Online News Detection on Twitter

Linn Christina Vikre
Henning Moberg Wold

Master of Science in Informatics
Submission date: May 2015
Supervisor: Jon Atle Gulla, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

In this thesis, we seek to find suitable methods for detecting news on Twitter within the fields of artificial intelligence, information retrieval, computational linguistics, and natural language processing. We combine these fields to find newsworthy tweets, cluster them based on time and similarity and find the most representative tweet for a news event. We compare different methods within the field of topic modeling to find news topics and tweets related to them in an online setting. Then we have a look at an online clustering approach to be able to detect news while they are being clustered based on time of arrival and similar content.

One of the greatest challenges is to find the tweets we can characterize as news in the ocean of tweets. Many tweets are about personal matters or two-way communication among friends. We call these uninteresting tweets "chatter". There are many tweets that contain abbreviations, misspellings, and lack of proper sentence structure. This makes it difficult for otherwise good natural language processing systems to evaluate the content and proper language in tweets.

Our study shows that finding news using topic modeling is difficult. Training a proper model is time consuming, and even when a working model is obtained, it is unclear how to effectively use it to detect news.

While developing the online news detection system for Twitter, we have found that the clustering approach elaborated on in this thesis works well for tweets. The system clusters similar tweets based on time and content, and it performs well doing so. Due to the information entropy and tuning of parameters in the clustering algorithm we were able to achieve a higher precision than the baseline clustering algorithm.

Lastly, we have found that the task of finding the most representative tweet for a news event is simple when the tweets have been clustered well, that is when the clusters contain mostly news relevant twets.

Sammendrag

I denne masteroppgaven forsøker vi å finne egnede metoder for å oppdage nyheter på Twitter. Vi benytter metoder innenfor kunstig intelligens, informasjonsgjenfinning, datalingvistikk og naturlig språk for å utføre oppgaven. Vi sammenligner ulike metoder innenfor "topic modeling" med et mål om å finne nyhetsrelaterte tweeter i en "online" setting. Deretter ser vi på muligheten for å klynge sammen tweets basert på tid og innhold med mål om å kunne benytte dette for å detektere nyheter.

En av de største utfordringene med dette er å finne de tweetene vi kan karakterisere som nyheter. Mange tweeter omhandler personlige forhold, eller er toveiskommunikasjon mellom venner. Videre inneholder mange tweeter forkortelser, feilstavelser og mangel på korrekt setningsstruktur. Dette gjør det vanskelig for standard verktøy innen naturlig språk å evaluere innholdet i disse.

Studien viser at å finne nyheter ved hjelp av "topic modeling" er vanskelig. Det å trene en bra modell er tidkrevene. Selv om man oppnår en slik modell, er det likevel uklart hvordan en effektivt kan benytte denne til å detektere nyheter.

Under utviklingen av nyhetsdeteksjonssystemet har vi kommet frem til at fremgangsmåten med å klynge sammen tweets som forklart i oppgaven fungerer bra for tweeter. Ved hjelp av informasjonsentropi og å justere ulike parameter i systemet greide vi å oppnå høyere presisjon enn det systemet hadde til å begynne med.

Til slutt har vi funnet ut at det å finne den mest representative tweeten for en nyhetshendelse er enkelt, gitt at tweetene er lagt i fornuftige klynger.

Preface

This report presents the work carried out in our master thesis as a part of the courses *IT3902 Informatics Postgraduate Thesis: Data and Information Management* and *IT3901 Informatics Postgraduate Thesis: Software*, and is the final submission to achieve the degree of Master of Science at the Norwegian University of Science and Technology. The work described in this report is carried out as a part of the SmartMedia project at the Department of Computer and Information Science at the Norwegian University of Science of Technology. Our supervisor is Professor Jon Atle Gulla.

Henning M. Wold and Linn Vikre

Trondheim, May 28, 2015

Acknowledgements

We would like to gratefully thank our supervisor Professor Jon Atle Gulla of the Department of Computer and Information Science at the Norwegian University of Science and Technology for his excellent guidance. We would also like to show our gratitude to Dr. Jon Espen Ingvaldsen for his help on the New York Times annotated corpus. We would also like to thank the rest of the members of the SmartMedia project for inspiration and many helpful discussions during the course of writing our master thesis. Finally, we would like to thank Pernille Wangshold for assisting with proofreading the thesis.

Contents

List of Figures

List of Tables

Part I

Research Overview

Chapter 1

Introduction

In this chapter we introduce the research and scope of our thesis, conducted over a year starting in late August 2014. In section 1.1 we present the background and motivation for our work. Section 1.2 gives an outline of the problems addressed in the thesis, and explores the state of the art. The research questions and goals of the thesis are provided in section 1.3. The main contributions of the thesis are described in section 1.4. Three papers have been written as a part of this thesis. Section 1.5 provides the subject and structure of these papers along with the rest of the thesis.

1.1 Background and Motivation

Events happen all the time, all around the world. Some are more mundane events, like a concert or the release of a new movie. Sometimes, however, the events occurring affect many people and are what we would call news. These could be things like elections, crimes, big sporting events and natural disasters. Before these events can be known to the general public, someone must report them. For news this "someone" has traditionally been news corporations with employees all over the world. If something happens, they quickly investigate to be able to be the first to report on a particular event. This early information about events that occur are what we are going to refer to as "breaking news" in this thesis.

Definition of breaking news : "news that has either just happened or is currently happening. Breaking news may contain incomplete information, fac-

> tual error or poor editing because of rush." [Phuvipadawat and Murata, 2010]

In the past these corporations were afforded some time to get these reports out. One reason for this is that television and radio news programs were only broadcast on a set schedule, and newspapers had to be set and printed and as such took some time. Later, for especially big events such as the Kennedy assassination, regular radio and television broadcasts were interrupted to bring the news out more quickly. These days many people utilize the Internet when getting their latest news updates. As this is a media platform that is virtually instant and free to update, there is now much more pressure to get news out more quickly.

1.1.1 Twitter

The Internet has has made it possible for anyone to reach out to a lot of people through social media network sites. One such site is *Twitter*. Since its foundation in 2006, it has gained several hundred million users. Numbers from the fourth quarter of 2014 shows that Twitter has an average of 288 million active users worldwide on a monthly basis[1].

Twitter is a so called microblogging service where users can post messages, called *tweets*, consisting of a maximum of 140 characters. These messages are then displayed on the Twitter homepages (or *feeds*) of any users *following* them. Any user A can follow any other user B, which makes any tweet posted by user B appear in the feed of user A. The act of following does not need to be reciprocated. It is up to user B whether or not to follow user A back when user A follows them.

Other than pure text, there also exist a few other facets to tweets that need to be explained. The first of these is mentions. If a user is mentioned in a tweet they will receive a notification about this on their profile. They can then decide for themselves how to deal with this. *Mentions* are used to reply to users asking questions, but can also be used to get their attention. Mentioning a user in a tweet is simple: anywhere in the body of the tweet, simply put their username prepended with a "@" character. If the username one wishes to mention is "genericuser", for instance, an example of a tweet mentioning that user would be "Went to see the new Star Wars movie with @genericuser".

Another facet to tweets is the *hashtag*. A hashtag is any word in the body of a tweet that is prepended with the character "#". Hashtags are used to tag topics in the tweets by users. When using Twitter's website to view tweets, one can click on any hashtag in a tweet to display all other tweets containing that

[1]http://www.statista.com/statistics/282087/

hashtag. There are no formal restrictions on the use of hashtags (other than the overarching restriction on tweet length) in tweets. Even so, Twitter still recommends using no more than two hashtags per tweet and urge users to keep hashtags relevant to the topic of the tweet[2].

The last facet of tweets we will mention here is *retweeting*. Retweeting is Twitter's facility for disseminating information. As mentioned previously, a tweet is only displayed directly in the feed of users following the author of the tweet. These followers might find the tweet in question interesting and deem it worth sharing with their own followers. To do this they *retweet* the tweet in question, spreading it to their own followers. The most simple way of doing this is pressing the retweet button on a tweet on Twitter's website; this is also the official method. An unofficial way of doing this is posting a new tweet containing "RT" followed by mentioning the user, immediately followed by a colon and lastly the entire contents of the original tweet. Optionally it is possible to include your own comments. Example of a retweet: "Good idea! RT @genericuser: I want to learn #programming."

1.1.2 News and Twitter

The combination of the ease of disseminating information and the fact that people today crave quick news delivery has presented Twitter as an interesting platform for the dissemination of news. Research (such as Sakaki et al. [2010] and Hu et al. [2012]) has shown that quite a few big news stories has broken on Twitter earlier than in more traditional news media. An example of this is the news about Osama bin Laden's death. In that case, several sources on Twitter leaked the information before the President of the United States announced that bin Laden had been killed [Hu et al., 2012].

Twitter is thus an interesting service to scan in order to detect news before they are published by more traditional news media, sating the public's craving for immediate notification if anything happens.

1.1.3 SmartMedia

The work presented in this thesis is part of the SmartMedia project of the Department of Computer and Information Science at the Norwegian University of Science and Technology. The project is affiliated with the NxtMedia, whose goal is to develop an industrial cluster for media technology based in Trondheim,

[2]https://support.twitter.com/articles/49309

Norway. NxtMedia is partnered with various academic and media institutions, including Adresseavisen and the Norwegian University of Science and Technology. The project aims to make a personalized news recommender system, where it is possible to customize various parameters about which news gets displayed. This personalization can be based on personal interests such as sport or politics. It can also be location based e.g. for a country, the world or just your home town. The news presented in the system developed by SmartMedia are news published by traditional news sources such as newspapers.

The SmartMedia project saw potential in leveraging Twitter to detect news. They wanted to research the possibility of including tweets in their news recommender system. The research accomplished during the work on this thesis has been performed to enable the system to accomplish this.

1.2 Problem Outline

The main motivation for this thesis is to discover news as the event they are describing is unfolding, by using tweets posted by users of Twitter. If successful, this should enable us to discover breaking news such as natural disasters, political unrest and big happenings. This, in turn, could lead to more efficient disaster relief. As this thesis and work is a part of the SmartMedia project, it also gives us more options for which sources are available in their news recommender system.

This motivation can be condensed into two problems we need to figure out how to solve in sequence:

1. Find those *tweets* that are news relevant in near real-time
2. Given a set of *tweets* concerning the same event, decide which of them is the most representative of the set

The first point is fairly general, though it specifies that we need to detect the news in near real-time. If we did not detect the news tweets in near real-time, there is a high likelyhood that the information they provide is already outdated when we detect them. Analyzing a data set that is a day old could detect news there, but they would be the news of yesterday. For the tweets to be of interest they should be fresh.

When detecting news on Twitter, it is likely that there will be several similar tweets concerning the same event in the same timeframe. In a news recommender setting (such as the SmartMedia system), however, we only want one of these tweets to be posted. This should be the tweet that contains the most information about the event.

The challenges of the task lie in processing the data received from Twitter. This processing has to be fast, or else it will cease to be near real-time. Generally the time it takes to process one tweet needs to be strictly less than the average time between two tweets arriving from Twitter. The processing also has to be able to evaluate the content of any tweet to decide whether or not it is a news tweet. This includes being able to discern between "chatter", spam and tweets describing an event.

1.3 Goals and Research Questions

The research presented in this thesis aims to develop a system for news detection on Twitter. We want to be able to represent breaking news in the SmartMedia news recommender system. This means we have to develop a system that is able to detect newsworthy tweets (RQ1 and RQ2). We also have to cluster them based on time and content (RQ2), which results in us being able to find the most representative tweet in these clusters (RQ3).

RQ1 To what extent can topic modeling be used to detect breaking news on Twitter in an online setting?

RQ2 How can we cluster tweets based on content and time to detect news?

RQ3 How do we select the most representative tweet from a cluster of tweets?

1.4 Research Contributions

The three main contributions of this thesis, which answer the research questions defined in the previous section, are:

C1 Keeping topic models of tweets accurate and up-to-date in a real-time setting is difficult.

As elaborated in chapter 3, topic modeling tweets is very difficult to accomplish well in a real-time setting, if one wishes to keep the model up-to-date.

Topic modeling could produce fairly good results if using a static topic model. In the long run, however, it is likely that such a static model would get more and more separated from actual current topics, rendering it effectively useless.

C2-1 Locality-sensitive hashing can be used both for clustering tweets and for detecting breaking news based on time and content.

C2-2 Utilizing named-entity recognition in tandem with locality-sensitive hashing provides higher precision than using locality-sensitive hashing alone.

As detailed in chapter 4 we can use locality-sensitive hashing in tandem with named-entity recognition to cluster similar tweets, and eliminate those clusters which do not contain news. The approach has good performance both considering processing time and memory footprint. The approach seems more suited for Twitter's nature (short messages and sparse text representation) than the topic modeling approach.

C3 Selecting the tweet of a cluster with the highest cosine similarity to the tf-idf weight centroid of the cluster is sufficient to decide the most representative tweet of the cluster.

As described in chapter 5, our approach for finding the most representative tweet in a cluster had uplifting results. Although the approach is relatively trivial it still had high precision, even in a somewhat noisy data set where the comparative approaches faltered.

1.5 Thesis Structure

As a part of the contribution of this thesis, three papers have been produced. These are organized as independent, self-contained chapters and can be found in chapters 3, 4 and 5.

The first paper explores the possibility to do topic modeling on tweets in order to find breaking news. The second paper suggests a method to cluster tweets in real-time, that also proves to be useful to find news on Twitter. The concluding paper aims to find a suitable method to decide which tweet of a cluster containing related tweets is the most representative.

Part I - Research Overview

Chapter 1 - Introduction This chapter describes the background and motivation for the research in this thesis in addition to elaborating on the research questions, goals, contributions and give a general description of the papers given in the subsequent chapters.

Chapter 2 - Theoretical Background This chapter accounts for some of the essential theories that this thesis relies on, but which are not specifically elaborated on in the articles.

Chapter 3 - Twitter Topic Modeling for Breaking News Detection This chapter describes topic modeling approaches for how to detect potential breaking news being tweeted about on Twitter.

Chapter 4 - Hybrid Entity Driven News Detection on Twitter This chapter describes a method for clustering tweets and how the tuning of parameters combined with named-entity recognition results in finding news relevant tweets.

Chapter 5 - Find the most representative news tweet In this chapter we describe a rather straightforward method for finding the most representative tweet in a cluster.

Chapter 6 - Evaluation and Results In this chapter we summarize the research and results obtained from the previous chapters. We illuminate the whole problem, and discuss how the previous chapters are able to find a suitable solution to the main issue. We also discuss some difficulties encountered during the research.

Chapter 7 - Discussion and Conclusion In this final chapter we summarize our findings, and draw up some suggestions for future work in the research field.

Chapter 2

Theoretical Background

In this chapter we elaborate on some of the theoretical background used when working on the thesis. Section 2.1 elaborates on various techniques for text processing, including tf-idf weighting and named-entity recognition. It also explains the difference of online versus offline processing of data. Section 2.2 discusses some background on topic modeling techniques. Finally, section 2.3 is about the principle of locality-sensitive hashing.

2.1 Text processing

In this section we give a short summary of prevalent text processing theory utilized in this work. We start by giving a short explanation of the vector space model (including term frequency (tf), inverse document frequency (idf) and cosine similarity), before discussing various methods to preprocess and tokenize documents in general, and tweets in particular.

2.1.1 The vector space model

The vector space model is a model used to represent text algebraically. It works by turning a text document into a vector, where each dimension of the vector corresponds to a separate term. If a document contains a term t, the dimension of the vector corresponding to that term will be non-zero. This model is also called a bag-of-words model. This is because each document is seen as a bag containing words (terms) of various quantities. Moreover the order of the words, as well as any syntax and semantic meaning, are ignored in this model.

Many schemes exist for weighting these dimensions, but one of the more well known schemes is the one we use in this thesis: term frequency-inverse document frequency, or tf-idf, weighting. To explain tf-idf weighting, we must first explain its constituent parts.

Term frequency (tf)

The term frequency $tf(t,d)$ for a term t in a document d is a measure of how often t occurs in d. It is a useful measure to see how important a term is to a document. Certain common words like "the" and "a" will get high term frequencies in almost all documents, but this is corrected for in the full tf-idf weighting.

The simplest choice for this measure is to take the number of times a term t occurs in d, or $tf(t,d) = f(t,d)$. There are other choices one can make for this measure. One is boolean frequency, where $tf(t,d) = 0$ if the term does not occur in the document, and 1 otherwise. Another is to logarithmically scale the frequency, i.e. $tf(t,d) = 1 + log f(t,d)$, which ensures the term frequency grows more slowly. We have chosen to use the $tf(t,d) = f(t,d)$. Our reasoning is that if a term is included more than once in a document containing at most 140 characters, it can be inferred that it is a very important term to the document.

Inverse document frequency (idf)

The inverse document frequency $idf(t,D)$ for a term t in a collection of documents D is a measure of how much information t provides. Another way of stating it is that it is a measure of how common t is in D. Common words such as "the" and "a" will have very low $idf(t,D)$ values, even as low as 0 if the term is present in every document. The idf is calculated as:

$$idf(t,D) = log \frac{N}{\{d\ inD : t \in d\}}$$

Where N is the total number of documents in D, and $\{d\ inD : t \in d\}$ is the number of documents d in the collection which contains t. If there are no such documents, however, we will get a division by zero. As all idf calculations we do are with documents already in the corpus this is not a problem we will encounter. If it is a potential issue, however, one can simply add one to the denominator to avoid this issue.

Term frequency-inverse document frequency weighting

Combining tf and idf enables you to give a high score to those terms that are frequent in one or a few documents, but rare in the corpus at large. It is defined as $tf - idf(t, d, D) = tf(t, d) \times idf(t, D)$.

By using tf-idf as the weighting model in the vector space model, each dimension in the vector of a document is a measure of how important that term is to the document.

Cosine similarity

Cosine similarity is commonly used to calculate the angle between two vectors 2.1. Using the vector space model, we can represent documents as n-dimensional term vectors, which enables us to calculate the cosine similarity between two documents [Huang, 2008]. Cosine similarity between two documents, d_1 and d_2, can be written formally as:

$$sim(d_1, d_2) = \frac{\vec{v}(d_1) \cdot \vec{v}(d_2)}{|\vec{v}(d_1)||\vec{v}(d_2)|}$$

Here $\vec{v}(d_1) \cdot \vec{v}(d_2)$ in the numerator signifies the inner product (that is sum of the pairwise multiplied elements) of the document vectors of d_1 and d_2. The denominator is a normalization factor that handles and discards the effect of document lengths on the similarity score [Lee et al., 1997].

If the two documents compared share no terms at all, their cosine similarity will be 0. If the documents are exactly identical it will be 1. The more terms the documents have in common, the higher this score will be. Note, however, that as our vector space model uses a bag-of-words approach, you can compare a document d to its inverse (that is every term in d in the opposite order), and still get a cosine similarity of 1. It is thus not a foolproof method of similarity scoring. It is, however, simple to implement and fast.

2.1.2 Preprocessing of text

Tokenization

Tokenization is the first process used in natural language processing (NLP) in order to form tokens known as strings of multiple characters from an input stream,

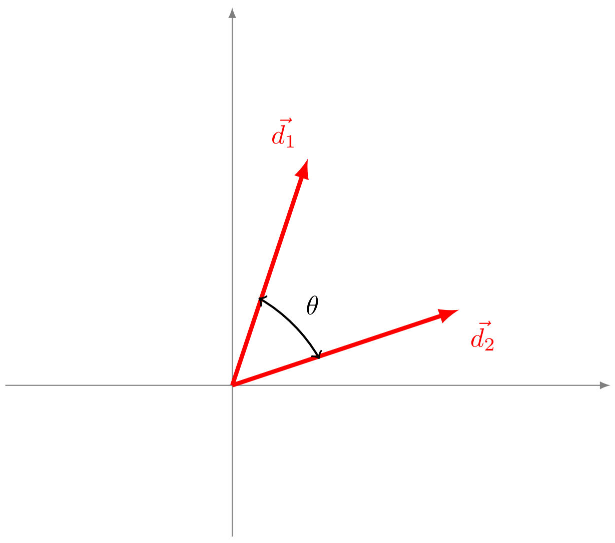


Figure 2.1: A graphical representation of a vector space model and the angle θ between the documents $\vec{d_1}$ and $\vec{d_2}$.

and group these together where it seems that the characters are related to each other [Webster and Kit, 1992]. This count for special characters as commas and digits as well as common known alphabetic characters.

Text normalization

Real text these days often contain non-standard words as abbreviations and acronyms or disambiguation of words. These words are not typically found in a dictionary and develops through the time, or as it is necessary e.g. in tweets with limited amount of characters. Text normalization of these acronyms or abbreviations can be done by replacing the shortened word with the contextual appropriate word or word sequence [Sproat et al., 2001].

Stemming and Lemmatization

Stemming is the computational procedure of reducing all word to the same root, or stem, and is done usually by stripping each word of its suffix and derivation [Lovins, 1968]. While lemmatization is the process of finding the lemma, or the normalization, of words such as reduce *running* to its base form *run* [Korenius et al., 2004]. Stemming is an algorithmic approach, while lemmatization is based on lexical analysis.

Online processing versus offline processing of data

There are two distinct approaches when processing data, such as tweets. The first is to collect all the data one wishes to work on and then, after it has been collected, analyse this data in its entirety all at once. This approach is called offline processing of data. With this approach all data is available at the start, which means you have certain liberties in how to perform the analysis. For one you know exactly how large the data set is, enabling you to tweak size dependent parameters in your analysis tool to better fit the data set. Another great benefit is that it enables you make several passes of the data to improve the accuracy and precision of the analysis. Furthermore, as all the data is already there, it enables you to selectively put those points that are of interest into memory, and keep the rest in whatever storage medium is used. The main caveat of this process is that as you need to gather all the data you wish to process in advance, it is inherently unsuited for processing streams of data arriving in real-time.

The other approach, called online processing, is to continuously analyse and process the data as it arrives from the source [Li et al., 2004; Babcock et al., 2002]. This means you are able to process data in real-time. This, however, brings some stringent restrictions. Data will be arriving (the data source becoming unavailable notwithstanding) constantly without end until the processing tool is shut down. Furthermore, processing in this way requires, on average, each data point to be fully analysed more quickly than new data points are generated. If this restriction is not upheld the queue of data points waiting to enter the processing tool will, in time, grow larger than the memory allocated to hold this queue.

Solutions for online processing of data requires smart designs. One is the choice of what to keep in memory, what to discard, and what to store to disk, or output in any way. In theory a stream of data points can go on forever, meaning keeping all the data points in memory will eventually exhaust available memory. Another choice is how to perform all necessary processing to a data point in a single pass, as when working on a stream of data points there is no feasible way to “rewind” to a previous point.

Despite these challenges, if one wishes to perform processing on a real-time stream of data, online processing is how it must be done.

2.1.3 Named-entity recognition

Named-entity recognition (NER) belongs to the domain of "information extraction" in computational linguistics and can be explained as the procedure of extracting specific information from documents. In other words is it a commonly used technique to analyse and understand the meaning behind a sentence or a longer sequence of word e.g. a document [Liu et al., 2011].

2.2 Topic modeling

In this section we elaborate on topic modeling, which the breaking news system of our first article (found in chapter 3) relies on.

Topic modeling is a statistical method used in both machine learning and natural language processing to discover the underlying semantics (i.e. topics) in a collection of documents [Papadimitriou et al., 1998]. This method assumes that any given document has at least one topic associated with it. This means we can expect tokens (words in our instance) related to that topic to appear in the document, as illustrated in figure 2.2. A document will typically have multiple topics associated with it. These different topics are distinct, and concern different parts of the document. We can use topic modeling on a corpus of documents to infer these each topics, creating probability distributions over words in the corpus enabling us to determine whether a document belongs to a given topic or not. The strength of topic modeling methods is that they do not require any prior labeling of the documents (although this can aid in the initial training of the topic model).

Several different topic modeling methods exist for solving different challenges. Those that we have examined in more detail while performing research and experiments for our thesis have been described in further detail in section 3.4.

2.3 Locality-Sensitive Hashing

Locality-Sensitive Hashing (LSH) was originally designed by Indyk and Motwani [1998] and was created for solving the nearest neighbor search problem in high dimensional spaces. As described in chapter 4, the LSH hashes the input into buckets where all items that are similar has a high probability of collision, thereby putting them in the same bucket.

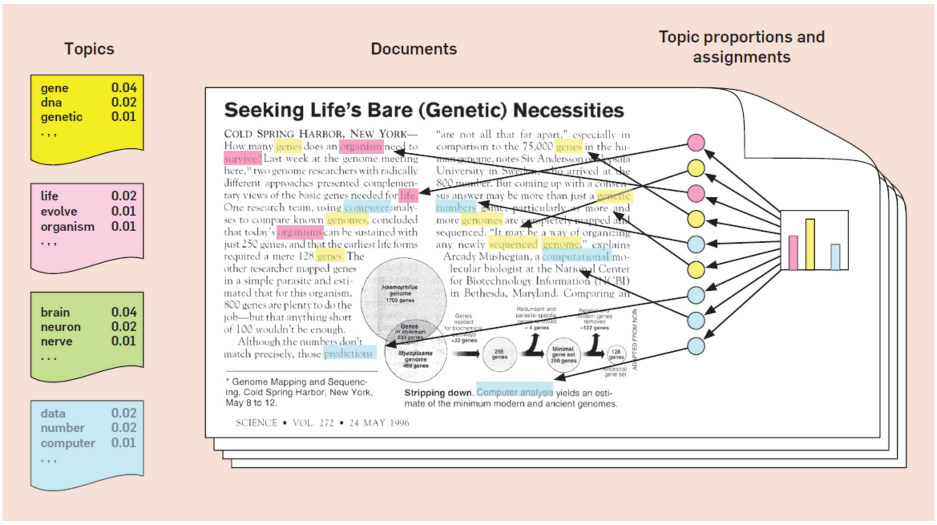


Figure 2.2: The intuition behind the LDA [Blei, 2012]. How the topics are represented by a probability distribution over words in the corpus.

When hashing, a vector of the tf-idf weights of the document's terms is created, before normalizing with the Euclidean norm. This normalization is done to minimize the effects of difference in length between documents. The dot product of this vector and each of k vectors, whose elements have been randomly selected from a random Gaussian distribution, is then used to determine the bits of the hash value. If the dot product between the tf-idf vector and a random vector is ≥ 0, then the hash bit that corresponds to that vector is set to 1, otherwise it is set to 0. As the random vectors do not change between each hash calculation, this means that very similar tweets get assigned the same hash value.

Increasing the value of k increases the number of possible hash values. It also has the effect that fewer document collide in any given hash value (also called bucket). This, however, also has the effect of decreasing the probability of a tweet colliding with its nearest neighbor. To alleviate this, we can create L hash tables each with independently created random vectors. As we use random vectors to determine the hash values, this will result in a tweet getting a different hash in each of the buckets, retaining the overall premise that it will get clustered with similar tweets.

We use the hashing scheme defined by Charikar [2002], which gives a locality sensitive hashing scheme where

$$\mathbf{Pr}[h(A) = h(B)] = 1 - \frac{\theta}{\pi}$$

where

$$\theta = \cos^{-1}\left(\frac{|A \bigcap B|}{\sqrt{|A| \cdot |B|}}\right)$$

Chapter 3

Twitter Topic Modeling for Breaking News Detection

Abstract

Social media platforms like Twitter have become increasingly popular for the dissemination and discussion of current events. Twitter makes it possible for people to share stories they find interesting with their followers, and write updates on what is happening around them. To make use of the information made available through Twitter, this article explores different methods to create topic models of *tweets* in real-time. The article will mainly explore two different methods of topic modeling: Latent Dirichlet Allocation (LDA) and Hierarchical Dirichlet Process (HDP). The methods are tested with each *tweet* in the training corpus as its own document, as well as with all the *tweets* of a unique user regarded as one document. This second approach emulates Author-Topic modeling (AT-modeling). The evaluation relies on manual scoring by a test panel of the accuracy of the modeling. The experiments indicates that it is hard to detect breaking news using topic modeling on *tweets* in real-time.

3.1 Introduction

Social media networks facilitate communication between people across the world. Not only have social networks made it easier for people to communicate, they have also made it possible for the media to capture breaking news as they are emerging. Social media networks have been used to provide information in real-time about larger crisis situations such as earthquakes and tsunamis [Mendoza et al., 2010].

Twitter is one such social network, and is a micro-blogging service founded in 2006. As of 2014 the company reports having 284 million active users per month[1]. The service is focused on micro messages, called *tweets*, which are restricted to a length of 140 characters. In addition to posting *tweets* about anything, users can also follow other users.

Sakaki et al. [2010] examined how earthquakes could be detected using Twitter. In their research they treated Twitter users as sensors and the *tweets* as sensor data. Using their approach they were able to detect 96% of the earthquakes with intensity 3 or more occurring in the examined area. In Hu et al. [2012] the authors show how the news of Osama Bin Laden's death spread on Twitter before the mass media could get the news confirmed.

These studies suggest that Twitter can be used effectively to detect breaking news before they are published in traditional news media. The issue then becomes to detect those *tweets* that can be considered news-worthy and filtering out those that are noisy. By noisy we mean *tweets* that are expressing personal matters. Subsequently, for those *tweets* detected in the first process, the *tweets* that are deemed untrustworthy need to be pruned. This article concerns the first of these issues and seek to find a possible strategy for collecting breaking news through Twitter using topic modeling techniques.

In section 4.2 we present some previous research in the field of topic modeling in general, and on *tweets* in particular. Section 4.3 describes how we aim to find a suitable topic modeling technique to handle tweets in real time. After that, we evaluate the results of the different topic modeling techniques in section 3.5. At the end or the paper, in section 3.7, we summarize our findings along with a conclusion in addition to a proposal of further work.

3.2 Related work

In recent years, topic modeling has been increasingly utilized for analyzing corpora of *tweets*. Latent Dirichlet Allocation, or LDA [Blei et al., 2003] for short, is today one of the most widely used techniques. One result of this is several modifications and extensions proposed to LDA that improves its performance in social media settings in general, and for *tweets* in particular.

[1]https://about.twitter.com/company

One example is Rosen-Zvi et al. [2004], who proposes an extension to LDA called the Author-Topic Model (AT model). In their research [Rosen-Zvi et al., 2010], they show that when the test documents only contain a small number of words, their model outperforms LDA. This research was done on a collection of 1,700 NIPS conference papers and 160,000 CiteSeer abstracts. This is relevant for our study as abstracts are summaries of texts, which are shorter than normal documents but still longer that a regular *tweet*.

Hong and Davison [2010] showed how training a topic model on aggregated messages resulted in a higher quality learned model, yielding better results when modeling *tweets*. Of particular interest to us is their USER scheme, in which they aggregate all the messages of a particular user before training the LDA model on them. This is a simple and straightforward extension to LDA, which gives a more accurate topic distribution than running LDA on each *tweet* individually.

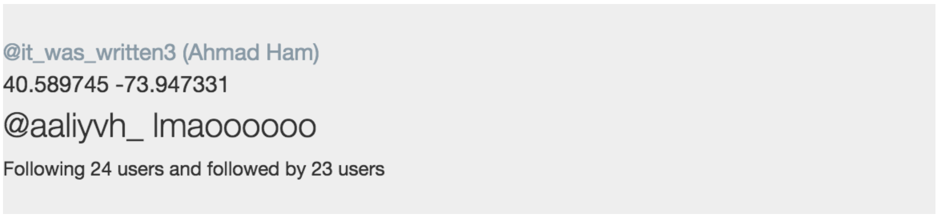
In Zhao et al. [2011] the authors did an empirical comparison between Twitter and the New York Times using their own extension to LDA which they dubbed Twitter-LDA. They found that Twitter could be a good source of news that has low coverage in other news media. They also found that while Twitter users are not exceptionally interested in world news, they do help spread awareness of important world events.

Another method that has been used for topic modeling *tweets* is Hierarchical Dirichlet Processes (HDP) [Teh et al., 2006]. HDP is a nonparametric Bayesian approach which is used to cluster related data. In our case, this would be to cluster *tweets* that have similar topics. Wang et al. [2013] describes how to use HDP to detect events occurring from *tweets* in real-time and shows how the clustering in HDP works on these *tweets*.

All of these studies suggest that it is possible to accomplish our goal. Additionally, they suggest that there are several techniques and methods which perform well in different areas that can be utilized for our purposes. Our challenges are:

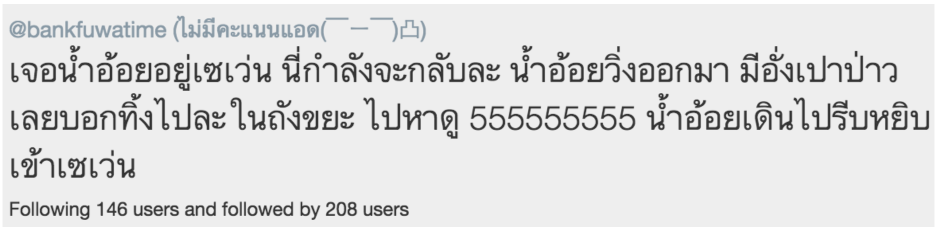
- Find a suitable topic modeling method for tweets
- Find out how to filter the news in the topics we get from the topic modeling
- Do this in real-time, or as close to as possible

Although there exist studies that have experimented with what we aim to achieve [Zhao et al., 2011], there are few who have tried to do this in real-time. Most studies have done this in semi real-time or looked at previous data to see if they could get an indication of whether or not it is possible to fetch potential breaking news using Twitter. Another aim we have is to continuously train our model as we acquire new *tweets*.



@it_was_written3 (Ahmad Ham)
40.589745 -73.947331
@aaliyvh_ lmaoooooo
Following 24 users and followed by 23 users

Figure 3.1: Example of a short and non-serious tweet



@bankfuwatime (ไม่มีคะแนนแอด(￣－￣)凸)
เจอน้ำอ้อยอยู่เซเว่น นี่กำลังจะกลับละ น้ำอ้อยวิ่งออกมา มีอั่งเปาป่าว เลยบอกทิ้งไปละในถังขยะ ไปหาดู 555555555 น้ำอ้อยเดินไปรีบหยิบเข้าเซเว่น
Following 146 users and followed by 208 users

Figure 3.2: Example of tweet written in a foreign language

3.3 Twitter News Detection

When news-worthy events happen, people who witness it are often quick to post about the event to their social network feeds in general, and to their Twitter accounts in particular. In Kwak et al. [2010] it was found that any retweeted *tweet* reached an average of 1,000 users. This means Twitter could be an interesting place for detecting breaking news as they are emerging. Unfortunately, many of the posts on Twitter are not news items, but rather address personal opinions and mundane status updates or similar, such as those found in figures 3.1 and 3.4. Other *tweets* are more serious and potentially related to news, such as figure 3.3. Lastly some *tweets* (e.g. figure 3.2) are entirely written in foreign languages often using non-latin alphabets. The first step in detecting breaking news is then to filter out the noise, leaving posts that are potential news for the analysis.

The main challenge of news detection on Twitter is the length restriction on *tweets*. A *tweet* can be a maximum of 140 characters long. Because *tweets* are so short, simply looking for certain keywords will result in only a handful of the actual news posts being detected, as these posts might not directly use any of these keywords. In addition, the list of keywords would have to be updated manually as the news picture evolves over time.

We have instead decided to focus on a few different topic modeling techniques

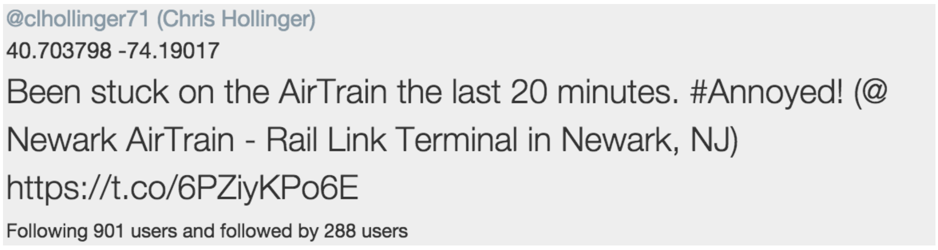

@clhollinger71 (Chris Hollinger)
40.703798 -74.19017
Been stuck on the AirTrain the last 20 minutes. #Annoyed! (@ Newark AirTrain - Rail Link Terminal in Newark, NJ) https://t.co/6PZiyKPo6E
Following 901 users and followed by 288 users

Figure 3.3: Example of long, serious tweet

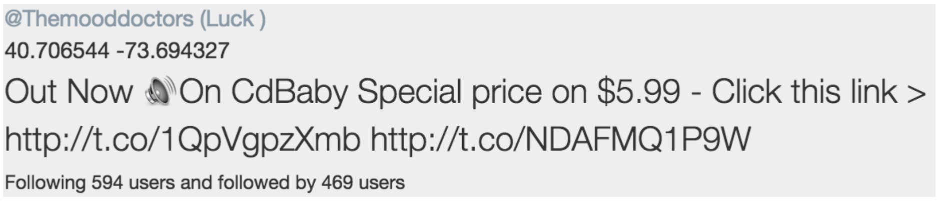

@Themooddoctors (Luck)
40.706544 -73.694327
Out Now 🔊On CdBaby Special price on $5.99 - Click this link > http://t.co/1QpVgpzXmb http://t.co/NDAFMQ1P9W
Following 594 users and followed by 469 users

Figure 3.4: Example of spam tweet

enabling us to dynamically find potential news when combining the techniques with a news index. These techniques allow for updating the models over time, ensuring they stay relevant.

3.4 Methodology

In the following section, we introduce a set of methods for training topic models concerning Twitter. After clarifying the theoretical aspect of the LDA and HDP models we discuss how they can be used to topic model *tweets* in real-time.

3.4.1 Latent Dirichlet Allocation

Latent Dirichlet Allocation (LDA) is a machine learning technique which can identify latent information about topics in large collections of documents. LDA treats each document as a vector of word counts where each document is a probability distribution over some topics, and each topic is the probability distribution over a number of words. For each document in the collection the LDA algorithm picks a topic according to the multinomial distribution of words in the document. It then uses the topic to generate the word itself according to the topic's multinomial distribution and repeats these two steps for all the words in the document [Blei et al., 2003].

LDA can be defined formally as follows:

1. A number of topics t, documents d, and a length of a document N.
2. T distributions over the vocabulary where β_t is the distribution over words in topic t.
3. D distributions over topics where θ_d is the distribution of topics in document d.
4. The topic given for document d is z_d, and $z_{d,n}$ is the assignment of a topic to the nth word in document d.
5. The observed words in document d are given as w_d, where $w_{d,n}$ is the nth word in document d.

This gives us

$$p(\beta, \theta, z, w) \qquad = \qquad \prod_{i=1}^{T} p(\beta_i) \prod_{d=1}^{D} p(\theta_d) (\prod_{n=1}^{N} p((z_{d,n}|\theta_d)) p(w_{d,n}|z_{d,n,\beta}))$$

LDA has become one of the "state-of-the-art" topic modeling algorithms in the later years after it was presented in Blei et al. [2003]. The algorithm uses the "bag-of-words" principle to represent the documents, which is satisfactory when you are dealing with larger documents where it is possible to explore co-occurrences at the document level. With this you achieve a clear overview of all topics related to a document [Titov and McDonald, 2008], which in many cases is exactly what you want.

The model has been shown to work fairly well when topic-modeling *tweets*. Some studies, however, suggest that slight modifications to the LDA model, such as the AT model [Hong and Davison, 2010] and Twitter-LDA [Zhao et al., 2011] perform even better. One reason for this is the document length, which has a fairly large impact on the outcome of the topic modeling result.

Online LDA

As we are interested in topic modeling a real-time stream of *tweets*, traditional LDA may not be sufficient for our purposes. For this reason, we have elected to use *Online LDA*. Online LDA is a variation of LDA in which the model can be updated on the fly. This means that you update and continuously build upon

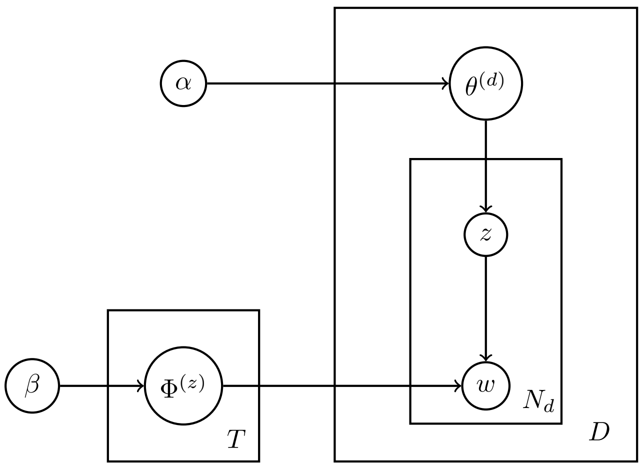


Figure 3.5: Latent Dirichlet Allocation [Blei et al., 2003], topic model

the already existing model, which is based on the information that comes from the data stream. In our case that means new documents (that is *tweets*) will be added to the model continuously as they are received by the system.

3.4.2 Author-topic

This method is an extension of LDA. It simultaneously models the content of each document and the interests of authors. AT uses probabilistic “topics-to-author” model, which allows a mixture of different weights (θ and ϕ) for different topics to be determined by authors of a specific document [Rosen-Zvi et al., 2004]. Figure 3.6 shows the generative model for documents in the AT-model as described by Rosen-Zvi et al. [2004].

The AT-model as defined would result in a very large memory footprint if directly implemented. The reason for this is that we would have to store all inbound *tweets* to be able to continuously update the documents written by each author, which would scale linearly in time. What we have instead elected to do is to replicate the approach of Hong and Davison [2010]. This involves aggregating all the documents made by a single author into one document. In our case this means concatenating all the tweets of a unique user into one document.

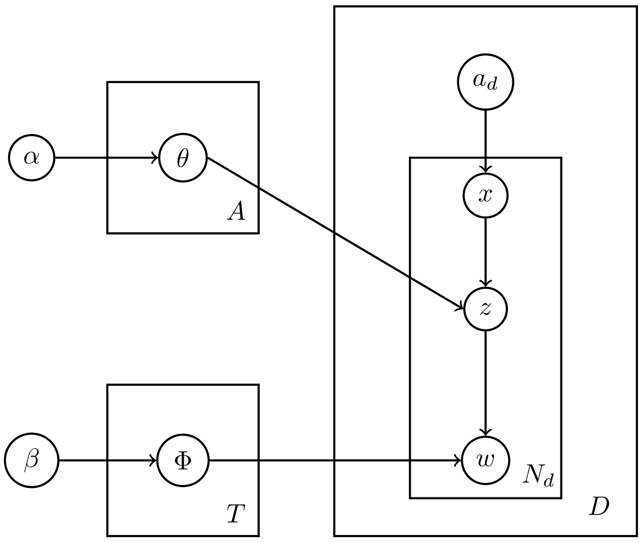


Figure 3.6: Author-topic model

3.4.3 Hierarchical Dirichlet Process

The Hierarchical Dirichlet Process (HDP) is a topic modeling technique for performing unsupervised analysis of grouped data. HDP provides a nonparametric topic model where documents are grouped by observed words, topics are a distribution over several terms, and every document shows the different distributions of topics. Its formal definition, due to Teh et al. [2006], is modeled in figure 3.7 and can be defined by:

$$G_0|\gamma, H \sim DP(\gamma, H)$$

$$G_j|\alpha_0, G_0 \sim DP(\alpha_0, G_0)$$

for each group j. Here G_j is the random probability measure of group j. Its distribution is given by a Dirichlet process. This Dirichlet process depends on α_j, the concentration parameter associated with the group, and G_0. G_0 is the base distribution shared across all groups. This base distribution is given by a Dirichlet process as well, where γ is the base concentration parameter associated with the group and H is the base distribution which governs the a priori distribution over data items.

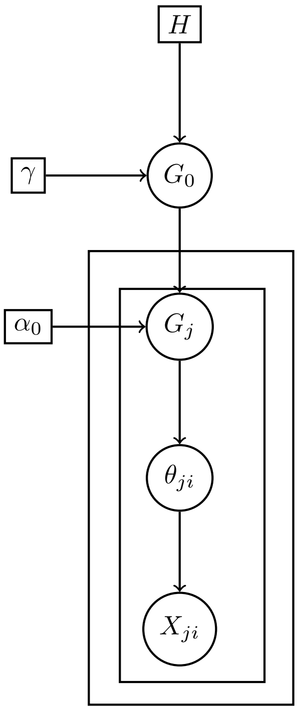


Figure 3.7: HDP topic model

One limitation of standard HDP is that it has to crawl through all the existing data multiple times. Therefore Wang et al. [2011] proposed a variant of HDP which they called *Online Hierarchical Dirichlet Process*. The Online-HDP is designed to analyze streams of data. It provides the flexibility of HDP and the speed of online variational Bayes. The *online* aspect of the Online-HDP is both a performance improvement and a variation in how models are updated. The improvement in performance is achieved by not having to crawl through the entire corpus repeatedly. Instead of having several passes with a fixed set of data, it was suggested by Wang et al. [2011] that you can optimize the updates of the model by iteratively choosing a random subset of the data, and then updating the variational parameters based on the current subset of data.

3.5 News Detection Experiment

In the experiment, we utilize a set of *tweets* to train a topic model using the LDA and HDP models. We train both models treating each *tweet* as its own document, in addition to aggregating all *tweets* of a single user into one document. This gives us four different approach to compare. Our motivation for training the models in two different ways using the same dataset is to see if we can counter the biggest problem with modeling *tweets*: their short length. As this aggregation of *tweets* is an attempt to emulate the AT-model, we refer to the experiments using this dataset as LDA-AT and HDP-AT.

We begin by describing the data collection, before briefly describing the pre-processing steps and training of the models. After that, we test the various methods described in the previous section on a set of test *tweets*, separate from the training *tweets*.

3.5.1 Data set

For our experiments, we have fetched data from Twitter's own streaming API[2]. Our training set contain approximately 600,000 *tweets* from the New York area, collected in the period from November 26, 2014 to December 5, 2014.

Using Twitter's API you get an extensive set of metadata connected to each *tweet*. Most of this metadata is of no significance for topic modeling, and is stripped away. For the purposes of the experiment, the only things we keep are the screen name of the author and the contents of the *tweet* itself.

As presented in section 4.2, there are several studies on topic modeling microblogging services like Twitter. We will use some of the same approaches as these earlier studies, but our main focus will be finding which approach is the best fit for topic modeling *tweets* in real-time.

Based on an idea from Meyer et al. [Meyer et al., 2011], we have elected to ignore any *tweet* containing non-ASCII characters. The rationale behind this is there were several *tweets* containing nothing but emoticons, as well as several *tweets* written using non-ASCII characters (such as Arabic and Chinese). The danger of doing this is that a few relevant *tweets* might also be removed. We have nevertheless decided this is a fair tradeoff. If we were to include *tweets* made in foreign languages using a non-ASCII alphabet, the complexity of the analysis would increase. Moreover we are interested in presenting the breaking news to a user who, presumably, knows English but not necessarily other languages; we would have to

[2]http://dev.twitter.com

assume this user would be unable to read any content presented to them written in other languages. Additionally *tweets* containing unicode emoticons are mostly status updates or chatter, which can be safely ignored. Finally, if an actual news post gets filtered out due to it including non-ASCII characters, chances are high someone else will have posted about this same event without utilizing non-ASCII characters. All these points considered, we decided that it was a fair tradeoff to ignore all *tweets* containing non-ASCII characters. In our data set around 30% of the *tweets* are ignored due to their inclusion of non-ASCII characters.

In our training set we have also removed common stop words, but we have not performed any stemming on the words or other word processing on the *tweets*. We have replaced all URLs with the word "LINK", but kept all hashtags. Additionally, we have removed all non-word characters from words, meaning hashtags lack the '#' sign and appear as normal words. We also made a copy of the data set where all *tweets* for a user were merged together.

Before performing the experiment, we trained the different algorithms on the dataset described above. We elected to use the Python library Gensim[3] as it has embedded support for both LDA and HDP. For both models, the number of topics were set to 50. This number was chosen based on earlier research done by Hong and Davison [Hong and Davison, 2010] which showed that the best results were given if the number of topic were set to 50. There are some additional parameters that can be tweaked for the models, and we tried a few different settings. At first we made the LDA model update in chunks of 1000. This means that the model is trained with 1000 *tweets* at a time. For the training set where each user's messages are aggregated, this meant that the model was updated in chunks of 1000 unique users. It turned out from our experiments that doing this caused one topic to be, for lack of a better word, "inflated". Almost every single message we attempted to classify using a model trained in this fashion was classified into the same, highly general topic. By increasing the chunk size by a factor of ten to 10,000, this phenomenon disappeared.

Furthermore, we set the number of passes to perform with LDA to 10. This was chosen to ensure the initial training set converged well on topics. For HDP, we set the chunk size to 256. When doing online training (that is updating the model with real-time *tweets*), it isn't possible to change these parameters - the model is simply updated straight away with the provided corpus (new messages received in real-time). It is somewhat possible to adjust the chunk size with real-time messages by doing batch updates. The size of the batches will have to be balanced around not being too rare in addition to not being too small so that they skew the models. A compromise here from our trials seem to be update at

[3]https://radimrehurek.com/gensim/

about every 500-1000 new *tweets* that arrive.

After training the models, we used them on a set of 100 *tweets* collected in the same time period as the testing set. These *tweets* were new in that they were not part of the training set.

Additionally we used the models on portions of the New York Times annotated corpus to assess which topics were most frequently found in actual news articles. We did this by tallying up the most relevant topic for each of the articles in the corpus. Doing this gave us a list over the topics most related to the articles in the New York Times corpus. The "score" of a topic, then, is the number of articles in the New York Times corpus that topic was the best fit for. This was done as a part of filtering tweets concerning news and not as part of the experiment described below.

3.5.2 Conducting the experiment

To conduct our experiment we asked 7 people to participate. As mentioned above, we had collected 100 *tweets* for this purpose. We utilized each of the trained models (LDA, LDA-AT, HDP, HDP-AT) to topic model these new *tweets*. The results of this topic modeling was distributed to our participants where they graded how well the topic assigned fit on a scale from 1-5. Here a score of 1 means the topic is not relevant, and 5 means it is a perfect fit.

3.5.3 Examples of distribution of topics

As shown in table 3.1, some of the resulting topics are very generic and cover a broad set of terms. This is not surprising, as many *tweets* are about the everyday affairs of their authors. As we collected *tweets* from the New York area we expected an abundance of *tweets* about things occurring there.

3.5.4 Results

Figures 3.8, 3.9, 3.10 and 3.11 show the scores given to the categorization of each tweet in our test set by our test subjects. As is evident there is a clear difference in performance between the four. The two HDP variants, HDP-AT in particular performed rather poorly with HDP having an average score of 1.792, and HDP-AT having an average score of 1.178. The LDA variants performed much better, with LDA having an average score of 2.000 and LDA-AT having an average score of 2.610 given by the participants of the experiment.

Topic #32	
WORD	**PROB.**
good	0.0216
love	0.0207
time	0.0205
day	0.0188
today	0.0150
night	0.0124
great	0.0103
work	0.0099
life	0.0093
youre	0.0089

Topic #1	
WORD	**PROB.**
game	0.0649
played	0.0490
team	0.0354
win	0.0345
play	0.0253
football	0.0147
games	0.0145
ball	0.0114
pick	0.0112
points	0.0102

Table 3.1: An example of words in topic #32 and #1 found by LDA-AT

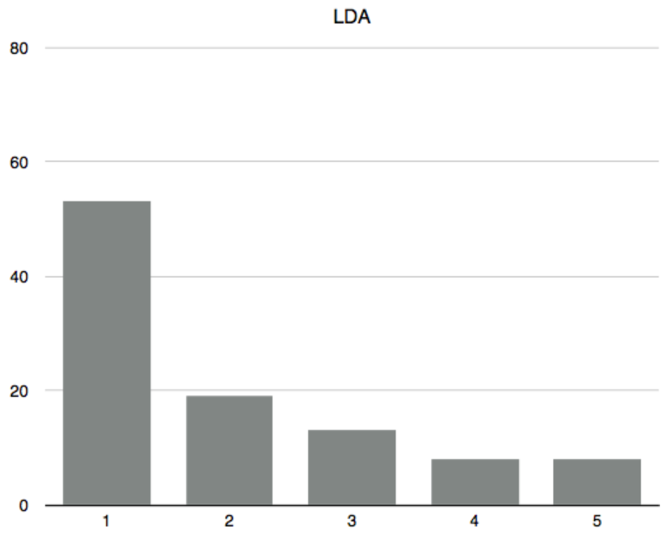


Figure 3.8: Results from modeling 100 tweets using LDA

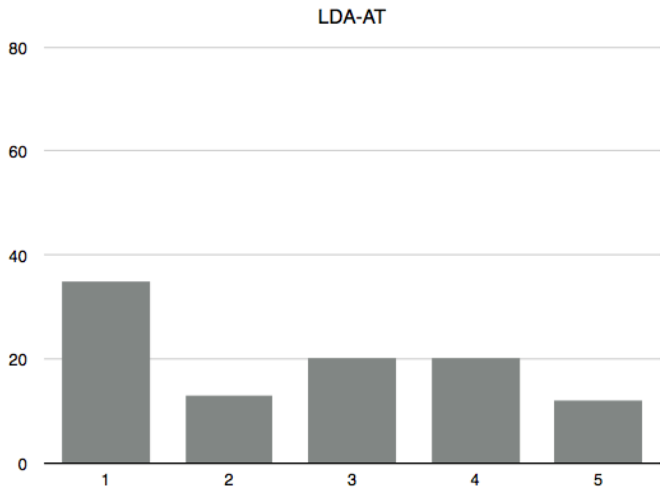


Figure 3.9: Results from modeling 100 tweets using LDA-AT

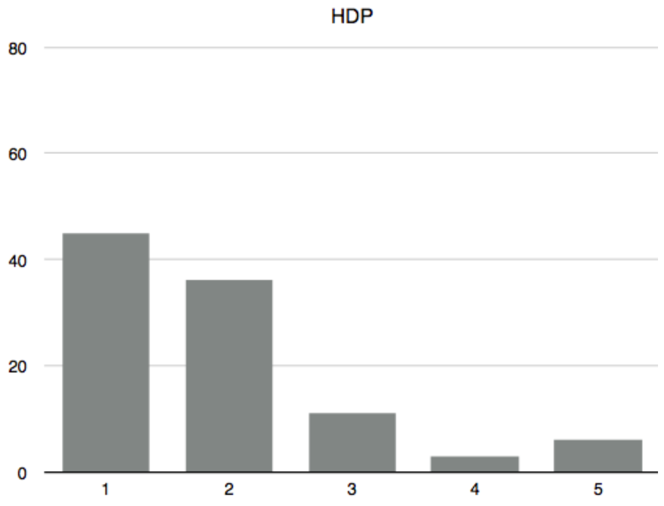


Figure 3.10: Results from modeling 100 tweets using HDP

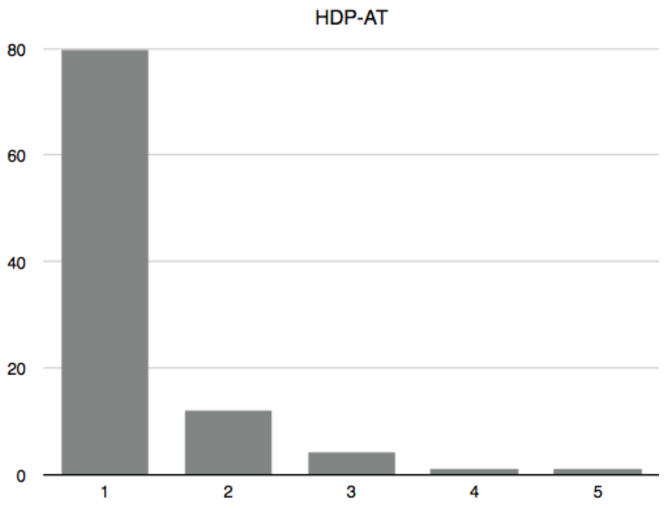


Figure 3.11: Results from modeling 100 tweets using HDP-AT

Method	**Precision**
LDA	0.287
LDA-AT	0.520
HDP	0.059
HDP-AT	0.198

Table 3.2: Precision results given by the different topic modeling methods

To measure the results and give a final score of the different methods, we use precision. We set the threshold for a topic being relevant for a *tweet* at 3. This means every assignment with a score of 3 or higher, gets marked as relevant, while any assignment graded 2 or lower gets marked as not relevant.

$$Precision = \frac{\text{number of relevant assignments}}{\text{total number of assignments}}$$

Using the scores given by the test participants, and the definition above, we calculated the precision of each method. These scores can be found in table 3.2.

As mentioned previously, the sparsity and noisy nature of *tweets* make it difficult to get reasonable data. This is especially true when strip them of stop words. Some *tweets* end up with a very low word count, and as such is not likely to get a suitable topic assigned. Furthermore, *tweets* in foreign languages are a challenge, and is not something we have taken into account; even when ignoring *tweets* containing non-ASCII characters, some languages (such as Spanish) still slip through.

As the results show, LDA-AT performs better than the other three models. The main rationale behind this is that by combining all *tweets* from a single author in the training set into document (meaning the training set then contains one document per author) you are able to somewhat counter the document length limitation inherent to *tweets*. As authors tend to stick to only a handful of topics, this should not skew the model in any meaningful way.

3.6 Topic modeling combined with news index

To be able to filter out what is news on Twitter, we utilized the New York Times annotated corpus, as described earlier. We modeled all the articles published by the New York Times in 2006 using the LDA-AT approach. We chose that approach, as our previous experiment had suggested it had the best performance

Experiment			
#Categories	**Relevant**	**Not relevant**	**Precision**
3 categories	7	247	0.028
2 categories	7	101	0.065
1 category	4	35	0.103

Table 3.3: Test results

I'm on grand jury watch in the Eric Garner case. They are meeting and could decide today. Follow here and @NYTMetro for breaking updates

Figure 3.12: An example of a news tweet found by using the topic modeling method combined with the news index

on *tweets* of the four approaches tested. This gave us a handful of topics that were much more likely to be assigned to actual news articles than others. We then ran some fresh *tweets* through the model, and saved those who were assigned one of the top ranked topics to see if they were actually news items. We did this three times; one with the top 3 topics, one with the top 2, and finally one with only the top topic. The results can be seen in table 3.3.

As is immediately clear, our approach for extracting news does not perform very well. The data set tested consisted of 3,454 tweets, meaning more than 90% of the total tweets were removed. Even so the best precision achieved was merely 0.103.

3.7 Discussion

In this paper, we have looked at different methods for topic modeling *tweets* as part of a breaking news detection system. From our experiments, it seems using an LDA model coupled with aggregating all the *tweets* of a user, called LDA-AT, is the most effective approach. The largest limitation in only using topic modeling for news detection, is that it is sometimes difficult to know what a topic represents. Another pervasive limitation is that of Twitter's 140 character limit on *tweets*.

Using the LDA-AT approach is a challenge when working with streams of real-time *tweets*. In a real-time setting the goal is to model *tweets* as they arrive. As our results show, combining all the *tweets* of a single user into one document is desired when performing this. In a real-time setting, however, one cannot afford to wait for *tweets* for an extended period of time in an effort to make sure each user's document is sufficiently large before updating the model. This is because that would compromise the goal of topic modeling the tweets as soon as they arrive.

One potential solution to this for news detection purposes would be to use a static topic model. This runs the risk of the model getting outdated as popular topics on Twitter drift.

Another potential solution is to incrementally update the model in set time increments so as to not overly delay the topic modeling of the *tweets* themselves. This solution comes with another issue, however. The topic model we have used does not allow for terms to be added to the dictionary after the model has been initialized. This can be alleviated by using a hash map based dictionary, with the caveat that certain terms will share the same index and as such potentially lowers the precision of the model. The alternative is to keep the dictionary static. The danger of doing this is that over time certain terms that are not in the dictionary could become important to identify news.

Finally, we attempted to use the trained topic model in a practical manner to detect news. We used the New York Times annotated corpus to decide which topics were most likely to be assigned to news articles, before using those topics to filter a new data set. The results of this experiment, show that there is the majority of tweets fetched using this method is non news, achieving a precision of only 0.103 in the best case.

This thus seems to be a flawed methodology. It is possible the results could be improved if the input data had been clustered by some method or similar in advance, which we suggest as future work.

Chapter 4

Hybrid Entity Driven News Detection on Twitter

Abstract

In recent years, Twitter has become one of the most popular microblogging services on the Internet. People can share their thoughts as well as comment on anything through Twitter. Tweets can include anything from a person's feelings, advertisements, to newsworthy events. Thus they have become a promising source for detecting breaking news at an early stage. This paper utilizes already existing research on locality-sensitive hashing (LSH) and named-entity recognition (NER) on tweets in an attempt to detect news events from real-time tweets. We come up with different adjustments to parameters in the detection to facilitate news detection, which show promising results.

4.1 Introduction

Twitter[1] is a popular social media platform where users can share their opinions and publish facts about everything from news to more personal matters. For example in an emergency situation, people can provide information about the situation as observers, or give relevant knowledge about the situation obtained by other sources and then share it through Twitter [Vieweg et al., 2010]. On

[1]https://twitter.com/

one hand, this gives Twitter great potential in being able to discover breaking news and follow all angles surrounding a novelty. On the other hand, it is easy to spread a rumor or unreliable facts using Twitter.

Focusing on news, there exist a plethora of different ways to convey the news depending on your location, your political standing, and so on. Therefore there are several factors that may influence how a tweet is formulated, and how it is interpreted. As Twitter is a fast growing social media platform with 302 million monthly active users and around 500 million tweets sent per day[2], it seems possible to discover information on not yet published news around the world there. Previous research, (Sakaki et al. [2010]; Phuvipadawat and Murata [2010]; Hu et al. [2012]) shows that it is possible to discover news on Twitter. It has, however, proven to be difficult to detect these news at early stages, namely before they are put on the newswire. The difficulty lies in the shortness of text in tweets as a user only can express themselves using 140 characters, which provokes the use of abbreviations, incorrect sentence structure, and language.

This paper seek to explore how data pre-processing, named-entity recognition and locality-sensitive hashing (LSH) can be used to achieve better results when attempting to detect breaking news. We will base our system for clustering tweets with similar content on the work of Petrović et al. [2010]. Their approach enables us to detect the topics that are being discussed on Twitter in a given window of time without using any Twitter-specific services, such as Trending Topics[3]. Our base assumption is that if an event occurs that can be considered breaking news, that will lead to a "burst of activity" of people tweeting about it as described for general document streams in Kleinberg [2003]. This will lead to a new, high activity, cluster appearing that contain tweets about the event. Keeping this in mind, we want to explore how the various parameters in the LSH implementation can be changed to increase the efficacy of our approach.

Moreover, we seek a way to do this while working with a live feed of real-time tweets. The main motivation for this is that finding breaking news is of little value if they are detected too long after the news event in question happened, as this makes the likelihood of it already having been picked up by the newswire all but certain.

In this paper, we will make the following contributions:

- Have a look at how parameters in the LSH clustering algorithm can help optimize the algorithm for news detection
- Evaluate how a combination of these parameters could be useful in detecting

[2]https://about.twitter.com/company
[3]https://twitter.com/trendingtopics

potential breaking news on Twitter

- Cluster tweets based on their contents using our chosen locality-sensitive hashing (LSH) algorithm implementation, and see how it performs on random tweets.
- Evaluate if named-entity recognition (NER) can be used as a filtering method to detect news

4.2 Related work

Locke [2009] found that using traditional NER systems on tweets were insufficient. Thus they developed and trained a classifier based on annotated tweets to recognize named entities. The classifier handled the three types *Location*, *Organization* and *Person*. Later Liu et al. [2011] performed research using these same three classes, in addition to a *Product* class. Their approach used k-nearest neighbors as a supervised method.

Not focusing on Twitter, but rather the general problem of NER, Chiticariu et al. [2010] designed the language NERL that is a high-level language to customize, understand, and build rule-based named entity annotators across different domains.

Li et al. [2012] presents a novel 2-step unsupervised method for automatic discovery of emerging named entities, which could potentially be linked to news events such as crises. Their system, called TwiNER, leverages global context that are obtained from Wikipedia [4] and the Web N-Gram corpus, and ranks segments that are potentially true named entities. The system currently does not categorize the detected named entities. However, the approach only works with targeted Twitter streams, which means it is not usable with our approach as the tweets come from a variety of users.

Ritter et al. [2011] addresses the issue of the noisy nature and incorrectness of language in tweets by re-building the NLP-pipeline consisting of part-of-speech tagging (POS-tagging), chunking, and finally named-entity recognition. Their study showed an increase in accuracy and obtained a large (41%) reduction in error compared with the Stanford POS tagger. They also found that the features generated by POS tagging and chunking gives a benefit to the segmenting of named entities. We utilize the system they created, though only the part that classifies named entities, and only as an additional filtering step in our aim to detect breaking news.

[4]http://wikipedia.com

Petrović et al. [2010] presented a locality sensitive hashing (LSH) algorithm for "First story detection" on Twitter. The LSH was able to overcome some of the limitations of the traditional approaches, e.g. centroids and vectors in term space weighted with idf. Their method drastically reduce the number of comparisons a tweet will need to have to find the N nearest neighbors, which is crucial in a system that should work as a real-time service. Their work shows that their system can find news events after analyzing an entire corpus. We utilize their approach in our system, but focus more on activity in tweet clusters in smaller time windows, to detect if something out of the ordinary is happening. This way, we should be able to detect news events in real-time.

Agarwal [2013] present an approach that takes a continuous stream of Twitter data, processes them to filter out tweets characterized as "noise" and get the informative ones. After that they use these filtered tweets to detect and predict trending topics at an early stage. We use some of their findings when pre-processing to disqualify tweets unlikely to be about a news event.

4.3 Methodology

In the following section, we will introduce the different methods used to conduct the experiments in section 4.4.

4.3.1 Locality sensitive hashing

Locality sensitive hashing (LSH) Indyk and Motwani [1998] is a technique for finding the nearest neighbor document in vector space utilizing vectors of random values and representing hyperplanes to generate hashes. This approach reduces the time and space complexity when finding the nearest neighbor.

LSH hashes the input items using k hyperplanes. Increasing the value of k decreases the probability of collision between non-similar items, while at the same time decreasing the probability of collision between nearest neighbors. To alleviate this, the implementation contains L different hash tables, each with independently chosen random hyperplanes. These hash tables are also called "buckets". This increases the probability of the item colliding with its nearest neighbor in at least one of the L hash tables. In other words there is a high probability that the item's nearest neighbor is contained in one of the buckets the item gets assigned to. LSH thus seek to achieve a maximum probability for collision on similar items. To be able to perform LSH on tweets, each tweet is converted to vector space, using tf-idf combined with Euclidean normalization.

There are several parameters that can be adjusted to change the results of LSH. This paper seek to find optimal values for these parameters with respect to detecting news, and filtering out as many of the non-relevant tweets as possible.

Our approach for LSH is based on the one described by Petrović et al. [2010]. We use the cosine similarity between document vectors $\vec{u}$ and $\vec{v}$ to decide the nearest neighbor:

$$cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{||\vec{u}|| \cdot ||\vec{v}||}$$

Specifics of the implementation is elaborated on in section 4.4.2.

4.3.2 Entropy

Once a cluster of tweets have been made, we can measure its Shannon entropy [Shannon, 1948], which also is called information entropy. This is done by concatenating the text of all the tweets of a cluster into a single document. By doing this, the Shannon entropy is given by

$$H(X) = -\sum_i P(x_i) \log_2 P(x_i)$$

where $P(X_i) = \frac{n_i}{N}$ is a tokens's probability of occurring in the document (found by dividing the number of times the token, n_i, appears in the document with how many tokens are in the document, N, in total).

Shannon entropy is a measure of how much information is contained within a document. Petrović et al. [2010] suggests that by ignoring clusters with low entropy, we filter out many clusters that are filled with spam as they tend to have very low entropy.

Tweets are limited to 140 characters, and as Shannon entropy effectively use the length of the text as a factor (in N), longer tweets will be given a higher entropy than shorter tweets by default; if a cluster contains five completely equal tweets of 50 characters each, and another cluster contains five completely equal tweets of 100 characters each, the latter will always have a higher entropy. For this reason, it would be unwise to set the entropy threshold too high (we could miss interesting stories that are simply too short), neither would setting it too low be wise (many stories containing spam and chatter would not be filtered out). The parameter governing the lowest entropy allowed in a cluster is *MIN_ENTROPY*.

4.3.3 Named-entity recognition (NER)

In addition to using LSH to detect breaking news, we want to examine if named-entity recognition can improve the results we get from using LSH to filter tweets.

Named-entity recognition (NER) is a type of natural language processing (NLP). NLP refers to research that explore how computers can be utilized to understand, interpret, and manipulate natural languages, such as text and speech [Chowdhury, 2003]. NER is a powerful tool for analyzing the deeper meaning behind sentences or longer sequences of words [Liu et al., 2011]. Thus NER is commonly understood as the task of identifying so-called named-entities in a text, such as organizations, products, locations, and persons.

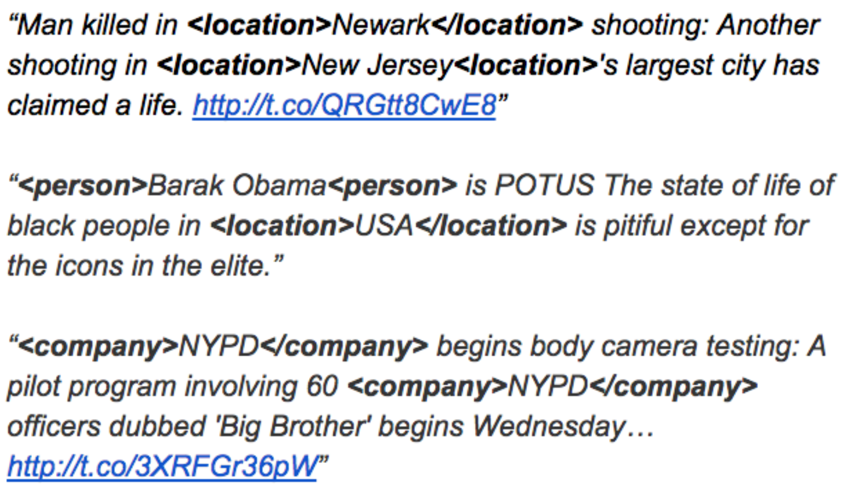
A few different methods have been proposed for performing NER on text documents: The rule-based method, the use of machine learning, and a hybrid between the two. The rule-based method has two extensive drawbacks; it lacks both portability and robustness. For this reason, machine learning has emerged as the better choice for coping with the problem [Zhou and Su, 2002], and is the method we have decided to utilize for our research.

4.3.4 Online processing of data

Because we are interested in breaking news, we have to set up our system in such a way that it is quick enough to get fed data from the Twitter streaming API without running behind. We examine the clusters in set windows of time that has a length of *WINDOW_TIME_IN_SECONDS* seconds. We take the difference between the timestamp of the tweet currently being processed and the last tweet processed before the previous output. If this difference is greater than the duration of our window of time, we output information about the clusters matching our set parameters. As keeping every tweet previously processed in memory would quickly exhaust available resources, we only keep the message of the original tweet for a cluster in memory in addition to the tweets added to it during the current window. A downside of this is that it also limits the amount of information available to decide whether a given cluster is news relevant or not.

4.4 Experiment

In this section, we present the experiment we have devised. We start by describing the data set, before we test the methods described in the previous section.


*"Man killed in **<location>**Newark**</location>** shooting: Another shooting in **<location>**New Jersey**<location>**'s largest city has claimed a life. http://t.co/QRGtt8CwE8"*

*"**<person>**Barak Obama**<person>** is POTUS The state of life of black people in **<location>**USA**</location>** is pitiful except for the icons in the elite."*

*"**<company>**NYPD**</company>** begins body camera testing: A pilot program involving 60 **<company>**NYPD**</company>** officers dubbed 'Big Brother' begins Wednesday… http://t.co/3XRFGr36pW"*

Figure 4.1: Example of tweets where the NER module has recognized location, person, and company entities.

4.4.1 Data sets

Our data set consist of 72,000 tweets collected from the San Fransisco area using the Twitter streaming API[5]. The data was collected over a period of 30 hours from May 11 to May 12, 2015. The API provides extensive metadata for the tweets, most of which are not of interest to us in this experiment. We thus, to conserve storage space, strip most of it away, only keeping the tweet text itself, the tweet id, the user id of the poster, and the timestamp the tweet was posted. As previously mentioned, our system need to work for tweets arriving in real-time. Despite this, we still chose to have a static data set for the experiments so that the same data would be used in each test.

4.4.2 Tools

Before hashing, the tweets were run through the NER engine. This put an extra field into the underlying JSON structure of the tweets where entities were detected called "entities". This field contained a list of all the entities detected in the tweets.

[5]http://dev.twitter.com

To perform the experiment, we require an implementation of the LSH algorithm. We have based our implementation on the ones outlined in Petrović et al. [2010] and Vogiatzis [2012]. It consists of two modules. The first one is the main LSH module and takes in a stream of tweets and outputs a stream of tweets augmented with information about their nearest neighbor (the id of the nearest neighbor, and its cosine similarity to that neighbor). The second module takes in a stream of tweets augmented with information about their nearest neighbor and outputs clusters of related tweets based on parameters discussed below.

We use the following static parameters in our LSH implementation:

- 13 bit hash values (k)
- 36 hash tables (L)
- 20 collisions per hash value
- 2000 previous tweets comparison

These values remain unchanged between experiments. In addition to these, we have parameters we adjust between experiments to find ideal values. These parameters, along with their initial values, can be found in table 4.1.

The first step is tokenizing the tweets by splitting the text into tokens while removing punctuation. Additionally, tokens identified URLs, mentions or hashtags are not included in the list of tokens. If the tweet contains any non-ASCII tokens, the entire tweet is discarded. The entire tweet is also discarded if any of its tokens are included in the *IGNORE* list, or if the number of tokens left after tokenizing is < *MIN_TOKENS*. After tokenizing, the nearest neighbor and the cosine similarity to this neighbor is found. Once this is done, the results are output and used in the next module responsible for creating clusters of related stories.

The next module transforms the stream output from the LSH module into clusters of tweets based on a few parameters. If the cosine similarity between a tweet and its nearest neighbor is at least *MIN_COSSIM_FOR_EXISTING*, it is added to the same cluster as its nearest neighbor. If not, a new cluster is created with the tweet as its first message. After *WINDOW_SIZE_IN_SECONDS* seconds have passed between the timestamp of the first tweet processed and the current tweet being processed, all stories matching certain parameters will be printed. A cluster must have had at least *MIN_TWEETS_IN_SLICE* tweets added to it during the current window to be printed. Additionally at least *MIN_UNIQUES* unique users must have had their tweets added to that cluster for it to be printed. The last parameter to be matched for a cluster to be printed is that its information entropy must be at least *MIN_ENTROPY*.

In addition to finding optimal LSH parameters for detecting breaking news on Twitter, we want to examine if utilizing named-entity recognition can further improve the results. To this end we have used the work of Ritter et al. [2011]. They achieved better results than the baseline presented in their work to customize the named-entity recognition engine to fit the nature of tweets. We have thus elected to use the implementation the authors created, which they have published open source on the Internet[6].

4.4.3 Conducting the experiment

When conducting the experiment, we first needed to establish a baseline. For this baseline, we gave the parameters the values listed in table 4.1 and calculated the precision achieved using those values. We define precision here as the proportion of the returned clusters deemed as news. The values of *MIN_COSSIM_FOR_EXISTING* and *MIN_ENTROPY* in the baseline were chosen based on the findings of Petrović et al. [2010], while the values of *MIN_TOKENS* and *IGNORE* were chosen based on the findings of Agarwal [2013].

The *MIN_TWEETS_IN_SLICE* parameter is initially set to 1 (that is any non-empty cluster may get through the clustering algorithm). The reasoning behind this is that we suspect the value to have an impact on the results that we want to test, but setting it to 1 for the baseline effectively switches it off as to not pollute the results.

Another parameter of interest to the baseline is *MIN_UNIQUES*, which dictates how many unique users must have tweets in a cluster for it be to qualified. If this value is set lower than 2, it opens the proverbial floodgates for various single-user spam accounts, irrelevant or out-of-context tweets and other similar phenomena. We thus elected to set this value to 2 in the baseline and only examine values higher than this. As a news event is highly unlikely to be of interest if only one person is writing about it, with no one retweeting them, we consider this a fair choice to make.

The final tested parameter is *WINDOW_SIZE_IN_SECONDS*, which dictates the length of time between clusters being printed. We were quite unsure of what would be the ideal values for this parameter. A short windows would enable the system to more rapidly detect news events. On the other hand, it is possible that more time has to elapse for the news events to become properly identifiable in the clusters. We thus defined the values to test to be from 5 minutes (300 seconds)

[6]https://github.com/aritter/twitter_nlp

Baseline	
Parameter	**Value**
MIN_TOKENS (MT)	2
IGNORE	[“i”, “im”, “me”, “mine”, “you”, “yours”]
MIN_TWEETS_IN_SLICE (MTIS)	1
MIN_COSSIM_FOR_EXISTING (MCFE)	0.5
MIN_ENTROPY (ME)	3.5
MIN_UNIQUES (MU)	2
WINDOW_SIZE_IN_SECONDS (WSIS)	900s

Table 4.1: Baseline parameters for filtering tweets and the LSH clustering

to 25 minutes (1500 seconds). We set the baseline to be the middle of the range; 15 minutes (900 seconds).

Though the precision values calculated are valuable, another interesting facet is how the number of relevant clusters and non-relevant clusters change between the tests. We want to maximize the number of relevant clusters while minimizing the number of not relevant tweets.

After recording the number of relevant and not relevant clusters in the baseline, we proceeded to change the parameters one at a time, leaving the others at their baseline value. This enabled us to see how changing a parameter changes results. An overview over the various tests we performed, as well as their results, can be found in table 4.2.

Next we used the information from the NER engine to further filter the clusters. This was done by simply dropping those not contain the “entities” field from the output. We did not perform this additional step on all tests; the ones we did perform it for are marked with NER in table 4.2.

Finally, we combined a few of the most promising candidates for news detection in a test. This test is marked FINAL in table 4.2.

4.5 Results

Our results, which can be found in table 4.2, show that the parameters we picked out for our baseline had a fairly average precision of 0.472. By combining various values for the different parameters, in addition to including named-entity recognition, we were able to increase this precision to 0.917.

From the results we can see that there were three things in particular that largely affected the result: the minimum amount of entropy allowed in a cluster (*MIN_ENTROPY* - ME), whether or not named entities were used to filter the cluster (NER), and the minimum cosine similarity between two tweets for them to be put in the same bucket (*MIN_COSSIM_FOR_EXISTING* - MCFE). We had assumed in advance that using NER to filter the clusters would have a positive impact. That this turns out to be the case is thus not surprising. The MCFE parameter is still within the range suggested for it in Petrović et al. [2010] for the optimal values we found. The experiments show that when MCFE is set lower than its suggested values, the precision value plummets. We did not test having this value set higher than their highest suggested value. The reasoning being that by increasing it, you increase the number of clusters and reduce the number of tweets landing in the same clusters. Meanwhile is the ME parameter set higher than the suggested value of the paper. This seems to indicate that news tweets have higher entropy than ordinary tweets, which seem reasonable. Even though the value is set higher, there was not a huge difference in the paper's results between their suggested value and the value we found to give the highest precision for detecting news tweets.

Another parameter for which previous research seemingly has found an optimal value is the one for excluding tweets containing less than a certain number of tokens (*MIN_TOKENS* - MT). Both increasing and decreasing this value led to a slight decrease in precision.

One parameter we thought ahead of time would be important in detecting news is the *MIN_TWEETS_IN_SLICE* (MTIS) parameter. This is the number of tweets assigned to a cluster during the current time window. Although it seemingly has little impact on the detection of breaking news, this could change if tested on a data set that is recorded when some catastrophe or similar crisis happens. This is because the parameters work as sort of a dial; the higher it is set, the more tweets must be generated in a cluster during a window for it to be detected, which indicates the importance of the event the cluster describes.

The parameter for excluding clusters with less than a certain amount of unique users posting to it (*MIN_UNIQUES* - MU) sees little change in precision by increasing it by one from the baseline. Although precision increases slightly, by looking at the raw number one can see that the number of relevant clusters and the number of not relevant clusters decrease almost uniformly. Keeping it low, but still above 1 to filter out an ocean of clusters containing only singular messages, seem like the best course of action.

Finally, we have the parameter governing the size of the time window we examine (*WINDOW_SIZE_IN_SECONDS - WSIS*). This parameter, like MTIS, did not greatly affect the results. Still it demonstrated that unlike what we had assumed in advance, having a shorter time window did not in general impact the precision negatively. As we want to detect breaking news as soon as possible we want this value to be as low as possible.

The test labeled *FINAL* in table 4.2 is thus a test where we have combined the optimal values for ME, MCFE, WSIS in addition to filtering the clusters using NER.

4.6 Discussion and Conclusion

In this paper, we have looked at how one can use an algorithm intended for clustering documents (locality-sensitive hashing) to filter tweets. We have shown that, by tweaking the parameters of LSH and including a named-entity recognition engine in the pipeline, it is possible to achieve a high precision value for news events in tweets. The tuned parameters outperform the baseline established in this paper.

The LSH implementation itself performs well, and its memory footprint only grows linearly with the size of the vocabulary of the input. With the rate of the free Twitter streaming API, the implementation processes the tweets quicker than they arrive, meaning the implementation is usable in a real-time system.

Many of the relevant tweet clusters found by our algorithm were tweets and retweets from the U.S. Geological Survey about small earthquakes occurring in the San Francisco area (where our tweets were gathered from). These, to us, are indeed interesting events, however they would only likely be picked up in areas with seismic activity. Thus, our findings should be tested on data sets gathered from other geographic areas.

Experiments				
Test	**Parameters**	**Relevant**	**Not Relevant**	**Precision**
BASELINE	All at standard (see table 4.1)	199	133	0.472
BASELINE + NER	All at standard	106	49	0.683
MIN_TOKENS #1	MT = 1	117	166	0.413
MIN_TOKENS #2	MT = 3	118	143	0.452
MIN_TWEETS_IN_SLICE #1	MTIS = 2	79	55	0.589
MIN_TWEETS_IN_SLICE #2	MTIS = 3	32	25	0.561
MIN_TWEETS_IN_SLICE #3	MTIS = 4	13	17	0.433
MIN_TWEETS_IN_SLICE #4	MTIS = 5	4	9	0.307
MIN_COSSIM_FOR_EXISTING #1	MCFE = 0.4	137	591	0.188
MIN_COSSIM_FOR_EXISTING #1 + NER	MCFE = 0.4	118	123	0.489
MIN_COSSIM_FOR_EXISTING #2	MCFE = 0.6	86	45	0.656
MIN_COSSIM_FOR_EXISTING #2 + NER	MCFE = 0.6	80	29	0.733
MIN_ENTROPY #1	ME = 3	121	345	0.259
MIN_ENTROPY #1 + NER	ME = 3	28	33	0.259
MIN_ENTROPY #2	ME = 4	53	23	0.697
MIN_ENTROPY #2 + NER	ME = 4	51	11	0.697
MIN_UNIQUES #1	MU = 1	55	56	0.495
MIN_UNIQUES #2	MU = 3	18	31	0.367
WINDOW_SIZE #1	WSIS = 300	107	78	0.578
WINDOW_SIZE #1 + NER	WSIS = 300	98	34	0.742
WINDOW_SIZE #2	WSIS = 600	116	114	0.504
WINDOW_SIZE #3	WSIS = 1200	114	144	0.441
WINDOW_SIZE #4	WSIS = 1500	112	150	0.427
FINAL	MT = 2, MTIS = 1, MCFE = 0.6, ME = 4, MU = 2, WSIS = 300	22	2	0.916

Table 4.2: Test with precision measure for the different tests where FINAL is the best LSH tuning combined with NER

Chapter 5

Find the most representative news tweet

Abstract

Through microblogging services such as Twitter, information about important events in the world can be spread rapidly. One way of detecting such events is to cluster similar tweets. Tweets in these clusters have a varying degree of relevance to the cluster as a whole. This paper proposes a method for extracting the tweet most representative of a cluster of tweets.

5.1 Introduction

In recent years technology has become an increasing part of our daily lives. We share everything that happens, from personal matters to things and events observed. In some cases the things we share might be news events, such as earthquakes, election results and so on.

Social media networks, such as Facebook [1] and Twitter [2], make it easier to share and get updates from other people world wide. Thus social media networks sometimes provide updates about important events world wide in real time. These updates, however, may have incomplete or scattered information. By clustering

[1]http://www.facebook.com
[2]http://www.twitter.com

several of these updates about the same event together, some of them will contain more complete information than others. The remaining problem is then to choose which of these updates is the most representative (i.e. gives the most information) of the event.

This paper seek to explore a way to analyze Twitter posts (tweets) that have been clustered together based on their similarity to find the most representative tweet that describe the event. Assuming the clusters are well formed, each cluster should describe one event. Finding the most representative tweet in the cluster would mean we have found the most representative tweet describing the event.

5.2 Related work

Sharifi et al. [Sharifi et al., 2010b,a] came up with an algorithm that takes a trending phrase, collects posts containing that phrase and creates a summary of the posts related to the phrase. Furthermore, they developed a hybrid tf-idf summarization algorithm where they handle and define a document as one single sentence while also taking the whole collection of posts into account. Thus they differentiate the term frequencies while keeping the idf component.

Nichols et al. [2012] looked at how to summarize events that were detected on Twitter using the word frequency of the longest sentence in each tweet and rank them using a phrase graph. After ranking the sentences, the top n sentences (having no overlapping tokens) are chosen to be included in the summary.

Marcus et al. [2011] developed an application, called TwitInfo, which was able to understand and summarize several weeks of information from sources on Twitter in a matter of minutes.

Inouye and Kalita [2011] evaluated and compared several summarization algorithms to a manually produced summary of events on Twitter. Their result showed that the simple frequency-based summarizers; Hybrid tf-idf [Sharifi et al., 2010b] and SumBasic [Vanderwende et al., 2007] get the best result according to F-measure and human evaluation scores.

Rosa et al. [2011] presented a study on how you automatically can cluster and classify tweets into different categories, and then find the most representative tweet in a given cluster. Their study suggests a straightforward algorithm based on creating centroids in the given clusters by using tf-idf similarity to find the most representative tweet.

Radev et al. [2004] developed a multi-document summarizer called MEAD. They summarized clusters of news articles that were automatically grouped by a topic detection and tracking system (TDT). The system used information from the centroids of the clusters to select relevant documents to the cluster topic.

5.3 Methodology

In the following section we will introduce the method used to conduct the experiment described in section 5.4. As this paper assumes we already have clusters of tweets talking about the same event, methods for obtaining such clusters are not discussed.

A natural extension after clustering similar tweets is to summarize the cluster, or find the most representative tweet of the cluster. Previous research (Sharifi et al. [2010a], Nichols et al. [2012] and Inouye and Kalita [2011]) has focused mainly on creating a summary, rather than finding the most representative tweet(s). This research thus focuses on finding a method to find the tweet that is the most representative of the cluster it is found in.

5.3.1 Algorithm

In order to find the most representative tweet(s) in a given cluster, we suggest a straightforward algorithm, as described in Rosa et al. [2011]. This algorithm is based on the use of centroids. It works by first calculating the tf-idf weights for each term in the cluster. When doing this, all tokens are turned into lower case representations, and any punctuation removed. Additionally, URLs are ignored when calculating these weights. After this a centroid is created based on these weights. This is done by taking the average of the tf-idf weights for each term. Once this is done, the cosine similarity is calculated between the centroid and each tweet in the cluster. The tweet with the highest cosine similarity is returned.

If we instead wanted to find the N most representative tweets in the cluster, only a small change would have to be made. After finding the first representative tweet, the process is simply repeated N times. When doing this, it is important to take care not to add tweets to the set that are too similar to the tweets already there.

5.3.2 Comparison

We want to examine how our algorithm holds up to more naive approaches. We therefore also check how often the newest and oldest tweet in a cluster are the most representative tweets of a cluster. Finally, we check how often the tweet with the most named entities assigned is the most representative.

5.4 Experiment

In this section, we present the experiment we have devised. We start by describing the data sets, before we test the method described in the previous section.

5.4.1 Data set

Our data set consist of 72,000 tweets collected from the San Fransisco area using the Twitter streaming API[3]. The data was collected over a period of 30 hours from May 11 to May 12, 2015. The API provides extensive metadata for the tweets, most of which are not interesting for us for this experiment. We thus, to conserve storage space, strip most of it away, only keeping the tweet text itself, the tweet id, the user id of the poster, and the timestamp the tweet was posted. Each tweet has to run through a named-entity recognizer, and those tweets where named entities were detected has been augmented to contain a field containing them.

This set have been clustered using a locality-sensitive hashing (LSH) algorithm, as the goal of our experiment is to pick out the most representative tweet from a cluster of tweets. The output of this algorithm is a series of popular topics within a given time window.

We have ran the data set through this clustering algorithm twice, with different parameters set. The first one clusters tweets in 15 minute windows. Additionally, the information entropy of a cluster must be at least 4 for the cluster to be valid. There is no minimal amount of tweets added to the cluster during the current window for it to be valid. The second run has a window size of 30 minutes, information entropy of 3 and a minimum amount of tweets added during the window of 5.

These two sets of parameters yield very different results. The first set contain 191 tweets contained in 77 different clusters. On average each cluster contain 2.5 tweets, while the median is 2 tweets per cluster. The tweets are also quite long, with an average of 112.2 and median of 136 characters per tweet, quite close to the maximum length of a tweet. Finally the average and median amount of entities detected per tweet is 1. The clusters in this set, as is apparent, are sparse. This

[3]http://dev.twitter.com

is due to the fairly high entropy threshold set, in addition to allowing clusters with only one tweet in them.

The second set contain 410 tweets divided into 36 clusters. The average number of characters in a tweet in this set is 67, while the median is 60. The clusters contain 11.4 tweets on average, while the median is 6. Each tweet contain on average 0.5 entities, while the median in 0. This set requires there to be at least five tweets in a cluster. The entropy requirement was thus lowered, and the time window of the cluster doubled compared to the other set.

5.4.2 System

The system to find the most representative tweet consist of a simple Python script. This script takes all tweets in a given cluster and turns them into tf-idf term vectors. Once this is done, a centroid is created by assigning each index in the centroid to the average of the value of the index in each term vector. If the index is absent from a term vector, this would be the same as if the value of the index was 0. After constructing the centroid, the cosine similarity between the centroid and all the term vectors are calculated. The term vector with the highest cosine similarity is selected as the most representative tweet for the cluster, and returned.

In addition the script returns the tweet of the cluster with the lowest ID (i.e. the oldest tweet), the tweet with the highest ID (i.e. the newest tweet) and the tweet with the most named entities assigned to it.

5.4.3 Evaluation

In order to evaluate the results of our experiment, we first manually rank the tweets in a cluster for both of our data sets. The ranking had the following possible states:

Not representative The tweet contain little or no discernible relation to the main story of the cluster

Partially representative The tweet is related to the main story of the cluster, but other tweets in the cluster give more information

Representative The tweet is a satisfactory representation of the main story the cluster conveys

For every cluster at least one tweet is marked as representative. The reasoning for this is that our goal is to find the most representative tweet in a cluster of related tweets, and not to filter out uninteresting clusters. The representative tweet is chosen among those tweets that give the most information about the event the cluster depicts.

After labeling, the results from our system are added to the data. Our precision is defined as $P = \frac{H}{N}$. Here H is the number of clusters for which our system's most representative tweet corresponded to a tweet which was manually labeled as representative. N is the total number of clusters in the datasets.

5.5 Results and Conclusion

Table 5.1 show the results of our experiment. From it we can see that the centroid method of finding the most representative tweet performs well on both data sets. The results marked *Experiment 1* used the data set which contained several, small, clusters. The results marked *Experiment 2* on the other hand, are from the data set containing fewer, larger clusters.

All four of the methods fared pretty well in the first experiment. This is largely due to the fact that there were few tweets in the clusters, thus increasing the probability that the newest or oldest tweets are representative of the cluster.

In the second experiment we can see that the centroid approach has roughly the same precision as in the first experiment, while the other approaches have a marked lower precision. This indicates that the centroid approach works far better than these other approaches when the size of the clusters increase.

Not all clusters had tweets containing entities. These clusters get marked as $NONE$ in table 5.1. Therefore the entity test has an overall lower precision than other methods. In a system where named entities had been used for filtering, this value would be higher as the total number of clusters would be lower. In this case, the precision in experiment 1 would be 0.833 while the precision in experiment 2 would be 0.871.

The results of the experiment indicate that using the centroid approach for finding the most representative tweet in a cluster work better than more naive approaches when the size of the clusters increase. Furthermore, most clusters contained tweets so similar that calling any of them more representative than the other would be meaningless. This was expected, as it is an artifact (and indeed the goal) of the clustering itself. The result confirm previous research, i.e. it is quite trivial to select the most representative tweet from a cluster as long as the clustering is performed well.

Experiment 1					
	Total	**Newest**	**Entities**	**Oldest**	**Centroid**
Representative	149	66	54	62	67
Partially representative	37	8	8	14	9
Not representative	5	3	0	1	1
NONE	0	0	15	0	0
Total	191	77	77	77	77
Precision	N/A	0.857	0.701	0.805	0.870
Experiment 2					
	Total	**Newest**	**Entities**	**Oldest**	**Centroid**
Representative	334	23	15	24	31
Partially representative	60	10	3	11	5
Not representative	16	3	0	1	0
NONE	0	0	18	0	0
Total	410	36	36	36	36
Precision	N/A	0.639	0.417	0.667	0.861

Table 5.1: Test results

Chapter 6

Evaluation and Results

In this chapter we present the research results from the papers found in chapters 3, 4 and 5. We also discuss and evaluate these results in order to tie them together, and show how they are a central part of the contributions of the thesis.

6.1 Topic modeling tweets to find breaking news

Our first paper, found in chapter 3, aimed to examine if it was possible to detect breaking news in a real-time stream of tweets by utilizing topic modeling.

The experiments performed during this work found that the topic modeling technique we called LDA-AT had by far the highest precision value out of the four techniques tested. While this means that it would be prudent to use this technique when topic modeling the tweets, it presents some challenges when applied to a stream of tweets.

The LDA-AT technique is based on treating all tweets by the same users as a single document. This is done by concatenating them, forming a single document. This works well in an offline setting, where the entire corpus of tweets is known ahead of time. In such a setting, one can simply loop through the corpus to perform this operation. This, however, is not possible in an online setting as explained in section 2.1.2. To use LDA-AT in an online setting, we had to perform this operation to the tweets as they arrived to our system. The LDA-AT technique only works properly for corpora where there are more tweets than authors. We thus had to collect a number of tweets before processing them, to hopefully have a few users with more than one tweet in the set of tweets processed. We did this by collecting tweets until the difference between the timestamp of the most recently collected and the last tweet to be collected for the previous batch exceeded 900 seconds. Thus roughly 15 minutes of tweets are collected, before being used to update the LDA model. We chose 15 minutes as a sort of tradeoff. On one hand we would like as many tweets as possible to increase the probability of users having more than one tweet in the set. On the other hand we cannot wait too long as that would inevitably delay the detection of any breaking news in the stream.

Another issue we encountered was that the LDA model does not allow resizing the dictionary after initial creation. To circumvent this we utilized a trick where instead of the dictionary providing a one-to-one mapping between an integer id and a word (increasing the size of the dictionary for every new word added), each id is instead given by the hash of the word it represents. Doing this we can initialize a dictionary consisting of a constant number of slots (2^{18} in our case). This means the size of the dictionary is never changed, and we can update the LDA model with fresh tweets. There is a downside to using a hash dictionary, however, as it will cause some words to get the same index. Not doing this would mean that the LDA model would be kept static and, over time, actual topics discussed would drift away from the topics defined in the LDA model.

6.2 Using clustering and filtering to detect news on Twitter

The second paper, found in chapter 4, aimed to find a suitable method for clustering tweets in real-time. It also examined whether or not it was possible to use a clustering algorithm on its own to detect news on Twitter.

The experiments performed, and presented in the paper, found that the use of locality-sensitive hashing (LSH) performed very well for clustering tweets. During experimentation with the values of the LSH parameters, we went from an initial baseline precision value of 0.472 to a respectable value of 0.916 for our data set. This was accomplished by tuning the LSH parameters, and also involving named-entity recognition (NER).

We were surprised to learn that imposing a requirement on the number of tweets in a cluster (the MTIS parameter) did not improve the results in any remarkable way. We still believe this parameter might be useful given a data set of tweets recorded in an actual emergency setting, as the volume of tweets regarding the emergency would presumably balloon.

One drawback of these experiments were the lack of data sets known to contain tweets about news events. As we use Twitter's streaming API to collect tweets ourselves, we rely on big news events actually occurring while we are capturing tweets. Unfortunately (or fortunately for those potential people affected by such a theoretical event) no such events occurred while we were fetching tweets.

Furthermore, many of the tweets deemed relevant in the experiments were earthquake notices by the United States Geological Survey. These notices were deemed news relevant, even though most of the quakes were fairly small. It is possible that this has inflated our positive findings, and we suggest experiments done in areas with less seismic activity to get fewer of these notices.

Finally, even though no big event occurred when we were fetching tweets, there is still a large amount of tweets in the data set. This means we did not have the resources required to manually annotate all of them, which would allow us to calculate the recall value in our experiments.

When compared to the topic modeling approach described in chapter 3, the use of our LSH algorithm yields higher performance when considering precision values. Furthermore, when working in an online setting, it is a far more suitable tool. Not only does the LSH algorithm process tweets as they come in, but it does so in constant time and near constant space. The reason it is not fully space constant is that the random vectors increase with every new addition to the dictionary of the corpus.

The NER part of the system is much slower. Even so the running time increases only linearly in the number of tweets, and using NER still results in a processing time that is lower than the rate of new tweet generation. Because of this, it is possible to use NER in an online setting with the rate of tweet generation we have encountered.

6.3 Selecting the most representative tweets

As part of this thesis we were to find the most representative tweet for a news event. Taking the work we had done with clustering into consideration, we decided to formulate this problem as finding the most representative tweet of a cluster. This task is elaborated on in chapter 5. Our approach utilized centroids and tf-idf weighting of the tweets in the same cluster, where the tweet with the highest cosine similarity is chosen as the most representative tweet. This approach was shown to achieve higher precision (0.861) than the ones it was tested against when the size of the clusters increased.

Our approach for this part might be considered naive, because it is possible to end up with spam and so on as representative tweets. Regardless, we believe the approach is sound in the context of “garbage in, garbage out”. If you supply our algorithm for finding the most representative tweet in a cluster with a cluster of spam then a spam tweet will, in fact, be the most representative tweet of that cluster. The task of filtering out non-relevant clusters must thus come at an earlier stage, such as when clustering.

6.4 Using Twitter as part of a news service

Using Twitter as part of a service that supplies news to its users seem possible, but difficult. Using LSH to create clusters of related tweets and then extracting the most representative tweet from that cluster works, but not all results will be news. These unwanted tweets range from “happy birthdays” to pure spam, both of which are undesired in a news service. One solution to this is tuning the parameters to such an extent that almost no irrelevant messages can get through. This, however, has the unfortunate consequence of eliminating some news items as well. The question is if such news items are interesting enough to the news service or not.

Most previous research into Twitter as a source of news has been done post-mortem by searching for related terms using functionality in Twitter’s API. They also mostly focus on natural disasters, events affecting a large number of people. This, together with our own findings, for us indicate that Twitter is mainly suitable as a source of news for events affecting a large number of people. The problem with this is acquiring an unfiltered data set recorded during the course of such an event to test the hypothesis.

Chapter 7

Discussion and Conclusion

In this chapter, section 7.1 contains a summary of the research and contributions of the thesis. In it we also draw some conclusions. In section 7.2 we discuss some key points for potential future work.

7.1 Contribution summary

The main topic of this thesis was to find a suitable way to detect breaking news on Twitter in an online setting. We combined various artificial intelligence, information retrieval, computational linguistics, and natural language processing methods to accomplish this. The motivation came from people desiring to be apprised quickly about big events occurring in the world. Twitter has proven to be a social media network where breaking news has been published shortly after occurring, and was thus chosen to be the subject of the research.

To answer the first research question of this thesis, we found that it is possible to use topic modeling for detecting breaking news. The extent to which this can be done, however, is more difficult to quantify as keeping a dynamic topic model up-to-date is difficult.

For out second research question, we found that the use of locality-sensitive hashing combined with named-entity recognition can be used to cluster tweets and detect news. We found that, with optimized parameters, we achieve better performance for detecting news than using the topic modeling approach.

Regarding our final research question, we found that using the tf-idf centroid approach for finding the most representative tweet in a cluster yields good performance. Another observation is that the quality of the clustering has little impact on the ability of the approach to find the most representative tweet.

More detailed conclusions of these findings can be found in their respective chapters, 3, 4 and 5. Finally, we have evaluated them in chapter 6, and there also present our opinion on using Twitter data as a part of a news service.

7.2 Future Work

In the following section, we account for potentially interesting areas of future research. These are extensions of the research described in chapters 3 , 4 and 5, and the research questions presented in section 1.3.

7.2.1 Topic modeling

One avenue of potential research is to see whether continuously training a topic model without ever discarding the effects of older documents negatively affect its ability to detect breaking news.

The filtering we used (the New York Times corpus) is a static filter. It is possible that this kind of static filtering impacts the results negatively, and finding out whether or not it does could lead to interesting results.

7.2.2 Named-entity recognition

The NER system used in this thesis mainly labels persons, organizations, and locations and is trained on data written in English. In order to accommodate Norwegian tweets and be able to discover entities specific to Norwegian, it would need to be adjusted or extended. Furthermore, the NER system used in our work is pre-trained and static. One suggestion here would be to research (or find research about) a system that could be trained unsupervised to detect new kinds of entities. The argument for this is that new potential news terms continuously develop as time goes on. Being able to detect such new terms would be very valuable for detecting breaking news at an early stage.

7.2.3 Text pre-processing

The language in tweets often have errors in grammar, spelling, sentence structure or punctuation. One reason for this could be the 140 character length limit of a tweet. As a consequence, a lot of abbreviations are used which pollute the text. This makes it harder for e.g. standard NER engines to detect entities correctly in tweets as opposed to rich text documents, which typically have much fewer language errors. Therefore, a rich pre-processing algorithm which analyses the text and translates it to the correct language would be very valuable, and make the subsequent tasks easier and more correct.

7.2.4 Locality-Sensitive Hashing of tweets

LSH is dependent on having data in memory to be able to hash tweets into the correct buckets in constant time and space. As tweets arrive in a stream in real-time, it is crucial to find a tuning parameter for how large the buckets should be. It should be relatively simple to convert our LSH implementation into a distributed system, which would vastly increase the volume of tweets it can handle.

In this thesis, we have used data sets from New York and San Fransisco instead of data sets from Norway. This was because the volume of tweets we received from the Twitter streaming API was much greater when set for locations in the United States rather than in Norway. Indeed when we set the API to collect tweets for the entirety of Norway, we still received about 100 times as many tweets in the same amount of time when we set the API to collect tweets from the New York City area. This means there is a possibility that the best fits for the parameters we found are intrinsically linked to a certain volume of tweets over specific amounts of time. Some kind of adjustment is thus most likely required when applying the implementation to Twitter streams with less volume, such as that for Norway.

7.2.5 Data set

As Twitter no longer provide data sets for research purposes, in addition to having updated their policy regarding sharing data sets between researchers, it is necessary to collect tweets oneself using Twitters free streaming API [1] or pay for Twitters firehose [2] access. With the free streaming API, you get a sample

[1]https://dev.twitter.com/streaming/overview
[2]https://dev.twitter.com/streaming/firehose

stream of approximately 1% of the messages posted on Twitter. If filtering is set up, to e.g. a location, you ensure that all the tweets received are from this area while still not receiving more than at most 1% of the total messages posted on Twitter [Donkor, 2014]. With the firehose, you get all posts. As we have utilized the free streaming API and collected our own tweets they, unfortunately, have been rather small, with the largest containing approximately 650,000 tweets. It is therefore recommended to gather tweets over a longer period of time, to get the volume of tweets required in order to perform experiments to detect news. That being said, the amount of tweets on Twitter that are actually about news events is quite small, and thus the data sets we have used have been realistic when considering the amount of newsworthy tweets they have had in them.

Appendices

Appendix A

Code

In the following sections we will briefly introduce the code created as a part of this thesis. All code was written in the Python programming language, using external libraries where appropriate (more on this in the relevant sections). The code is modularized, each module having its own responsibilities. The modules work by reading from standard input and writing to standard output, which means they work as a standard *NIX pipeline.

A.1 Fetching tweets using Twitter's streaming API

The first Python module in our system is in many ways the simplest one. The module's main and only task is to connect to Twitter's streaming API and get all the *tweets* it can from a limited geographic area.

Twitter's streaming API

Twitter provides access to a streaming API for all its users. What this means is that anyone with a Twitter account can connect to this API using their access keys to get a live feed of *tweets* that match their criteria. For our purposes, we are interested in gathering as many *tweets* as possible for analysis. The API gives us a few options for this.

The first option is the so called *firehose* access. This access streams the entirety of Twitter updates to the client. This would result in an enormous amount of data, and give complete insight to all *tweets* published in the entire world. The caveat is that it is a restricted API, meaning you have to pay to be able to use it. One provider[1] of access to the *firehose* has their price set to USD 0.10 per 1,000 *tweets*, meaning this access quickly becomes very expensive for our uses.

Another option is the free public statuses sample stream. This stream is free to use, and returns "a small random sample of all public statuses", whereas the *firehose* returns all the public statuses. This form of access returns approximately 1% of the public statuses.

The final option, which we have elected to use, is the public filter access. This access allows you to provide query parameters to the API. The query parameters available are specific users (returns all *tweets* by the specified user), specific keywords (returns *tweets* containing the words in either the tweet itself, the username, URL or hashtags) and lastly specific locations (returns geolocated *tweets* falling within the supplied bounding box(es)). We have decided to use location filtering. The reasoning behind this is that we are mainly interested in breaking news from a geographically limited area. By limiting our bounding box to the New York City area, we improve the 1% figure of overall *tweets* we would get from the public status sample stream, in addition to reducing the number of non-english *tweets*. The downside is we only get *tweets* that have in some way been geolocated, which we have decided is a fair trade-off, since we are interested in this geolocation data as well for our system.

Tweepy

We use the Python library Tweepy[2] to handle our connection to Twitter's streaming API. The way we use it is quite simple. We set up a connection to the API using OAuth tokens set up in Twitter's dev console[3]. After doing this the stream is set up and set to filter on location, in our case a bounding box around New York City. Whenever Tweepy receives a *tweet* from Twitter, a callback function in the *listener* class, which subclasses *StreamListener* (a Tweepy base class) is called. Normally this is the *on_data(self, data)* callback, although the other two callbacks are called on error or on timeout. All callbacks return *True* to keep the connection to the streaming API alive.

[1]http://datasift.com/platform/datasources/twitter/
[2]http://www.tweepy.org/
[3]https://apps.twitter.com/

Data processing

The *on_data(self, data)* function does the brunt of the work in this module. Every time the streaming API serves us a *tweet*, it is sent to this function. The function decodes the JSON encoded content from the API into Python dictionaries. From here we extract the fields we are interested in into a new Python dictionary. After extracting all the fields, the data is written to standard out using JSON encoding.

A.2 Topic modeling code

A.2.1 Training LDA model using live tweets

This module concerns the training of the LDA model used in detecting news in the real-time Twitter feed. The module has two versions: one to train an initial LDA model, and another that updates an already existing LDA model using data coming in from standard input.

The modules utilize the Python library Gensim[4] for generating the LDA model. Gensim does most of the heavy lifting for us, including converting our *tweet* corpus to a vector space model and actually initializing and training the LDA model given initial parameters.

Initial training module

The module to train the model takes one input parameter, namely the path to the file containing *tweets* to parse. To run the module, issue the command:

```
# python topic.py /path/to/data/file
```

Running this command will make the module start lazily reading the file at the path supplied. The module expects a file containing the JSON-encoded *tweet* objects output by the stream module separated by newlines. Every time this lazy reader is asked to return the next *tweet* object it returns a list of all the words (tokens) in the next *tweet*. This is because the Gensim dictionary function expects the documents to be in this format when initializing. The lazy reader generates this list of words by doing the following:

1. Read the next line from the file

[4]https://radimrehurek.com/gensim/

2. Decode the JSON and get the contents of the "tweet" attribute
3. Return empty list if the *tweet* contains non-ASCII characters
4. Make entire message lowercase and iterate over every word
5. If word is a URL, add it to list as-is
6. Remove punctuation and other special characters from message
7. If the word is not in the stop list, and is not a mention (a username prepended with a ‘’ character), add it to the list
8. Return the list

Once the dictionary has been created, the vector space corpus is created by once again lazily looping over the input, using the dictionary created previously to store it as term id/document frequency pairs. The dictionary and corpus is then supplied to Gensim for generation of the LDA model along with the model parameters. After the model is created it is saved to the current working directory as the files "lda", "lda.expElogbeta.npy", "lda.state.sstats.npy" and "lda.state". The real-time training module requires these LDA files to work.

Real-time training module

The module to update the trained model using real-time *tweets* reads JSON encoded data from standard input. This module uses a hash dictionary with 2^{18} elements. As we use a hash dictionary, we risk having some terms collide and share the same hash value. This is, however, a necessary downside to enable us to add new data to the topic model. Since this hash dictionary was not used in our initial experiments, a modified initial training module which also use a hash dictionary must be used to train the initial model. This can be found in the file *topic-hash.py.*

The module is run by issuing the command

```
# python stream-training.py /path/to/lda/file
```

The module stores up *tweets* for a configurable amount of time (adjustable in the SIZE_OF_WINDOW variable) before training the model. A high number makes the model more accurate, but in return makes the model update more rarely. As a default it is set to update every 900 seconds (more precisely when more than 900 seconds have passed between the timestamp of the tweet which previously triggered the training and the timestamp of the current one). Once a sufficient amount of time has passed, we first concatenate all the *tweets* made by the same user into one document (meaning we end up with the same number of documents and authors in the corpus). This is due to the findings we describe in section 3.7, and is the reason waiting for more *tweets* makes the model more accurate.

After this we update the dictionary and model in the same way they were originally created, before storing them to disk. After training, the number of tweets used for training is output on its own line on standard output, followed by each tweet used for training on their own line on standard output.

A.2.2 Classifying live tweets as news

This module is a fairly short script and outputs those tweets that get assigned one of the topics in a list of topics. This list is created based on the topics assigned to articles from the New York Times annotated corpus and the module that does this is described in section A.2.3.

The module is run by issuing the following command:

```
# python stream-testing.py /path/to/lda/file
```

The module reloads the topic list and LDA model each time it classifies a tweet. As the training module only outputs tweets shortly after having saved the updated LDA model, we exploit this by reading all the tweets output from the training module and handling them all at once. The training module, before writing tweets to standard output, also outputs the number of tweets it is about to write. Thus the classifier module knows how many tweets to wait for before handling them.

As soon as that number of tweets have arrived, the LDA model is loaded and the topic list is read. The tweets are converted into term vectors before using the LDA model to find which topic is the best fit for the tweet. If the probability of the best topic is higher than a set value, and the detected topic is in the list of possible news topics, the tweet is written to standard output (in addition to a file "out.json" in the current working directory).

A.2.3 Finding the most news relevant topics

This module is used to create the list of possible news topics used by the module described in section A.2.2. It can be run by issuing the command

```
# python eval-categories /path/to/nyt/corpus /path/to/lda/model
```

The module decompresses the NYT corpus recursively from the level directed to in the input parameter. If the parameter is given as the root of the structure containing the NYT compressed corpus, the entire corpus is uncompressed. If directed to the level of a single year or month, only the pertinent data is uncompressed.

When compressed, the XML files are traversed, and the article text extracted. This text is then tokenized and the most probable topic found. The module contains a hash map of topics to scores. Once a topic has been found for an article, that topic's score is updated by one point.

After the corpus has been traversed, the map is sorted according to each topic's score, and the five topics with the highest score are written to the topic list file used by the classifier module.

A.3 Code for locality-sensitive hashing

In this section we describe the code that cluster tweets using locality-sensitive hashing (LSH). Section A.3.1 describe the main LSH implementation which finds the closest common neighbor to a tweet, while section A.3.2 describe the code that uses the nearest neighbor information to construct clusters of related tweets.

A.3.1 Locality-sensitive hashing code

This module consists of six python files: **tweet.py**, **tfidf.py**, **buckets.py**, **recenttweets.py**, **utils.py** and **main.py**. We will briefly describe the functionality of each script below, before explaining the overall flow of the program.

tweet.py

The code in this file functions as a data object for tweets. The fields that needs to be supplied on construction is:

- **msg** - the text of the tweet
- **timestamp** - the timestamp for when the tweet was posted
- **msgid** - the unique identifier of the tweet
- **uid** - the unique identifier of the author of the tweet

There are also two fields that are not set on construction, but get set later. The **tokens** field gets set the first time the **getTokens** method is called, and the **vector** field gets set as soon as the tf-idf vector for the tweet has been calculated.

The only method of note in this file is the **getTokens** method. This method tokenizes the text of the tweet this instance of the tweet object represents, and returns a list of tokens. It does this by first converting the text to all lowercase letters, and then splitting it up into a list of words on whitespaces. Any token added to the list must consist of only ASCII characters, and must not be any of a URL (starts with 'http'), hashtag (starts with '#') or mention (starts with a '@'). After this check, any punctuation is removed from the token. If any characters remain in the token, it is added to the token list. If the method has not been called before (which results in the tokens field being set), the tokens field of the object instance is set and returned.

tfidf.py

This is a class used to calculate the tf-idf vector of a tweet. It keeps various maps in memory for this use. As the system is made to be used in a real-time setting, the idf values get continuously updated. This means they will not be very accurate in the beginning, but over time this will even out.

First the tokens are fetched. This token list is compared to a supplied map of misspelled to correctly spelled words, swapping where necessary (for example the token "4get" would be switched to "forget" before proceeding). After this step, the tf map for the tweet is updated. If this is the first time for this tweet the current token is encountered, the map of the number of documents the current token is present in is updated.

After creating the tf map, the tf-idf vector of the tweet is created while updating the idf map. The method returns the number of new tokens encountered in the corpus while processing the current tweet. This number is used later to increase the size of the random vectors used for hashing.

buckets.py

This file contains two classes: Bucket and BucketsDB. The Bucket class is an object which encapsulates a hash table. It has methods for insertion and retrieval.

The BucketsDB class is an object which encapsulates all the different buckets and takes care of hashing. Each of the L buckets has its own associated set of k random vectors. The random vectors are variable in size, increasing linearly with the amount of unique tokens (words) encountered in the corpus. The **updateRndVec** method increases the size of each random vector by *size*, which is the parameter supplied to the method.

The **getPossibleNeighbors** method is where the hashing happens. The hash is found by taking the dot product of the tweet's vector and each of the bucket's k random vector. If the dot product is non negative, the corresponding bit is set to 1, while it is set to 0 if the result is negative. The resulting hash is then used to fetch the contents of that hash from the current bucket, before the tweet is inserted in that hash.

recenttweets.py

This file is simply a container for the most recent *size* tweets to be processed by the system. The **getClosestNeighbor** method traverses this list, computing the cosine similarity between the supplied tweet and each of the tweets in the list, returning the tweet with the highest cosine similarity.

utils.py

This file contains several utility methods. The **isAscii** method returns *True* if the supplied string only contains ASCII characters and *False* otherwise. The **qualified** method returns *True* for strings that are ASCII, has more than *mintok* tokens and does not contain any of the tokens in *ignorelist*. If any of these checks fail, it returns *False*.

The **getEuclidNorm** method computes the Euclidean norm of a supplied vector, while the **normalizeVector** uses that norm to normalize a supplied vector.

closestCossim takes a tweet and a list of tweets, and returns the tweet from that list which has the highest cosine similarity with the supplied tweet.

The last method in this file is the **computeCosineSim** method, which takes two vectors and computes the cosine similarity between them. If the Euclidean norm of any of these vectors is 0, the cosine similarity between the vectors defaults to 0 as well.

main.py

This is the file actually ran when using the LSH system. It is executed by issuing the command:

```
# python main.py oov
```

Where *oov* is the path to the "out of vocabulary" file which contain mappings from misspelled words, to the words used when calculating the tf-idf weights.

The program reads JSON encoded tweets from standard input. The format is expected to be the same as the one output from the fetching module described in section A.1.

When the program reads a line from standard input, it first decodes it to a Python dictionary using the json module. After this it checks if tweet is qualified for use in the LSH, by calling the **qualified** method of the utils.py file. If a tweet is qualified, the program proceeds to create a Tweet object, before calculating the tf-idf vector of the tweet by calling **TfIdf.getVals**. Once this is done, the return value of the previous call is used to increase the size of the random vectors in the buckets.

The next step is to find the closest neighbors from the buckets. This is done by fetching all collisions from the buckets, followed by aggregating duplicates. What we mean by this is the same tweet could be returned from multiple buckets. By aggregating, all the tweets are represented once, but augmented with the amount of times it was returned. The list of collisions are then sorted on the amount of times a tweet was returned and the highest cosine similarity between the tweet and the top $3L$ collisions are found by using the utility method **closestCossim**.

If the closest collision in the buckets had a lower similarity than 0.5, the RECENT_TWEETS most recent tweets are also checked for cosine similarity. Once the closest neighbor for a tweet has been found, it is json encoded and printed to standard output. The tweet printed, in addition to data it had before, now also has fields noting its cosine similarity to its closest neighbor and the id of the tweet to which it had this cosine similarity.

A.3.2 Clustering code

This module consists of two files **entr.py** and **get-stories.py**. The module reads tweets output from the LSH module, described in section A.3.1. Its output is clusters of similar tweets occurring within a set amount of time.

entr.py

This is a utility script that contains a method to get the tokens of a tweet in addition to a method for calculating the Shannon entropy (information entropy) of a cluster of tweets. The method for getting the tokens of a tweet is identical to the one in the main LSH program.

When calculating the Shannon entropy, we concatenate all tweets into one document before calculating the entropy using the formula

$$H(X) = -\sum_{i} P(x_i) \log_2 P(x_i)$$

Where $P(X_i)$ is a word's probability of occurring in the text (found by dividing the number of times the word appears in the text with how many words are in the text in total).

get-stories.py

This is the main script of the program. It reads each tweet from standard input, then decodes the JSON they arrive in into a Python dictionary representation. After this it creates a new story with the tweet as the first post if the cosine similarity between the tweet and its nearest neighbor is less than *MIN_COSSIM_FOR_EXISTING*. If it is not, it is inserted into the same story as its nearest neighbor. After this, if the delta between the timestamp of the tweet being processed and the variable *starttime* is more than *WINDOW_SIZE_IN_SECONDS* seconds, the current stories are printed to standard output, before updating *starttime* with the timestamp of the current tweet. The program then proceed to read the next tweet from standard input.

When creating a new story, a few tracking variables are set up along with it. These are the number of tweets added to the story within the current window, the total number of tweets added to the story, the number of unique users who have posted tweets to the story and the number of windows in a row a story has had less than *MIN_TWEETS_IN_SLICE* new tweets added to it. Inserting into an existing story is likewise unremarkable: these tracking variables are updated, and the tweet is added to the list of tweets belonging to the story.

When printing a story, several checks are performed. Stories with too few new tweets added to them have their counter for inactivity increased. If a story has adequate new tweets added to it, however, this counter is reset to 0. Next the information entropy of a story is checked. If this entropy is at least *MIN_ENTROPY* and at least *MIN_UNIQUES* unique users have tweets assigned to this story, the story is added to the output. The output is tagged with a start and end timestamp, signifying which window it is from. Each window generates at most one output. Lastly we do cleanup, which includes emptying the tweet lists of the stories (so that only new tweets during a window is added to the output). In addition, any stories that have gone for at least *MAX_EMPTY_RUNS* output cycles without enough new tweets added to them are voided. If a story is voided, and a tweet later has one of the tweets assigned to that story as its nearest neighbor, a new story is created using the same id as the old one.

A.4 Frontend for named-entity recognition

This script is a frontend for the Twitter NLP[5] system. It reads tweets from standard input, augments them with entities, and writes them back to standard output. What we mean by augmenting here, is that if a tweet is found to contain entities, they are added to the tweet object for later retrieval before being written to standard output. It can be run by issuing the command:

```
# python augment-tweets.py
```

The script converts the tweets into the input required by Twitter NLP, meaning the text of one tweet per line with words not encodable with UTF-8 removed. After collecting WINDOW_SIZE seconds of tweets, it feeds those tweets to Twitter NLP. The results from there is inserted to the tweet objects who has identified named-entities.

The Twitter NLP tool requires some local configuration. Namely the path to the working directory of the tool must be possible to read from the environmental variable TWITTER_NLP. This frontend sets that variable in the NER_DIR variable. The path in this variable has to be changed to the appropriate location of the Twitter NLP installation. If using a relative here, the frontend can only be successfully ran from the directory for which the relative path resolves to the correct installation directory.

A.5 Deciding the most representative tweet in a cluster

This last script reads the clustered tweets output by the tool described in section A.3.2 from standard input. It then writes to standard output the most representative tweet from each cluster using four different approaches. It is called by issuing the command:

[5] `https://github.com/aritter/twitter_nlp`

```
# python get-representative.py
```

For each cluster it creates a tf-idf vector for each of the tweets. After this, it creates a centroid from these vectors by taking the mean value of each dimension from the tweet vectors. After this, it uses cosine similarity to find which tweet is most similar to this centroid. In other words, the most average tweet is found. Additionally the script also outputs the oldest and newest tweets in the cluster as well as the tweet with the most named entities detected. If none of the tweets in the cluster contains named entities, this field is *null*.

Appendix B

Tweet data object

```
{
"timestamp": "1431351193314",
"tweet": "Woke up and threw up \ud83d\ude30 not
   even sick?",
"hashtags": [],
"coordinates": null,
"user": {
    "name": "Justin Lopez",
    "statusCount": 37354,
    "followers": 3103,
    "following": 1288,
    "id": 327166726,
    "screenName": "RFlopez148"
    },
"mentions": [],
"id": 597756345979273216,
"retweetedUsingUI": false
}
```

Bibliography

Agarwal, P. (2013). *Prediction of Trends in Online Social Netwok*. PhD thesis, Indian Institute of Technology New Delhi.

Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J. (2002). Models and issues in data stream systems. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–16. ACM.

Blei, D. M. (2012). Probabilistic topic models. *Communications of the ACM*, 55(4):77–84.

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

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

Chiticariu, L., Krishnamurthy, R., Li, Y., Reiss, F., and Vaithyanathan, S. (2010). Domain adaptation of rule-based annotators for named-entity recognition tasks. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1002–1012, Stroudsburg, PA, USA. Association for Computational Linguistics.

Chowdhury, G. G. (2003). Natural language processing. *Annual review of information science and technology*, 37(1):51–89.

Donkor, B. (2014). Twitter apis vs twitter firehose: Why the difference matters.

Hong, L. and Davison, B. D. (2010). Empirical study of topic modeling in twitter. In *Proceedings of the First Workshop on Social Media Analytics*, pages 80–88. ACM.

Hu, M., Liu, S., Wei, F., Wu, Y., Stasko, J., and Ma, K.-L. (2012). Breaking news on twitter. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2751–2754. ACM.

Huang, A. (2008). Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56.

Indyk, P. and Motwani, R. (1998). 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.

Inouye, D. and Kalita, J. K. (2011). Comparing twitter summarization algorithms for multiple post summaries. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pages 298–306. IEEE.

Kleinberg, J. (2003). Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397.

Korenius, T., Laurikkala, J., Järvelin, K., and Juhola, M. (2004). Stemming and lemmatization in the clustering of finnish text documents. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 625–633. ACM.

Kwak, H., Lee, C., Park, H., and Moon, S. (2010). What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. ACM.

Lee, D. L., Chuang, H., and Seamons, K. (1997). Document ranking and the vector-space model. *Software, IEEE*, 14(2):67–75.

Li, C., Weng, J., He, Q., Yao, Y., Datta, A., Sun, A., and Lee, B.-S. (2012). Twiner: Named entity recognition in targeted twitter stream. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 721–730, New York, NY, USA. ACM.

Li, H.-F., Lee, S.-Y., and Shan, M.-K. (2004). An efficient algorithm for mining frequent itemsets over the entire history of data streams. In *Proc. of First International Workshop on Knowledge Discovery in Data Streams.*

Liu, X., Zhang, S., Wei, F., and Zhou, M. (2011). Recognizing named entities in tweets. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT

'11, pages 359–367, Stroudsburg, PA, USA. Association for Computational Linguistics.

Locke, B. W. (2009). Named entity recognition: Adapting to microblogging.

Lovins, J. B. (1968). *Development of a stemming algorithm.* MIT Information Processing Group, Electronic Systems Laboratory.

Marcus, A., Bernstein, M. S., Badar, O., Karger, D. R., Madden, S., and Miller, R. C. (2011). Twitinfo: aggregating and visualizing microblogs for event exploration. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 227–236. ACM.

Mendoza, M., Poblete, B., and Castillo, C. (2010). Twitter under crisis: Can we trust what we rt? In *Proceedings of the first workshop on social media analytics*, pages 71–79. ACM.

Meyer, B., Bryan, K., Santos, Y., and Kim, B. (2011). Twitterreporter: Breaking news detection and visualization through the geo-tagged twitter network. In *CATA*, pages 84–89.

Nichols, J., Mahmud, J., and Drews, C. (2012). Summarizing sporting events using twitter. In *Proceedings of the 2012 ACM international conference on Intelligent User Interfaces*, pages 189–198. ACM.

Papadimitriou, C. H., Tamaki, H., Raghavan, P., and Vempala, S. (1998). Latent semantic indexing: A probabilistic analysis. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 159–168. ACM.

Petrović, S., Osborne, M., and Lavrenko, V. (2010). 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.

Phuvipadawat, S. and Murata, T. (2010). 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.

Radev, D. R., Jing, H., Styś, M., and Tam, D. (2004). Centroid-based summarization of multiple documents. *Information Processing & Management*, 40(6):919–938.

Ritter, A., Clark, S., Mausam, and Etzioni, O. (2011). Named entity recognition in tweets: An experimental study. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, pages 1524–1534, Stroudsburg, PA, USA. Association for Computational Linguistics.

Rosa, K. D., Shah, R., Lin, B., Gershman, A., and Frederking, R. (2011). Topical clustering of tweets. *Proceedings of the ACM SIGIR: SWSM*.

Rosen-Zvi, M., Chemudugunta, C., Griffiths, T., Smyth, P., and Steyvers, M. (2010). Learning author-topic models from text corpora. *ACM Transactions on Information Systems (TOIS)*, 28(1):4.

Rosen-Zvi, M., Griffiths, T., Steyvers, M., and Smyth, P. (2004). The author-topic model for authors and documents. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 487–494. AUAI Press.

Sakaki, T., Okazaki, M., and Matsuo, Y. (2010). Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860. ACM.

Shannon, C. E. (1948). A note on the concept of entropy. *Bell System Tech. J*, 27:379–423.

Sharifi, B., Hutton, M.-A., and Kalita, J. (2010a). Summarizing microblogs automatically. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 685–688. Association for Computational Linguistics.

Sharifi, B., Hutton, M.-A., and Kalita, J. K. (2010b). Experiments in microblog summarization. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 49–56. IEEE.

Sproat, R., Black, A. W., Chen, S., Kumar, S., Ostendorf, M., and Richards, C. (2001). Normalization of non-standard words. *Computer Speech & Language*, 15(3):287–333.

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2006). Hierarchical dirichlet processes. *Journal of the american statistical association*, 101(476).

Titov, I. and McDonald, R. (2008). Modeling online reviews with multi-grain topic models. In *Proceedings of the 17th international conference on World Wide Web*, pages 111–120. ACM.

Vanderwende, L., Suzuki, H., Brockett, C., and Nenkova, A. (2007). Beyond sumbasic: Task-focused summarization with sentence simplification and lexical expansion. *Information Processing & Management*, 43(6):1606–1618.

Vieweg, S., Hughes, A. L., Starbird, K., and Palen, L. (2010). Microblogging during two natural hazards events: What twitter may contribute to situational awareness. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1079–1088, New York, NY, USA. ACM.

Vogiatzis, M. (2012). Using storm for real-time first story detection. Master's thesis, University of Edinburgh, School of Informatics, 11 Crichton Street, Edinburgh, Midlothian EH8 9LE, Great Britain. Retrieved online on May 14 2015 at `https://micvog.files.wordpress.com/2013/06/vogiatzis_storm.pdf`.

Wang, C., Paisley, J. W., and Blei, D. M. (2011). Online variational inference for the hierarchical dirichlet process. In *International Conference on Artificial Intelligence and Statistics*, pages 752–760.

Wang, X., Zhu, F., Jiang, J., and Li, S. (2013). Real time event detection in twitter. In Wang, J., Xiong, H., Ishikawa, Y., Xu, J., and Zhou, J., editors, *Web-Age Information Management*, volume 7923 of *Lecture Notes in Computer Science*, pages 502–513. Springer Berlin Heidelberg.

Webster, J. J. and Kit, C. (1992). Tokenization as the initial phase in nlp. In *Proceedings of the 14th conference on Computational linguistics-Volume 4*, pages 1106–1110. Association for Computational Linguistics.

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

Zhou, G. and Su, J. (2002). Named entity recognition using an hmm-based chunk tagger. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 473–480, Stroudsburg, PA, USA. Association for Computational Linguistics.