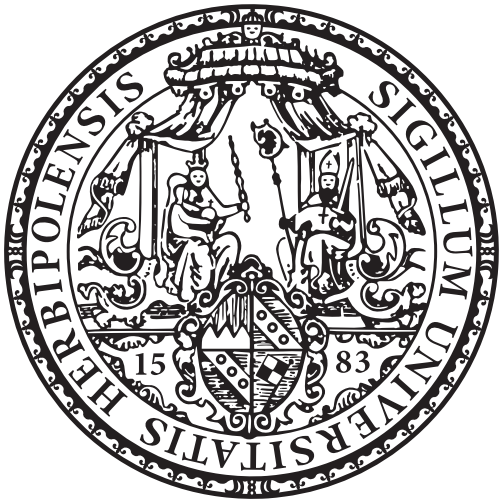
UNIVERSITÄT WÜRZBURG

INSTITUT FÜR INFORMATIK

LEHRSTUHL FÜR KÜNSTLICHE INTELLIGENZ
UND ANGEWANDTE INFORMATIK



Evaluation of Event Detection Systems in Twitter

Masterarbeit

Vorgelegt von: Gerhard Grimm
Karl-Philipp-Straße 21
91626 Schopfloch

Datum der Abgabe: 25.08.2014

Betreuer: Prof. Dr. A. Hotho
Dipl.-Inf. M. Becker

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Würzburg, 25.08.2014

_____

(Gerhard Grimm)

Abstract

Microblogging services, like Twitter, have become widely used tools for sharing opinions, communicating with friends or broadcasting news and thus grown to powerful sources of information of all kind. Subsequently, the application of event detection on this immense and continuous flow of information is promising and can lead to fully automatic news detection in real-time, but it is not a trivial task. While the detection of events in traditional media, like newswire, has long been a topic of interest, Twitter's immense scale of data, the streaming environment as well as the low quality of big parts of Twitter's messages force adaptions to be made in the process of event detection. Tackling this challenge, promising algorithms were proposed, however, these systems are rarely evaluated and compared in performance properly, which can be referred to the lack of suitable corpora for a long time. Therefore, this thesis picks up and introduces three state-of-the-art systems for event detection in Twitter as well as suitable corpora that recently were created. We implement, evaluate and compare these event detection techniques on the biggest existing, publicly available and most suitable corpus and discuss the results.

Contents

1 Introduction

Within the last years, microblogging services, like Twitter, have become widely used tools for sharing opinions, communicating with friends or broadcasting news and thus grown to powerful sources of information of all kind. Twitter is conceivably the most popular microblogging platform in the world; more than 500 million tweets are posted per day [Reh12]. Subsequently, the increasing interest in event detection systems which are suitable for Twitter is understandable. Twitter's users share and exchange information on real-world events as they occur. For instance, the death of Osama bin Laden was first reported on Twitter and thus informed millions of people before the official announcement [HLW+12]. In contrast to other media sources, tweets contain timely and fine-grained information about any kind of event and yield valuable information, which reflect personal perspectives, social information, conversational aspects, emotional reactions and controversial opinions [AK13]. Thus, the application of event detection on this immense and continuous flow of information is promising and can lead to fully automatic news detection in real-time, which, for example, would be very useful for business or intelligence analysts [POL12]. Especially Twitter's short-message structure (i.e., a tweet is limited to a maximum of 140 characters), which leads to fast and summarised information, as well as the real-time flow nature make Twitter a preferable source for event detection.

While the detection of events in traditional media (e.g. newswire) has long been a topic of interest [YPC98], Twitter's immense scale of data generated and the given streaming environment force adaptations to be made. Twitter's messages often show low quality, containing typographical errors, bad grammar, abbreviations, mixed language and/or user specific terms, which has to be considered in event detection. The high flood of so called noise in Twitter's data [POL10] (i.e., about 40% are meaningless babbles and 37% conversational messages [Ana09]) also has to be addressed. Furthermore, the shortness of the messages leads to poor performance of existing algorithms due to feature sparsity [LWC+11]. The basic underlying assumption for an event happening is an increase in event specific keywords in a given time interval [YPC98], which can be monitored using text retrieval and clustering techniques. For example the keyword "iPhone" is supposed to occur frequently when Apple releases a new version

of their smartphone, however, "good morning" also shows strong bursts. Thus, it is not sufficient to just observe bursting keywords and infer that an event has been detected. In order to carry out valuable event detection in Twitter, the differentiation of real events from the trivial ones has to be done, which existing algorithms largely fail to do [WYLL11].

Recently, several researchers have focused on this challenge. While promising algorithms were proposed [SST+09, PM10, POL10, BNG11, WYLL11, LWC+11, AS12, Cor12], detailed evaluations and performance comparisons with other systems are rarely done, which leaves the reader with considerable ambiguity in finding the best system for this very specific task. In most cases, the authors evaluated their systems by assessing the detection results on a self-collected, non-public and unannotated Twitter corpus or just used a corpus from another medium, which prevents a general rating and comparison of the system's performance on Twitter. Furthermore, these evaluations usually were limited to only high-volume events detected by their systems [POL12]. Detailed evaluations on a common and public corpus are urgently needed in order to objectively assess and compare the systems.

The main reason for this lack of useful corpora is the massive scale of Twitter, which makes the creation of corpora difficult, time-consuming and expensive [MMJ13]. However, the creation of such corpora has become highly necessary and recently a few were proposed [BNG11, POL12, MMJ13]. Nonetheless, detailed evaluations and comparisons of event detection techniques in Twitter have still not yet been done.

We tackle this challenge as the main goal of this thesis. Therefore, we pick, introduce and implement three state-of-the-art systems for event detection in Twitter [POL10, SST+09, AS12] with a clear focus on systems that seem to be applicable on large-scale corpora. Further, we analyse the existing corpora and choose the most convincing one for our evaluation. We evaluate the event detection techniques on the proposed corpus using typical evaluation measures in the information retrieval task and assess the results.

2 Structure of the Thesis

This thesis is organized as follows. Chapter 3 provides necessary background information; we define an event, introduce event detection in general and focus on the more special task of detecting events in Twitter. Subsequently, chapter 4 presents related work and three systems for event detection in Twitter are chosen. Chapter 5 continues with a detailed description of these systems. In chapter 6, we investigate in existing Twitter corpora as well as evaluations of event detection systems in Twitter that have been carried out. Chapter 7 starts with a short review of the Twitter corpus we chose and explains how we obtained the data. Subsequently, in order to conduct the evaluation, we introduce necessary postprocessing steps and present the evaluation measures we apply. Next, we determine suitable parameters for the systems on a smaller part of the corpus. After choosing suitable setups for each system, we run the evaluation on the full corpus and outline the results. Finally, the results are discussed and a conclusion is given.

3 Background

This chapter provides background information; we pinpoint a clear definition for an event, introduce event detection in general, investigate Twitter as data source and talk about necessary requirements for a system being capable of detecting events in Twitter.

We want to mention here that, since we investigate in event detection in Twitter, the terms "document" and "tweet" are used interchangeable in this thesis. As a rule of thumb, we use "document" if the proposed systems or techniques are more general and "tweet" if we handle more specific approaches for Twitter.

3.1 Definition of Event

There exist a vast variety of definitions of an *event*. Since we evaluate and compare event detection systems in Twitter, we need to pinpoint a suitable definition of an *event*. McMinn et al. [MMJ13] tackled this problem and provided a new and general definition which better fits the characteristics of event-based detection in Twitter. It is as follows:

(i) An *event* is a *significant* thing that happens at some specific time and place.

(ii) Something is *significant* if it may be discussed in the media. For example, you may read a news article or watch a news report about it.

Furthermore, they underline that something does not necessarily have to be in the media to be considered an *event*. Emergence in the media just serves as a measure of significance. We think this definition performs well. Therefore, we adopt it in this thesis.

3.2 Event Detection

This section gives a brief overview of event detection in general. We show different theoretical approaches and classify them. Furthermore, we provide examples with application to Twitter.

Event detection is the task of algorithmically detecting events from a given set of input documents and outputting these events in a usable manner. It utilizes techniques from many different fields, like machine learning, data mining, natural language processing, text mining, information extraction and information retrieval [AK13]. Events range from very significant and popular ones (e.g., elections) to smaller or local events (e.g., concerts) [BNG11]. Traditionally, event detection is applied on formal document collections (e.g., academic papers and news articles) and has been a topic of interest for a long time [YPC98]. It was studied extensively in the Topic Detection and Tracking project (TDT) [ACD+98]. Consequently, a lot of techniques were proposed, which can differ greatly depending on the task, application or goal they are made for. Therefore, a classification as it is proposed by Atefeh and Khreich [AK13] is helpful and provides a better overview. They divide event detection systems into *document-pivot* or *feature-pivot* approaches, *specified* or *unspecified* approaches, *retrospective event detection* or *new event detection* approaches and *supervised* or *unsupervised* approaches. Subsequently, these classifications are introduced in detail.

An event detection technique is either *document-pivot*, *feature-pivot* or a combination of both.

Document-pivot techniques use the documents' textual features (e.g., word occurrences as weights in a term vector) to create clusters of similar documents (documents with low semantic distance). This type of technique performs well in traditional event detection (e.g., newswire and news broadcasts as data source) as it was investigated in the TDT [ACD+98] for a long time, however, a direct application to Twitter poses problems. In contrast to the traditional media, a big amount of Twitter's data is useless noise. Furthermore, TDT's corpus only covers around 16.000 news stories, which is rather small compared to Twitter's massive scale of documents emergence and thus a direct application of these techniques without any modifications is problematic. Examples of typical *document-pivot* techniques are the UMass system [ALMS00] and the CMU system [YPC98].

Feature-pivot techniques do not cluster documents by their textual similarity but utilize the distribution of words. A sharp rise in a few words' frequency at a given point

in time is interpreted as an event emerging [Kle02]. Intuitively, this approach makes sense; an increase in the usage of event related words can surely be detected with every emerging event [YPC98], but also non-events may show bursty keyword behaviour (e.g., "good morning" bursting every day). Therefore, similar to *document-pivot* techniques, a direct application to Twitter yields problems. Twitter's temporal distributions of features are too noisy [AK13]. Examples of this type of techniques are Kleinberg [Kle02] and Fung et al. [FYYL05].

Specified event detection and *unspecified event detection* provide another classification for an event detection technique.

Specified event detection aims to detect and monitor a clearly specified event (e.g., earthquakes or social events). The specification can include a precise description of the event itself but may also be constrained in features like place or time. This type of detection technique usually makes use of traditional information retrieval and extraction techniques (e.g., filtering, query generation, query expansion, clustering) [AS12] which are applied on the textual content. Examples of *specified event detection* systems are Sakaki et al. [SOM10] and Lee and Sumiya [LS10].

Unspecified event detection focuses on detecting unknown events (e.g., breaking news detection). This type of event detection cannot rely on monitoring predefined keywords and thus traditional filtering and query generation techniques cannot be applied as it is done in *specified event detection* [Cor12]. Therefore, these techniques usually utilize the temporal emergence of documents. Bursting keywords are monitored and used as a bias towards an event happening. Consequently, as already explained earlier, these techniques, if applied on Twitter, require the separation between trivial and real events (e.g., the keywords "good" and "morning" show strong bursts every morning but should not be considered an event). Other approaches cluster similar documents together (using their textual content) and return the fastest growing clusters. Examples of *unspecified event detection* systems are Long et al. [LWC+11], Petrovic et al. [POL10] and Weng and Lee [WYLL11].

Another classification is given through *retrospective event detection* and *new event detection*. Depending on the task, a technique is considered the former or latter.

Retrospective event detection describes the task of detecting unknown events in accumulated historical data. Therefore, the entire document collection is available and accessible at any time, which can be utilized by hierarchical clustering approaches (e.g., bottom-up). Furthermore, these approaches usually utilize typical information retrieval techniques (e.g., querying an existing document collection, query expansion

and so forth). Examples are Benson et al. [BHB11] and Metzler et al. [MCH12].

New event detection aims to detect new events in a real-time stream of documents. Therefore, the decision whether a new document should be considered a new event or not has to be done directly when the document unfolds. This is usually done by utilizing the temporal information provided by the streaming setup. Incremental algorithms are frequently used for this task, however, these algorithms tend to be resource intensive and thus highly efficient and scalable techniques are required. This proposes new challenges; older studies on new event detection usually focused on detection accuracy without taking efficiency into consideration. For example, in an earlier TDT5 competition, many systems needed several days to process a corpus of just 280,000 news articles, which is clearly too slow for new event detection in a massive real-time streaming scenario [LTY07]. Examples of *new event detection* systems are Becker et al. [BNG11], Petrovic et al. [POL10] and Sankaranarayanan et al. [SST+09].

Finally, event detection techniques can be divided into *supervised* and *unsupervised* techniques. *Hybrid* systems also exist.

Supervised techniques are provided with labelled training data; each dataset of this training data consists of the input (e.g., tweets) and the desired output (e.g., the corresponding events). The *supervised* algorithm uses this training data to learn a mapping function which predicts the output for unseen input data. This explains why supervised techniques are typically used for specified event detection; having clear information about the type of event to detect, it is possible to choose and prepare labelled training data. Examples of *supervised* systems are Sakaki et al. [SOM10] and Popescu et al. [PP10].

Unsupervised techniques are only provided with the actual input data and do not have labelled examples to learn from. Therefore, these systems learn a mapping function only looking at the input (e.g., tweets). *Unsupervised* techniques for event detection usually rely on clustering approaches which try to cluster common features together. Examples of such systems are Cordeiro [Cor12], Petrovic et al. [POL10] and Aggarwal and Subbian [AS12].

Hybrid approaches combine supervised and unsupervised techniques. For example, Sankaranarayanan et al. [SST+09], first, filter away noisy data by the application of a naive Bayes classificator (supervised) and subsequently cluster similar documents together (unsupervised) in order to detect events. Becker et al. [BNG11] start with the clustering of similar documents (unsupervised) and then apply a Support Vector Machine (supervised) in order to decide whether a cluster discusses an event or not.

3.3 Twitter as Data Source

In this section, we look at Twitter as our data source of choice and identify advantages and disadvantages.

As mentioned before, Twitter has become the conceivably most popular and fastest-growing microblogging platform in the world [AK13]. Twitter provides functionality to share a user's opinion, thought, idea, news or just communicate with friends by writing a tweet (i.e., one microblogging post in Twitter) [ZR09]. More than 500 million tweets are posted per day, while this number is also growing with the ongoing increase in active users. Twitter can be accessed using web, smartphone as well as other third party applications, giving people the possibility to easily share text wherever they are and whenever they want. These are among the main reasons for Twitter's high popularity. Therefore, Twitter reflects the dynamics of our social society [LWC+11] and thus can be seen as the "what's-happening-right-now search engine" [Sch09], which is interesting for individuals, corporations and other organizations [JZSC09] (e.g., the government). Companies have seen Twitter's marketing potential and use it nowadays to influence consumers and build reputation [JZSC09]. Other applications are crime prediction [WGB12] and monitoring terrorist activities [AK13].

As clear difference to other microblogging platforms, Twitter's messages are limited to a maximum of 140 characters, which narrows the informative content but creates a very fast and interactive experience. The users' requirements in points of time and content are lowered compared to classical blogging [JSFT07]. Some users see this limitation as a problem, while others see it as the reason that sets Twitter apart (i.e., short information is easier to absorb) [AK13]. A problem provided by the 140 character limitation is the posting of URL strings which can consist of much more characters [SST+09]. Therefore, forwarding services (e.g., http://tinyurl.com) are used which allow a user to shorten these strings and refer to some external resource. Another feature frequently found in tweets is the markup vocabulary; due to Twitter's limited functionality in the early days, the users created a well-defined markup vocabulary in order to underline the origin and/or the purpose of a tweet [Bai14]. Later on, Twitter introduced and implemented the desired functionality (e.g., retweeting), however, the markup vocabulary can still be found in many tweets. A retweet (i.e., a repost of an existing tweet) is usually marked with the prefix "RT @username", where username is the tweet's original author. The prefix "@" before a username creates a mention or reply link to the referenced user account. Such a mention is also displayed in the referenced account. Another feature frequently used in tweets is the hashtag annotation which is used to form topics; this is done by putting "#" followed by the actual topic (e.g., #sb45) and

thus people show what their messages are about (e.g., "watching the superbowl 45 right now #sb45") [BNG11]. This allows users to easily follow topic specific tweets. For this purpose, Twitter itself provides an algorithm which displays trending topics. Even though this trending topics system sometimes outputs current events, popular conversations that should not be considered as events are found frequently (e.g. "getting ready", "#bieberfever") [BNG11]. Furthermore, similar trends are not grouped together (e.g., when Michael Jackson died, a large number of the top trending topics were about him) [KLPM10]. A list of the top 10 trending topics of the moment is shown on each user's Twitter homepage by default.

After posting a tweet, it directly appears on the writer's Twitter profile and is shared to the writer's followers (i.e., a user in Twitter can be followed by other users; this allows the follower to receive all Twitter messages published by the followed user). If a user follows someone else, the followed user's messages appear in chronological order on the following user's own timeline. By default, all tweets are publicly available to anyone (this can be changed in the user settings). Furthermore, as already mentioned, tweets can be retweeted, which forwards a tweet to the retweeting user's followers; these followers can retweet the tweet again and thus information can spread easily beyond the followers of the original tweet's writer. This feature is responsible for the frequently seen snowball effect, which makes event detection in Twitter so feasible (interesting or important messages are usually retweeted more than others [PM10]); if something of high interest happens, it gets retweeted over and over again and thus reaches a big audience all around the world in between a short time [LWC+11]. Kwak et al. (2010) [KLPM10] find that 50% of a tweet's retweeting is done within one hour after the original tweet was written and 75% within the first day. Furthermore, they note that any retweeted tweet reaches an average of 1,000 users, independent of the original user's number of followers. Therefore, millions of people use Twitter as a source of news about significant events which Twitter delivers faster than any other media. Besides the messaging functionality, Twitter also provides typical social network features; users have a profile which can be cultivated with additional information and pictures. The follower network is comparable to the friends function in other social networks. However, following and being followed is not a reciprocating relationship and thus differs from other social networks' friends function [KLPM10].

Besides using Twitter's website, tweets can be retrieved using the Twitter Streaming API[1] or the REST API[2]. The Streaming API provides a continuous stream of tweets emerging in real-time (with up to a few seconds delay), while the REST API returns a set of tweets corresponding to a given query. Depending on the access level, Twitter

[1] Twitter Streaming API, https://dev.twitter.com/docs/api/streaming/ (accessed on April 28, 2014)
[2] Twitter REST API, https://dev.twitter.com/docs/api/ (accessed on April 28, 2014)

only provides a specific amount of its data; the default access level "Spritzer" returns approximately one percent of all public tweets if the sample stream is accessed through the Streaming API. The data is returned in the JavaScript Object Notation[3] (JSON) format; each JSON document contains the tweet's text as well as additional information (e.g., user information, follower and geographical information).

Seeing this immense scale of data, Twitter's real-time nature, the summarised information about latest local and worldwide things happening and the vast possibilities in accessing this data, Twitter is a predestinated source for event detection and thus the increasing interest in suitable systems for event detection in Twitter is reasonable. Successful event detection could lead to fully automatic news detection [POL12]. Many examples have already shown the efficiency of Twitter reporting disasters and social movements [OAR11]. For example, during the protests about the Iranian presidential elections in 2009, many Iranian people posted news about the ongoing protests on Twitter before they were available in the traditional media [POL10]. Another example is the death of Michael Jackson; tweets about his disease were posted on Twitter more than one hour before the conventional news media reported his death [SST+09]. Furthermore, events from Twitter may provide different perspectives compared to traditional media [WYLL11]. Therefore, we investigate in the necessary requirements for a system being able to detect events in Twitter in the next section.

3.4 Event Detection in Twitter

After having an overview of event detection in general as well as about Twitter as our data source of interest, we can point out necessary requirements for a system capable of detecting events in Twitter's large-scale real-time streaming setting. This is done in this section. Furthermore, we investigate the theoretical approaches' suitability.

The detection of events in Twitter is not a trivial task. While having a lot of advantages, Twitter also provides new challenges for event detection that are different from event detection in the traditional media and have to be investigated and solved. These are as follows: (i) the immense scale of data generated forces systems to provide high processing rates and good memory utilization (ii) the streaming environment needs the systems to decide after each new tweet if it belongs to an event or not (iii) in contrast to the traditional media, where the news articles are well structured and written on a high linguistic level, tweets often show low quality which has to be considered; tweets

[3] JSON, http://www.json.org/ (accessed on April 28, 2014)

contain typographical errors, bad grammar, abbreviations, mixed languages, diversity in vocabulary, informal or irregular words and/or user specific terms. This makes traditional text analysis techniques less suitable (iv) the high flood of noise has to be handled; 40% of Twitter's data consists of meaningless babble and 37% of conversational messages [Ana09] which may be important to understand the peoples' opinion (sentiment analysis) but not helpful for event detection. Therefore, the systems have to find a decision whether a tweet, if marked as an event, discusses a real event or a trivial one (v) the shortness of tweets leads to poor performance of existing algorithms due to feature sparsity [LWC+11].

Furthermore, the task of detecting events in the high scale data stream of Twitter is confronted with the same challenges as a data mining algorithm that learns from an endless stream of documents [Cor12]. Bifet et al. (2009) [BK09] clearly propose the requirements for such a data mining algorithm and thus we can adopt these requirements which provide more constraints and help to find suitable systems: (i) process a tweet only once as it unfolds and in the arriving order. A specific number of previous tweets can be remembered, however, the necessary amount of resources should be kept at a low level (ii) keep the memory usage limited (iii) process one tweet in a limited amount of time (systems with growing processing time per tweet, as more tweets become known to the system, are not suitable) (iv) be able to output the current results at any time and after any number of processed tweets. Hence, a decision whether a new tweet belongs to an event or not should be done as the tweet arrives. Additional calculations on a given query should be avoided.

Given our goal of evaluating three event detection systems suitable for Twitter, the general overview of theoretical event detection approaches (section 3.2), the problems posed through Twitter's characteristics as well as the just introduced requirements of Bifet et al., we can narrow the possible systems. We are interested in detecting any event, consequently, we do not investigate specified event detection approaches but focus on unspecified systems. We do not look at retrospective event detection, since we aim to evaluate techniques that are capable of detecting events in a real-time stream. The techniques should be unsupervised and thus do not require any labelled data for training. Both, document-pivot and feature-pivot techniques are applicable, but we will see that usually a combination of both is used. Efficiency and scalability are important factors. Considering the high volume of data and the real-time streaming scenario of Twitter, the event detection algorithm has to work online which means that the system is processing one tweet after another and is not able to see the whole dataset [SST+09]; a decision whether the tweet belongs to a new event or not has to be made straightaway. Putting it together, we are interested in unspecified unsupervised new event detection approaches that work online and fulfill Bifet et al.'s requirements.

4 Related Work

This chapter provides an overview of related work. First, we give a short overview of work about microblogging and Twitter's characteristics. Secondly, we provide a brief summary of traditional event detection systems. Thirdly, we investigate in event detection systems suitable for Twitter's real-time stream. As described in the last chapter, we are interested in unspecified unsupervised new event detection approaches that work online and are highly efficient and scalable. Therefore, we choose the three systems out of the presented approaches which we think are most suitable for our desired task. We will see that most of the systems do not provide all of the desired characteristics (as proposed in section 3.4); systems that completely work online are rarely found.

Since event detection has grown rapidly within the last years, we want to mention that the following survey of related work is far from being complete; a more comprehensive summary of event detection systems in Twitter can be found in Atefeh and Khreich's survey [AK13]. The following related work functions as introductory reading about Twitter, event detection in general and systems suitable for performing event detection in Twitter's massive real-time stream while being unspecified and unsupervised.

Before we focus on event detection, we provide a short overview of work on the typical characteristics of microblogging and Twitter. Examples are Java et al. [JSFT07] who focus on understanding how and why people use Twitter or Krishnamurthy et al. [KGA08] who identify different classes of Twitter users, their behaviours and geographic growth patterns. Furthermore, they investigate in the overall network's size. Jansen et al. [JZSC09] analyse a dataset of 150,000 microblog postings containing branding comments, sentiments and opinions. Further, they apply automated methods for classifying the microblogs' sentiment, evaluate the corresponding results and discuss microblogging as a tool for a corporation's marketing. Kwak et al. [KLPM10] study the topological characteristics of Twitter. For this purpose, they perform an extensive follower-following analysis over the entire Twitter network.

Next, a brief introduction to traditional event detection is given.

Early event detection systems used to represent documents as weighted term vectors using a weighting metric like term frequency[1] (TF) or term frequency-inverse document frequency[2] (TF-IDF). Subsequently, for each new document, the similarity to the already unfolded documents is computed (using a similarity measure like the cosine similarity [KA04]) and if a similarity threshold is reached or exceeded, the documents are clustered together forming events. Examples are the UMass system [ALMS00] and the CMU system [YPC98], which participated in the Topic Detection and Tracking project (TDT); the TDT investigates in systems which try to detect and follow events in a stream of documents [ACD+98] (Reuters newswire and CNN broadcast news transcripts). However, systems, as they were proposed in the TDT, tend to be far too slow for an application in a real-time streaming scenario [MMJ13]; those approaches imply that all documents are kept in memory and thus processing time and memory allocation grows with each new document, making these systems unsuitable to the massive amount of data unfolding through Twitter. Simple solutions to overcome this problem include limiting the number of features to the n highest weighted features in the document [POL10]; consequently, memory usage and processing time grow slower but still without a bound. Other improvements limit the similarity computations to a fixed number of last unfolded documents and discard all older documents, but as a result, the scope of last documents to detect events in is rather small. Therefore, heuristical approaches that keep enough documents in memory to still provide sufficient scope for the detection but reduce the search complexity have emerged (e.g., Indyk and Motwani [IM98] propose a heuristical method that efficiently finds a nearest neighbour on a given query; this approach is used by one of the systems which we chose for this work).

Other approaches for event detection use topic modelling techniques instead of representing documents as vectors in term space; topic modelling reduces the lexical variability of the data but maintains the semantic structure [Cor12]. For example, Latent Dirichlet Allocation (LDA), a widely used topic modelling technique, clusters co-occurring words into topics [WG08] and thus is able to detect common characteristics of the documents. However, inference calculations for a topic model are computationally expensive and consequently an application in the online streaming environment poses problems. Therefore, faster topic models were proposed, which performed well in the TDT, but still remain rather expensive in point of computation time [Cor12]. A

[1] TF, http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.html (accessed on May 27, 2014)

[2] TF-IDF, http://nlp.stanford.edu/IR-book/html/htmledition/inverse-document-frequency-1.html, http://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html (accessed on May 27, 2014)

state-of-the-art example of an (accelerated) topic modelling system for event detection is Pan and Mitra [PM11].

Many systems for event detection are based on clustering approaches, as also are the systems we finally chose. A survey of text clustering algorithms is given by Aggarwal and Zhai [AZ12]. Their work covers a variety of clustering algorithms which are frequently utilized by event detection systems (e.g., distance-based, agglomerative and hierarchical clustering algorithms).

Next, an overview of event detection systems that are designed especially for Twitter or at least for a massive streaming setting is given.

Luo et al. [LTY07] are among the first researchers that address the detection of events in a massive document streaming environment. They introduce an efficient, resource-adaptive framework for event detection in a stream of documents from different sources (mainly traditional media like newswire). Therefore, they apply various indexing and compression methods to increase the document processing rate by two orders of magnitude compared to traditional event detection systems, while maintaining good detection accuracy. Their system dynamically adjusts to the available resources; it focuses on important documents and discards less important ones depending on the resources available. Furthermore, on too high events' arrival rate, they filter and prioritize new events in order to only present the most important ones to the consumer. Another efficiency issue applied is parallel processing. Since they use their system on several document sources, the importance of these sources is computed (utilizing the timeliness of the sources regarding the detected events) and, appropriate to that result, the documents ranked. Efficiency improvements compared to brute-force are realized by limiting both the number of most recent documents kept in memory as well as the number of terms kept for each saved document (they limit the number of terms per document to the 100 highest weighted ones). This works due to the assumption that the discussion of an event lasts for a finite amount of time and it is unlikely that an old event is discussed again. However, this system is quite general using multiple different input sources and thus does not address many problems posed through Twitter (e.g., handling noise).

Phuvipadawat and Murata [PM10] propose a web-based application called Hotstream that detects events in Twitter. Therefore, they extract tweets by predefined search queries (e.g., hashtags like "#breakingnews") using the Twitter Streaming API. For each new tweet, an index based on the textual content of the tweet is updated using Apache Lucene (i.e., an information retrieval software library). Similar tweets are clustered together, forming events. Similarity is measured using TF-IDF as well as

an additional boost factor which is made of event typical proper nouns (e.g., "China", "England", "news"), hashtags and usernames. Proper nouns are selected using the Stanford Named Entity Recognizer (NER). In order to keep the tweets in a cluster related to the first tweet (which initially formed the event), they compare a new tweet to the first one in the cluster as well as the cluster's k most frequent terms. If a predefined similarity threshold is reached, the new tweet is added to the cluster. Otherwise, a new empty cluster is created and the new tweet added. For each cluster, a score is generated by taking reliability and popularity factors into account. Phuvipadawat and Murata measure this reliability by the number of followers of the clusters' tweets' users, while popularity is made up of the number of retweets within a cluster. The score is further adapted by considering the tweets' novelty. This calculation is carried out on a predefined number of newest tweets in a cluster. Phuvipadawat and Murata find that the utilization of proper nouns is important for successful similarity comparisons. Sole TF-IDF is a too weak weighting because of the tweets' term sparsity. We decide to not use this system due to only extracting tweets by using predefined search queries; this requires prior knowledge. Furthermore, the system is not able to catch events which suddenly happen and whose tweets are not yet marked with specific hashtags or event specific keywords.

Weng and Lee [WYLL11] try to tackle the challenge of event detection in Twitter through clustering of wavelet-based signals. The system monitors the words distribution over time rather than similarity between tweets. For each word, a frequency signal is built according to the number of word occurrences over time and transformed into a wavelet, which is less expensive in storage. A wavelet is an oscillating function that is localized in both time and frequency. Consequently, this transformation provides exact information about when and to what scale bursting words emerge. The wavelet computation can be done fast using wavelet analysis. Utilizing a sliding window, the signals' change over time is captured. Subsequently, trivial words are filtered away by using the signals' auto correlation. Finally, events are formed by clustering the remaining signals together. This is done by employing a modularity-based graph partitioning technique. Weng and Lee differentiate between real events and trivial ones by computing the events' significance. Therefore, they utilize the cross correlation of words which are related to the event. Despite the system's fairly good performance and efficiency in storage consumption, it actually does not work online. While the wavelet signals could be updated incrementally, performing the calculation of the signals' correlation as well as the clustering process are too expensive [Cor12]. A decision cannot be made efficiently with each new tweet as it unfolds.

Becker et al. [BNG11] apply an incremental online clustering technique for detecting events in Twitter. In order to keep the algorithm scalable, the number of clusters used is

determined by empirically tuning a threshold parameter during a training phase. Subsequently, events are detected by clustering similar tweets together. They compute and incrementally update a centroid representation for each cluster; this centroid is calculated as the average weight of each term of all the cluster's tweets. The tweets' terms are weighted using TF-IDF and the similarity is computed applying the cosine similarity measure. Furthermore, they double the weight of hashtag terms in the tweets' representation. In the output process, each cluster is analysed in order to rate its relevance in being an event. This is done by incrementally calculating revealing features for each cluster and using them to train a classifier which distinguishes between event and non-event clusters. The used features are as follows: (i) temporal features (i.e., volume of messages belonging to an event during the event's time of taking place) (ii) social features (i.e., percentage of the cluster's messages containing user interactions like retweets and mentions) (iii) topical features (i.e., based on the assumption that an cluster's tweets mainly focus on one central topic, while non-event clusters center around common terms, for instance, "sleep" and "work", which do not reflect the same single theme) (iv) Twitter-centric features (i.e., frequently occurring patterns in non-event clusters like tag usage and presence of multi-word hashtags). These features are updated incrementally, as the clusters evolve over time. Finally, a decision whether the cluster is considered an event or non-event cluster is made through the application of a Support Vector Machine (SVM) which is trained on a labelled set of cluster features. The system allows the extraction of the top events at any given point in time without additional calculations. We decide against this system because it requires a manually labelled set of cluster features.

Long et al. [LWC+11] propose an event detection system that works in the microblog streaming scenarios. Furthermore, they investigate in event tracking and summarisation. First, they detect events from a microblog stream using a traditional clustering approach with respect to some specific microblogging features. Therefore, topical words (i.e., words with frequent co-occurrence like "Apple" and "iPad") are clustered together. The selected criteria for a word being topical are word frequency, occurrence in hashtags and the word's entropy. This is done within a time period of one day, and later, in the tracking part, the one-day pieces are grouped together if they are related, thus forming event chains. Subsequently, they monitor for topical words' co-occurrence. A co-occurrence graph is built upon all topical words, where each vertex represents a topical word and each edge a co-occurrence of the connected words. The edges' weight is calculated as the number of co-occurrences. Utilizing a hierarchical divisive clustering approach on this graph, the topical words are clustered and thus form events. Secondly, a novel algorithms tracks the evolution of the detected events; this is reasonable since events evolve over time and topic drifts should be recognized. Therefore, related one-day event clusters are grouped together, forming

event chains. This is done by a maximum-weighted bipartite graph matching, using an adaption of the Jaccard coefficient as similarity measure between clusters. Thirdly, they summarise the event chains to provide the user a better understanding of what is going on; this is important since topical words itself do not provide a good understanding of the event. The step is similar to document summarisation and thus a top k set (forming the summary) of most relevant documents in a given cluster is determined. Here, most relevant means the documents that contain as much content coverage of the cluster as possible. These summaries are linked to the event chain clusters. The system's experimental results show that the applied feature selection outperforms the sole use of document frequency features as well as the use of document frequency features in combination with entropy. Another result is that the applied top-down divisive clustering outperforms traditional hierarchy clustering as well as K-means, which perform rather poor. Long et al. also find that their system's detection performance is relatively low compared to the systems proposed in the TDT project. Furthermore, this system does not work in an online fashion. The detection, tracking and summarisation cannot be performed after each or at least after a few documents in an efficient way. Furthermore, the hierarchical divisive clustering approach requires the full similarity matrix and thus does not scale well to the large amount of data received through Twitter [AK13].

Cordeiro [Cor12] uses a similar approach as Weng and Lee [WYLL11]. He presents an event detection system using wavelet signal analysis of hashtag occurrences in Twitter's public stream. Only the tweets' hashtags, which are retrieved in five minute intervals, are used for building the signals. A peak in the usage of a given hashtag is rated as bias for an event happening. The text itself is discarded. For each hashtag, a signal is constructed by counting the number of hashtags in each time interval. This is realized through application of map reduce techniques. The event detection itself is performed by signal analysis using two different wavelet tools: a peak analysis detects the actual events in the signal and a local maxima detection monitors changes. Besides the actual event detection, an event summarisation is conducted. This is due to frequent ambiguity in the hashtags' meaning. Therefore, on an event detected, a topic inference model (Latent Dirichlet Allocation using Gibbs sampling) is built on the textual content of all tweets that contain the event's hashtags within a five minutes interval around the event. This method produces a summarisation for detected events. Putting it together, this system is a valid lightweight approach for event detection in Twitter, however, events are detected by only monitoring hashtags. This may be efficient and thus suitable for the massive scale of Twitter's data, but all kind of events whose tweets are not strictly marked with proper hashtags cannot be detected. For instance, something of high interest and happening unexpectedly may not be marked by common hashtags and thus not be detected. Another problem is the usage of multiple hashtags for the same event (e.g., "f1jp", "f1nijigen" and "f1"). A merging would be desirable, but

would make the system less efficient (comparisons on the textual content would have to be done). We drop this system because Cordeiro only uses the tweets' hashtags and thus completely discards the textual content; only events which are marked with proper hashtags may be detected.

Petrovic et al. [POL10] introduce a system for event detection in Twitter's real-time stream. Their focus lies on an efficient algorithm that works in bounded time and with bounded memory. Therefore, they use locality sensitive hashing (LSH) [IM98] as a backoff towards exact search; LSH generates over an order of magnitude speedup in processing time over other state-of-the-art system on this task, while still maintaining competitive results. On a new tweet, LSH returns the ϵ-approximate nearest neighbour with a high probability (the already unfolded tweet with the highest similarity to the new tweet). Nevertheless, pure LSH provides poor performance as well as high variance and thus Petrovic et al. introduce a variency reduction strategy. For each tweet declared being a new event by LSH (i.e., no old tweet found with sufficient similarity) a search through a specific number of most recently unfolded tweets is started. Having a strict upper bound of tweets kept, Petrovic et al. achieve bounded memory usage for the overall system as well as bounded processing time for each new tweet. The event detection itself is done by grouping each new tweet into the same cluster which the most similar tweet belongs to as long as the similarity between these tweets reaches or exceeds a predefined threshold. Otherwise, a new empty cluster is created and the new tweet added. Subsequently, they look at the fastest growing clusters which is an indication for an event spreading. This growth rate is also used as a measure for the cluster's importance. Another issue Petrovic et al. investigate in is noise. They find that a high amount of spam in the output can be reduced by taking the clusters' entropies into account. We use this system as our baseline due to our earlier work with the system and the promising results in Twitter.

Sankaranarayanan et al. [SST+09] propose an application called TwitterStand that tries to detect events in Twitter's real-time data stream. They do not only receive Twitter's public stream as input but also extract tweets from 2,000 handpicked users, so called seeders that are known to publish news. Furthermore, they automatically identify 200,000 users that tend to interest in or write about news; these users are selected as being followers of the handpicked seeders. In addition, tweets are received by tracking for news specific keywords; the Twitter Streaming API provides this functionality. They also use the artifacts existent in Twitter; an artifact is an external resource (e.g., articles, images and videos) which is linked in a tweet. Receiving a huge number of tweets (from seeders, automatically chosen news-interested users, tweets by tracking for event specific keywords as well as Twitter's public stream) they initially apply a naive Bayes classifier which is trained on a training corpus of premarked tweets (classifying

a tweet either as news or junk). The classifier is not applied on the seeders' input. This sorts out a lot of noisy data before the actual clustering and event detection process but may also sort out relevant tweets. Subsequently, all as news classified tweets are clustered together applying an online clustering algorithm. For each cluster, a time centroid (i.e., the mean publication time of the cluster's tweets) is stored and updated and if the time centroid exceeds a predefined threshold, the cluster is marked inactive (no tweets can be added anymore); this process keeps the number of clusters limited. The similarity computations between tweets and clusters are extended by a Gaussian attenuator which takes the temporal information into account; a tweet whose publication time is nearer to the cluster's time centroid is rated higher. They accelerate the search for a tweet's most similar cluster by using an inverted index. Paired with the clusters' active / inactive feature described earlier, Sankaranarayanan et al.'s system minimizes the number of computations necessary, but they also find this approach not working well (because of Twitter's noisy input) without additional improvements. A possible solution is to let tweets which are received from the public stream not form new clusters. This would lead to relatively noise-free clusters, which is desirable, but would also eliminate the whole possibility of detecting breaking news reported by Twitter's users before conventional news sources pick them up (e.g., the seeders). Therefore, they allow a public stream's tweet to create a new cluster, but also introduce the constraint that this cluster becomes inactive if none of the k first added tweets is received from a seeder. Other topics Sankaranarayanan et al. investigate in are fragmentation (i.e., several clusters on a single topic), weight upper bounds (i.e., features that enter the system for the first time have too large TF-IDF values; weight upper bounds limit these weights), phrases (e.g., "San Francisco"), geographical aspects of the clusters (e.g., extracting each clusters geographical focus) as well as providing a user interface for their application. Sankaranarayanan et al. find their system being quite fast and robust. The noisy random stream proposes the biggest challenge but also provides the fastest breaking news delivery. We decide to use this system; we think Sankaranarayanan et al.'s approach of cleaning up old clusters and also the steps introduced for improving the similarity computations scale well to Twitter's real-time streaming setting.

Aggarwal and Subbian [AS12] propose an event detection system which is extended by a network component. They show that the presence of network structure in social streams like Twitter can be used to establish a more precise event detection system compared to the text-only clustering approaches. The network structure is made up of the actors as nodes as well as the sent messages as edges from sender to receiver. For each new unfolded document, its similarity to the existing clusters is calculated. This is done by computing both the textual similarity as well as the similarity in points of network structure. They design a sketch-based technique in order to speed up these network computations and thus make the system suitable for the massive streaming

setting. Their evaluation shows that the information gained by network structure outperforms the information gained by the text content in their clustering method. This reveals interesting possibilities for improving current state-of-the-art event detection systems by adding additional network structure information into the similarity computations. The system works online and seems to be rather fast [MMJ13], which makes it suitable for Twitter's streaming setting and thus we choose it to be the third system for our evaluation.

Putting it together, we choose Petrovic et al.'s system [POL10] to be our baseline. This is due to our earlier work with their system as well as the promising results. The technique works online and Petrovic et al. put a lot of effort into efficiency and scalability, thus making their approach suitable for Twitter's massive real-time streaming setting. Systems that also work online and seem suitable are Sankaranarayanan et al. [SST+09], Phuvipadawat and Murata [PM10], Becker et al. [BNG11], Aggarwal and Subbian [AS12] as well as Cordeiro [Cor12]. The systems proposed by Weng and Lee [WYLL11] as well as Long et al. [LWC+11] do not scale well to the large amount of data received from Twitter's real-time stream [AK13] and thus we discard these systems. We also drop Phuvipadawat and Murata's system because they initially only extract tweets using predefined search queries and thus prior knowledge is required; Petrovic et al.'s system, our baseline, analyses the random stream of tweets without any constraints. Furthermore, we ignore Cordeiro's system since he only uses the tweets' hashtags; the textual content is completely discarded, which makes a comparison to Petrovic et al. (who deliberately neglect hashtags to keep their system more universal) unreasonable. Additionally, only about 10% of all tweets contain hashtags which leads to a huge loss of information [Pet12]. Further, we do not use Becker at al.'s system because it requires a manually labelled set of cluster features. We decide to use the system proposed by Sankaranarayanan et al. [SST+09] because we think their online clustering approach scales well to Twitter's streaming setting. The third system we choose is the event detection approach proposed by Aggarwal and Subbian [AS12] because it works online and seems to be rather fast [MMJ13]. We introduce these three systems in the next chapter in detail.

5 The Event Detection Systems

In this chapter we introduces the systems for event detection which we use for our evaluation. These are Petrovic et al. [POL10], Sankaranarayanan et al. [SST+09] as well as Aggarwal and Subbian [AS12].

All three systems which we chose cluster tweets (document-pivot approach) instead of words (feature-pivot approach); the reason for this decision is that the corpus we use for our evaluation contains labelled tweets for each of the labelled events. Having clusters of tweets instead clusters of words as the systems' output, we can assess how the systems perform in detecting an event's tweets.

Each of the systems represents tweets (after being preprocessed) as vectors in term space using TF-IDF weighting. Subsequently, similar tweets (corresponding to a similarity measure) are clustered together using a clustering approach. Event clusters are finally output using an output mechanism (e.g., the fastest growing clusters).

We do not investigate in the different preprocessing steps which are reasonable on Twitter's noisy data in this chapter; the preprocessing steps we apply are described in the evaluation chapter.

While all three systems have the representation of tweets in common, the clustering approaches differ substantially. Petrovic et al. uses an approximate technique in order to efficiently find the most similar tweet for each new tweet and if the similarity is high enough, the new tweet is added to the most similar tweet's cluster. Sankaranarayanan et al. does not compare tweets to tweets, but determines for each new tweet the most similar cluster while only looking at clusters that have at least two terms in common. Furthermore, the time difference between the tweets creation time and the mean creation time of the clusters' tweets is taken into account. Aggarwal and Subbian limit the number of clusters kept to a fixed number and for each new tweet a content similarity and a structural similarity (based on the underlying network) to each cluster is computed. All three systems provide techniques to keep the amount of memory and the amount of processing time low and bounded, which is an important point for us.

The output generation for Petrovic et al. and Sankaranarayanan et al. are similar. Here, we look at a fixed number of fastest growing and big enough clusters for the output. Aggarwal and Subbian monitor the clusters' growth ratio with respect to a given time horizon and output a cluster if this ratio reaches or exceeds a specific threshold.

Subsequently, we introduce the three systems in detail.

5.1 Petrovic et al.'s System

This section introduces Petrovic et al.'s system (2010) [POL10] in detail. The system is developed for event detection in Twitter's real time stream with a clear focus on an efficient algorithm, working both in bounded time and with bounded memory. After preprocessing, a tweet is represented as vector in term space using TF-IDF. Subsequently, for each new tweet, the nearest neighbour is searched and if the similarity (cosine similarity) to this nearest neighbour reaches or exceeds a predefined threshold, the new tweet is added to the nearest neighbour's cluster. In the case that the similarity is not big enough, an empty cluster is created and the new tweet added. In order to accelerate the nearest neighbour search, Petrovic et al. introduce an approximate technique (locality sensitive hashing) which generates over an order of magnitude speedup in processing time. However, only using the approximate method generates high variance; therefore, a variance reduction strategy is introduced which uses an inverted index to compare the new tweet to a fixed number of most recent tweets if the locality sensitive hashing approach fails. Finally, event clusters are output by looking at the fastest growing clusters. Petrovic et al. use a cluster's entropy to decide if the cluster should be considered spam.

In the following sections, we investigate in the different substeps document representation, similarity computation, locality sensitive hashing, variance reduction strategy, bounded space, bounded time as well as output generation issues.

5.1.1 Document Representation and Similarity Computation

Petrovic et al.'s system clusters tweets instead of words; a common document representation for such a document-pivot approach is to use vectors in term space. Here, the vector's dimensions represent the different terms in the collection, where a dimen-

sion's value is usually computed by a weighting function (e.g., TF-IDF). Petrovic et al. use an incremental TF-IDF weighting as it was already used by Yang et al. (1998) [YPC98]:

$$weight_t(w,d) = f(w,d) * \frac{\log(\frac{|D_t|+0.5}{|\{d':w\in d',d'\in D_t\}|})}{\log(|D_t|+1)} \tag{5.1}$$

where $weight_t(w,d)$ is the weight of term w in document d at time t. $f(w,d)$ is the number of occurrences of w in d (term frequency). D_t describes the complete collection of documents processed at time t, including d. $|\{d' : w \in d', d' \in D_t\}|$ is the number of documents that contain term w (inverse document frequency). It is noteworthy that Petrovic et al. do not decrement a term's IDF when a tweet containing the term is removed from the system (Appendix, E-Mail Conversation with Sasa Petrovic); this is explained through the IDF's purpose of reflecting how common a term is in the underlying language. Consider five tweets processed, each containing term a and five more tweets processed, each containing term b. If the first five tweets are removed from the system and term a's IDF is decremented respectively, on a new tweet t_{new} containing term a and term b entering the system, term a's TF-IDF weighting is higher than term b's weighting[1], however, term a is a more frequent used term in the underlying language and thus should be weighted lower. Consequently, we do not decrement a term's IDF when a tweet containing the term is removed from the system.

Having textual documents represented as vectors in term space, a similarity between two documents d_1 and d_2 can be computed using a similarity function. A frequently used measure is the cosine similarity, which outperforms other similarity measures (e.g., weighted sum) on the new event detection task [ALMS00]. Hence, Petrovic et al. employ the cosine similarity as it was defined by Kumaran and Allan (2004) [KA04]:

$$sim_t(d_1,d_2) = cos\_sim(d_1,d_2) = \frac{\sum_w weight_t(w,d_1) * weight_t(w,d_2)}{\sqrt{\sum_w weight_t^2(w,d_1)} * \sqrt{\sum_w weight_t^2(w,d_2)}} \tag{5.2}$$

5.1.2 Locality Sensitive Hashing

As already mentioned earlier, the problem of traditional document-pivot approaches is caused through the increasing processing time with each document processed; this is due to a new document being processed to all already unfolded documents. However, if the number of s kept is limited, the detection scope may be too small in order to perform suitable event detection. Therefore, Petrovic et al. decide to use an approximate

[1] $weight_t(a,t_{new}) = 1 * \frac{log(\frac{11+0.5}{1})}{log(11+1)} = 0.9829, \quad weight_t(b,t_{new}) = 1 * \frac{log(\frac{11+0.5}{6})}{log(11+1)} = 0.2618$

technique (locality sensitive hashing) which keeps a sufficient number of tweets in the system but significantly reduces the time for finding the nearest neighbour. This is realized by hashing the tweets into multiple hash tables and only inspecting the tweets which crashed most frequently in the different hash tables.

Locality sensitive hashing (LSH) was initially introduced by Indyk and Motwani [IM98] as a solution for the ϵ-approximate nearest neighbour problem (ϵ-NN-problem); consider a set of n points $P = \{p_1, ..., p_n\}$ in the metrical space X. On a given query point $q \in X$ find a point $p \in P$ that is an ϵ-approximate nearest neighbour of the query q, which means for all $p' \in P$ and $d(p, q)$ being the distance between p and q : $d(p, q) \leq (1 + \epsilon)d(p', q)$. With own words, we try to find any point that lies within $(1 + \epsilon)r$ distance to the query point q, where r is the distance to the real nearest neighbour [POL10].

Up to now, no efficient algorithm for the exact search exists when the dimensionality is high and thus restricting on this approximate search is reasonable if a speedup over the linear search is striven for. Optimized approaches only provide little improvement over the brute-force algorithm (i.e., comparing the query $q \in X$ to all points $p \in P$) [IM98]. Therefore, Indyk and Motwani [IM98] introduce the notion of locality sensitive hashing as an approach for solving the ϵ-approximate nearest neighbour problem in sublinear time. LSH is defined as a distribution on a family F of hash functions where the probability of two documents x and y having the same hash value for hash functions $h \in F$ is equal to the similarity between the two documents. This similarity is determined by a similarity function defined on the collection of documents [Cha02]:

$$Pr_{h \in F}[h(x) = h(y)] = sim(x, y) \tag{5.3}$$

where $Pr_{h \in F}[h(x) = h(y)]$ is the probability of documents x and y having the same hash value for hash functions $h \in F$ and $sim(x, y)$ is a similarity function defined on the collection of documents [Cha02].

In order to use this LSH scheme for our purpose, each tweet, represented as vector in term space, is hashed into a hash table in such a way that two tweets that are similar to each other crash into the same bucket with a much higher probability than vice versa. On a new tweet's crash, all tweets in the bucket are inspected and the most similar one returned. Since Petrovic et al. use the cosine similarity for the similarity comparisons, they apply a LSH scheme introduced by Charikar [Cha02] where the probability of two tweets crashing into the same bucket is proportional to the cosine of the angle between their vectors [POL10].

Subsequently, the LSH scheme introduced by Charikar [Cha02] is described in detail. The hashing scheme is realized by intersecting the space with k random hyperplanes (through the origin). Thus, the buckets are formed by the subspaces between these hyperplanes. Each hyperplane $i \in \{1, ..., k\}$ is represented by a random vector $\vec{r_i}$ from the d-dimensional Gaussian distribution. The appropriate hash functions $h_{\vec{r_i}}$ are defined as follows:

$$h_{\vec{r_i}}(\vec{x}) = \begin{cases} 1 & \text{if} \quad \vec{r_i} * \vec{x} \geq 0 \\ 0 & \text{if} \quad \vec{r_i} * \vec{x} < 0 \end{cases} \qquad i \in \{1, ..., k\} \tag{5.4}$$

$\vec{x}$ is the tweet's vector in the d-dimensional space. The tweet's final hash value is a concatenation of each of the k hash functions' resulting bit (forming a binary number of length k). This number denotes the bucket in the hash table the new tweet is hashed into. Consequently, using k hyperplanes, the hash table consists of 2^k buckets. Having this scheme, the exact probability of two tweets $\vec{x}$ and $\vec{y}$ having the same hash value for hash functions $h \in F$ is given by

$$Pr_{h \in F}[h(\vec{x}) = h(\vec{y})] = 1 - \frac{\theta(\vec{x}, \vec{y})}{\pi} \tag{5.5}$$

where $\theta(\vec{x}, \vec{y})$ corresponds to the angle between vectors $\vec{x}$ and $\vec{y}$. θ is proportional to the function $\cos(\theta)$ making θ closely related to the cosine similarity measure.

Using more hyperplanes reduces the number of collisions but also lowers the probability of colliding with the nearest neighbour. Therefore, using L independent hash tables ($L > 1$ and each hash table constructed by its own random k hyperplanes) improves the chance of finding the nearest neighbour in at least one of them. Denote δ the probability of missing a nearest neighbour in all L hash tables. Furthermore, let P^k_{coll} be the probability of tweets $\vec{x}$ and $\vec{y}$ colliding into the same bucket in one hash table with k hyperplanes. Then $\delta = (1 - P^k_{coll})^L$ and thus we can compute the number of hash tables L in order to miss the nearest neighbour in all L hash tables with probability δ [Pet12]:

$$L = \log_{1-P^k_{coll}} \delta \tag{5.6}$$

Using this scheme, Petrovic et al. speed up their search for the most similar tweet by an order of magnitude while keeping a sufficient number of tweets in the system for being possible nearest neighbour candidates. However, the sole application of LSH performs rather poor [POL10] due to high variance and thus Petrovic et al. introduce a variance reduction strategy, which is introduced in the next section.

5.1.3 Variance Reduction, Bounded Time and Bounded Space

As described in the last section, LSH only returns the nearest neighbour with a certain probability, which leads to high variance in the results [POL10]. Therefore, petrovic reduce this error by introducing a variance reduction strategy. If LSH fails to return a nearest neighbour (i.e., not finding a tweet that reaches or exceeds a predefined similarity threshold) the new tweet is compared to a fixed number b_n of recent tweets (ignoring tweets that have already been inspected through LSH). For this purpose, Petrovic et al. [Pet12] introduce two different strategies: (i) recency and (ii) uniform. The former strategy just compares the new tweet to b_n of most recent tweets that share at least one term in common with the new tweet and that have not yet been returned by LSH. The latter strategy uses an inverted index to compare the new tweet for each of the new tweet's distinct terms to at most $b_n/||d||_0$ tweets that contain this term and have not yet been returned by LSH. $||d||_0$ is the number of distinct terms in the new tweet. If there are more than $b_n/||d||_0$ tweets for a term, the $b_n/||d||_0$ most recent ones are used. Therefore, the recency strategy purely focuses on the tweets' freshness, while the uniform strategy inspects a greater variety of tweets [Pet12]. Petrovic et al.'s results show that the uniform strategy clearly outperforms the recency strategy applied on tweets. Consequently, we use the uniform strategy. The variance reduction strategy lowers the variance significantly, as Petrovic et al. show in their results.

It remains to explain how Petrovic et al. proceed after a new tweet is hashed using LSH. Consequently, the set S of tweets which crashed with the new tweet in the L hash tables is inspected and ordered according to the tweets which crashed with the new tweet in the most hash tables. Subsequently, only the top $3L$ tweets of this set are used for the similarity computation with the new tweet. Having this constraint and the limited number of comparisons through the variance reduction strategy, Petrovic et al.'s system has a clear upper bound in similarity computations performed on a new tweet entering the system and thus bound processing time per document is achieved.

Furthermore, in order to keep the amount of space limited, the hash tables' buckets' size is fixed to a constant. If a bucket is full and a new tweet crashes into it, the oldest tweet is removed. Looking at the inverted index that is used for the variance reduction strategy, Petrovic et al. decide to only hold tweets in the inverted index as long as the tweets are represented in at least one of the LSH hash tables [Pet12]. Consequently, the upper bound for the number of tweets hold in the system is determined through LSH (i.e., at most $L * 2^k *$ bucket_size tweets).

5.1.4 Event Clustering

Having LSH as well as the variance reduction strategy, a new tweet's similarity to the nearest neighbour as well as the nearest neighbour itself is returned (with the introduced inaccuracy). In order to group tweets together and form events, Petrovic et al. proceed as follows: if the similarity between a new tweet and its nearest neighbour reaches or exceeds a predefined threshold t, the tweet is added to the nearest neighbour's cluster. Otherwise, a new empty cluster is created and the new tweet added. Consequently, the threshold t specifies the granularity of the clusters; a high threshold leads to smaller and more specific clusters, while a small threshold causes bigger and broader clusters.

5.1.5 Overall Algorithm for Event Detection

Algorithm 1 shows Petrovic et al's overall algorithm for event detection without the output generation process; the output generation is described in the next section.

Algorithm 1: Petrovic et al's algorithm for event detection

input: stream of documents

1 $X \leftarrow$ empty cluster list;
2 **foreach** *document d* **do**
3 preprocess d and represent its textual content as vector in term space using TF-IDF weighting;
4 add d to LSH;
5 $S \leftarrow$ set of $3L$ documents that collide with d in LSH most frequently;
6 $sim \leftarrow 0$;
7 $nearest\_neighbour \leftarrow$ null;
8 **foreach** *document* d' *in* S **do**
9 $c\_sim = cosine\_similarity(d, d')$;
10 **if** $c\_sim > sim$ **then**
11 $sim \leftarrow c\_sim$;
12 $nearest\_neighbour \leftarrow d'$;
13 **if** $sim < threshold$ **then**
14 $S \leftarrow$ set of at most b_n recent tweets obtained through the inverted index;
15 repeat steps 8 to 12 and then jump to 16;
16 **if** $sim < threshold$ **then**
17 create a new empty cluster c and add d;
18 add c to the list of clusters X;
19 **else**
20 add d to the $nearest\_neighbour$'s cluster c;
21 add d to the inverted index;

Recalling Petrovic et al.'s system, when arriving at the system, a new tweet is preprocessed and represented as vector in term space using TF-IDF (line 3). Subsequently, the tweet is hashed into the L LSH tables and the resulting set S ordered according to the top $3L$ tweets which crashed with the new tweet in the most LSH tables (lines 4-5). Subsequently, for each tweet in S, the cosine similarity to the new tweet is calculated and the highest similarity as well as the most similar tweet (nearest neighbour) saved (lines 6-12). If this similarity does not reach the predefined threshold, the variance

reduction strategy is used (lines 13-15). If the similarity still does not reach the predefined threshold, a new empty cluster is created and the new tweet added (lines 16-18). Otherwise, the tweet is added to the nearest neighbour's cluster (line 19-20). Finally, the tweet is added to the inverted index which is needed for the variance reduction strategy (line 21).

5.1.6 Output Generation

Petrovic et al. consider two different output generation approaches, depending on the application. One possibility is to just have a fixed threshold for the number of tweets in a cluster and if this threshold is reached, the cluster is output. This approach is better for showing results to humans (e.g., having a demo or an application showing live results) because results can be seen continuously (Appendix, E-Mail Conversation with Sasa Petrovic). Another possibility is to define an output time interval and output the n fastest growing and big enough clusters (e.g., n = 100 and 25 tweets as a minimum in a cluster) each time interval. This approach is better for in-depth evaluation (Appendix, E-Mail Conversation with Sasa Petrovic); a cluster's high growth rate shows that many people are talking about something closely related in a specific time interval, which is a clear indication for an event spreading [POL10]. Furthermore, the growth rate can be used for measuring an event's importance. Doing an evaluation, we decide to use the second approach.

Furthermore, due to many clusters containing spam, whenever a cluster is a candidate for being output, Petrovic et al. compute the clusters' entropy H:

$$H_{cluster} = -\sum_{i} \frac{n_i}{N} \log_2 \frac{n_i}{N} \tag{5.7}$$

where n_i is the count of word i in the cluster and N the overall number of words in the cluster. If a cluster's entropy does not reach or exceed a predefined threshold, the cluster is considered spam and destroyed. Petrovic et al. find that taking the clusters' entropies into account significantly improves the results; this is due to spam clusters contain little information and thus have low entropy.

After a cluster is either output as event, marked as spam or just has too low growth rate / is too small, the cluster is destroyed in all cases (Appendix, E-Mail Conversation with Sasa Petrovic). Since Petrovic et al. does not set any clear standard how to remove these clusters, we introduce a time to live (e.g., one hour or 100,000 documents processed). Each time we inspect all clusters for being output candidates (after the

output time interval has passed), we check if the clusters' time to live is over and, in case, remove the clusters afterwards.

Algorithm 2 shows the output generation process as we use it for our evaluations by periodically checking over all clusters at regular time intervals.

Algorithm 2: Petrovic et al's algorithm for output generation

input: list of clusters

```
1 foreach cluster c do
2     if c belongs to top n fastest growing and big enough clusters then
3         c_entropy = -∑_i (n_i/N) log (n_i/N);
4         if c_entropy < entropy_threshold then
5             mark as spam;
6         else
7             output cluster as event;
8     if c.creation_time < current_time - time_to_live then
9         destroy cluster;
```

Giving a conclusion of algorithm 2, after the output time interval has passed, we iterate over all clusters in the system and inspect the n fastest growing and big enough clusters (lines 1-2). For each of these top clusters, compute the entropy and if the entropy does not reach a predefined entropy threshold, mark the cluster as spam (lines 3-5). Otherwise, output the cluster as event (line 7). Furthermore, for all clusters in the system (not only the top n), check if their time to live is over and remove them from the system, in case it is.

5.2 Sankaranarayanan et al.'s System

Sankaranarayanan et al. (2009) [SST+09] propose an event detection system for Twitter, called TwitterStand. The system uses multiple streams as input, consisting of seeders (2,000 handpicked users which are known to publish news), Twitter's public stream, up to 200,000 users that tend to interest in or write about news (these are dynamically identified while the system is running) as well as tweets requested by the tracking functionality provided by Twitter (tracks tweets by specific keywords). They also use the artifacts existent in Twitter; an artifact is an external resource (e.g., articles, images and videos) which is linked in a tweet. Their system is a hybrid approach; first,

in order to filter away junk, a supervised classifier is applied on the input data. Subsequently, tweets are clustered together using an unsupervised clustering approach. Finally, event clusters are output by looking at the fastest growing clusters.

Since we conduct our evaluations using a corpus generated from Twitter's public stream, we only focus on the components of Sankaranarayanan et al.'s system that are important for detecting events in the public stream. The crucial steps, which we investigate in, are a naive Bayes classifier, document representation, detecting events, fragmentation, weight upper bounds as well as the output generation process; these are described as follows. Other issues Sankaranarayanan et al. cover are phrases, geotagging, computing the topic's geographical focus as well as providing a user interface for their system.

5.2.1 Naive Bayes Classifier

Sankaranarayanan et al.'s system differs from many approaches for event detection in Twitter; it is an hybrid approach, being both supervised and unsupervised. This is due to their initial step applied on the input data (except the input from seeders). A naive Bayes classifier is used to either classify a new tweet being junk or news and only tweets that are classified being news are forwarded to the actual event detection step. While this makes sense intuitively, it is hard to provide an accurate classifier for the dynamically evolving data source Twitter while doing unspecified event detection. The whole system's detection performance depends on the classification accuracy; relevant tweets could be discarded before reaching the actual event detection process [AK13]. However, the classifier requires a suitable labelled document collection for the training, which we do not have. Consequently, we decide to skip this step and try to handle noise after the actual event detection process, as it is also done by Petrovic et al. (using entropy).

5.2.2 Document Representation and Similarity Computation

Sankaranarayanan et al. represent a document as vector in term space and choose TF-IDF as the vector's dimensions' weighting (same as Petrovic et al. do). Since they do not specify the exact TF-IDF formula they apply, we just use the same incremental TF-IDF version that is used by Petrovic et al. (formula 5.1).

Similarity computations are performed using the cosine similarity, which is also deployed by Petrovic et al. The only difference is that Sankaranarayanan et al. compare a new document with a cluster, not a document. This, however, works exactly the same way since a cluster is represented by a feature vector which is the vector obtained by averaging the weights of the various terms of all documents in the cluster [KKS00]. Consequently, the cluster's feature vector can be seen as a document's vector and the cosine similarity can be used in the same way as applied by Petrovic et al. (formula 5.2). In addition, Sankaranarayanan et al. extend the similarity computation by a temporal dimension. Therefore, a Gaussian attenuator is applied on the cosine similarity; this attenuator increases the overall similarity between a new tweet t and a cluster c if they are close in time to each other. The Gaussian attenuator is made up of the difference in days between the new document t's publication time and the cluster c's time centroid. Therefore, the overall similarity formula is:

$$\delta(t, c) = cos\_sim(t, c) * e^{\frac{-(T_t - T_c)^2}{2(\sigma)^2}} \tag{5.8}$$

where $cos\_sim(t, c)$ is the cosine similarity as defined in formula 5.2. T_t is the document t's publication time and T_c the cluster c's time centroid. σ controls how fast moving the events have to be. If σ is set to a smaller value, an event expires quicker (no tweets will be added to the cluster anymore since the attenuator and consequently $\delta(t, c)$ decrease fast) while larger values keep an event longer. Sankaranarayanan et al. use $\sigma \sim 1$ in order to prevent events sticking around longer than a day (Appendix, E-Mail Conversation with Jagan Sankaranarayanan).

5.2.3 Event Clustering

Events are detected by the application of an online clustering approach which clusters in both content and time. For each cluster a feature vector is maintained, which is the vector obtained by averaging the weights of the various terms of all documents in the cluster. This vector is updated incrementally with each new document joining the cluster. Furthermore, a cluster stores and incrementally updates the time centroid; this is the mean publication time of all documents belonging to the cluster. A cluster can be either active or inactive; on a cluster's time centroid exceeding the publication time of three days, it is marked inactive and thus no documents can be added anymore. In order to make the search for a cluster c computationally feasible, an inverted index pointing from the clusters' features f (the terms) to the clusters' feature vectors containing f is stored and incrementally updated. Furthermore, this inverted index only keeps track of active clusters. When a cluster becomes labelled inactive, it is removed

from the inverted index. Having this setup, a new tweet arriving at the system, first, is represented using TF-IDF. Subsequently, the similarity $\delta(t, c)$ to all active clusters containing at least one feature of t is computed using the inverted index. If $\delta(t, c)$ reaches a predefined threshold, the new document t is added to cluster c. Otherwise, a new empty cluster is created and t added.

Sankaranarayanan et al. [SST+09] think that this sole approach does not work well and thus introduce some improvements which make it more compatible with the challenges proposed by Twitter. These improvements are described in the following sections.

5.2.4 Fragmentation

Fragmentation addresses the error of multiple clusters formed on a single topic. This happens frequently in clustering and occurs if a new document t that belongs to the topic represented by cluster c does not reach the predefined threshold and thus a new empty cluster c' is created and t added. Consequently, new documents which belong to this topic can either join cluster c or cluster c' and thus a 50% loss of the topic's documents is possible. Sankaranarayanan et al. solve this fragmentation issue by periodically iterating over all active clusters and checking if duplicate ones exist (e.g., after each interval of 100,000 documents processed). This is done by taking the cosine similarity between the clusters' feature vectors and if two clusters are close to each other (e.g., cosine similarity greater than 0.8), they are marked duplicate (Appendix, E-Mail Conversation with Jagan Sankaranarayanan). Subsequently, if a duplicate cluster is detected, the newer cluster (regarding the time centroid) is marked as master cluster while the older one becomes a slave. Then, on a new document having the highest similarity to a slave cluster, they just add the document to the slave's master cluster. Sankaranarayanan et al. find this approach working well; it practically removes fragmentation for the price of sacrificing a few documents to the slave clusters.

5.2.5 Weight Upper Bounds

Another optimization Sankaranarayanan et al. investigate in are weight upper bounds. If a new feature (an unseen term) enters the system it has a large TF-IDF value compared to the more frequent features. While this is intended in the most cases, misspelled terms or common but unimportant terms entering the system for the first time may lead to high similarities to clusters that have only this feature f in common but

are significantly different regarding the other terms. The solution deployed by Sankaranarayanan et al. is to prohibit new documents clustering just because of one feature f; at least k features (k > 1) need to be common in both the new document and the cluster which the new document is added to (we use $k = 2$).

5.2.6 Overall Algorithm for Event Detection

Algorithm 3 shows Sankaranarayanan et al's overall algorithm for event detection without the actual output generation; the output generation is described in the next section.

Algorithm 3: Sankaranarayanan et al's algorithm for event detection

input: stream of documents

```
1  X ← empty cluster list;
2  foreach document d do
3      preprocess d and represent its textual content as vector in term space using
       TF-IDF weighting;
4      S ← set of clusters containing at least two terms that are also in d (obtained
       through the inverted index);
5      sim ← 0;
6      closest_cluster ← null;
7      foreach cluster c in S do
8          c_sim = cosine_similarity(d, c);
9          g_att = gaussian_attenuator(d.time, c.time_centroid);
10         o_sim = c_sim * g_att;
11         if o_sim > sim then
12             sim ← o_sim;
13             closest_cluster ← d';
14     if sim < threshold then
15         create a new cluster c with d as first document;
16         add c to the list of clusters X;
17     else
18         add d to the closest_cluster;
19         update the closest_cluster's feature vector and time centroid;
20     add c to the inverted index pointing from word tuples to the clusters containing
       these tuples;
```

Recalling Sankaranarayanan et al.'s system, a new tweet d is preprocessed and represented as vector in term space using TF-IDF (line 3). The set of clusters S which have at least two terms in common with the new tweet d is obtained (line 4). Subsequently, the most similar cluster is determined using the cosine similarity (extended by the Gaussian attenuator); the highest similarity to the cluster as well as the cluster itself is returned (lines 5-13). If this similarity does not reach a predefined threshold, a new empty cluster is created and d added (lines 14-16). On the other hand, if the predefined threshold is reached or exceeded, the new tweet d is added to the corresponding cluster (lines 17-19). Finally, the inverted index pointing from word tuples to the corresponding clusters is updated (line 20).

5.2.7 Output Generation

Sankaranarayanan et al. do not clearly specify how to output their clustering results and thus we just output the clusters in the same way we do it for Petrovic et al's system (algorithm 2); this also makes the two systems' results easier to compare. In detail, we periodically check amongst the n fastest growing and big enough clusters and compute their entropy (formula 5.7). If a cluster's entropy does not reach or exceed a predefined threshold, the cluster is considered spam and destroyed. Otherwise, the cluster is output as event. Note that lines 8 and 9 in algorithm 2 are replaced by the time centroid specific clean-up proposed in section 5.2.3.

5.3 Aggarwal and Subbian's System

Next, we describe the event detection system proposed by Aggarwal and Subbian [AS12]. They extend the typical text-based clustering used in current state-of-the-art systems for event detection with a network component. For this purpose, they use the social network's underlying network structure (e.g., the follower-following relationship in Twitter). In this way, they attempt to improve detection performance while maintaining high efficiency and scalability through only using a fixed number of clusters. Furthermore, they apply count-min sketch tables in order to speed up similarity computations and reduce the space required for storing network node counts.

First, we describe the applied document representation and similarity computation. Subsequently, the count-min sketch data structure is introduced and its application in

the event detection process explained. Finally, we look at the output creation.

5.3.1 Document Representation and Similarity Computation

Aggarwal and Subbian use both textual features as well as network features. The network structure is made up of the senders as nodes as well as the sent messages as edges from sender to receiver. Consequently, a document S_i is represented as tuple $S_i = (q_i, R_i, T_i)$, where q_i is the document's origination node (i.e., the sender), R_i is the set of one or more receiver nodes and T_i is the document's textual content. Similar to Petrovic et al. and Sankaranarayanan et al., the textual content is represented as vector in term space using TF-IDF for the vectors' dimensions' weighting. Since there is no specific formula given for the TF-IDF calculations, we use the same incremental TF-IDF version used by Petrovic et al. (formula 5.2).

Both textual content and network structure are used for calculating the similarities. As done by Sankaranarayanan et al., similarities are computed between a new document and the existing clusters in order to determine the most similar cluster and add the document to this cluster if the similarity reaches a predefined threshold.

A cluster is represented by a tuple $\psi_i(C_i) = (V_i, \eta_i, W_i, \phi_i)$, where V_i is a set of nodes and η_i the corresponding node frequencies. V_i is obtained as the union over all nodes of each document in the cluster and η_i updated accordingly. W_i is the set of term identifiers and ϕ_i are the corresponding term frequencies (similar to the cluster's feature vector used by Sankaranarayanan et al.).

Having this document representation as well as the cluster's representation, the similarity between a new document S_i and a cluster C_r is computed as

$$Sim(S_i, C_r) = \lambda * SimS(S_i, C_r) + (1 - \lambda) * SimC(S_i, C_r) \tag{5.9}$$

where λ is a balancing parameter between the structural similarity $SimS(S_i, C_r)$ and the content similarity $SimC(S_i, C_r)$.

$SimC(S_i, C_r)$ is simply the TF-IDF based similarity between the new document's textual content and the cluster's textual content. For this purpose, the cosine similarity is used, as it is also done by Petrovic et al. (formula 5.2).

The structural similarity $SimS(S_i, C_r)$ is computed by taking the relationship of iden-

tical nodes in S_i and C_r to the overall number of nodes in S_i and C_r into account. Having the bit-vector representation $B(S_i) = (b_1, b_2, ..., b_{s_r})$ of $R_i \cup \{q_i\}$ where a bit is set for each node that is also in V_r as well as the cluster's nodes' frequency vector $\eta = (v_{r1}, v_{r2}, ..., v_{rs_r})$, the structural similarity $SimS(S_i, C_r)$ is computed as

$$SimS(S_i, C_r) = \frac{\sum_{t=1}^{s_r} b_t * v_{rt}}{\sqrt{||R_i \cup \{q_i\}||} * (\sum_{t=1}^{s_r} v_{rt})} \tag{5.10}$$

Due to the possibly large number of nodes per document (e.g, celebrities may have many millions of followers) it is clear that the structural similarity computations provide the biggest challenge of this algorithm. Therefore, the count-min sketch is introduced and applied in order to speed up these computations and reduce the amount of space needed for storing the node counts. The count-min sketch is described in the next section.

5.3.2 Count-Min Sketch

As described in the last section, Aggarwal and Subbian extend the similarity computations between tweets and clusters by a structural similarity based on the underlying network (e.g., follower-following relationship in Twitter). However, remembering this structure for each user is cumbersome (e.g., a user may have many millions of followers). Therefore, Aggarwal and Subbian use a count-min sketch data structure for speeding up the similarity computations and reducing the amount of space needed for storing the network data.

A count-min sketch is a data structure for summarising large amount of data in a space efficient way and was first introduced by Cormode and Muthukrishnan (2005) [CM05]. It solves the problem of storing a numerical value associated with an element (e.g., the number of occurrences of an element in a document stream). A count-min sketch can efficiently store such numbers by introducing a small trade off in the probability of providing an exact result on a query. Nevertheless, the structure's parameters can be tuned in such a way that a desired accuracy is reached.

In detail, the count-min sketch is a hashing approach that uses $d = \lceil ln(\frac{1}{\delta}) \rceil$ pairwise independent hash functions and each of these functions maps on one table with $w = \lceil \frac{e}{\epsilon} \rceil$ cells. Here, e is the base of the natural logarithm and the tuple (ϵ, δ) parameterizes this scheme in such a way that the error in answering a query is within ϵ with probability δ. The count-min sketch table's data structure is realized by a two-dimensional array with depth d (number of sub-tables) and width w (number of cells in each sub-table)

and thus only needs $d * w$ space. Each entry of the array is initialized with the value 0 [CM05].

The mentioned pairwise independent hash functions are constructed from a universal hash family that ensures a low number of collisions. We use the original formula as proposed by Carter and Wegman (1977) [CW77] to generate such functions:

$$h_{a,b}(x) = ((ax + b) \mod p) \mod m \tag{5.11}$$

where p is some prime greater or equal the universe to be hashed and m is the desired table size (the universe we want to map onto). Furthermore, a and b are randomly chosen integers with $1 \leq a < p$ and $0 \leq b < p$.

On an element to be counted, the sketch table is updated as follows: the element's hash value for each of the d hash functions is computed and the values at the corresponding sub-tables' cells incremented by 1 (each of the d sub-tables is updated for exactly one entry).

On a query for an element's count, the d hash functions are calculated for the given element and the value in each of the d sub-tables at the computed cell inspected. The minimum of these values is then taken as result. This makes sense, since there may be some bigger values due to collisions among hash cells during the updating process. Therefore, this minimum is the closest candidate to provide the exact result and a biased estimator; it is never less than the true value c_t for the count being estimated, making it either the true value or an overestimation (with low probability). Furthermore, a probabilistic upper bound for the estimation is given; on a given data stream with T documents having arrived already, the probability for an estimate being at most $c_t + \epsilon * T$ is at least $1 - \delta$ [AS12].

Aggarwal and Subbian utilize this scheme to lower the space for keeping track of the documents' node counts in each cluster and to accelerate the appropriate lookups. For this purpose, a sketch table is maintained for each cluster and when a new document is added to a cluster, the regarding sketch table is updated for each of the new document's node's ($R_i \cup \{q_i\}$).

The structural similarity computations (formula 5.10) are sped up in a similar way. The numerator $\sum_{t=1}^{s_r} b_t * v_{rt}$ is estimated by summing up each minimum (the result of each lookup) of values obtained by looking up the cells at the document's nodes' hash values. The denominator's first part $\sqrt{|R_i \cup \{q_i\}|}$ is given directly through the document and the second part $\sum_{t=1}^{s_r} v_{rt}$ is also known exactly by summing up all values in one of

the d sketch sub-tables. This sum provides an exact result for the overall sum of node counts in the cluster since even on a crash happening (and thus a wrong element's count being incremented) the overall count is incremented correctly.

5.3.3 Event Clustering

Next, we look at the actual event clustering process. Events are detected by the application of an online partition-based clustering method. A fixed number of k clusters are maintained together with their corresponding cluster summaries $\psi_1(C_1)...\psi_k(C_k)$. Furthermore, the mean μ and standard deviation σ of the highest similarity value between each new document and each cluster are updated incrementally. This can be done easily by additively keeping track of the zeroth, first and second order moments M_0, M_1 and M_2. Subsequently, μ and σ are computed as

$$\mu = \frac{M_1}{M_0}, \qquad \sigma = \sqrt{\frac{M_2}{M_0} - \left(\frac{M_1}{M_0}\right)^2} \tag{5.12}$$

With a new document arriving at the system, the most similar cluster is determined (by checking all clusters) and if the similarity reaches or exceeds the threshold of $\mu - 3 * \sigma$, the document is added to this cluster. Otherwise, a new empty cluster is created, the document added and the cluster which was updated the least recently is replaced with the new cluster (as long as empty clusters exist, these are replaced).

Subsequently, the changed (or newly created) cluster's summary is updated as well as μ and σ. Each node (also the document's source node q_i) of the new document S_i is added to the cluster's node set V_r and the nodes' frequency set η_i is updated appropriately (if a node is already included in V_r, its frequency in η_i is incremented by 1, otherwise the frequency is set to 1). The set of terms W_i and their corresponding frequencies ϕ_i are updated in the same way despite a term's frequency is incremented by the number of the term's occurrences in the new document.

5.3.4 Overall Algorithm for Event Detection

Algorithm 4 shows Aggarwal and Subbian's overall algorithm for event detection. The output generation in line 19 is described in the next section.

Algorithm 4: Aggarwal and Subbian's algorithm for event detection

input: stream of documents

1 $X \leftarrow$ list of k null clusters;
2 $M_0, M_1, M_2, \mu, \sigma \leftarrow 0$;
3 **foreach** *document d* **do**
4 preprocess d and represent its textual content as vector in term space using TF-IDF weighting;
5 $sim \leftarrow 0$;
6 $closest\_cluster \leftarrow$ null;
7 **foreach** *cluster* c *in* X **do**
8 $s\_sim = structural\_similarity\_cmin\_sketch(d.nodes, c.nodes)$;
9 $c\_sim = cosine\_similarity(d, c)$;
10 $o\_sim = \lambda * s\_sim + (1 - \lambda) * c\_sim$;
11 **if** $o\_sim > sim$ **then**
12 $sim \leftarrow o\_sim$;
13 $closest\_cluster \leftarrow d'$;
14 **if** $sim < \mu - 3 * \sigma$ **then**
15 create a new cluster c with d as first document and replace the most stale cluster in the system;
16 **else**
17 add d to the $closest\_cluster$;
18 update the $closest\_cluster$'s summary;
19 output the $closest\_cluster$ if it is an event happening (algorithm 5);
20 update moments M_0, M_1 and M_2 additively;
21 $\mu = M_1/M_0$;
22 $\sigma = \sqrt{M_2/M_0 - (M_1/M_0)^2}$;

Recalling Aggarwal and Subbian's system, a new tweet d is preprocessed and its textual content represented as vector in term space using TF-IDF (line 4). Subsequently, for each of the k clusters, the similarity to the new tweet is computed using both the content as well as the structural similarity and the most similar cluster as well as the similarity returned (lines 5-13). If the similarity does not reach $\mu - 3 * \sigma$, a new empty cluster is created, d added and the most stale cluster in the system replaced with the

new cluster (lines 14-15). Otherwise, d is added to the most similar cluster, the cluster's summary is updated and the cluster is inspected if it is an event happening (lines 16-19). Finally, the mean μ and standard deviaton σ of the highest similarities are updated (lines 20-22).

5.3.5 Output Generation

For outputting the system's results, Aggarwal and Subbian continuously track for so called evolution events using a time horizon H. H has to be chosen in dependence on the used data stream (i.e., how fast significant events tend to spread) (Appendix, E-Mail Conversation with Karthik Subbian). Having the current time t_c and each cluster's creation time $t(C_i)$, the ratio of number of documents added to the cluster in time interval $[t_c - H, t_c]$ to the number of documents added in time interval $[t(C_i), t_c - H]$ is calculated and if a threshold α is reached or exceeded, an alarm for an event taking place is triggered. α has to be chosen empirically (e.g., starting with $\alpha = 1$ and increase / decrease the value based on the data stream and the events' desired minimum significance) (Appendix, E-Mail Conversation with Karthik Subbian). The formula applied is as follows:

$$\frac{F(t_c - H, t_c, C_i)}{F(t(C_i), t_c - H, C_i)} \geq \alpha \tag{5.13}$$

where $F(t_1, t_2, C_i)$ denotes the fractional cluster presence; this is the number of documents added to the cluster C_i during the time interval $[t_1, t_2]$ and α the threshold. Furthermore, in order to make the above formula work in a stable way, the constraint $t_c - 2H > t(C_i)$ is introduced (i.e., the cluster has to exist for at least $2H$ in order to be considered a candidate for the output).

Having this formula, each cluster C_i's ratio is monitored continuously (Aggarwal and Subbian do not clearly specify when they check a cluster's ratio and thus we just do it whenever a new document is added to a cluster) and if it reaches or exceeds α, the cluster is output as an event taking place.

Algorithm 5 shows the output generation process.

Algorithm 5: Aggarwal and Subbian's algorithm for output generation

input: cluster c which the new document has been added to

1 **if** $t_c - 2H > t(c)$ **then**
2 **if** $\frac{F(t_c-H,t_c,C_i)}{F(t(C_i),t_c-H,C_i)} \geq \alpha$ **then**
3 output c as an event taking place;

Whenever a new tweet is added to a cluster and the cluster has already existed for at least $2H$ (line 1), the ratio of number of tweets added to the cluster in time interval $[t_c - H, t_c]$ to the number of tweets added in time interval $[t(C_i), t_c - H]$ is calculated and if it reaches or exceeds α, the cluster is output as an event happening (lines 2-3).

6 Evaluations and Corpora for Event Detection in Twitter

This chapter reveals till now common methodologies in evaluating event detection techniques in Twitter, existing problems in doing so, the underlying reasons as well as the effort taken towards better solutions. We list state-of-the-art systems and their evaluations as examples of the common proceeding. Afterwards, we propose the emerging effort in creating suitable corpora for these evaluations and introduce these corpora in detail.

As we mentioned before, the lack of suitable Twitter corpora for evaluating and comparing event detection systems' performances hinders the progress in this field. There is a need for publicly available testbeds for detailed and objective evaluations of different systems [AK13]. The reason for this circumstance is mainly caused through the massive scale of Twitter, which makes the creation of large-scale evaluation corpora difficult, time-consuming and expensive [MMJ13]. Therefore, in the most cases, the authors evaluated their system by raising the detection results on a self-collected, non-public and unannotated Twitter corpus or just used a corpus from another medium, which prevents a general rating and comparison of the systems' performances on Twitter. Indeed, Petrovic et al. [POL12] state that there is evidence for varying performance of event detection techniques on non-Twitter corpora and Twitter corpora. Besides the used datasets, the applied evaluation measures are also not satisfying: some of the authors only applied measures like average precision or precision@K, which represents the proportion of real events in the top K detected ones [AK13].

A few examples of evaluations of state-of-the-art systems for event detection in Twitter are given as follows: Sankaranarayanan et al. [SST+09] as well as Phuvipadawat and Murata [PM10] simply provide a qualitative conclusion about their systems, which is clearly subjective. Petrovic et al. [POL10] use 163.5 million self-collected tweets as dataset and evaluate their system by calculating the average precision over 1000 threads (high-volume events) of their output, which are manually labelled. Long et al. [LWC+11] collected 22 million Sina microblog posts and use precision as measure

over the number of clusters formed by their system. Weng and Lee [WYLL11] use a self-collected corpus containing 4.3 million tweets related to Singapore based users and their followers. They apply a precision evaluation on 21 events detected by their system. Cordeiro [Cor12] gathered 13.6 million tweets and offers a visual illustration of the detected events as evaluation.

This related work clearly shows the strong variety in evaluating such systems. As a consequence, the results only give a weak bias instead of a strong assessment. Common metrics, as applied in the TDT, are desirable. The TDT measures a system's performance by computing detection error trade-off (DET) curves and the minimal normalized cost [POL10]. Another more meaningful statement is the percentage of real events detected out of all real events in the corpus (recall). Consequently, all real events in the corpus have to be known and thus the corpus first has to be annotated. Common, publicly accessible and suitable corpora have become highly necessary and since 2011 a few corpora were proposed. The following summary of corpora focuses on the examination given by McMinn et al. [MMJ13].

The Text Retrieval Conference (TREC) published a corpus containing 16 million tweets for its ad-hoc retrieval task in 2011, however, their collection contains tweets in all languages (only 4 million tweets in English language) and is designed for ad-hoc retrieval, which makes the relevance judgements unsuitable for event detection evaluations in Twitter.

Becker et al. [BNG11] collected 2.6 million tweets in 2011 and annotated a training set containing 504 high-volume clusters as well as a testing set of 300 high-volume clusters by human annotators. These clusters were generated by their own system for event detection. Despite the corpus's applicability in evaluating the authors own system, it is not suitable for evaluating event detection in Twitter in general [POL12]; first, the tweets are based on New York users only, which reduces the type of events available, secondly, the clusters are generated by using their own system, which shows a strong bias towards their systems capability and thirdly, the clusters are chosen from the high-volume events only, while low-volume events are dropped. We want to mention here that Becker et. al calculates the macro-averaged F_1 measure (a combination of precision and recall) using their corpus, which is a step into the right direction of evaluating systems for event detection in Twitter.

Later (2012), Petrovic et al. [POL12] also faced the described problem and thus proposed their own corpus for evaluating event detection in Twitter together with a novel method for improving event detection performance. They created their corpus applying a similar approach to that used by the National Institute of Standards and Technology

(NIST) when creating the TDT corpora. First, they crawled around 50 million tweets between July 2011 and mid-September 2011. This timespan covers major events like the death of Amy Winehouse or the earthquake in Virginia. Secondly, they defined a set of 27 events that were likely to appear in the given timespan, using Wikipedia as well as common knowledge about important events. Thirdly, the annotators labelled an average of 112 on-topic tweets per event using event specific keywords to find appropriate tweets in the corpus and decide whether they belong to the event or not. Due to the size of the corpus they limited the window of search to a time window of one day around the specific event; this prevents too many candidates being returned. Putting all together, the corpus contains 50 million tweets, 27 events and 3035 tweets labelled as being on-topic for one of them. With the corpus only delivering the tweet IDs as well as the events and annotations, the tweets have to be crawled by the user himself, which can easily be done using one of the freely available crawlers. Petrovic et al. [POL12] used this corpus to evaluate their improved system for event detection in Twitter. Therefore, to the best of our knowledge, they are the first researches who evaluated their system on a publicly available Twitter dataset that approaches the necessary criteria of a suitable corpus for this task.

While the corpus proposed by Petrovic et al. [POL12] already contains a relatively high number of 50 million tweets and being the biggest and most suitable corpus up to this point, the labelling of only 27 events is a far too small sample and could result in misleading results [MMJ13]. Thus, McMinn et al. [MMJ13] tackled the challenge of creating a large-scale and more suitable corpus for evaluating event detection in Twitter. They point out that the approach used by Petrovic et al. for creating their corpus does not scale well past a certain size; human annotating requires expensive experts which have to identify the events carefully by hand. Therefore, McMinn et al. used three different systems, each creating a set of events. The systems used are as follows: (i) the system for event detection proposed by Petrovic et al. [POL10] in 2010 (ii) the system for event detection proposed by Aggarwal and Subbian [AS12] in 2012 (iii) the Wikipedia Current Events Portal[1] (CEP). Human annotators assessed the resulting sets, which led to 382 events for (i), 53 events for (ii) and 342 events for (iii). Subsequently, they applied a self-developed event clustering approach to combine the sets, which led to a final number of 506 top-level events in the corpus. McMinn et al. provide 152,950 relevant tweets' IDs as being on-topic for these events. To the best of our knowledge, this results in the by far biggest corpus for event detection in Twitter containing over 120 million tweets. Furthermore, the corpus is publicly available for further research and development. Since we do an evaluation and comparison using the systems proposed by Petrovic et al. [POL10] as well as Aggarwal and Subbian [AS12], it may seem unfair to use this corpus because these systems were also used

[1] http://en.wikipedia.org/wiki/Portal:Current_events (accessed on March 5, 2014)

to create this corpus. Nevertheless, the application of these two detection methods only produced 186 top-level events, while the Wikipedia Current Events Portal contributed 342 [MMJ13]; the resulting event set, which was produced by Petrovic et al.'s system and used in the corpus's annotating process, is highly compressed and modified by the annotators. For instance, the third U.S. Presidential debate produced 40 sub-events which were clustered together to meet the definition of event. Hence, we assume that the advantage of the named systems is negligible. Therefore, we decide to use this corpus in this thesis.

7 Evaluation

In this chapter we evaluate the three event detection systems using the Events 2012 Twitter Dataset proposed by McMinn et al. [MMJ13]. Having this labelled corpus, we are able to compute common measures (e.g., precision and recall) and give an objective assessment for the systems' performance compared to each other.

First, we provide information about obtaining the corpus. Subsequently, we discuss necessary postprocessing steps; the final output of the event detection systems is merged in order to make a better evaluation possible. The evaluation measures we apply are introduced and subsequently we take a deeper look at the systems' parameters we have to set. Finally, we evaluate the chosen system setups on the full corpus and assess the results.

7.1 Obtaining the Corpus

As proposed in the last chapter, we decided to use the Events 2012 Twitter Dataset created by McMinn et al. [MMJ13]. The corpus only provides the tweet IDs, the corresponding user IDs as well as the labelled events (with tweet IDs of tweets being labelled on-topic); due to Twitter's regulations, the corpus has to be crawled by the user himself. We used a publicly available crawler (e.g., twitter-dataset-collector[1]). Originally, the corpus consists of 121,747,031 tweet IDs, however, due to tweets and user accounts got deleted or suspended as well as other restrictions, we were only able to crawl 90,078,794 tweets (we also performed a few recrawls). Knowing the set of 31,668,237 missing tweets, we removed the tweets which we were not able to crawl from McMinn et al's labelling (14,534 unique tweets removed of 152,950 labelled tweets).

[1] twitter-dataset-collector, https://github.com/socialsensor/twitter-dataset-collector (accessed on August 5, 2014)

In addition, in order to evaluate Aggarwal and Subbian's approach, we were interested in the users' followers. Aggarwal and Subbian utilize this user-follower network for computing a structural similarity between a new tweet and a cluster (section 5.3.1). However, obtaining this data is cumbersome. The 90,078,794 tweets we were able to crawl were written by 15,340,769 unique users and each of them can have an unbound amount of followers (e.g., Justin Bieber has over 50 million followers). Not only the huge amount of data but also Twitter's properties in obtaining this data makes it a much more challenging task than crawling the corpus itself; accessing a user's web profile only provides a small number of followers at once and only delivers more results if needed (making it hard for a web-crawler) and the Twitter REST API restricts a user to 15 requests per 15 minutes and only returns a maximum of 5000 followers per request. As a consequence, for the complete corpus run, we had to evaluate Aggarwal and Subbian's system without using the network structure (as also done by McMinn et al. [MMJ13] when creating the corpus, since they faced the same problems in obtaining the followers data). Nevertheless, we performed a smaller evaluation of Aggarwal and Subbian's approach with network structure similarity computations. Therefore, we crawled a small part of users' followers in the following way: we ran the three event detection approaches introduced in this work on the first two million tweets of McMinn et al.'s corpus. This generated a set of relevant events (judged by looking at McMinn et al.'s event labelling). Subsequently, we only crawled the follower IDs of users who wrote labelled tweets about the corresponding events. This concluded in a set of 2043 unique users and 1,971,245 follower IDs. We used the Twitter REST API and thus we only obtained the 5000 latest followers per user per request. Having this set of users and their followers, we think we can provide a bias if Aggarwal and Subbian's approach works, however, the evaluation is incomplete and a full run having each user's followers would be desirable but with Twitter's current limitations it is hard to obtain this data.

7.2 Merging the Output

In order to assess the event detection systems' performance by using the labelled corpus, we first have to decide how to use the systems' output, how to compare the output with the labelled events / tweets and what evaluation measures to use.

The introduced event detection systems usually output their results in predefined time intervals (Petrovic et al. and Sankaranarayanan et al.) or if a cluster is growing at a specific rate (Aggarwal and Subbian). However, in many cases, an output cluster is just a fraction of the actual event and it may be output again at a later time as a new

cluster (since the old cluster has already disappeared from the system). Since we do not only want to catch the events but also the ratio of tweets labelled by McMinn et al. we were able to detect, we merge the systems' output in such a way that clusters talking about the same event are merged together; this allows us to catch more tweets and thus possibly more labelled tweets for the corresponding event.

We do so in two subsequent steps. First of all, for each cluster in the output, we look for the most similar cluster using the cosine similarity measure and if the two clusters are similar (e.g., a cosine similarity reaching or exceeding the merging threshold $t = 0.8$), the clusters are merged together. We think this approach works well since two clusters discussing the exact same event tend to have many of the event specific terms in common and these terms tend to have high TF-IDF weightings (uncommon term in the whole system but common for this event). Consequently, high cosine similarities are reached easily.

Nevertheless, this merging step fails to merge sub-events. As an example, consider an election. This is an event itself but can also be broken down into many sub-events (e.g., the speech of a politician). Algorithmic event detection systems, as we described them in this work, tend to detect these sub-events (due to people in Twitter mainly writing about such sub-events) and these clusters are not merged together in the previously described merging step since the main keywords, which made the sub-event cluster together, differ from the other sub-events (e.g., consider the main keywords "obama, speech, washington" and "paul, ryan, debate") and thus the cosine similarity between such clusters is low. Nevertheless, we want to measure the ratio of labelled tweets the systems were able to find for an event as it is labelled in McMinn et al.'s corpus. Since the merging using a textual similarity like the cosine similarity does not perform well in this case, we introduce an event fraction threshold (e.g., 0.05); for each cluster in the output, we look for the labelled event in the corpus having the highest ratio of tweets in common and if this ratio reaches or exceeds the event fraction threshold, we mark the cluster as successfully detected event fraction of the corresponding event. Finally, we sum up the tweet ratios of an event's detected event fractions and if this sum reaches or exceeds an event threshold (e.g., 0.15) we consider the event as detected and merge the event fractions.

Due to this second step merging all event fractions of a labelled event, the first merging step, which uses the cosine similarity, may seem unnecessary. However, if we skip the initial merging step, event fractions that would have been merged together in this first step may not contain enough of the labelled tweets to reach the event fraction threshold in the second step anymore and thus be discarded. We tested the three systems with and without the initial merging step and the application of both steps usually performed

best (mainly due to higher precision caused by less clusters returned), however, the difference is negligible. Therefore, we use both steps in our setup.

7.3 Evaluation Measures

After merging the output in the way described in the last section, we can evaluate the results. Precision, recall and the F score are typical performance measures in the information retrieval task [AK13] and provide an objective measure in the given scenario. Precision is the ratio of relevant documents detected over all documents detected. Recall is the ratio of relevant documents detected over all relevant documents. The F score, as proposed by Van Rijsbergen [vR79], computes the weighted harmonic mean of precision and recall and uses β as a weighting parameter, where recall is β times more important than precision. The three measures are computed as follows:

$$\text{precision} = \frac{|\{\text{relevant documents}\}| \cap |\{\text{detected documents}\}|}{|\{\text{detected documents}\}|} \tag{7.1}$$

$$\text{recall} = \frac{|\{\text{relevant documents}\}| \cap |\{\text{detected documents}\}|}{|\{\text{relevant documents}\}|} \tag{7.2}$$

$$F_\beta = (1 + \beta^2) * \frac{\text{precision} * \text{recall}}{(\beta^2 * \text{precision}) + \text{recall}} \tag{7.3}$$

Having our detection setup, we use these measures as follows: we compute precision, recall and the F score for both events and tweets. With own words, precision over events is the ratio of labelled events we are able to detect out of all detected events and recall over events is the ratio of labelled events we are able to detect out of all labelled events. The F score over events combines both precision over events and recall over events. This works as a measure for how good we detect events in general. When computing precision, recall and F score over tweets, we just look at the labelled events we are able to detect. Here, precision over tweets is the ratio of labelled tweets we are able to detect out of all detected tweets and recall over tweets is the ratio of

labelled tweets we are able to detect out of all labelled tweets. The F score over tweets combines both precision over tweets and recall over tweets. Therefore, this works as a measure for how good we detect the events' tweets.

Looking at these measures, there are a few issues we had to take care about. McMinn et al's labelling consists of 506 events, however, as already described in the last subsection, the systems in this work frequently just output sub-events belonging to one of these events or events that are not labelled at all. An example is given in table 7.1: it shows three sub-events detected by Petrovic et al's system (Missy Elliot, Snoop Dogg and Diddy performing). Each sub-event belongs to the BET hip hop awards 2012. However, this overall event is not labelled in the McMinn et al. corpus (only a few other sub-events of the BET hip hop awards 2012), but it appears in the system's output through many sub-events (Petrovic et al.'s system caught over 30 of those sub-events belonging to the BET hip hop awards 2012). Since the overall event and many of the sub-events are not labelled in the corpus, the precision is lowered significantly, however, the system performed well detecting these sub-events. Looking at the labelled tweets for an event, we observed the same problem. The number of tweets labelled for a given event tends to be far too low compared to the number of tweets detected by the systems used in this work. For example, the singer Omarion danced on the stage at BET hip hop awards and this sub-event is labelled in the McMinn et al. corpus having 269 labelled tweets. Using Petrovic et al's system, we detected at least 2000 tweets for this sub-event, however, we only detected 169 of the 269 labelled tweets (recall $= \frac{169}{269} \approx 0.628$, precision $= \frac{169}{2000} \approx 0.085$, if we detected exactly 2000 tweets for this subevent). This example shows that the precision over tweets, similar to the precision over events, only gives a weak assessment in this scenario; the system may work well but produces low precision values. Therefore, we decide to use the F_2 score ($F_2 \approx 0.276$ in the given example) instead of the F_1 score ($F_1 \approx 0.15$ in the given example). Consequently, recall is given more importance, however, precision still works as balancing factor for the corresponding recall value. This makes sense due to the possibility of tuning a system in such a way that it only produces either high precision or recall values. Consider the system returning many specific and small clusters (much more than labelled events). In this case it is likely that we catch a big part of the labelled events and thus the recall over events is high, however, the precision over events is low. On the other hand, consider the system returning only a few big and broad clusters (much less than labelled events). Here, it is likely that the few clusters returned are events and thus the precision over events is high, however, the recall over events is low. The opposite behaviour occurs for precision and recall over tweets. If the system returns many specific and small clusters, we may lose labelled tweets to other clusters (which may not be returned as event fraction) and thus the recall over tweets is low, however, the precision over tweets is high. On the other hand, if the system only

returns a few big and broad clusters and these are labelled events, the precision over tweets is low, however, the recall over tweets is high.

Having a system's F_2 score over both events and tweets, we compute an overall assessment score F_{final} for the system's performance. Therefore, we combine the F_2 score over events with the F_2 score over tweets by computing the (unweighted) harmonic mean between them (the unweighted harmonic mean is equal to the F_1 score computation). This gives the same importance to the system's performance in detecting events in general as well as to how good it detects the events' tweets.

Table 7.1: Sub-event example

cluster size: 568 entropy: 5.117233526028828
love - elliot - missy
elliot - wtf - missy
yo - freak - elliot - missy
favorite - elliot - missy
shit - awww - elliot - missy
shit - HASH_hiphopawards - elliot - holy - missy
HASH_betcyphers - legend - HASH_bethiphopawards - elliot - missy
hype - elliot - mom - missy
turn - straight - elliot - missy
...
cluster size: 1280 entropy: 5.864526620350165
found - snoop - lion - uncle - 2chainz
dogg - snoop - kendrick - lamar - HASH_hiphopawards - game - betta
thought - dogg - snoop - changed - lion - fuckin
snoop - dog - HASH_hiphopawards
thought - lmao - snoop - changed - lion - nigga
hosting - dogg - snoop - tf - home - cypher
snoop - dog - hear - ready - lion
thought - dogg - snoop - HASH_hiphopawards - lion
ahh - shit - excited - im - dogg - snoop - westside - cypher
...
cluster size: 1900 entropy: 6.705120710478181
fin - watch - diddy - jam
lmaooo - diddy - kill
damn - time - remix
HASH_bethiphopawards - diddy
ass - rich - diddy
diddy - tho - fucked
bout - diddy - kill
turn - diddy - back
lil - love - cousin - song - diddy - pa
...

7.4 Parameters

Having the three system (Petrovic et al. [POL10], Sankaranarayanan et al. [SST+09] as well as Aggarwal and Subbian[AS12]), the McMinn et al. corpus [MMJ13] as well as the described output merging and evaluation measures, we are able to run the evaluation. What is left is setting the necessary parameters. Since a run on the full corpus is time-consuming, we determine suitable parameters for the different steps by performing tests on the first two million tweets of the McMinn et al. corpus. First, we investigate the steps which are common to all three systems. These include preprocessing, removing spam (usage of entropy) and the output merging. Subsequently, we test the system specific parameters. The most promising setups are finally used for the full evaluation. Note, that the parameter testing is time consuming and thus we were just able to test a small subset of possible parameter combinations. Usually, the systems' authors provided a parameter evaluation which helped us in determining suitable values.

7.4.1 Preprocessing

Each new tweet is preprocessed before being forwarded to the actual event detection process; we try to remove or modify textual content that does not add to the detection process in a positive manner.

Table 7.2 shows three examples of the preprocessing we applied. The subsequent steps are explained in the following paragraphs. In order to measure the different steps influence on the systems' performance, we ran a few tests for each system on the first two million tweets of the McMinn et al. corpus; table 7.3 shows the results.

First of all, we removed mentions (starting with an "@" character and followed by a username), retweet markup (character sequence "RT"), URLs and signs that are no characters or digits. These data rarely contributes to the content of a tweet and thus the removal should always improve the performance of an event detection system. [Pet12]. Petrovic finds that removing username mentions and links from tweets improves the results substantially [Pet12]. This is confirmed by our results; the performance of all three systems improved.

In the next step, we removed stopwords; stopwords are frequent terms in a language that do not contribute to the informative content (e.g., "the", "it", "a"). Therefor, we

used an English stopword list (e.g., the SMART stopword list[2]). It has been shown that stopword removal in the information retrieval search task usually improves the results when compared to no stopword removal [DS10]. The systems' performance after removing stopwords is unexpected; the performance of Petrovic et al.'s as well as Aggarwal and Subbian's system deteriorated slightly while Sankaranarayanan et al.'s system's performance decreased significantly.

In the last step, we performed stemming; stemming is the process of reducing words to their root form and a frequently used step in information retrieval systems. However, Petrovic et al. [Pet12] find stemming used on tweets to hurt the performance; they conclude that this is due to stemmers being designed for clean data, however, Twitter's data is very noisy. Having our results, we agree looking at Petrovic et al.'s as well as Aggarwal and Subbian's system, however, Sankaranarayanan et al.'s system 's performance improved when using stemming.

Putting it together, this preprocessing evaluation does not provide a clear result. Removing mentions, retweet markup, URLs and signs that are no characters or digits clearly improves the results. However, the outcome of stopword removal and stemming varies. Further tests have to be done which we leave open for further research. Since we seek to perform runs on the full McMinn et al. corpus, which is timely intense, we decide to use the preprocessing steps which make the systems have low running time while maintaining good performance. Therefore, we decided to remove stopwords (which lowers the overall processing time significantly) but do not apply stemming. Since hashtags contribute to the content of tweets [Pet12], we keep them in the tweets, however, we treat them as normal terms (giving hashtags more weight would probably lead to better results, but we decided not to so since two of the systems are designed being universal and not only for Twitter).

[2]SMART stopword list, http://jmlr.org/papers/volume5/lewis04a/a11-smart-stop-list/english.stop (accessed on August 8, 2014)

Table 7.2: Three examples of the preprocessing

RT @gerhard test123 :) http://www.test.de I am you organize organizing #owned
test123 i am you organize organizing HASH_owned
test123 organize organizing HASH_owned
test123 organ organ HASH_own
we are all going to twitter jail tonight because of the #HipHopAwards
we are all going to twitter jail tonight because of the HASH_hiphopawards
twitter jail tonight HASH_hiphopawards
twitter jail tonight HASH_hiphopaward
@friend12345: I go deaf when I'm texting. http://www.page123.com
i go deaf when i m texting
deaf texting
deaf text

Table 7.3: Preprocessing results

system	removed mentions, RTs, URLs, misc	removed stopwords	stemmed	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
petro	-	-	-	0.01375	0.22288	0.02591
petro	x	-	-	0.01821	0.29187	0.03429
petro	x	x	-	0.01790	0.21368	0.03303
petro	x	x	x	0.01750	0.16673	0.03168
sanka	-	-	-	0.01103	0.27563	0.02122
sanka	x	-	-	0.01428	0.26062	0.02709
sanka	x	x	-	0.01151	0.29132	0.02214
sanka	x	x	x	0.01342	0.27495	0.02559
agga	-	-	-	0.01628	0.31111	0.03095
agga	x	-	-	0.01648	0.32621	0.03137
agga	x	x	-	0.01634	0.33372	0.03116
agga	x	x	x	0.01619	0.32479	0.03085

7.4.2 Retweets

Around 30% of the tweets in the McMinn et al. corpus are retweets [MMJ13]. Since retweets are frequently used to spread spam and thus cause systems to wrongly output clusters as events and because they are just an unmodified copy of another tweet, McMinn et al. decide to not include retweets in the labellings, however, they are kept in the corpus. Therefore, looking at our evaluation measures, a system detecting many retweets of a given event has a lower precision over tweets than not detecting these retweets. Nevertheless, we performed some tests on the first two million tweets of the McMinn et al. corpus in order to determine if we should use retweets or not. Table 7.4 shows our results. The performance of Petrovic et al.'s as well as Aggarwal and Subbian's system improved when removing retweets; as expected, $F_{2_{tweets}}$ increased

significantly while $F_{2_{events}}$ stayed on the same level (i.e., same number of clusters output and relevant events detected). However, Sankaranarayanan et al.'s system was able to detect more events leading to a higher $F_{2_{events}}$. Therefore, we decide to remove retweets for Petrovic et al. as well as Aggarwal and Subbian but test Sankaranarayanan et al.'s system using both setups.

Table 7.4: Retweets results

system	removed retweets	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
petro	-	0.01746	0.15684	0.03143
petro	x	0.01771	0.20845	0.03265
sanka	-	0.01539	0.27835	0.02916
sanka	x	0.01150	0.29128	0.02213
agga	-	0.01613	0.25526	0.03034
agga	x	0.01634	0.33372	0.03116

7.4.3 Entropy

Another step that can be used for all three systems is the usage of the clusters' entropies H (formula 5.7) in order to remove spam as it was initially applied by Petrovic et al. [POL10]. We ran a few tests for each system and found that removing clusters with small entropy always improves the results. Table 7.5 shows the results. Each system's performance improved until the entropy threshold 3.5 (i.e., we only output clusters which entropy reaches or exceeds the threshold 3.5). Higher entropy thresholds (4.5 and 5.5) performed worse and at an entropy threshold of 7.0 no clusters are output anymore (Petrovic et al.'s system still outputs a few clusters but the performance is already lowered significantly). Note that this step only influences $F_{2_{events}}$ directly; it has no effect on the tweets itself in the output clusters. The results for Sankaranarayanan et al.'s system and Aggarwal and Subbian's system show this effect. $F_{2_{tweets}}$ remains constant for the first entropy values while $F_{2_{events}}$ improves. A change in $F_{2_{tweets}}$ indicates a change in the number of clusters that are output (e.g., entropy threshold 5.5 for Sankaranarayanan et al's system or 4.5 for Aggarwal and Subbian's system) and thus $F_{2_{tweets}}$ differs depending on the tweets in these clusters. The fluctuations of $F_{2_{tweets}}$ for Petrovic et al.'s system are caused through the locality sensitive hashing approach being approximate (as described in section 5.1.2). Having these results, we decide to only output clusters reaching or exceeding the entropy threshold of 3.5 (among the best results of each system). Note that McMinn et al. applied spam filtering when creating the corpus and thus a lot of spam is removed already; this explains the rather small performance benefit when using entropy. Applied on unfiltered data, a bigger improvement should be monitored.

Table 7.5: Entropy results

system	entropy H	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
petro	0.0	0.01557	0.21625	0.02905
petro	2.5	0.01748	0.20920	0.03227
petro	3.5	0.01749	0.22834	0.03249
petro	4.5	0.01775	0.24435	0.03309
petro	5.5	0.01629	0.21198	0.03026
petro	7.0	0.00491	0.17341	0.00955
sanka	0.0	0.01185	0.26427	0.02269
sanka	2.5	0.01187	0.26427	0.02271
sanka	3.5	0.01191	0.26427	0.02280
sanka	4.5	0.01196	0.26427	0.02288
sanka	5.5	0.00486	0.27127	0.00954
sanka	7.0	0.00000	0.00000	0.00000
agga	0.0	0.01206	0.331237	0.02327
agga	2.5	0.01207	0.331237	0.02330
agga	3.5	0.01211	0.331237	0.02337
agga	4.5	0.01212	0.284687	0.02324
agga	5.5	0.00975	0.224334	0.01868
agga	7.0	0.00000	0.000000	0.00000

7.4.4 Merging the Output

In section 7.2 we described the output merging step which is done in order to enable a better comparison between the systems' output and McMinn et al.'s labelling. In order to find suitable output merging parameters, we performed tests for each system, which are discussed in this section. The output merging parameters are as follows: (i) the merging threshold to reach or exceed in order to merge two output clusters together (using the cosine similarity between the clusters) (ii) the event fraction threshold which a cluster needs to reach or exceed in order to be labelled an event fraction (iii) the event threshold which a (set of) cluster(s) need(s) to reach or exceed in order to be labelled a successfully detected event.

Table 7.6 shows the tests' results. The green lines mark the setups which performed best for each system. The results give us a clear bias for suitable parameters. A merging threshold of 0.3 leads to clusters being merged wrongly. The cluster may still be regarded as an event detected (due to reaching the event fraction threshold and the event threshold) but contain many off-topic tweets which can be derived from the low $F_{2_{tweets}}$ values. On the other hand, if we do not execute the cosine similarity merging (merging threshold 1.0) we risk to lose clusters due to not reaching the necessary event fraction threshold and event threshold. In this case, the results are still acceptable,

however, the application of merging if two clusters are similar (merging threshold 0.8) performed best. When looking at the event fraction threshold, we do not notice a big difference between the event fraction threshold 0.02 and 0.05, however, we intuitively think that a cluster should at least reach the ratio 0.05 of the corresponding labelled event's tweets in order to be labelled an event fraction. The event threshold performed best being set to 0.10; we intuitively think that this value is low but if we set it higher (e.g. event threshold 0.50) we lose a lot of events due to not catching enough tweets. Therefore, we decide to use the event threshold 0.10; if a system is able to detect more events due to this low threshold, the system's bad performance in detecting the event's labelled tweets is reflected in $F_{2_{tweets}}$ and thus F_{final} (and we think this is better than frequently throwing away events in order to reach high $F_{2_{tweets}}$ values). In conclusion, we use the following setup: merging threshold 0.8, event fraction threshold 0.5 and event threshold 0.10. These parameter setup is also among the four best setups of each system.

Table 7.6: Output merging results

system	merging threshold	event fraction threshold	event threshold	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
petro	0.3	0.02	0.10	0.01848	0.10351	0.03136
petro	0.5	0.02	0.10	0.01610	0.16169	0.02928
petro	0.8	0.02	0.10	0.01582	0.17678	0.02908
petro	1.0	0.02	0.10	0.01756	0.21715	0.03249
petro	0.3	0.05	0.10	0.01312	0.06909	0.02205
petro	0.5	0.05	0.10	0.01787	0.23346	0.03319
petro	0.8	0.05	0.10	0.01771	0.23165	0.03292
petro	1.0	0.05	0.10	0.01487	0.20056	0.02769
petro	0.3	0.10	0.20	0.01605	0.11233	0.02808
petro	0.5	0.10	0.20	0.01560	0.17968	0.02872
petro	0.8	0.10	0.20	0.01479	0.26911	0.02805
petro	1.0	0.10	0.20	0.01421	0.27368	0.02702
petro	0.8	0.05	0.15	0.01719	0.24344	0.03212
petro	0.8	0.05	0.25	0.01571	0.23445	0.02945
petro	0.8	0.10	0.50	0.00604	0.39161	0.01190
sanka	0.3	0.02	0.10	0.01182	0.27441	0.02266
sanka	0.5	0.02	0.10	0.01199	0.29426	0.02305
sanka	0.8	0.02	0.10	0.01196	0.38269	0.02320
sanka	1.0	0.02	0.10	0.01186	0.38538	0.02302
sanka	0.3	0.05	0.10	0.01183	0.27441	0.02267
sanka	0.5	0.05	0.10	0.01200	0.29426	0.02305
sanka	0.8	0.05	0.10	0.01196	0.38268	0.02320
sanka	1.0	0.05	0.10	0.01186	0.38538	0.02302
sanka	0.3	0.10	0.20	0.00946	0.28949	0.01831
sanka	0.5	0.10	0.20	0.00959	0.34438	0.01866
sanka	0.8	0.10	0.20	0.00957	0.48610	0.01877
sanka	1.0	0.10	0.20	0.00949	0.49077	0.01862
sanka	0.8	0.05	0.15	0.00961	0.42164	0.01879
sanka	0.8	0.05	0.25	0.00961	0.42164	0.01879
sanka	0.8	0.10	0.50	0.00482	0.54214	0.00955
agga	0.3	0.02	0.10	0.01202	0.30718	0.02313
agga	0.5	0.02	0.10	0.01204	0.31597	0.02319
agga	0.8	0.02	0.10	0.01207	0.31597	0.02324
agga	1.0	0.02	0.10	0.01202	0.35589	0.02325
agga	0.3	0.05	0.10	0.01193	0.33196	0.02302
agga	0.5	0.05	0.10	0.01200	0.34244	0.02320
agga	0.8	0.05	0.10	0.01204	0.34244	0.02326
agga	1.0	0.05	0.10	0.01199	0.36363	0.02321
agga	0.3	0.10	0.20	0.01193	0.33196	0.02302
agga	0.5	0.10	0.20	0.01192	0.34571	0.02304
agga	0.8	0.10	0.20	0.01198	0.34571	0.02315
agga	1.0	0.10	0.20	0.01191	0.36356	0.02307
agga	0.8	0.05	0.15	0.01204	0.34244	0.02326
agga	0.8	0.05	0.25	0.01204	0.34244	0.02326
agga	0.8	0.10	0.50	0.01198	0.34571	0.02315

7.4.5 Petrovic et al.'s System

This section focuses on finding the most suitable system specific parameters for Petrovic et al.'s system, which was introduced in section 5.1. Due to many possible setups, we divided the parameters into the most related parts. These are the system's output parameters, the locality sensitive hashing parameters, the bucket size, the similarity threshold and the variance reduction strategy parameter.

7.4.5.1 Output

First, we investigate in the output generation of Petrovic et al.'s system. Therefore, we tested different setups for the following parameters: (i) the output timer that determines the time intervals after which the output is generated (ii) the maximum number of fastest growing and big enough clusters (see third parameter) which are inspected for the output generation (their entropy is checked and if it reaches or exceeds the entropy threshold they are output) (iii) the minimum number of tweets in a cluster in order for being considered a candidate for the output.

With our own words, we tried to determine a parameter setup that affects the system in such a way that it outputs all possibly relevant clusters but nothing more. This means that we look for a minimal number of fastest growing and big enough clusters to inspect in relation to the output time intervals in such a way that we do not miss any relevant clusters. In the same moment, we search for the maximal number of tweets a relevant cluster always contains without dropping any relevant cluster because of not reaching that number; this prevents clusters with only a few tweets being returned (which may have a big enough growth rate) while not loosing relevant clusters.

Table 7.7 shows the tests' results. The four best working setups are marked green.

Table 7.7: Output generation results

output timer	number of fastest growing & big enough clusters inspected	minimum tweets in cluster	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
100,000	50	25	0.01813	0.19046	0.03310
200,000	50	25	0.01849	0.16965	0.03335
500,000	50	25	0.01426	0.23312	0.02688
100,000	100	25	0.01744	0.19818	0.03206
200,000	100	25	0.01800	0.19790	0.03300
500,000	100	25	0.01860	0.15167	0.03313
100,000	200	25	0.01587	0.25289	0.02987
200,000	200	25	0.01735	0.19160	0.03181
500,000	200	25	0.01755	0.20579	0.03234
100,000	50	50	0.01835	0.23947	0.03409
200,000	50	50	0.01864	0.17988	0.03378
500,000	50	50	0.01189	0.15513	0.02208
100,000	100	50	0.01768	0.21406	0.03266
200,000	100	50	0.01804	0.23034	0.03345
500,000	100	50	0.01390	0.16865	0.02568
100,000	200	50	0.01784	0.22806	0.03309
200,000	200	50	0.01547	0.01833	0.02855
500,000	200	50	0.01582	0.14588	0.03257
100,000	50	100	0.01628	0.22477	0.03037
200,000	50	100	0.01662	0.19312	0.03061
500,000	50	100	0.01423	0.17111	0.02627
100,000	100	100	0.01612	0.22319	0.03007
200,000	100	100	0.01617	0.20473	0.02997
500,000	100	100	0.01413	0.14886	0.02582
100,000	200	100	0.01631	0.20400	0.03021
200,000	200	100	0.01651	0.19645	0.03046
500,000	200	100	0.01621	0.19533	0.02994
100,000	50	200	0.01679	0.17567	0.03065
200,000	50	200	0.01678	0.18606	0.03078
500,000	50	200	0.01199	0.25108	0.02289
100,000	100	200	0.01651	0.20114	0.03051
200,000	100	200	0.01681	0.22409	0.03127
500,000	100	200	0.01673	0.18847	0.03076
100,000	200	200	0.01673	0.19608	0.03084
200,000	200	200	0.01669	0.18900	0.03066
500,000	200	200	0.01643	0.17177	0.02999
100,000	500	500	0.01702	0.20102	0.03138
200,000	500	500	0.01219	0.18544	0.02288
500,000	500	500	0.01692	0.13284	0.03002

7.4.5.2 Locality Sensitive Hashing

Next, we look at the parameters which influence the locality sensitive hashing (LSH) scheme used by Petrovic et al. and introduced in section 5.1.2. These are as follows: (i) the number of LSH tables L (ii) the number of hyperplanes k.

These two parameters influence the LSH approach as a trade-off between detection accuracy and running time [Pet12]. More LSH tables increase the chance of finding the nearest neighbour, however, more calculations have to be done. Increasing the number of hyperplanes leads to less collisions in a LSH table and thus to less similarity computations in the LSH step but the chance of finding the nearest neighbour is lowered.

As introduced in section 5.1.2, Petrovic et al. propose the formula $L = \log_{1-P_{coll}^k} \delta$ (formula 5.6) to compute the number of LSH tables L, where P_{coll}^k is the probability of two points p and q colliding into the same bucket in one LSH table with k hyperplanes and δ is the desired probability of missing the nearest neighbour in all L LSH tables. Since we were not sure how to set P_{coll}^k and δ in order to achieve a good overall performance, we tested different setups for L and k empirically.

Table 7.8 shows the tests that we performed. Note that we used LSH in combination with the variance reduction strategy in order to see LSH's parameters' influence on the running system. Therefore, if the system was able to find a nearest neighbour with high enough similarity (similarity threshold), we noted if it was found through LSH or the variance reduction strategy. This helps to better understand the influence of L and k on the overall system.

Looking at L with fixed k, we can see that increasing L past a value of 10 does not improve the overall performance, which seems confusing in the first moment; more LSH tables increase the chance of finding the nearest neighbour at the cost of higher runtime and thus should increase the overall performance. However, if LSH fails, the variance reduction strategy is used and thus for lower L the variance reduction strategy does more work which leads to a similar performance compared to higher L. This can be seen looking at the numbers found by LSH and found by the variance reduction strategy; increasing L for fixed k leads to increased numbers found through LSH but decreased numbers found through the variance reduction strategy. Summing up the number of found through LSH and the number of found through the variance reduction strategy results in fairly similar values for increasing L and fixed k. Note that this does not mean that the results are identical; LSH returns a nearest neighbour no matter

when it was written, while the variance reduction strategy focuses on recency due to only comparing to a fixed number of most recent tweets [Pet12]. Looking at the running time, changing L produces results in the way expected; using more LSH tables causes higher running time due to more hashing and more similarity computations for each new document and vice versa for less LSH tables. More LSH tables also cause the variance reduction strategy to be used less, however, this win in running time does not outweigh the loss caused through higher running time of LSH.

When looking at the number of hyperplanes we observe a behaviour which is independent from the number of LSH tables. The running time increases until a number of 10 to 11 hyperplanes and decreases again for higher k. In detail, having a small k leads to only a few buckets (2^k) in the LSH tables and thus the number of overall possible nearest neighbour candidates is low. Note that the inverted index used for the variance reduction strategy is also only holding a tweet as long as it is represented in at least one of the L LSH tables [Pet12] and thus does not contain many tweets for low k. Therefore, the overall number of possible nearest neighbour candidates and thus similarity computations is low which causes the low running time. Increasing k leads to more tweets in the LSH tables as well as the inverted index and consequently higher running time through more similarity computations. After exceeding 11 hyperplanes, the hashing becomes more selective (only highly similar tweets are hashed into the same bucket); as from now, for each new tweet hashed, the buckets tend to contain less tweets which are more similar and thus the number of similarity computations through LSH lowers which causes less nearest neighbours found through LSH and a lower running time; this can be seen looking at the decreasing numbers of nearest neighbours found through LSH after reaching a peak at 12 to 15 hyperplanes.

We find it interesting that the overall performance is highest using 7 hyperplanes for each L we tested; having 7 hyperplanes and 30 tweets as maximum capacity in a bucket leads to at most $2^7 * 30 = 3840$ tweets hold by one LSH table (and a big ratio of duplicate references in the other LSH tables). We think this shows that the performance is strongly influenced by the timely connected characteristics of an event; having only a few thousand most recent tweets as possible nearest neighbour candidates guarantees that the clusters created contain timely connected tweets and thus we implicitly model time. Therefore, we decided to evaluate this setup ($L = 10$, $k = 7$) as well as the setups suggested by Petrovic et al. ($L = 10$, $k = 13$ and $L = 70$, $k = 13$) on the full corpus.

Another issue we find interesting about the prior described behaviour is that it is exactly the opposite which was monitored by Petrovic et al. [Pet12] when evaluating their system on the TDT5 dataset; here, the running time decreases starting with 5 hyperplanes until a number of 10 to 11 hyperplanes and increases again for higher k. Petrovic et

al. explain the observations they made through the combination of LSH and the variance reduction strategy [Pet12] (as we do); a lower k leads to more frequent usage of the variance reduction strategy and thus the running time is higher. As k increases, more time is spent for hashing, however, the variance reduction strategy is used less (leading to lower running time) due to LSH returning more similar nearest neighbour candidates. If k exceeds 10 to 11 hyperplanes, the hashing time further continues to increase while the number of variance reduction strategy usages stays same, which leads to overall higher running time.

We think the reason for this differing outcome in running time is the difference between tweets and documents in the TDT5 dataset; after preprocessing, a tweet only contains a few terms, while a document in the TDT5 dataset still may contain many hundreds to thousands of terms. As a consequence, even for only 5 hyperplanes used, the variance reduction strategy is able to compare a new document of the TDT5 dataset to a much bigger number of documents than if applied the system is working with tweets; the reason for this can be found in the variance reduction strategy looking at the most recent documents that share at least one term in common with the new document. Having only a few terms per tweet, there are not many tweets which the variance reduction strategy can use for the comparisons, however, if applied on documents from the TDT5, nearly each document has at least one term in common with a new document and this leads to many similarity computations in the variance reduction strategy even for low k and thus to high processing time. Consequently, looking at the system applied on the TDT5 dataset, we can think of the variance reduction strategy always (independent of k) making the maximum number of similarity comparisons (as determined by the variance reduction strategy parameter b_n) whenever used and thus the running time is inversely proportional to the number of successfully found nearest neighbours through LSH. This explains the inverse running time monitored by Petrovic et al.

Table 7.8: LSH results

L	k	running time in seconds	found through lsh	found through variance	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
10	5	105	107570	61068	0.01408	0.28607	0.02684
10	6	131	128941	95224	0.01622	0.24037	0.03039
10	7	163	145661	141902	0.01804	0.23367	0.03349
10	8	248	161306	195951	0.01753	0.20754	0.03234
10	9	319	175917	249661	0.01617	0.19603	0.02987
10	10	432	186088	302925	0.01582	0.21288	0.02945
10	11	457	193501	342642	0.01515	0.24882	0.02856
10	12	481	198435	353758	0.01549	0.24198	0.02912
10	13	451	202499	349967	0.01551	0.26104	0.02928
10	14	447	211394	341021	0.01528	0.21544	0.02854
10	15	429	219970	332660	0.01486	0.20169	0.02768
10	16	430	210747	341894	0.01582	0.24138	0.02970
10	17	442	193819	358777	0.01572	0.26057	0.02966
10	18	450	176567	376054	0.01566	0.25146	0.02948
40	5	335	149641	28061	0.01650	0.23802	0.03087
40	6	403	191903	43417	0.01607	0.23165	0.03005
40	7	488	239649	60954	0.01819	0.20804	0.03345
40	8	550	282646	87113	0.01742	0.22890	0.03238
40	9	707	314685	124031	0.01631	0.15283	0.02948
40	10	756	337164	162601	0.01567	0.18193	0.02885
40	11	743	347301	195294	0.01572	0.23162	0.02944
40	12	709	345359	206528	0.01559	0.25796	0.02940
40	13	683	337254	215075	0.01467	0.26820	0.02782
40	14	562	337642	214798	0.01531	0.24640	0.02884
40	15	527	342257	210027	0.01559	0.26554	0.02945
40	16	643	328064	224172	0.01509	0.25597	0.02849
40	17	485	300337	252098	0.01574	0.25600	0.02966
40	18	511	274800	277889	0.01507	0.25358	0.02844
70	5	596	155978	24992	0.01652	0.23498	0.03086
70	6	651	205584	33149	0.01618	0.24150	0.03033
70	7	841	258583	46336	0.01799	0.19577	0.03296
70	8	790	316034	57734	0.01701	0.23159	0.03169
70	9	980	361745	80625	0.01622	0.17797	0.02973
70	10	1255	395054	108491	0.01541	0.19344	0.02854
70	11	1111	412163	132678	0.01510	0.22699	0.02832
70	12	992	412925	139734	0.01558	0.24440	0.02929
70	13	786	398015	154628	0.01535	0.20997	0.02861
70	14	670	392442	159533	0.01549	0.24760	0.02915
70	15	602	394499	158262	0.01555	0.24499	0.02925
70	16	569	379298	173613	0.01517	0.25401	0.02862
70	17	557	349335	202939	0.01519	0.24027	0.02858
70	18	577	321706	230685	0.01582	0.25350	0.02979

7.4.5.3 Bucket Size

The bucket size determines the maximum capacity of one bucket in the LSH tables. This limit is introduced in order to achieve bounded space. Petrovic et al. do not provide a suggestion how to set this value. Looking at their TDT5 evaluation, the bucket size is set to 50% of the expected number of collisions, which is computed as $n/2^k$ (n is the overall number of documents to process and k is the number of hyperplanes). Having 221,306 documents in the TDT5 dataset, this leads to a reasonable value of at most 27 tweets per bucket when using 13 hyperplanes. However, we process a dataset of 120 million tweets and applying the same formula leads to a bucket size of 14,648 tweets at most, which is not suitable[3]. Therefore, we empirically tested values for the bucket size. Table 7.9 shows our results. Having a low bucket size clearly deteriorates the system's performance due to LSH only returning few nearest neighbour candidates. Increasing the size improves the overall performance until a specific threshold (in our test we reached this point at a bucket size of 30 tweets at most). Further increasing the bucket size did not improve the performance due to LSH only taking a constant number ($3L$) of tweets which crashed the most frequently in all LSH tables' buckets and thus more tweets in a bucket do not increase the number of possible nearest neighbour candidates a new tweet is compared to in the LSH step. Having this result, we decided to use a bucket size of 30 tweets at most for our evaluation on the full corpus.

Table 7.9: Bucket size results

bucket size	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
2	0.00928	0.34630	0.01807
5	0.01144	0.32705	0.02211
10	0.01324	0.34162	0.02550
15	0.01345	0.29860	0.02573
20	0.01327	0.29400	0.02539
30	0.01564	0.30690	0.02976
50	0.01562	0.27883	0.02958
100	0.01578	0.23825	0.02960
200	0.01573	0.26394	0.02969
500	0.01569	0.30398	0.02983
1000	0.01573	0.27745	0.02978

[3] Assume a tweet takes up 200 bytes of memory. If we use 10 LSH tables, 13 hyperplanes and 14,648 tweets as the buckets' capacity, we have $10 * 2^{13} * 14648 * 200$ bytes $= 240$ gigabytes as upper bound.

7.4.5.4 Similarity Threshold

The similarity threshold t is used to determine if two tweets' similarity to each other is high enough to cluster them together. In detail, first LSH tries to find a nearest neighbour whose similarity to the new tweet reaches or exceeds the similarity threshold t and if this step fails, the variance reduction strategy is used. If both steps fail, a new cluster is created and the new tweet added, being the first tweet in the cluster. Otherwise, the tweet is added to the cluster of the nearest neighbour.

Consequently, the similarity threshold t specifies the granularity of the clusters [POL10]. Using a high threshold t leads to smaller and specific clusters, while a low threshold causes bigger and broader clusters.

Petrovic et al. suggest the value to be $t \in [0.4, 0.5]$. We performed some tests (table 7.10) and found $t = 0.5$ is working best.

Table 7.10: Similarity threshold results

similarity threshold t	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
0.2	0.01629	0.04520	0.02395
0.3	0.01592	0.08835	0.02698
0.4	0.01680	0.17644	0.03067
0.45	0.01743	0.24514	0.03254
0.5	0.01770	0.31823	0.03353
0.55	0.01777	0.28780	0.03347
0.6	0.01572	0.31165	0.02993
0.7	0.01203	0.24858	0.02296
0.8	0.00246	0.12755	0.00483

7.4.5.5 Variance Reduction Strategy

As described in section 5.1.3, the variance reduction strategy is used if LSH fails to find a nearest neighbour which is close enough (similarity threshold) to the new tweet. Therefore, an inverted index is used to compare the new tweet for each of the new tweet's distinct words to at most $b_n/||d||_0$ tweet that contain this word and have not yet been returned by LSH. Here, b_n is the number of maximum comparisons done through the variance reduction strategy and $||d||_0$ is the number of distinct words in the new tweet. If there are more than $b_n/||d||_0$ tweets for a word, the $b_n/||d||_0$ most recent ones are used.

Petrovic et al.'s tests showed that using $b_n \in [1000, 3000]$ performs well, while having reasonable running time. Table 7.11 shows the tests we performed. The result is convincing; the performance is improved substantially when using the variance reduction strategy compared to just using LSH without the variance reduction strategy ($b_n = 0$), however, at the cost of running time. We can confirm Petrovic et al's results and decide to use $b_n = 1000$ for our runs on the full corpus.

Table 7.11: Variance reduction strategy results

variance reduction comparisons b_n	running time in seconds	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
0	123	0.01148	0.28957	0.02208
500	261	0.01526	0.26603	0.02887
1000	337	0.01587	0.25006	0.02985
1500	429	0.01532	0.24997	0.02886
2000	447	0.01615	0.17588	0.02958
3000	527	0.01568	0.24930	0.02951
10000	884	0.01554	0.21944	0.02902

7.4.6 Sankaranarayanan et al.'s System

Next, we choose suitable parameters for Sankaranarayanan et al.'s system, which was introduced in section 5.2. We divide the parameter evaluation into three parts: (i) output (ii) similarity threshold, Gaussian attenuator, time centroid clean-up (iii) fragmentation.

7.4.6.1 Output

First, we investigate in the output generation parameters of Sankaranarayanan et al.'s system. The output generation process and thus the parameters are identical with Petrovic et al.'s system: (i) the output timer that determines the time intervals after which the output is generated (ii) the maximum number of fastest growing and big enough clusters (see third parameter) which are inspected for the output generation (their entropy is checked and if it reaches or exceeds the entropy threshold they are output) (iii) the minimum number of tweets in a cluster in order for being considered a candidate for the output.

Since the output generation works exactly as it does for Petrovic et al.'s system, we just took the setups which performed best for Petrovic et al.'s system and did some

additional tests to make sure these settings also work for this system. Table 7.12 shows the tests' results. The two best working setups are marked green.

Table 7.12: Output generation results

output timer	number of fastest growing & big enough clusters inspected	minimum tweets in cluster	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
100,000	50	25	0.01167	0.26983	0.02237
200,000	50	25	0.01189	0.25617	0.02273
100,000	50	50	0.01191	0.26427	0.02280
200,000	50	50	0.01198	0.25806	0.02289
100,000	100	50	0.01191	0.26427	0.02280
200,000	100	50	0.01197	0.26620	0.02291
100,000	200	50	0.01191	0.26427	0.02280
200,000	200	50	0.01197	0.26620	0.02291
100,000	200	200	0.00980	0.24102	0.01884
200,000	200	200	0.00982	0.24102	0.01888

7.4.6.2 Similarity Threshold, Gaussian Attenuator and Time Centroid Clean-up

In this section we determine suitable values for the similarity threshold, the Gaussian attenuator parameter (σ) and the time centroid clean-up threshold.

The similarity threshold t is used to determine if a new tweet's similarity to a cluster is large enough in order to add the tweet to the cluster.

The Gaussian attenuator is an additional coefficient in the similarity computation between a new tweet and a cluster (formula 5.8) and increases the overall similarity between tweets and clusters which are close in time to each other. The attenuator is parameterized by σ (formula 5.8) which adjusts the speed the clusters expire; a small sigma leads to clusters expiring quickly, while a large sigma keeps them active longer. For example, if we choose $\sigma = 0.5$ and the time difference between the cluster centroid and the new tweet is a half day, the Gaussian attenuator is computed as $e^{\frac{-(T_t - T_c)^2}{2(\sigma)^2}} = e^{\frac{-0.25}{0.5}} = 0.6065$ and thus the overall similarity is already lowered by a big factor. If $\sigma = 1.0$ and the time difference is a half day, the Gaussian attenuator is $e^{\frac{-0.25}{2}} = 0.8825$ which leads to a higher overall similarity.

Since the overall similarity between a tweet and a cluster is the composition of the cosine similarity and the Gaussian attenuator and thus dependent on the Gaussian attenuator, we decided to set the similarity threshold to a fixed value $t = 0.45$ (performed

well in earlier tests) and try to tune the Gaussian attenuator in such a way that the system performs well.

Furthermore, we have to take the time centroid clean-up into consideration; the time centroid of a cluster is the mean publication time of all tweets belonging to the cluster. Sankaranarayanan et al. use this time centroid for removing clusters which are too old; if their time centroid is behind in time for some threshold (time centroid clean-up threshold), the cluster is removed. Looking at the Gaussian attenuator, if we choose a small σ and thus let the clusters expire fast (e.g., after one hour), it makes no sense to keep these clusters in the system for a day. On the other hand, if we decide to remove clusters if the difference to their time centroid is greater than two hours, setting sigma to a high value has nearly no effect on the similarity computations (e.g., $\sigma = 2.0$ and a time centroid difference threshold of six hour leads to the Gaussian attenuator $e^{\frac{-0.0625}{8}} = 0.9922$). Consequently, we test σ in combination with the time centroid clean-up threshold and fix the similarity threshold to $t = 0.45$. The results can be seen in table 7.13. The four setups which performed best are marked green.

Table 7.13: σ and time centroid clean-up threshold results

σ	time centroid clean-up threshold in hours	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
0.01	1	0.00724	0.23090	0.01404
0.02	1	0.00958	0.23814	0.01842
0.05	1	0.01204	0.27632	0.02307
0.10	1	0.01199	0.28321	0.02301
0.50	1	0.01201	0.30108	0.02310
1.00	1	0.01201	0.30104	0.02310
0.01	2	0.00964	0.21650	0.01846
0.02	2	0.00956	0.23814	0.01838
0.05	2	0.01198	0.27632	0.02296
0.10	2	0.01183	0.28117	0.02271
0.50	2	0.01176	0.27738	0.02257
1.00	2	0.01175	0.27727	0.02255
0.01	4	0.00964	0.21650	0.01846
0.02	4	0.00955	0.23814	0.01836
0.05	4	0.01195	0.27632	0.02291
0.10	4	0.01174	0.28117	0.02255
0.50	4	0.01383	0.28640	0.02638
1.00	4	0.01383	0.28716	0.02639
0.01	8	0.00963	0.21650	0.01845
0.02	8	0.00957	0.26654	0.01849
0.05	8	0.00950	0.28806	0.01840
0.10	8	0.00928	0.28574	0.01797
0.50	8	0.01106	0.28383	0.02129
1.00	8	0.01101	0.28469	0.02120
0.01	24	0.00970	0.23093	0.01862
0.02	24	0.00957	0.26654	0.01849
0.05	24	0.00950	0.28806	0.01840
0.10	24	0.00927	0.28574	0.01796
0.50	24	0.01104	0.28383	0.02125
1.00	24	0.01099	0.28461	0.02117

7.4.6.3 Fragmentation

In section 5.2.4 we introduced the defragmentation step performed by Sankaranarayanan et al. in order to lower fragmentation errors. Therefore, they periodically iterate over all active clusters and check for duplicate ones. While this step may improve the overall results, as Sankaranarayanan et al. find, it is not necessary for our evaluations. The final merging step which we apply (section 7.2) eliminates fragmentation in a similar way. Table 7.14 shows the test's result; using defragmentation did not improve the results compared to not using it (fragmentation clean-up timer set to 0). Consequently, we will not use this step in our full corpus run.

Table 7.14: Fragmentation results

fragmentation clean-up timer	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
0	0.01201	0.30104	0.02310
50,000	0.01203	0.30104	0.02313
100,000	0.01203	0.30104	0.02313
200,000	0.01202	0.30104	0.02312
500,000	0.01201	0.30104	0.02310

7.4.7 Aggarwal and Subbian's System

In this section we determine suitable parameters for Aggarwal and Subbian's system [AS12], which was introduced in section 5.3. The parameters are divided into the system's output parameters and the number of clusters used. Furthermore, as described in section 5.3.1, Aggarwal and Subbian utilize the user-follower network in order to calculate a structural similarity between a tweet and a cluster (formula 5.9) and thus extend the common text-only similarity. This approach is also evaluated in this chapter.

7.4.7.1 Output

Aggarwal and Subbian's output generation (section 5.3.5) differs from the output generation used for Petrovic et al. and Sankaranarayanan et al. The system continuously tracks for so called evolution events using a time horizon H and a threshold α (formula 5.13). Having the current time t_c and the cluster's creation time $t(C_i)$, if the ratio of tweets added in time interval $[t_c - H, t_c]$ to the number of tweets added in time interval

$[t(C_i), t_c - H]$ reaches or exceeds α, the cluster is output as an event happening. Furthermore, the cluster has to exist for at least $2H$ in order to be considered a candidate for the output. H and α depend on the data stream to process; we determine suitable values empirically (Appendix, E-Mail Conversation with Karthik Subbian). Note that we use the number of documents processed for measuring time; this means that a point in time (e.g., t_c) is represented through the document counter at this point in time and a time interval (e.g. H) is represented through a number of documents processed. Furthermore, we test the performance for different values of tweets a cluster needs to have as a minimum to be output. We introduced this constraint due to bad performance monitored when not having such a limitation. Consider $H = 1000$, $\alpha = 1$ and no constraint for the minimum tweets in a cluster. Since the cluster has to exist for at least $2H$, we would output the cluster if one tweet was added within the first thousand documents and one more tweet within the second thousand documents; this is not desirable. Consider $H = 1000$, $\alpha = 0.5$ and the minimum number of tweets in a cluster is set to 25; here, an example of an output candidate is a cluster where 16 tweets are added within the first thousand documents and further 9 tweets within the second thousand documents ($\frac{9}{16} = 0.5625 \geq 0.5$, 25 tweets in cluster). We think this constraint is suitable, which can be seen in table 7.15 (tests without the constraint performed worse). Note that this constraint is not necessary if H is set high; if a cluster still exists after $2H$, tweets have been added frequently. Otherwise, the cluster would have been removed already (as described in section 5.3.3). Nevertheless, our results for high H are poor.

Table 7.15: Output generation results

H	α	minimum tweets in cluster	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
500	0.5	25	0.01426	0.30001	0.02722
1000	0.5	25	0.01634	0.33372	0.03116
2500	0.5	25	0.01583	0.32581	0.03019
5000	0.5	25	0.01357	0.26887	0.02584
10000	0.5	25	0.00930	0.24877	0.01793
500	1.0	25	0.01194	0.26813	0.02286
1000	1.0	25	0.01422	0.29293	0.02713
2500	1.0	25	0.01192	0.34762	0.02304
5000	1.0	25	0.00929	0.28377	0.01799
10000	1.0	25	0.00696	0.23400	0.01352
500	1.5	25	0.01206	0.27167	0.02310
1000	1.5	25	0.01188	0.28698	0.02282
2500	1.5	25	0.00965	0.33741	0.01877
5000	1.5	25	0.00481	0.31358	0.00947
10000	1.5	25	0.00477	0.23544	0.00935
500	0.5	50	0.01186	0.27109	0.02273
1000	0.5	50	0.01427	0.30671	0.02728
2500	0.5	50	0.01429	0.34393	0.02744
5000	0.5	50	0.01407	0.27027	0.02675
10000	0.5	50	0.00937	0.24877	0.01807
500	1.0	50	0.00727	0.36389	0.01426
1000	1.0	50	0.01190	0.31843	0.02294
2500	1.0	50	0.01211	0.33124	0.02337
5000	1.0	50	0.00961	0.28199	0.01858
10000	1.0	50	0.00702	0.23391	0.01363
500	1.5	50	0.00245	0.17391	0.00483
1000	1.5	50	0.00956	0.28732	0.01850
2500	1.5	50	0.00723	0.33581	0.01415
5000	1.5	50	0.00489	0.31935	0.00962
10000	1.5	50	0.00480	0.23420	0.00941
1000	0.5	0	0.00917	0.32113	0.01784
1000	1.0	0	0.00922	0.33040	0.01793
1000	1.5	0	0.00933	0.30932	0.01811
25000	0.5	0	0.00954	0.17936	0.01811
25000	1.0	0	0.00472	0.13051	0.00912
25000	1.5	0	0.00245	0.08036	0.00475
100000	0.5	0	0.00244	0.17391	0.00482
100000	1.0	0	0.00245	0.17391	0.00483
100000	1.5	0	0.00000	0.00000	0.00000

7.4.7.2 Clusters

Aggarwal and Subbian's system uses a predefined number of clusters. In this subsection we determine the number of clusters to use for our evaluation on the full corpus. Table 7.16 shows the tests' results, which are similar to the results of Aggarwal and Subbian. The overall system's performance increases until 750 clusters and stays on the same level for more clusters. We think less clusters lead to clusters being removed even they are relevant due to the system's cluster removal process. On a new tweet arriving and not reaching or exceeding the similarity threshold to any of the existing clusters, the system removes the cluster that was not updated for the longest time. Looking at Twitter, there may unfold many hundreds of tweets between two tweets belonging to the same topic and thus the cluster which contains the first of the two related tweets may be removed already as the second tweet arrives. Therefore, the system is only able to catch such tweets if the number of clusters is large enough.

Table 7.16: Clusters results

number of clusters	running time in seconds	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
250	50	0.01418	0.30147	0.02709
500	92	0.01399	0.33737	0.02686
750	138	0.01628	0.33236	0.03103
1000	220	0.01634	0.33372	0.03116
1250	363	0.01613	0.32814	0.03076
1500	589	0.01631	0.34416	0.03115
1750	805	0.01638	0.33233	0.03122
2000	992	0.01631	0.33057	0.03109

7.4.7.3 Structural Similarity

As proposed in section 5.3.1, Aggarwal and Subbian extend the typical text-based similarity computation between a tweet and a cluster by a structural similarity (formula 5.9) by looking at the underlying network structure; this network consists of the senders (tweets' authors) as nodes and the sent messages (tweets) as edges from sender to receiver (the sender's followers). Consequently, each tweet contains a set of nodes consisting of the tweet's author as well as the author's followers. A cluster contains a set of nodes which is the union over all nodes of each tweet in the cluster as well as the nodes' frequency (i.e., whenever a node occurs again its count is incremented). When computing the structural similarity between a tweet and a cluster, the relationship of identical nodes in the tweet and in the cluster to the overall number of nodes in the tweet and in the cluster is taken into account (formula 5.10).

Finally, the overall similarity between a tweet and a cluster is computed as composition of the structural similarity and the content similarity where λ is used as balancing parameter (5.9); $\lambda = 0.0$ means we are only using the content similarity while $\lambda = 1.0$ only takes the structural similarity into account. $\lambda = 0.5$ gives both similarities equal weighting.

As an example, consider tweet A and tweet B, both being labelled relevant for the same event in the McMinn et al. corpus. If tweet A's author and tweet B's author have followers in common and tweet A's similarity to a cluster containing tweet B is computed, the overall similarity is increased compared to not having followers in common.

As described in section 7.1, due to Twitter's current limitations, we were just able to crawl a small number of nodes; we crawled the follower IDs of all users who wrote tweets that are labelled as being relevant within the first two million tweets in the McMinn et al corpus. Further, we were only able to obtain at most 5000 latest follower IDs per user. This concluded in a set of 2043 unique users and 1,971,245 follower IDs.

Tabel 7.17 shows our tests' results. As we found in Aggarwal and Subbian's paper, we used a depth $d = 2$, each having width $w = 262,213$ for the count-min sketch (section 5.3.2). Having $\lambda = 0.5$ we can see a slight improvement over not using the structural similarity. $\lambda = 0.25$ and $\lambda = 0.75$ deteriorates the performance. Using $\lambda = 1.0$, we are not able to detect any event; only having follower IDs for 2043 users, the majority of tweets do not cluster (due to a structural similarity ≈ 0) and thus clusters grow too slow or do not contain enough tweets for being output. Even obtaining a slight improvement in performance for $\lambda = 0.5$, we do not have enough evidence to give a clear assessment of this similarity computation technique tested on our data. More detailed tests having a bigger set of users and their follower IDs are necessary.

Table 7.17: Structural similarity results

λ	$F_{2_{events}}$	$F_{2_{tweets}}$	F_{final}
0.00	0.01631	0.33057	0.03109
0.25	0.01622	0.26601	0.03057
0.50	0.01643	0.31919	0.03121
0.75	0.01181	0.39541	0.02294
1.00	0.00000	0.00000	0.00000

7.5 Full Corpus Run

After having tested different setups for preprocessing, retweets, entropy, merging the output as well as the different system specific parameters, we are able to choose suitable setups for the systems' evaluation on the full McMinn et al. corpus.

For all three systems, we use the same setup for preprocessing, entropy and merging the output. In detail, for each tweet, we remove mentions, retweet markup, URLs and signs that are no characters and digits (section 7.4.1). Furthermore, we remove stopwords (section 7.4.1). A cluster is only output if its entropy is equal or greater than 3.5 (section 7.4.3). For the output merging we proceed as follows: we merge clusters together using a similarity threshold of 0.8 (section 7.2, section 7.4.4). In order to label an output cluster an event fraction, the event fraction threshold of 0.05 has to be reached (section 7.2, section 7.4.4). An event is successfully detected if the event threshold of 0.10 is reached (section 7.2, section 7.4.4).

Subsequently, the system specific setups we used are listed and the corresponding results of the evaluation on the full corpus pointed out. The setups and results are shown in table 7.18 and table 7.19 (Petrovic et al.), table 7.20 and table 7.21 (Sankaranarayanan et al.) as well as table 7.22 and table 7.23 (Aggarwal and Subbian).

Table 7.18: Setups tested for Petrovic et al.'s system

	LSH tables L	hyperplanes k	similarity threshold t	variance reduction comparisons b_n	bucket size	output timer	number of fastest growing & big enough clusters inspected	minimum tweets in cluster
p_1	10	7	0.5	1000	30	100,000	200	50
p_2	10	13	0.5	1000	30	100,000	200	50
p_3	70	13	0.5	1000	30	100,000	200	50
p_4	10	13	0.6	1000	30	100,000	50	50

Table 7.19: Results of Petrovic et al.'s system

	output clusters before merging	output clusters after merging	relevant event fractions	detected relevant events	running time in seconds	precision over events	recall over events	$F_{2_{events}}$	precision over tweets	recall over tweets	$F_{2_{tweets}}$	F_{final}
p_1	25863	11841	236	150	38646	0.01993	0.29703	0.07857	0.05133	0.45818	0.17724	0.10887
p_2	86337	35805	350	201	49512	0.00978	0.39802	0.04450	0.04380	0.45600	0.15822	0.06947
p_3	86819	33169	380	204	82148	0.01146	0.40396	0.05145	0.03665	0.46227	0.13914	0.07512
p_4	25141	7611	239	156	47076	0.03140	0.30891	0.11162	0.07289	0.44637	0.22047	0.14821

Table 7.20: Setups tested for Sankaranarayanan et al.'s system

	Gaussian attenuator parameter σ	time centroid clean-up threshold in hours	similarity threshold t	fragmentation clean-up timer	output timer	number of fastest growing & big enough clusters inspected	minimum tweets in cluster	retweets
s_1	1.0	1	0.45	0	200,000	200	50	-
s_2	1.0	4	0.45	0	200,000	200	50	-
s_3	1.0	8	0.45	0	200,000	200	50	-
s_4	1.0	4	0.45	0	200,000	200	50	x

Table 7.21: Results of Sankaranarayanan et al.'s system

	output clusters before merging	output clusters after merging	relevant event fractions	detected relevant events	running time in seconds	precision over events	recall over events	$F_{2_{events}}$	precision over tweets	recall over tweets	$F_{2_{tweets}}$	F_{final}
s_1	3453	1193	67	57	3746	0.05616	0.11287	0.09391	0.08670	0.40784	0.23428	0.13407
s_2	14284	3645	175	131	11121	0.04801	0.25941	0.13794	0.07940	0.3810628	0.21654	0.16852
s_3	18967	5208	210	152	18164	0.04032	0.30099	0.13127	0.09385	0.39841	0.24160	0.17011
s_4	25312	9021	232	153	19412	0.02572	0.30297	0.09599	0.05187	0.40804	0.17192	0.12320

Table 7.22: Setups tested for Aggarwal and Subbian's system

	number of clusters	similarity balancing parameter λ	time horizon H	growth ratio threshold α	minimum tweets in cluster
a_1	1000	0.0	1000	0.5	25
a_2	1750	0.0	1000	0.5	25
a_3	2000	0.0	1000	0.5	25

Table 7.23: Results of Aggarwal and Subbian's system

	output clusters before merging	output clusters after merging	relevant event fractions	detected relevant events	running time in seconds	precision over events	recall over events	$F_{2_{events}}$	precision over tweets	recall over tweets	$F_{2_{tweets}}$	F_{final}
a_1	5874	2818	174	106	11507	0.06175	0.20990	0.14184	0.08399	0.42916	0.23555	0.17706
a_2	3510	2451	141	92	22811	0.05753	0.18218	0.12710	0.13231	0.48018	0.31470	0.18107
a_3	3279	2363	130	84	27310	0.05501	0.16634	0.11841	0.11464	0.54084	0.31020	0.17140

7.6 Discussion of the Results

In this section, we discuss the results (see last section) of the systems' evaluation on the full McMinn et al. corpus [MMJ13].

First, we recall the different attributes which can be seen in the result tables. Each system outputs clusters corresponding to its output generation technique (e.g., outputting the fastest growing and big enough clusters each 100,000 documents). Therefore, in many cases, an output cluster is just a fraction of an actual event and may be output again in a later output interval. Therefore, we applied a final merging step which merges clusters that are similar (section 7.2). The attributes "output clusters before merging" and "output clusters after merging" catch the effect of this step. Subsequently, we check all clusters for "relevant event fractions" (section 7.2). We label a set of such event fractions a successfully detected event (attribute "detected relevant events") if the set reaches a predefined ratio of tweets labelled as being relevant for the corresponding event (section 7.2). The "running time in seconds" is straightforward. The "precision over events" is computed as fraction of the relevant event fractions and the output clusters after merging (i.e., the final output of the systems). Further, "recall over events" is calculated as fraction of relevant events and the number of labelled events in the corpus (i.e., 505 events - we removed one event because this event has exactly one tweet labelled as being relevant in McMinn et al.'s labellings and we were not able to crawl this tweet and thus it cannot be detected). When computing "precision over tweets" and "recall over tweets", we only look at the relevant events we actually detected. Then, the "precision over tweets" is the fraction of the overall number of detected labelled tweets for the corresponding events to the overall number of tweets detected for the corresponding events. The "recall over tweets" is computed as fraction of the overall number of detected labelled tweets for the corresponding events to the overall number of labelled tweets for the corresponding events. $F_{2_{tweets}}$, $F_{2_{events}}$ and F_{final} are computed as described in section 7.3.

Petrovic et al.'s system was tested by performing four full runs (tables 7.18 and 7.19). For the first three runs, we tested different combinations of the number of LSH tables L and hyperplanes k. Using more hyperplanes clearly led to more clustering (and thus more clusters in the output) due to a bigger scope of tweets hold. However, the number of detected relevant events only increased slightly; having a small number of tweets hold in the system (when using 7 hyperplanes) still produced many events detected. We think this is explained through the timely connected character of an event and thus it is not necessary to keep tweets in the system for a long time. Looking at the precision over events, we see the reason why the first setup performed best in

the overall assessment F_{final}; using $k = 13$ hyperplanes led to many clusters returned which are not labelled as event and thus produced bad precision over events values, however, as described earlier, this does not necessarily mean that the returned clusters are not events. Therefore, looking at the third setup p_3, even having the worst overall score F_{final}, the highest number of relevant events was detected as well as the highest recall over tweets. Consequently, we decided to perform a fourth run p_4 but limited the number of fastest growing and big enough clusters to a lower value (50) in order to output less clusters and we also raised the similarity threshold t to 0.6 in order to produce more specific and thin clusters. As expected, this concluded in a much better overall performance, being the best run for this system. We lost a few relevant events in the output due to the reduced number of clusters output but precision over events as well as precision over tweets increased significantly. We think the precision over events is mainly affected by less clusters in the output while the increase in precision over tweets is a consquence of the more specific clusters produced by the higher similarity threshold t. Nevertheless, still having many clusters in the output, we think the results could further be improved by tuning the parameters in such a way that less and more specific clusters are output. The runtime behaves as expected and as it was described in the parameter evaluation (section 7.4.5.2).

We ran four full runs using Sankaranarayanan et al.'s system (tables 7.20 and 7.21). We used different time centroid clean-up threshold values for the first three setups (1, 4 and 8 hours). As the results show, keeping clusters in the system for a longer time leads to more tweets clustered and thus more clusters output. However, precision over events deteriorates, which means that the ratio of non-event clusters to event clusters output grows, however, more events are detected and the recall over events increases. Looking at precision and recall over tweets, the results stay on the same level; consequently, the ratio of labelled tweets detected is not dependent on the length of the clusters being hold in the system. If an event cluster is removed faster due to a low time centroid clean-up threshold, the event forms a new cluster which is successfully merged with the other clusters discussing the event later on (otherwise recall over tweets would differ substantially). The fourth setup uses retweets. We decided to test the system with retweets because our pre-evaluation indicated that this system performs better using retweets (section 7.4.2), however, tested on the full corpus, F_{final} decreased significantly. In fact, the system was able to detect 153 labelled events (1 more than the best setup without retweets) and thus retweet add to the system's ability of detecting as many events as possible (leading to a high recall over events). Nevertheless, precision both over events and tweets are low compared to not using retweets; the low precision over events indicates that retweets lead to many clusters formed and output that are not events or even spam, which is the reason McMinn et al. decided against using retweets [MMJ13]. The low precision over tweets shows the expected be-

haviour if an event is detected successfully; looking at a successfully detected event, the number of tweets detected tends to be much higher (due to retweets), however, retweets are not included in the labellings of McMinn et al. Another observation we made is the increase in running time, however, it behaves as expected; keeping clusters in the system for a longer time leads to more clusters being used for the similarity computations and thus to a higher running time.

Aggarwal and Subbian's system was tested by performing three runs on the full corpus (tables 7.22 and 7.23). Hereby, we tested three different cluster setups (1000, 1750 and 2000). Because using 1750 clusters performed better than 1000 clusters, we decided to also test 2000 clusters in order to find out if more clusters would further improve the results, however, it did not. We observe that increasing the number of clusters lowers the number of clusters in the output and thus the setup a_1 using 1000 clusters outputs the most clusters. Furthermore, a_1 performed best in the number of detected relevant events, precision over events and recall over tweets. We think this happens because less clusters lead to bigger and broader output clusters (since less candidates exist for a new tweet) and thus the chance for such a cluster to be output is higher. The low value of precision over tweets also confirms this theory; there are many unlabelled tweets in the output. Having more clusters provides a bigger scope for new tweets and thus leads to more specific and thinner clusters. Consequently, less clusters are in the output but a higher precision over tweets is monitored. The running time increases proportional to the number of clusteres used, which we think is straightforward. In conclusion, a_2 achieved the highest overall score, however, detected fewer labelled events than a_1.

Giving a final comparison of the three systems, Petrovic et al.'s system was able to detect by far the most of the labelled events (40%) but suffers from too many output clusters and thus low precision over events values. Furthermore, the system has low precision over tweets values which we think is due to the more aggressive clustering approach (tweet to tweet clustering) which leads to clusters growing less controlled. Sankaranarayanan et al.'s system performed more balanced. It detected not as many events as Petrovic et al.'s system, but it produced fewer output clusters which contained less unlabelled tweets. On the other hand, Aggarwal and Subbian's system detected the least labelled events but performed very good in precision over tweets and recall over tweets; we think this is due to their output generation process which is capable of catching sudden rises in tweets written about an event when it happens. In conclusion, Aggarwal and Subbian's system performed best in our evaluation corresponding to the measures we introduced, however, we think Petrovic et al.'s system has the most potential for improvement.

8 Conclusion

Concluding this master thesis, we performed and in-depth evaluation of three state-of-the-art event detection systems for Twitter. To this end, we first provided necessary background information. Therefore, we started with an easy and more general definition of an event which better fits the characteristics of event-based detection in Twitter. We introduced event detection in general and provided a classification that helped to choose possible event detection systems for our evaluation. Subsequently, we investigated in Twitter as our data source of choice and exposed pros and cons in using Twitters real-time stream. An overview of related work was given and three promising event detection systems for our evaluation were chosen. Subsequently, these three systems were introduced in detail. In order to find a suitable corpus for the evaluation, we examined existing Twitter corpora and chose the biggest existing annotated Twitter corpus containing 120 million tweets as well as 150 thousand labelled tweets being on-topic for 506 events. Furthermore, we revealed till now common methodologies in evaluating event detection systems in Twitter and gave a summary of other evaluations executed. The corpus was crawled using a publicly available web crawler. Looking at our evaluation, we explained necessary postprocessing steps and introduced the evaluation measures applied. We conducted an in-depth parameter evaluation; being time-consuming, we were only able to perform a few runs on the full corpus and thus we evaluated suitable parameters on a smaller part of the data. We have seen that the selection of suitable parameters is crucial. Finally, we ran the evaluation for the chosen setups on the full McMinn et al. corpus computing typical evaluation measures which led to an objective assessment. Looking at the results, we can conclude that Twitter can be used as a valuable resource for detecting events. However, the recall values we obtained are far from being perfect; the best run only detected 40% of the labelled events in the corpus which clearly demonstrates the possibility of improvement. To the best of our knowledge, this is the first comprehensive evaluation and comparison of event detection systems in Twitter performed on a large-scale corpus.

9 Bibliography

[ACD+98] James Allan, Jaime G Carbonell, George Doddington, Jonathan Yamron, and Yiming Yang. Topic detection and tracking pilot study final report. 1998.

[AK13] Farzindar Atefeh and Wael Khreich. A survey of techniques for event detection in twitter. *Computational Intelligence*, 2013.

[ALMS00] James Allan, Victor Lavrenko, Daniella Malin, and Russell Swan. Detections, bounds, and timelines: Umass and tdt-3. In *Proceedings of Topic Detection and Tracking Workshop (TDT-3)*, pages 167–174. Vienna, VA, 2000.

[Ana09] Pear Analytics. Twitter study–august 2009. *San Antonio, TX: Pear Analytics. Available at: www. pearanalytics. com/blog/wp-content/uploads/2010/05/Twitter-Study-August-2009. pdf*, 2009.

[AS12] Charu C Aggarwal and Karthik Subbian. Event detection in social streams. In *SDM*, volume 12, pages 624–635. SIAM, 2012.

[AZ12] Charu C Aggarwal and ChengXiang Zhai. A survey of text clustering algorithms. In *Mining text data*, pages 77–128. Springer, 2012.

[Bai14] Jonathan Bailey. Twitter, plagiarism and retweeting. https://www.plagiarismtoday.com/2014/07/17/twitter-plagiarism-retweeting/ (accessed on August 18, 2014), 2014.

[BHB11] Edward Benson, Aria Haghighi, and Regina Barzilay. Event discovery in social media feeds. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 389–398. Association for Computational Linguistics, 2011.

[BK09] Albert Bifet and Richard Kirkby. Data stream mining a practical approach. 2009.

[BNG11] Hila Becker, Mor Naaman, and Luis Gravano. Beyond trending topics: Real-world event identification on twitter. *ICWSM*, 11:438–441, 2011.

[Cha02] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

[CM05] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[Cor12] Mário Cordeiro. Twitter event detection: Combining wavelet analysis and topic inference summarization. In *Doctoral Symposium on Informatics Engineering, DSIE*, 2012.

[CW77] J Lawrence Carter and Mark N Wegman. Universal classes of hash functions. In *Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112. ACM, 1977.

[DS10] Ljiljana Dolamic and Jacques Savoy. When stopword lists make the difference. *Journal of the American Society for Information Science and Technology*, 61(1):200–203, 2010.

[FYYL05] Gabriel Pui Cheong Fung, Jeffrey Xu Yu, Philip S Yu, and Hongjun Lu. Parameter free bursty events detection in text streams. In *Proceedings of the 31st international conference on Very large data bases*, pages 181–192. VLDB Endowment, 2005.

[HLW+12] Mengdie Hu, Shixia Liu, Furu Wei, Yingcai Wu, John Stasko, and Kwan-Liu Ma. Breaking news on twitter. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*, pages 2751–2754. ACM, 2012.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[JSFT07] Akshay Java, Xiaodan Song, Tim Finin, and Belle Tseng. Why we twitter: understanding microblogging usage and communities. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 56–65. ACM, 2007.

[JZSC09] Bernard J Jansen, Mimi Zhang, Kate Sobel, and Abdur Chowdury. Twitter power: Tweets as electronic word of mouth. *Journal of the American society for information science and technology*, 60(11):2169–2188, 2009.

[KA04] Giridhar Kumaran and James Allan. Text classification and named entities for new event detection. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 297–304. ACM, 2004.

[KGA08] Balachander Krishnamurthy, Phillipa Gill, and Martin Arlitt. A few chirps about twitter. In *Proceedings of the first workshop on Online social networks*, pages 19–24. ACM, 2008.

[KKS00] Michael Steinbach George Karypis, Vipin Kumar, and Michael Steinbach. A comparison of document clustering techniques. In *TextMining Workshop at KDD2000 (May 2000)*, 2000.

[Kle02] Jon Kleinberg. Bursty and hierarchical structure in streams. 2002.

[KLPM10] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 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, 2010.

[LS10] Ryong Lee and Kazutoshi Sumiya. Measuring geographical regularities of crowd behaviors for twitter-based geo-social event detection. In *Proceedings of the 2nd ACM SIGSPATIAL international workshop on location based social networks*, pages 1–10. ACM, 2010.

[LTY07] Gang Luo, Chunqiang Tang, and Philip S Yu. Resource-adaptive real-time new event detection. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 497–508. ACM, 2007.

[LWC+11] Rui Long, Haofen Wang, Yuqiang Chen, Ou Jin, and Yong Yu. Towards effective event detection, tracking and summarization on microblog data. In *Web-Age Information Management*, pages 652–663. Springer, 2011.

[MCH12] Donald Metzler, Congxing Cai, and Eduard Hovy. Structured event retrieval over microblog archives. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 646–655. Association for Computational Linguistics, 2012.

[MMJ13] Andrew J. McMinn, Yashar Moshfeghi, and Joemon M. Jose. Building a large-scale corpus for evaluating event detection on twitter. pages 409–418. ACM, 2013.

[OAR11] Onook Oh, Manish Agrawal, and H Raghav Rao. Information control and terrorism: Tracking the mumbai terrorist attack through twitter. *Information Systems Frontiers*, 13(1):33–43, 2011.

[Pet12] Sasa Petrovic. Real-time event detection in massive streams. 2012.

[PM10] Swit Phuvipadawat and Tsuyoshi Murata. Breaking news detection and tracking in twitter. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*, volume 3, pages 120–123. IEEE, 2010.

[PM11] Chi-Chun Pan and Prasenjit Mitra. Event detection with spatial latent dirichlet allocation. In *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*, pages 349–358. ACM, 2011.

[POL10] Saša Petrović, Miles Osborne, and Victor Lavrenko. Streaming first story detection with application to twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 181–189. Association for Computational Linguistics, 2010.

[POL12] Saša Petrović, Miles Osborne, and Victor Lavrenko. Using paraphrases for improving first story detection in news and twitter. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 338–346. Association for Computational Linguistics, 2012.

[PP10] Ana-Maria Popescu and Marco Pennacchiotti. Detecting controversial events from twitter. In *Proceedings of the 19th ACM international confer-*

ence on Information and knowledge management, pages 1873–1876. ACM, 2010.

[Reh12] Daniel Rehn. Twitter: 500 million tweets per day. http://www.socialmediastatistik.de/twitter-500-millionen-tweets/ (accessed on August 15, 2014), 2012.

[Sch09] Erick Schonfeld. Mining the thought stream. techcrunch weblog article, 2009.

[SOM10] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. 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, 2010.

[SST+09] Jagan Sankaranarayanan, Hanan Samet, Benjamin E Teitler, Michael D Lieberman, and Jon Sperling. Twitterstand: news in tweets. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 42–51. ACM, 2009.

[vR79] C.J. van Rijsbergen. *Information retrieval. 2nd ed.* 1979.

[WG08] Xiaogang Wang and Eric Grimson. Spatial latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 1577–1584, 2008.

[WGB12] Xiaofeng Wang, Matthew S Gerber, and Donald E Brown. Automatic crime prediction using events extracted from twitter posts. In *Social Computing, Behavioral-Cultural Modeling and Prediction*, pages 231–238. Springer, 2012.

[WYLL11] Jianshu Weng, Yuxia Yao, Erwin Leonardi, and Francis Lee. Event detection in twitter. 2011.

[YPC98] Yiming Yang, Tom Pierce, and Jaime Carbonell. 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, 1998.

[ZR09] Dejin Zhao and Mary Beth Rosson. How and why people twitter: the role

that micro-blogging plays in informal communication at work. In *Proceedings of the ACM 2009 international conference on Supporting group work*, pages 243–252. ACM, 2009.

Appendix

The appendix contains E-Mail conversations with different researchers. The vertical lines mark our questions. The paragraphs which are not marked are the corresponding answers.

E-Mail Conversation with Sasa Petrovic

> Dear Mr. Petrovic,
>
> My name is Gerhard Grimm and I am conducting in an evaluation of event detection systems in Twitter at University of Würzburg, Germany. I am also reimplementing your system for event detection as proposed in your paper “Streaming First Story Detection with application to Twitter”. Great work! Anyway, there are some unanswered questions by my side. I would really appreciate your help here.
>
> 1) Your system wants to achieve bound memory usage as well as bound processing time per tweet. Anyway, you are not writing about cleaning up the threads itself and since each tweet is hold by the threads the number of tweets hold by the system would grow without a bound. When and how do you clean up threads?

The threads themselves have a limited time to live (normally one hour, but it’s a parameter). After the thread is created, one hour after the first tweet in the thread we have to make a decision – either output the thread as news, or don’t. In both cases, the thread is deleted. However, note that the threads themselves don’t really take up almost any memory. It’s the tweets that take up memory, and they are deleted according to a different schedule and can outlive threads by many days. For details on how I delete tweets, see my PhD thesis, in particular section 4.2.3. "Constant space approach".

> 2) The same question applies to the words. Do you clean up your wordlist (words to idf values...) of words that havent been seen for a long time and how frequently?

There's no real need to do this. The vocabulary size grows very slowly (note that I discard URLs and usernames), so there's no need to do this in practice. What I've done in the past (when I wanted to use all features, i.e., including URLs and usernames) is just hash everything into a big enough but constant space, say 4 GB. This way there's a very small chance of two words actually having the same hash code, you're guaranteed constant space, and the loss in performance is negligible.

> 3) Furthermore, if you are cleaning up your wordlists periodically, are you removing these words from the tweets hold by the system? If you do and recalculate such a tweet's tfidf representation and vector length upon a comparison, do you adjust the threshold, since the similarities tend to be much higher upon a tweet having unimportant words removed and therefore a shorter vector, while maintaining the same numerator in the cosine similarity computations.

Like I said, I don't clean up words. I do however clean up tweets, which is much simpler – you just remove the tweet from the index. Note that I don't update the IDF weights to reflect the deleted weights. This makes sense, because IDF should reflect how common is a term, and this should never decrease or depend on the internal implementation of the index.

> 4) The last question relates to the entropy which you use for reducing spam in the output. Do you apply this procedure periodically while the system is running, or only after the system is queried for output?

I apply this on a thread level, so in a sense, this is happening all the time. Whenever I create a new thread, I note the timestamp of the tweet that is the first. Then, one hour (or whatever period you settle on) after the first tweet in the thread I compute the entropy of the thread and decide whether to output it or delete it.

> Thanks a lot for your answer! It helped a lot.
>
> There is one more question regarding your answer:
>
> If you destroy a thread after you have looked at it after some time period (no matter if it is output as event, marked as spam or just little growth rate), how are you handling the tweets that belonged to the thread? If you keep them as possible nearest neighbour

> candidates in your system, wouldn't this lead to a lot of fragmentation since there is no relationship between these tweets anymore and new emerging tweets belonging to this topic would just cluster to any of these single tweets. How do you handle this issue?

If a tweet comes along that belongs to this cluster, and the cluster (thread) was already deleted, but the tweets are still alive, what I do is simply nothing. The new tweet clearly belongs to either an old topic, a spam topic, or a topic that was not popular enough. It would be wrong to start a new cluster, as this topic already had a chance to "become news". So, when this happens I simply discard this tweet. Maybe there are better ways of handling this, I haven't really invested a lot of time into looking at this. It would certainly be interesting if you could figure out some better approach to handling this situation :)

> Thanks a lot! I will figure something out. Having the output scheme you described I have one more point I am not completely sure about: what means "we just output the fastest growing clusters" in this setup. You look at a cluster when its output_time_interval has passed and decide whether it is growing "fast enough" or not. How do you make this decision?

You could do two things: either have a threshold on number of tweets in the cluster, or at regular time intervals output the top n clusters (say, n=100). This depends on your intended application. For showing things to humans (say, you want to make a demo), the first is better as it's more precision-oriented. For more in-depth evaluations, the second is better.

E-Mail Conversation with Jagan Sankaranarayanan

> Dear Mr. Sankaranarayanan,
>
> My name is Gerhard Grimm, University of Würzburg. Right now, I am conducting in an evaluation of event detection systems and I am reimplementing parts of your system proposed in the paper “TwitterStand: News in Tweets”. Great work!!
>
> There are two points I am not exactly sure about and I would appreciate your help:
>
> 1) You apply a Gaussian attenuator when calculating the similarities. What sigma (standard deviation) do you use in the formula?

This basically controls how fast moving your stories need to be. For instance, if you want news stories to expire quickly then set a smaller sigma while larger one makes them expire slower. I do not recall the value of sigma but we did not want a cluster to stick around for more than a day or so at the most.

> 2) You also talk about solving the problem of fragmentation by periodically checking amongst all active clusters if there are duplicate clusters. How do you decide that a cluster is a duplicate one?

We do that by taking the cosine similarity between the cluster centroids. If they are close to one (say > 0.8) then they are identical. This can be done by comparing the cluster centroids with one another. Note that this does not require all-pair comparison since we are aided by an inverted list on the centroid terms. So it goes pretty quick.

E-Mail Conversation with Karthik Subbian

> Dear Mr. Subbian,
>
> My name is Gerhard Grimm, University of Würzburg, Germany. I am conducting an evaluation of event detection systems in Twitter. Right now, I am reimplementing your system for event detection proposed in your paper “Event Detection in Social Streams”. Great work!
>
> There are a few questions that have arised recently and I would highly appreciate your help here.
>
> When you generate the output of the unspecified event detection process, you continuously monitor the relationship of number of documents that have been added to a cluster in the time interval [t_c – H , t_c] to [t(C_i), t_c – H] and if it exceeds the threshold alpha you output the cluster as an event emerging. How do you choose alpha? Furthermore, I do not understand how this formula works if H is a constant time horizon. If a cluster is already really big it will not be output even if a lot of documents get added in between H. Do you choose H constant or adjust it dynamically?

The formula looks at the rate of change of cluster sizes for the last horizon H. If the cluster is already big, the rate of change has to be significantly larger in order to make the event significant. H has to be fixed upfront, and it depends on the data stream you wish to monitor. If significant events do not happen in hourly intervals for your data stream, make it larger then (may be few hours). The threshold alpha has to be chosen empirically by looking at several events that happened during some validation interval. For instance, start with 1 and increase/decrease the value based on your data stream.