shows (CNN, ABC, NileTV). Radio and TV sources come in an audio form, but they are transcribed (both automatically and manually), and provided to participants in this form. Furthermore, the TDT input data comes in three languages: English, Chinese, and Arabic. This substantially complicates processing as it adds another component into the system (machine translation). In social media, especially Twitter, things are even worse: nearly all languages with a reasonable number of speakers are represented, and in most cases we don't even have the information about which language the story is written in, or this information is wrong. Because handling this massive multilinguality is a major undertaking on its own, we do not address it in this thesis and choose to work only with English stories. However, most of the techniques we develop in this thesis are language-independent and could be readily used for detecting events in other languages.

2.1.3 Approaches to Solving FSD

There are several different approaches to tackling first story detection. These include various probabilistic approaches, IR-based approaches, and those based on supervised classification. By far the most prevalent and most successful systems today are the ones based on information retrieval. Because our approach is also based on IR, we will describe this approach in more detail, but also touch upon other approaches.

2.1.3.1 IR-based approaches to FSD

Systems based on information retrieval include, among others, Allan et al. (2000b), Yang et al. (1998), and Brants et al. (2003). In this approach, documents are represented as vectors in a space where the axes are the different terms in the collection. Let us denote by $f(w,d)$ the *term frequency* of term w in document d, i.e., the number of times w appears in d, and by D the entire collection of documents. Then, the weight of term w in document d is equal to:

$$weight(w,d) = f(w,d) * idf(w), \tag{2.1}$$

$$idf = \frac{\log(\frac{|D|+0.5}{|\{d':w\in d',d'\in D\}|})}{\log(|D|+1)}, \tag{2.2}$$

where $|\{d' : w \in d', d' \in D\}|$ is the number of documents that contain the term w. The weight in equation (2.1) is known as the TF-IDF weight. The *inverse document frequency* (IDF) measures the general importance of the term. Note, however, that FSD is inherently an online task, and this presents the question of how to handle the growing vocabulary and the updating of corpus-level statistics, such as IDF. Generally, there are two approaches for addressing this problem: i) use fixed statistics obtained from a corpus from a related domain, or ii) incrementally update the statistics every time a new document arrives. There is also work on combining the two above approaches (Yang et al., 1998). In this work we choose the latter approach of incrementally updating the statistics, which means that we cannot consider the document collection D to be static anymore. Let us assume that document d arrived at time t, and denote by D_t the collection of documents up to that point, including d. By substituting D_t for D in (2.2), we get the **incremental TF-IDF** model used in Yang et al. (1998) and Brants et al. (2003). We use this incremental TF-IDF model as the choice of document representation in this thesis, and it is defined as:

$$weight_t(w,d) = f(w,d) * \frac{\log(\frac{|D_t|+0.5}{|\{d':w\in d',d'\in D_t\}|})}{\log(|D_t|+1)}. \tag{2.3}$$

Once we have the vector representations, we can use them to calculate the similarities between documents. The most widely used similarity measure in TDT, which we will also use in this thesis, is the cosine similarity:

$$sim_t(d_1,d_2) = \cos(d_1,d_2) = \frac{\sum_w weight_t(w,d_1) * weight_t(w,d_2)}{\sqrt{\sum_w weight_t^2(w,d_1)} * \sqrt{\sum_w weight_t^2(w,d_2)}}. \quad (2.4)$$

It is worthwhile noting that other similarity measures have been tried in the literature. For example, Brants et al. (2003) compared the Hellinger distance to the cosine and found that it worked much better in their system. However, this is not a definitive proof that Hellinger distance is better than cosine, as other FSD systems based on the cosine distance (Yang et al., 1998) outperformed the system of (Brants et al., 2003). Other metrics that have been used include the Kullback-Leibler divergence, Jensen-Shannon distance, and the Clarity-based distance (Lavrenko et al., 2002).

To decide whether a new document describes an event that we have not previously seen, we compute its *novelty score*. The novelty score for a document d is taken to be one minus the cosine similarity (we will sometimes refer to this as the *cosine distance*) between d and the document most similar to d (aka its *nearest neighbor*) in the collection D_t:

$$novelty(d) = 1 - \max_{d' \in D_t} sim_t(d,d'). \quad (2.5)$$

If the novelty score for d is greater than a threshold θ, the system outputs a decision YES, meaning that d describes a new event. Otherwise, document d is considered to discuss a previously known event and the system outputs the decision NO. The threshold θ can be determined using labeled training data. However, systems are not required to output a hard YES/NO decision, but just the novelty score. In that case, a threshold sweep is performed and the score corresponding to the best threshold is used to score the system (cf. equation (3.5)).

Because this approach is so widely used, we have given its pseudocode in Algorithm 1. The particular variation given in Algorithm 1 uses an inverted index for efficiency, and this is the approach used by the UMass FSD system (Allan et al., 2000b). The use of an inverted index changes the novelty score in equation (2.5) to

$$novelty(d) = 1 - \max_{d': words(d') \cap words(d) \neq \emptyset} sim_t(d,d'), \quad (2.6)$$

where $words(d)$ is the set of words in document d. The formulation in equation (2.6) gives exactly the same scores as the one in equation (2.5), while avoiding comparison to those documents that have no words in common with d.

Algorithm 1: Traditional first story detection using an inverted index (used by, for example, the UMass system).

input: Novelty threshold t

```
1  index ← []
2  foreach document d in the stream do
3      // S(d) is the set of documents that have a non-zero
4      // cosine similarity with d.
5      S(d) ← ∅
6      foreach term t in d do
7          foreach document d' in index[t] do
8              update distance(d, d')
9              S(d) ← S(d) ∪ d'
10         end
11         index[t] ← index[t] ∪ d
12     end
13     dis_min(d) ← 1
14     foreach document d' in S(d) do
15         if distance(d, d') < dis_min(d) then
16             dis_min(d) ← distance(d, d')
17         end
18     end
19     if dis_min(d) ≥ t then
20         report d as a first story
21     end
22 end
```

Unfortunately, this approach is not scalable to unbounded streams. To see why, note that the max in equation (2.5) takes $O(|D_t|)$ time to compute in the worst case. This means that as we see more data, it takes longer and longer to process each document. If we are to use this in a scenario where the data is unbounded, such as microblog posts coming from Twitter, it is clear that at some point processing a single document will take longer than it takes for a new document to arrive. By using an inverted index in equation (2.6), the average time complexity of the approach is somewhat lower, and can be shown to be $O(|D_t|^b)$, where b is approximately 0.5, with the worst case complexity remaining unchanged. While this means that such an approach can handle larger datasets, it still fails to handle an unbounded stream of documents. Ideally, we want a system where the time to process a single document does not depend on the amount of data seen so far, i.e., we would want the time complexity to be constant. Later in this chapter we will describe the techniques that present a key step towards achieving this goal.

2.1.3.2 Probabilistic approaches to FSD

Besides the IR-based approaches, there is a lot of other work that deals with FSD in different ways. One strand of research uses probabilistic models – these systems are based mostly on non-parametric Bayesian approaches (Zhang et al., 2005; Ahmed et al., 2011). While the non-parametric nature of these approaches makes them appealing for FSD and detection tasks because they naturally handle an increasing number of clusters and can model cluster uncertainty, they are still lagging behind simpler approaches in terms of the official TDT evaluation metric. Also, these approaches are computationally more expensive then other approaches, which makes them even less suitable for processing large amounts of data, especially in an online setting. On the other hand, probabilistic approaches are capable of presenting data to the user in a more structured way, and have a natural probabilistic interpretation. For example, Ahmed et al. (2011) present a system that is capable of supporting structured browsing and creation of storylines from a stream of news.

2.1.3.3 Supervised machine learning approaches for FSD

Another direction in FSD is using supervised classifiers to arrive at the decision of whether a new document talks about a new event. Perhaps not surprisingly, this approach has been the most successful so far. Braun and Kaneshiro (2004) used three

classifiers, combined in a majority voting scheme. The first classifier was exactly the model used by other systems like UMass or CMU, one classifier was the standard model, but with special handling of location features, and the last classifier was based on detecting old documents via sentence linking. All the classifiers were trained on TDT3 and TDT4. In the official TDT5 evaluation, this system achieved the highest score. Kumaran and Allan (2005) presented another system that used a supervised classifier. They output three different novelty scores: one for the full document, one for only the named entities in the document, and one for the documents with only topic terms kept. The intuition behind this approach is that every event is described by a set of names that answer questions like *where, who,* and *when*, and a set of topic-specific terms that answer the question *what*. If two stories match both the names and the topic terms, they describe the same event. If either the names or the topic terms don't match, the stories describe different events. The three system scores are combined using a support vector machine (Cortes and Vapnik, 1995) trained on previous TDT corpora. To the best of our knowledge, results reported in Kumaran and Allan (2005) are the highest reported FSD results on the TDT5 corpus (although not the highest in the official evaluation). The obvious shortcoming of these approaches is that they require training data, which, for FSD, is very expensive to come by.

The only work to focus on the scalability of FSD systems is Luo et al. (2007). They introduce a number of heuristics to achieve constant document processing time: keeping only documents from the last N days, keeping only top K terms for each document, keeping only those documents deemed to be first stories, and comparing documents only if they overlap in at least one of top M terms. While they are able to achieve impressive speedups (two orders of magnitude), they do so at the expense of accuracy – ideally, we want a system that can scale to large data, where the loss in accuracy is a tunable parameter and more clearly understood. Furthermore, finding the right parameter setting is not easy, and Luo et al. (2007) show that the performance can vary substantially if the parameters are not correctly chosen. Lastly, they keep stories from the last 30 days, which was still manageable for TDT data, but would be infeasible for truly high-volume streams like Twitter.

2.2 Approximate Nearest Neighbor

We devote this section to the nearest neighbor problem, and its relaxation, the *approximate* nearest neighbor. Algorithms used for efficiently solving these problems will be

crucial for our development of an efficient first story detection system that can be used on unbounded textual streams.

Let us first consider the following *nearest neighbor* (NN) problem: given a set S of n points $S = \{p_1, \ldots, p_n\}$ in some metric space X, and a query point q, find the point in S that is closest to q. This problem can be easily solved using brute force by calculating the distance from q to every p_i in S, and picking the one that is closest. This algorithm requires $O(n)$ distance computations, and hence becomes prohibitively expensive for large n. Thus, we are interested in solving the nearest neighbor problem *efficiently*.

Unfortunately, both empirical and theoretical results show (Weber et al., 1998) that as the dimensionality of space X grows, it is not possible to achieve improvement in terms of time complexity over the simple brute force algorithm. Weber et al. (1998) showed that all the popular data- and space-partitioning methods like grid-file, k-d tree, quadtree, R-tree, TV-tree, clustering, and many others are outperformed by simple sequential search whenever the dimensionality of the data is greater than 10. In fact, Weber et al. (1998) showed an even stronger result, proving that there is no clustering or partitioning method in high-dimensional vector spaces that does not degenerate to sequential search once the dimension becomes sufficiently high. This is a very important result, as the dimensionality of the vector representation of documents is typically in the tens of thousands or even millions.

However, there are many cases where we are willing to accept a suboptimal solution, as long as it is within certain bounds of the optimal one. This is why recent research has focused on solving a relaxed version of the nearest neighbor problem, called the *approximate nearest neighbor*. First, define the R-near neighbor of a point q to be any point whose distance to q is less than or equal to R. Then, the approximate nearest neighbor problem is defined as follows.

Definition 2.1. *Randomized c-approximate R-near neighbor, or (c,R)-NN. Given a set of points S in a d-dimensional space, parameters $R > 0$, $\delta > 0$, and a query point q, if there exists an R-near neighbor of q in S, return any cR-near neighbor of q with probability $1 - \delta$.*

In the above definition, δ is a constant bounded away from 1. By increasing the approximation factor c we are willing to allow for a greater error, but the algorithms will typically be more efficient. We now define a related reporting problem that will also be of interest.

Definition 2.2. *Randomized R-near neighbor reporting problem. Given a set of points*

S in a d-dimensional space, parameters $R > 0$, $\delta > 0$, and a query point q, report each R-near neighbor of q with probability at least $1 - \delta$.

2.2.1 Locality Sensitive Hashing

The first approach to solve the approximate NN problem in sublinear time was described in Indyk and Motwani (1998), where the authors introduced a new method called *locality sensitive hashing (LSH)*. The idea behind their approach is to hash query points into buckets in such a way that the probability of collision is much higher for points that are nearby in X.

This method relies crucially on the concept of *locality-sensitive hash functions*. First, define a ball of radius r for a distance measure D as $B(q,r) = \{p : D(q,p) \leq r\}$.[1] Here, D is one minus the underlying similarity measure of our space X. Let $\mathcal{H}$ be a family of hash functions mapping the original space X to some universe U, $\mathcal{H} = \{h : X \to U\}$, and consider a process where we pick $h \in \mathcal{H}$ uniformly at random. The family $\mathcal{H}$ is then called (R, cR, p_1, p_2)-sensitive if for any two points $p, q \in X$:

- if $p \in B(q,R)$ then $\Pr_{\mathcal{H}}[h(p) = h(q)] \geq p_1$,
- if $p \notin B(q,cR)$ then $\Pr_{\mathcal{H}}[h(p) = h(q)] \leq p_2$.

Here, $\Pr_{\mathcal{H}}[h(p) = h(q)]$ is used to denote the probability that two points, p and q, hash to the same value given a hash function h that was picked uniformly at random from $\mathcal{H}$. In order for a locality sensitive hash family to be useful, it has to satisfy $p_1 > p_2$ and $c > 1$. Next, for some k, define a function family $\mathcal{G} = \{g : X \to U^k\}$, such that $g(p) = (h_1(p), \ldots, h_k(p))$, where $h_i \in \mathcal{H}$ are drawn randomly from $\mathcal{H}$. The functions g basically paste together the output of k hash functions h to produce a longer key,[2] thus increasing the gap between the probability of close points having the same key and the probability of far away points having the same key. The algorithm is then split into two phases: preprocessing, used to "index" the initial set of points S we are searching over, and querying, where we query this index and return the answer.

Preprocessing. The preprocessing step is given in Algorithm 2. This step indexes the set of points that we want to search over by assigning them hash codes and storing all the points with the same code into the same bucket in a hash table. Note that, even though the description of the algorithm assumes a static collection S we are searching

[1] A careful reader will notice that $B(q,R)$ is actually the set of all R-near neighbors of q.

[2] Notice that the *key* is in our case the output of the hash function, i.e., $g(p)$ or $h(p)$, not p itself.

Algorithm 2: Preprocessing step of the locality-sensitive hashing algorithm.

input: Number of hash tables L, set of points S

```
1 for j = 1 to L do
2 |   g_j ~ G
3 |   create hashtable T[j]
4 end
5 for i = 1 to |S| do
6 |   for j = 1 to L do
7 |   |   insert p_i in T[j][g_j(p_i)]
8 |   end
9 end
```

over, it is trivial to make the preprocessing step deal with streaming data. If we simply replace the line 5 to make i loop over all the documents in the stream, this will make the preprocessing step suitable for streams.

Querying. Pseudocode for the query phase is given in Algorithm 3. *Querying* refers to the process of finding the nearest neighbor, given the preprocessed dataset and the query. There are two different strategies for querying, each one solving a different problem. In Strategy 1, we examine all the points that collide with the query in any of the hash tables, and return those that are within distance R. On the other hand, Strategy 2 stops after finding $3*L$ points that are within cR of the query point. Strategy 1 solves the R-near neighbor reporting problem, while Strategy 2 solves the (c,R)-near neighbor problem. The difference between Strategies 1 and 2 is shown in Algorithm 3 with different letters on the lines; Strategy 1 corresponds to lines labeled with a, and Strategy 2 corresponds to lines labeled with b. Lines without any letters (e.g., the loops) are used by both strategies.

Until now, we have given a very general description of LSH that does not rely on any particular similarity measure. In practice, many locality sensitive families are known for various similarity measures. For the simplest case of the distance in a d-dimensional Hamming space H^d (the space of binary-valued vectors), the following family is locality-sensitive:

$$\mathcal{H}_{Hamming} = \{h_i : h_i(p) = p_i, p \in H^d\}. \tag{2.7}$$

A randomly drawn hash function from this family simply maps the input point p to its value at a randomly picked coordinate. It can be shown (Indyk and Motwani, 1998)

Algorithm 3: Querying step of the locality-sensitive hashing algorithm.

input : Query point q, hash tables T containing preprocessed set S

output: A set of points S' that are within R (Strategy 1) or cR (Strategy 2) of q

```
1  S' = []
2  for j = 1 to L do
3      for p in T[j][g_j(q)] do
4a         if D(q,p) < R then
5a             add p to S'
6a         end
4b         if D(q,p) < cR then
5b             add p to S'
6b         end
7b         if |S'| ≥ 3 * L then
8b             break
9b         end
10     end
11     return S'
12 end
```

that the probabilities p_1 and p_2 for this family are $p_1 = 1 - R/d$ and $p_2 = 1 - cR/d$. This means that, as long as the approximation factor c is greater than one, the family from (2.7) is locality-sensitive ($p_1 > p_2$). Other distance measures for which locality-sensitive hash families are known include the ℓ_s distance in $\mathbb{R}^d$, for any $s \in [0,2)$ (Andoni and Indyk, 2006; Datar et al., 2004),[3] ℓ_2 distance on a unit hypersphere (Terasawa and Tanaka, 2007; Andoni and Indyk, 2008), the Jaccard distance (Broder et al., 1997; Broder, 1997), and the cosine distance (Charikar, 2002). As we have seen in Section 2.1, the cosine distance performs best for the TDT tasks. Because in this thesis we also heavily use the cosine distance, we will review the locality-sensitive hashing for cosine in more detail in Section 2.2.2.

Bounds. Strategy 1 solves the randomized R-near neighbor reporting problem. To see that this is the case, let p be any R-near neighbor of our query q. For any function $g_i \in \mathcal{G}$, the probability of p colliding with q is at least p_1^k, i.e., $\Pr_{\mathcal{G}}[g_i(p) = g_i(q)] \geq p_1^k$. Then, the probability that p collides with q in at least one of the L hash tables is at least

[3] ℓ_s distance is the Minkowski distance of order s, defined as $\ell_s(p,q) = (\sum_{i=1}^{n} |p_i - q_i|^s)^{1/s}$.

$1-(1-p_1^k)^L$. Thus, if we want to report any R-near neighbor p with probability at least $1-\delta$, we can simply set the number of hash tables L to be

$$L = \log_{1-p_1^k} \delta. \tag{2.8}$$

While this strategy guarantees to report any R-near neighbor of q with high probability, it unfortunately has no guarantee on the query time. In particular, the query time could be as high as $\Theta(n)$, where n is the number of points, if the dataset is such that all the points lie within distance R from q.

On the other hand, Strategy 2 has a clear bound of $O(L)$ on the query time because we only inspect a constant number of points in each of the L hash tables. Gionis et al. (1999) showed that if we set L to $\Theta(n^\rho)$, where $\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$, this strategy solves the randomized c-approximate R-near neighbor problem. Note that if $p_1 > p_2$, the query time for this strategy is strictly sublinear. As we can see, the guarantee in running time comes at the expense of solving a weaker problem than the R-near neighbor problem.

To see how the number of bits used in the hash function (parameter k) affects the probability of collision, we look at Figure 2.1. This figure shows the probability of two points being in the same bucket as a function of the distance between them. Here we measure the distance in terms of the angle between the two points. Curves are plotted for several values of k. We can see that as the two points move further apart, there is a very low probability that they will be in the same bucket. On the other hand, if the points are very close, the probability of collision tends to 1. This is what we referred to as *locality-sensitive* behavior of the hash functions. As we add more bits to the hash function (i.e., as we increase k), we can see that it becomes more selective, i.e., there is a larger gap between the probabilities of close and far points colliding.

Figure 2.2 shows the effect that the number of hash tables L has on the probability of two points colliding in at least one of the tables. We can read the figure like this: if q is our query point and p is its nearest neighbor, the curves in Figure 2.2 show one minus the probability of failing to retrieve that nearest neighbor. We can see that, as we increase the number of hash tables, we reduce the probability of failing to find the nearest neighbor. This, of course, comes at a cost of more time spent hashing. It is interesting to note here the effect of k and L on the tradeoff between the time spent hashing and the time spent searching for the nearest neighbor. As we increase the number of bits k our hash functions become more selective, as we have seen in Figure 2.1, which means the buckets contain less points, and we thus spend less time comparing the query point to other points in the same bucket. To make sure the nearest

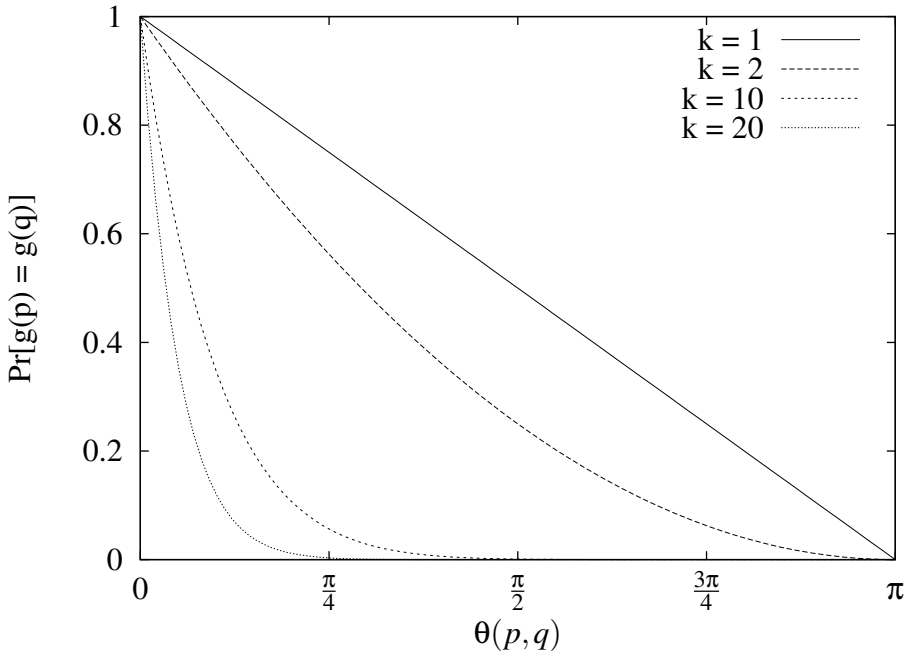

Figure 2.1: The probability of two points, p and q, colliding as a function of their distance. The hash function g is taken to be the hash function from Charikar (2002) which we will use in this thesis. This demonstrates the locality-sensitive nature of the hash functions – when the points are close the probability of collision is much higher. We show graphs for several values of k, the number of bits per hash function.

neighbor collides with the query point with a high enough probability, this in turn means we have to also increase the number of hash tables L, which means we spend more time hashing. On the other hand, if we lower k, we can also lower L, and thus spend less time hashing, but the buckets then contain more points and more time is spent comparing the query point to other points that fall in the same bucket.

2.2.2 Locality Sensitive Hashing with Random Hyperplanes

Because in this thesis we are dealing with text, a particularly interesting measure of distance will be the cosine between two documents. This is why we use the family of hash functions proposed by Charikar (2002) in which the probability of two points (documents) having the same hash key is is proportional to the cosine of the angle between them. This family was used, e.g., for creating similarity lists of nouns collected from a web corpus in Ravichandran et al. (2005). Intuitively, these hash functions partition the space with random hyperplanes, and the buckets are defined by the subspaces

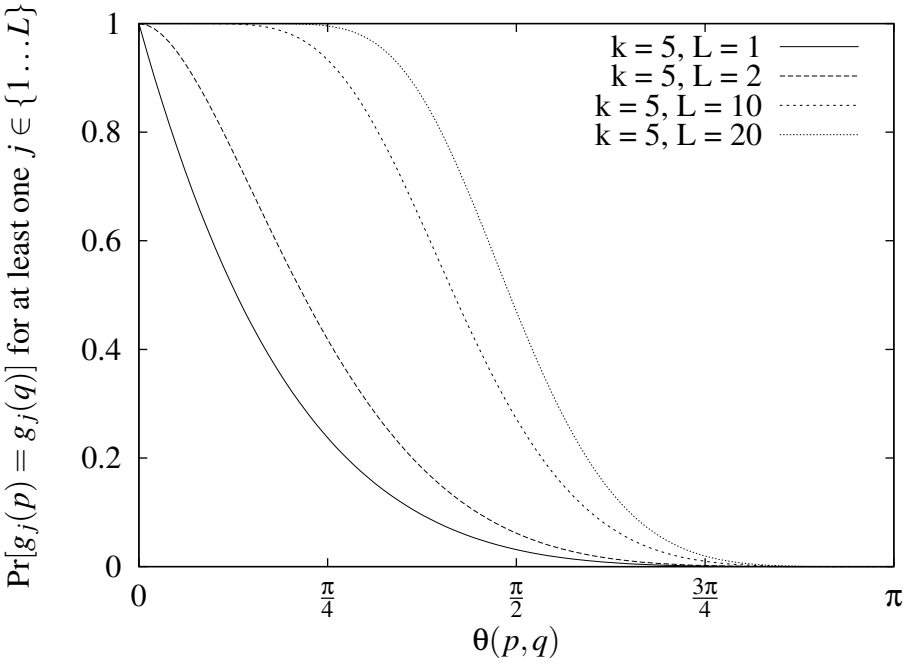

Figure 2.2: The probability of two points, p and q, colliding in at least one of the hash tables as a function of their distance. Increasing the number of hash tables L increases the probability of collision. The hash function g is the same as that used in Figure 2.1.

formed this way. More formally, the locality sensitive family of hash functions for the cosine distance is defined as follows:

$$\mathcal{H} = \{h : h(\mathbf{x}) = \text{sgn}(\langle \mathbf{u}, \mathbf{x} \rangle)\}, \tag{2.9}$$

where $\mathbf{u}$ is a random vector whose components are drawn from a Gaussian distribution with zero mean and unit variance, $u_i \sim N(0,1)$. Here, the sign of the inner product simply tells us which side of the hyperplane vector $\mathbf{x}$ is on. It is easy to see that the probability of two vectors $\mathbf{x}$ and $\mathbf{y}$ colliding under such a hash function is

$$\Pr_{\mathcal{H}}[h(\mathbf{x}) = h(\mathbf{y})] = 1 - \frac{\theta(\mathbf{x}, \mathbf{y})}{\pi}, \tag{2.10}$$

where $\theta(\mathbf{x}, \mathbf{y})$ is the angle between $\mathbf{x}$ and $\mathbf{y}$. We can read Equation (2.10) as saying *the probability of drawing a random hyperplane that separates two vectors is proportional to the angle between them.*

To illustrate this approach, we turn to Figure 2.3 which shows an example two-dimensional set of points that we are searching over, partitioned by three randomly drawn hyperplanes. Buckets are formed by the subspaces, with the "key" for each bucket, i.e., the value returned by hash functions g for all points that fall in that bucket,

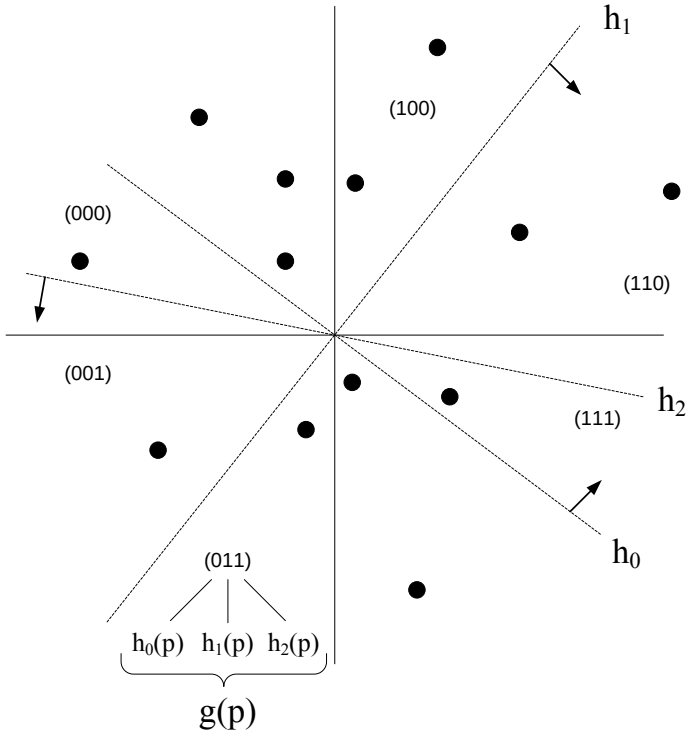

Figure 2.3: A set of points partitioned by random hyperplanes into buckets.

marked in the figure. In Figure 2.4 we have marked a query point Q red, and the blue points show the set of points that we will compare with our query point because they fall in the same bucket. We can see from the figure that this set is smaller than the set of points we started with. Finally, Figure 2.5 shows an example where LSH fails to return the nearest neighbor. Although point Q (marked red in the figure) clearly has a nearest neighbor in the point P (marked blue in the figure), P is never returned by the algorithm because it does not fall in the same bucket as Q. By keeping multiple independent hash tables (parameter L from Section 2.2), we can reduce the probability of cases like this happening.

2.3 Event Detection in Social Media

Here we review some of the related work on detecting events in social media, paying close attention to event detection in Twitter. While we make every attempt to be thorough in our review, we note that the field of event detection in social media has grown rapidly in the last few years, making it impossible to survey every article on this topic.

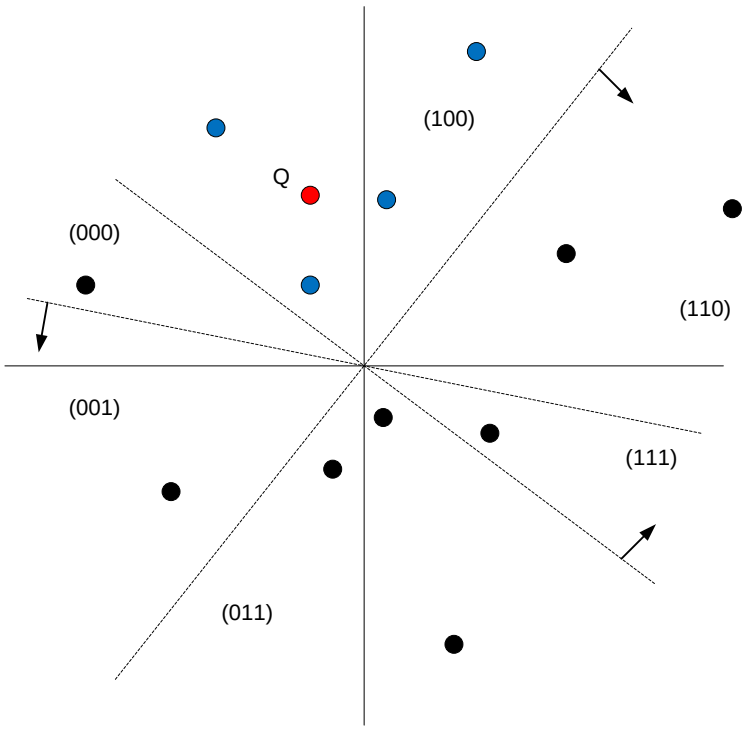

Figure 2.4: Search for the nearest neighbor of a query point Q is limited to those points that fall in the same bucket as Q. Arrows indicate the normal vectors to the hyperplanes.

Before Twitter, blogs were the most popular form of social media. Detecting events in blogs shares some of the challenges of detecting events in Twitter: there is a large amount of data that needs to be processed in an online (incremental) fashion, documents are unstructured, and there is a high amount of noise. Jurgens and Stevens (2009) addressed the problem of scale in blogs by using temporal random indexing (TRI) for event detection. TRI helps address the scale by projecting words from a high-dimensional semantic space to a random low-dimensional space. The authors then detect events by detecting semantic shifts in words between two time periods. This is also a major drawback of their approach – events are based around words, and their approach requires a user to explicitly define a set of keywords which she wants to track. This is clearly not suitable for large-scale event detection, as many events introduce completely new words that could not be specified a priori. Another problem with this approach is that it seems to require a lot of volume for the semantic shift to be accurately determined – Jurgens and Stevens (2009) used a month of data, which means they discover events with one month lag.

One of the earliest approaches to detecting events from Twitter posts is the work by Sakaki et al. (2010). This work was concerned with quickly detecting specific types

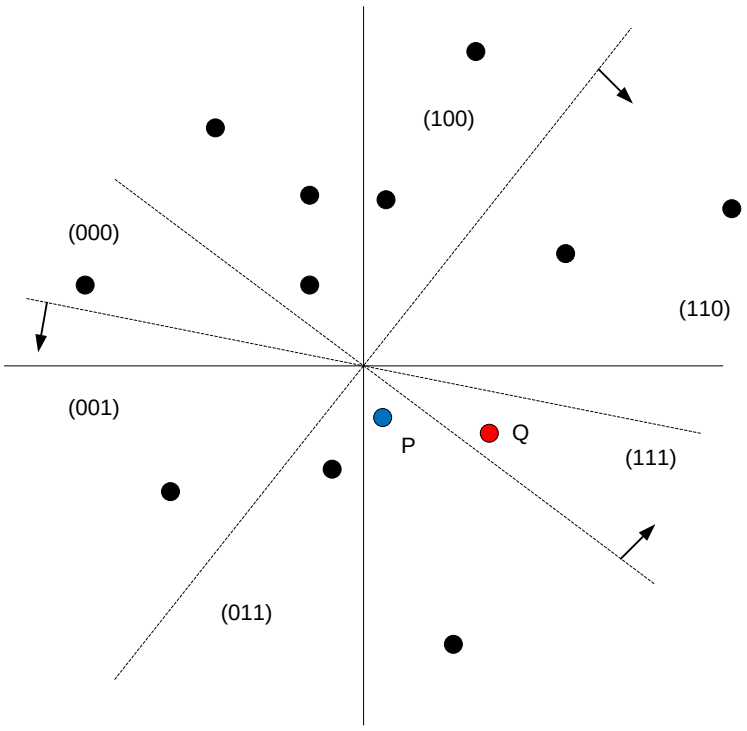

Figure 2.5: Example of LSH failing to find the nearest neighbor for a query point Q. P is never returned because it does not fall in the same bucket as Q.

of events, earthquakes and typhoons, in order to issue a timely warning for areas that are about to be hit by these disasters. Their approach consists of manually defining a set of keywords relevant for the types of events they want to detect (*earthquake*, *shaking*, and *typhoon*), and then monitoring the stream of tweets for those containing the keywords. For each tweet that contains a relevant keyword, they use a support vector machine (SVM) to classify it as being about an event or not. When enough tweets have been classified as being about an event in a short period of time, their system decides that the event is actually happening. While the authors showed promising initial results, their approach is fairly limited for two main reasons: i) a set of keywords needs to be manually defined for each event, and there was no attempt to find the keywords automatically, and ii) for every event type we want to find, a separate classifier needs to be trained, which requires labeled data. This shows that such an approach might be attractive if we want to find a small set of events with very high precision, but is unfeasible if we want to detect *any* type of event, which is the goal in this thesis.

Dealing with noise and scale issues was previously addressed in Sankaranarayanan et al. (2009). To handle noise, a preprocessing step was added which classified each tweet as being about news or not, using a naive Bayes classifier. If the tweet was not

about news, it was not processed. To address scalability, the authors compare new tweets only to clusters that are less than three days old, making this approach constant-time. This way of handling scale was previously used in TDT by Luo et al. (2007). This is probably the earliest work on event detection in Twitter, and the presented system heavily relies on many heuristics for improving the quality of found events. Unfortunately, no quantitative evaluation of the system was performed.

Becker et al. (2011a) use the clustering algorithm of Yang et al. (1998), and a classifier to handle noise like in Sankaranarayanan et al. (2009). Unlike Sankaranarayanan et al. (2009), however, the classifier is used in a postprocessing step for classifying each cluster as being about an event or not. To classify a cluster, the classifier uses a set of features derived from all the tweets in the cluster, such as the proportion of retweets, or the number of tweets containing the most popular hashtag. While this approach showed very good results in identifying event clusters, it does not address the scalability issues. Furthermore, this approach relies on having labeled data to train a classifier, and it is not clear if retraining the classifier is needed, i.e., whether its performance degrades over time.

Recent work has also used wavelets for detecting events in Twitter. Weng et al. (2011) transform the words' DFIDF[4] histogram using wavelets and then compute the cross-correlation for those words that show significant change over time. This cross-correlation is used as the weight of edges in a graph which is then partitioned, with the partitions corresponding to clusters of words about the same event. Evaluation in Weng et al. (2011) showed that their system is able to detect events with reasonable precision (76%), but also that the performance is extremely sensitive to the parameter setting (slightly increasing or decreasing one parameter lowers the precision to about 20%). Also, it is not clear how this approach would perform if a lag lower than one day was chosen, potentially limiting this approach to detecting only very high volume events. A similar wavelet-based approach was also used in Cordeiro (2012), with the difference that this approach only relied on bursty hashtags, making it more efficient. A downside to using only hashtags is that only events that contain hashtags can be detected. Given that only about 10% of all tweets contain hashtags, this severely limits the usefulness of the approach. Furthermore, Cordeiro (2012) perform no evaluation of their approach, making it unclear how well it performs. Another approach that relies on using only hashtags is Ozdikis et al. (2012). The aim there was to perform document

[4]DFIDF (Document frequency-inverse document frequency) is a counterpart of TFIDF, where a term's document frequency is used instead of its term frequency in a document. In Weng et al. (2011) DFIDF is used to measure a term's importance in a certain time period.

expansion via distributional similarity of hashtags. While authors report positive initial results, the approach still suffers from the problem of ignoring all documents that do not contain a hashtag.

Most recently, Agarwal et al. (2012) used a graph clustering algorithm for real-time event detection. The approach relies on discovering highly dense clusters in a graph where nodes are words, and edges correspond to a user using both words in her tweets. Therefore, events are discovered as sets of words. The efficiency of this approach stems from the fact that the clusters can be discovered using an approximate algorithm that performs only a small number of local computations. While this approach is very efficient, it can only detect sufficiently bursty events. Furthermore, like other approaches based on detecting sets of related bursty words, the discovered sets can be hard to interpret. Unfortunately, the evaluation of this approach is somewhat lacking, as it ignores non-bursty events, considers spam to be actual events, and does not measure the amount of meaningless or irrelevant events that are detected. Combined with the fact that evaluation is based on words, and not documents, this makes it very hard to interpret the results.

Other papers that deal with event detection in Twitter specifically include Li et al. (2012b), who detect events related to crime and disasters, but require a user to input a query in order to initiate search for events. Phuvipadawat and Murata (2010) also claim to do real-time event detection, but only focus on tweets that are tagged with the *#breakingnews* hashtag and provide no evaluation of their system. Cataldi et al. (2010), like many others, use burstiness of terms in a time interval to detect when an event is happening, but the approach is not scalable to a streaming setting, and no evaluation of the system is provided. Popescu and Pennacchiotti (2010) worked on detecting controversial events related to known named entities in Twitter. Using a supervised machine learning approach with a rich feature set, they showed that their system can achieve high precision at high ranks, with a lag of one day in detecting the events. Li et al. (2012a) detect bursty phrases in Twitter and cluster them based on the context in which they appear. These clusters are then passed through a filter to detect the ones that are *newsworthy*, where Wikipedia is used to define if something is newsworthy (phrases that appear as anchor text in Wikipedia articles are deemed to be more newsworthy than those which do not).

There has also been some work on event detection in other social media sites. For example, Hu et al. (2008) use popular queries submitted to search engines to filter the stream of news and blogs and effectively detect only those events that have a large

user interest. Using multiple social media sites to create a more complete description of events was the focus of Becker et al. (2012). They aim at identifying content for planned events in a setup similar to the TDT tracking task, but they consider three streams: Twitter posts, Youtube videos, and Flickr images. It is interesting to note that Becker et al. (2012) found that it is possible to leverage the wealth of information on Twitter to retrieve more relevant results from the other two sites. This further shows the advantages of Twitter's real-time, high-volume properties.

In Table 2.1 we summarize all of the discussed event detection approaches with respect to the properties that a modern system should satisfy, as set out in Chapter 1. We will say that an approach is *general* if it can detect any type of event, and not only, say, earthquakes. *Scalability* is satisfied if the approach can handle unbounded streams, i.e., if the processing costs are $O(1)$ per document. However, while some approaches do satisfy this condition, the hidden constant is fairly large, making them impractical for high-volume streams. This is why we say they only partially satisfy scalability. The *real-time* property is satisfied if an approach can detect events within minutes of them happening, which is another crucial property of a useful detection system. A system is *unsupervised* if it requires no training data or user interaction. Finally, we also compare approaches based on the kind of *evaluation* they perform. While this is not a property of a system, we saw in our discussion above that a lot of authors do not perform any kind of evaluation of their system, and a lot of those that do focus on intrinsic evaluation which does not show actual utility of the system. Performing proper evaluation of the system based on both recall and precision is very important as it helps other researchers establish the usefulness of an approach and facilitates comparison. We will focus more on evaluation in Chapter 6.

We can see from Table 2.1 that none of the previous work satisfies all of the requirements set forth in this thesis. Agarwal et al. (2012) come close, but their approach does not detect any event in general, but only bursty events. As we will see in Chapter 6, a lot of events do not satisfy this property. This means that we would either have to accept a low recall rate or a high latency in detecting the events so they would start exhibiting the burstiness property. On the other hand, the work that we present in this thesis is the first work to satisfy all of these properties.

Paper	General	Scalable	Real-time	Evaluation	Unsupervised
Hu et al. (2008)	✗	✗/ ✓	✗	✗	✓
Jurgens and Stevens (2009)	✗	✗/ ✓	✗	✗	✓
Sankaranarayanan et al. (2009)	✗	✓	✗	✗	✗
Sakaki et al. (2010)	✗	✓	✓	✓	✗
Popescu and Pennacchiotti (2010)	✗	✗	✗	✗	✗
Cataldi et al. (2010)	✗/ ✓	✗	✓	✗	✗/ ✓
Mathioudakis and Koudas (2010)	✗/ ✓	✗/ ✓	✗	✗	✓
Phuvipadawat and Murata (2010)	✗	✗	✗	✗	✓
Becker et al. (2011a)	✗/ ✓	✗	✗	✓	✗
Weng et al. (2011)	✗	✗	✗	✗	✓
Cordeiro (2012)	✗	✓	✓	✗	✓
Li et al. (2012a)	✗	✓	✗	✗	✓
Li et al. (2012b)	✗	✗	✗	✗	✗
Agarwal et al. (2012)	✗/ ✓	✓	✓	✗	✓
Ozdikis et al. (2012)	✗	✗	✗	✗	✓
This work	✓	✓	✓	✓	✓

Table 2.1: Comparison of several event detection approaches currently found in the literature.

2.4 Streaming

Data streams and the streaming model of computation became very popular in the recent years with the massive increase in the rate at which new data is generated. First domains that exhibited this growth in data were database systems, and routing and network monitoring. These domains are traditionally characterized by millions or even billions of transactions every second, and have a requirement to compute some kind of aggregate statistic over the whole stream. For example, one might want to know what are the most frequent destinations for IP packets passing through a particular router, or what is the size of the result of a join operation in a database, without actually performing the join.

A more formal definition of streaming (Muthukrishnan, 2005) says that data streams are represented by data that comes at a rate that is high enough to make it hard to transmit, store, or perform some computation over it. Furthermore, Muthukrishnan (2005)

defines different models of computation over data streams:

1. Time series model. In this model, each new update to an element of the stream overwrites the old value. This model might be suitable, e.g., for the volume of trade each minute.

2. Cash register model. In this model, each new update represents an increment to the old value of the particular element in the stream. This is probably the most widely used of the three models, as most practical problems fit the cash register model. For example, the number of IP packets flowing through a router, number of items in the database that have a particular value of a chosen attribute, or the number of particles having energy in a certain interval can all be expressed as a problem in the cash register model.

3. Turnstile model. This is the most general of the three models, and it allows the updates to a particular element to be negative. For example, the number of people in a subway can be expressed as a turnstile model because people can both enter and leave the subway.

Muthukrishnan (2005) defines the following desiderata for any streaming algorithm (N is the number of items that the algorithm has seen up to time t): *"At any time t in the data stream, we would like the per-item processing time, storage, as well as the computing time to be simultaneously* $o(N,t)$*, preferably polylog(N, t)."* This desiderata reveals the intimate connection between randomized algorithms (of which LSH is one example) and streaming. Where most exact algorithms provably cannot meet these conditions, randomized algorithms satisfy them perfectly.

With the amount of available textual data also growing rapidly, much thanks to the popularity of the Internet and social media, we are more and more starting to think about text as a stream, instead of a static collection of words. This trend in NLP is more than obvious; recent work on streaming in NLP includes a streaming language model (Levenberg and Osborne, 2009), a sketch for scaling distributional similarity to web-scale corpora (Goyal et al., 2010), computing mutual information between verbs in a streaming context (van Durme and Lall, 2009), and a streaming translation model (Levenberg et al., 2010), to name but a few.

Having the definition of a data stream from Muthukrishnan (2005) in mind, we can see that Twitter's microblog posts fit this definition perfectly. The posts come in sequentially at a very high rate, and it is very expensive to store them all or perform

any kind of non-trivial computation over them. That is why in this thesis we require that any algorithms we develop satisfy Muthukrishnan (2005)'s desiderata. In fact, we go even further and strengthen the conditions set forth by Muthukrishnan (2005). We will require that the *per-item processing time, storage, as well as the computing time in our algorithms be* $O(1)$. This is the strongest possible condition on an algorithm in terms of complexity, and it will ensure that the algorithms will still be efficient in the future, despite the very likely increase in the amount of available data.

Finally, we would like to highlight the main differences between the IR and the streaming view of the world. First, in the streaming approach we will normally keep only a small subset of the data and discard the rest. In IR, on the other hand, we tend to keep all the data we have, and the general consensus is that having more data is always better. Next, a streaming algorithm sees data in an online fashion and has to constantly update global statistics, whereas IR systems usually assume a static dataset and only compute all the global statistics once on the whole set. Lastly, the field of IR is slightly more biased towards efficacy, whereas streaming usually has a bias towards efficiency, allowing more approximate solutions.

2.5 Conclusion

In this chapter we presented the key background material that the work in this thesis builds upon. We first introduced the task we are dealing with in more detail, and the current state-of-the-art approaches to solving it. Because one of the key techniques in our work is LSH, we devoted some space to explaining how and why it works. We also reviewed previous work on detecting events in social media, and explained how the work here sits with respect to it. In the rest of this thesis we develop algorithms for event detection that address all the key challenges outlined in the introduction.

Chapter 3

Twitter Event Corpus

3.1 Motivation

Event detection research on Twitter is hampered by the lack of standard corpora that could be used to measure performance. Recently, Becker et al. (2011b) created a corpus of Twitter posts where tweets were labeled with events. Unfortunately, that corpus is not suitable for FSD evaluation for two main reasons: i) the events were picked from the highest-volume events identified by a specific system, introducing not only a bias towards high-volume events, but also a bias toward the kinds of events that their system can detect, and ii) only tweets by New York-based users were considered, which introduces a strong bias towards the type of events that can appear in the corpus. Other researchers working on event detection in Twitter created various ad-hoc corpora, all of which suffer from one of the two problems: either the events are very specific (e.g., Sakaki et al. (2010)), or the labeling is done on a set of potential events discovered by a particular system (e.g., Weng et al. (2011)).

In this chapter we describe the corpus we created for the purposes of measuring FSD performance in Twitter. Our corpus has several advantages over the previous Twitter event corpora: i) it is not tied to an output of any particular system, ii) it contains both low- and high-volume events, and iii) it covers a broad range of event types. The corpus was created by adhering to NIST's TDT annotation guidelines as much as possible, and adapting the rules where necessary.

	First story	Non-first story
Labeled as first story	True positive (TP)	False positive (FP)
Labeled as non-first story	False negative (FN)	True negative (TN)

Table 3.1: FSD contingency table.

3.2 First Story Detection Evaluation

Before describing our corpus, we will first give a brief overview of how FSD systems are evaluated; for a more detailed discussion, we refer the reader to Fiscus and Doddington (2002). In the official TDT evaluation, each FSD system is required to assign a score between 0 and 1 to every document upon its arrival. Assigning this score can be made either immediately after the story arrives, or after a fixed number of new stories have been observed. Because we assume that we are working in a true streaming setting, we require that systems assign a score as soon as the new story arrives. Higher scores correspond to novel stories, and vice versa. Evaluation is then carried out by first sorting all stories according to their scores and then performing a threshold sweep. For each value of the threshold, stories with a score above the threshold are considered novel, and all others are considered non-novel. Therefore, for each threshold value, one can compute the probability of a *false alarm*, i.e., probability of declaring a story novel when it is actually not, and the *miss probability*, i.e., probability of declaring a novel story non-novel (missing a novel story). Having computed all the miss and false alarm probabilities, we can plot them on a graph showing the tradeoff between these two quantities – such graphs are called detection error tradeoff (DET) curves. However, the primary evaluation metric used in TDT is the *detection cost* C_{det}. This quantity is defined as follows:

$$C_{det}(\theta) = C_{miss} * P_{miss}(\theta) * P_{target} + C_{FA} * P_{FA}(\theta) * P_{non-target}, \tag{3.1}$$

where C_{miss} and C_{FA} are costs of miss and false alarm (1 and 0.1, respectively), $P_{miss}(\theta)$ and $P_{FA}(\theta)$ are the miss and false alarm rate at the score threshold θ, and P_{target} and $P_{non-target}$ are the prior target and non-target probabilities. Given a contingency table as in 3.1, the miss and false alarm probabilities are calculated as:

$$P_{miss} = FN/(FN+TP) \tag{3.2}$$

$$P_{FA} = FP/(FP+TN). \tag{3.3}$$

Because it is somewhat hard to interpret the C_{det} measure because of its dynamic

range, it is usually normalized by the minimum expected performance of a system that either answers YES or NO to all decisions. This normalization is defined as:

$$(C_{det})_{norm} = C_{det}/MIN(C_{miss} * P_{target}, C_{FA} * (1 - P_{target})). \quad (3.4)$$

This way a score of 1.0 indicated a system that does no better than always answering YES or NO. Different FSD systems are compared using the minimal cost C_{min}, which is the minimal value of $(C_{det})_{norm}$ over all threshold values:

$$C_{\min} = \min_{\theta}(C_{det})_{norm}(\theta). \quad (3.5)$$

This means that in FSD evaluation, a *lower* value of C_{min} indicates a better system.

Because TDT evaluation uses many topics to evaluate the performance, we have to somehow report average performance of the system. This is done by averaging $C_{\min}$ across all the topics, and it is called the *topic weighted cost*. The topic weighted cost is similar to the macro averaging technique that is standard in classification and IR evaluation. Alternatively, we could use *story weighted cost* (the equivalent of micro averaging), which averages the performance over all stories. Because topics with a large number of stories would dominate in this case, the official evaluation metric uses the topic weighted cost; this is also the cost that we will report in this thesis.

3.3 Annotation Process

In this section we describe the annotation process for our event corpus. Note that due to Twitter's terms of service, we distribute the corpus as a set of tweet IDs and the corresponding annotations – users will have to crawl the tweets themselves, but this can be easily done using any one of the freely available crawlers for Twitter. This is the same method that the TREC microblog track[1] uses to distribute their data. All our Twitter data was collected from the streaming API[2] and consists of tweets from June 30th 2011 until September 15th 2011. After removing non-English tweets, our corpus consists of just over 51 million tweets.

In our annotation process, we have adopted the approach used by the National Institute of Standards and Technology (NIST) in labeling the data for TDT competitions. First, we defined a set of events that we want to find in the data, thus avoiding the bias of using events that are the output of a particular system. We choose the events from

[1] http://trec.nist.gov/data/tweets/

[2] https://stream.twitter.com/

the set of important events for the time period in our corpus, according to Wikipedia.[3] Additionally, we used common knowledge of important topics at that time to define more events. This way we define a total of 27 events.

While we tried to adhere to the NIST annotation guidelines, some changes were necessary due to the different domain we deal with. While the original definition of the event states that it is *something that happens at a specific time and place along with all necessary preconditions and unavoidable consequences*, we change this to drop the part about "all necessary preconditions and unavoidable consequences". This is to make labeling more manageable – there were more tweets about the actual event than was possible to label, and adding all the preconditions and consequences of the event would make the labeling task even harder. However, this does not pose a major problem because we are mostly interested in the real-time detection of events. For example, if an earthquake strikes, we want to know that the event happened as soon as possible, and we can find out about the relief efforts from the news.

Furthermore, the TDT definition of a story no longer makes sense in the context of Twitter. We simply define a story to be a *single micro-blog post (tweet), written by a specific user*. If we contrast a tweet with the TDT definition of a story ("a topically cohesive segment of news that includes two or more declarative independent clauses about a single event"), we see that almost no properties of the original definition are retained. Tweets need not be topically cohesive, nor do they have to refer to news (in fact, most of them do not). Also, because of the 140 character limit on tweet length, many tweets do not contain even two clauses. This shows just how drastically different Twitter is from newswire, and what a significant challenge it poses for event detection.

We faced the same problems as NIST when labeling the events – there were far too many stories to actually read and decide which (if any) events it corresponds to. In order to narrow down the set of candidate posts for each event, we use the same procedure as used by NIST. The annotator would first read a description of the event, and from that description compile a set of keywords to retrieve possibly relevant tweets. He would then read through this set, labeling each tweet as on- or off-topic (i.e., whether the tweet discusses the event or not), and also adding new keywords for retrieving the next batch of tweets. After labeling all the tweets in one batch, the newly added keywords were used to retrieve the next batch, and this procedure was repeated until no new keywords were added. This process was known in TDT as *search-guided annotation*, and it was shown to provide better consistency of annotations, compared to an

[3] `http://en.wikipedia.org/wiki/2011`

approach of exhaustively labeling each story (Fiscus and Doddington, 2002). Unlike in TDT, however, when retrieving tweets matching a keyword, we do not search through the whole corpus, as this would return far too many candidates than is feasible to label. For example, the word *earthquake*, which was one of the keywords used for retrieving tweets related to the Virginia earthquake, yields a candidate set of 16 thousand tweets just for a single day. Instead, we limit the search to a time window of one day around the time the event happened (for each event, we know the time it happened to a resolution of at least a day). Because most stories about an event are posted right after the event, this ensures that we get most of the on-topic stories, while reducing the labeling effort as much as possible. It is very likely that there are more on-topic stories outside this time window, but the tradeoff between the number of such stories and the time spent annotating makes it infeasible to label them. Note that this does not affect the TDT results in any way.

Next, we narrow down the definition of an on-topic story. This is because the NIST guidelines say a story should be considered on-topic if more than 10% of the article is about the event – with the average tweet being about 10 words long, 10% of that does not make much sense. This is why we use the following guidelines for deciding whether a tweet is on-topic:

1. The tweet had to be written in English. As we mentioned before, many languages are represented on Twitter, but we only focus on English in this thesis. Finding tweets in different languages that describe the same event could be an interesting avenue for future work.

2. The tweet must explicitly mention the event and the reader should not need any outside knowledge to infer what happened after reading the tweet. This means that tweets like “Just heard about Lokomotiv, this is a terrible summer for hockey” are off-topic, even though the author was most likely referring to the plane crash in which the Lokomotiv hockey team died.

3. The main purpose of the tweet should be to inform of the event, and stories that only briefly refer to the main event are labeled off-topic. This means that tweets like “About 200 people gathered at the hospital where the three Muslim men died” would be off-topic for the event of three Muslim men dying in England riots. This is similar to the case of stories marked BRIEF in TDT evaluation. While TDT considered these stories potential non-first stories, we do not due to

the sheer brevity of tweets and the potential to quickly drift very far away from the main topic.

4. The author had to be sure that the event happened – tweets like "Is Amy Winehouse dead?" would be off-topic. This is to prevent detecting speculation.

5. Links were not taken into account. For example, if the tweet was "NASA's big announcement: http://...", where the link pointed to the story about the event, the tweet was off-topic because it would require the annotator to read the story, not just the tweet.

6. Usernames and hashtags were taken into account. For example, a tweet like "#amywinehouseisdead" would be labeled on-topic for death of Amy Winehouse, as would the tweet *@AmyWinehouse is dead*.

7. Retweets were taken into account – if the retweet is too long and breaks the original tweet because of the 140 character limit, we label the original tweet.

8. Tweets that were about multiple events were not labeled on-topic. This is in line with NIST guidelines which ignore stories that discuss multiple topics in the evaluation (Fiscus and Doddington, 2002).

Finally, we should also note that while searching for first tweets, *fake* first tweets were sometimes discovered. For example, in the case of the death of Richard Bowes (victim of London riots), there was a tweet posted by a journalist from the Telegraph informing of the man's death more than 12 hours before he actually died. This tweet was later retracted by the said journalist for being incorrect, but the man then died a few hours later. Cases like this were labeled off-topic.

3.4 Corpus Statistics

In this section, we give some basic statistics about our corpus. In total, the corpus consists of over 51 million tweets that have to be processed by the FSD system. Of those, 3034 are labeled on-topic for one of the 27 events. The number of events in our corpus is comparable to the first TDT corpus which contained 25 events. However, in terms of the total number of documents, our corpus is three orders of magnitude larger than the first TDT corpus, and two orders of magnitude larger than the biggest TDT corpus (TDT5). The biggest event in our corpus had over 1,000 on-topic tweets (death

of Amy Winehouse), and the smallest event had only two on-topic tweets (arrest of Goran Hadžić).

Full details of our corpus are given in Table 3.2. Along with the topic description and the number of on-topic documents, we also give the topic's *broad type*, a set of broad categories defined by NIST to help classify topics.[4] Furthermore, we also note which events were *planned* and which ones were *unplanned*. A planned event is simply one that was known that it will happen before it happened, e.g., presidential elections in the US. All other events are unplanned, like earthquakes, plane crashes, or terrorist attacks. Planned and unplanned events were also known as *expected* and *unexpected* events in TDT terminology. Almost all the related work deals with detecting events regardless of their type, with the exception of Becker et al. (2012) that detect only planned events. Deciding which events are planned is not an easy task – in many cases it was known that something would happen, but it was not known exactly which of the possible outcomes will occur. For example, when US raised the debt ceiling, it was known well in advance that there will be a vote on whether to raise the ceiling. However, no one could say with certainty what the outcome of the vote would be. Cases like this were labeled as planned.

Because news on Twitter are often claimed to be reported in real time, we decided to test this claim on the events in our corpus. For each event, we found the exact time (to a resolution of one minute) it happened and measured the time it took for the first tweet to appear in our data. We could not find exact time for eight of the events in our corpus, either because an exact time does not make sense for an event (e.g., it is hard to define a single time for the event *Three men die in riots*), or because this information was unavailable (e.g., Betty Ford's family did not disclose the exact time of her death). Therefore, all the statistics we report concern only those 19 events for which we knew the exact time when they happened. The average lag between an event happening and the first tweet in our data is 129 minutes, or just over two hours. The event *Goran Hadžić arrested* had by far the largest lag of all the events in our corpus. This is because the news was first broken in other languages (non-English), and only later picked up by English speakers on Twitter. In particular, the first non-English tweet to report this news in our data was a tweet in German that lagged only 42 minutes behind the actual event. This was followed by a tweet in Italian with a lag of 2 hours and 12 minutes, and a tweet in Serbian with a lag of 3 hours 41 minutes, only to be first reported in English with a lag of 22 hours and 18 minutes.

[4] Broad topic type was also known under the title *rule of interpretation* in the TDT project.

Topic description	On-topic	Broad topic type	P/U	Lag
Amy Winehouse dies	1021	Celebrity/Human interest news	U	1h17m
Atlantis shuttle lands	49	Science and discovery news	P	0m
Betty Ford dies	14	Celebrity/Human interest news	U	N/A
Richard Bowes killed in riots in England	39	Acts of violence or war	U	39m
Flight 4896 crash	11	Accidents	U	1h52m
S&P downgrade US credit rating	334	Financial news	P	N/A
US increases debt ceiling	89	Financial news, also New laws	P	0m
Terrorist attack in Delhi	39	Acts of violence or war	U	9m
Earthquake in Virginia	318	Natural disasters	U	2m
First victim of London riots dies	85	Acts of violence or war	U	0m
War criminal Goran Hadžić arrested	2	Legal/Criminal cases	U	22h18m
India and Bangladesh sign a border pact	4	Political and diplomatic meetings	P	N/A
Plane with Russian hockey team Lokomotiv crashes	277	Accidents	U	54m
Explosion in French nuclear plant in Marcoule	162	Accidents	U	57m
NASA announces there might be water on Mars	127	Science and discovery news	P	8m
Google announces plans to buy Motorola Mobility	145	Financial news	U	N/A
Car bomb explodes in Oslo, Norway	28	Acts of violence or war	U	12m
Gunman opens fire in youth camp in Norway	32	Acts of violence or war	U	47m
First artificial organ transplant	17	Science and discovery news	P	N/A
Petrol pipeline explodes in Kenya	32	Accidents	U	2h47m
Famine declared in Somalia	86	Natural disasters	P	N/A
South Sudan becomes independent country	34	Misc news	P	5m
South Sudan becomes UN member state	9	Misc news	P	N/A
Three men die in riots in England	16	Acts of violence or war	U	N/A
Riots break out in Tottenham, England	30	Acts of violence or war	U	48m
Rebels capture International Tripoli Airport	5	Acts of violence or war	U	0m
Ferry sinks in Zanzibar	29	Accidents	U	7h56m

Table 3.2: Statistics about our corpus. The broad topic type denotes the broad category the event falls into, as defined by TDT5. P/U column denotes if the event is (P)lanned or (U)nplanned. The lag between the time when the event actually happened and the time when it was first reported in our data is shown in the last column.

The median lag of the 19 events for which we know the exact time they happened was 39 minutes. We can see that the lag for planned events is much lower, because people expect them to happen and tweet about them as soon as the outcome is known. Remember also that our data constitutes a 1% sample of Twitter. This means that the lag we report is actually an upper bound on the true lag present in Twitter, and that for many events the actual lag is going to be much lower. Thus, it is certainly fair to say that most events are indeed reported on Twitter in real time or very close to real time.

3.5 Conclusion

In this chapter, we presented a corpus of tweets with labeled events that will be used throughout this thesis to evaluate the performance of our FSD approaches on Twitter. This is the first corpus that can be used for this purpose and it addresses all of the shortcomings of the previous Twitter event corpora. By measuring the lag between an event taking place and its first mention in our data, we confirmed that most events are indeed reported on Twitter in real time, which was one of the most important reasons for undertaking the work in this thesis. In the next chapter we show how to scale FSD systems to handle unbounded streams such as Twitter.

Chapter 4

First Story Detection in a Streaming Setting

4.1 Motivation

The first challenge to overcome in modern event detection is dealing with the current scale of the data. To illustrate this problem, consider that in March 2011 Twitter users were posting over 140 million tweets per day, with a peak of $6,939$ tweets per second.[1] By June 2011, there were more than 200 million posts being written each day.[2] Even though we are only able to access a small sample (1%) of the full stream, this still means that we deal with more than 2 million documents per day. For comparison, the largest TDT corpora (TDT5) contained $278,108$ documents for a period of six months (April 1st 2003 through September 30th 2003). This amounts to an average of $1,545$ documents per day, three orders of magnitude less than our small sample of Twitter posts, or five orders of magnitude less than the full Twitter stream. If we recall the discussion from Section 2.1.3, all the current approaches to event detection don't scale well with the number of documents. With the exception of Luo et al. (2007), all the approaches take $O(n)$ time to process a new document, which becomes prohibitive quickly. In this chapter we present a novel algorithm for first story detection that achieves $O(1)$ processing time per document, while making no sacrifice in accuracy, i.e., our system achieves the same performance as the state-of-the-art systems.

The algorithms we develop in this chapter will be scalable in terms of the per-document processing time, which will enable us to process streams of unbounded size.

[1] Source: `http://blog.twitter.com/2011/03/numbers.html`
[2] Source: `http://blog.twitter.com/2011/06/200-million-tweets-per-day.html`

In order to process high bandwidth streams of documents, we will need to apply some kind of parallel processing, and here we also give an example of how our approach might be parallelized.

4.2 Scaling FSD to Unbounded Streams

As we already mentioned in Section 2.1.3, the main reason why existing FSD approaches are not scalable comes from the maximization in equation (2.5) which takes $O(n)$ time to compute in the worst case. The main idea we introduce here is to compute this maximum in an approximate way, thereby replacing exact search with an algorithm for finding an approximate nearest neighbor. By using LSH we introduce a one-sided error: the distance to the approximate nearest neighbor will always be greater than or equal to the distance to the true nearest neighbor. The assumption we make here is that the accuracy does not depend crucially on the value of this distance, i.e., that small errors in the distance will not significantly hurt performance.

In particular, we replace the brute force search from equation (2.5) with locality sensitive hashing based on the cosine distance (Charikar, 2002). Using LSH in this manner guarantees that the time to find an approximate nearest neighbor is $O(n^{1/c})$ (where c is the chosen approximation factor) in terms of the number of documents seen so far. The pseudocode for this algorithm is shown in Algorithm 4. Here we used the same notation from Algorithm 2, where $\mathcal{G}$ is a family of k-bit locality-sensitive hash functions. We can see that the loop which computes the distance to the nearest neighbor now iterates only over the documents in the same LSH bucket as the query point, computed in step 7.

We should note here that LSH is not the only technique that achieves fast retrieval of similar documents. Semantic hashing (Salakhutdinov and Hinton, 2009) is an alternative technique that assigns binary codes to documents by using multiple-level restricted Boltzmann machines. While Salakhutdinov and Hinton (2009) claim that this technique outperforms LSH in both accuracy and running time, they failed to take into account the time required to train the RBMs, which is substantially longer than the time it takes LSH to index the collection. It is also not clear that semantic hashing yields better accuracy, as the authors used E^{2}LSH (which works in ℓ_2 space) for retrieving candidates, but evaluated on cosine similarity. However, the main reason why we do not use semantic hashing is because it is ultimately unusable in a streaming scenario – it cannot handle online updates in any other way but retraining the whole system, and

Algorithm 4: First story detection with approximate-NN based on LSH.

input: Number of hash tables L, novelty threshold θ

for $j = 1$ *to* L **do**
  $g_j \sim \mathcal{G}$ // Draw a random hash function.
  create hashtable T[j]
end
foreach *document d in the stream* **do**
  // Find the set of documents that collide with d.
  $S(d) \leftarrow \bigcup_{i=1}^{L} T[i][g_i(d)]$
  for $j = 1$ *to* L **do**
    $T[j][g_j(d)] \leftarrow T[j][g_j(d)] \cup d$
  end
  $dis_{min}(d) \leftarrow 1$
  if $S(d) \neq \emptyset$ **then**
    $dis_{min}(d) \leftarrow 1 - \max_{d' \in S(d)} \cos(d', d)$
  end
  if $dis_{min}(d) \geq t$ **then**
    report d as a first story
  end
end

there is no way of handling deletions. These two points are the main requirements for a model to be useful for streaming applications, and LSH easily handles both.

4.2.1 Variance Reduction Strategy

Unfortunately, we cannot simply replace exact nearest neighbor search with approximate search because of the one-sided error introduced by LSH. LSH only returns the true near neighbor if it is reasonably close to the query point, but if the query point lies far away from all other points (i.e., its nearest neighbor is far away), there is a high probability that LSH will fail to find the true near neighbor. This can be easily seen from Figure 2.2. For example, a document whose nearest neighbor resides at distance $\pi/4$, has only about 40% chance that LSH with $k = 5$ and $L = 2$ will find its true nearest neighbor. In FSD, this means that a large portion of moderately novel stories (e.g., stories that discuss a novel aspect of a previously known event) would be thought to

discuss a previously unseen event. Clearly, this is not desirable, and the experimental results from Section 4.3 confirm this.

To overcome this problem, we introduce a strategy by which, if LSH declares a document novel (i.e., sufficiently different from all others), we start an exact search (the details of which are given in Algorithms 6 and 7), but only compare the new document with a small and fixed number of documents, ignoring those documents that we have already inspected. If this additional search does not find a closer nearest neighbor, we declare the document novel. Otherwise, we revise our decision based on the distance to this "better" nearest neighbor. On the other hand, if LSH declares the document to be non-novel, we do not have to do anything. This is again a consequence of the one-sided error introduced by LSH – a document can never be incorrectly declared non-novel (i.e., the distance to the true nearest neighbor is never more than what LSH reports). To the best of our knowledge, we are the first to notice that pure LSH is unsuitable for FSD. This is because all previous tasks which used LSH only care about the close nearest neighbors (document/image retrieval, clustering), and thus operate in the region where LSH introduces only a small amount of error. However, because novel and non-novel documents are both important in FSD, this means that we cannot afford the one-sided error from LSH. Thus, the strategy proposed here is a first step into making LSH a feasible option for other tasks that rely on nearest-neighbor search, but do not rely only on finding close nearest neighbors (any task that involves finding novelty in the data or spotting outliers falls into this category).

Because the accuracy of our FSD system exhibits a lot of variance when using pure LSH, we dub the strategy introduced here a *variance reduction strategy*. We will sometimes also use the term *backoff strategy* to refer to it because it can be seen as a backoff from approximate search to limited exact search.

The pseudocode shown in Algorithm 5 summarizes our approach based on LSH with the variance reduction strategy. In short, we hash the document into L buckets and find its nearest neighbor amongst the documents in those buckets. If there was a high chance that the true nearest neighbor was missed (because it is far away from the document), we additionally compare the document to b_n most recent documents and make a final decision based on this.

For efficiency, we use an inverted index to store documents, much like other systems (Allan et al., 2000b; Yang et al., 1998). By specifying the desired probability of missing the nearest neighbor, we can easily compute the threshold at which we should resort to our variance reduction strategy. If the nearest neighbor is at distance r (recall

Algorithm 5: First story detection based on LSH with variance reduction.

input: Number of tables L, novelty threshold θ, backoff threshold b_t, number of documents in the backoff set b_n

```
for j = 1 to L do
    g_j ~ G                                   // Draw a random hash function.
    create hashtable T[j]
end
foreach document d in stream do
    S(d) = ⋃_{i=1}^{L} T[i][g_i(d)]
    for j = 1 to L do
        // Insert d in the appropriate hash table.
        T[j][g_j(d)] ← T[j][g_j(d)] ∪ d
    end
    dis_min(d) ← 1
    foreach document d' in S(d) do
        c = 1 − cos(d, d')
        if c < dis_min(d) then
            dis_min(d) ← c
        end
    end
    if dis_min(d) >= b_t                      // Variance reduction step.
    then
        // Try to find a better nearest neighbor by performing a
        // search over b_n additional documents.
        // This step is explained in Algorithms 6 and 7.
        dist_b = BackoffNN(d, S(d), b_n)
        if dis_min(d) < dist_b then
            dis_min(d) ← dist_b
        end
    end
    if dis_min(d) >= θ then
        report d as a first story
    end
end
```

that r is an angle here) from the query point, the probability that it is not reported is

$$P_{miss} = (1 - (1 - \arccos(1 - r)/\pi)^k)^L. \tag{4.1}$$

Therefore, we tune the parameter b_t by choosing to tolerate a $p\%$ chance of missing the nearest neighbor at distance of r or more from the query point.

4.2.2 Constant Time Approach

The approach we presented in the previous section paves the way for an efficient FSD system, but the time complexity of this approach depends crucially on the number of hash tables L, and so far we said nothing about how to set this parameter. As we mentioned in Section 2.2, there are two ways of setting this parameter, which we called Strategy 1 and Strategy 2. Both of these strategies have weaknesses. In Strategy 1 L is set independently of the number of documents n, which makes it suitable for stream processing, but on the other hand the query phase has no guarantees on the running time – it could be $\Theta(n)$ in the worst case because we have to inspect all the points in the appropriate buckets. Strategy 2, however, lets us examine only a constant number of points in the buckets, but at the cost of setting L dependent on n.

To achieve constant processing time per document, we thus combine Strategy 1 and 2: we set L according to Strategy 1 (see (2.8)), and examine the buckets according to Strategy 2. This way we are guaranteed to have L independent of the stream size, which is important for use in a streaming setting, and we are also guaranteed that the query phase will take constant time. This desirable property comes at a cost – our combined strategy is no longer guaranteed to solve either the R-near neighbor or the (c,R)-near neighbor problem. While we lose the theoretical properties guaranteed by Strategy 1 and 2, this is unfortunately necessary if we hope to achieve constant processing time. Otherwise, it has been proven that the R-near neighbor requires $\Theta(n)$ query time, and that solving (c,R)-near neighbor requires setting L dependent on n (Andoni and Indyk, 2008). Inspecting the buckets according to Strategy 2 is already shown in Algorithms 4 and 5, and choosing L is done by the user so we do not show this in pseudocode.

Note that until now we said nothing about how to actually inspect the additional b_n documents once we decide to use the variance reduction strategy. In Algorithm 5 this is done in the *BackoffNN* function. This function takes as input the current document, collision set (set of all documents that fall in the same bucket as the query point, cf. line 6 in Algorithm 5), and the b_n parameter and inspects at most b_n documents not in the collision set, returning the distance to the closest of those documents. We propose two

Algorithm 6: Recency strategy for performing search in variance reduction.

```
BackoffNN(d, S, b_n)
i ← 0
B ← ∅                    // Set of additional documents to compare with.
while (|B| < b_n) ∧ (t > i)          // Inspect at most b_n documents.
do
    i ← i + 1
    // Skip documents that were already inspected
    // or have no words in common with d.
    if (d_{t−i} ∩ d ≠ ∅) ∧ (d_{t−i} ∉ S) then
        B ← B ∪ d_{t−i}
    end
end
dist_b = 1 − max_{d′∈B} cos(d′, d)
return dist_b
```

different strategies for performing the search: a recency based one where fresh documents are preferred, and a uniform search where diversity of documents is preferred over recency. The recency based strategy is shown in Algorithm 6, and the uniform one is in Algorithm 7. We can see that the recency strategy simply iterates over the last b_n documents that share at least one word in common with the query document. On the other hand, the uniform strategy uses an inverted index to inspect $b_n/\|d\|_0$ documents[3] that share each of the $\|d\|_0$ words with the query document. When there are more than $b_n/\|d\|_0$ such documents for a word, the most recent $b_n/\|d\|_0$ are inspected (not shown in the algorithm). These two strategies have different strengths and weaknesses: recency prefers fresh data, which is intuitively a desirable property, but it might spend all of its time comparing the new documents to documents that have little in common with it, other than a possibly very common word. On the other hand, uniform search will inspect a greater variety of documents, but it might compare the new document to very old documents which, though they may seem similar, are usually not related.

[3] $\|d\|_0$ is the ℓ_0 norm of document d, and is equal to the number of distinct words in d.

Algorithm 7: Uniform strategy for performing search in variance reduction.

```
1  BackoffNN(d, S, b_n)
2  p ← b_n/||d||_0          // Number of documents to inspect per term.
3  B ← ∅                    // Set of additional documents to compare with.
4  foreach word ∈ d do
5      i ← 0
6      // Use inverted index for efficient iteration.
7      // We iterate in the reversed order, i.e., from most recent
8      // documents first.
9      foreach d' ∈ reverse(InvertedIndex[w]) do
10         if (d' ∈ B) ∨ (d' ∈ S) then
11             // Skip documents that have already been added or
               inspected.
12             continue
13         end
14         B ← B ∪ d'
15         i ← i + 1
16         if p = i then
17             break
18         end
19     end
20 end
21 dist_b = 1 − max_{d'∈B} cos(d', d)
22 return dist_b
```

4.2.3 Constant Space Approach

In Section 4.2.2 we showed how to achieve constant time using a combined strategy for LSH. Despite being a constant time algorithm, the approach described there is still not suitable for stream processing because it uses an unbounded amount of space. To see this, simply note that we never delete any documents from the LSH buckets (or the inverted index). To achieve bounded space usage, at some point we will have to start deleting the stored documents.

We present a number of ways for choosing which document to delete when a new document comes in. We first define two main axes along which we choose the documents: i) what is the set of documents we consider for deletion (we will call this the *candidate set*), and ii) how to pick a document from the candidate set. Note that the candidate set is defined for each of the L hash tables, and thus we have L candidate sets. We propose three strategies for picking each candidate set ($CS_i, 1 \leq i \leq L$):

1. *Global*: $CS_i =$ the set of all documents stored so far. Note that this implies that all candidate sets are the same.

2. *Collision*: $CS_i = \bigcup_{m=1}^{L} \{d' : h_{mj}(d_t) = h_{mj}(d'), \forall j \in [1 \dots k]\}$, i.e., all the candidate sets are equal to the set of documents that collide with the new document.

3. *Bucket*: $CS_i = \{d' : h_{ij}(d_t) = h_{ij}(d'), \forall j \in [1 \dots k]\}$, i.e., the candidate set is different for hash table i, and is equal to the set of documents that fall in the same bucket as the new document in that particular hash table.

If our candidate set is chosen according to global or collision strategy, the document we pick for deletion is immediately deleted. If, however, the candidate set is chosen according to the bucket strategy, the document is deleted only from one hash table, and may still be present in other hash tables. We delete the document altogether (including deleting from the inverted index) only when it is deleted from all the hash tables. Given a candidate set of documents that we are considering for deletion, we have two ways of choosing which document to delete:

1. Choose the oldest document

2. Choose a random document

In addition to choosing the oldest document and a random document from the CS, another simple strategy would be to delete the newest document. Of course, one would

not expect that this strategy will be a very successful one (or indeed a very reasonable one) because it ignores any new information and relies solely on old data. Despite this, we ran initial experiments using this strategy and the results we obtained were no better than random. This just shows that in FSD looking only at the old data is the same as having no data at all and just making random decisions. Because of this, we do not devote any more space to experimenting with this strategy.

4.2.4 Parallelizing Our Approach

While the constant time approach presented in this chapter is able to process an unbounded stream of documents, it still has a limit on the bandwidth of the documents it can process. It is unreasonable to expect our approach to process the entire Twitter stream ($\sim$ 200 million tweets per day) on a single core on one machine. This is the same problem all large scale systems face, and it is normally addressed by using more cores and/or machines to distribute the computation load. Currently, there are two major directions for doing this. Approaches such as Google's Mapreduce framework and the open-source Hadoop project distribute the computation by splitting the data into chunks and having each node in the cluster process one chunk. Another way of parallelizing is carried out by having every node perform different calculations on the same set of data. Probably the best known example of this kind of approach is multithreading (implemented, e.g., in the pthread library).

While the Mapreduce paradigm has been gaining more and more popularity in recent years, it is unfortunately inherently unsuitable for the kind of online processing that FSD requires. This is because establishing whether a story is about a new event requires information about all the stories that arrived before it, and this means that a shared global state must be maintained and updated after every new story comes in. As noted in Lin and Dyer (2010), maintaining this global state becomes very expensive because of the frequent synchronization needed. Thus, the speedup gained from distributing the computation is outweighed by the extra amount of time spent starting jobs and performing synchronization, making Mapreduce unsuitable for such online processing. Another way to look at this is the following: Mapreduce is very efficient for large batch jobs, but in FSD (and other online tasks) the batch size is one, making Mapreduce inefficient for such tasks.

On the other hand, our approach naturally lends itself to parallelization using the multithreading paradigm. In particular, there are two functions that can easily be par-

allelized: hashing the stories, and comparing the new story to other stories in the collision set. We show the parallel versions of these two functions in Algorithm 8. Note that in step 27 of the algorithm we use the slice notation $S[i:j]$ to denote the subset $\{S_i, \ldots, S_{j-1}\}$ of S. We can see that the hashing is parallelized by having each thread hash the new document into a subset of the hash tables. The collision set S is then formed by simply performing a union over all the collision sets returned by individual threads. Distance computation is parallelized in a similar manner, where each thread computes the distance from the new document to a subset of candidates. The final nearest neighbor is taken to be the document closest to the new document, out of the set of most similar documents returned by each thread. We will explore the gains we get from parallelizing our approach in Section 4.3.

4.3 Experiments

4.3.1 Scaling and Variance Reduction

In this section, we perform experiments that show the efficacy and efficiency of our system, as well as explore the tunable parts of it. First, we will show that our system based on LSH together with the variance reduction performs as well as a state-of-the-art system. For this purpose, we compare our system to the UMass system (Allan et al., 2000b). The UMass system has participated in the TDT2 and TDT3 competitions and is known to perform at least as well as other existing systems who also took part in the competition (Fiscus, 2001). Because we use the same settings as the UMass system (1-NN clustering, cosine distance, incremental TFIDF), this ensures that any difference in results is due to approximations made by LSH. We limit both systems to keeping only the top 300 features in each document, where the features are sorted according to decreasing TFIDF score. Using more than 300 features barely improves accuracy for either system, while taking significantly more time for the UMass system. In other words, using more features only increases the gap in running time between our system and the UMass system. Because stemming is usually found to be helpful, we explored using both the Krovetz stemmer (Krovetz, 1993) and the Porter stemmer (Porter, 1980) in both systems. The difference between the two is that the Porter stemmer tends to stem too aggressively, and the Krovetz stemmer was designed to fix the overstemming caused by Porter. We set the LSH parameter k to 13, and L such that the probability of missing a neighbor within the distance of 0.2 is less than 2.5%. The distance of 0.2 was

Algorithm 8: Parallelized FSD based on LSH.

input: Number of hash tables L, number of threads N

```
ParallelHash(d, thread_id)
tables_per_thread ← L/N
begin_index ← thread_id * tables_per_thread
end_index ← begin_index + tables_per_thread
S = {d' : ∃i ∈ [begin_index..end_index], g_i(d') = g_i(d)}
return S

DistanceComputer (d, S)
for d' in S do
    current_max ← 0
    if cos(d, d') > current_max then
        current_max ← cos(d, d')
    end
    return current_max
end

Main thread
foreach document d in the stream do
    for i = 0 to N - 1 do
        create ParallelHash(d, i)                    // Distribute hashing.
    end
    // Wait for all ParallelHash threads to finish.
    S_i ← Receive(ParallelHash)
    S ← ⋃_{i=1}^{N} S_i
    // Number of documents each thread has to process.
    dpt ← |S|/N
    for i = 0 to N - 1 do
        // Distribute distance computation.
        create DistanceComputer(d, S[i * dpt : (i + 1) * dpt])
    end
    // Wait for all DistanceComputer threads to finish.
    local_max_i ← Receive(DistanceComputer)
    dis_min(d) ← 1 − max_{1≤i≤N} local_max_i
    ...
end
```

System	$C_{\min}$
UMass (Porter)	0.708
UMass (Krovetz)	0.742
Luo et al. (2007)	0.758
LSH w/o variance reduction (Strategy 1)	0.845 (0.039)
LSH w/ variance reduction (Krovetz)	0.683 (0.012)
LSH w/ variance reduction (Porter)	0.713 (0.013)

Table 4.1: Comparison of our system to a state-of-the-art system. Numbers in parentheses are the standard deviations of 10 runs. All systems used 300 features.

chosen as a reasonable estimate of the threshold when two documents are very similar; we shall see later how the choice of k and L affects the performance of our system.

TDT5 results. Table 4.1 shows the $C_{\min}$ scores for different FSD systems. For all of our systems, the $C_{\min}$ score we report is the mean $C_{\min}$ of 10 runs of the system, and the number in parentheses is the standard deviation of those 10 runs. First, we compare our systems to the state-of-the-art UMass system. We can see that without variance reduction our system achieves a rather poor result of 0.845, with a standard deviation of 0.039. By using variance reduction, this result is improved by approximately 19%, and the variance is reduced by an order of magnitude (equivalently, standard deviation is reduced over three-fold). Comparing our approach that uses variance reduction with the UMass system, we can see that we perform equally well, meaning that our system achieves the performance level of a state-of-the-art system. The DET curves for the UMass system and our approach that uses variance reduction are shown in Figure 4.1.

One natural question that arises here is whether our LSH-based approach is really needed to achieve constant processing time and whether a simpler approach might achieve the same accuracy. To answer this question, we compare our approach to the one in Luo et al. (2007) which just keeps a history of last n documents. We can see that our approach outperforms this baseline by 10%. This shows that there are benefits in using a more principled approach like LSH over the simple sliding window algorithm. Finally, we look at the effect of stemmers on $C_{\min}$. While UMass benefits more from using the Porter stemmer, our system performs better using Krovetz stemmer. In fact, our system based on LSH even slightly outperforms the UMass system when using the Krovetz stemmer. On the other hand, using the Porter stemmer our system performs slightly worse than the UMass system, but only by 0.7%.

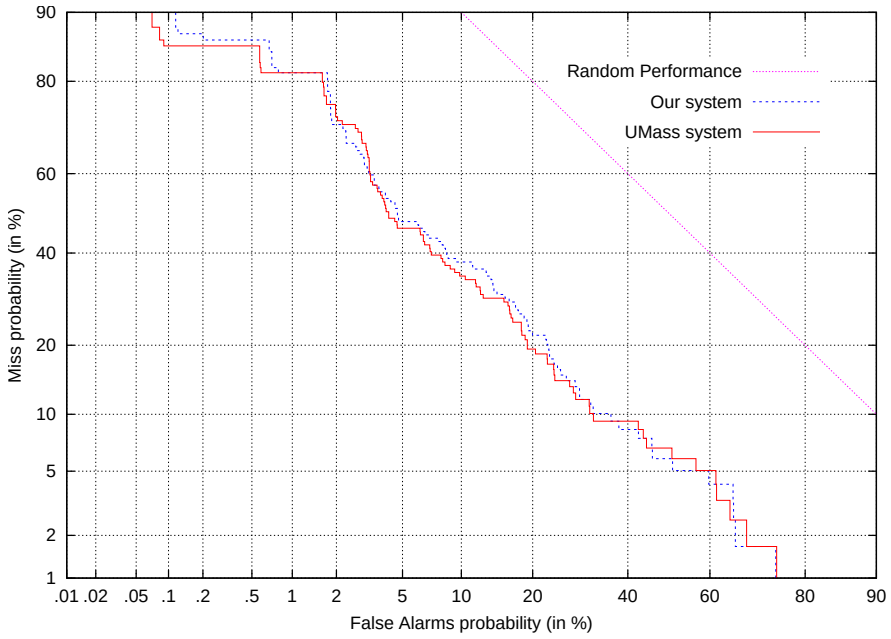

Figure 4.1: DET curve for our system and the UMass FSD system.

Figure 4.2 shows the time required to process 100 documents (in seconds) as a function of the number of documents seen so far. This confirms the previous theoretical results: our system maintains constant time, whereas the UMass system processing time grows without a bound (roughly linear with the number of previously seen documents). The total time it took the UMass system to process the TDT5 corpus was 46 hours, compared to 2 hours for our system. This means that our approach performs equally well as a state-of-the-art system, while being more than 20 times faster. More importantly, our system maintains constant processing time for each document, whereas the state-of-the-art system's processing time grows linearly.

Twitter results. Performance of our system on the Twitter data is shown in Table 4.2. Due to its linear time complexity, we cannot run the UMass system on this data, as discussed earlier. First, we examine the effect of different Twitter-specific features on the FSD performance. We can see that simply removing the username mentions and the links from tweets improves the results substantially. This is because usernames rarely contribute to the content of the tweet, and mostly serve to throw off the nearest neighbor search. Links hurt performance mostly because of the use of multiple services for shortening URLs, i.e., one article appearing as many different links shortened through various services (bit.ly, t.co, goo.gl, ow.ly, etc.). On the other hand, we can see that removing hashtags has the opposite effect. This shows that hashtags do contribute to

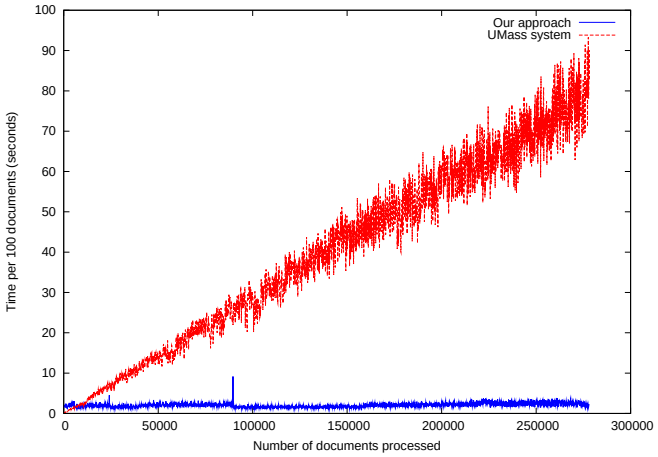

Figure 4.2: Comparison of processing time for our and the UMass system.

System	$C_{\min}$
Baseline (no stemming, nothing removed)	0.828
Removed usernames and links	0.700
Removed the hash sign	0.832
Removed the hashtags altogether	0.876
Removed usernames and links, (Porter)	0.805
Removed usernames and links, (Krovetz)	0.747

Table 4.2: FSD results for Twitter data.

the content of tweets and should be kept. Treating hashtags as normal words (by removing the hash sign) also seems to degrade performance, albeit only a little. This indicates that hashtags should be treated as separate tokens, instead of trying to treat them as normal words. Finally, we examine the effect of stemming the tweets. As the last two rows of Table 4.2 show, any kind of stemming hurts the performance, which is exactly the opposite of what was the case with TDT data. We conjecture that this is because stemmers are designed for clean data, and Twitter data is so noisy (i.e., full of grammatical errors and misspellings) that stemmers do more damage than good.

k \L	10	30	50	70
5	0.672	0.703	0.736	0.718
6	0.689	0.703	0.720	0.716
7	0.666	0.693	0.721	0.717
8	0.666	0.701	0.704	0.725
9	0.673	0.679	0.692	0.711
10	0.678	0.694	0.704	0.719
11	0.666	0.685	0.707	0.696
12	0.674	0.680	0.661	0.682
13	0.671	0.675	0.680	0.685
14	0.674	0.671	0.663	0.686
15	0.666	0.684	0.663	0.685

Table 4.3: Effect of k and L on the $C_{\min}$ score in TDT5.

4.3.2 Effect of LSH Parameters

We first look at the effect of LSH parameters k and L on the performance of our system. These two parameters determine the tradeoff between accuracy of our system and its running time. Table 4.3 shows how $C_{\min}$ depends on k and L. Looking at increasing L for any fixed k, we can see that using a higher L actually hurts performance slightly, and that this degradation is more pronounced for lower values of k. This effect is a consequence of the combination of LSH and the variance reduction strategy, and we explain it in detail in Section 4.3.3.

Having seen how k and L affect $C_{\min}$, it is also important to know how they affect the efficiency of our system. We show the running time of our system in Table 4.4. We look at the results along two axes. First, changing L for a fixed k gives the obvious effect – as we increase L, the running time goes up because we have to do more hashing and we compare each document to a larger set of potential candidates. However, changing k for a fixed L is much more interesting. We can see that initially increasing k leads to decreased running time, achieving a minimum around $k = 10$, and further increasing k leads to increasing running time. These results illustrate the tradeoff between the time our system spends hashing the document, and the time it spends comparing the document to candidates for nearest neighbor. Initially (small k), the system does very little hashing, but as a result the LSH frequently fails to find a

"good" nearest neighbor, so the variance reduction strategy fires more often, leading to lots of time spent comparing the document to potential nearest neighbors. As we increase k, we spend more time hashing, but the hash functions become more selective. As a result, the variance reduction fires less frequently, having a total positive effect on the running time of our system. Further increasing k means even more time spent hashing, but now the frequency with which variance reduction is invoked does not decrease, so this has a total negative effect on the running time.

We can now give a more general description of how to choose a good setting for k and L in our FSD system. Looking at both Tables 4.3 and 4.4, one might be tempted to conclude that setting k around 10 or 11, and L to 10 gives the best tradeoff between performance and running time. Indeed, this would yield a $C_{\min}$ score of around 0.67, and it would take only 28 minutes to process the entire TDT5 collection, which means that we would improve the state-of-the-art results by 5%, while also being 64 times faster. However, this kind of reasoning would lead to overfitting the parameters to the TDT5 data. There are two main reasons for this. First, setting L to such a low value like 10, with a fairly high k of 10 leads to poor guarantees from LSH. In particular, with $k = 10$ and $L = 10$, the probability of LSH finding a nearest neighbor with a cosine similarity of 0.6 (which constitutes a moderately similar document) is only 26%. This means that our variance reduction strategy would fire often, which is not very desirable because our performance then depends more on the variance reduction parameters, which do not have as clear probabilistic interpretation. Also, when processing a bigger dataset one would likely have to set the b_n parameter to a higher value, which would mean that the running time would increase. Thus, we want to set L to a high value. Now, looking back at Table 4.3, we see that with high values of L, it is better to choose a higher value of k. As a result of this discussion, we suggest setting both k and L higher than indicated by the numbers in Table 4.3 and 4.4. This way, our hash functions are more selective, and we rely less on the variance reduction strategy because it fires less frequently. Another advantage of such a parameter setting is that more time is spent hashing, which is beneficial because hashing is more amenable to parallelizing, especially using GPUs.[4] The exact k and L we use are 13 and 70, corresponding to a $C_{\min}$ score of 0.685, and a running time of 93 minutes. With this parameter setting, the probability of LSH finding a nearest neighbor with a cosine similarity of 0.6 is 54%, twice as much as when k is 10 and L is 10.

[4]This is because hashing in LSH involves multiplying a dense random vector with a sparse document vector. This is easier to parallelize than computing the distance between two documents, which involves multiplying two sparse vectors. We do not experiment with parallelizing our approach using GPUs.

k \L	10	30	50	70
5	75	212	320	343
6	53	119	186	244
7	37	75	121	162
8	31	55	83	113
9	27	45	67	90
10	27	42	61	81
11	28	47	60	92
12	28	44	61	90
13	31	56	75	93
14	33	51	73	109
15	32	50	73	120

Table 4.4: Effect of k and L on running time. Time is shown in minutes.

4.3.3 Why is Approximate Better than Exact?

As we see from the results in Tables 4.1 and 4.3, our approximate technique sometimes outperforms the exact technique. This is somewhat surprising, as one would normally expect that the approximations we make should only make our system perform worse, not better, than the exact approach. This is why we devote some space here in explaining why this happens.

The reason for outperforming the exact system lies in the combination of using LSH and the variance reduction strategy. Using LSH means that, with very high probability, we will find the nearest neighbor of a newly arrived document if this neighbor is very close, regardless of when it was written (i.e., how old this nearest neighbor is, compared to the newly arrived document). Using the variance reduction strategy ensures that we will find the nearest neighbor that may be somewhat similar to the newly arrived document, but that was written recently. What our approach will *not* find are nearest neighbors that are i) somewhat similar to the newly arrived document, and ii) written long ago, compared to the newly arrived document. This means that our approach implicitly models time and that it will improve the novelty scores of first stories that have a somewhat similar nearest neighbor that was written long ago, because it will not find such nearest neighbors.

This also explains why using a lower L for a fixed k gave better results in Table 4.3.

A high value of L increases the probability of finding the nearest neighbor, regardless of when it was written. In the limit, if we let $L \to \infty$ our approach would find every nearest neighbor exactly and thus its performance would be the same as that of the exact system. However, with a lower value of L we only find *close* nearest neighbors with high probability. This means that our approach will find two types of nearest neighbors: those that are very similar to the document, or those that were written recently. As we lower L this effect will become more pronounced, which is in line with the numbers in Table 4.3. In short, lowering L for a fixed k will increase the effect of our implicit time penalty.

In Table 4.3 we also saw that the difference in accuracy between using a small L and a high L was more pronounced for smaller k and less pronounced for higher k. This is easily explained by noting that k has an exponential effect (increasing k by one will reduce the number of candidates by half), whereas L has a linear effect (increasing L by one will only add a constant number of candidates for a nearest neighbor). This means that for a higher k we would need to increase L a lot more to see the same difference as we do when we compare a low and high L for a lower k. In short, while the implicit time penalty almost disappears for a low k and high L, it is still present for a higher k with the same value of L.

In order to illustrate this effect, we manually inspected the stories where the novelty score was improved by using a lower L. We found, e.g., that the novelty score of the first story for the topic *Edward Said dies* was improved from 0.67 to 0.77 by using $L = 10$ instead of $L = 70$. The first story on this topic was published on September 25^{th} 2003. When using $L = 70$, the nearest neighbor of the first story in this topic was from April 3^{rd} 2003, while setting $L = 10$ yielded a nearest neighbor that was written on September 23^{rd} 2003. This example shows that we are implicitly imposing a time penalty, and that we can set how "aggressive" this penalty should be through k and L.

Note that we are not the first to explore using time penalty for FSD. Similar (but more explicit) models that take time into account have been successfully used in TDT1. For example, Yang et al. (1998) used a fixed window of n most recent documents and only searched over documents within this window, much like the approach of Luo et al. (2007). However, these models were unsuccessful in TDT2 and TDT3, and were thus largely abandoned by the TDT participants. The fact that we are finding improvements with our implicit time modelling suggests that such models would have been useful in TDT5, had they been tried.

4.3.4 Effect of Parameters for Variance Reduction Strategy

In this section, we investigate the effect of the two parameters in our variance reduction strategy. The first one, backoff threshold b_t, determines if the nearest neighbor returned by LSH was good enough and if we should try and find a better nearest neighbor by doing a limited search through the inverted index. The second one, b_n, determines the number of documents included in this limited search.

We first examine the effects of the backoff threshold b_t. Lower values of this threshold mean that we will resort to the additional search more often, whereas higher values mean that the additional search will only be done in rare cases. Figure 4.3 shows that our approach is fairly robust with respect to this parameter – any value up to 0.6 yields results comparable to the exact system. There is no point in trying values lower than 0.3 because this just means we are getting closer to the exact system in performance. In fact, for $b_t = 0$ our system would always perform exact search, and thus reduce to the basic approach used in other systems. We can see that setting b_t too high (0.8 or more) yields much worse results. A very high backoff threshold means that the variance reduction strategy "fires" only rarely, and it is thus not surprising that the $C_{\min}$ score then approaches the score of the system without variance reduction (cf. Table 4.1). Taking an extreme case where $b_t = 1$, our system would reduce to the approach with no variance reduction whose performance is given in Table 4.1. Based on Figure 4.3, we recommend setting b_t anywhere in the range between 0.5 and 0.6. In our experiments, we always use 0.6, which means we prefer the variance reduction to fire less frequently.

The other parameter of our variance reduction strategy determines exactly how many additional documents to inspect, once we choose to use this strategy. As we mentioned before in Section 4.2.2, we will compare two different ways of inspecting additional documents. The first one is based on *recency* and thus simply looks at the last N documents that share at least one word with the current document. The other strategy is to inspect, for every word in the query document, $N/\|d\|_0$ most recent documents which contain that word. We will call this strategy *uniform*. We compare these two strategies on both TDT and Twitter data.

TDT results are shown in Figure 4.4. We can see that the strategies give comparable results, but have very different behavior with respect to the number of documents used for backoff. While for recency it is generally better to search over a larger number of documents, uniform search achieves very good performance much earlier, using only 500 or 1000 documents (compared to about $10,000$ for recency). Figure 4.5 shows

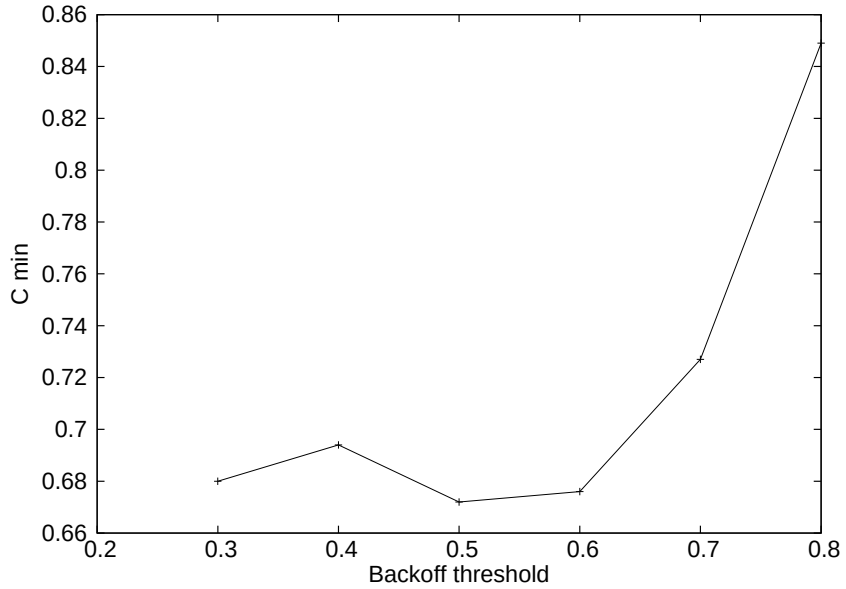

Figure 4.3: Effect of the backoff threshold on $C_{\min}$.

the same results for Twitter data. The difference between the two strategies is much more apparent now: even though the recency-based strategy might achieve the same performance as the uniform one, it is unclear how many documents it would take for this to happen. With 5,000 documents in the backoff the system basically achieves random performance, and even with 50,000 documents, performance using recency strategy is substantially below that when using uniform strategy. Using more than 50,000 documents for backoff would lead to a very slow system (remember that with $L = 70$ we only inspect 210 documents suggested by LSH) so we did not try higher values. Figures 4.4 and 4.5 clearly show that uniform search is the preferred way of inspecting additional documents within our variance reduction strategy.

4.3.5 Comparison of Deletion Strategies

A crucial part of making our system suitable for stream processing is its ability to maintain constant space, which means deleting documents. Here we compare the different strategies for choosing which document to delete once the allocated space fills up. We compare six strategies which correspond to all the combinations of choosing the candidate set and choosing the document from that set, as defined in Section 4.2.3.

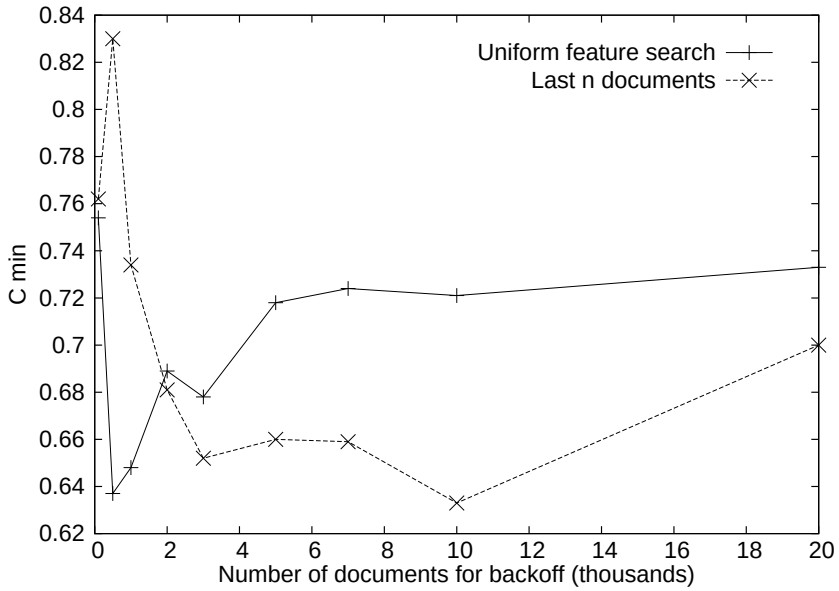

Figure 4.4: Comparison of the recent and the uniform strategy for backoff on TDT data.

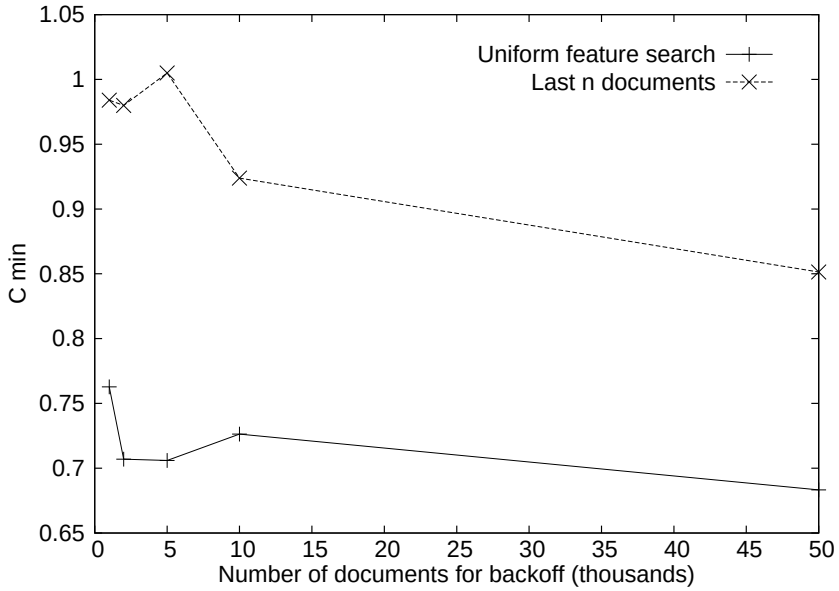

Figure 4.5: Comparison of the recent and uniform strategy for backoff on Twitter data.

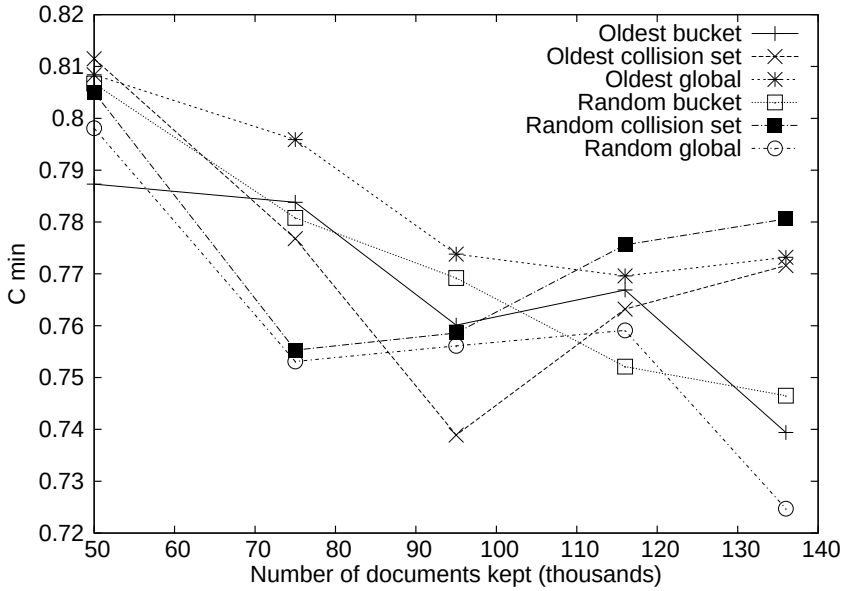

Figure 4.6: Comparison of deletion strategies on TDT5 data.

Results for TDT data are shown in Figure 4.6, and results for Twitter data are shown in Figure 4.7. For the TDT data two things are apparent: i) in general, the more documents are kept, the better the results are, and ii) it is very hard to say if one strategy outperforms others, i.e., all strategies perform similarly. On Twitter, the situation is a bit clearer: all the strategies that delete the oldest document perform very similarly, and in general they seem to perform better than the random strategies. One main difference between Twitter and TDT is that keeping more documents in Twitter hurts performance. This result supports the intuition that Twitter is a real-time information source where fresh news are very important, and looking too far back in the past will hurt performance. The number of documents kept that achieves the best $C_{\min}$ cost for Twitter corresponds to roughly one hour of data that we receive, which shows just how fast-paced this stream is.

4.3.6 Parallelization

Finally, we want to know if parallelizing our approach can help us reduce the running time. We run our system that uses parallelization as described in Algorithm 8 with the default parameters ($k = 13$, $L = 70$, $b_t = 0.6$, and $b_n = 2000$). Figure 4.8 shows the

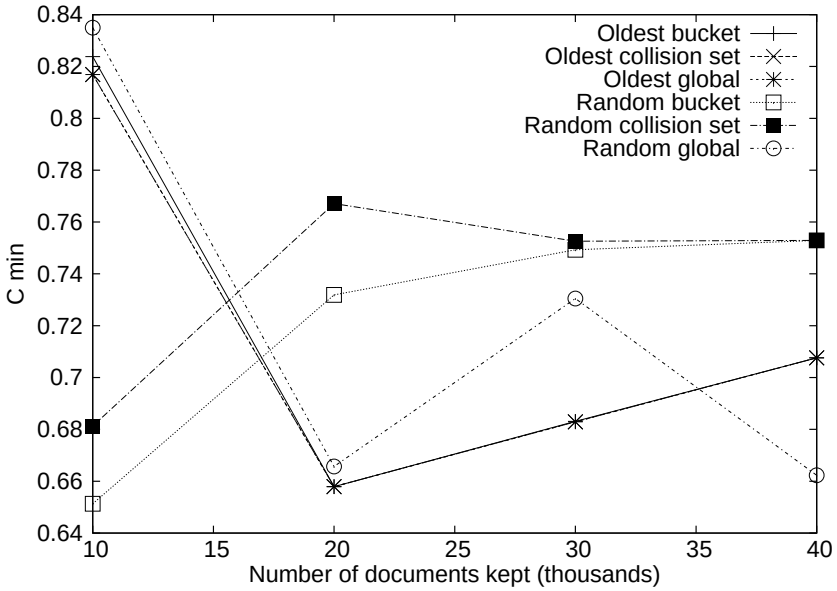

Figure 4.7: Comparison of deletion strategies on Twitter data.

running time of our system with different number of threads (the number of cores on the machine was 24). We can see that initially there is an almost linear gain in speedup, but it quickly flattens out for more than four threads. When using four threads, we get a three-fold speedup over the single-threaded version. This experiment shows that there is definitely gain to be had in parallelizing our approach, but when using more than four threads further gains are insignificant, if any. This is most likely due to the fact that the gain in distributing the computation load is offset by the overhead of creating and destroying threads.

4.4 Conclusion

The first, and arguably the most important challenge that modern event detection systems face is dealing with high-volume, unbounded nature of the data. In this chapter we introduced a way to scale existing FSD approaches to unbounded streams, while keeping state-of-the-art results. The core of our approach consisted of using LSH for fast search coupled with a strategy for reducing the variance of results introduced by the one-sided error of LSH. We further showed how to make this approach constant in both space and time, and what different steps we can make in that respect. We ex-

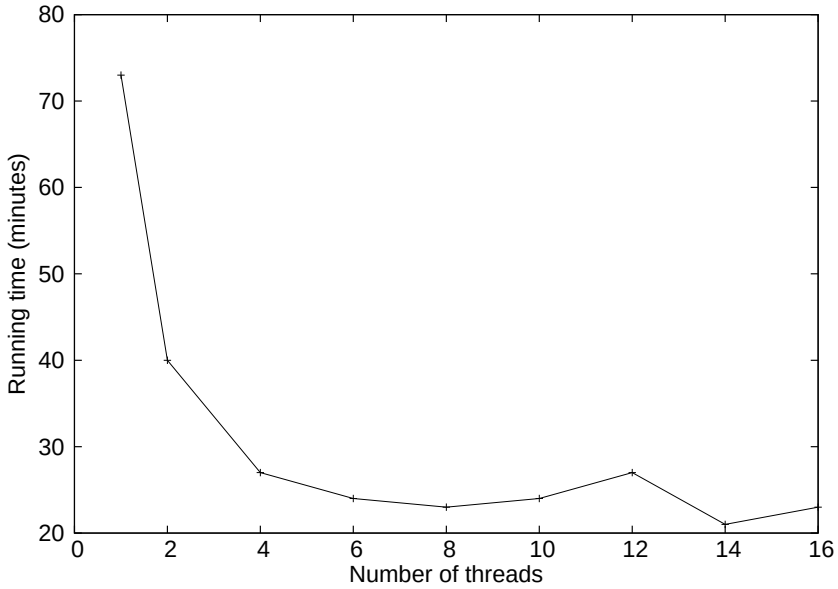

Figure 4.8: Running time of our system as a function of the number of parallel threads.

plored the different aspects of our approach and gave an overview of how they affect performance and how to set the appropriate parameters. Our final system achieves state-of-the-art performance on the FSD task, while processing the TDT5 dataset an order of magnitude faster than previous state-of-the-art system. Finally, we showed a way to parallelize our approach in order to increase the bandwidth of documents it can handle. In the next chapter, we will look into further improving the state-of-the-art in FSD, while keeping in mind the need for efficient and scalable solutions.

Chapter 5

Improving First Story Detection Using Paraphrases

5.1 Motivation

In the previous chapter we showed how to scale first story detection to unbounded data streams, which is a typical scenario when doing event detection in both news and social media. In this chapter, we focus on addressing another major issue present in all of the TDT tasks – the problem of *lexical variation*. This includes situations when the same event is being referred to using different words. In the information retrieval literature, this is sometimes also referred to as the *vocabulary mismatch* problem. To illustrate the problem, consider the following three tweets:

1. *Bomb blast near the Delhi high court*
2. *Explosion outside Delhi high court*
3. *Waiting outside Delhi high court*

Although any human would say that the first two documents are much more similar, all standard similarity measures used today will say that documents two and three are the most similar. In particular, for the cosine measure, assuming that no IDF weighting has been applied, the similarity between documents one and two is 0.507, compared to a similarity of 0.8 between documents two and three. Even with IDF (or similar) weighting, the situation would not change significantly – documents two and three will still be more similar than one and two, while the gap between them might increase or decrease slightly, depending on the corpus used to collect the IDF counts.

In this chapter, we introduce a novel approach to dealing with lexical variation in FSD. Our approach is based on using paraphrase information to detect when two documents talk about the same event, even though they may use different words. In our example above, *explosion* would be a paraphrase of *blast*, and knowing this information would help us establish that tweets 1 and 2 are more similar than originally thought. We show that our approach not only outperforms competing approaches on the FSD task, but that it is also more efficient, which was one of the main requirements set out at the beginning of the thesis.

5.2 Background

The problem of lexical variation is not specific to the TDT domain. If anything, it is inherent to natural language, and thus plagues many IR and NLP tasks. It is therefore not surprising that the problem has received a lot of attention in the literature. In this section we review the main approaches proposed for dealing with this problem.

5.2.1 IR Approaches

Some of the earliest work on solving this problem can be found in the information retrieval literature under the umbrella term *query expansion*. The basic idea is simple (assuming an ad-hoc retrieval[1] scenario): the original query is expanded with additional terms in order to match documents that otherwise would not have been retrieved. There has been a great deal of work on the different ways in which this can be done. The first work to use query expansion for TDT was Papka et al. (1999), where multiword features were used to expand documents for tracking (thus, the new document was considered to be the query). Papka et al. (1999) found that this helped the story-weighted cost, but decreased the topic-weighted cost. Overall, the differences were very small and the conclusion was that such expansion was not successful.

There are many other ways of performing query expansion, but here we focus on *relevance models* because they are generally found to perform the best across a number of IR tasks (Lavrenko, 2004). Relevance models (Lavrenko and Croft, 2001) are an instance of a more general technique called *pseudo-relevance feedback*, which uses the set of documents retrieved by the original query in order to estimate additional context

[1] Ad-hoc retrieval is the standard retrieval task where a user's information need is specified through a query that then initiates a search over a static set of documents in order to find those that are likely relevant to the user.

for the query. Relevance models assume that both the query and the documents relevant to the query are just samples from the same underlying language model R. Therefore, given a query Q and a set of documents D, if we can estimate the language model R, we can expand Q with additional words according to $p(w|R)$. Lavrenko and Croft (2001) present a principled way of estimating R given Q and D based on co-occurrence statistics. In the context of TDT, relevance models have been shown beneficial for tracking (Lavrenko and Croft, 2001) and linking (Lavrenko et al., 2002), but they have not been used in other TDT tasks. Here we explore for the first time the utility of using relevance models for first story detection.

Most recently, Ozdikis et al. (2012) used document expansion for event detection in Twitter. Their expansion was based on second-order relations, which is also known under the name *distributional similarity* in NLP. While expanding the tweets this way showed some promising initial results, this approach is inherently unsuitable for our purposes as it does not scale to the amount of data that we deal with.

5.2.2 Machine Learning Approaches

Another way to mitigate the problem of lexical variation is by using machine learning approaches, particularly those based on topic models (Blei et al., 2003). These approaches explicitly model topics as latent variables, usually represented as a multinomial distribution over words. These models can thus capture the notion of homonymy (e.g., two meanings of the word *crane* can be captured by the fact that it appears in the topic *animals* and in the topic *construction*) and synonymy (e.g., words *soldier* and *warrior* can appear in the same topic). Because documents are represented in the latent topic space, this alleviates the problem of lexical mismatches.

In the context of event detection, topic models have been explored in Ahmed et al. (2011). They propose a hybrid clustering and topic model approach with rich features such as entities, time, and topics. This probabilistic approach is able to support structured browsing and creation of storylines. Thus, it is able to provide the user with a rich experience of browsing a collection of news documents. The drawback of this approach is the complex inference method that makes it fairly slow. For example, even when using multiple threads, this approach is much slower than our approach when using a single thread. Ahmed et al. (2011) do not explicitly work out the time complexity of their system, but they do note that it is not constant. Because Ahmed et al. (2011) also use this approach for FSD, we will compare it with our approach.

5.2.3 Paraphrases

In this section we give a short overview of what paraphrases have been used for in the past. There are several levels of paraphrasing – lexical paraphrases, where the relationship is restricted to individual lexical items, phrasal paraphrases, where longer phrases are considered, and sentential paraphrases, where entire sentences are in a paraphrastic relationship. Here we use the simplest form, lexical paraphrases, but our approach is general and it would be trivial to use phrasal paraphrases in the same way.

Paraphrases were already shown to help in a number of tasks. For example, Callison-Burch et al. (2006) found that translating paraphrases of unknown phrases improves the performance of a machine translation system. Paraphrases have also been used for query expansion in information retrieval (Spärck Jones and Tait, 1984; Jones et al., 2006), or for improving question answering (Riezler et al., 2007). A much more detailed discussion on the use of paraphrases and ways of extracting them is given in Madnani and Dorr (2010).

5.3 Using Paraphrases in First Story Detection

5.3.1 Paraphrasing as a Bilinear Form

In this section, we explain how to use paraphrases in a first story detection system. We account for paraphrases by changing the way the cosine similarity is computed in equation (2.4). We do this by redefining the inner product using a bilinear form induced by a binary word-to-word matrix of paraphrases $\mathbf{Q}$. An entry of 1 at row i and column j in this matrix indicates that words i and j are paraphrases of each other, and a 0 means that they are not. To see that such a matrix defines a bilinear form, it is sufficient to note that the matrix is symmetric (because the paraphrasing relation is symmetric), and contains real entries (zeroes and ones in our case).[2] Our new inner product is defined as:

$$\langle \mathbf{x}, \mathbf{y} \rangle_Q = \mathbf{y}^T \mathbf{Q} \mathbf{x}, \tag{5.1}$$

which means that the new cosine distance we use is given by

$$\cos_Q(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{y}^T \mathbf{Q} \mathbf{x}}{\sqrt{\mathbf{x}^T \mathbf{Q} \mathbf{x}} \sqrt{\mathbf{y}^T \mathbf{Q} \mathbf{y}}}. \tag{5.2}$$

[2]This is of course a simplification – in general, one might like the entries in the matrix to be real numbers corresponding to the probability that two words are paraphrases. We leave this for future work.

Note that technically equation (5.1) does not define a valid inner product because the positive definiteness property does not hold. However, this is not a problem in practice as all the document vectors in the TFIDF representation have positive entries, effectively making (5.1) behave like a proper inner product. If we revisit the example with three short documents from the beginning of the chapter, by having information that the word *explosion* is a paraphrase of the word *blast*, the cosine similarity between the first two documents would increase from 0.507 to 0.676. When using IDF weighting, words *explosion* and *blast* would likely get much higher weights than words *the* and *near*, ultimately leading to tweets 1 and 2 being more similar than tweets 2 and 3.

As a running example in this chapter we will use the following simple paraphrasing matrix consisting of five words:

$$\begin{array}{c} \\ jump \\ spring \\ source \\ informant \\ witness \end{array} \begin{array}{c} \begin{array}{ccccc} jump & spring & source & informant & witness \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array} \tag{5.3}$$

For example, this matrix tells us that words *spring* and *source* are paraphrases, and that *witness* and *jump* are not. This matrix will be used later in the chapter to illustrate some key properties of our approach.

This approach of using paraphrases can be viewed as a special case of query/document expansion using a similarity matrix. This general technique has appeared before under many names – generalized vector space models (Wong et al., 1985), expansion based on a similarity thesaurus (Qiu, 1995), expansion based on a statistical thesaurus (Crouch and Yang, 1992), etc. All of these techniques use equation 5.1 to define a new similarity measure between documents, where the matrix $\mathbf{Q}$ is used to introduce some notion of similarity between terms.

As an example, we show that our bilinear form (5.1) can be seen as a special case of the GVSM used in Tsatsaronis and Panagiotopoulou (2009). Other work, e.g. Qiu (1995), makes this connection even more obvious. Tsatsaronis and Panagiotopoulou (2009) define the inner product of two n-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ as

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} \sum_{j=1}^{n} SR(i,j)(x_i y_i + x_i y_j + x_j y_i + x_j y_j) \quad (5.4)$$

$$= \mathbf{y}^T \mathbf{Q} \mathbf{x} + \mathbf{x}^T \mathbf{Q} \mathbf{y} + 2 \sum_{i=1}^{n} x_i y_i \sum_{j=1}^{n} SR(i,j) \quad (5.5)$$

$$= 2\mathbf{y}^T \mathbf{Q} \mathbf{x} + 2 \sum_{i=1}^{n} x_i y_i \sum_{j=1}^{n} SR(i,j), \quad (5.6)$$

where $SR(i,j)$ is the semantic relatedness of terms i and j, and $\mathbf{Q}$ is the matrix where the (i,j)-th entry is equal to $SR(i,j)$. In our case, this relatedness can take on the values zero or one (words i and j are either paraphrases or not). Furthermore, comparing (5.6) to (5.1) we can see that Tsatsaronis and Panagiotopoulou (2009) give more weight to terms that have larger total semantic relatedness (the $\sum_{j=1}^{n} SR(i,j)$ factor). We perform no such weighting, as it is not clear what it is trying to achieve.

While previous work has largely concentrated on obtaining this matrix using statistical techniques, e.g., from co-occurrence counts of terms in the documents, there were some attempts towards using linguistic knowledge to construct the similarity matrix. Fox et al. (1988) use a parser to parse definitions of terms in several dictionaries in order to obtain similar terms. They report "mild improvements" in retrieval performance, but unfortunately do not quantify those improvements. Wallis (1993) is the only work we are aware of that explored using linguistic paraphrases in information retrieval. They obtain paraphrases from a dictionary (Longman Dictionary of Contemporary English) and use them to expand queries in an ad-hoc retrieval task. Wallis (1993) report improvements in the high recall area, but in the low-recall high-precision area which is of interest in standard retrieval applications they obtain worse results than the baseline. Our approach based on paraphrases falls into this category of linguistically inspired methods for document expansion. To the best of our knowledge, we are the first to use paraphrases for first story detection.

5.3.2 Using Paraphrases with LSH

While equation (5.1) introduces the similarity measure that we would like to use with paraphrases, it tells us nothing about how we would use it in our efficient LSH-based system. An obvious attempt would be to hash documents as before and then apply (5.1) to compute cosine on the set of candidates returned by LSH. Unfortunately, this would mean that LSH operates in a different space from the one we are computing the cosine

in, and this would result in many good candidates being missed. This means that the probabilistic bounds of finding a nearest neighbor would not hold any more, which is very undesirable. We thus have to transform the original document vector $\mathbf{x}$ to a new vector $\mathbf{x}'$, such that when we compute $\langle \mathbf{x}', \mathbf{y}' \rangle$ we get $\langle \mathbf{x}, \mathbf{y} \rangle_Q$. It is clear that by using:

$$\mathbf{x}' = \mathbf{Q}^{\mathbf{1/2}}\mathbf{x} \tag{5.7}$$

we have achieved our goal: $\langle \mathbf{x}', \mathbf{y}' \rangle = \mathbf{y}'^T\mathbf{x}' = (\mathbf{Q}^{\mathbf{1/2}}\mathbf{y})^T(\mathbf{Q}^{\mathbf{1/2}}\mathbf{x}) = (\mathbf{y}^T\mathbf{Q}^{\mathbf{1/2}^T})(\mathbf{Q}^{\mathbf{1/2}}\mathbf{x}) = \mathbf{y}^T\mathbf{Q}\mathbf{x} = \langle \mathbf{x}, \mathbf{y} \rangle_Q$.

We can now define the new locality-sensitive hash functions that can account for paraphrases. Recall from before that a single hash function h_{ij} in the original LSH scheme hashes the vector $\mathbf{x}$ to:

$$h(\mathbf{x}) = \text{sgn}(\mathbf{u}^T\mathbf{x}), \tag{5.8}$$

where $\mathbf{u}$ is a random vector. If we want to use paraphrases with LSH, we simply change the hash function to

$$h_1(\mathbf{x}) = \text{sgn}(\mathbf{u}^T(\mathbf{Q}^{\mathbf{1/2}}\mathbf{x})). \tag{5.9}$$

We now show that by doing this, the LSH bounds for probability of collision hold in the new inner product space defined by the matrix $\mathbf{Q}$:

$$\begin{aligned} p(h(\mathbf{Q}^{\mathbf{1/2}}\mathbf{x}) = h(\mathbf{Q}^{\mathbf{1/2}}\mathbf{y})) &= 1 - \frac{\theta(\mathbf{Q}^{\mathbf{1/2}}\mathbf{x}, \mathbf{Q}^{\mathbf{1/2}}\mathbf{y})}{\pi} \\ &= 1 - \frac{\arccos(\cos(\mathbf{Q}^{\mathbf{1/2}}\mathbf{x}, \mathbf{Q}^{\mathbf{1/2}}\mathbf{y}))}{\pi} \\ &= 1 - \frac{\arccos(\frac{\mathbf{y}^T\mathbf{Q}\mathbf{x}}{\|\mathbf{x}\|_Q\|\mathbf{y}\|_Q})}{\pi} \\ &= 1 - \frac{\arccos(\cos_Q(x,y))}{\pi} \\ &= 1 - \frac{\theta_Q(x,y)}{\pi}. \end{aligned} \tag{5.10}$$

This shows that the LSH bounds now hold in the new space defined by equation (5.1). The first equality in (5.10) comes from Charikar (2002), and the rest simply follow from basic algebra.

We now look at our approach from a more general perspective. Equation (5.1) can be viewed as defining a linear kernel, and in this case equation (5.7) represents the

mapping into the kernel's feature space. This does not depend on the kernel – it is trivial to show that by mapping each point to its image in the feature space the LSH bounds hold in the kernel space (the proof follows the same basic steps as (5.10)). Of course, performing this mapping is not feasible for many types of kernels, but in the case of a linear kernel it only introduces a small amount of overhead.

Recently, Kulis and Grauman (2009) introduced a way of hashing vectors such that the probability of two points colliding is proportional to their distance in a feature space induced by a specified kernel, i.e., they introduced a kernelized version of LSH. The main idea there is to, instead of changing the input points $\mathbf{x}$, change the way the random vectors $\mathbf{u}$ are constructed. In particular, vectors $\mathbf{u}$ are no longer sampled completely at random, but constructed as a weighted sum of t randomly sampled points. Here we will highlight the main differences between our approach and kernelized LSH as described in Kulis and Grauman (2009). The advantage of the approach in Kulis and Grauman (2009) is that it works with any kernel, and avoids the sometimes expensive explicit mappings into the feature space that we perform. However, a major disadvantage of their approach is that, because it requires a representative sample of the dataset to construct the random vectors, it only works with static datasets, making it unusable in a streaming setting. Our approach, on the other hand, naturally handles streaming data, making it possible to use kernelized LSH in problems that are inherently online in nature, such as FSD.

5.3.3 Approximating $\mathbf{Q}^{1/2}$

Unfortunately, the square root of $\mathbf{Q}$ does not integrate well with LSH. To see why, let us look at the square root of our small example matrix defined in (5.3). Its square root is given by

$$\begin{bmatrix} 0.824+0.071i & 0.592-0.123i & -0.057+0.142i & -0.115-0.123i & 0.117+0.071i \\ 0.592-0.123i & 0.766+0.213i & 0.477-0.247i & 0.059+0.213i & -0.115-0.123i \\ -0.057+0.142i & 0.477-0.247i & 0.884+0.285i & 0.477-0.247i & -0.057+0.142i \\ -0.115-0.123i & 0.059+0.213i & 0.477-0.247i & 0.766+0.213i & 0.592-0.123i \\ 0.117+0.071i & -0.115-0.123i & -0.057+0.142i & 0.592-0.123i & 0.824+0.071i \end{bmatrix} \tag{5.11}$$

We can easily see two problems that arise here: i) the entries in this matrix are complex, even though matrix $\mathbf{Q}$ had only zeroes and ones, and ii) this matrix is dense, while $\mathbf{Q}$ was sparse. These problems arise because $\mathbf{Q}$ is not positive definite, which is a consequence of non-transitivity of paraphrasing. Problem i) complicates the represen-

tation of $\mathbf{x}'$ and the operations involved (we now have to store and multiply complex numbers), and there is currently no known way of hashing complex numbers with a cosine-preserving locality-sensitive hash function. However, problem ii) is even more serious because having a dense $\mathbf{Q^{1/2}}$ effectively means that we are expanding every document with every word in the vocabulary. This would increase the size of documents by several orders of magnitude, rendering our approach infeasible on all but toy datasets. Thus, we have to find a way to approximate $\mathbf{Q}$ by a positive definite matrix, which is equivalent to finding an approximation of $\mathbf{Q^{1/2}}$ with all real entries. To this end, we introduce the following approximation:

$$\tilde{\mathbf{Q}}^{\mathbf{1/2}}_{ij} = \frac{\mathbf{Q}_{ij}}{\sqrt{\sum_k (\mathbf{Q}_{ik} + \mathbf{Q}_{kj})/2}}. \tag{5.12}$$

To understand the intuition behind the approximation (5.12), we first start with computing the square root of a block diagonal matrix. First, let us define the direct sum of two matrices, $\mathbf{A}$ (of size $m \times n$) and $\mathbf{B}$ (of size $p \times q$), as

$$\mathbf{A} \oplus \mathbf{B} = \begin{bmatrix} a_{11} & \dots & a_{1n} & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} & 0 & \dots & 0 \\ 0 & \dots & 0 & b_{11} & \dots & b_{1q} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & b_{p1} & \dots & b_{pq} \end{bmatrix}. \tag{5.13}$$

We can then write the block diagonal matrix $\mathbf{Q}$ as a direct sum of n smaller matrices $\mathbf{Q}^k$, called *blocks*:

$$\mathbf{Q} = \bigoplus_{k=1}^{n} \mathbf{Q}^k. \tag{5.14}$$

Such a matrix thus has the following form:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}^1 & 0 & \dots & 0 \\ 0 & \mathbf{Q}^2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \mathbf{Q}^n \end{bmatrix} \tag{5.15}$$

where all $\mathbf{Q}^k$ are non-zero square matrices. If $\mathbf{Q}$ is a binary matrix, as is the case

here, the blocks $\mathbf{Q}^k$ would be matrices of all ones:

$$\mathbf{Q}^k = \begin{bmatrix} 1 & \dots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \dots & 1 \end{bmatrix} \tag{5.16}$$

Because $\mathbf{Q}$ is a paraphrase matrix, each block in this matrix would correspond to a cluster of words where any one of these words can substitute any other.

Finding the square root of each $\mathbf{Q}^k$ now becomes very easy. We are looking for a matrix $\mathbf{B}$ such that $\mathbf{B}^T\mathbf{B} = \mathbf{Q}^k$. This can be written as a system of linear equations:

$$q^k_{ij} = \sum_{m=1}^{n_k} b_{im}b_{mj} = 1, \forall i, j = 1, \dots, n_k, \tag{5.17}$$

where n_k is the dimension of $\mathbf{Q}^k$. One possible solution to (5.17) (although not the only one) is to have all b_{ij}'s equal:

$$b_{ij} = \frac{1}{\sqrt{n_k}}, \forall i, j = 1, \dots, n_k. \tag{5.18}$$

Unfortunately, it is not clear which n_k to use here because paraphrasing matrices do not have block diagonal form as in (5.15). The reason for this is simple – paraphrasing is not a transitive relation. For example, words *jump* and *spring* are paraphrases, and *spring* and *source* are paraphrases, but *jump* and *source* are not paraphrases. However, if, for example, the words *spring* and *source* were not paraphrases, and *source* and *witness* were, our example matrix would have the block diagonal form as in (5.15):

	jump	*spring*	*source*	*informant*	*witness*
jump	1	1	0	0	0
spring	1	1	0	0	0
source	0	0	1	1	1
informant	0	0	1	1	1
witness	0	0	1	1	1

The main problem now becomes how to transform matrix $\mathbf{Q}$ into a block diagonal matrix. If we try to simply form a transitive closure from $\mathbf{Q}$,[3] we would introduce a lot of paraphrase relations that do not hold. In our example matrix, a transitive closure over the relation defined in the matrix would mean that $\mathbf{Q}$ would become a matrix of

[3]This means that we force the relation expressed by the matrix to be transitive by adding pairs of items to the relation until the transitivity property holds.

all ones, introducing six new paraphrase pairs. For larger matrices, this would be even worse and in the end the majority of paraphrase relations in the approximation would not have been present in the original matrix.

This is why we suggest forming *local blocks* which will be much smaller and introduce fewer paraphrase relations that are not present in the original paraphrase matrix. This approach works as follows. For each non-zero element q_{ij} in $\mathbf{Q}$, there are two ways we can try to make it part of a block. One way is to take all the words that word i can be paraphrased to, and do two things: i) form a transitive closure over those words, which makes sure that the block that q_{ij} is a part of has all ones, and ii) make sure that paraphrases of word i can only be paraphrased to other paraphrases of i. For example, this means that if we look at word *jump* in our example matrix, we cannot have the word *spring* paraphrased to *source*, as *source* is not itself a paraphrase of *jump*. Another way is to perform the same procedure, only for the word j. Because it is not clear that either of the two ways should be preferred, we use an approximation for n_k that interpolates between them. In particular, n_k now depends on the b_{ij} that we are trying to compute (cf. equation (5.18)):

$$n_k^{ij} = \sum_{l=1}^{n} (q_{il}^i + q_{lj}^j)/2, \forall i, j = 1, \ldots, n, \tag{5.19}$$

where q_{il}^i and q_{lj}^j are entries in $\mathbf{Q}$ when we make the local approximations based on words i and j, respectively. When $\mathbf{Q}$ is a block diagonal matrix, $q_{il}^i = q_{il}$ and $q_{lj}^j = q_{lj}$, and hence n_k given by (5.19) is exact. Finally, the square root of $\mathbf{Q}$ is now just a direct sum of the square roots of the diagonal matrices:

$$\mathbf{Q}^{1/2} = \bigoplus_{k=1}^{n} (\mathbf{Q}^k)^{1/2}, \tag{5.20}$$

which is just another way of writing (5.12).

It is easy to verify that our approximation solves both problems that arise when using exact $\mathbf{Q^{1/2}}$: i) all the entries in the matrix are real (a direct consequence of equation (5.12) and the fact that entries in $\mathbf{Q}$ are zeroes and ones), and ii) $\tilde{\mathbf{Q}}^{\mathbf{1/2}}$ is as sparse as $\mathbf{Q}$, which can be seen by noting that whenever $q_{ij} = 0$, $\tilde{\mathbf{Q}}_{ij}^{\mathbf{1/2}}$ is also zero because of (5.12). Another advantage of using our approximation is that it is very simple and, with proper implementation, takes $O(n^2)$ time to compute, as opposed to $O(n^3)$ for $\mathbf{Q^{1/2}}$, making it scalable to very large matrices. Furthermore, for sparse matrices, such as those that we are dealing with, computing this approximation is even faster and takes time linear in the number of non-zero elements in the matrix.

We will take a moment here to compare this approach to that of Ture et al. (2011), which was concerned with using LSH to efficiently find similar documents across two languages. Their approach is similar to ours in that it uses a matrix (in their case a translation matrix) to define a new inner product space in which cosine is computed. The main difference, however, lies in the input documents: Ture et al. (2011) have two types of documents, ones in the foreign language, and ones in the target language. This asymmetry allows them to hash the documents in the target language as normal, and hash the documents in the foreign language using

$$h(\mathbf{x}) = \mathbf{u}^T(\mathbf{Q}\mathbf{x}). \tag{5.21}$$

In our case there is no such asymmetry, which is why we have to use $\mathbf{Q^{1/2}}$ as in equation (5.7). As we have already explained, the main problem lies in the fact that exact $\mathbf{Q^{1/2}}$ cannot be used and we have to make approximations, and this issue does not arise in Ture et al. (2011).

Space efficient LSH. While LSH can significantly reduce the running time, it is fairly expensive memory-wise. This memory overhead is due to the random vectors $\mathbf{u}$ being very large. To solve this problem, van Durme and Lall (2010) used a *hashing trick* for space-efficient storage of these vectors. They showed that it is possible to project the vectors onto a much smaller random subspace, while still retaining good properties of LSH. They proposed the following hash function for a vector $\mathbf{x}$:

$$h_2(\mathbf{x}) = \text{sgn}(\mathbf{u}^T(\mathbf{A}\mathbf{x})), \tag{5.22}$$

where $\mathbf{A}$ is a random binary matrix with exactly one non-zero element in each column. This approach guarantees a constant space use which is bounded by the number of rows in the $\mathbf{A}$ matrix. Here we show that our paraphrasing approach can be easily used together with this space-saving approach by defining the following hash function:

$$h_3(\mathbf{x}) = \text{sgn}(\mathbf{u}^T(\mathbf{A}\tilde{\mathbf{Q}}^{\mathbf{1/2}}\mathbf{x})). \tag{5.23}$$

The above equation simply states that, instead of hashing original documents $\mathbf{x}$, we hash the expanded documents $\tilde{\mathbf{Q}}^{\mathbf{1/2}}\mathbf{x}$. This way we get the benefits of the hashing trick (constant space use for random vectors), while also being able to use paraphrases.

5.3.4 Where Do Paraphrases Come From?

Until now we said nothing about where we get the matrix $\mathbf{Q}$ from. In fact, research on ways of extracting paraphrases from text is a challenging task in its own right, and

here we simply use the resources provided by researchers in this field. In particular, we use three sources of paraphrases: Wordnet (Fellbaum, 1998), a carefully curated lexical database of English, Microsoft Research paraphrase tables (Quirk et al., 2004), a set of paraphrase pairs automatically extracted from news texts, and syntactically-constrained paraphrases from Callison-Burch (2008) which are extracted from parallel text. For all three corpora, we remove any paraphrase pairs that contain at least one stopword. We also considered using paraphrases from Cohn et al. (2008), but using them provided only minor improvement over the baseline model. This is likely due to the small size of that corpus (a total of 7 thousand pairs). We do not show results for this paraphrase corpus in our results. All three resources we use are very different: they come from different domains (news text, legal text, general English), and they have very little overlap (less than 5% of pairs are shared between any two resources).

Wordnet. We use Wordnet's synonym sets (synsets) as a source of paraphrases by considering all the pairs of words in a synset to be possible paraphrases of each other. By doing this, we obtain 42 thousand adjective pairs, 4 thousand adverb pairs, 29 thousand verb pairs, and 80 thousand noun pairs. Note that by using Wordnet's synsets as a source of paraphrase data, we in effect reduce the more general definition of a paraphrase to a simple synonym relation.

MSR paraphrases. Because Wordnet is a very expensive linguistic resource unavailable in most languages, we want to know how using an automatically obtained paraphrase corpus compares to using Wordnet. The MSR paraphrase tables[4] we use were obtained automatically from clusters of news articles on the same topic, where the paraphrase pairs were extracted using an approach based on machine translation. For each phrase pair, the corpus contains the most likely alignment of words in the pair along with the forward and backward probability of alignment. We use these phrase pairs to extract word pairs by retaining only the aligned words where the probability of both forward and backward alignment was greater than 0.2. In our preliminary experiments we varied this threshold and found it has little effect on results. Using this method, we extracted just over 50 thousand paraphrase pairs.

Syntactic paraphrases. Finally, we use the method of Callison-Burch (2008) to extract syntactically constrained paraphrases from a parallel corpus. This method requires that phrases and their paraphrases to be of the same syntactic type, and has been shown to substantially improve the quality of extracted paraphrases (Callison-Burch,

[4]Available from http://research.microsoft.com/en-us/downloads/eceb4aba-f3a9-4535-9a07-95959611f613/

2008). We extracted paraphrases for all the words that appeared in the MSR paraphrase corpus, and then kept all the pairs that had the paraphrase probability of at least 0.2. This way, we extracted 48 thousand pairs.

Generalizing our approach to multi-word units. Although in our experiments we focus on single words, our approach can be easily generalized to multi-word paraphrases, i.e., to phrasal paraphrases like *(get married, tie the knot)*. To see how, simply note that the dimensions in our matrix $\mathbf{Q}$ will no longer be single words, but entire phrases, much like in a phrase table in machine translation. This will, of course, require the document representation to change from bag-of-words to bag-of-ngrams, making the vectors much longer and thus making our approach less efficient. A simple way to get around this is to, instead of keeping all the n-grams in a document, keep only those that have a paraphrase. This will reduce the total number of possible n-grams by several orders of magnitude, thus incurring only a small computational overhead for using phrasal paraphrases.

5.4 Experiments

5.4.1 Efficacy of Query Expansion for First Story Detection

5.4.1.1 TDT Results

Much like in Chapter 4, we run experiments on both TDT and Twitter data to test the efficiency and effectiveness of our approach. All the experiments are conducted on the same data as in Chapter 4. The default parameters for our streaming FSD system are $k = 13$, $L = 70$, $b_t = 0.6$, and $n = 2000$ for both TDT and Twitter.

In addition to comparing our approach to a state-of-the-art system, we also compare it to relevance models (RMs), which are a state-of-the-art query expansion method in tasks like ad-hoc retrieval, and have also been shown to improve results in other TDT tasks. We tune the parameters of relevance models on the TDT1 corpus (Allan et al., 1998). We expand each document in the TDT1 corpus using all the documents from the same corpus, and then run our FSD system on the expanded documents. For relevance models, we use the RM3 variant (relevance models interpolated with original query) which performed the best on the linking task (Lavrenko et al., 2002). We run a grid search over the parameter values and pick the setting that gave us the best $C_{\min}$ score. The particular setting we use is ce = 0.1, rew = 20, perp = 500. We then use these parameters to expand TDT5 documents in the same way – we expand each document

from TDT5 using all other documents from TDT5 and run the FSD system on the expanded documents. Note that by doing this we are in fact ignoring the streaming nature of the data as we are effectively using documents from both past and future when expanding each document. This is of course a serious caveat in this experiment, and we note here that the results we obtain this way are not the results we would expect to obtain from using RMs in a realistic scenario. Rather, we use this as a way of getting an upper bound on performance of RMs in FSD. It is reasonable to assume that respecting the stream order or using a different collection of documents to expand (e.g., Gigaword) will only perform worse than this approach. This should be kept in mind when interpreting the results. We expand each document to the length of 1000 words, which was roughly the same as what expansion using paraphrases achieved. This is the first time that relevance models have been used in first story detection.

Table 5.1 shows the results for TDT5 data. UMass 1000 is the run that was submitted as the official run in the TDT competition.[5] The best supervised system in Table 5.1 is the highest reported score in literature on the TDT5 dataset, described in Kumaran and Allan (2005). Note that this is a supervised system trained on TDT3 and TDT4 corpora, whereas our system is fully unsupervised and requires no training data.

It is clear that using document expansion is beneficial, as all the systems that used it outperform both supervised and unsupervised state-of-the-art systems. Furthermore, we can see that using paraphrases performs better than relevance models. In fact, the automatically extracted paraphrases from Callison-Burch (2008) outperform relevance models by a large margin. This is a very promising result not only because we outperform relevance models, but also because these paraphrases were extracted automatically, which means that we do not have to rely on expensive hand-crafted resources like Wordnet as our source of paraphrases.

We also compare our approach to the hybrid clustering and topic model approach from Ahmed et al. (2011). We perform the comparison on a subset of the TDT5 corpus which consists of 46,793 articles published in May 2003. The same subset was used in Ahmed et al. (2011), and because their system is not available, using this subset of TDT5 was the only option in order to perform a fair comparison. In Table 5.2 we compare the UMass system and the system from Ahmed et al. (2011) to our system that uses syntactic paraphrases. We can see that the topic model approach performs substantially worse than the baseline UMass system. This just confirms what years

[5] Our experiments, and experiments in Allan et al. (2000b) showed that keeping full documents does not improve results, while increasing the running time.

System	C_{min}	Running time (hours)
UMass 100	0.721	13.6
UMass 1000	0.706	46.3
LSH-based approach	0.713	2.3
Kumaran and Allan (2005)	0.661	n/a
Relevance models	0.655	37.6
Paraphrases Wordnet	0.657	6.8
Paraphrases MSR	0.642	10.5
Paraphrases syntactic	0.575	7.8

Table 5.1: TDT5 results when using paraphrases, lower is better. The number next to UMass system indicates the number of features kept for each document (selected according to their TFIDF). All paraphrasing systems work with top 300 documents. Running time for Kumaran and Allan (2005) was unavailable but is lower bounded by the running time of the UMass system.

System	C_{min}
Ahmed et al. (2011)	0.714
UMass	0.571
Paraphrases syntactic	**0.499**

Table 5.2: Comparison with Ahmed et al. (2011) on the May subset of TDT5.

of research in FSD have shown – a simple approach based on 1-NN clustering and cosine distance makes for a very competitive baseline. As expected given the numbers in Table 5.1, our system that uses paraphrases outperforms both systems.

Finally, we look at the distribution of first and non-first stories as a function of the novelty score output by the systems. Ideally, a good FSD system would assign first stories a high and non-first stories a low novelty score. Figures 5.1 and 5.2 show this distribution for an approach that does not use paraphrases (UMass) and our approach that does. We can see that we have achieved a similar effect as the approach of Kumaran and Allan (2005). We have improved the detection of non-first (old) stories, which can be seen by the larger left skew of the distribution of non-first stories, compared to the UMass system. This is exactly what we hoped using paraphrases would achieve – identifying documents previously thought to be new as being about an exist-

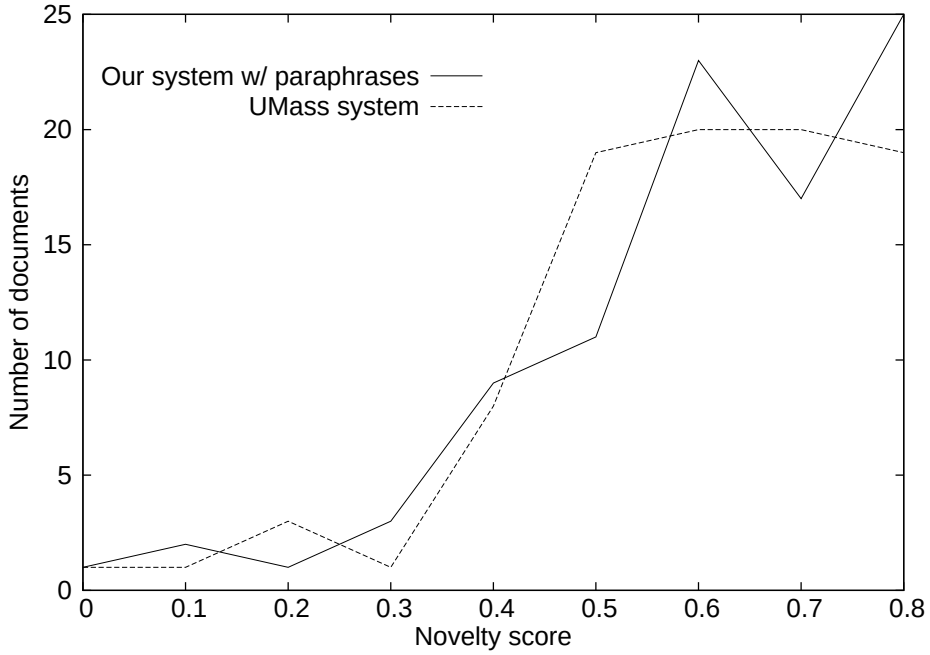

Figure 5.1: Distribution of first stories with and without paraphrasing.

ing event, only written using different words. Note that there is no such effect in the distribution of first stories – while a few first stories did receive lower novelty scores, the distribution as a whole did not skew towards the lower novelty scores.

5.4.1.2 Twitter Results

For Twitter, we compare four systems: the baseline system that does not use paraphrases and three systems that use three different sources of paraphrases, like in the previous section. Note that we do not compare to relevance models for two main reasons. First, because using relevance models would take too long in this case. Expanding each tweet with all the tweets in the corpus, like we did for TDT, would mean expanding 50 million documents using a database of a couple of million documents (depending on which database we would use for expansion). We tried this approach and found that expanding the tweets using English articles from the Gigaword corpus (a total of about 1.6 million articles) took on average 1.8 seconds per tweet, whereas expanding using a sample of 8 million tweets from May 2011 (time period just before our corpus) took on average 2 seconds per tweet. A simple calculation suggests that expanding the whole dataset would take 25,000 hours, or almost three years, if we

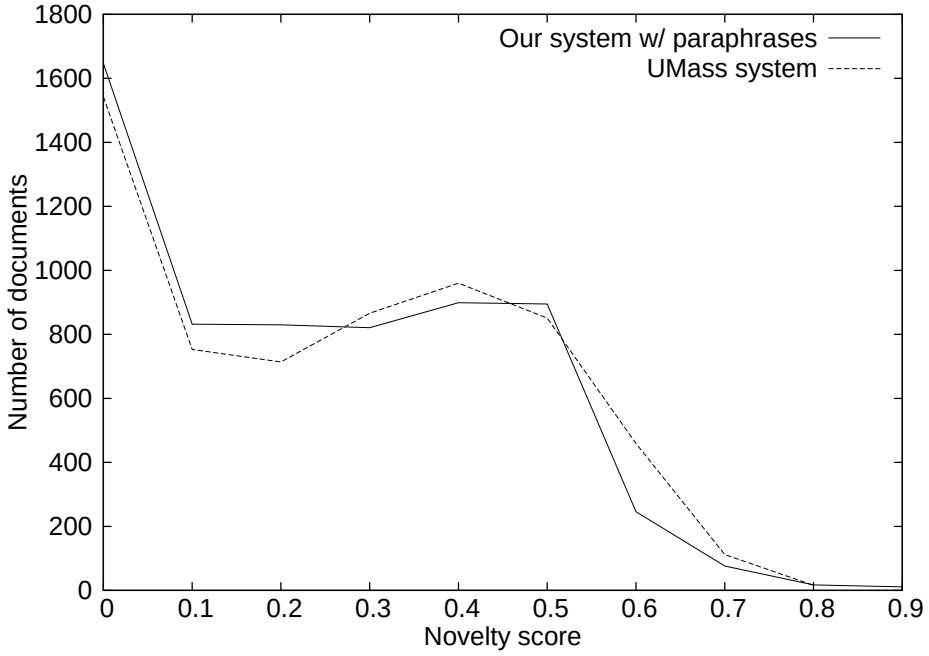

Figure 5.2: Distribution of non-first stories with and without paraphrasing.

used Gigaword, and expanding using the Twitter sample would take over three years on a single machine. Clearly, this is not feasible even if we were to parallelize the approach – in order to perform the expansion in, say, one week, we would have to use 165 machines, which is much more than the amount of processing power that is normally available to academic researchers.

Second, given the very different nature of news and Twitter (e.g., length of documents, different domain), it is very likely that the parameters we used for TDT would not work well on Twitter, and we have no comparable data on which we could tune these parameters for Twitter. This shows another advantage of our use of paraphrases – we have no additional parameters to set and we can use paraphrases in the same way as we did for TDT.

Despite not being able to use relevance models here, we note that related work in this area suggests that they would not be very useful. In particular, Metzler et al. (2012) used relevance models to expand queries in an event retrieval task using Twitter data, and found that they did not outperform a baseline that does not expand the queries. Given that they work in the same domain (Twitter), and that the queries are event-related, this suggests that using relevance models here would not result in any

significant gains.

System	C_{min}
Baseline	0.694
Wordnet	**0.679**
MSR Paraphrases	0.739
Syntactic paraphrases	0.729

Table 5.3: Twitter results when using paraphrases, lower is better.

Twitter results are shown in Table 5.3. We can see that the results here are mixed. Syntactic paraphrases and the MSR paraphrases do not help, whereas the paraphrases extracted from Wordnet did improve the results, although the gains are not as large as in TDT. It is hard to make statements about the suitability of any paraphrase set for either task, but it appears that the MSR and syntactic paraphrases work better on TDT because they come from a similar domain (news and legal text), whereas Wordnet works better on Twitter. In Section 5.4.3 we will investigate in more depth the possible reasons as to why the results are different in Twitter.

5.4.1.3 How does quality of paraphrases affect results?

We have shown that using automatically obtained paraphrases to expand documents is beneficial for first story detection. Because there are different ways of extracting paraphrases, some of which are targeted more towards recall, and some towards precision, we want to know which techniques would be more suitable to extract paraphrases for use in FSD. In this context, precision is the ratio between extracted word pairs that are actual paraphrases and all the word pairs extracted, and recall is the ratio between extracted word pairs that are actual paraphrases, and all the possible paraphrase pairs that could have been extracted. In this experiment we focus on the syntactic paraphrases which yielded the best results and test how lowering precision and recall affects the results. To lower recall, we randomly remove paraphrase pairs from the corpus, and to lower precision, we add random paraphrase pairs to our table. All the results are shown in Figure 5.3. Numbers next to precision and recall indicate the proportion of added random pairs and the proportion of removed pairs, respectively. For example, recall of 0.4 means that 40% of pairs were removed from the original resource, and a precision of 0.4 means that we have added 40% of random word pairs to our paraphrase table.

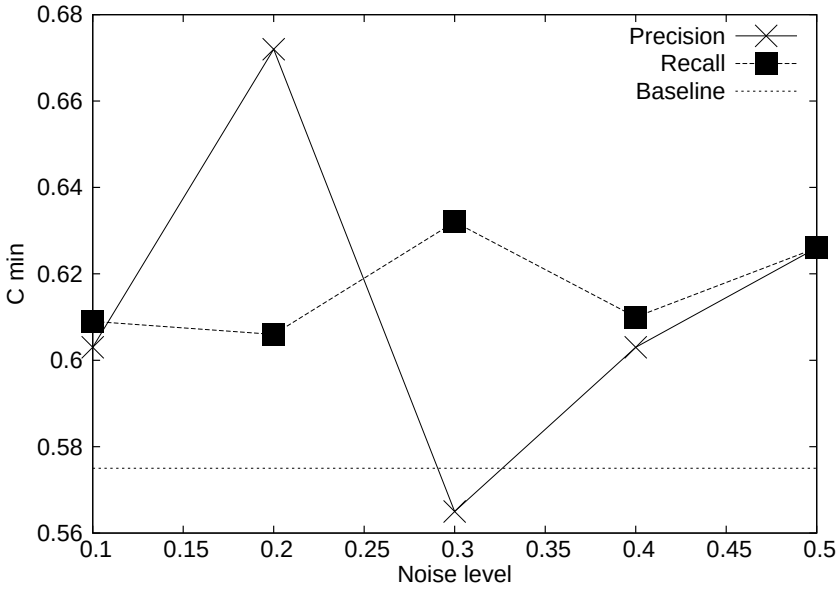

Figure 5.3: Effect of precision and recall of paraphrases on TDT5 FSD performance. Numbers next to recall and precision indicate the sampling rate and the proportion of added random pairs, respectively.

We can see from the figure that the results are much more stable with respect to recall – there is an initial drop in performance when we remove the first 10% of paraphrases, but after that removing more paraphrases does not affect performance very much. On the other hand, changing the precision has a bigger impact on the results. For example, we can see that our system using a paraphrase corpus with 30% of pairs added at random performs even slightly better than the system that uses the original corpus. On the other hand, adding 20% of random pairs performs substantially worse than the original corpus. These results show that it is more important for the paraphrases to have good precision than to have good recall. This is a desirable property of our approach, as most modern paraphrase extraction systems are targeted towards maximizing precision.

5.4.2 Efficiency

Having explored the effectiveness of our approach, we turn to measuring its efficiency. As we mentioned before, our approach that uses paraphrases has constant per-document time complexity, but in practice the documents we deal with have more

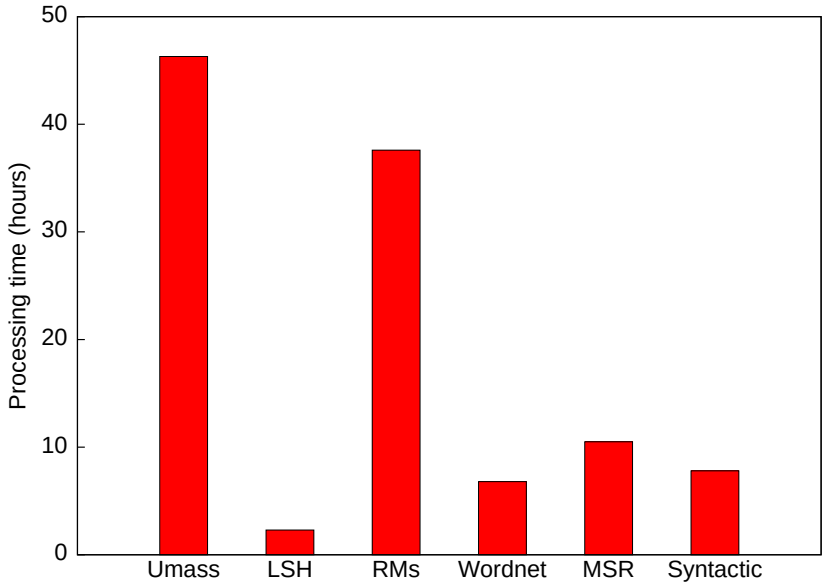

Figure 5.4: Time required to process the TDT5 collection for different systems.

features and therefore the constant will be larger. We thus compare the running time of our system that uses paraphrases against the running time of the baseline approach that does not use any paraphrase information, as described in Chapter 4. Additionally, we also compare against the system that uses relevance models. We only compare the running time on TDT5 data because, as we previously discussed, relevance models did not scale to Twitter data. We can see from Figure 5.4 that using paraphrases is, as expected, slower than the vanilla approach from Chapter 4, but is still much faster than the UMass system. The paraphrasing approach is also faster than using relevance models (about five-fold). It is interesting to note that about two thirds of running time for the system that used relevance models was spent on expanding the documents, and only one third on processing them in the FSD system. This shows just how efficient our expansion method really is, compared to state-of-the-art query expansion methods like relevance models.

5.4.3 Exploring the Use of Paraphrases in Twitter

In this section, we explore possible reasons why using paraphrases did not yield the same level of improvement as it did in newswire data. We conjecture that there are

FLASH: S&P **downgrades** U.S. credit **rating** to AA+ with negative outlook
#S&P **lowers** US credit **score**. Shld a comp who mis-valued mortgage securities allegedly on purpose to help banks be able t...

Table 5.4: Example of the paraphrase coverage problem in Twitter.

Paraphrases	Coverage TDT	Coverage Twitter
Wordnet	52.5	56.1
MSR	33.5	31.0
Syntactic	35.6	31.7

Table 5.5: Coverage results for TDT and Twitter.

two main reasons for this: coverage and length of tweets. We perform experiments to confirm this.

Coverage. The sources of paraphrases that we use either come from general English (Wordnet) or newswire (MSR and syntactic paraphrases). This means that there are a lot of words in the Twitter data that possibly have paraphrases, but this information is not present in our paraphrase tables. Table 5.4 shows an example of two tweets that talk about the same event, with paraphrases bolded. Pairs (*downgrade*, *lower*) and (*rating*, *score*) are paraphrases, but none of our three sources contain either of the two.

To investigate the extent of this problem, we measure the proportion of words in the documents that have at least one paraphrase in our data. We call this proportion *coverage*. Results for the three corpora and for TDT5 and Twitter are shown in Table 5.5. We can see that for MSR and syntactic paraphrases, the coverage is lower in Twitter than in TDT. On the other hand, the coverage for Wordnet is actually higher for Twitter, which is interesting if we recall that Wordnet was the only resource that improved performance in Twitter. This suggests that the problem of coverage is at least partially responsible for poorer results in Twitter.

Finally, there are also cases when paraphrases can hurt performance by making two documents that are about different topics seem similar. We show an example of this in Table 5.6. Before paraphrasing, tweet one was most similar to tweet two, which is reasonable because both seem to discuss the debt ceiling deal. After paraphrasing, however, tweet three became most similar to tweet one because the word *roof* is a paraphrase of the word *ceiling*. Clearly, this is not desirable because tweets one and

1	Top news #debt #deal #ceiling #party
2	New debt ceiling deal explained
3	The roof the roof the roof is on fire!

Table 5.6: Examples of tweets where paraphrasing words hurts performance.

Nairobi oil terminal explodes near sinai slum... many feared dead
Police close Lunga Lunga road, Industrial area after a huge fire outbreak. So far, 15 people rushed to Mater Hospita...
NASA scientists have found evidence of what they believe to be flowing water on Mars. WATCH LIVE
7 confirmed sites of warm season flow w/ some sort of volatile material, i.e., water in mid latitudes of #Mars' soute...

Table 5.7: Examples of tweets where paraphrasing does not help.

three do not discuss the same topic. This is also an example of the difference between the paraphrasing resources – this problem only occurs when Wordnet or syntactic paraphrases are used, but not with MSR paraphrases as that resource does not contain the *(roof, ceiling)* pair.

Length of Twitter documents. The other major reason which we believe is limiting the usefulness of paraphrases in Twitter is the fact that tweets are very short. Table 5.7 shows examples of tweets that are about the same events, but where there are no obvious paraphrase pairs. In particular, the first pair of tweets have no words in common because they talk about the same event from different perspectives. The length of the tweets is the limiting factor here – given a limited amount of space, authors of tweets have to decide which information about the event to preserve and what to discard. It is not surprising that different people decide on preserving different aspects of the events. On the other hand, authors of newswire articles have no such limits on the space (or, rather, their limits are not as strict) and can thus preserve more information about events, thus allowing for more possibilities of finding paraphrases.

To test whether length is indeed a limiting factor here, we reproduce similar conditions in the TDT5 data. We reduce each document in the TDT5 collection to the top n terms, where the terms are scored according to their TFIDF score for the document, and then use paraphrases to expand only these top n terms. This way, documents are

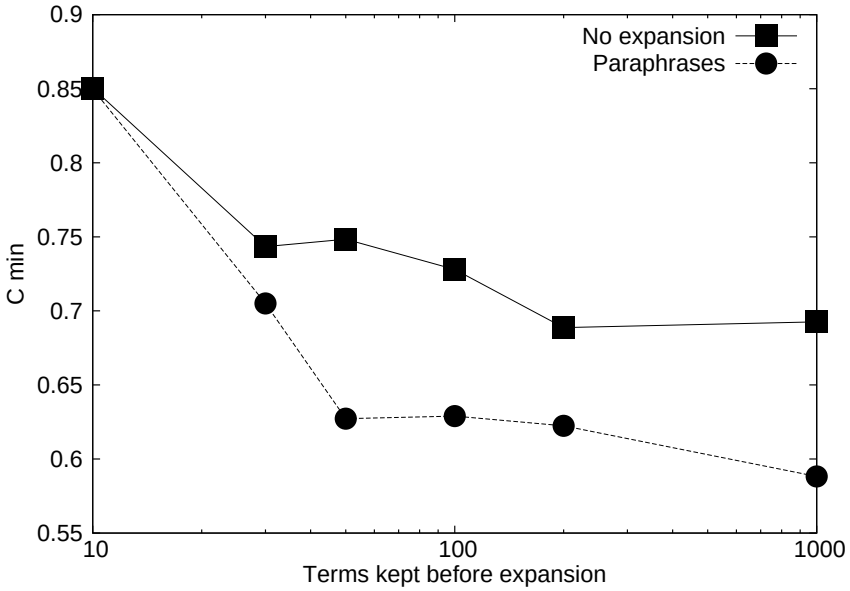

Figure 5.5: Effect of document length on document expansion.

reduced to only the "essential" terms, similar to Twitter. We measure the $C_{\min}$ score for the baseline approach (using only the top n terms) and for the approach that expands using paraphrases. For paraphrases, we only test the syntactic paraphrases because they performed best in previous experiments. Results are shown in Figure 5.5. If we compare the curve that does not use expansion with the curve that does, we can see that the benefits of using expansion only start to show when documents are about 50 words in length or more. When documents are reduced to 10 words, which is the length of an average tweet, we see that there is no benefit in doing expansion. This explains why we did not see any substantial improvement from using paraphrases in Twitter.

5.5 Conclusion

Lexical variation, also known as vocabulary mismatch, is a serious problem in almost all NLP and IR applications. In this chapter we introduced an approach based on the use of paraphrases that is capable of mitigating this problem, while at the same time integrating well with the LSH framework. This means that our approach is efficient and is suitable for use in streaming applications. While we evaluate the benefits of our approach on the first story detection task, we emphasize here that our approach is not

specific to the problem of FSD – we presented a general technique for fast retrieval of documents that may talk about the same topic, but using different words. In first story detection in newswire, we showed that our approach is beneficial and achieves state-of-the-art results on the task, while at the same time maintaining constant per-document time complexity. Results on Twitter data suggest that there are gains to be had using this approach, but that they are fairly small. To better understand what causes the disparity in results between news and Twitter, we performed an analysis and found that the two main reasons for this are poor coverage of paraphrases in Twitter data and the brevity of tweets. In the next chapter, we will address the problem of noise in the data that comes from Twitter. This is another problem that was not present in the original formulation of the FSD task, but which any modern approach to event detection should address.

Chapter 6

Combining Novelty with Latency

We mentioned in the introduction that there are two main problems – scale and noise – when dealing with social media streams such as Twitter. While we focused on the problem of scale in Chapter 4, in this chapter we address the issue of noise. The presence of noise in Twitter (and social media in general) is an important and well-known problem. For Twitter specifically, it is estimated that anywhere between 1.5% (Twitter, 2010) and 14% (Yardi et al., 2009) of all tweets are spam. However, when it comes to event detection, spam is not the only source of noise. For our purposes, noise is anything that is not event-related or is not of interest to a large number of people. This is of course a subjective criterion, and it is sometimes hard to say if something is interesting enough to be considered an event in this sense. However, in most cases, it is fairly clear if something should be considered noise or not. For example, noise in this context is spam, but also all the trivial status updates (*I'm cooking bacon for breakfast, yay!*) or casual conversation between users (*@friend Did you know she went out with him last night?*). When this is taken into account, it turns out that a large majority of tweets are noise, and only a handful are related to events. In fact, a study in 2009 showed that only 3.6% of all tweets are news-related (Analytics, 2009), which would mean that, for our purposes, 96.4% of tweets are noise. In this chapter we show that not much has changed since 2009 and that less than 1% of the first stories detected by our approach are related to actual events. Clearly, this is a major problem and shows that just having a system that can detect novelty (i.e., first stories) is not enough for a useful event detection system, as over 99% of reported events, while being novel, would not be interesting.

In this chapter we address the problem of noise by combining our novelty-based approach with the approach based on burstiness that is frequently used in event detec-

tion literature. We achieve this by trading off latency in detecting events for improving their quality. A non-zero latency allows the system to wait for a given amount of time before making the decision about whether something is an event. Because noise was not an issue in the original work on TDT, we have to construct a new way of evaluating event detection systems that accounts for noise. To the best of our knowledge, we are also the first to explore the relationship between latency, burstiness, and their effect on precision and recall in event detection.

6.1 Clustering with Latency

Up until now we were only concerned with detecting the first story that reports a new event. However, when there is a large amount of noise in the data, such as is the case with Twitter, most of the "new events" that are detected are not interesting. To show this, we perform a simple experiment. We run our FSD system on the Twitter data, and select all of the detected first stories (we used a threshold of 0.6 as a cutoff), which gives us a set of 35.5 million stories. We randomly sample 100 stories from that set and manually label each one as being an actual event or not. Out of the 100 randomly selected stories, not a single one was labeled as an actual event. This means that less than 1% of the first stories detected by our system are about actual events. We show examples of the non-event related first stories in Table 6.1. We increased the novelty threshold to 0.9 and performed the same experiment, i.e., labeled 100 randomly picked tweets that satisfy this condition. Increasing the novelty threshold did not change the results, and again not a single story was labeled as an event. This can be explained by the fact that tweets with such a high novelty score mostly contain misspellings or extremely rare hashtags. Combined with the fact that none of the first stories in our dataset obtained a novelty score higher than 0.8, this leads to the conclusion that true first stories in Twitter have moderately high novelty scores and Twitter stories with a very high novelty score are mostly noise. This is in contrast with TDT5, where 15% of first stories got novelty scores over 0.8, and shows just how much more challenging this domain is.

This example shows that simply detecting first stories is not enough in the presence of such a large amount of noise. In many real scenarios, however, we can allow for some latency in detecting the events (say, a couple of minutes), if this would mean that the quality of detected events would substantially improve. This is why here we explore the connection between detection quality and latency.

Juicy Couture, Ed Hardy, Coach, Kate Spade and many more! Stay tuned for more brands coming in http://...
i lovee my nephew hair :D
Going to look at houses tomorrow. One of them is right behind Sonic & Taco Casa. If I live there, I might weigh 400 lbs within a year.
Hope a bad morning doesnt turn into a bad day...

Table 6.1: Examples of tweets that are novel (i.e., detected as first stories), but not related to events.

We now describe how the extra time (latency) that is given to the system can be used to improve event detection. The main idea is to form clusters of documents that talk about the same topic, which is essentially the same as doing the TDT detection task (cf. Section 2.1.1.3). When performing clustering, latency is used to constrain how long we are allowed to wait after forming a new cluster before having to make a decision if the cluster is about an actual event. This effectively adds a postprocessing step in which, given a cluster, the system has to decide if the cluster is about an actual event or not. There are a number of ways in which clusters can be used to improve the quality of detected events. For example, a simple strategy would be to use the size of clusters and keep only those whose size exceeds some threshold. In TDT, this postprocessing step was not necessary because all the documents were "newsworthy" and thus all the clusters were reported. However, in Twitter, as we just showed, most clusters should not be reported, and the postprocessing step is essential if we want to get reasonable output from the system.

A related concept was previously explored in the context of TDT, known as the *deferral period*. In particular, a system with a deferral period of say, ten units, could decide if a story is a first story after being presented with ten stories that follow it. A major disadvantage of defining the deferral period in terms of the number of documents is the fact that it is disconnected from real time, becoming stream-specific. In TDT5, a deferral period of 100 documents would correspond to roughly one and a half hours, while in Twitter 100 documents corresponds to a fraction of a second. This means that comparing deferrals across streams becomes very hard and we therefore advocate here defining *latency* in terms of real time. This will enable us to compare latencies across different streams and also make our results much more interpretable. It is important here to make a distinction between the deferral period used in TDT and the latency that

we use. The deferral period in TDT was used to improve the accuracy of the system for a given task (FSD, detection, tracking, etc.), while here we use latency to reduce the noise in the detected events. One way to think about this is the following: given a perfect detection system, i.e., one that with 100% accuracy clusters stories into topics, most of the clusters it would produce from the Twitter stream would still be noise. The deferral period in TDT was used to push the accuracy of systems towards these 100%, but latency as we use it here is used to deal with the output of a detection system, i.e., with the clusters produced.

The clustering algorithm that we use is essentially the same as that used in the UMass system (Allan et al., 2000b), with the added limit on the lifetime of clusters (explained below). This is an online single-link clustering approach that integrates well with our nearest neighbor-based approach to solving FSD. All the input that this algorithm requires is, for each document, its nearest neighbor and the distance to that neighbor, which is exactly what our FSD approach computes. This makes it very inexpensive to perform the clustering, as no additional information needs to be computed. On the other hand, agglomerative clustering approaches such as that used in Yang et al. (1998) and Becker et al. (2011a) have to find the nearest cluster for every document, where clusters are represented by centroids. This increases the time complexity of their algorithm to $O(m)$, where m is the number of clusters, whereas our approach is $O(1)$. Obviously, by keeping a constant number of clusters the agglomerative approaches can also be made constant, but this introduces additional complexity (how to decide how many and which clusters to keep?), and the constant is much larger than in our case.

We summarize our online single-link clustering with latency in Algorithm 9. This approach can be seen as performing standard single-link clustering, but a document can only be assigned to a cluster if it arrived within some time window from the oldest document in that cluster. If the document arrived outside of this allowed time window, it is simply discarded. This ensures that, once we detect a first story, we make a decision if this story is about an actual event within the allowed latency. After we make a decision, we can delete the cluster as no other document can be assigned to it. Note that computing the nearest neighbor and the distance to it, shown in steps 3 and 4, is actually performed by our FSD system, but we do not show this in the pseudocode for simplicity. We emphasize here that we think of document clusters produced by our system as events, so in the rest of the thesis we will use the term *detected event* to mean a cluster of documents output by our system.

In order to make sure that the clusters produced by our algorithm are sensible and

Algorithm 9: Online single-link clustering with latency.

input: Stream of documents D, clustering threshold θ, latency l

```
c ← []                                  // Initially, all the clusters are empty.
foreach Document d_i in stream D do
    d_nn ← arg max_{j∈{1,...,i−1}} cos(d_j, d_i)
    dist_min ← 1 − cos(d_nn, d_i)
    if dist_min > θ then
        c_i ← {d_i}                     // Start new cluster with seed document d_i.
        t(c_i) ← t(d_i)                 // Store the timestamp of the seed document.
    end
    else if t(d_i) − t(c_{d_nn}) ≤ l then
        // Add d_i to the same cluster as d_nn if d_i arrived at
        // most l seconds after the first document in that
        // cluster.  Otherwise, discard d_i.
        c_{d_nn} ← c_{d_nn} ∪ d_i
        c_{d_i} ← c_{d_nn}
    end
end
```

that they correspond to topics, we first evaluate our approach on the TDT detection task. Notice that if we use a latency of $+\infty$ in Algorithm 9, we exactly recover the clustering algorithm used by UMass (Allan et al., 2000b). Therefore, doing this should give us the same results on the detection task as the UMass system. We run our algorithm on the detection task in TDT5 with a latency of $+\infty$, and compare the results for different thresholds θ to the results of the UMass system. Results are shown in Table 6.2. Because detection has exactly the same time and space complexity as FSD, we note that our system is again over an order of magnitude faster than the UMass system, and is scalable to unbounded streams. As we can see from Table 6.2, our system is comparable to the UMass system, and in fact performs slightly better. This experiment shows that our clustering approach is competitive and produces reasonable clusters.

Threshold	Our approach	UMass
0.21	0.1636	0.1525
0.22	0.1440	0.1499
0.23	0.1623	**0.1473**
0.24	0.1593	0.1551
0.25	**0.1406**	0.1530
0.26	0.1532	0.1580
0.27	0.1612	0.1624
0.28	0.1725	0.1664
0.29	0.1646	0.1734
0.3	0.1675	0.1820

Table 6.2: Detection cost C_{det} for our system and the UMass system. Lower is better.

6.2 Evaluating Event Detection in Noisy Streams

Traditionally, evaluation of FSD systems, as performed in the TDT project, assumed that the documents come from a noiseless stream such as newswire. This means that all the documents in the stream are considered newsworthy and come from a reputable source. As a result, TDT evaluation has ignored precision and focused instead only on miss and false alarm rate (cf. Section 3.2). However, Twitter (and any other social media stream) is very noisy. Not only is there a significant amount of spam present in the stream, but most of the tweets that are not spam are not news-related. This is expected given that Twitter is primarily a social, and not a news reporting site, but it means that the same way of evaluating event detection as used in TDT is no longer suitable here. Therefore, it is maybe not that surprising that a lot of the work on event detection in Twitter has focused on using precision or related metrics, while ignoring recall. However, it should be fairly clear that, in order to get a complete picture of the usefulness of an event detection system, we should measure both recall and precision.

Measuring either recall or precision exactly is practically impossible on this scale. In order to measure recall, we would need to find every single event in our dataset of 51 million tweets, and in order to measure precision, we would need to label every event that our system detects (which can be on the order of hundreds of thousands). However, as we describe later in the chapter, there are ways of providing an unbiased estimate of these quantities that is manageable from the perspective of human effort involved.

Unfortunately, we are not aware of any prior work that reliably measures both recall and precision. For example, Weng et al. (2011) and Hu et al. (2008) do not use recall at all and instead focus only on other measures (precision and semantic shift). Those authors who do measure recall, only measure it on a small subset of events that passed some filter. This is because all of these approaches are concerned with detecting a specific type of event, and not with any event in general. Unfortunately, this also means that the recall numbers presented for such approaches only hold for specific types of events, and do not generalize to all event types. For example, Sakaki et al. (2010) measure recall of their system on the set of events that concern earthquakes and typhoons, which is a very biased sample. Becker et al. (2011a) use the size of a cluster together with a condition that the author is based in New York as a filter for their events, which is again a very strong bias. Finally, Popescu and Pennacchiotti (2010) only use tweets that talk about celebrities. On the other hand, we define events independently from our Twitter data and do not restrict ourselves to any specific type of event. While we used Wikipedia to help guide us in choosing the events, we did not consult our Twitter data in any way to make sure that, e.g., the event is even present in the data. This is obvious from the event *arrest of Goran Hadžić*, which, although being a major event, was only mentioned twice in our Twitter dataset. Therefore, we will present the first evaluation of a system for general event detection in Twitter. We will now explain how we compute recall and precision in our experiments.

6.2.1 Computing Recall

We compute recall using the set of 27 reference events described in Chapter 3. To measure recall, for every event detected by our system (a cluster of tweets), we have to decide if it "covers" any of the reference events, i.e., we have to define what makes a *true positive*. To decide if a detected event covers a reference event, we use an approach similar to that used in the TDT detection task (Fiscus and Doddington, 2002). We compute the proportion of tweets in the detected event (i.e., cluster) that are part of a single reference event. If this proportion is greater than some threshold, we say that the detected event *covered* a reference event. Finally, recall is computed by dividing the number of covered reference events (true positives) by the total number of events (27 in our case). This approach is presented in Algorithm 10. In all our experiments we use the threshold $\theta = 0.5$, i.e., we require that at least half of the tweets in the detected cluster appear in the same reference cluster.

Algorithm 10: Algorithm for computing recall.

input : Set of detected events E, set of reference events R, threshold θ

output: Recall

```
covered ← ∅
foreach Event e in E do
    foreach Reference event r in R do
        p ← |e∩r| / |e|
        if p > θ then
            covered ← covered ∪ r
        end
    end
end
return |covered| / |R|
```

6.2.2 Computing Precision

Computing precision is more difficult than computing recall because now we cannot use the corpus from Chapter 3. The reason for this is that our corpus only contains positive examples (actual events), and precision should measure how many non-events (spam, trivial chatter) we detect along with actual events. This is why we adopt the following approach: from the set of all detected events we randomly sample 100 events and manually label which ones correspond to actual events. This will give us an estimate of the precision of the detected events. While this approach is somewhat manually intensive, it is unbiased and therefore provides us with a reasonably accurate estimate of the true precision. For the purposes of labeling, each cluster is represented by its centroid tweet, and this is shown to the annotator. We do not show all the tweets in the cluster as some of the clusters contain thousands of tweets, which would make the labeling task too intensive. However, our detection experiments from Section 6.1 showed that the clusters produced by our system are reasonable, and thus the centroid should be a good representative of the cluster (we also confirmed this by manual inspection of a few random clusters). Because of the substantial manual effort involved in labeling these tweets, we only use one annotator. However, as we discuss later in the chapter, we used a second annotator to label a random subset of these tweets and found that the agreement between two annotators was substantial. This shows that the amount of variance introduced in the results by using only one annotator is minimal.

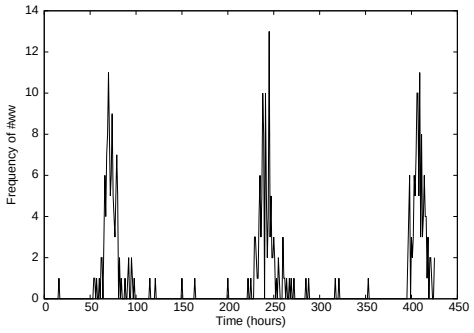

Figure 6.1: Frequency of the hashtag *#ww*. We can see the bursty behaviour repeating in a weekly pattern.

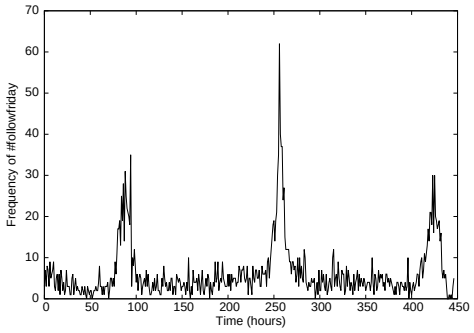

Figure 6.2: Frequency of the hashtag *#followfriday*. We can see the bursty behaviour repeating in a weekly pattern.

6.3 Baseline Approach – Bursty Clusters

Burstiness, a sudden increase in the frequency of some item, is often used in event detection. Its use is motivated by the observation that event-related words exhibit bursty behavior – when an event happens (say, Amy Winehouse dies), words related to that event (Amy, Winehouse, die, dead) will be used much more frequently than in the past. This idea forms the basis of many approaches to event detection in Twitter (Mathioudakis and Koudas, 2010; Cataldi et al., 2010; Weng et al., 2011; Cordeiro, 2012; Li et al., 2012a; Agarwal et al., 2012; Ozdikis et al., 2012). However, using burstiness in place of novelty has two major flaws. First, just because something suddenly became popular does not mean that it is related to a new event (leading to a loss of precision), and second, a lot of new events will never become popular (leading to a loss of recall). As an example of the first problem, note that many words on Twitter exhibit cyclical bursty behaviour. This means that simple burstiness approaches will detect these words as events every time they become bursty, thus reducing precision. In Figures 6.1 and 6.2 we show the frequencies of hashtags #ww and #followfriday over time. It is clear that both hashtags exhibit cyclical burstiness, but they are not related to any events – #ww means *woof wednesday* and is used by animal lovers to communicate on Wednesdays, while #followfriday is used on Fridays by users who want to get more followers. Because our approach incorporates novelty, it does not suffer from this problem, as tweets that contain these hashtags will not be deemed novel, and thus will not start new clusters.

To illustrate the lack of burstiness in new events on a real example, we show the

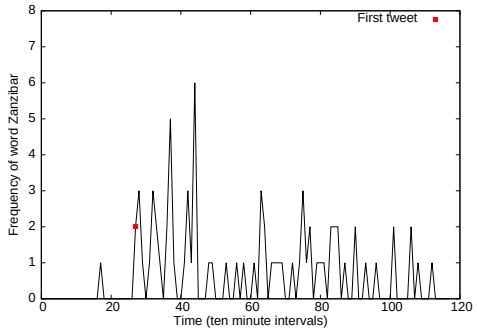

Figure 6.3: Frequency of the word *Zanzibar* around the time of the ferry sinking accident. Red point shows the first tweet that mentions this event.

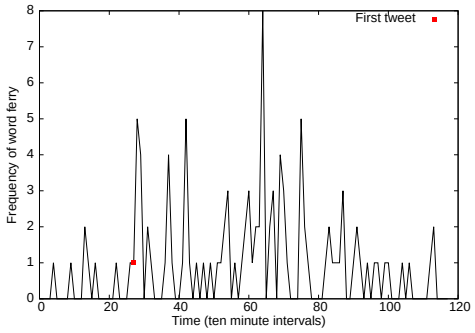

Figure 6.4: Frequency of the word *ferry* around the time of the ferry sinking accident. Red point shows the first tweet that mentions this event.

frequency of words *Zanzibar* and *ferry* in the Twitter data around the time when a ferry sank in Zanzibar, killing almost 3,000 people. The frequencies are shown in Figures 6.3 and 6.4. We can see that neither of the two words that are crucially related to the event exhibit significant spikes around the time when the event happened. This lack of burstiness would make most existing event detection approaches miss this event. Agarwal et al. (2012) were the first to mention the problem of low recall, noting that 45% of events in their data did not pass the burstiness threshold. Unfortunately, they simply ignore this problem and measure recall as a percent of the events that are bursty. We repeated their experiment on our data with the same settings (20 minute latency, burstiness threshold of four), and found that 44% of events did not pass this filter. This confirms the findings of Agarwal et al. (2012) and shows that burstiness-based approaches with an aggressive threshold[1] result in low recall, which is rarely acknowledged in the literature.

While we have shown that using novelty also suffers from the problem of low precision, it alleviates the problem of low recall because all new events should be novel. Therefore, we combine novelty with burstiness in hope of still achieving good recall, while improving precision. We combine burstiness and novelty in the following way: for each cluster output by the clustering algorithm 9, we examine its size, and if the size exceeds a given threshold, we report the cluster as an event, otherwise we discard it. Burstiness is accounted for by the cluster size, whereas novelty comes from the fact that a new cluster can be started only by a novel tweet. Note that, unlike the

[1] Other approaches like Li et al. (2012a) and Ozdikis et al. (2012) use even higher thresholds.

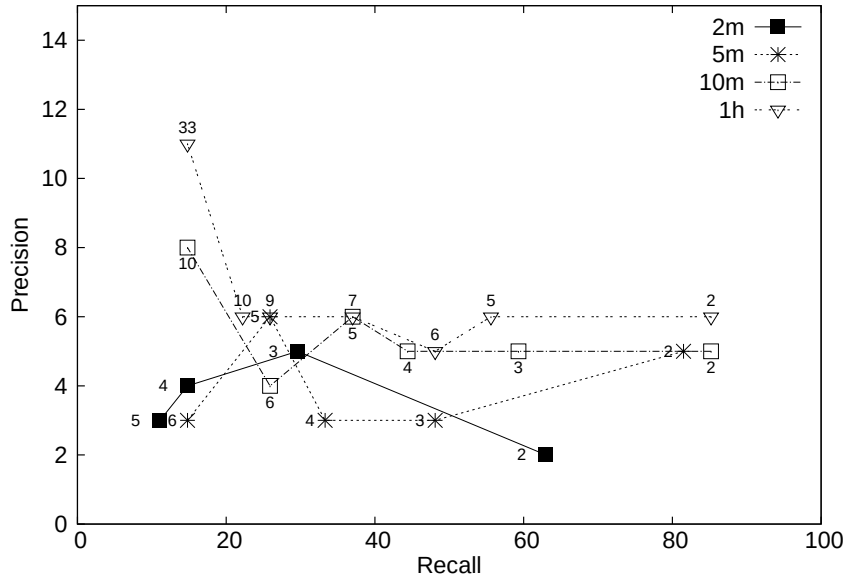

Figure 6.5: Effect of latency on precision and recall of detected events. Numbers next to points indicate the size threshold that was used.

previous work, we deal with entire documents, instead of single words. This is an advantage of our approach, as event detection is about detecting breaking stories, not detecting breaking words. Because mapping clusters of words to real events is a very hard and subjective task, we do not compare our approach to other approaches based on burstiness of single words. In the rest of the thesis, we will refer to this approach as *Size* for the sake of simplicity.

To get an idea of the relationship between latency, the size threshold (i.e., the burstiness threshold), recall, and precision, we turn to the precision-recall plot in Figure 6.5. There are a couple of things to note from this figure. First, by using burstiness with the lowest possible threshold of two, we are able to significantly ($p = 0.05$) improve upon the approach that does not use size, which had less than 1% precision. By using this threshold, recall dropped from 93% to 85%, which is acceptable.

For very low latencies (two and five minutes) further increasing the size threshold does not significantly improve precision, while significantly affecting recall. For a latency of ten minutes or higher we see an improvement in precision as we increase the threshold. This suggests that, if we want very low latency, using a size threshold of more than two does not make sense. Finally, we see that as we increase latency, results

generally improve, but very slowly. For example, for a latency of one hour, precision at recall 15% is significantly ($p = 0.05$) higher than the precision at the same recall for latency of two and five minutes, but not for latency of ten minutes. Based on these results, we will use a latency of ten minutes in the future experiments as this still means we are detecting events in near real-time, while at the same time achieving results that are generally better (although not significantly) than those achieved by lower latencies.

6.4 Conclusion

In this chapter we showed how to use our FSD system from earlier chapters for detecting events. We first showed that a traditional approach based on novelty alone performs poorly in the presence of noise, as over 99% of detected events are not interesting. Given that this is the approach that was used in detection systems in TDT, this shows just how challenging Twitter is from the perspective of event detection, compared to newswire. We then combined the novelty-based approach from TDT with burstiness and showed that this can significantly improve precision, with only a slight drop in recall. Finally, we explored the relationship between latency in detecting the events and the burstiness threshold and found that a threshold higher than two does not significantly improve results, and that a latency of ten minutes gives us the best trade-off between improved precision and detecting events in real-time. In the next chapter, we will see how external information could be used to further improve precision.

Chapter 7

Cross-stream Event Detection

Today, new events are reported in multiple streams (newswire, Twitter, blogs, Wikipedia, etc.). Modern approaches to event detection should be wise to take advantage of this redundancy in order to improve different aspects of their performance. For example, we might want to reduce noise by noting that if something is mentioned only in Twitter, but not in newswire or blogs, then it is less likely to be related to a real event. Another idea might be to use the fact that some events break first in Twitter and others break first in newswire, in order to reduce the overall latency in detecting the events. In this chapter, we look at ways of combining evidence from multiple streams in order to improve different aspects of our system.

First, we combine Twitter and Wikipedia and show that by doing so we can significantly reduce the amount of noise in the detected events. We then combine Twitter with newswire and show that combining the two streams can reduce the latency in detecting the events, as well as detect more events than would be possible from using newswire alone. Because we are mostly interested in combining social with non-social media streams, we leave separately combining Wikipedia and newswire for future work.

7.1 Data

We first describe the two datasets that are used throughout this chapter as a source of additional information for event detection.

7.1.1 Wikipedia Page Views

Wikipedia is a free online encyclopedia that can be edited by anyone. It is a rich, constantly evolving information source that contains over 4 million articles in English alone, and over 27 million articles in total across multiple languages. Events can be reflected in Wikipedia in many ways: through new page creation (e.g., when a plane crashes a new Wikipedia page is created about this event), edits to existing pages (e.g., when a celebrity dies their page is updated with this information), or through page view counts (e.g., when a celebrity dies many people visit their Wikipedia page to get more information about the circumstances, making the page view count for that page spike). Of these, page view counts are used pretty much exclusively in the literature on event detection. This is perhaps not surprising given that Ciglan and Nørvåg (2010) found that spikes in Wikipedia page views correspond to real-world events, suggesting that it is sufficient to use this data for the purposes of event detection. Therefore, in keeping with the existing work on event detection and Wikipedia, we focus on page view counts in this thesis.

The page view counts[1] come in the form of page names and the number of visits to that Wikipedia page in a particular hour.[2] If an event happens, we expect the view counts for pages concerning the involved entities to spike. As an example, consider the page view counts for the page *Amy Winehouse* and the page *Glasgow* in the period from June 1st 2011 to November 31st 2011, shown in Figure 7.1. While the page views for *Glasgow* remain more or less at the same level (presumably because no major events concerning Glasgow happened during that time), the view counts for *Amy Winehouse* exhibit a spike on the date when she cancelled her European tour and a very big spike on the date of her death. This example backs up the intuition that real-world events are reflected in the stream of Wikipedia page counts, and can be identified by looking at the spikes in the data. In Section 7.2 we will investigate this claim more formally.

7.1.2 Newswire

As our source of newswire data we use articles from eight major news sources: BBC, CNN, Google News, Guardian, New York Times, Reuters, The Register, and Wired.

[1] Obtained from Wikipedia's public logs `http://dumps.wikimedia.org/`

[2] To be completely precise, the number represents the number of page *requests*, not page views. Page requests can be generated by crawlers and even pages that do not exist in Wikipedia can have a non-zero request count (e.g., if someone tries to enter the URL directly in the browser but misspells the page or if the crawler requests a non-existing page). However, this is a minor technical difference and in the rest of the thesis we will refer to this as page views.

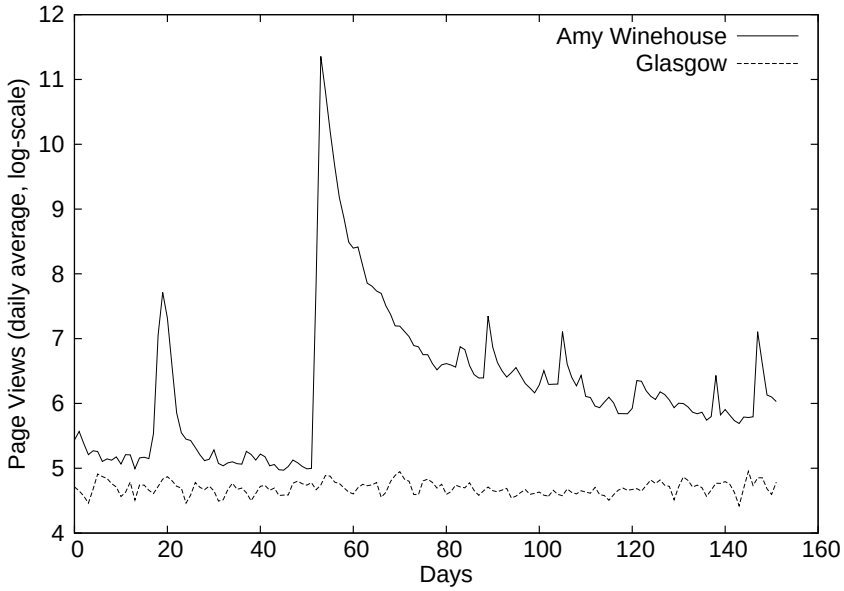

Figure 7.1: Page view counts for pages *Amy Winehouse* and *Glasgow*. Note the logarithmic scale on the y-axis.

This data was provided to us by the Terrier team at the University of Glasgow.[3] It consists of 180 thousand articles spanning the whole of 2011. The data was collected through the RSS feeds of these sources, and contains the article's headline and the full text, as they appear in the feed. Each article also comes with a timestamp, to a resolution of one second, that indicates the time when the article was published in the RSS feed. This timestamp is not the same as the time when the article was first published on the website, and this should be taken into account when reading the results. Mostly, our experiments which use timestamps should be interpreted as examining the relationship between newswire RSS feeds and Twitter. In some of our later experiments we do try to estimate how much these timestamps are off from the true timestamps and take this into account in our results. In the rest of the thesis, when we say *newswire* we will use it to mean the eight newswire sources that are covered in our data, unless it is explicitly stated otherwise.

The distribution of sources and the average lengths of documents for each source are shown in Figure 7.2. We can see that majority of articles (about 55%) come from the BBC and the New York Times, whereas other six sources account for about 45% of

[3] `http://terrier.org/`

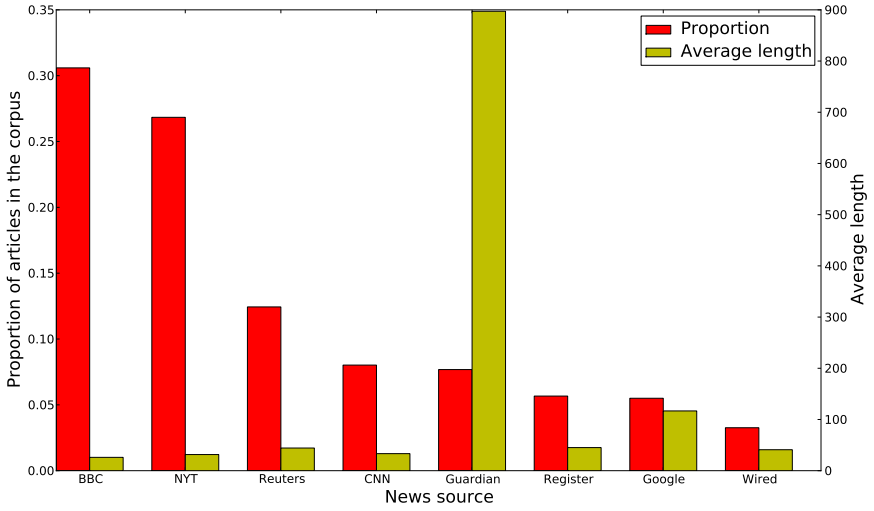

Figure 7.2: Distribution of news sources in our data and average document length.

the data. In terms of document length, all sources have similar length articles, except for Guardian, whose articles are an order of magnitude longer than others.

7.2 Combining Twitter and Wikipedia

Wikipedia has already been used for a number of purposes: from ontology building to text mining tasks. Listing all of its applications here is beyond the scope of this thesis, but an interested reader is referred to Medelyan et al. (2009) for a very comprehensive list of related work in this area. Given its widespread use, it is perhaps not surprising that researchers have also started looking into using Wikipedia for event detection. The first step in this direction was Ciglan and Nørvåg (2010) who showed that increases in Wikipedia page views correspond to events happening in the real world. Ahn et al. (2011) made the first attempt to detect events from Wikipedia based on the spikiness of the page views for an article. They identify 100 spiking pages on a daily basis, cluster, and summarize them in order to present the result to a user. In this work we will be more interested in finding spiking pages with a lower lag, and combining this information with events detected from Twitter.

The only work that we are aware of which uses both Twitter and Wikipedia is the work by Li et al. (2012a). They use Wikipedia to define what is "newsworthy", and then use this to rerank the detected events according to newsworthiness. The re-ranking

function looks at how likely a phrase p is to be the anchor text in a Wikipedia article that contains p. However, this approach treats all pages equally (i.e., ignores the page views) and there is no treatment of time. Finally, Li et al. (2012a) do not measure how much improvement in their results is due to the use of Wikipedia.

To better understand the relationship between the two streams and what kind of results we might expect, we first examine the lag between the Twitter stream and the stream of Wikipedia page views. We perform a twofold analysis of the lag: a manual one, where we measure the lag between Twitter and Wikipedia on the set of 27 reference events described in Chapter 3, and an automatic one where we try to automatically determine the lag. This is the first time that Twitter and Wikipedia have been compared with respect to their time properties. After this, we will explore possible ways of combining these two streams for the purposes of event detection.

7.2.1 Detecting Spikes in Wikipedia

Our first step is to produce a stream of "events" from the page view data. Clearly, we cannot use the standard FSD approaches here as the data does not consist of documents, but of page titles and their view numbers. This is why we use a simple, but effective algorithm for detecting potential events in this stream. The basic idea is that big spikes in page view data indicate that an event might be happening. We treat the data as a time series and use a moving window of n hours (where n is a parameter) to compute the historic mean and standard deviation of page views in this period. We then test to see if the new page view count differs substantially from the mean. More precisely, if the count is more than k standard deviations greater than the historic mean, we report it as an event (k is a parameter of the algorithm). This is shown in Algorithm 11.

Algorithm 11 is essentially the same as Grubbs' test (Grubbs, 1969) with a minor difference that we outline below. Detecting outliers from a time series is a vast field in itself and our aim here is not to provide a comprehensive study of the different outlier detection methods. We choose this particular technique because of its efficiency and simplicity, but we note that other methods might perform better. We leave this for future work. For a review of the many techniques that exist in the literature, we refer the reader to Kriegel et al. (2009).

The main difference between our algorithm and Grubbs' test is that the G statistic computed in step 4 cannot be interpreted in terms of significance levels, as is the case in Grubbs' test. The reason for this is the fact that the page view data does not follow

Algorithm 11: Detecting events in a stream of Wikipedia page views.

input: Wikipedia page view statistics P, window size n, threshold k

foreach x_t *in* P **do**
  $\bar{x}(t) = \frac{1}{n}\sum_{i=t-n}^{t-1} x_i$
  $s^2(t) = \frac{1}{n-1}\sum_{i=t-n}^{t-1}(x_i - \bar{x}(t))^2$
  $G = \frac{x_t - \bar{x}(t)}{s}$
  if $G > k$ **then**
    Report page t as event
  end
end

a normal distribution, which is an assumption made in Grubbs' test. To illustrate that the normality assumption does not hold, we show histograms of the page view data for Betty Ford's and Motorola's Wikipedia page in Figures 7.3 and 7.4. It can be easily seen from the figures that the distribution is not normal and has a long tail to the right. Note that we did not show page views higher than 200 in the histograms. If we had shown the page views above 200, the right tail would extend all the way to 40,000, making the histogram illegible. To actually test the normality assumption we performed the Shapiro-Wilks test (Shapiro and Wilk, 1965) on a number of pages. We found that the null hypothesis (normality assumption) could always be rejected at $p = 10^{-15}$, which is a definitive proof that the distribution is not normal. Therefore, we use Algorithm 11 to detect outliers (i.e., events) but note that the parameter k does not have the same interpretation as it does in Grubbs' test (i.e., it cannot be interpreted in terms of significance levels).

7.2.2 Manually Determining the Lag

In order to establish the utility of using Wikipedia page views for event detection, we first carry out a manual investigation of the lag between this stream and the Twitter stream. First, we note that because the page view counts arrive in hourly intervals, this puts a lower bound on the latency. Assuming that there is an equal chance of an event happening at any minute of the hour, which is a very weak and reasonable assumption, this means that in expectation the minimum lag of the Wikipedia stream is 30 minutes. Using the stream of events detected from the Wikipedia page counts using Algorithm 11 (with $n = 24$ and $k = 7$), we measure the lag from the first tweet

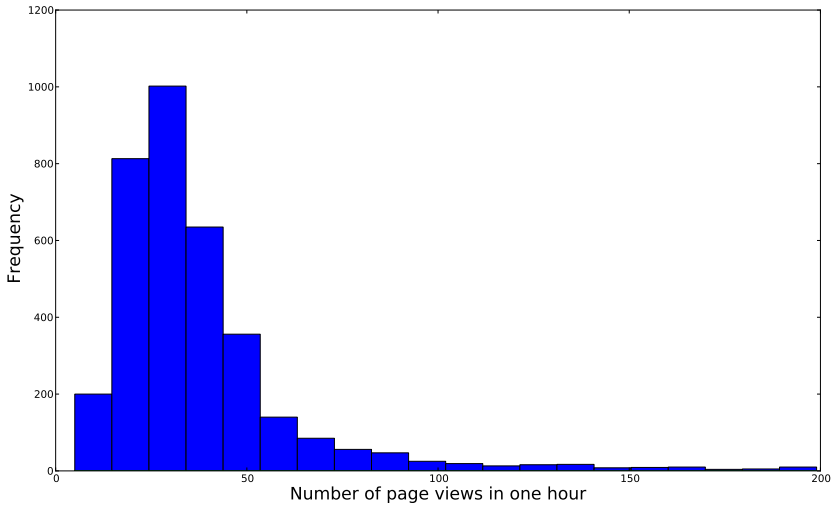

Figure 7.3: Histogram of page view counts for *Betty Ford*.

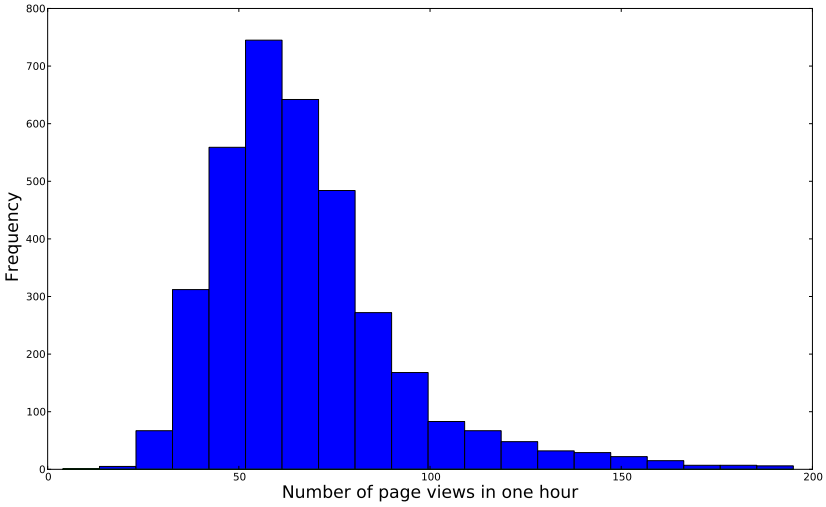

Figure 7.4: Histogram of page view counts for *Motorola*.

of each of the 27 reference events in our Twitter corpus. We do this by manually examining the stream of detected events in Wikipedia from the time that the event happened and looking for page titles that are related to the event. Whether a page title is related to an event is of course a subjective measure, and this should be kept in mind when interpreting the results.

Results of this evaluation are shown in Table 7.1. Overall, Wikipedia lagged behind Twitter on 20 events, Twitter lagged behind Wikipedia on one event, and six events were not detected in the Wikipedia stream at all. The only event where Wikipedia was leading was the arrest of Goran Hadžić because news of this event first broke in non-English media and non-English Twitter (we only deal with English tweets). For the 20 events where Wikipedia lagged, the median lag was one hour and 22 minutes.

7.2.3 Automatically Determining the Lag

As we mentioned in the previous section, manually determining the lag between Wikipedia and Twitter has certain limitations. For example, for the event *India and Bangladesh sign a border pact*, in manual evaluation we determine that the spike in page views for the page *Bangladesh* is caused by this event. However, this may or may not be true, which is why we also carry out further evaluation in an attempt to determine the lag between two streams.

In this experiment, we use our FSD system and the single-link clustering approach described in Algorithm 9 to obtain a large set of potential events from Twitter. In particular, we keep all the clusters that grew to ten or more tweets in one hour. Each cluster here is represented by the mean of all tweets in the cluster. This gave us 14,181 potential events in Twitter (keep in mind that around 94% of these clusters, according to Figure 6.5, will not be actual events). We then use this stream together with the Wikipedia stream of outliers described in Section 7.2.2 in the following way: for each detected event in the Twitter stream (represented by the centroid tweet), we find its nearest neighbor in the stream of spiky Wikipedia pages in terms of the cosine distance. Because Wikipedia page dumps come in an hourly interval, we also bucket the tweets in hourly intervals by rounding up their timestamps to the hour. By changing the hourly interval in which we search for the nearest neighbor, we can simulate the desired lag between the two streams. For real events, we would expect the distance to the nearest neighbor to be lower than for spurious events. This is because we expect that for real events (e.g., death of Amy Winehouse), a matching title will be spiking in Wikipedia

Event	Lag
Amy Winehouse dies	1h 49m
Atlantis shuttle lands	1h 4m
Betty Ford dies	1h 3m
Richard Bowes killed in riots in England	n/a
Flight 4896 crash	1h 14m
S&P downgrade US credit rating	1h 42m
US increases debt ceiling	n/a
Terrorist attack in Delhi	2h 7m
Earthquake in Virginia	1h 7m
First victim of London riots dies	n/a
War criminal Goran Hadzic arrested	-20h 18m
India and Bangladesh sign a border pact	36m
Plane with Russian hockey team Lokomotiv crashes	1h 1m
Explosion in French nuclear plant in Marcoule	1h 18m
NASA announces there might be water on Mars	1h 52m
Google announces plans to buy Motorola Mobility	1h 22m
Car bomb explodes in Oslo, Norway	1h 22m
Gunman opens fire in youth camp in Norway	1h 46m
First artificial organ transplant	n/a
Petrol pipeline explodes in Kenya	4h 43m
Famine declared in Somalia	n/a
South Sudan becomes independent country	1h 55m
South Sudan becomes UN member state	n/a
Three men die in riots in England	n/a
Riots break out in Tottenham, England	1h 52m
Rebels capture International Tripoli Airport	52m
Ferry sinks in Zanzibar	1h 4m

Table 7.1: Lag between events detected in Wikipedia and their counterparts in Twitter. Lag less than zero means that we detected an event in Wikipedia sooner than in Twitter. n/a means that the event was not detected in Wikipedia.

(e.g., Amy Winehouse's Wikipedia page), whereas for spurious events no good match will be found (e.g., for the spurious event *One hour is not enough! #ANTM* we would not expect to find a spiking Wikipedia page with a similar title). Therefore, when the two streams are properly aligned, i.e., when the best lag is chosen, we would expect the mean distance to the nearest neighbor to be minimum.

At the same time, we would expect the standard deviation of the distances to be maximum when the two streams have optimal alignment. To see why this is so, consider, e.g., the event *ferry sinks in Zanzibar*. If we choose a lag of -3 hours, i.e., we think that Wikipedia leads Twitter by three hours, this means that the tweets for the ferry sinking event will be compared to Wikipedia page titles that were spiking three hours before the event happened. Because it is extremely unlikely that any page related to this event would be spiking at that time because the event had not happened yet, the distance from tweets that talk about this event to the nearest spiking Wikipedia page title will be large. Because distances to the nearest spiking Wikipedia page title for spurious events will also be large, the overall standard deviation of the scores will be small. On the other hand, for a correctly chosen lag, the distances of real events to spiking page titles will decrease, while the distances of spurious events will not, causing an increase in the standard deviation of the scores.

The mean distances and the standard deviation of these distances are shown in Figure 7.5, for different lags. We can see that when Wikipedia lags Twitter by one hour, the mean distance is minimal, with the standard deviation being maximal. Because we rounded up Twitter timestamps,[4] this means that we have introduced on average half an hour lag in the Twitter stream. Together with the one hour lag that we measured, this sums up to a lag of one and a half hours, which is similar to what was measured manually on the set of 27 reference events. Note that we have tried varying the k parameter in Algorithm 11 that is used to define how spiky the page has to be in order to be considered an outlier in Wikipedia. We lowered this parameter to $k = 3$ and the results were still the same, with the minimum average distance and the maximum standard deviation still achieved for a lag of one hour. This shows that our method of measuring the lag does not depend on the setting of the k parameter in Algorithm 11.

To completely remove any doubt about the possible biases that the parameters in Algorithm 11 might have introduced in measuring the lag, we conduct a third experiment. This experiment will take all of the available data into account, and is therefore

[4]Twitter timestamps have a resolution of one second, but Wikipedia timestamps have a resolution of one hour, so we have to round Twitter timestamps to an hour. We chose to round up the timestamps, but this makes no difference for the final results.

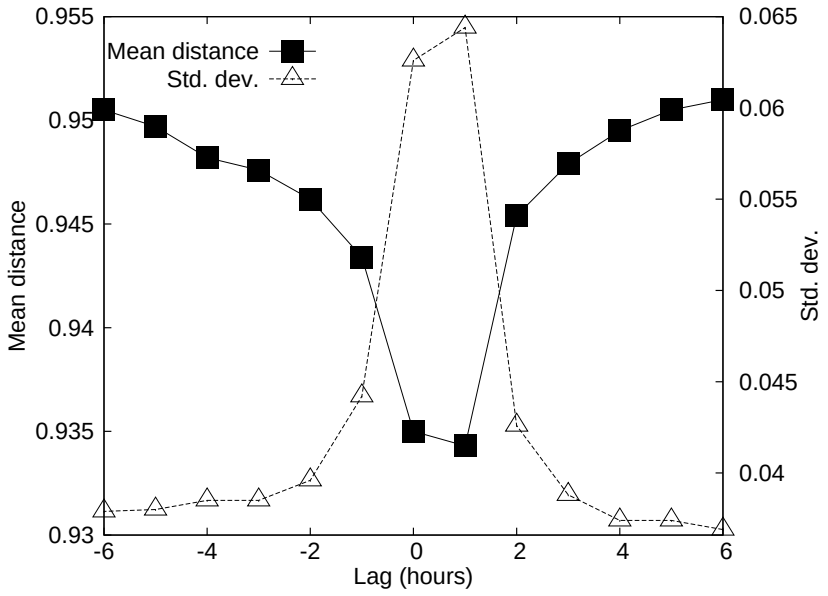

Figure 7.5: Mean and standard deviation of distances from Twitter first stories and their nearest Wikipedia page titles.

independent of the method for detecting events in either Wikipedia or Twitter. In this experiment, we compute the *cross-correlation* between the full Twitter[5] and the full Wikipedia stream. Cross-correlation is a commonly used similarity measure between two streams in the signal processing literature (Orfanidis, 1996).

In order to compute the cross-correlation we first have to represent the streams as time series. We do this by simply transforming each stream into multiple time series, one for each word in the vocabulary. The value of the series for word w at time τ (we shall denote this $f_w(\tau)$ for Twitter and $g_w(\tau)$ for Wikipedia) is then simply defined as the frequency of word w at hour τ in the stream. For Wikipedia, we get these frequencies from the page view counts, whereas for Twitter we count the number of occurrences of w in a particular period. We then compute the cross-correlation as

$$(f \star g)(t) = \sum_{\tau=-\infty}^{+\infty} \sum_{w} (f_w(\tau) * g_w(t+\tau)). \tag{7.1}$$

A higher value of cross-correlation for a certain lag means that the streams are better aligned for that lag. We plot the cross-correlation values for different lags in Fig-

[5]By full Twitter stream we mean using all the Twitter data that we have available. This is not the full Twitter stream, but the 1% sample that is publicly available.

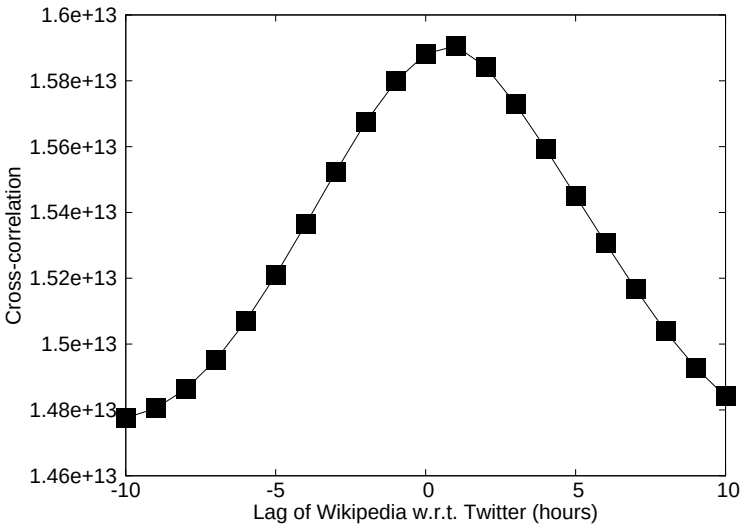

Figure 7.6: Cross-correlation between Twitter and Wikipedia streams.

ure 7.6. We can see that the maximum cross-correlation is achieved when Wikipedia lags behind Twitter by one hour. Taking into consideration the fact that we rounded up Twitter timestamps, we again get the same result as before, i.e., that Wikipedia lags by one and a half hours.

7.2.4 Improving the Quality of Detected Events Using Wikipedia

Here we present two ways (strategies) of combining the Twitter stream with the stream of Wikipedia page views. Both of them aim at assigning a score to each detected event from the Twitter stream that should correspond to how likely this event is a real event, and not some Twitter-specific artifact (spam, trivial/non-interesting event, etc.). By choosing a threshold for this score, we declare all events with the score above the threshold to be real events, and everything else to be a spurious event. In all of our experiments we use the clustering with latency approach described in Section 6.1 with a latency of ten minutes to obtain the list of potential events in Twitter.

The first way of combining the two streams explicitly uses Wikipedia's stream of outliers, and the score it assigns to Twitter events can be summarized as *similarity to the nearest Wikipedia outlier*. That is, the score assigned to each detected Twitter event is equal to the cosine similarity between the cluster's centroid tweet[6] and the

[6]Recall that events are just clusters of tweets.

nearest page title in the stream of Wikipedia outliers (obtained as in Algorithm 11). When searching for the nearest neighbor, we only look at the Wikipedia pages from the appropriate time interval, i.e., using the latest available (with respect to the most recent tweet in the cluster) page view statistics. The intuition behind this approach is that real events will be very similar to a page that is "spiking" in Wikipedia. We call this strategy *Wiki NN*, and it is shown in Algorithm 12.

Algorithm 12: Wiki NN algorithm for filtering Twitter events using Wikipedia.

input : Potential Twitter events T, Wikipedia outliers W, threshold θ, lag l
output: A stream of filtered Twitter events

```
foreach Cluster c in T do
    d ← centroid tweet in cluster c
    // t(d) is the timestamp of tweet d.
    // W[t(d)+l] is the set of outlier pages in the hour t(d)+l.
    sim ← max_{p∈W[t(d)+l]} cos(d, p)
    if sim ≥ θ then
        Report c as an event
    end
end
```

The other way of combining the two streams does not explicitly use the stream of outliers because producing this stream involves defining a threshold that decides if a page is spiking or not. A different threshold will produce a different stream of outliers, which may (or may not) change the results. Rather than using this threshold to define if a Wikipedia page is spiking or not, the second strategy uses the "spikiness" of the page directly. In this strategy, we assign a score to each detected Twitter event according to the spikiness of its nearest neighbor in the full Wikipedia stream. The spikiness of a page is the G score that was used in Algorithm 11 to decide if a Wikipedia page is an outlier. We call this strategy *Wiki spike* because it uses spikiness of a Wikipedia page to decide if an event is real. As before, we only use the Wikipedia information from the appropriate time interval. Pseudocode for this approach is shown in Algorithm 13.

We use these two strategies to filter out those events that are artifacts of the noise present in the Twitter stream. As before, we measure precision and recall of this approach. As the basis of the events detected from Twitter, we use the basic clustering approach from Section 6.1 with clustering threshold 0.4 and latency of ten minutes. We remove all singleton clusters formed this way. All the Wikipedia information used

Algorithm 13: Wiki spike algorithm for filtering Twitter events using Wikipedia.

input : Potential Twitter events T, Full Wikipedia stream W, spikiness scores of Wikipedia pages S, threshold θ, lag l

output: A stream of filtered Twitter events

foreach *Cluster c in T* **do**
  $d \leftarrow$ centroid tweet in cluster c
  // $t(d)$ is the timestamp of tweet d.
  // $W[t(d)+l]$ are all the pages viewed in the hour $t(d)+l$.
  $p_{nn} \leftarrow \arg\max_{p \in W[t(d)+l]} \cos(d, p)$
  // $S(p)$ is the spikiness score of a Wikipedia page p.
  // This is the G score computed in Algorithm 11.
  if $S(p_{nn}) \geq \theta$ **then**
    Report c as an event
  end
end

in these experiments has a lag of 0 with respect to the Twitter stream, which means that we only use Wikipedia information that was available at the time when the event was detected and thus no additional lag is introduced by using Wikipedia.

Because the Wiki NN strategy depends on the set of pages that are determined to be spiking using Algorithm 11, we first look at how the parameter k of Algorithm 11 affects the results of this strategy. Lowering k will give us more spiking pages, but they will contain more "noise", i.e., pages that are not spiking because of an event, whereas increasing k will give us fewer spiking pages, but they are more likely to be related to actual events. We try $k = 3$ and $k = 7$, as well as setting $k = -\infty$. Setting $k = -\infty$ means that, for every time interval, we consider all Wikipedia pages as spiking. This is somewhat similar to the approach used by Li et al. (2012a), in that time and page view numbers are ignored, and Wikipedia is only used to determine if the tweet contains a newsworthy concept. Results are shown in Figure 7.7. First, we note that, for higher levels of recall, there is no significant difference between precisions of different thresholds. However, for lower recall, using a threshold for page spikiness achieves significantly better precision than not using it (i.e., setting it to $-\infty$). In fact, notice that when we increase the threshold of the Wiki NN algorithm when $k = -\infty$, this leads to lower precision. On the other hand, when using $k = 3$ and $k = 7$ increasing θ leads to lowering recall and increasing precision. This is somewhat expected as

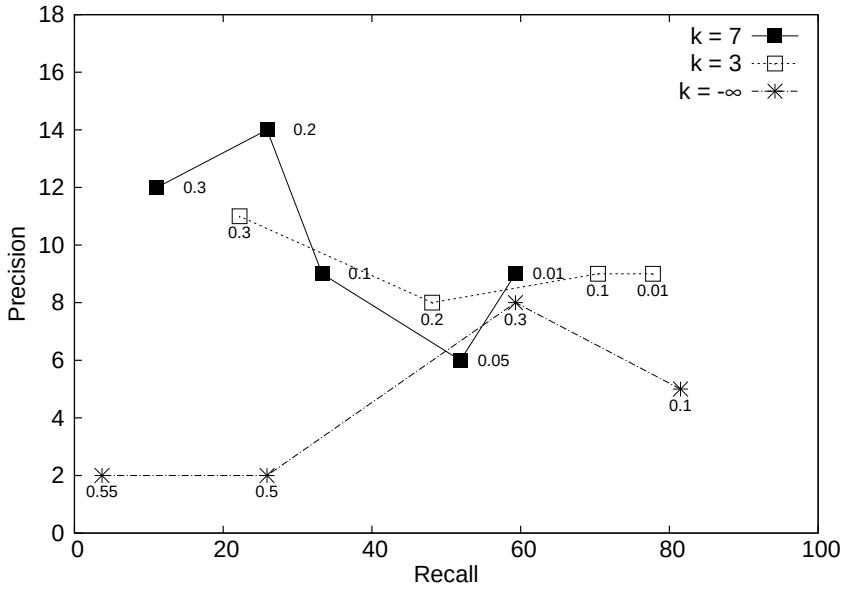

Figure 7.7: Performance of Wiki NN strategy for different thresholds of page spikiness. $k = -\infty$ means that all pages are used.

tweets which look a lot like general Wikipedia page titles are not likely to be reporting a real event. Note also that setting k to a value different than $-\infty$ implicitly takes time into account because we only keep a set of pages that spike in a certain time interval. This experiment therefore also shows that taking time information in Wikipedia into account results in a significantly higher precision (for low recall values).

We now show the precision-recall plot of the two strategies that use Wikipedia data (Wiki NN and Wiki spike) as well as the baseline strategy (size). Labels next to the points indicate the value of threshold used for the particular strategy. We can see from Figure 7.8 that using Wikipedia achieves the expected effect – precision is increased, while recall is lowered compared to the size strategy. The main reason for the lower recall is the latency in the Wikipedia stream – a lot of true events get low scores because the spike in Wikipedia for the corresponding pages only happened an hour after the event was detected in Twitter. For example, if we use the Wikipedia data with a lag of one hour, Wiki spike can achieve 30% recall with 28% precision. For comparison, with a lag of 0 hours, the same strategy achieves a recall of 26% with a precision of 13%. This would mean that using the lag of one hour would achieve a higher recall with a significantly higher precision ($p = 0.01$). However, a latency of

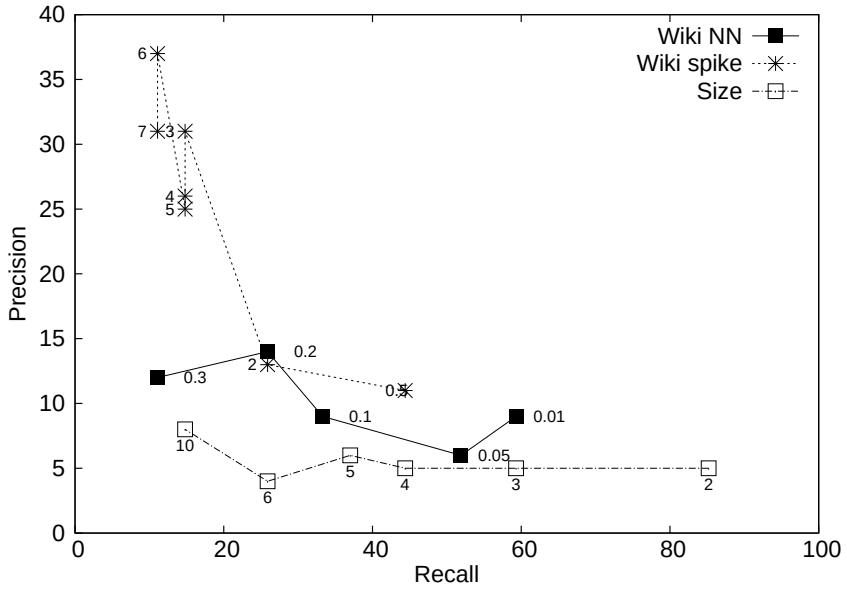

Figure 7.8: Comparison of strategies that use Wikipedia with the size-based baseline. Numbers next to points indicate different threshold values for the particular strategy.

one hour is unacceptable for the purposes of real-time event detection, so we only use Wikipedia information with a lag of 0 hours, i.e., no lag.

We can see that in general, Wiki spike achieves a much higher precision, but also somewhat lower recall, than Wiki NN. As we lower recall, both strategies outperform the baseline strategy that uses only size. For example, for a recall of 25%, both strategies that use Wikipedia achieve significantly ($p = 0.05$ using a two-sample z-test) higher precision than the size-based strategy. For the same recall level, the Wiki spike strategy also significantly outperforms Wiki NN. However, we can also see that both strategies cannot achieve as high recall levels as the size strategy. We tried lowering the thresholds for both strategies as low as possible without the threshold being zero, but this did not improve recall. When the threshold is zero (i.e., Wikipedia data is not taken into account), both of these strategies reduce to the size strategy, and thus achieve the same recall and precision as this baseline strategy.

Like we mentioned in Section 6.2.2, each point in Figures 6.5, 7.7, and 7.8 involved one annotator labeling 100 tweets. To make sure that using only one annotator does not introduce bias into our results, we randomly sampled 500 clusters from the set of all clusters labeled by the first annotator (roughly 4,500), and had a second annotator

label them. We measured Cohen's kappa coefficient (Cohen, 1960) between the two annotators on this set of 500 clusters and found it was 0.70, which is usually taken to mean substantial agreement. This shows that using only one annotator is justified, as there is a good agreement between humans on which clusters represent events.

Comparing the two strategies that use Wikipedia data, we find the Wiki NN strategy to be less preferred. Firstly, it uses one additional parameter (the threshold k that determines if a page is spiking or not). As we saw in Figure 7.7, results are not very sensitive to this parameter, but having less parameters is always better. If we take into account that Wiki spike can achieve significantly higher precision for lower recall levels (e.g., precision 37% at recall 11%, vs. precision 12% for the same recall), we conclude that Wiki spike is the preferred strategy for incorporating information from Wikipedia into a system that detects events in Twitter.

7.3 Combining Twitter and Newswire

There has been relatively little work on comparing newswire and Twitter streams. The work that does exist in this area mostly focuses on comparing the two streams in terms of topical similarity, but ignoring the time aspect. Subašić and Berendt (2011) compared tweets and blogs to articles from Reuters, Associated Press (AP), and professional news outlets in terms of the similarity of the underlying language models. They found that tweets were very similar to headlines of news articles, but also that they were dissimilar from the full texts. This is why in our cross-correlation experiments below we compare tweets and headlines and not tweets and full texts. The only work that we are aware of which does take the time aspect into account is the work by Kwak et al. (2010), where the authors compare CNN and Twitter. While they provide no quantitative analysis, the authors note that CNN mostly leads Twitter, but that some events, like sports matches and accidents, do break on Twitter sooner.

Zhao et al. (2011) compared Twitter and the New York Times (NYT) using topic models, LDA in particular. They found that both cover the same broad category of topics, but that the distribution of the topics is different. For example, they found that entity-oriented news dominates in Twitter, whereas event-oriented news dominates in NYT. Zhao et al. (2011) also found that there are topics that are covered only in the newswire (mostly world news), as well as those covered only in Twitter (mostly celebrity-related news). This indicates that combining news and Twitter could indeed be beneficial as there is the potential to capture a wider range of events than by using

either stream alone. In this section we conduct the first comparison of newswire and Twitter that takes time into account, which is crucial for any applications that rely on real-time properties of either stream.

7.3.1 Manually Determining the Lag

Like with Wikipedia, we first manually inspect the set of 27 reference events in order to establish their lag in the newswire. Unlike in Wikipedia, there is no need to perform any kind of preprocessing of the data as we have full texts of all the articles and we can establish if an article talks about an event with absolute certainty. We show the results of this experiment in Table 7.2. We can see that the situation here is quite different than in Wikipedia (cf. Table 7.1). First of all, two events were not found in the newswire stream, compared to six events that were not found in Wikipedia. Second, we can see that newswire leads Twitter on four events, compared to only one in Wikipedia. Lastly, we see that the lag in newswire is much lower than the lag in Wikipedia, with a median lag of 28 minutes. This is in line with the expectation that newswire is much more real-time, but it is interesting that Twitter still seems to lead in reporting breaking news most of the time.

7.3.2 Automatically Determining Lag Between News and Twitter

In this section we perform similar experiments as in Section 7.2.3 in order to automatically establish the lag between the newswire and Twitter streams. In the first experiment, we try to determine the lag between the events detected in Twitter and events detected in newswire. For Twitter, we use the same set of events as was used in Section 7.2.3 for determining the lag with respect to Wikipedia. For newswire, we run our FSD system on the whole newswire dataset to obtain the novelty scores, and then perform clustering as described in Section 6.1 with a threshold of 0.4, and we keep all the clusters. For each event detected in Twitter we then measure the cosine distance to the nearest event detected in newswire, where both Twitter and newswire events are represented by the centroid story. Like in Section 7.2.3 we measure the average distance and the standard deviation of the distances for all of the Twitter events. Results of this experiment for different values of the lag are shown in Figure 7.9. We can see that the minimum average distance and the maximum standard deviation are both achieved for a lag of zero hours. This again shows that the two streams are in sync and suggests that newswire might be more useful than Wikipedia for real-time event detection.

Event	Lag
Amy Winehouse dies	30m
Atlantis shuttle lands	2m
Betty Ford dies	46m
Richard Bowes killed in riots in England	7h 29m
Flight 4896 crash	19m
S&P downgrade US credit rating	25m
US increases debt ceiling	7m
Terrorist attack in Delhi	28m
Earthquake in Virginia	8m
First victim of London riots dies	36h 43m
War criminal Goran Hadzic arrested	-22h 3m
India and Bangladesh sign a border pact	n/a
Plane with Russian hockey team Lokomotiv crashes	50m
Explosion in French nuclear plant in Marcoule	19m
NASA announces there might be water on Mars	3m
Google announces plans to buy Motorola Mobility	-3m
Car bomb explodes in Oslo, Norway	18m
Gunman opens fire in youth camp in Norway	40m
First artificial organ transplant	-22m
Petrol pipeline explodes in Kenya	1h 48m
Famine declared in Somalia	3m
South Sudan becomes independent country	29m
South Sudan becomes UN member state	21m
Three men die in riots in England	2h 40m
Riots break out in Tottenham, England	40m
Rebels capture International Tripoli Airport	n/a
Ferry sinks in Zanzibar	-2m

Table 7.2: Lag between newswire and their counterparts in Twitter. Lag less than zero means that an event was reported in newswire first, while n/a means that the event was not mentioned in newswire.

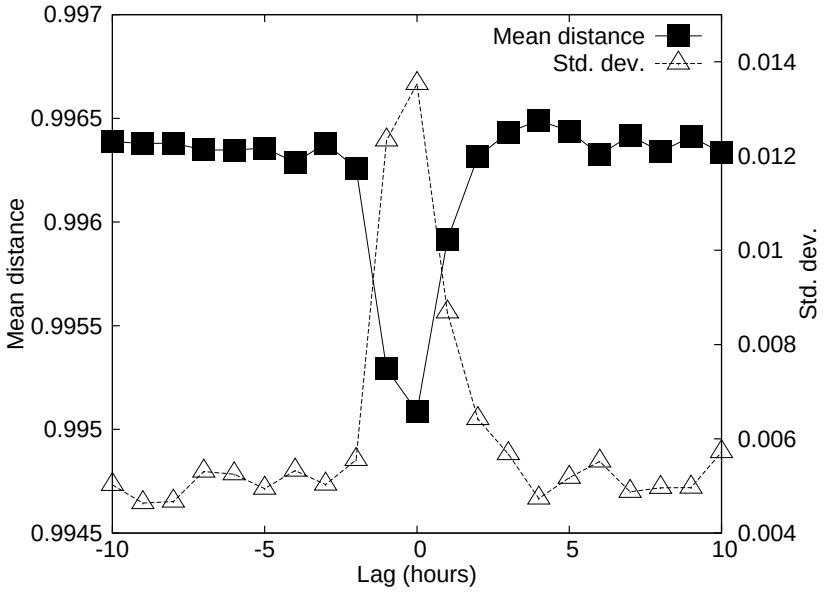

Figure 7.9: Mean and standard deviation of distances from Twitter first stories to the nearest first story in the newswire.

In the second experiment, we measure the cross-correlation between the two streams. For Twitter, we use the whole stream that we have available (over 51 million documents), whereas for newswire we take the headlines of all the documents in our dataset. We show the cross-correlation for the headlines because it has been shown that tweets have much more similar language to news headlines than to full texts (Subašić and Berendt, 2011). We also computed the cross-correlation between tweets and full newswire articles and got the same results.[7] We plot the correlation between the two streams in Figure 7.10. This figure confirms the results of the previous two experiments – the maximum correlation between two streams is achieved with zero lag. Because timestamps in newswire have a much finer resolution than the Wikipedia timestamps, we can measure the cross-correlation on a finer scale. Figure 7.11 shows the cross-correlation where the unit lag is ten minutes. We can still see that the maximum correlation is achieved for a lag of zero, which means that the lag between the two streams is less than ten minutes. Based on the results of the manual inspection, as well as the two experiments in this section, we conclude that the lag between Twitter and newswire is

[7]The only difference was that, because of very long Guardian articles, we had to normalize the cross-correlation by the number of words in each interval. Without this, we would essentially be measuring the lag between Guardian and Twitter.

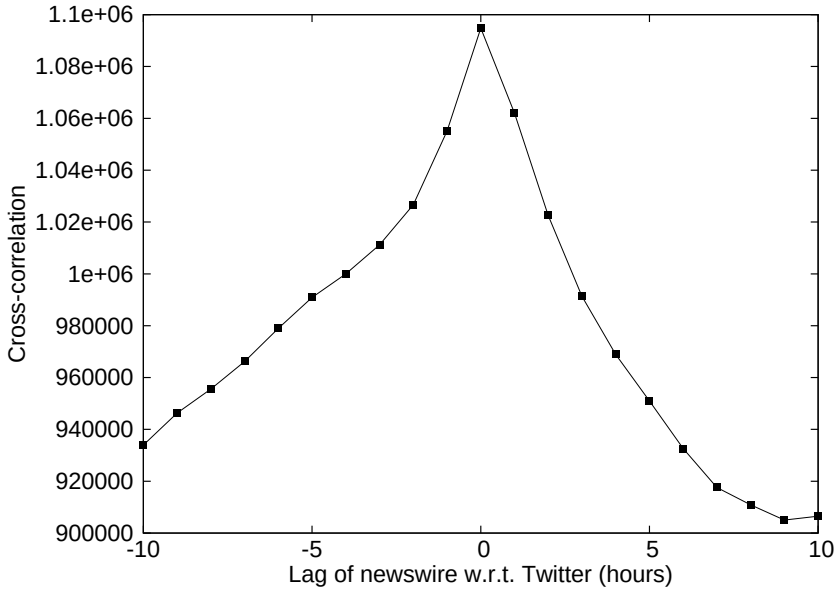

Figure 7.10: Cross-correlation between Twitter and newswire streams.

less than one hour, which is very promising from the perspective of using newswire for real-time event detection.

7.3.3 Can Twitter Help Newswire and Vice Versa?

In Section 7.2 we used Wikipedia page views as an additional source of information to help us reduce the amount of detected events that do not correspond to real events. In this section we will see if combining Twitter and newswire is at all helpful for either stream. We will focus on two questions: i) do both streams report the same set of events, and ii) do both streams break events at the same time? A negative answer to either of the two questions would show that combining the two streams is meaningful from the perspective of event detection.

First of all, why do we not use newswire in a similar manner to Wikipedia and try to reduce the amount of noise in events detected from Twitter? The reason for this is simple – if we have access to the newswire stream, this already provides us with a very high quality stream of events. In fact, the newswire stream has precision 100% (all the events reported there are actual events) with very high recall, so performing an intersection of Twitter and newswire, like we did with Wikipedia and Twitter, makes

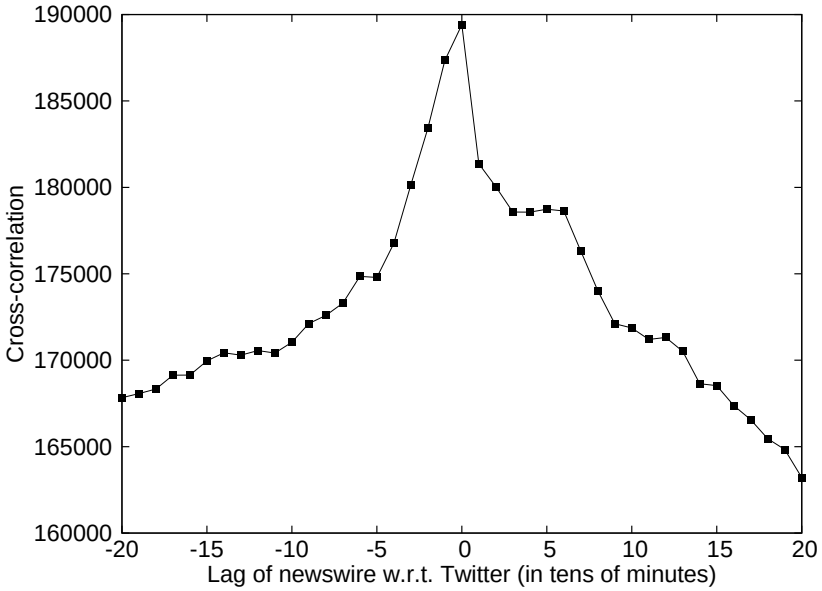

Figure 7.11: Cross-correlation between Twitter and newswire streams on a finer scale.

little sense as we can only reduce precision and recall. This then begs the question: if we do have the newswire stream, is Twitter at all helpful for the purpose of event detection? Or, equivalently, why would one be at all interested in detecting events from Twitter? This is a fundamental question, yet no previous research in event detection in social media has even tried to address it. In this section we will show that Twitter is indeed useful for event detection even if we have access to the newswire, and for several reasons: i) most importantly, it can reduce the latency of detecting the events in cases where news breaks on Twitter before, ii) there are events that are reported in Twitter, but not in newswire, and iii) Twitter serves as an aggregator of all the different newswire sources, which would be very difficult to keep track of individually. While point i) has been hinted at before in the literature, to the best of our knowledge, it was never tested. The best that one can find in the literature are examples of isolated events where news broke on Twitter sooner than in newswire, but it was never measured to which extent this is the case. Here we provide the first evidence that this is true on a large scale. To the best of our knowledge, point ii) was never addressed in the literature before and we are the first to provide evidence that there are events that appear in Twitter, but not in newswire. Point iii), while being a somewhat practical issue, is very important nevertheless. Keeping track of all the possible newswire sources is not

an easy task in practice because of the sheer number of possible sources. Depending on where the line is drawn as to what constitutes newswire (e.g., do tech blogs like Ars Technica or Mashable count as newswire?), there are thousands of websites that one would have to crawl in order to get a "full" newswire stream. On the other hand, Twitter integrates all of these sources into one, easy-to-consume stream which can serve as a replacement for the newswire.

7.3.3.1 Improving Twitter Detection with the Help of Newswire

We start off by focusing on Twitter and using newswire as additional information for improving the clustering of tweets into event clusters. In the next sections, we will look at other ways that Twitter can benefit from newswire, and also ways in which newswire can benefit from information in Twitter.

The main idea here is to use newswire to help us deal with brevity of tweets. As we noted in Chapter 5, the length of tweets can be a limiting factor because authors often have to focus on one aspect of the event, making tweets that discuss the same event seem very dissimilar. One way in which we addressed this problem was by using paraphrases to expand tweets. Here we use newswire articles to define additional terms for expansion. The main idea is to find a newswire article that discusses the same event as the tweet and use the terms in that article for expansion. Because we cannot be sure if the retrieved article talks about the same event or not, we simply use the cosine similarity to define a weighting on the terms in the article. Let t be the timestamp of tweet $\mathbf{x}_t$, and $\mathbf{y}_{nn}$ be the newswire article most similar to $\mathbf{x}_t$:

$$\mathbf{y}_{nn} = \underset{j \in \{1 \ldots t\}}{\arg\max} \cos(\mathbf{x}_t, \mathbf{y}_j). \tag{7.2}$$

Note that $\mathbf{y}_{nn}$ is chosen from the newswire articles written up to the time when tweet $\mathbf{x}_t$ arrived, which makes sure that we do not use information from the future. We expand the tweet $\mathbf{x}_t$ into $\mathbf{x'}_t$ as follows:

$$\mathbf{x'}_t = \mathbf{x}_t + \cos(\mathbf{x}_t, \mathbf{y}_{nn})\mathbf{y}_{nn}. \tag{7.3}$$

We then use the expanded tweet $\mathbf{x'}_t$ to perform which ever task is required, just as we would use the original tweet.

Using an external collection to expand queries is not a new idea – it is at least a decade old with Kwok et al. (2000) adding documents from an external collection to

the set of documents used for pseudo-relevance feedback. Diaz and Metzler (2006) introduced the idea of using external corpora in a mixture model and showed that it can improve the estimation of relevance models. More recently, Weerkamp et al. (2009) gave a general framework for using external corpora for expansion, showing, e.g., that the model of Diaz and Metzler (2006) is a special case of one of their models. What is new here is that i) we are the first to perform expansion for detection or FSD with an external collection, and ii) we study expansion in a streaming setting, where both the original collection and the external collection arrive incrementally, as opposed to previous work which assumed that both collections are static. This is also the reason why we do not use prior methods for query expansion in our experiments – these methods are batch and not designed with efficiency in mind. On the other hand, the approach we suggest in Equation 7.3, while simple, is incremental and amenable to integration with LSH, as it only uses information about similarity to the nearest neighbor.

We first try this approach on the Twitter FSD task. We found that the C_{min} cost increased from 0.69 to 0.75, indicating that this approach is not helpful for the FSD task. Upon closer inspection, we found that there are two main reasons for this: i) expanding the first story with a non-related newswire document makes this story look old, and ii) our document expansion makes first stories about planned events seem less novel. Problem i) occurs in cases where Twitter leads newswire, and thus there is no good document to expand with at the time when the first story in Twitter arrives. Therefore, the first story ends up being expanded with a non-related newswire story that was already reported in Twitter, making the first story seem less novel. For example, the first story about Amy Winehouse's death is expanded using the newswire article with the headline *Sunday Mirror phone-hacking claim* because the first newswire article about Amy's death arrived 30 minutes after the first tweet. Because the phone hacking scandal was already discussed in Twitter, this made the first story about Amy Winehouse's death look less novel. To show the extent of this problem, we ran the same experiment (expanding tweets with newswire), but with the newswire stream shifted one day into the past. This means that problem i) should be non-existing, as we are effectively looking one day into the future of the newswire stream. Doing this improved the C_{min} score from 0.75 to 0.67, showing that tardiness of the newswire stream with respect to Twitter is indeed problematic here.

To illustrate the second problem, consider the event *UN declares famine in Somalia*. This is a planned event, as it was speculated that this would happen for a long while before the famine was actually declared. The first tweet about this event in our

data is *#UN declares famine in two drought-hit areas #Somalia - southern Bakool & lower Shabelle*, but there were many prior tweets predicting that this will happen, e.g., *U.N. set to declare #famine in parts of #Somalia*. By performing expansion, tweets like these will only look more similar, making the true first story seem less novel. Because FSD evaluation is very sensitive to how well the first story is identified (given a novelty threshold, the miss rate in FSD can be either 0% or 100%), this means that reducing the novelty of first stories has a severe effect on the overall C_{min} score. Both of the problems we mentioned make first stories appear less novel, and it is therefore not surprising that performing such expansion in FSD is not helpful.

Despite the negative results in FSD, we posit that the overall quality of the clusters produced from the tweets expanded with newswire articles is improved. We test this claim by performing the detection task with and without the expansion. Like we mentioned before, detection is targeted at evaluating the quality of entire clusters, not just the first stories, and therefore measures exactly the quantity we are interested in.

In Chapter 5 we introduced a method for expanding tweets based on paraphrase information. This method improved the FSD score (albeit not much), suggesting that it should also be useful for the detection task. Therefore, we compare expansion using the newswire articles with this method. While news has been used before for expanding tweets in the context of event retrieval (Metzler et al., 2012), it was used as a static dataset, and the news articles were not from the same time period. In this experiment, we will also explore the effect that recency has on expansion using an external source like the newswire. To this end, we use the same newswire stream for expansion as in equation (7.3), but we introduce a lag of one day into this stream. This way we can test if recency of the external source has any effect on the efficacy of the expansion.

Results are shown in Figure 7.12. Comparing these results to those for detection in TDT5 (Table 6.2), it becomes immediately apparent that Twitter is a much more challenging domain than newswire. Several things can be seen from Figure 7.12. First, expanding tweets using either newswire or paraphrases is beneficial. Second, while using paraphrases only improved the FSD score by 2%, using them for detection improved the score by 8.5%, showing again how challenging the FSD task really is. Third, expansion using newswire outperforms expansion using paraphrases, probably because newswire introduces more relevant terms. Finally, we can see that recency does matter: introducing a lag of just one day into the newswire stream decreases the results by 3.5%. Overall, the cost for the best threshold is improved from 0.398 to 0.316, or 20.6%, when fresh newswire is used for expansion. This clearly shows that

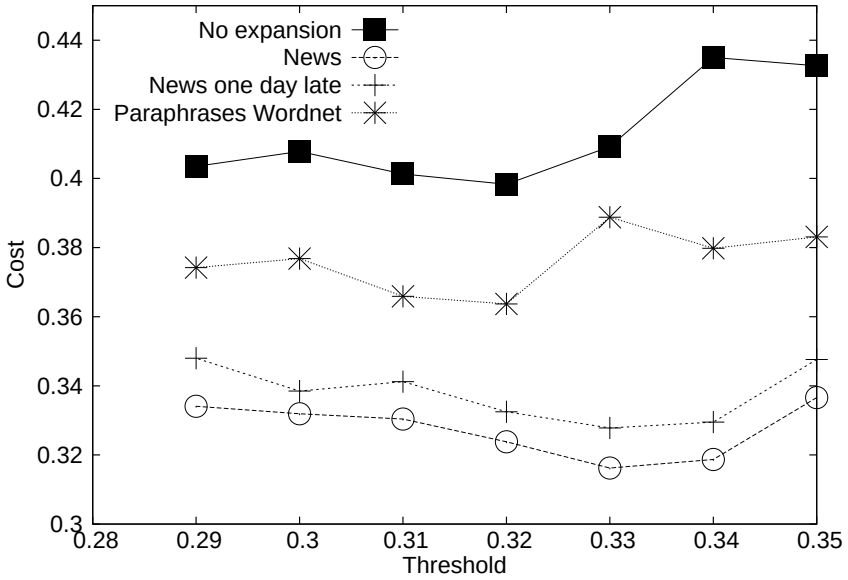

Figure 7.12: Twitter detection cost with and without expansion. Lower is better.

expanding the tweets using the information from newswire is beneficial and produces a better clustering. We note here that clustering tweets according to the events they discuss also has applications outside of event detection, e.g., event retrieval, identifying popular events, or finding out which users are interested in which events. To the best of our knowledge, this is the first work that uses an external corpus to expand tweets for the purpose of detection or first story detection. We are also the first to treat the external source as a stream and to show that recency in this case is beneficial.

7.3.3.2 How Many Newswire Events Are Reported in Twitter?

We now focus on the problem of determining how many events that are reported in the newswire are also reported in Twitter. We do this in order to determine if there is a bias in Twitter, i.e., if there are events that are not mentioned on Twitter for some reason. We would expect that all of the events reported in the newswire are also reported in Twitter, if for no other reason, then because every news organization has a Twitter account that tweets links to the articles they publish. In this section, we conduct experiments to test if this is true by looking for news events that are not reported in Twitter.

We first cluster the newswire stream using Algorithm 9 with a threshold of 0.4 and latency of $-\infty$, meaning that we do not impose a lifetime on the clusters. We

Iranian actor Pegah Ahangarani arrested in Teheran
Court upholds decision to impose control order on terror suspect in London
Support of Assad government shows signs of weakening
French socialist project 'sharing and caring' in bid to beat Nicolas Sarkozy
Dick Cheney autobiography heaps praise on Tony Blair

Table 7.3: News events not found in our Twitter data.

perform the same procedure on the Twitter stream, keeping all, even singleton clusters. We then align the clusters from the newswire and Twitter streams using the following simple procedure: for every newswire cluster, we find its nearest neighbor (in terms of the cosine similarity) in the stream of Twitter clusters. We will refer to the cosine similarity between the news cluster and its closest Twitter cluster as *pair similarity*. In both streams, clusters are represented by the sum of all document vectors in the cluster. Note that, while clustering either stream is not necessary, we do it in order to improve the running time of the algorithm for aligning the streams. By using clustering, the newswire stream is reduced four, and the Twitter stream is reduced two times in size.

Because news-tweet pairs whose similarity is lower are more likely to be "bad alignments", i.e., not refer to the same event, these pairs represent a good starting point for finding news events that are not found in Twitter. We choose a pair similarity score of 0.03 as a cutoff and keep only those pairs whose similarity is below this threshold. This threshold is arbitrary, but it does not affect our results, as we just need it to be low enough to make sure there are "bad alignments" in the data. Out of 27,000 newswire events in the data, 3,000 events had the pair similarity score lower than 0.03. Out of those 3,000 news-Twitter pairs we randomly sample 100 and label them according to whether these pairs actually refer to the same event or not. We found that 79 of those pairs do refer to the same event, whereas 21 did not. Because the alignment algorithm is not perfect, we further inspect the 21 news events manually. In particular, we manually search our dataset of 51 million tweets using keyword search, looking for mentions of those 21 events. Using the manual keyword search, we find that 16 of those events were mentioned in Twitter, whereas five were not mentioned at all in our Twitter data. The five events not found in our data are shown in Table 7.3.

This result suggests that around 5% of newswire events are not reported in our data. However, keeping in mind that we work with the 1% sample of full Twitter, it is very likely that the full Twitter stream contains all of the events mentioned in the

newswire. We performed a manual search over the full Twitter stream using Google and found that all five of the missing events were reported on Twitter, suggesting that this intuition is right. This experiment shows that we can effectively view newswire as a (cleaner) subset of the full Twitter stream, and also that even the small 1% stream contains around 95% of the events reported in the newswire.

7.3.3.3 How Many Twitter Events Are Reported in Newswire?

In this section we focus on the events that are reported in Twitter, but not in newswire. This should answer the question about whether Twitter is just a subset of newswire, or if it carries additional information not found in traditional media. To answer this question, we perform the following experiment. We take the potential events detected from the Twitter stream, and for each one we find the closest matching event in the newswire stream (exactly the opposite of what was done in Section 7.3.3.2). We then look for news-tweet pairs where the similarity between the Twitter cluster and the newswire cluster is lower than some threshold. This gives us a list of potential Twitter events that have no good match in the newswire stream, and are thus not reported in the newswire. However, a lot of these events will be noise, and we thus resort to the same procedure as when measuring precision of detected events: we sample n events and label them as corresponding to real events or not, which will give us an estimate of how many real events exist in Twitter that are not covered in the newswire.

We use two different lists of potential Twitter events: one obtained by using the *size* strategy with a size threshold of two (we will call this list *size*), and one obtained by using the Wiki spike strategy with a threshold of three (we will call this list *Wiki spike*). These two lists are quite different, as previous experiments showed. Using size strategy will give us higher recall, but lower precision of events, whereas using Wiki spike will give us lower recall but higher precision. We note here that by using the *Wiki spike* list, we are in effect combining three streams at once: Twitter is combined with Wikipedia to produce this cleaner list of events, and we align this with newswire in order to detect more events. By combining the *size* list with newswire and keeping all potential events whose closest match in newswire has a cosine similarity below 0.05,[8] we obtain 710,000 Twitter events that are not matched in the newswire. We randomly sample 500 of those events and label them as being real events or not. We find that 12 of those correspond to real events. When combining the *Wiki spike* list of

[8]This score is arbitrary and does not guarantee that all the alignments that have a score below 0.05 will be "bad". We will account for this in our experiments by manually inspecting the alignments.

	Wiki spike	Size
True events that passed the 0.05 filter	42	12
After removing duplicate events	37	12
After removing bad alignments	26	7
After removing events reported in other newswire sources	24	4

Table 7.4: Statistics about true events in the sample of 500 detected events that did not have a good match in the newswire.

potential Twitter events with newswire and keeping those events that had the newswire nearest neighbor with a cosine similarity below 0.05, we obtain 4,800 events. Again we sample 500 events and label them, leaving us with 42 real events. After obtaining the list of real events from the two samples, we need to perform additional pruning to make sure that those events are indeed not mentioned in the newswire. First, we remove any duplicate events. After that, because the score of 0.05 that we used for cutoff does not guarantee that the news-tweet pair is not related to the same event, we manually inspect the alignments and remove those pairs where the news item refers to the same event as the Twitter item. Because the alignment method itself is not perfect, i.e., it is possible that cosine similarity did not find the correct news item to align with the Twitter item, we also manually inspect our newswire data and remove cases where there exists a news article about the same event. Finally, we also remove all the Twitter events where the tweets were obviously posted by some other newswire source that is not present in our data (AP, CBS, Slashdot, etc.). In the end, this leaves 24 out of 42 Twitter events from the *Wiki spike* sample, and 4 out of 12 events from the *size* sample. This experiment is summarized in Table 7.4.

Based on the *Wiki spike* numbers, we estimate that 4.8% of events reported by the *Wiki spike* strategy, and not having a news item more similar than 0.05 cosine, represent true events that are not reported by the newswire. Given that there are 4,800 such events, 4.8% of that is 230 estimated events that were not reported in the newswire, but were reported on Twitter during the 80-day period in our data. The same calculations based on the *size* numbers suggest that there are 5,680 such events. These two numbers (230 and 5,680) give us an idea on the order of magnitude of the number of events that are reported on Twitter and not in newswire. Remember also that we are working only with a 1% sample of the full Twitter stream, so the actual numbers will likely be higher.

In Table 7.5 we show examples of the events that were reported on Twitter but not

Apparently looters broke into Derby County's trophy cabinet last night. Police are searching for 3 youths covered in cobwebs! #riots
RIP Rick Rypien. Sad to see another death in the NHL. Too many tragedies in the world lately...
UFC on Versus 5 results: Jacob Volkmann def. Danny Castillo via unanimous decision (29-28, 29-28, 29-28)
RT @NASA: NASA is ready to move forward with Space Launch System, a new capability for human exploration beyond Earth
Car reg NP05 LPU looting PC World Charlton. Retweet and shame.
RT @DerbysPolice: To reiterate rumours circulating there is disorder or looting in Derby city are untrue. Please RT. #derby #police

Table 7.5: Examples of events reported on Twitter, but not in our newswire data.

in the newswire. Out of the 28 events that passed our rigorous inspection, we note that 15 of these events were sports-related, while the rest were a mix of all other event types. This is in line with the findings of Kwak et al. (2010) where it was found that Twitter leads CNN mostly in sports events, and suggests that Twitter is a very good source of up-to-date sports news, probably because a lot of sports fans tweet about the games as they unfold. The fifth example in Table 7.5 shows another strength of Twitter. During the London riots, there were a lot of tweets about minor acts of violence that did not make it into the mainstream news, but Twitter served as a medium that carried all of these micro-events. The last example in Table 7.5 also shows how the law enforcement used Twitter to dispute rumours during the riots. This information would not make it into the newswire as it only had value for a very limited period of time, making Twitter the ideal medium to carry it.

7.3.3.4 Does News Really Break on Twitter Before it Does in Newswire?

In section 7.3.3.3 we showed that there are events that are not reported in the newswire. In this section we want to find out in how many cases, if any, Twitter breaks the news before newswire, out of those events reported in both streams. One of the main reasons Twitter is so popular for event detection is the fact that news sometimes breaks there sooner than in the newswire. However, all the literature on event detection in Twitter cites a handful of most famous examples, but the extent to which Twitter breaks the news before newswire does is not clear. There has also been no prior work that we

know of which tries to automatically identify events where Twitter leads newswire. The work in this section represents the first work that investigates this phenomenon on a large scale, beyond a small number of handpicked examples, and is also the first work to semi-automatically extract events where Twitter leads newswire.

In this experiment, we start with the same stream of news-tweet event pairs as in Section 7.3.3.2. However, unlike in Section 7.3.3.2, we are now interested in news-tweet pairs which do refer to the same event. Therefore, we keep all the pairs with an alignment score greater than 0.05. Out of 27 thousand news-tweet pairs, this leaves us with 13 thousand pairs, or about a half. We then remove all pairs where the event was reported in newswire sooner, leaving us with 5.5 thousand pairs. At this point, the 5.5 thousand out of 27 thousand total cases where Twitter leads mean that Twitter leads the RSS feeds of eight major newswire sources in approximately 20% of cases. We find that the median lead was 3209 seconds, or 53 minutes.

However, there are many caveats in the results we just presented. For one, we want to know if Twitter leads the true newswire stream, not just the RSS feed. We also want to generalize over more newswire sources, and there is also no guarantee that using a threshold of 0.05 yields good news-tweet alignments. This is why we perform a much more rigorous experiment where we use a series of filtering steps to remove news-tweet pairs that should not be considered as cases where Twitter broke the news about the event first. In this experiment we try to be as conservative as possible, always choosing to err on the side of reporting fewer cases where Twitter leads. By doing this we most likely remove some cases where Twitter does indeed lead, so the results we obtain here can be considered as a sort of a lower bound on the number of instances where Twitter leads newswire. We now explain the filtering steps used to remove news-tweet pairs.

Removing tweets with a link. We remove all the news-tweet pairs where the tweet contained a link. This is because posting a link in the tweet usually means that there is already an article somewhere on the web, indicating that the news was already reported, possibly in some other newswire source not present in our data. An example of a tweet that is removed by this filter is *A judge ends house arrest for former IMF leader Dominique Strauss-Kahn: http://t.co/zJgwyVZ.* This filter removed over 4.5 thousand pairs, leaving us with 725 pairs.

Removing pairs where the tweet mentions a newswire source. This is a simple, yet effective way to remove cases where the tweet is simply a repost of some newswire headline. An example of such a tweet is *[nytimes] Europe gives $17 Billion to Greece to Avoid Default: Euro zone finance ministers have decided to. . . .* We remove all pairs

where the tweet contains one of the following words: BBC, AP, NYT, CNN, Reuters, NYTimes, AFP, AJE, AAP, BBCWorld, or WashingtonPost. We also remove all tweets that contain the word *via*, which most often indicates that the tweet was read through some newswire source. After applying this filter, we were left with 580 pairs.

Remove all retweets. While a retweet does not necessarily mean that the news comes from newswire (e.g., it might be a retweet of a tweet that broke the news on Twitter), we choose to err on the side of being conservative about the pairs we keep and thus remove all pairs where the tweet is a retweet. This filter will capture cases where the news came from a less known newswire source or simply one not listed in the previous step. For example, the tweet *RT @nytimespolitics: In Ohio, a new governor is off to a smooth start* would be removed by this filter. After applying this filter, we were left with 391 pairs.

Remove bad alignments. Choosing a threshold of 0.05 does not guarantee that the news-tweet alignment will be correct, i.e., it does not guarantee that both items in the pair will refer to the same event. This is why we manually inspect the remaining 391 pairs and remove those where the tweet and the newswire article to not refer to the same event. As an example of a "bad" alignment, consider the pair where the news article was *Assistant Commissioner John Yates, who reviewed a Met police investigation into phone hacking, says he is 99% certain his own mobile was hacked.*, whereas the tweet that was aligned with this article was *John Yates, senior Metropolitan police officer who reviewed the 1st inquiry into phone hacking repeatedly lied to parliament / #MET #Police*. While the two stories are related, they do not refer to the same event. After applying this filter, we were left with 331 pairs.

Remove tweets copied from the newswire. Twitter users often pick up news from the newswire and then post the same text in the form of a tweet, but omitting the link to the original article and not giving credit to the newswire source that broke the news. This is why we inspect each of the remaining 331 pairs and remove those cases where it is obvious that the tweet is a copy of the news article. An example of such a case would be the news article *Author uncovers Lennon's letters: More than 250 letters and cards sent by John Lennon to family and friends are to be published for the first time by Beatles biographer Hunter Davies.*, whereas the matching tweet was *#InstantFollowBack #TeamFollowBack Author uncovers Lennon's letters #IfollowBack #TeamAutoFollow*. In this stage we also removed tweets that mentioned newswire sources which did not appear in our previous list of major newswire sources (e.g., we found tweets from Yahoo News and NBC). After this stage, there were 129 pairs left.

Remove duplicate events. Because the clustering on the newswire side is not perfect, some events appear twice in our list because they were assigned to multiple clusters. In this stage we remove all duplicate mentions of events, making sure that each event in our list is mentioned only once. There were 11 events mentioned more than once, leaving us with 118 unique events after this step.

Remove wrong lead. Another consequence of imperfect clustering on the newswire side is that, because some events are split into multiple clusters, we may find that Twitter leads one of the later mentions of the event. This problem is illustrated in Figure 7.13. Let some event E be split into three clusters N_1, N_2, and N_3 in the newswire, and let the same event be split into two clusters, T_1 and T_2, in Twitter. If we align N_1 with T_2, and N_2 with T_1, it will appear like Twitter is leading event E because T_1 leads N_2, and we have no way of knowing that N_1 and N_2 correspond to the same event. Because of this, we manually inspect the full newswire data and realign each remaining Twitter cluster with the earliest newswire cluster that talks about the same event. In Figure 7.13, this is equivalent to realigning T_1 with N_1. This will cause some pairs to be dropped from our list, as it will become apparent that the newswire article really leads Twitter, not the other way around. After this step, we were left with 97 pairs.

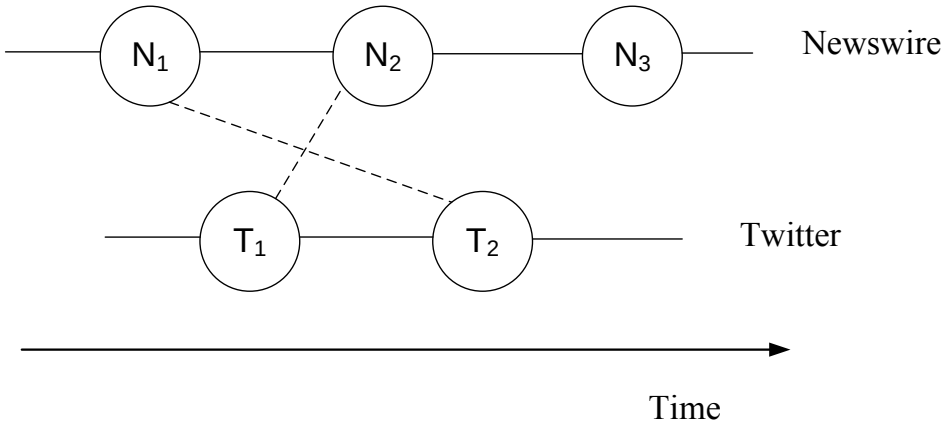

Figure 7.13: "Wrong lead" problem. Even though event is reported first in cluster N_1, Twitter seems to lead because cluster N_1 is aligned with cluster T_2, and cluster N_2 is aligned with cluster T_1.

All of these steps are summarized in Table 7.6. Number for each filter is obtained after applying it along with all the previous filters. We can see that removing tweets with links removes around 85% of all pairs, while the other steps remove a smaller number of pairs. In the end, we are left with 97 news-tweet pairs for which we are

Filter	Pairs left
No filter	5,471
Removed links	725
Removed newswire source mentions	580
Removed retweets	391
Removed bad alignments	331
Removed copied from newswire	129
Removed duplicate events	118
Removed "wrong lead"	97

Table 7.6: Summary of filtering steps for extracting events where Twitter leads.

fairly certain that they refer to events that broke on Twitter sooner than in the newswire. Examples of these events are shown in Table 7.7. Because there are around 27 thousand events in our newswire data, this means that in our dataset Twitter leads newswire in 0.4% of cases. This estimate is a lower bound – some of the filtering steps removed true cases where Twitter lead newswire, and there are less than 27 thousand events in the newswire because some events were split into multiple clusters.

In order to make sense of what kinds of events break on Twitter before they do in the newswire, we categorize the 97 events into seven broad categories: politics, sports, disasters & accidents, business & economy, entertainment, technology, and other. These categories are fairly self-explanatory, with perhaps the exception of disasters & accidents which, along with natural disasters and accidents contained events like terrorist attacks or shootings. The breakdown of these events is shown in Figure 7.14. Perhaps not surprisingly, we can see that a lot of events when Twitter leads are sports events (one third) and disaster-related events. However, it is interesting to note that a lot of cases when Twitter leads represent politics and business events, which we would expect the newswire to carry first.

Finally, we inspect by *how much* Twitter leads in these 97 events. First, we find that the median lead in the 97 events is 4,980 seconds, or about one and a half hours. This is of course not conclusive evidence that Twitter actually leads by this much because of the problems with our timestamps as discussed earlier.[9] However, when we removed tweets copied from the newswire, this also provided us with data to estimate the possi-

[9]It does mean that for those cases Twitter leads by one and a half hours over the RSS feeds of the major newswire sources.

Magnitude 5.4 earthquake hits western Japan
Rapper Lil Wayne ends up in hospital after a skateboarding accident
Malaysian police use tear gas on protesters
Baidu senior VP resigns
Sherwood Schwartz dies
Thor Hushovd wins stage 13 of Tour de France
Japan wins FIFA Women's World Cup
Michele Bachmann wins Iowa straw poll

Table 7.7: Examples of events where Twitter leads newswire.

ble error in our timestamps. We can use the 202 pairs of news-tweets where the tweets were the same as newswire article, only with an earlier timestamp, to estimate by how much our RSS feed timestamps are off. Calculating the median of the differences between the newswire timestamp and the Twitter timestamp, we find it is 2,889 seconds, or about 48 minutes. This suggests that our RSS timestamps are on average about 48 minutes late after the actual article publication time. Given that Twitter was leading by about one and a half hours, this would still suggest that Twitter is leading by about 40 minutes. We note again that this is not conclusive proof about the amount of time by which Twitter leads, but it does give us an indication about the order of magnitude.

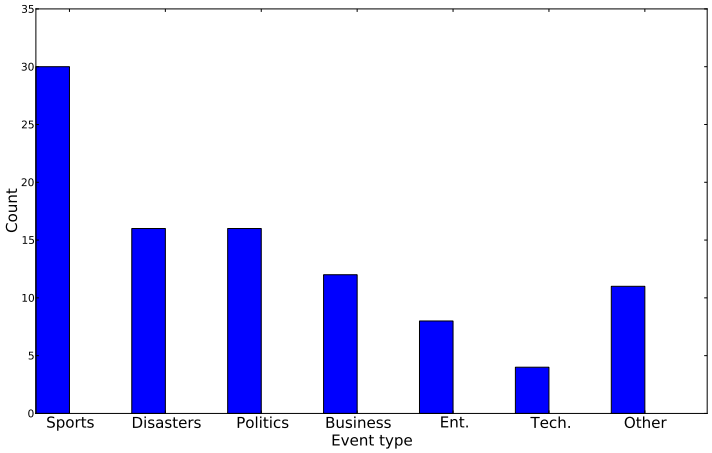

Figure 7.14: Breakdown of events where Twitter leads into broad categories.

7.4 Conclusion

In this chapter, we showed that combining information from multiple streams is helpful for event detection. We used Wikipedia page view data as an additional information source to improve the quality of detected events. We first showed that the page view data lags behind Twitter by about one hour. We then presented two novel methods for combining the information from Wikipedia with Twitter and showed that they can be used to significantly improve the precision of detected events for low recall levels over the simple method that just uses burstiness.

We also explored combining Twitter with the newswire stream. We first showed that we can expand tweets using terms found in related newswire articles, which leads to improved clustering of the tweets. We explored the time relationship between the two streams and showed that the newswire RSS feed is time-aligned with Twitter, i.e., that on average, neither stream leads or lags. We confirmed that all of the events in newswire are reported in Twitter, and that even the 1% Twitter stream contains about 95% of the events reported in the newswire. However, we found that the reverse is not true – there are many events reported in Twitter that are not reported in the newswire, like outcomes of sports events or events that are of interest only for a very short while. Finally, we showed that there are many cases where Twitter leads the RSS feed of newswire, and that, although not many, there are also cases where Twitter leads genuine newswire. Our experiments show that combining Twitter and newswire can be beneficial for both streams, as we would be able to identify more events and with a lower latency than is possible with either stream alone.

Chapter 8

Conclusion

This thesis presented original work on streaming algorithms for detecting events from unbounded streams of text, such as those typical of social media. There are four main contributions of this thesis. First, we presented algorithms for scaling traditional approaches so that they handle unbounded streams. We then presented a way of improving state-of-the-art performance in this task by incorporating paraphrase information into our approach. We next looked at ways of reducing the noise present in social media streams by allowing for some small latency in detecting events and by combining evidence from Twitter and Wikipedia streams. Finally, we showed that combining Twitter and newswire can be beneficial for several reasons. First, newswire can be used to improve clustering of tweets according to events they discuss. Second, there are cases where Twitter breaks news of the event before newswire does, and lastly, there are events reported in Twitter that are not mentioned in the newswire. The last two points show that event detection in Twitter is a meaningful task, even if one has access to the newswire stream.

In a world where almost everyone has a phone with a camera and an internet connection, the rate at which data is produced is increasing, and the lag with which people report events in only getting smaller. This will lead to problems like event detection in social media streams becoming more and more important, as we start to receive information directly from the source, instead of through an intermediary like the newswire. The work presented in this thesis provides a foundation for modern event detection systems that will have to deal with these huge amounts of data and high levels of noise typical of social media streams.

Applications of our work range from journalism, where users are interested in organizing the incoming stream of documents into topically related clusters, stock market

where traders are interested in finding about novel information with minimal lag, to homeland security and emergency response, where one is typically interested in detecting specific types of events, like riots or earthquakes. The work presented here also has impact outside of the field of event detection. Our variance reduction strategy combined with an approximate nearest neighbor search technique can be used for fast outlier detection in large datasets, while the work on combining paraphrases with LSH could help reduce the effects of vocabulary mismatch in other large-scale IR applications, e.g., large-scale ad-hoc retrieval. Finally, our analysis of the relationship between Twitter, Wikipedia, and newswire is the first comparison of these sources that takes their time properties into account, and we show that we can use the respective strengths of these streams to improve performance beyond using just one.

While we primarily focused on Twitter in this thesis, our models are general and we also tested them on newswire. Because there is nothing specific to Twitter in our approach, it is readily usable for other social media streams such as blogs or forums.

There are several ways in which the work in this thesis could be extended. First, dealing with noise could be addressed from the perspective of supervised learning, where the main challenges lie in choosing an efficient/incremental learning algorithm, using a minimal amount of expensive training data, and addressing the potential need for retraining. A different direction could be looking into and scaling many other document expansion techniques in order to deal with the brevity of documents in Twitter.

Bibliography

Agarwal, M. K., Ramamritham, K., and Bhide, M. (2012). Real time discovery of dense clusters in highly dynamic graphs: Identifying real world events in highly dynamic environments. *Proceedings of the VLDB Endowment*, 5(10):980–991.

Ahmed, A., Ho, Q., Eisenstein, J., Xing, E., Smola, A., and Teo, C. H. (2011). Unified analysis of streaming news. In *Proceedings of the 20th International Conference on World Wide Web*, pages 267–276. ACM.

Ahn, B. G., van Durme, B., and Callison-Burch, C. (2011). Wikitopics: What is popular on Wikipedia and why. In *Proceedings of the Workshop on Automatic Summarization for Different Genres, Media, and Languages*, pages 33–40. Association for Computational Linguistics.

Allan, J. (2002). *Topic detection and tracking: event-based information organization*. Kluwer Academic Publishers.

Allan, J., Lavrenko, V., and Jin, H. (2000a). First story detection in TDT is hard. In *Proceedings of The 21st ACM International Conference on Information and Knowledge Management*, pages 374–381. ACM.

Allan, J., Lavrenko, V., Malin, D., and Swan, R. (2000b). Detections, bounds, and timelines: UMass and TDT-3. In *Proceedings of Topic Detection and Tracking Workshop*, pages 167–174.

Allan, J., Yang, Y., Carbonell, J., Yamron, J., Doddington, G., and Wayne, C. (1998). TDT pilot study corpus. Catalog no. LDC98T25.

Analytics, P. (2009). Twitter study. http://www.pearanalytics.com/blog/wp-content/uploads/2010/05/Twitter-Study-August-2009.pdf.

Andoni, A. and Indyk, P. (2006). Efficient algorithms for substring near neighbor problem. In *Proceedings of the 17th annual ACM-SIAM symposium on Discrete algorithms*, pages 1203–1212. ACM.

Andoni, A. and Indyk, P. (2008). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122.

Becker, H., Iter, D., Naaman, M., and Gravano, L. (2012). Identifying content for planned events across social media sites. In *Proceedings of the 5th ACM international conference on Web search and data mining*, pages 533–542. ACM.

Becker, H., Naaman, M., and Gravano, L. (2011a). Beyond trending topics: Real-world event identification on Twitter. In *Proceedings of the 5th International Conference on Weblogs and Social Media*, pages 438–441. The AAAI Press.

Becker, H., Naaman, M., and Gravano, L. (2011b). Selecting quality Twitter content for events. In *Proceedings of the 5th International Conference on Weblogs and Social Media*, pages 442–445. The AAAI Press.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Brants, T., Chen, F., and Farahat, A. (2003). A system for new event detection. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 330–337. ACM.

Braun, R. K. and Kaneshiro, R. (2004). Exploiting topic pragmatics for new event detection in TDT-2004. Technical report, National Institute of Standards and Technology.

Broder, A. Z. (1997). On the resemblance and containment of documents. In *Proceedings of Compression and Complexity of Sequences*, pages 21–29. IEEE Computer Society.

Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G. (1997). Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8-13):1157–1166.

Callison-Burch, C. (2008). Syntactic constraints on paraphrases extracted from parallel corpora. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 196–205. Association for Computational Linguistics.

Callison-Burch, C., Koehn, P., and Osborne, M. (2006). Improved statistical machine translation using paraphrases. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 17–24. Association for Computational Linguistics.

Cataldi, M., Caro, L. D., and Schifanella, C. (2010). Emerging topic detection on Twitter based on temporal and social terms evaluation. In *Proceedings of the 10th International Workshop on Multimedia Data Mining*, pages 4:1–4:10. ACM.

Charikar, M. S. (2002). Similarity estimation techniques from rounding algorithms. In *Proceedings of the 34th annual ACM symposium on Theory of computing*, pages 380–388. ACM.

Ciglan, M. and Nørvåg, K. (2010). Wikipop: Personalized event detection system based on Wikipedia page view statistics. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1931–1932. ACM.

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46.

Cohn, T., Callison-Burch, C., and Lapata, M. (2008). Constructing corpora for the development and evaluation of paraphrase systems. *Computational Linguistics*, 34(4):597–614.

Cordeiro, M. (2012). Twitter event detection: Combining wavelet analysis and topic inference summarization. In *Doctoral Symposium in Informatics Engineering*, pages 123–138.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Crouch, C. J. and Yang, B. (1992). Experiments in automatic statistical thesaurus construction. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 77–88. ACM.

Datar, M., Immorlica, N., Indyk, P., and Mirrokni, V. (2004). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th annual symposium on Computational geometry*, pages 253–262. ACM.

Diaz, F. and Metzler, D. (2006). Improving the estimation of relevance models using large external corpora. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 154–161. ACM.

van Durme, B. and Lall, A. (2009). Streaming pointwise mutual information. In *Advances in Neural Information Processing Systems 22*, pages 1892–1900.

van Durme, B. and Lall, A. (2010). Online generation of locality sensitive hash signatures. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 231–235. Association for Computational Linguistics.

Fellbaum, C. (1998). *WordNet: An electronic lexical database*. The MIT press.

Fiscus, J. (2001). Overview of results (NIST). In *Proceedings of the TDT 2001 Workshop*.

Fiscus, J. G. and Doddington, G. R. (2002). Topic detection and tracking evaluation overview. *Topic detection and tracking: event-based information organization*, pages 17–31.

Fox, E. A., Nutter, J. T., Ahlswede, T., Evens, M., and Markowitz, J. (1988). Building a large thesaurus for information retrieval. In *Proceedings of the 2nd conference on Applied natural language processing*, pages 101–108. Association for Computational Linguistics.

Genc, Y., Sakamoto, Y., and Nickerson, J. V. (2011). Discovering context: Classifying tweets through a semantic transform based on Wikipedia. *Foundations of Augmented Cognition. Directing the Future of Adaptive Systems*, pages 484–492.

Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529. Morgan Kaufmann Publishers Inc.

Goyal, A., Jagarlamudi, J., Daumé III, H., and Venkatasubramanian, S. (2010). Sketch techniques for scaling distributional similarity to the web. In *Proceedings of the Workshop on GEometrical Models of Natural Language Semantics*, pages 51–56. Association for Computational Linguistics.

Grubbs, F. E. (1969). Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21.

Hu, M., Sun, A., and Lim, E.-P. (2008). Event detection with common user interests. In *Proceedings of the 10th ACM workshop on Web information and data management*, pages 1–8. ACM.

Indyk, P. and Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th annual ACM symposium on Theory of computing*, pages 604–613. ACM.

Jones, R., Rey, B., Madani, O., and Greiner, W. (2006). Generating query substitutions. In *Proceedings of the 15th International Conference on World Wide Web*, pages 387–396. ACM.

Jurgens, D. and Stevens, K. (2009). Event detection in blogs using temporal random indexing. In *Proceedings of the Workshop on Events in Emerging Text Types*, pages 9–16. Association for Computational Linguistics.

Kriegel, H.-P., Kröger, P., and Zimek, A. (2009). Outlier detection techniques. Tutorial at the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining.

Krovetz, R. (1993). Viewing morphology as an inference process. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 191–202. ACM.

Kulis, B. and Grauman, K. (2009). Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of 12th International Conference on Computer Vision*, pages 2130–2137. IEEE.

Kumaran, G. and Allan, J. (2005). Using names and topics for new event detection. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 121–128. Association for Computational Linguistics.

Kwak, H., Lee, C., Park, H., and Moon, S. (2010). What is Twitter, a social network or a news media? In *Proceedings of the 19th International Conference on World Wide Web*, pages 591–600. ACM.

Kwok, K.-L., Grunfeld, L., Dinstl, N., and Chan, M. (2000). TREC-9 cross language, web and question-answering track experiments using PIRCS. In *TREC*.

Lavrenko, V. (2004). *A generative theory of relevance*. PhD thesis, University of Massachusetts.

Lavrenko, V., Allan, J., DeGuzman, E., LaFlamme, D., Pollard, V., and Thomas, S. (2002). Relevance models for topic detection and tracking. In *Proceedings of the 2nd international conference on Human Language Technology Research*, pages 115–121. Morgan Kaufmann Publishers Inc.

Lavrenko, V. and Croft, B. W. (2001). Relevance based language models. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 120–127. ACM.

Levenberg, A., Callison-Burch, C., and Osborne, M. (2010). Stream-based translation models for statistical machine translation. In *Human Language Technologies: The 11th Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 394–402. Association for Computational Linguistics.

Levenberg, A. and Osborne, M. (2009). Stream-based randomised language models for SMT. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 756–764. Association for Computational Linguistics.

Li, C., Sun, A., and Datta, A. (2012a). Twevent: Segment-based event detection from tweets. In *Proceedings of ACM Conference on Information and Knowledge Management*. ACM.

Li, R., Lei, K. H., Khadiwala, R., and Chang, K. C.-C. (2012b). TEDAS: A Twitter-based event detection and analysis system. In *Proceedings of 28th International Conference on Data Engineering*, pages 1273–1276. IEEE Computer Society.

Lin, J. and Dyer, C. (2010). *Data-Intensive Text Processing with MapReduce*. Morgan & Claypool Publishers.

Luo, G., Tang, C., and Yu, P. S. (2007). Resource-adaptive real-time new event detection. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 497–508. ACM.

Madnani, N. and Dorr, B. (2010). Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, 36(3):341–387.

Mathioudakis, M. and Koudas, N. (2010). Twittermonitor: Trend detection over the Twitter stream. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1155–1158. ACM.

Medelyan, O., Milne, D., Legg, C., and Witten, I. H. (2009). Mining meaning from Wikipedia. *International Journal of Human-Computer Studies*, 67(9):716–754.

Metzler, D., Cai, C., and Hovy, E. (2012). Structured event retrieval over microblog archives. In *Proceedings of Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics*, pages 646–655. Association for Computational Linguistics.

Muthukrishnan, S. M. (2005). Data streams: algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2):117–236.

O'Connor, B., Balasubramanyan, R., Routledge, B. R., and Smith, N. A. (2010). From tweets to polls: Linking text sentiment to public opinion time series. In *Proceedings of the 4th International Conference on Weblogs and Social Media*, pages 122–129. The AAAI Press.

Orfanidis, S. J. (1996). *Optimum signal processing: An introduction*. Macmillan New York.

Osborne, M., Petrović, S., McCreadie, R., Macdonald, C., and Ounis, I. (2012). Bieber no more: First story detection using Twitter and Wikipedia. In *Proceedings of the SIGIR workshop on Time-Aware Information Access*.

Ozdikis, O., Senkul, P., and Oguztuzun, H. (2012). Semantic expansion of hashtags for enhanced event detection in Twitter. In *Proceedings of the 1st International Workshop on Online Social Systems*.

Papka, R., Allan, J., and Lavrenko, V. (1999). UMass approaches to detection and tracking at TDT2. In *DARPA: Broadcast News Workshop*, pages 111–116.

Petrović, S., Osborne, M., and Lavrenko, V. (2010). Streaming first story detection with application to Twitter. In *Proceedings of the 11th annual conference of the North American Chapter of the Association for Computational Linguistics*, pages 181–189. Association for Computational Linguistics.

Petrović, S., Osborne, M., and Lavrenko, V. (2012). Using paraphrases for improving first story detection in news and Twitter. In *Proceedings of Human Language Technologies: Conference of the North American Chapter of the Association for Computational Linguistics*, pages 338–346. Association for Computational Linguistics.

Phuvipadawat, S. and Murata, T. (2010). Breaking news detection and tracking in Twitter. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, pages 120–123. IEEE Computer Society.

Popescu, A.-M. and Pennacchiotti, M. (2010). Detecting controversial events from Twitter. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 1873–1876. ACM.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, 14(3):130–137.

Qiu, Y. (1995). *Automatic query expansion based on a similarity Thesaurus*. PhD thesis, ETH Zürich.

Quirk, C., Brockett, C., and Dolan, W. (2004). Monolingual machine translation for paraphrase generation. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 142–149. Association for Computational Linguistics.

Ravichandran, D., Pantel, P., and Hovy, E. (2005). 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*, pages 622–629. Association for Computational Linguistics.

Riezler, S., Vasserman, A., Tsochantaridis, I., Mittal, V., and Liu, Y. (2007). Statistical machine translation for query expansion in answer retrieval. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 464–471. Association for Computational Linguistics.

Sakaki, T., Okazaki, M., and Matsuo, Y. (2010). Earthquake shakes Twitter users: real-time event detection by social sensors. In *Proceedings of the 19th International Conference on World Wide Web*, pages 851–860. ACM.

Salakhutdinov, R. and Hinton, G. (2009). Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978.

Sankaranarayanan, J., Samet, H., Teitler, B. E., Lieberman, M. D., and Sperling, J. (2009). Twitterstand: news in tweets. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 42–51. ACM.

Shapiro, S. and Wilk, M. (1965). An analysis of variance test for normality (complete samples). *Biometrika*, 52(3/4):591–611.

Spärck Jones, K. and Tait, J. (1984). Automatic search term variant generation. *Journal of Documentation*, 40(1):50–66.

Subašić, I. and Berendt, B. (2011). Peddling or creating? Investigating the role of Twitter in news reporting. *Advances in Information Retrieval*, pages 207–213.

Terasawa, K. and Tanaka, Y. (2007). Spherical LSH for approximate nearest neighbor search on unit hypersphere. *Algorithms and Data Structures*, pages 27–38.

Tsatsaronis, G. and Panagiotopoulou, V. (2009). A generalized vector space model for text retrieval based on semantic relatedness. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 70–78. Association for Computational Linguistics.

Ture, F., Elsayed, T., and Lin, J. (2011). No free lunch: Brute force vs. locality-sensitive hashing for cross-lingual pairwise similarity. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information*, pages 943–952. ACM.

Twitter (2010). State of Twitter spam. http://blog.twitter.com/2010/03/state-of-twitter-spam.html.

Wallis, P. (1993). Information retrieval based on paraphrase. In *Proceedings of the 1st Pacific Association for Computational Linguistics Conference*.

Weber, R., Schek, H.-J., and Blott, S. (1998). A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases*, pages 194–205. Morgan Kaufmann Publishers Inc.

Weerkamp, W., Balog, K., and de Rijke, M. (2009). A generative blog post retrieval model that uses query expansion based on external collections. In *Proceedings of the*

Joint conference of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, pages 1057–1065. Association for Computational Linguistics.

Weng, J., Yao, Y., Leonardi, E., and Lee, F. (2011). Event detection in Twitter. In *Proceedings of the 5th International Conference on Weblogs and Social Media*, pages 401–408. The AAAI Press.

Wong, S. K. M., Ziarko, W., and Wong, P. C. N. (1985). Generalized vector spaces model in information retrieval. In *Proceedings of the 8th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 18–25. ACM.

Yang, Y., Pierce, T., and Carbonell, J. (1998). A study of retrospective and on-line event detection. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 28–36. ACM.

Yardi, S., Romero, D., Schoenebeck, G., and dannah boyd (2009). Detecting spam in a Twitter network. *First Monday*, 15(1).

Zhang, J., Ghahramani, Z., and Yang, Y. (2005). A probabilistic model for online document clustering with application to novelty detection. In *Advances in Neural Information Processing Systems 17*.

Zhao, W. X., Jiang, J., Weng, J., He, J., Lim, E.-P., Yan, H., and Li, X. (2011). Comparing Twitter and traditional media using topic models. *Advances in Information Retrieval*, pages 338–349.