Impact Assessment & Prediction of Tweets and Topics

by İnanç Arın

Submitted to the Graduate School of Engineering and
Natural Sciences
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Sabancı University

August, 2017

Impact Assessment & Prediction of Tweets and Topics

APPROVED BY:

Prof. Dr. Yücel Saygın
(Thesis Supervisor) .....

Prof. Dr. Berrin Yanıkoğlu .....

Assoc. Prof. Dr. Hüsnü Yenigün .....

Prof. Dr. Şule Gündüz Öğütücü .....

Asst. Prof. Ali İnan .....

DATE OF APPROVAL: 01/08/2017

To my beloved family...

Acknowledgments

This thesis would not have been possible without valuable support of many people. Firstly, I wish to express my appreciation to my thesis supervisor, Prof. Dr. Yücel Saygın for his endless assistance for years. He always has been quite helpful as an instructor and as a valuable advisor with his patience and knowledge. I am also thankful to members of my thesis defense committee: Prof. Dr. Berrin Yanıkoğlu, Assoc. Prof. Dr. Hüsnü Yenigün, Prof. Dr. Şule Gündüz Öğütücü, and Asst. Prof. Dr. Ali İnan for their presence and feedbacks. Additionally, I am highly indebted to all my other instructors who had contributed to me during my education years. Besides, Prof. Dr. Nihat Kasap who is my second advisor during my thesis stage and Selcen Öztürkcan made an important contribution on the works that we built up together. Special thanks to Mert Kemal Erpam for his great support on some of the parts in my thesis.

Last but not least, I want to express my special appreciation and thanks to my beloved family as they have always supported and encouraged me. I am always proud of being a part of this family.

Impact Assessment & Prediction of Tweets and Topics

İnanç Arın

Computer Science and Engineering

Sabancı University

Ph.D. Thesis, 2017

Thesis Supervisor: Prof. Dr. Yücel Saygın

Thesis Co-supervisor: Prof. Dr. Nihat Kasap

Keywords: *Impact Prediction, Hidden Retweets, Tweet Clustering, Lexical based Clustering, Density Based Clustering, Generalized Suffix Tree, Locality Sensitive Hashing*

Abstract

People tend to spread information and share their ideas in Twitter, while researchers and policy makers would like to understand public opinion and reactions of people in Twitter towards various events. One way to do that is assessing and predicting the *impact* of tweets. In this thesis, we tried to answer three questions: (1) "What does impact of a tweet mean?", (2) "How do we measure the impact of tweets or topics?", and (3) "Can we predict the impact of tweets or topics?". In order to address these questions, we first emphasize the role of *retweets* and their importance in impact assessment. We then show that we can build a model through supervised learning to predict if a tweet will get a high number of retweets. We extracted various features from tweets including content based features through Convolutional Neural Networks (CNN).

In order to have a more accurate impact assessment, we introduced the concept of *hidden retweets*. People tend to re-post tweets by adding some extra comments to the beginning or to the end of original tweet. Also they intentionally or unintentionally post the exact or near exact tweets with other people without explicitly retweeting them. Therefore hidden retweets are quite important for measuring the real impact of tweets. However, it is also computationally expensive to identify and count the number of hidden retweets. We

show that aggregating hidden retweets can be done efficiently through a lexical similarity based clustering algorithm enhanced with a tree structured index and locality sensitive hashing. We adopted a document clustering based approach for discovering the hidden retweets. We developed and evaluated several clustering algorithms with lexical similarity as the distance measure between tweets. Longest Common Subsequence (LCS) is a widely accepted method to calculate the lexical similarity between short text documents such as tweets, but it is also very expensive. Therefore, we utilized an advanced data structure which is Generalized Suffix Tree (GST) based on Longest Common Substring which is an approximation of LCS. We, then developed a density based clustering approach based for tweet clustering and improved its performance by integrating GST and Locality Sensitive Hashing.

Tweetlerin ve Konuların Etkisinin Değerlendirilmesi ve Önceden Tahmin Edilmesi

Bilgisayar Bilimi ve Mühendisliği

Doktora Tezi, 2017

Tez Danışmanı: Prof.Dr. Yücel Saygın

Tez Eş Danışmanı: Prof. Dr. Nihat Kasap

Etki Tahmini, Gizli Retweetler, Tweet Kümeleme, Karakter Bazında Kümeleme, Yoğunluk Bazında Kümeleme, Genelleştirilmiş Son Ek Ağacı, Lokal Duyarlılık Adresleme

Özet

İnsanlar Twitter üzerinde bilgi ve fikir paylaşırlarken, araştırmacılar ve politika belirleyiciler de çeşitli olaylara karşı toplumsal algıyı öğrenmek isterler. Bu amacı gerçekleştirmenin bir yolu da tweetlerin etkisini ölçmektir. Bu tez içerisinde 3 tane araştırma konusunu cevaplamaya çalıştık: (1) "Bir tweetin etkisi nasıl tanımlanır?", (2) "Tweetlerin ve konuların etkisini nasıl ölçeriz?", (3) "Tweetlerin ve konuların etkisini önceden tahmin edebilir miyiz?". Bu sorulara cevap bulabilmek için öncelikle retweetlerin tweet etkisi üzerindeki önemini vurguluyoruz. Sonrasında bir tweetin yüksek sayıda retweet alıp almayacağını tahmin edebilmek için bir öğrenim modeli hazırladık. Bunun dışında kıvrımsal sinir ağlarını kullanarak tweetlerden içerik bazında bazı özellikler de çıkardık. Tweetlerin gerçek etkisini daha doğru bir şekilde ölçebilmek adına "gizli retweetler" kavramını tanımladık. İnsanlar var olan tweetleri yeniden gönderirlerken tweetin başına ya da sonuna bazı yorumlar ekleyebiliyorlar. Bunun dışında bilerek ya da bilmeyerek başka insanlarla tamamen aynı ya da çok benzer tweetleri yazabiliyorlar. Bu yüzden gizli retweetlerin incelenmesi tweetlerin gerçek etkisini ölçmek için son derece önemlidir. Bununla beraber gizli retweetlerin bulunması ve sayılarının tam olarak belirlenmesi çok pahalı bir işlemdir. Ağaç bazlı yapılarla ve lokal duyarlılık adresleme tekniğiyle geliştirdiğimiz

karakter bazlı kümeleme yöntemlerinin bu pahalı işlemi çok etkili bir şekilde tamamlayabildiğini gösterdik. Tweetlerin arasındaki uzaklığı karakter bazlı metriklerle ölçen çeşitli kümeleme yöntemleri geliştirdik ve bunları deneysel olarak değerlendirdik. En uzun ortak altdizi yöntemi tweet gibi kısa metin dokümanları arasındaki benzerliği ölçmek için çok kullanılan bir yöntemdir. Ancak bu yöntem bir o kadar da pahalıdır. Bu sebeple en uzun altdizgi bazlı genelleştirilmiş son ek ağaçlarından faydalandık. Ayrıca yoğunluk bazlı kümeleme algoritması geliştirdik; sonrasında bu algoritmayı genelleştirilmiş son ek ağaçları ve lokal duyarlılık adresleme yöntemini kullanarak bu algoritmayı hızlandırdık.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Twitter was founded in 2006 and its creator Jack Dorsey sent the very first tweet which says *"just setting up my twttr"* on March 21^{th} 2006. After that day, Twitter continues to be used in an incredibly increasing manner. After 3 years, the billionth tweet was sent [91]; and around 6000 tweets are sent every second, which means approximately 500 million tweets in a single day as of 2015 [81, 72]. The number of active users in a month over time is given in Figure 1.1[1].

As it is stated in [40], Twitter has features that are different than other social network platforms such as Facebook. Account types on Twitter can be *public* or *protected*. A user can follow any other user with a public account without any permission needed. This allows users to follow and share all the tweets from these public accounts. If a user wants to follow a protected account, it is only possible with the permission of that account's owner. However, only 11.84% of the Twitter accounts are protected [6] which means that a large portion of the Twitter data is easily accessible and shareable.

Through all this information flow, people are constantly sharing their feelings, opinions, reactions towards events and life in general. Since, Twitter is one of the most important communication and information/opinion sharing tools of this decade, and it can be considered as a reasonable reflection of the society [76]; investigating the impact of tweets

[1]The image was retrieved from `https://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/`

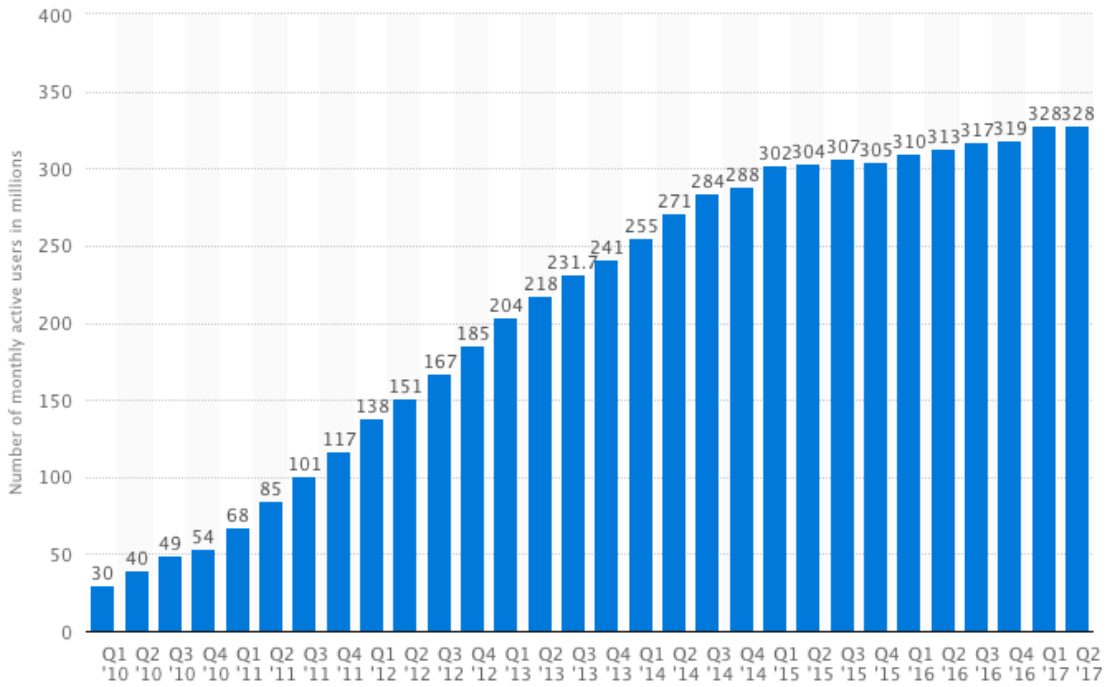


Figure 1.1: The Number of Monthly Active Users in Millions

is an interesting research direction. More specifically, the research questions that we tried to answer in this thesis are: (1) "What does impact of a tweet mean?", (2) "How do we measure the impact of tweets or topics?", and (3) "Can we predict the impact of tweets or topics?". By definition, *impact* means effect and the force exerted by a new idea, concept, technology, or ideology[2]. It is also a synonym for *impression*[3] and *influence*[4]. The nature of Twitter is that people post their tweets to share and spread their ideas, impress and influence other people. Impression is considered as a crucial concept by Twitter and *impression of a tweet* is defined in their formal Activity Dashboard [93] as the number of times people saw this tweet. Any user in Twitter, can reach the impression value of their own tweets by clicking the *View Tweet activity* button in the detail of any individual tweet (see Figure 1.2). However, Twitter does not allow us to see the impressions of the tweets

[2]Retrieved from `http://www.dictionary.com/browse/impact`

[3]Retrieved from `http://www.thesaurus.com/browse/impression?s=t`

[4]Retrieved from `http://www.thesaurus.com/browse/influence?s=t`

that were sent by other users. This restriction directed us to use retweet information to measure the impact of any individual tweet. Below, we investigate two concepts which are *retweet* and *hidden retweets* to express the impact of tweets.

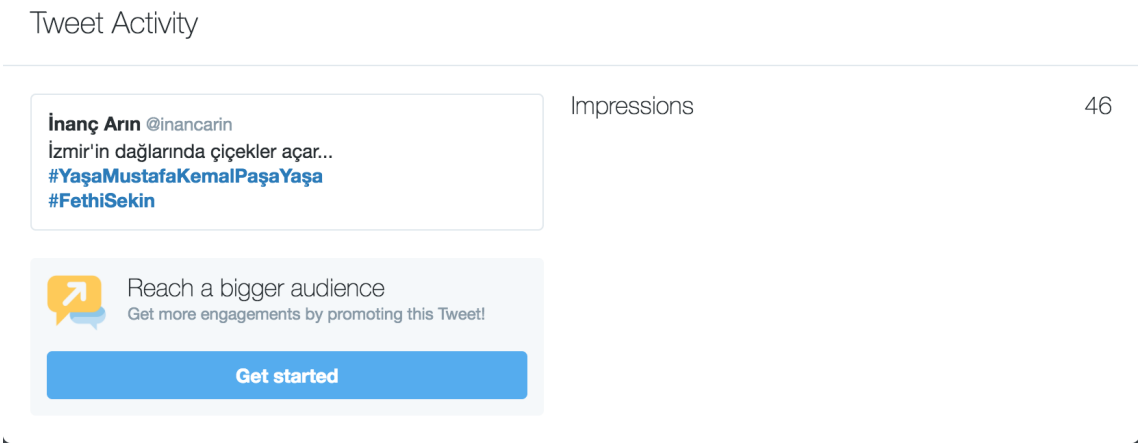


Figure 1.2: Impression of a Tweet in Twitter Activity Page

1.1 Why do people *Retweet*?

Retweet simply means re-posting another tweet [92]. Twitter allows a user to retweet any tweet sent by a public account, even if the user is not following that public account. This allows for a tweet to be instantly shared with the followers of the original sender as well as the followers of the retweeting accounts. In other words, as it is stated in [40], *retweet* option enables users to transmit information far beyond the coverage map of the followers of the original sender. Margarita Noriega who is the Director of Social Media at Newsweek, commented that retweeting is more than a button, it actually is a means of contributing to public knowledge within our social network [53]. She also mentions in [53] about four different reasons why people retweet:

- **Sharing people and specific accounts:** Retweeting certain people or accounts, is as important as the content of the tweet. It is a way of introducing new users or accounts to the community.

Brian Ries, live news editor, Mashable (@moneyries): *"I retweet the best tweets sent by our reporters. I retweet the most notable tweets sent by politicians, celebrities, or other brands . . . sometimes I'll retweet notable users who are sharing our stuff. All of this is meant to signal-boost"* [53].

One other example, when CIA (Central Intelligence Agency) opened an account on Twitter and sent their first tweet, it made one of the biggest impressions on Twitter history with more than 320.000 retweets and 255.890 likes (Figure 1.3). People have retweeted this tweet not only because it has an interesting and thought-provoking content but also it was sent by CIA.

Figure 1.3: The first tweet of CIA

- **Sharing information:** As it was stated before, Twitter is one of the most important communication and information/opinion sharing tools. The primary purpose of Twitter is to provide information flow and retweeting is one of the best ways for enabling users to quickly spread information over their social network.

Elana Zak, social media editor, the Wall Street Journal (@elanazak): *"You'll win points with me if the tweets are well-written and make sense to a reader new to the information. No typos or tons of text-speak. The tweet should either share a piece*

of information or make me want to click on the link to read more. In terms of what @WSJ retweets, it varies greatly depending on what's happening, news-wise" [53].

- **Sharing jokes and humorous contents:** Although people tend to like it when they see a tweet with a joke, they also show their reaction by retweeting them. An example of these tweets, which got 1712 retweets, is given in Figure 1.4.

Figure 1.4: An example of funny tweets

- **Building and engaging in an online community:** One of the reasons of a retweet is conveying the questions/responses to other user, so more people can engage in this topic.

  Samir Mezrahi, senior editor, BuzzFeed (@samir): *"If someone replies to a question I have for others to see their response/the answer to the question"* [53].

Based on our observations, we can also add one more item for retweet reasons which is to criticize, protest or insult an event/opinion. Especially, for political domain, people tend to retweet some tweets from the opposite opinion and they intend to say "Look at that idea, how ridiculous it is". However, in this thesis, we assume that if some user retweets a specific tweet, that means this user wants to accomplish one of the five items above. Whatever the real reason is, according to our assumption this user tends to share and propagate this tweet which is a strong motivation for understanding the impact of a tweet by discovering the number of retweets.

1.1.1 Impact Prediction of Tweets

In the first part of this thesis, we show that we can predict whether a tweet will get high number of retweets with supervised learning techniques. Content based features like *hashtags*, *links*, *special words*, *lowercase/uppercase letters*; and user based features like *number of followers* were utilized to create a learning model. Then, this learning model was experimentally evaluated to make predictions on whether tweets will have high impact or low impact in terms of the number of retweets. For this process, infrastructure of an advanced real-time Twitter monitoring tool had been customized and used. Using this tool, we started to trace some popular and relevant keywords and hashtags from Twitter's Streaming API[5]. Since Syrian conflict has been one of the hot topics, we have analyzed 450K tweets which had been collected between February 1^{st} 2015 and February 27^{th} 2016 with "*Suriye*" (*Syria* in Turkish) keyword. Following this, only text fields of the tweet objects were used to create a learning model with Convolutional Neural Networks (CNN). CNN approach was adopted to predict the impact of tweets on a dataset related to the Soma mining disaster that had a huge impact on Twitter, especially among Turkish users, which had been collected just after the incident between May 12^{th} 2014 and March 23^{rd} 2015 with "*Soma*" keyword.

1.2 Motivation for *Hidden Retweets*

In the second part of this work, we have focused on measuring the impact of tweets more precisely for several reasons. Primarily, we observe that people tend to re-post tweets by adding some extra comments to the beginning or to the end of those tweets. For instance, Adam Saleh (@*omgAdamSaleh*) sends a tweet which protests Delta Airlines (Figure 1.5), and then a user (@*HumanX86*) adds a reaction to this tweet and re-posts it again (Figure 1.6). Note that the second tweet is not a retweet of the first tweet sent by @*omgAdamSaleh* which means that it is not one of the 836235 retweets. This extra comment can be supportive of or against the original tweet. The point is that we are not

[5] `https://dev.twitter.com/streaming/overview`

interested to know whether this user supports that tweet, instead we are interested in the fact that the user tends to talk about that tweet/topic and increase awareness on that topic in some way which could be positive or negative. In other words, our concept of *impact* is independent from the sentiment. Once a user positively or negatively mentions another user's tweet, this user actually contributes to increasing the impact of the mentioned tweet.

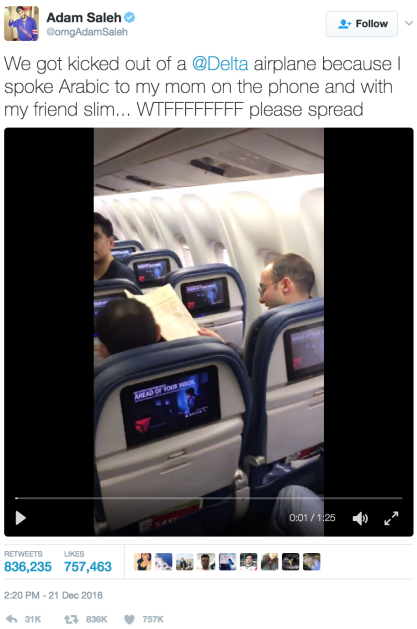


Figure 1.5: @omgAdamSaleh

Figure 1.6: @HumanX86

Some users prefer to copy and paste another tweet instead of retweeting it. Copied tweet does not appear to be the retweet of the original, however it is apparent that the copied tweet was inspired by or influenced from the original tweet. An example has been given in Figure 1.7 and Figure 1.8. An account with name @*girlpost* sends a tweet at 2:25 AM 21 Dec 2016 and says *"I hate it when I drop my makeup accessories"* with attaching a video in the post. Then, another account with name @*glowpost* sends another tweet at 4:08 AM on the same day with exactly same text and video. Although, @*glowpost* did not formally retweet @*girlpost*'s tweet, the fact is that these two accounts are sharing same contents to their social network. Thus, the impact of the original tweet is not 21289 nor 909 which are the number of retweets of these tweets respectively, but the real impact

is the sum of their values which is 22198. Another case is that two different people may use almost or exactly the same sequence of characters while talking about the some topic in their tweets without knowing or influencing each other. Even so, they still mention and want to share the opinion about the same topic. We identify the tweets in the use cases explained above as *hidden retweets*.

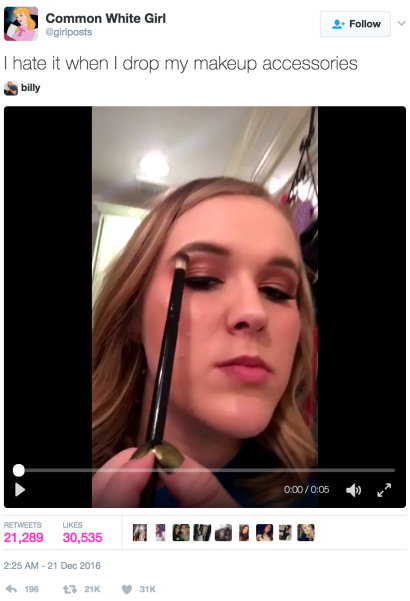


Figure 1.7: Tweet sent by @girlpost

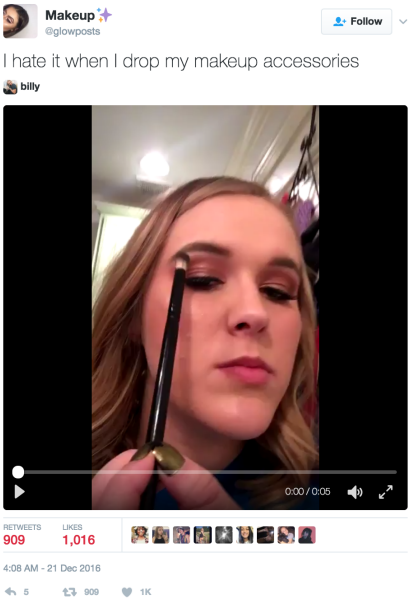


Figure 1.8: Tweet sent by @glowpost

Hidden retweets need to be discovered in order for an accurate impact assessment of a tweet, but how significant are they? In other words, what is the ratio of the number of hidden retweets to the number of retweets for a popular tweet? We observed that in some cases people tend to use the retweet option, thus only a few modified versions of the tweet are spread around. Figure 1.9 provides an example to show how people only retweet and stick with the original tweet (see Figure 1 in Appendix for tweet contents). Hidden retweets we were able to identify compose only 0.067% of all impact for this specific tweet. On the other hand, in some cases people tend to retweet modified versions of the original tweet or to retweet the same content from different sources as in Figure 1.10

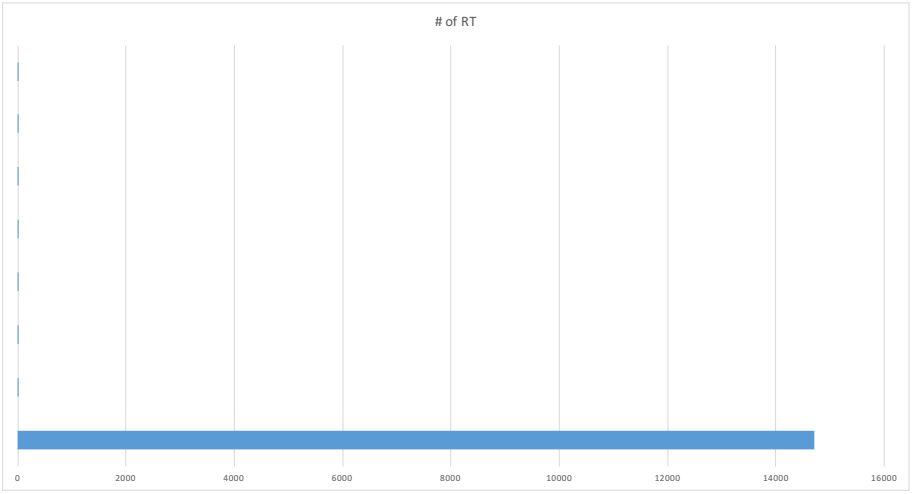


Figure 1.9: Distribution of original and hidden RTs - 1

.

and Figure 1.11 (see Figure 2 and Figure 3 respectively in Appendix for tweet contents). If spreads of these tweets are carefully analyzed, we can see that hidden retweets have significant impacts on the spread of these tweets where hidden retweets compose 73% and 57% of the impacts respectively. These examples demonstrate that hidden retweets may have a crucial role in measuring the real impact of tweets.

1.2.1 Methodology for Discovering Hidden Retweets

As it was mentioned, people tend to re-post tweets by adding some extra comments to the beginning or to the end of the original tweet. Also they intentionally or unintentionally post the exact same or nearly the same tweets as the tweets sent by other people without retweeting them. Therefore hidden retweets are quite important for measuring the real impact of tweets, but it is computationally expensive to discover them from a large collection of tweets. We claim that capturing hidden retweets can be done very efficiently by a lexical similarity based clustering algorithm integrated with generalized suffix tree or locality sensitive hashing method. The identification of hidden retweets is defined as a document clustering problem in this thesis. The reason is that, we try to group similar tweets whose similarity is above a predefined threshold. However, stan-

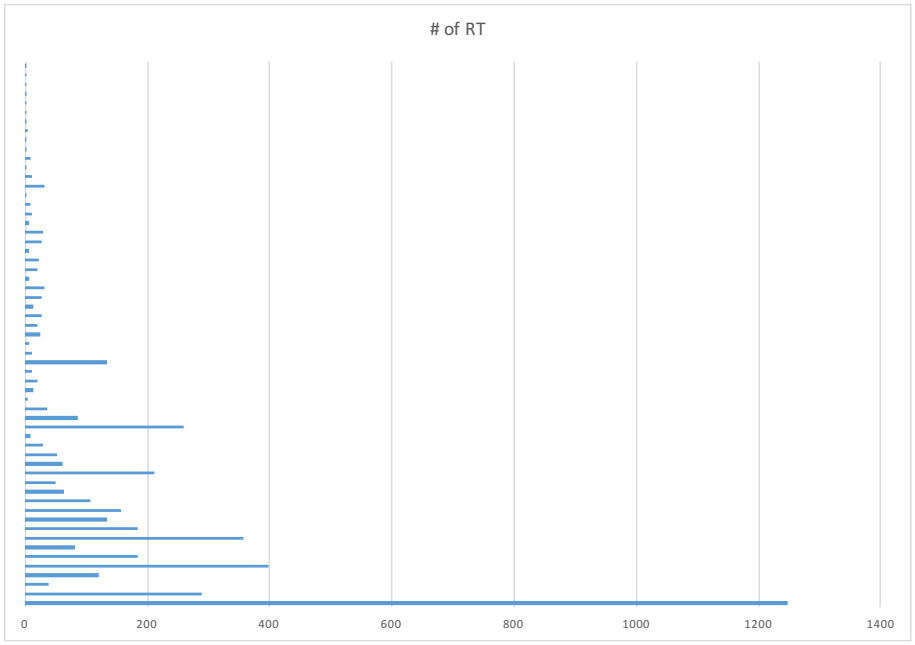


Figure 1.10: Distribution of original and hidden RTs - 2

.

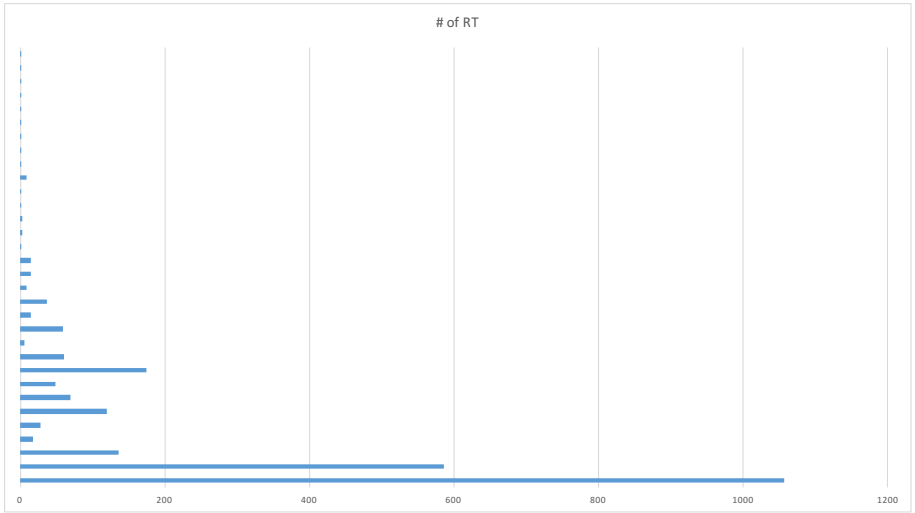


Figure 1.11: Distribution of original and hidden RTs - 3

.

dard document clustering algorithms cannot be directly applied to tweets, because tweets have two distinct characteristics which differentiate them from standard documents such as blogs, news etc. First, tweets are very short due to the nature of Twitter where there is a character limit of 140. Therefore standard document clustering algorithms which use word-based similarity metrics will not work well with tweets. Second, Twitter has no writing format, people can use informal language, emoticons, abbreviations, and there are lots of misspellings in their tweets. As a result, Twitter needs a specific clustering methodology based on lexical clustering to identify similar tweets in terms of content. In order to cluster similar tweets in an efficient way, we developed a lexical clustering algorithm based on Longest Common Subsequence. Furthermore, we implemented different versions of this algorithm with advanced data structures based on Suffix Trees, and Locality Sensitive Hashing. We also adopted a Density Based Clustering approach for efficient order independent clustering of tweets. Proposed methods are evaluated in terms of time and cluster quality performance to show their effectiveness compared to the state of the art.

1.3 Outline

The rest of the thesis is organized as follows. We first discuss the related work in the literature for retweet prediction and short text clustering in chapter 2. Background information about the methodologies used in the thesis are provided in chapter 3. In chapter 4, the "impact" is associated with the concept of "retweet" and we show that we can predict whether a tweet will receive high number of retweets. chapter 5 extends the meaning of "impact" with hidden retweets and presents different methods on how to discover hidden retweets very efficiently.

Chapter 2

Related Work

Different methods for predicting retweet number of the tweets have been studied in recent years. One of the well known works was presented by Zaman et al. [104]. Authors in this work defined *retweets* as a practice to spread information on Twitter network. They trained their data with probabilistic collaborative filtering models and their training data contains some user based and item (tweet) based features. Our study differentiates from this work by presenting a more extensive work (like analyzing accelerating velocity etc), and by using different features to represent a tweet. Petrovic et al. [59] also studied on predicting whether if a tweet will be retweeted and their method was based on Passive-Aggressive algorithm developed by Crammer et al. [13]. They also used different features than ours like *number of times the user was listed*, *is the user verified*, and *is the user's language English* etc. Yang and Counts [103] worked on a network analysis to understand information diffusion in Twitter. They tend to investigate user interactions by finding username mentions in a network. In the following chapters, we also use Convolutional Neural Networks to get maximum information from the content of the tweet. Zhang et al. [107] also used attention-based deep neural networks on tweets, however their purpose was to predict users' attention interests based on their historical tweets.

As we also focused on efficient tweet clustering methods for hidden retweet capture, we also made a literature review on document clustering methods. There is exiting work on clustering documents and analyzing the data collected from social networking plat-

forms. However, most of these works use vector space model to represent textual documents which are then used for similarity calculation. For example, Ma et al. [44] propose a topic based document clustering with three phases where conventional techniques were used in each phase which are LDA, k-means++, and k-means respectively. Jun et al. [34] also proposed a model that converts the text data into vector space model. Their model works on this sparse structured data, reducing the number of dimensions and then performing the clustering task. The clustering method they use is k-means based on support vector clustering and the Silhouette measure. Rangrej et al. [64] converted text documents into vector space format with tf-idf values and then used k-means clustering with cosine and jaccard distances in order to group short text documents. Tu and Ding [88] and Li et al. [43] represent tweets and event segments respectively with tf-idf weights and then used cosine similarity metric to calculate distance between tweets. Tang et al. [84] represent tweets as word vectors but they enrich these vectors with Wikipedia concepts. They focused on tweet representation, which maps each tweet to a space of Wikipedia concepts. Similar to tf-idf values, they count cf-itf (i.e. concept frequency and inverse tweet frequency) to fill vector representations. Becker et al. [5] focused on online identification of real-world events from Twitter and used an incremental clustering algorithm where the number of clusters is not pre-determined. They also represent tweets with tf-idf vectors and use cosine similarity approach. In this work, our aim is to group tweets which are very similar in content with small additions, deletions, and updates. Therefore, we do not convert tweets into vector representations, instead we utilize *longest common subsequence* and *longest common substring* methods to find similarities between tweets.

In the literature, there is some work which assigns documents (or tweets) into set of pre-defined categories. For instance, Miller et al. [50] had two categories: *spam* and *not spam* and assigned each tweet to one of these two categories. Nishida et al. [52] propose a new method for classifying an unseen tweet as being related to an interesting topic or not. Zubiaga et al. [110] categorized tweets into 4 different classes that are news, ongoing events, memes, or commemoratives. Saraçoğlu et al. [70] developed a tool for clustering documents; however, their task is to determine the documents which belong to

more than one class using fuzzy clustering. In our work, we do not have a predefined set of categories.

It is worth mentioning related work on clustering long-text documents like news. Among those, Song et al. [77] developed a hybrid evolutionary computation approach to optimize text clustering. Their approach takes advantage of quantum-behaved particle swarm optimization (QPSO) and genetic algorithm (GA). Their experiments were on 4 subsets of standard Reuter-21578 and 20Newsgroup datasets which are quite different than Twitter data. Zamora et al. [106] also proposed an efficient document clustering method based on locality-sensitive hashing (LHS), but similarly their experiments were only based on formal language, long texts like 20Newsgroup and DOE (Department of Energy) datasets which contain abstracts about energy documents. The methodology used in long-text clustering is different than tweet clustering which has informal language.

There are some studies on clustering in social media platforms. For instance, Dominguez et al. [15] propose a method for clustering geolocated data from Instagram for outlier detection. However, their focus is not textual data. Martinez-Romo and Araujo [47] worked on Twitter text data to detect malicious tweets in trending topics. They split the data into two groups (spam and not spam) as in text categorization, and then predict whether the tweets are spam using statistical language analysis. Cheong and Lee [12] studied patterns in Twitter, however their work is mainly based on clustering users who exhibit some patterns and they only used data sets of size 13K tweets in their experiments.

We propose ST-TWEC for lexical clustering and the underlying data structure of this method is suffix tree as it will be explained later in detail. In literature, there is existing work which use suffix trees for document clustering. The most known suffix tree clustering algorithm is Suffix Tree Clustering (STC) algorithm [105] which uses word-based suffix tree for clustering. It is important to stress out differences between ST-TWEC and STC as most state-of-the-art suffix tree clustering algorithms are based on STC. STC uses a word-based suffix tree to create clusters and then merges clusters based on the overlap of their document sets. To achieve linearity, STC can only merge k clusters with other clusters, hence it returns only top-k clusters. On the other hand, ST-TWEC uses a

character-based suffix tree and achieves linearity for datasets of fixed size documents such as tweets. It is able to return all clusters and it is also able to capture character variations when comparing two tweets.

In Twitter domain, currently there are three papers which use suffix trees for clustering. Thaiprayoon et al. [85] uses STC on Thai Tweets to create clusters and a two-label clustering structure. Similarly, Poomagal et al. [61] uses STC along with semantic similarity to cluster tweets and determine topics of interest. On the other hand, Fang et al. [19] uses suffix tree to detect the common phrases between tweets and uses it as a feature to detect popular events. Although these methods use suffix tree to employ different clustering techniques, the main limitation of these methods is that they return top-k clusters/events, discarding the rest. Atefeh and Khreich [3] compare event detection methods for Twitter. Authors explain both event detection methods in Twitter and in traditional media. One of the event detection methods explained in traditional media uses an n-gram approach for event detection in news and uses suffix tree to speed up the retrieval of n-gram words, however clustering was not considered.

Chapter 3

Preliminaries and Background

Making predictions on tweets and developing adaptive methods for tweet clustering process requires usage of some advanced data structures and algorithms. In this chapter, we will provide some background information regarding these concepts. We will start with explaining our infrastructure for tweet collecting/storing in section 3.1. We believe that it is worth to explain how we retrieve and store the data which will be used in our experiments. Following this, in section 3.2, we will mention about *tf-idf* which is a numerical statistic used in most of the Text Mining applications. *Cross-Validation* technique will be explained in section 3.3 to show how supervised learning methods used in chapter 4 will be evaluated. In chapter 4, we will be using Convolutional Neural Networks for a deep content based analysis, thus we give some background information on Deep Learning methods in section 3.4 to have a better understanding of these concepts. All the algorithms we developed for tweet clustering in chapter 5 are based on lexical similarity, for that reason Longest Common Subsequence and Longest Common Substring methods will be introduced in section 3.5. While traditional clustering algorithms used in this thesis are explained in section 3.6, some advanced data structures and indexing methods to improve the performance of the clustering algorithms are provided in section 3.7.

3.1 Infrastructure - ELK

As the advanced infrastructure to collect/store tweets to be used in experiments, we preferred Elastic's[1] open source products: Elasticsearch, Logstash and Kibana (ELK). Origins of all these three tools come from same company and they can easily be integrated to work together.

3.1.1 Elasticsearch

Elasticsearch is an open-source, largely scalable, distributed, lucene based full-text search engine. It stores documents in JSON format with key-value pairs and it allows us to index and maintain text documents in such a way that the text searching becomes really fast. Relational databases are not suitable for full text searching; it takes more than 10 seconds for a particular query to get the result via SQL while it takes 10 milliseconds to search with Elasticsearch on the same hardware [96]. It was designed for scaling up to thousands of servers with petabytes of data. Elasticsearch works on standard RESTful API (as shown in Figure 3.1); additionally, it provides some other APIs for different programming languages like Java, Python, PHP, Perl, Ruby, C# [16].

3.1.2 Logstash

Logstash is an open-source data processing and transferring tool which transmits data from one source to another. It provides large number of input[2] and output[3] plugins which enables data transfering to/from Elasticsearch, RDMS, csv files, mongodb, solr, tcp, udp events and so on. One of the input plugins is *twitter* plugin which enables reading events from Twitter Streaming API. In order to execute Logstash Twitter Plugin, some parameters in the configuration file should be specified. For our case, these parameters are credentials to be retrieved from Twitter (*consumer_key*, *consumer_secret*, *oauth_token*,

[1] `https://www.elastic.co/`

[2] `https://www.elastic.co/guide/en/logstash/current/input-plugins.html`

[3] `https://www.elastic.co/guide/en/logstash/current/output-plugins.html`

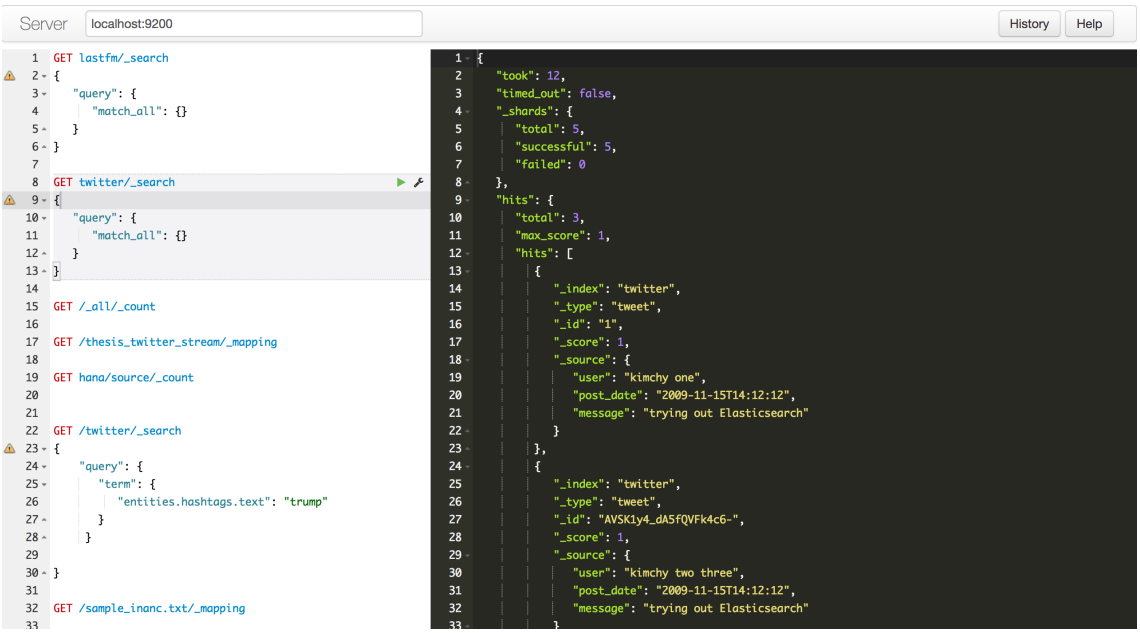


Figure 3.1: Elasticsearch with RESTful API. It is possible to query from the browser with Chrome plugin *Sense*

.

oauth_token_secret), languages (the languages of the tweets to be collected), and keywords (keywords to be tracked).

3.1.3 Kibana

Kibana is an open-source tool to monitor, visualize, analyze and discover the data in Elasticsearch. It allows to plot some histograms, some type of charts and more by taking advantage of aggregation capabilities of Elasticsearch (see Figure 3.2[4]).

These 3 tools (Elasticsearch, Logstash and Kibana) have been used to set up the infrastructure mentioned above. We have prepared a video[5] which is publicly available in YouTube to demonstrate constructing a sample infrastructure to retrieve live streaming tweets from Twitter, import them into Elasticsearch by using Logstash, and visualize them in Kibana.

[4]The figure was retrieved from `https://www.elastic.co/products/kibana`

[5]`https://www.youtube.com/watch?v=J5BX7ECIsjY`

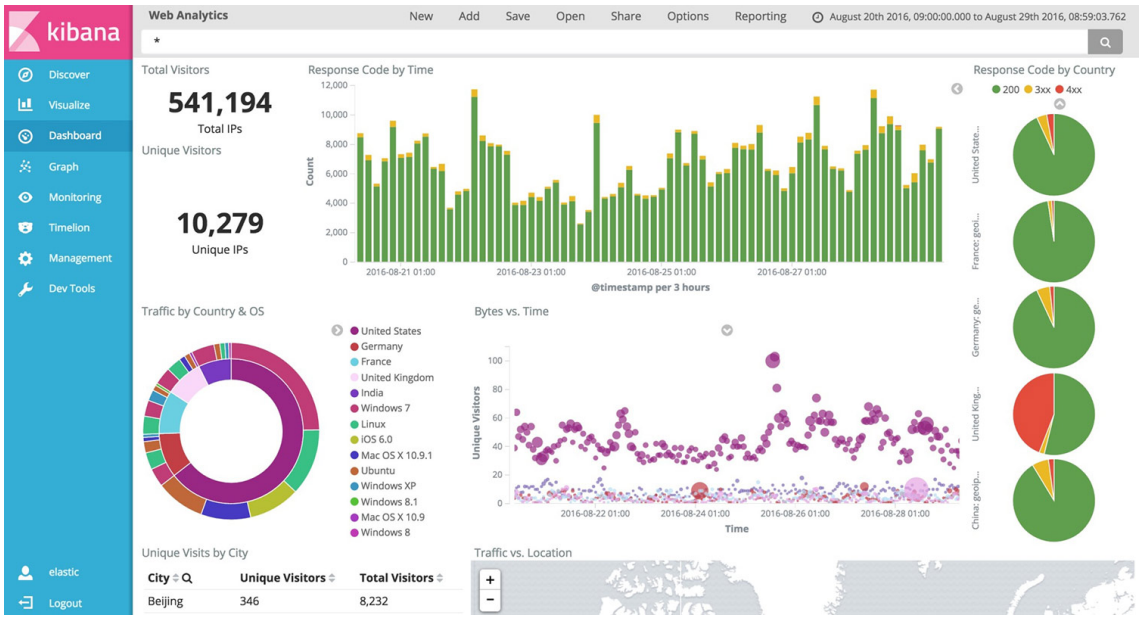


Figure 3.2: Visualizing the data in Elasticsearch

.

3.2 Text Mining

3.2.1 tf-idf

tf–idf, abbreviation of *term frequency–inverse document frequency*, is a statistical method that is intended to reflect how important a word is to a specific class. One particular word becomes more important for a class as it occurs more frequently in this class and as it occurs less frequently in other classes.

- **tf: term frequency,** measures how frequently a term (token) occurs in a specific class (Equation 3.1).

$$tf(t) = \frac{(Number\_of\_times\_term\_t\_appears\_in\_a\_class)}{(Total\_number\_of\_terms\_in\_the\_class)} \tag{3.1}$$

- **idf: inverse document frequency,** measures how important a term is by calculating the number of other classes that contain this term (Equation 3.2). For instance, one particular class c may have "is" so many times as term. However, since this term also occurs in many other classes, this term is not that important for the class c.

$$idf(t) = log\frac{(Total\_number\_of\_classes)}{(Total\_number\_of\_classes\_with\_term\_t)} \tag{3.2}$$

Lastly, *tf-idf* is defined as in Equation 3.3.

$$tf - idf(t) = tf(t) * idf(t) \tag{3.3}$$

3.3 Evaluation Methods for Classification

3.3.1 Cross-Validation

Cross-Validation is a technique which is being used for estimating the accuracy of a classifier induced by supervised learning algorithms [99]. *k-fold cross-validation* randomly splits data into *k* different parts; use $k-1$ of them as training data and 1 of them as testing data. Then, it repeats this process k time by choosing another part as testing. At the end, the average of the results gives the overall accuracy of the learning model.

3.4 Deep Learning Methods

Neural Networks is not a new concept, actually the history of neural networks comes from 1950s when Hebb [25] pointed the strength of neural pathways. It becomes more popular in 1990s after the invention of the back propagation algorithm [98]. However, it lost its attraction in the beginning of the 2000s with the high usage of other techniques like Support Vector Machines(SVM), Random Forests etc. In recent years, the popularity of neural networks increased again (and still increasing exponentially - see Figure 3.3) due to availability of huge amount of data and hardware designed for high computational processes (i.e. GPUs). The reason that we say "deep" is the depth of the learning structures.

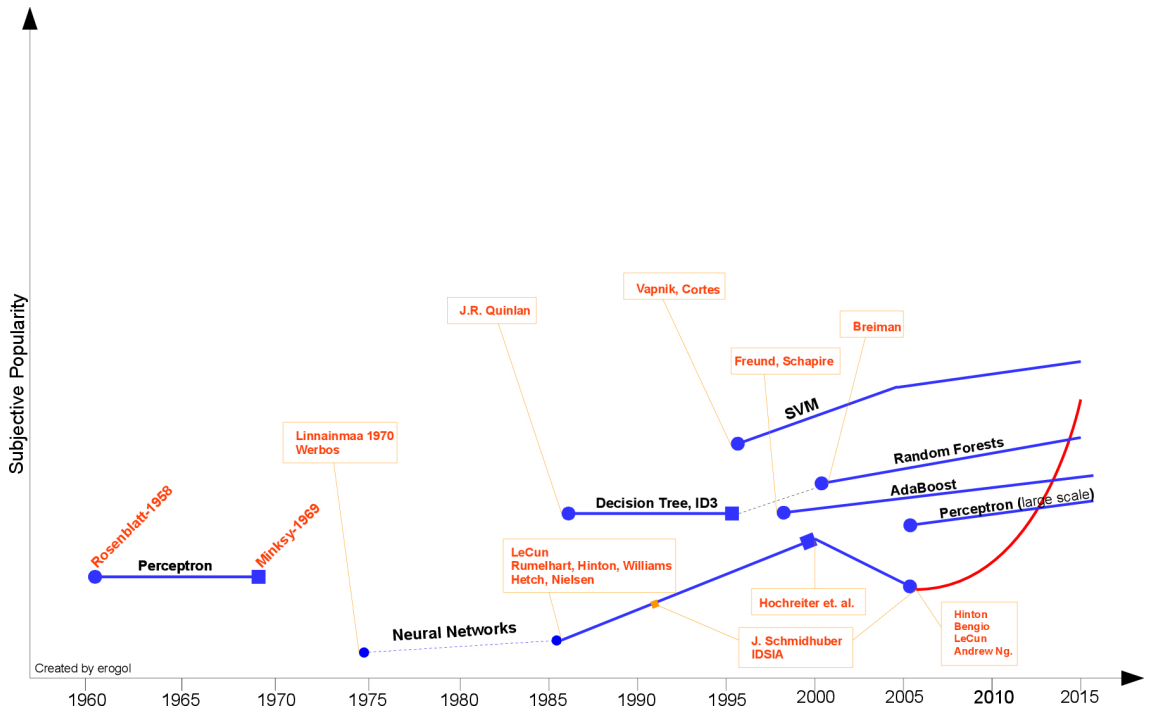


Figure 3.3: History of Neural Networks

3.4.1 Fully Connected Neural Network

A sample Fully Connected Neural Network is given in Figure 3.4[6]. In this neural network, there is an input layer, several hidden layers (they are called as hidden layers since the information transmission between each of them is unknown) and an output layer.

Let's assume that we want to train a logistic classifier (i.e. linear classifier) that is denoted by Equation 3.4

$$WX + b = y \tag{3.4}$$

In Equation 3.4, X refers to input (for example, pixels of an image), W refers to weights, b refers to bias and y refers to a vector that contains scores (logits) for each class. These scores in y are needed to be converted to the probabilities such that the probability of the correct class is close to 1 and the probability of the incorrect classes are

[6]The figure was retrieved from `http://neuralnetworksanddeeplearning.com/chap6.html`

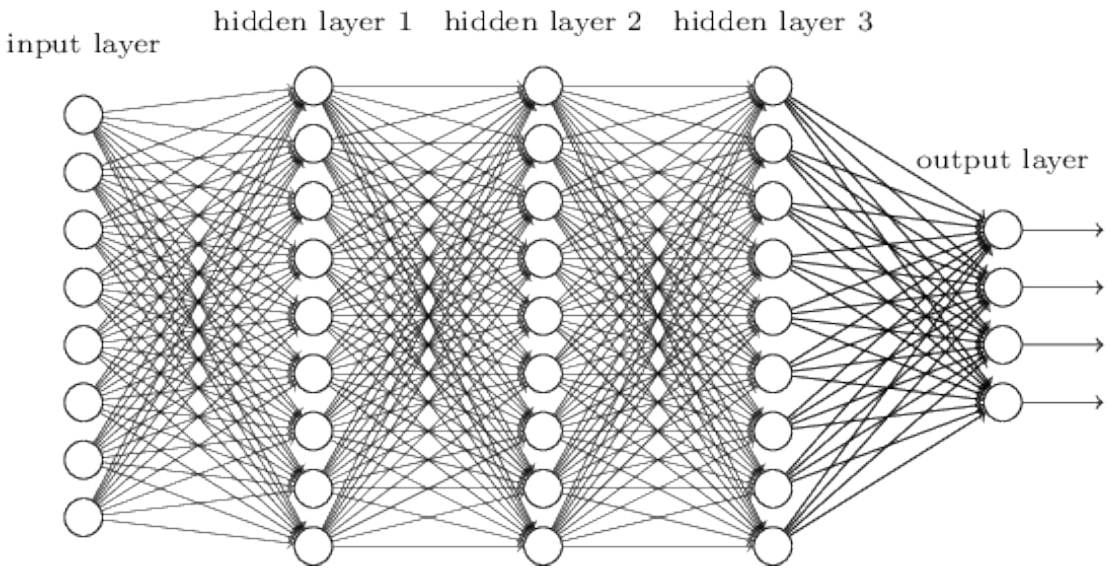


Figure 3.4: Fully Connected Neural Network

close to 0. Actually, this is the expected result, but it is not the case all the time. In order to convert these scores into probabilities, we use "softmax" function which is denoted by Equation 3.5.

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{3.5}$$

For instance, let's say *y* is [2.0, 1.0, 0.1] and we want to convert this vector into vector of probabilities. And each probability will be calculated through softmax function as below:

$$y = \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix} \Rightarrow \textbf{Softmax} \Rightarrow \begin{bmatrix} p = 0.7 \\ p = 0.2 \\ p = 0.1 \end{bmatrix} = S(y)$$

3.4.2 Cross Entropy and Loss Function

The probability vector, $S(y)$, that is created after softmax function will be compared with the "one hot encoding" vector which is denoted by L. In L, the correct class gets the value of 1.0 and all other classes get the value of 0.0. The function that calculates distance between $S(y)$ and L is called "Cross Entropy" and denoted by Equation 3.6.

$$D(S(y), L) = -\sum_i L_i \, log(S_i) \tag{3.6}$$

We have lots of pieces until so far, let's summarize them below:

$$x = \begin{bmatrix} .. \\ .. \\ .. \end{bmatrix} \underset{WX+b}{\rightarrow} y = \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix} \underset{S(y)}{\rightarrow} \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix} \underset{D(S(y),L)}{\rightarrow} \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix} = L$$

At the end, we obtain the following distance function $D(S(WX + b), L)$, where W and b are needed to be found such that the distance function should be very low for correct predictions and high for wrong predictions. For this purpose, we define a "Loss" function (i.e average cross entropy) as in Equation 3.7 where N is the number of examples.

$$\pounds = \frac{1}{N} \sum_i D(S(wx_i + b), L_i) \tag{3.7}$$

3.4.3 Optimizing Loss Function

As Vincent Vanhoucke, who is a Principal Scientist in Google Brain, points out in [97], output of the loss function should be as small as possible. It is an optimization problem, and one of the most widely known algorithms for this problem is "Gradient Descent". For the sake of simplicity, let's assume that we have two weights as in Figure 3.5[7]. In order to get this loss function smaller in each step, we take the derivative of the loss function and follow that derivative in the opposite direction.

Vanhoucke [97] also states that initialization of W and b is quite important, we need to assign random values with zero mean and equal variance. The problem is that we may have lots of parameters (weights) and the number of examples in Equation 3.7 can be quite high. Additionally, we need to repeat this several time. In other words, although gradient descent works great to minimize the loss function, its complexity is quite high. Instead of considering all the examples in our training data, we just pick a random sample and calculate the loss function and its derivative accordingly. Each time (actually many

[7]Figure 3.5 and Figure 3.6 were retrieved from [97]

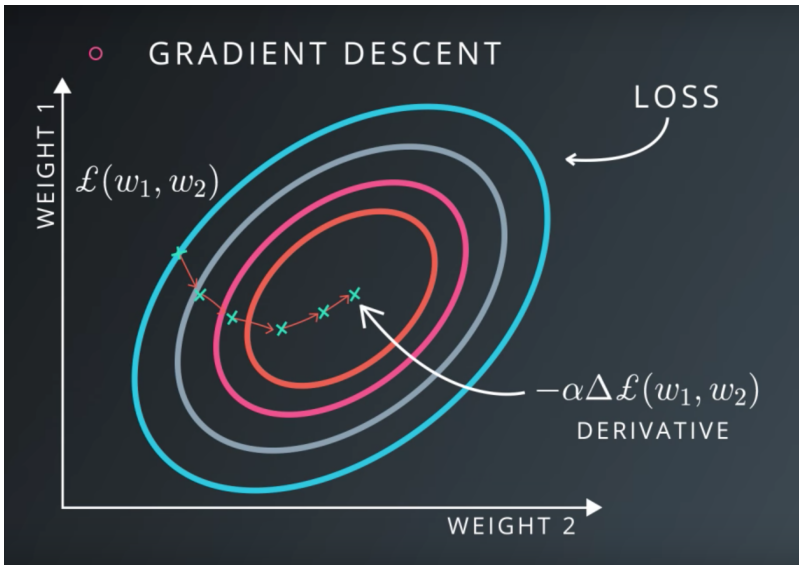


Figure 3.5: Gradient Descent

times) we take a small step instead of a large step (and sometimes it may be in the wrong direction); however, we reach to the intended position in the long term as it is shown in Figure 3.6. This technique is called "Stochastic Gradient Descent" and it is much cheaper. Another stochastic optimization technique that we used in our experiments is called "Adam" (see [38] for details).

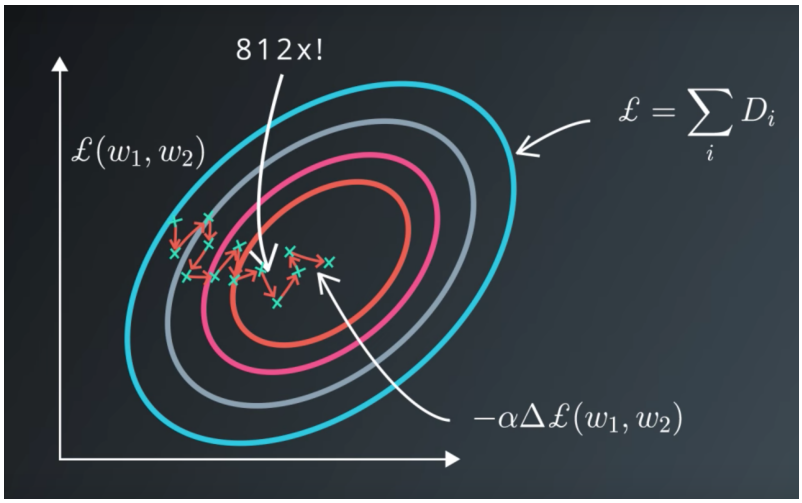


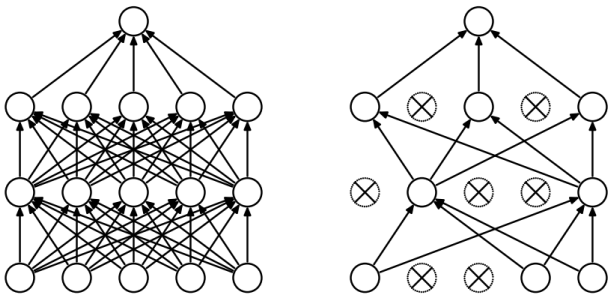
Figure 3.6: Stochastic Gradient Descent

3.4.4 L2 Regularization and Dropout

Overfitting is one of the biggest problems in Deep Neural Networks. There are several methods that can be applied in order to prevent overfitting. One of them is "L2 Regularization". Actually, it adds another value to the loss function to decrease the effect of large weights and generates a new loss function $\pounds'$ as in Equation 3.8 where $\| w \|_2^2$ is L2 norm of weights and β is a small constant.

$$\pounds' = \pounds + \beta \frac{1}{2} \| w \|_2^2 \tag{3.8}$$

Another regularization type is "Dropout". Srivastava et al. [79], who are the inventors of the method, define dropout technique as dropping random units from neural network in order to prevent co-adapting as shown in Figure 3.7[8]. According to the experimental results in [79], it greatly reduces the overfitting and it outperforms other regularization techniques as well. Since units to be dropped are randomly chosen at each step, it basically enforces the neural network model to learn different models of the same data in the long term.



(a) Standard Neural Net (b) After applying dropout.

Figure 3.7: Same Neural Network Model without and with Dropout

3.4.5 Word Embeddings

Mikolov et al. [49] represents words with vectors where these vectors contain some num-

[8]Figure 3.7 was retrieved from [79]

ber of weights. The idea behind the embeddings is that similar words occur in similar context. In other words, the distance (i.e. cosine distance) between vectors of semantically similar words is very low. Embeddings also allow us to apply some mathematical operations among words. Let's represent vector of a specific word *w* with *V(w)*.

$$V' = V(\text{“puppy”}) - V(\text{“dog”}) + V(\text{“cat”})$$

V' is another vector that is very close to V(“kitten”) in embedding space (of course if we have a good model). Therefore, words are represented as vectors and documents are represented as sequence of vectors in deep learning models.

3.4.6 Convolutional Neural Networks

Vanhoucke [97] defines Convolutional Neural Networks (CovNets or CNNs) as the neural networks that share their parameter across space. For instance, let's say we want to determine whether an image contains a cat. It does not matter where the cat exists in the image. The only important thing is its existence in anywhere. Similarly, assume a ”cat” word inside of a sentence. The meaning of this word will not change depending on its position in the sentence. CovNets are widely used both in image classification [39] and text classification [37] processes. For the easy understanding of the concepts related to CovNets, we will first define these concepts within an image classification problem and then show how to use them in a text classification problem. CovNets are composed of 4 main phases:

1) **Convolution:** In this phase, the purpose is to get a deeper feature map that contains semantic information. For this process, we use filters (or also called as patches). Assume that we have a 5x5 input image and 3x3 filter as in Figure 3.8 and Figure 3.9 respectively[9].

   In order to extract the feature map, we stride filter matrix on the input image step by step as shown in Figure 3.10 and Figure 3.11.

[9]Figure 3.8, Figure 3.9, Figure 3.10, and Figure 3.11 were retrieved from `https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/`

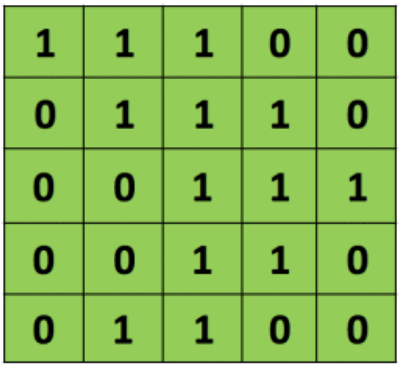


Figure 3.8: Input Image

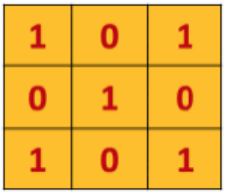


Figure 3.9: filter

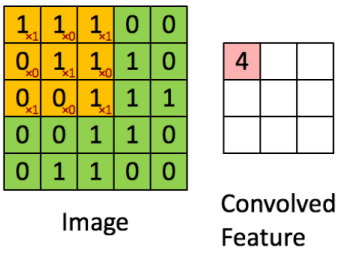


Figure 3.10: Convolution - Step 1

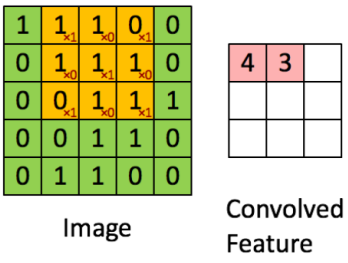


Figure 3.11: Convolution - Step 2

Please note that every input image has a depth. For instance, the depth of an image with RGB channels is 3, and this convolution process is applied on each depth. At the end of the convolution phase, we have another image with different width, height and depth. The height and width of the feature map depends on the size of the filter and the stride length in each step. On the other hand, the depth of the feature size depends on the number of filters. If we use more filter, then it will generate a deeper feature map which has more semantic information. As it is stated before, the purpose of convolution is to generate a deeper image as shown in Figure 3.12[10].

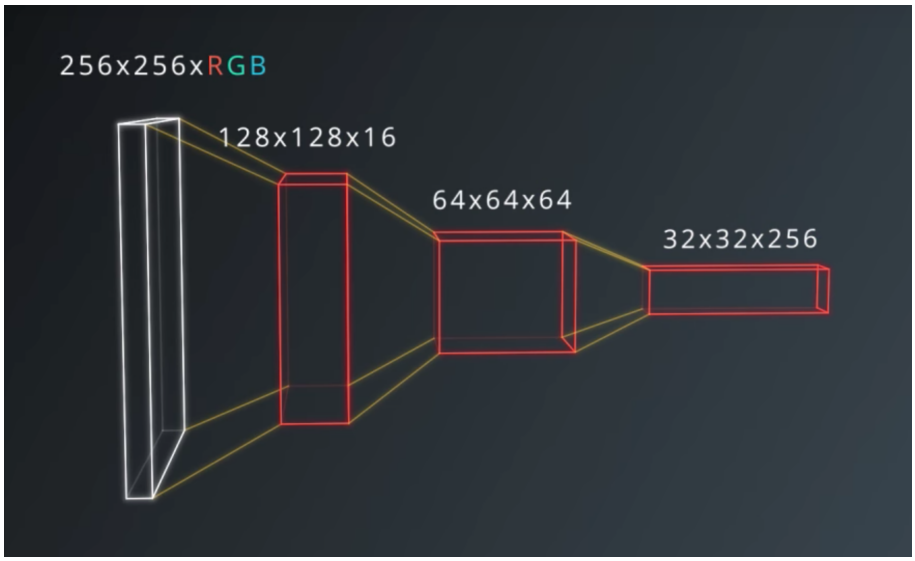


Figure 3.12: A Deep Representation of Consequent Convolution Processes

2) **ReLU:** ReLU is abbreviation of Rectified Linear Unit and it replaces all the negative pixels values with 0. Since convolution is a linear operation and most of the real-world problems are non-linear problems, ReLU contributes by adding non-linearity to the problem as shown in Figure 3.13.

   There are some other non-linear functions like "tanh" or "sigmoid", but ReLUs are the most popular and generally more accurate than others.

3) **Pooling (subsampling):** Remember that we have a stride size parameter in convolu-

[10]Figure 3.12 was retrieved from [97]

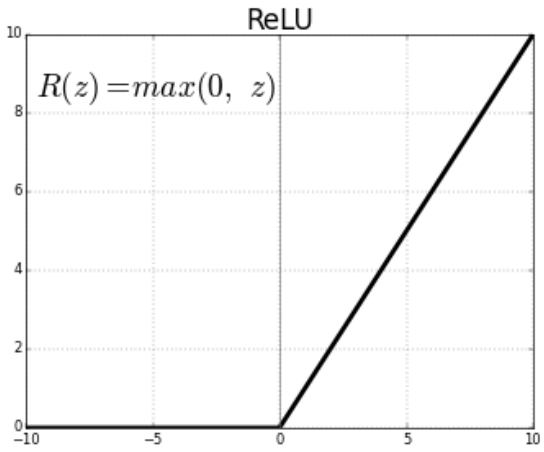


Figure 3.13: Rectified Linear Unit

tion phase while we are extracting the feature map. If we choose stride length too big, then we will lose a lot of information. Instead, it's better to select smaller stride lengths (like 1 or 2), then select the maximum value inside a pooling area as in Figure 3.14[11].

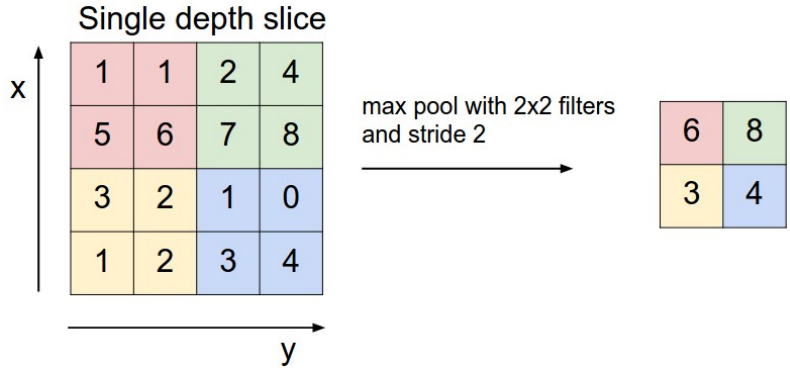


Figure 3.14: Max Pool

This operation reduces the dimension size, so decreases the complexity; but it still stores the most important information. There are other pooling techniques like "average pooling", but max pooling generally performs better.

4) Fully Connected Layer: In the last phase, we have fully connected layer(s) which executes the classification process which was explained in subsection 3.4.1.

[11] Figure 3.14 was retrieved from `http://cs231n.github.io/convolutional-networks/`

We have several steps until so far as below:

Image → *Convolution* → *Max Pooling* → *Convolution* → *Max Pooling* → *Fully Connected Layer* → *Fully Connected Layer* → *Output*

LENET-5 (see Figure 3.15) was the first model that uses the architecture above, presented by Lecun et al. [41].

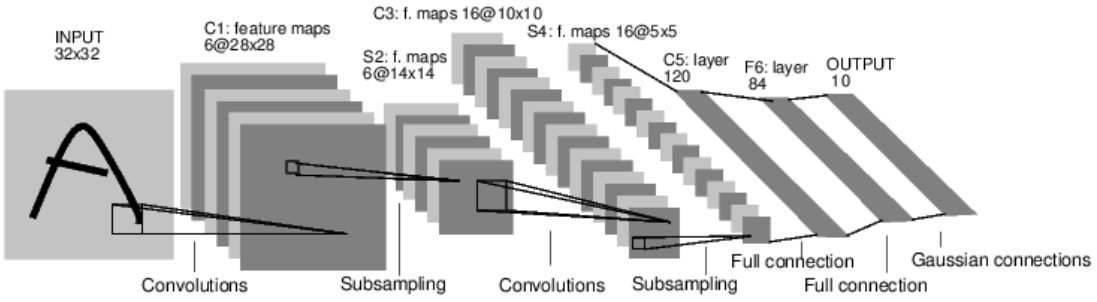


Figure 3.15: LENET-5

3.4.7 Text Classification with CovNets

As we mentioned in subsection 3.4.6, CovNets also works impressively good for text categorization process. Text classification with CovNets is quite similar to image classification problem, the only thing we need to do is that finding an appropriate input format for the convolutions phase. Now, the input features are sentences which are represented with sequence of word embeddings (see subsection 3.4.5). Kim [37] presents an overall architecture for text categorization with CovNets in Figure 3.16.

A sentence is composed of one or more words. Each row in the input matrix represents a word, or it is more meaningful to say that each row represents a vector that corresponds to the related word (i.e. word embedding). One other point is that each sentence in the corpus should be represented with the same size. It is clear that number of columns in the input matrix is the length of the embeddings which is the same constant for each word, so this is not a problem. But, what about the number of rows? The number of rows indicates the total number of words in the sentence that is different for each of them. For that reason, we represent sentences with pad sequences. In order to do that we define a

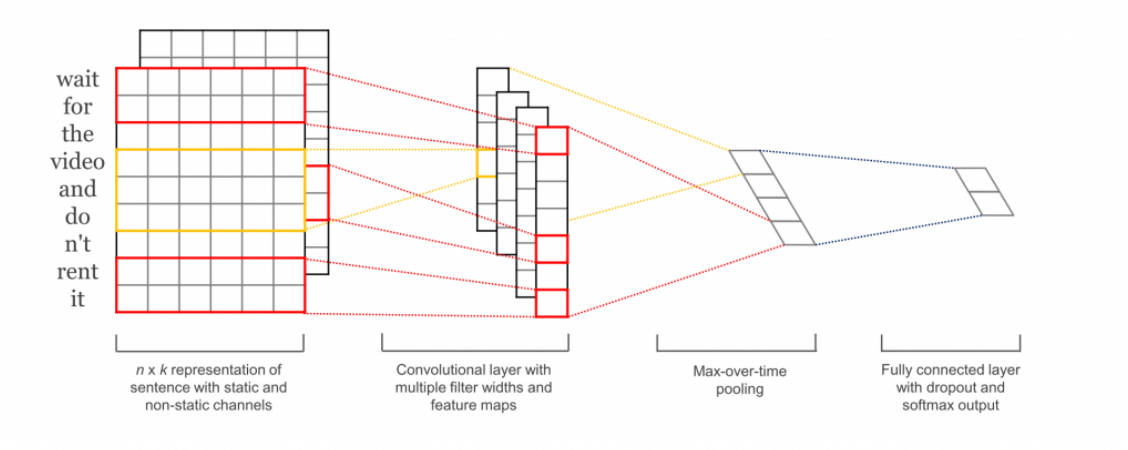


Figure 3.16: Model Architecture with Two Channels for an Example Sentence

maximum length (let's say it is 10 for simplicity) for a document (or tweet). We have 3 different cases for pad sequences:

- The number of words in the sentence is 10: There is not any problem, we use all these 10 words.
- The number of words in the sentence is more than 10: We ignore all the words starting from 11^{th} position. Therefore, only the first 10 words will be used in the input.
- The number of words in the sentence is less than 10: Let's say we have 7 words in the sentence. Now, we use all these 7 words and also 3 three more words which are all the same "<PAD>" word.

After the sequence preprocessing, we guarantee that all sentences are in the same input size. Zhang and Wallace [109] presents a model (Figure 3.17) in their work that makes everything clear about the text categorization.

The model in Figure 3.17 takes an input vector and applies convolution with 6 different filters. They have 3 different region sizes and 2 filters for each region size. After the convolution phase, they have 6 different feature maps retrieved from each filter, then they use 1 max pooling operation on each map. Consequently, they get 6 different univariate

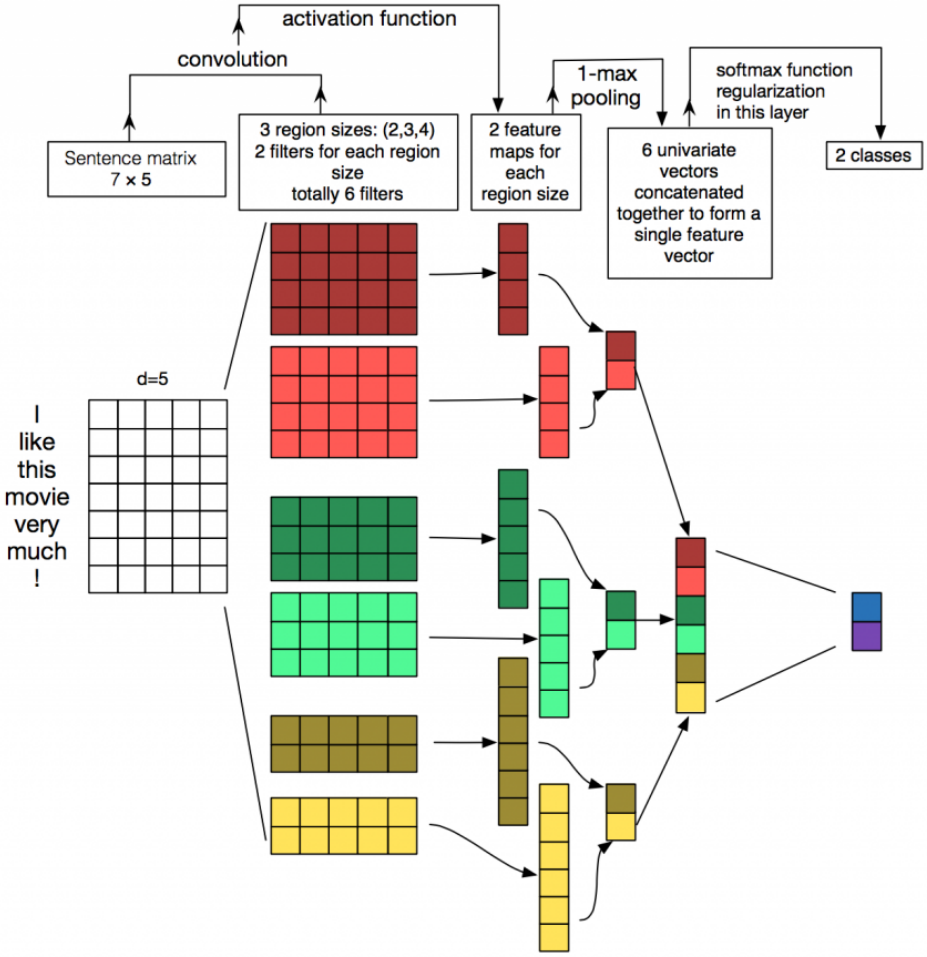


Figure 3.17: Model Architecture by Zhang and Wallace [109]

vectors to be concatenated before the fully connected layer. At the end, they have two different classes which means it is a binary classification.

3.5 Lexical Similarty Measures

3.5.1 Longest Common Subsequence

The longest common subsequence (LCS) is a way to find the longest subsequence common to two sequences. Assume X and Y are two strings and $X_i = x_1x_2...x_i$ is the prefix of X and $Y_j = y_1y_2...y_j$ is the prefix of Y. Then the length of the LCS can be found as follows:

$$LCS(X_i, Y_j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ LCS(X_{i-1}, Y_{j-1}) + 1 & \text{if } x_i = y_j \\ max(LCS(X_{i-1}, Y_j), LCS(X_i, Y_{j-1})) & \text{if } x_i \neq y_j \end{cases}$$

LCS problem can be solved with a dynamic programming approach in $O(m * n)$ time and space where m and n are the length of two strings.

3.5.2 Longest Common Substring

The longest common substring is a way to find the longest substring occurs on two strings. Note that substrings and subsequences are different concepts. Substrings are special cases of subsequences where characters should be consecutive. For example, for the string *abc*, the substrings are *a*, *b*, *c*, *ab*, *bc*, *abc* and empty substring. For the same string, the subsequences are *a*, *b*, *c*, *ab*, *ac*, *bc*, *abc* and empty subsequence. We would like to stress that *ac* is a subsequence but not a substring since *a* and *c* are not consecutive characters in *abc*.

The longest common substring between two strings can be found in $O(m*n)$ time and space with dynamic programming approach again where m and n are the length of two strings. However, it can be found in $O(m + n)$ time and space with Generalized Suffix Tree which will be discussed in the following subsections.

3.6 Clustering Methods

3.6.1 K-Means Clustering

K-means algorithm is a famous unsupervised learning technique that firstly used by MacQueen et al. [45]. The idea is actually quite basic as follows:

1. We firstly define K cluster centroids which represent K different clusters
2. Assign each data point to the closest cluster depending on the distance between the point and the centroids
3. Update centroid of the clusters
4. Repeat step 2 and step 3 until the centroids no longer move

3.6.2 DBSCAN

DBSCAN is a density based clustering algorithm designed by Ester et al. [18]. The basic idea behind the algorithm is to group all data points that are densely close to each other and assign data points as outliers that are placed in the low-density regions.

There are two parameters in DBSCAN, $minPts$ and ε. Several definitions stated in [18] should be known for this algorithm:

- A point p is called as *core point* if there are at least $minPts$ points (including itself) within distance ε.
- If p is a core point, then all points within distance ε to p are called as *directly density reachable* from point p wrt. ε.
- If there is a chain of points like $p_1, p_2, ..., p_n$ where p_1 is p, p_n is q and p_i is directly density reachable from p_{i+1}; then p is *density reachable* from q.
- A point which is not directly density reachable from any other point called as *outlier*.

- Each cluster has at least one core point. In addition to this, non-core points can also join to the cluster, and we call them as *border points* since they cannot expand the cluster with more points.

The pseudocode of the algorithm is given in Figure 3.18. The complexity of the algorithm is normally $O(n^2)$, however if nearest documents can be retrieved by an index based structure (like R* tree in original paper [18]) in *regionQuery* function, then complexity can become $O(n \log n)$.

3.7 Data Structures/Indexing Methods to Improve DBSCAN

We will be updating *regionQuery* function in Algorithm 4 and Algorithm 5; and utilizing generalized suffix tree and locality sensitive hashing to efficiently retrieve nearest tweets in a density based clustering algorithm.

3.7.1 Suffix Trie

Suffix trie is a trie that contains all the suffixes of a string. To construct a suffix trie of a string T, we add a special terminal character (for example \$) to the end of T. This terminal character should not occur anywhere in T and should be alphabetically lower than any other character in T. In trie, each edge represents a character from the alphabet and each node has at most one outgoing edge. An example of suffix trie is given in Figure 3.19[12] when T = "abaaba".

If we want to check any string is a substring of T, we need to follow the path from the root and check whether this path exists. For example, if we look for "aba" in T, there is a path a-b-a from the root. So this is a substring. If we look for "abb", there no such path, so this is not a substring. Another feature is that when we find a substring in the trie, if

[12]Figure 3.19, Figure 3.21, Figure 3.22, Figure 3.23, Figure 3.24 were retrieved from `https://www.youtube.com/watch?v=hLsrPsFHPcQ`

```
DBSCAN(D, eps, MinPts) {
   C = 0
   for each point P in dataset D {
      if P is visited
         continue next point
      mark P as visited
      NeighborPts = regionQuery(P, eps)
      if sizeof(NeighborPts) < MinPts
         mark P as NOISE
      else {
         C = next cluster
         expandCluster(P, NeighborPts, C, eps, MinPts)
      }
   }
}

expandCluster(P, NeighborPts, C, eps, MinPts) {
   add P to cluster C
   for each point P' in NeighborPts {
      if P' is not visited {
         mark P' as visited
         NeighborPts' = regionQuery(P', eps)
         if sizeof(NeighborPts') >= MinPts
            NeighborPts = NeighborPts joined with NeighborPts'
      }
      if P' is not yet member of any cluster
         add P' to cluster C
   }
}

regionQuery(P, eps)
   return all points within P's eps-neighborhood (including P)
```

Figure 3.18: DBSCAN

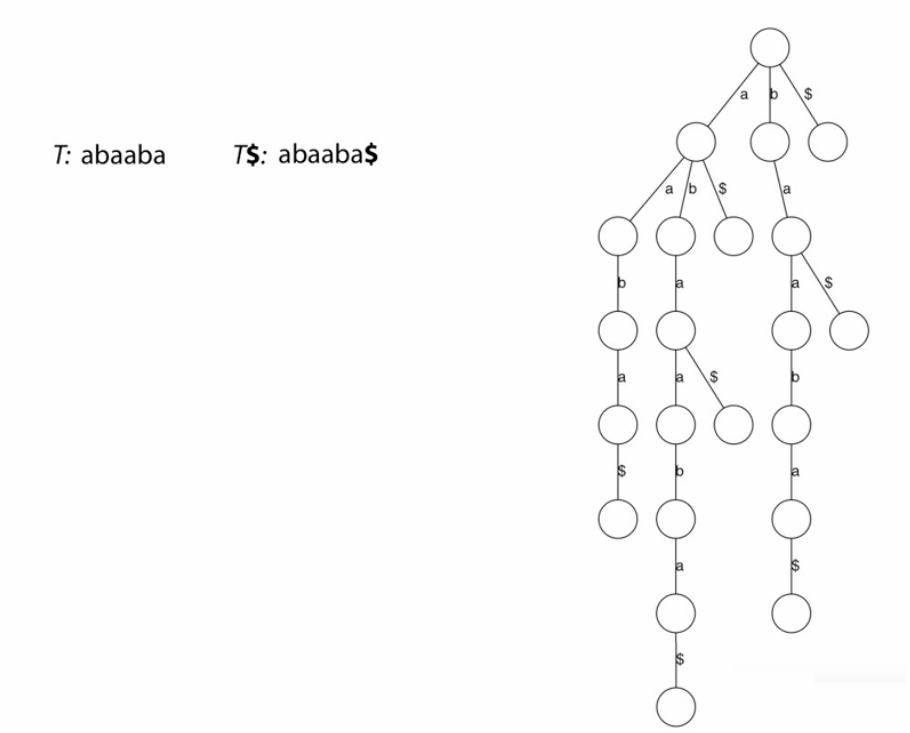


Figure 3.19: An Example of Suffix Trie

the final node we arrive has an outgoing edge with $ sign, it means that this substring is a suffix as well. When we follow “aba”, the node we end up has an outgoing edge with $, so “aba” is a suffix of T. However, when we follow “ab”, the final node does not have an edge with $, so this is a substring but not a suffix of T. This point is one important roles of $ sign in the trie. We can also count the number of times a string occurs in T. Let’s check it for “aba” again. When we reach the final node, we need to consider this node as the root of the sub-tree and then we see that this sub-tree has 2 leaves with a DFS (Depth First Search). Therefore “aba” occurs 2 times in T.

When we consider about the worst case space complexity scenario of suffix trie, T becomes $a^n b^n$ as in Figure 3.20[13]. In this scenario, we will have n nodes for the sequence of b on the right. n nodes for the sequence of a in the middle. Then n*n nodes for the each b sequence coming after a. Additionally, there will be 2*n + 1 $ nodes as well. In

[13]Figure 3.20 was taken from `https://www.cs.cmu.edu/~ckingsf/class/02-714/Lec10-suffixtrees.pdf`

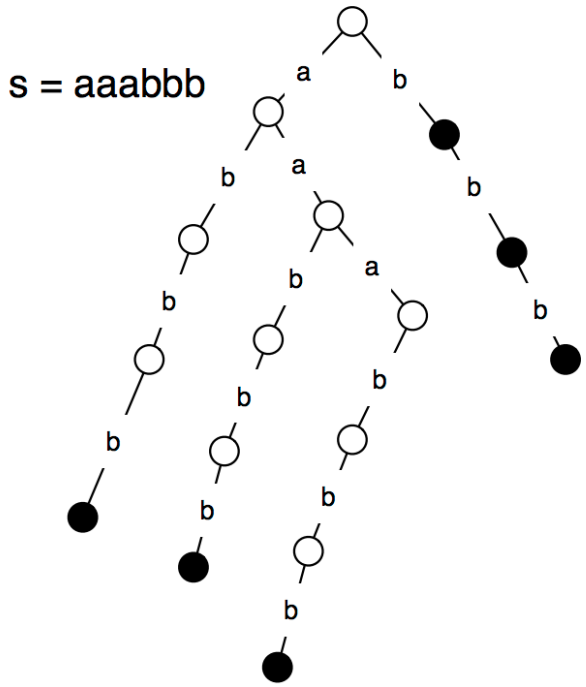


Figure 3.20: Worst Case Space Complexity of Suffix Trie

total we will have $O(n^2)$ nodes which is not that efficient.

3.7.2 Suffix Tree

As we mentioned, suffix trie is not efficient in terms of space requirements, therefore we need to make it smaller. Suffix tree is actually a smaller version of suffix trie where non-branching paths are merged as in Figure 3.21.

Let's call the length of T as m. In suffix tree, there are $m + 1$ leaves (one for \$ sign alone), and every non-leaf node has at least two children. This property leads us suffix tree will have at most $m - 1$ non-leaf nodes (i.e tree is a full binary tree). Therefore, we have $O(m)$ space complexity in terms of the node number for suffix tree which is quite good. However the total size of the tree is not $O(m)$ yet, since we still keep the edge labels as strings which increases the amount of space. So, instead of keeping actual strings as edge label, we will keep "(offset, length)" information as in Figure 3.22. Now, upper bound for space requirements in suffix tree is $O(m)$.

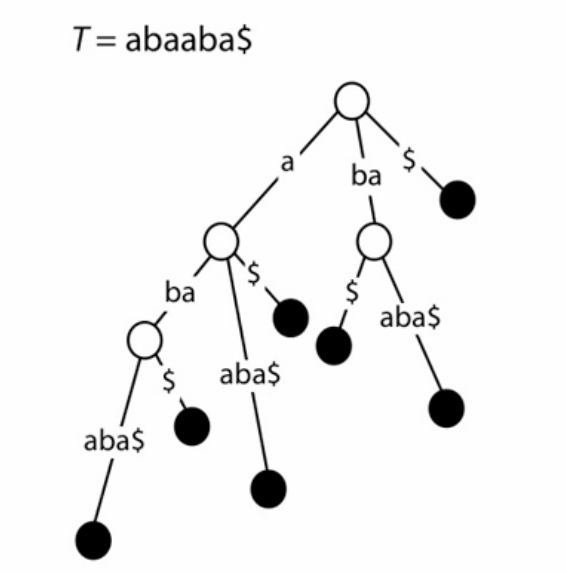


Figure 3.21: An Example of Suffix Tree

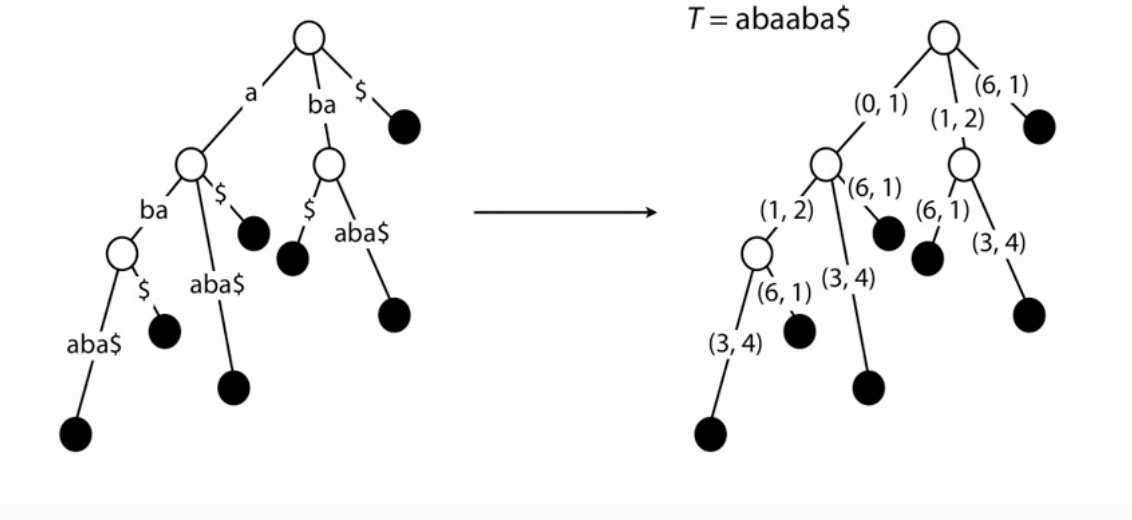


Figure 3.22: Converting Edge Labels into (offset, length)

One additional detail regarding suffix tree is to store a number which is the offset of the suffix that leaf represents for time efficiency reasons. For example, consider the suffix "abaaba$" in T which is actually the longest suffix of T. The leaf that represents this suffix has value 0 (since the offset is 0) as shown in Figure 3.23.

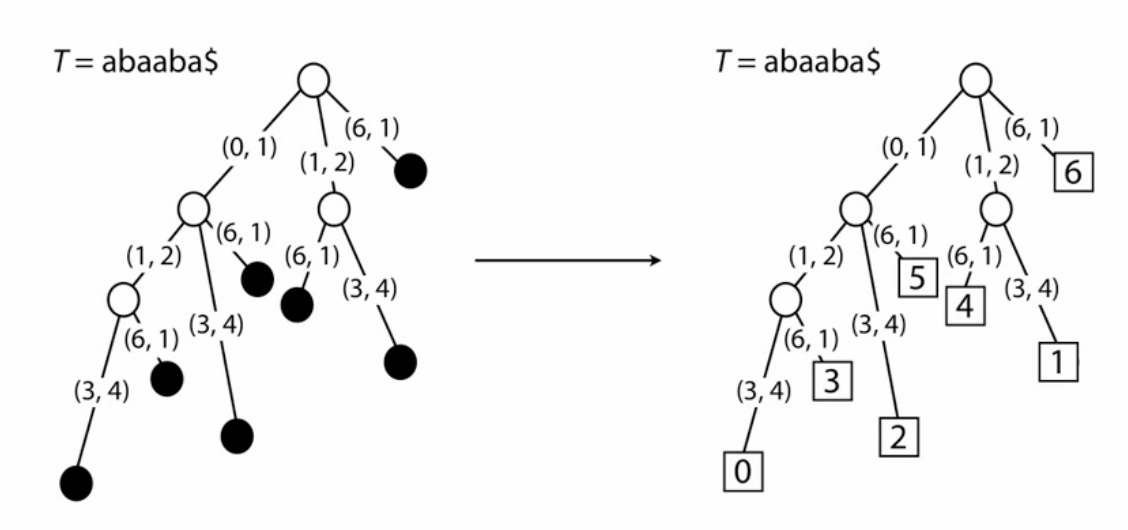


Figure 3.23: Storing Offsets in the Leaves

Now we know that the number of nodes in the suffix tree is at most $2 * m + 1$. The number of edges is always equals to $numberOfNodes - 1$ which is at most $2 * m$. In each edge, we store 2 integer values (offset and length), and we keep an additional integer for each $m + 1$ leaves. Therefore, the total amount of space is still $O(m)$. We do not keep node labels explicitly, however we can implicitly extract a node label by merging the edge labels from the root to that node.

Gusfield [24] presents a naive algorithm to build such a suffix tree which has a $O(m^2)$ time and $O(m)$ space complexity. Efficiently constructing a suffix tree is completely another research area, however linear space and time complexity achieved by Ukkonen [94]. Therefore, we use Ukkonen's algorithm; time and space complexity to construct a suffix tree are both noted as $O(m)$ in the rest of the thesis.

It is also possible to use many strings together to build one suffix tree. This kind of suffix tree is called as "Generalized Suffix Tree" (GST). For instance, we have two different strings $X = xabxa$ and $Y = babxba$. We append these strings together with

different terminal characters # and $ respectively. Therefore appended string becomes $X\#Y\$ = xabxa\#babxba\$$ which builds the suffix tree in Figure 3.24.

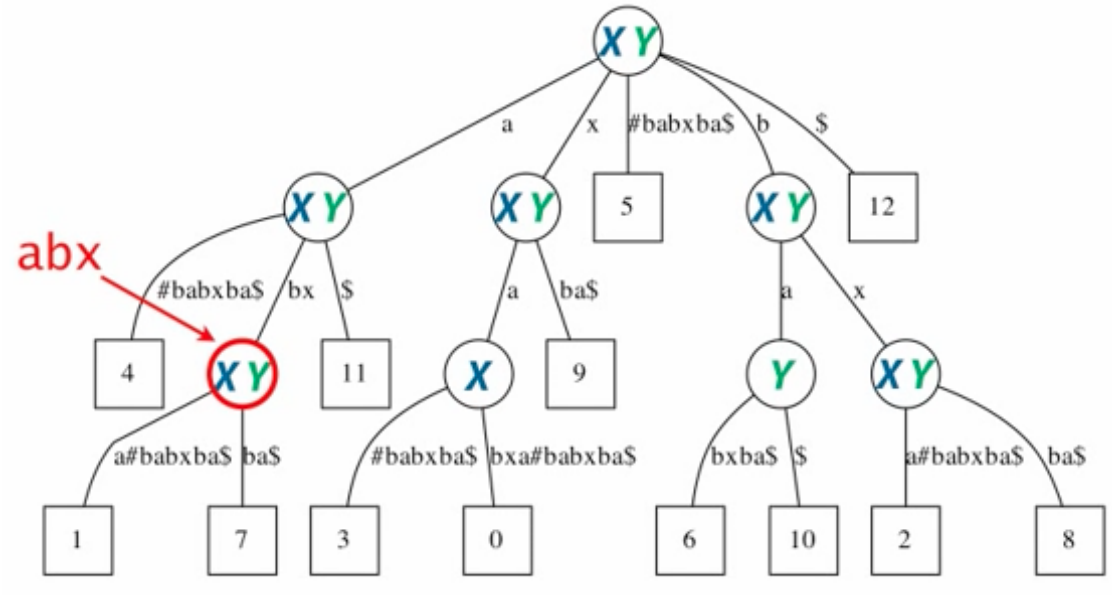


Figure 3.24: Generalized Suffix Tree of X and Y

First we know that if the offset is in [0,4] range then it refers to X, if the offset is in [6,11] range then it refers to Y. If we want to find common substrings between X and Y, we need to look up for the nodes which has both X and Y representing leaves. In order to find longest common substring between these strings, then we need to get deepest such node which represents "abx" label in Figure 3.24. The good thing is that whole process takes $O(\mid X \mid + \mid Y \mid)$ space and time complexity.

3.7.3 Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) is a well known technique that is used to retrieve nearest neighbours efficiently [22]. In LHS, we generate K bits hash codes for each document and we expect that similar documents will have same hash code as well.

For instance, we have 5 different documents in Figure 3.25[14], and we will generate

[14]Figure 3.25 and Figure 3.26 were retrieved from `https://www.youtube.com/watch?v=dgH0NP8Qxa8`

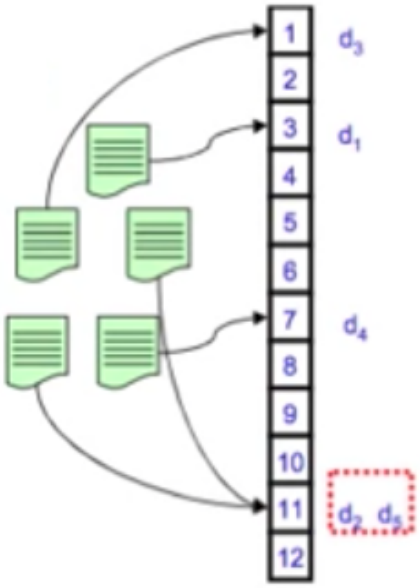


Figure 3.25: Locality Sensitive Hashing

hashes for each of them. As it will be noticed, d_2 and d_5 is placed in the same bucket since they generated same hash code. If we are looking for similar documents to d_2, all the documents placed in the same bucket with d_2 are the "candidates" for being similar documents to d_2. We say "candidates" since even very different documents can generate the same hash code by chance and can be placed in the same bucket. For instance, if we use K bits as hash length, then we can produce 2^K hashes (buckets). Therefore, we will have $N/2^K$ documents averagely in each bucket where N is the total number of documents in the corpus. We need to check each of the candidates in a specific bucket whether they are really similar to our document. In other words, false positives may occur in the same bucket; however they do not cause any problem since we make pair check for each candidate in the bucket.

On the other hand, the weakness of LHS is that although we guarantee exact same documents generate same hash code and placed in the same bucket, very similar documents might end up with a different hash code. Therefore, there is always a chance

for missing similar documents in LSH. In order to decrease the probability of missing a similar document, we repeat the same process with L different hash tables as in Figure 3.26. For example, although d_2 and d_5 are similar documents, they produced different hash codes in H_1 and H_3; but the same hash code in H_2. So that, when we have a data point or a document (a tweet), it averagely needs $L * N/2^K$ comparisons to retrieve the similar documents. It would take N comparisons if we used a naive approach to retrieve them. Of course, even we repeat the same process L times, there is no guarantee to get all of the similar documents. There is a trade off here, if we increase the value of L then the probability of missing a similar document will decrease, but the time complexity will increase.

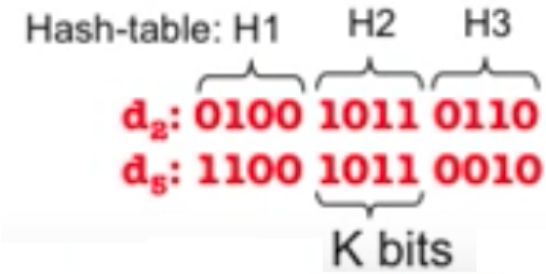


Figure 3.26: Generating L Hash Tables for LSH

While generating hashes, the random sampling method is used with *MinHash* algorithm [9]. To do this, we represent text documents with *shingles*. For example, if a document d = "I support FB" and set $shingleSize$ as 3 then this document is represented with $length(d) - shingleSize + 1$ shingles as follows:

$shingles\_d = \{$"$I\ s$", " su", "sup", "upp", "ppo", "por", "ort", "$rt\ $", "$t\ F$", " FB"$\}$

When we want to generate a hash code with length K, we follow the pseudocode in Algorithm 1.

For each one of the K information in the result, we generate different hashes using FVN hash by Fowler et al. [20] for each shingle and then assign the minimum of them. Surely, we generate K of them for a document in one hash table. For other hash tables, we repeat this process L time with different $initializedRandomSeeds$.

Algorithm 1: Generating Hash with Length K for Document d

Parameters: $Document\ \ d, List\ \ initializedRandomSeeds, Integer\ \ K$

$result \leftarrow new\ List[K]$;

for $j \leftarrow 0$ ***to*** K **do**
  $result[j] \leftarrow MAX\_VALUE$;
end

$shingles\_d \leftarrow getShingles(d)$;

for $shingle$ ***in*** $shingles\_d$ **do**
  for $i \leftarrow 0$ ***to*** K **do**
    $seed \leftarrow initializedRandomSeeds[i]$;
    $currentHashValue \leftarrow FVN\_Hash(token, seed.left, seed.right)$;
    if $currentHashValue < signatures[i]$ **then**
      $result[i] \leftarrow currentHashValue$;
    end
  end
end

return $result$;

Chapter 4

Predicting Impact of Tweets and Topics

Twitter, a micro-blogging social media, which was initially adopted for networking and entertainment [29, 66], has also started to be used in cases of social and political movements [62, 51, 57, 80, 82, 86, 87]. As a *cyber-ekklēsía*[1] with effortless accessibility and impetuous information sharing, its users continuously and contagiously declare their emotions and ideas on topics of their choice [33, 58, 74, 83]. Twitter produces immense amounts of data allowing to probe temporal behavioral patterns [31]. It records 500 million tweets per day together with some basic information on its 313 million monthly active Twitter users. As of year 2016 the volume of tweets is estimated to grow at around 30% per year[2]. It is estimated that there will be around 2.67 billion social network users worldwide by 2018[3]. Twitter data analysis uncovers meaningful findings on individual and group tweeting characteristics, often revealing situational phenomena while predicting future events [31, 71]. Among Twitter's top markets of active users, Turkey ranks 8^{th} with a share of 3.0 per cent of global users [65] and Twitter is the 7^{th} most popular

[1] a political assembly of citizens of ancient Greek states; especially: the periodic meeting of the Athenian citizens for conducting public business and for considering affairs proposed by the council (`http://www.merriam-webster.com/dictionary/ecclesia`).

[2] The data retrieved from (`http://www.internetlivestats.com/twitter-statistics/`).

[3] The data retrieved from (`http://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/`).

website in Turkey [1]. Twitter is a gigantic data source, which can allow us to capture and analyze reflections of the society in general in Turkey.

Twitter analysis has been used on a wide array of research topics ranging from financial decision making to political science. For example, emotions can profoundly affect individual behavior and decision-making and analyses of Twitter have been used in predicting the tweet moods effect on daily up and down changes in the stock market [8, 108]. On the other hand, election results have been predicted successfully by Twitter analysis [90]. Use of Twitter analysis is not only limited to choice making, but also spread in conflict and emergency management as well. Twitter is known to exhibit cues that can enable analyzers to detect real-time events specifically of emergency nature [67, 68].

A well-known regional conflict with recent worldwide effects is the Syrian conflict. Turkey is the most affected country of the Syrian conflict in political, social and economic terms and is living a dramatic demographic change as one of the main host country of Syrian refugees. In Turkey, 80% of Syrian refugees are living in urban areas, while some camps are built for them for accommodation [21]. Aras and Şahin Mencütek [2] showed how variations in foreign policies immediately reflect on states' responses. Turkey first adopted an "open door" policy towards Syrian refugees welcoming all, however, upon confrontation with massive flows of refugees needed to revise her foreign policy orientations. Heisbourg [26] claims that one of the quickest and least difficult policy for EU to implement is financial supports massively to the UN High Commissioner for Refugees in the region to cope with refugee flows from Syria. He also claims that the negotiation between EU and Turkey for funding to support refugee relief in Turkey is vital to eliminate refugee flows since Turkey has become a major transit point for refugees and other migrants trying to reach Europe [21]. One of the main issues regarding Syrian crisis is education of refugee children who need to access to basic education at all levels. Bircan and Sunata [7] mention that collaboration among public and private partners at the local, national and international levels is crucial for the education program development mainly due to lack of financial matters.

Dekker and Engbersen [14] argue that social media both provides new communication

channels in migration networks and facilitates migration. Social media helps people to access a widespread informal information and thus expands the perspectives of candidate migrants. Emerging field of research on uses of Social Media during social movements, crises, and conflicts includes several finding with regards to digital activism [32, 42, 69, 89]. However, most of this research focuses on a narrow group of people since digital activists are referred as individuals who engage online for social change. This study has a wider perspective of the host population while researching Syrian conflict. Therefore, one of the contributions of our study is to fill the gap in the literature by analyzing Twitter reflections on Syria conflict with a wider perspective.

The Syrian conflict led to an out-sized impact on many aspects of life, from individual to public, in many parts of the globe [30]. Reports indicate that the economic consequence exceeds $35 billion with more than 4.6 million people relocated [48]. Among Syria's immediate neighboring countries, Turkey is known to host the largest population of Syrian refugees [54]. Some reports on refugee studies indicate emerging tensions among host communities, displaced Syrians and humanitarian policymakers and practitioners [10, 60] while others underline the positive impact of Syrian businesses in Turkey [36]. In absolute terms, it is obvious that the 2.7 million Syrian refugees [95] registered in Turkey which has a population of 79.7 million [100] should have caused some changes. In this respect, our work also aims to utilize Twitter analysis to explore the reflections present in the Turkish social networks about Syria and the related sub-topics. Our findings are based on the analysis of 450 thousand tweets that were streamed between February 1^{st} and February 27^{th} 2016 with content in Turkish language. Main research question in this chapter is "can we predict whether a tweet will have high number of retweets". Additionally, we focus on the findings regarding following questions as well: (1) what are the widely discussed topics on Twitter about Syria by the Turkish users, (2) were these topics attracting more users to Twitter, or encouraging further engagement of the already existing users, (3) how are the fading out characteristics of the most popular topics could be described?

4.1 Data and Methodology

A Twitter Stream API[4], which allows for a random small sample of all related tweets is employed to collect real time publicly available tweet contents and features that contain the keyword of "Syria" ("Suriye" in Turkish) between February 1^{st} 2015 and February 27^{th} 2016. A total of 450000 tweets are collected, all in Turkish language. As a minor limitation, unfortunately our server received an attack and it lost its data connection between 24^{th} March 2015 and 8^{th} May 2015.

Distribution of daily tweet numbers, distribution of daily user numbers and cumulative user numbers are explored for comprehending the data characteristics in relation to the research questions 1 and 2. Additionally, the most retweeted tweets that provided the highest spread and their related dispersion are analyzed. The acceleration of spread that the highly retweeted tweets amass until reaching a saturation point is investigated with regards to the 3^{rd} research question. Here, the saturation point is defined as the time or the point, where a specific highly retweeted tweet loses its acceleration but starts receiving lower numbers of retweets. For this purpose, we extracted the top highest retweeted 50 tweets in our data set and analyzed their spread and dispersion. Next, a deeper analysis based on tweet's content characteristics is conducted by increasing the number of tweets included in the analysis step by step. Some content-based features are extracted to construct a learning model by using data mining techniques. Lastly, prediction capability and accuracy of retweet acceleration of our model is tested with 3000 tweets.

4.2 Analysis and Results

4.2.1 Reflection of Real Life Phenomenon in Tweets

Previous research indicates that Tweet volume is an indication of social trends taking place in real life [51, 57, 86, 87]. Therefore, we analyzed our data to understand the distribution

[4]Twitter API Rate Limits from Twitter Developers: `https://dev.twitter.com/rest/public/rate-limiting` Retrieved on May 12, 2016

of the total tweets per day. There were 10 days, which had more than 5000 tweets posted (Table 4.1). We mapped events and news during the top 10 highest numbers of Tweets posted daily to the Twitter volume time-series (Figure 4.1). The peak day in our data set is on July 20^{th}, 2015 with 11722 tweets posted. Common themes that lead to an increase in the number of tweets posted are identified by chronologically associating real life events and news. When we categorize events about the relevant news that occurred on specific peak dates, we realize that a common theme is emerging around 'armed conflicts'. This is followed by 'religious and political sensitivities', and 'in-group collectivistic cultural traits' among the Turkish public. Turkish culture is described as high on power distance and in-group collectivism [23, 27], while carrying paternalistic values [4, 35, 101]. House et al. [28] defined in-group collectivism as "the degree to which individuals express pride, loyalty, and cohesiveness in their organizations or families". The high volume of tweets posted upon news about border issues (*249 people were arrested at the border; 'wall' to protect the Syrian border; Cilvegozu border gate was closed*), and beyond border action (*Humanitarian aid was delivered to the enclaves; land operations to Syria is necessary*) could be related with both in-group collectivism and paternalism of Turkish culture.

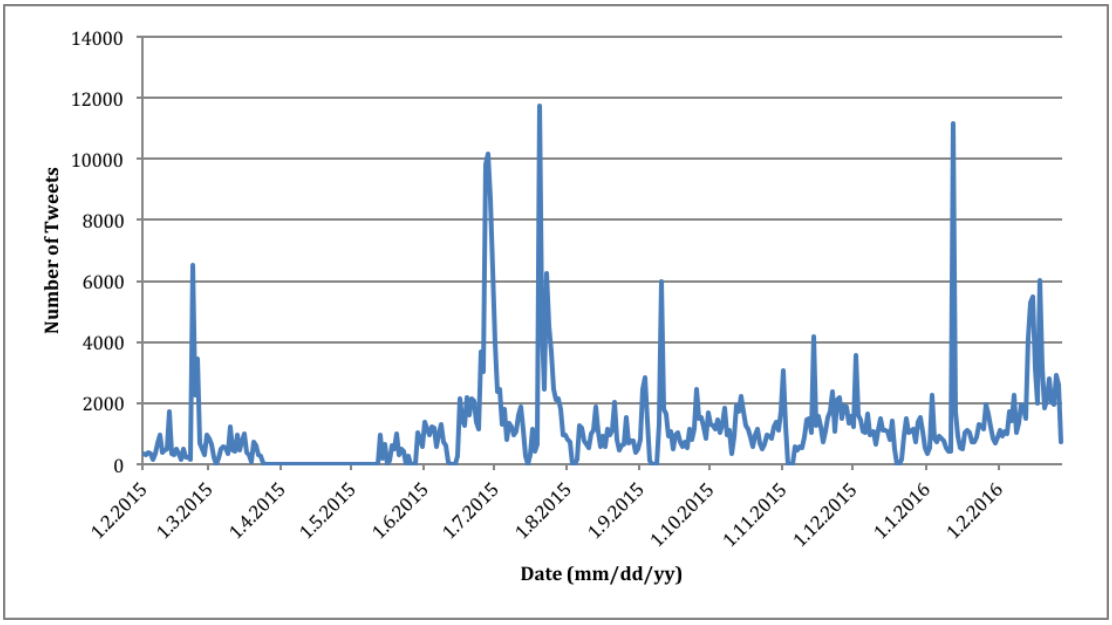


Figure 4.1: Daily Number of Tweets (01 Feb 2015 – 27 Feb 2016)

	Date	# of Tweets	Real Life Occasion
1	20 July 2015	11722	Assad attacked the Turkmen village; Suruc attack; (19 July 2015: ISIS used chemical weapons in Syria and Iraq)
2	12 Jan 2016	11146	Explosion occurred in Sultanahmet; Operation to ISIS in 5 cities
3	28 Jun 2015	10163	Sound of cannons across the Kilis; 249 people were arrested at the border;
4	27 Jun 2015	9805	ISIS attacked Al-Hasakah; The second largest massacre in Kobani; The 'wall' protection to Syrian border; Kobani again in YPG's hands
5	22 Feb 2015	6533	The place of Suleyman Shah Tomb has been changed; Tomb was evacuated, conflicts were increased
6	23 Jul 2015	6248	Conflict in Kilis, 1 Turkish soldier was killed; Turkish F-16 hit the ISIS targets; Early elections commentary from the leader of the main opposition party
7	18 Feb 2016	6014	Humanitarian aid was delivered to the enclaves; Prime minister unclosed the perpetrators
8	10 Sep 2015	5985	Cilvegozu border gate was closed; 5.5 tons of bomb in trucks loaded with onions
9	15 Feb 2016	5467	Opened fire from YPG regionin Syria, howitzers began to respond; 50 kilograms of explosives were seized in Şanlıurfa; YPG took control of 70% of Tel Rifat
10	14 Feb 2016	5309	Minister Yilmaz said TSK is hitting YPG positions; A concrete wall is put up at the Turkey-Syria border; land operations to Syria is necessary

Table 4.1: Top 10 Highest Numbers of Tweet posting Days and Related Real Life Events

4.2.2 Attracting New Users to Post

There is a similar pattern between the daily number of tweets and daily number of distinct users in the dataset (Figure 4.1, Figure 4.2). We conclude that users, who prefer to post tweets about Syria, choose to post only 1-2 tweets per day. The high resemblance in the patterns of daily distinct users and tweet numbers distributions indicates that a wide range of individuals chose to reflect on the topic of Syria.

One other interesting feature of the dataset is cumulative number of users who are talking about Syria during this period. We observed that cumulative number of users is linearly increasing day by day, reaching more than 175000 users towards the end of our dataset. The topic somehow managed to engage attention of different people as we see that different people continuously joined and increased number of distinct people. Therefore, we conclude that tweets containing Syria have not been only posted by a small specific group of people, but indeed attracted new individuals to post (Figure 4.3).

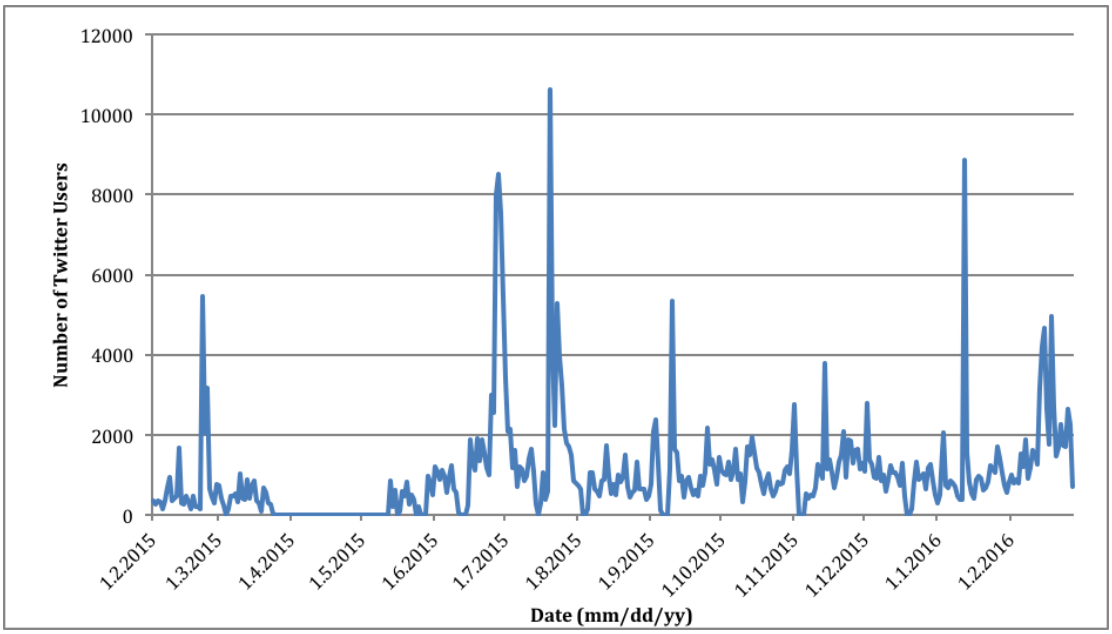


Figure 4.2: Daily Number of Twitter Users (01 Feb 2015 – 27 Feb 2016)

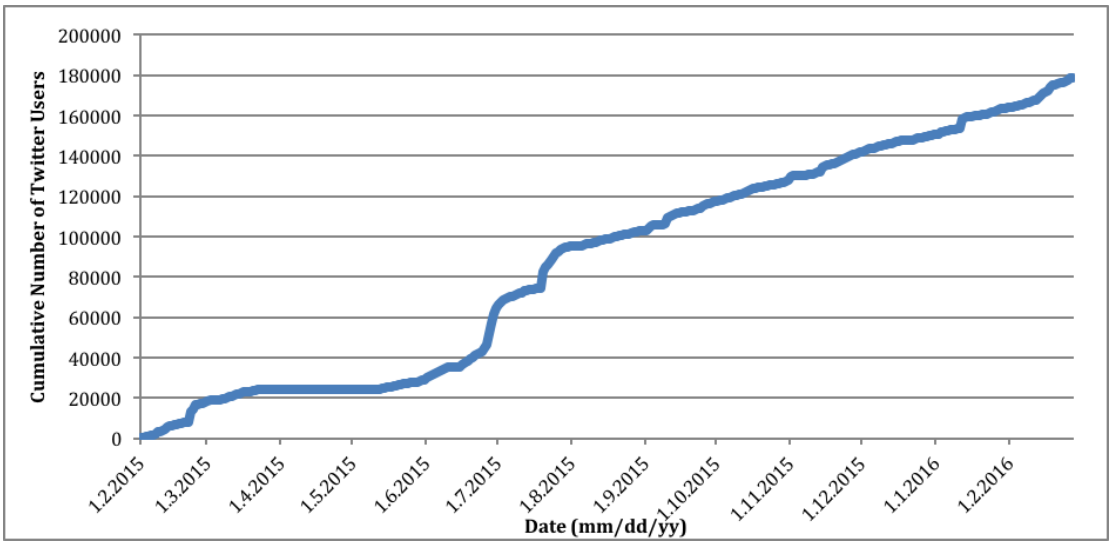


Figure 4.3: Daily Cumulative Numbers of Twitter Users (01 Feb 2015 – 27 Feb 2016)

4.2.3 Spread and Fade Out Characteristics

The highly retweeted top 50 tweets are identified in Table 4.2 (Spread and fade out patterns for these tweets are also given in Figure 4.4). Apart from that we also extracted the propagation velocity of the tweets. In order to calculate it, we used Simple Linear Regression method to discover relations in the data model. Simple Linear Regression is kind of modeling the relation between two variables to represent the state of the data [102]. Therefore, we fit our tweet distribution data for each tweet into a line by using Simple Linear Regression; and then calculated the slope of the line that gives us the idea about the acceleration of the tweet. *Tweet1*, which was posted by *user0* on 20/07/2015 12:26:43, has received the highest number of retweets with a total of 9973. *User0* had 1.873.272 followers when posting the *Tweet1*. The retweet line has exercised a slope of 305.64 before reaching a saturation point.

It is observed that some tweets experience a sharp decline in retweets once a saturation point is reached. On the other hand, there are also tweets that enjoy a rather smooth decline with a lower diminish of retweets. The sharpness of retweet decline, which is an indicator of a tweets fade out characteristics, is examined for the top 50 retweeted tweets (Table 4.2). *Tweet6* exercised the sharpest slope (706.50) while amassing 1481 retweets.

On the other hand, *Tweet3* reached some 3851 retweets but observed a smoother trend with the most curved slope (109.41) with highest number of followers. *Tweet38* observed the smoothest slope with only 442 retweets (Figure 4.5).

Tweet ID	**User ID**	**# of Followers**	**Timestamp**	**# of Retweet**	**Slope**
Tweet1	user0	1.873.272	Mon 20/07/15 12:26:43	9973	305.64
Tweet2	user1	1.277.607	Fri 26/06/15 23:56:12	4908	188.45
Tweet3	user2	6.614.251	Sat 27/06/15 21:52:22	3851	109.41
Tweet4	user1	2.610.948	Sun 14/02/16 22:52:37	1633	61.62
Tweet5	user3	37.275	Thu 12/02/15 20:25:58	1530	461.50
Tweet6	user4	16.810	Sun 22/02/15 12:02:36	1481	706.50
Tweet7	user5	73.650	Thu 25/02/16 11:51:55	1095	373.70
Tweet8	user6	640.376	Thu 13/08/ 15 15:20:54	1070	106.38
Tweet9	user7	10.995	Mon 23/02/15 21:42:16	1053	524.00
Tweet10	user8	1.520.728	Sun 03/01/16 13:14:29	1020	151.17
Tweet11	user8	1.521.315	Tue 12/01/16 11:39:20	988	111.61
Tweet12	user9	884.047	Mon 29/06/15 15:52:58	765	46.84
Tweet13	user10	33.886	Tue 12/01/16 14:04:41	730	110.57
Tweet14	user11	47.849	Thu 29/10/15 04:18:42	726	139.20
Tweet15	user12	381.451	Fri 25/09/15 08:25:12	724	91.57
Tweet16	user11	58.074	Sun 14/02/16 10:40:03	722	35.74
Tweet17	user13	634.235	Fri 24/07/15 07:52:51	712	133.63
Tweet18	user14	19.260	Thu 10/09/15 15:10:10	674	137.80
Tweet19	user15	676.881	Wed 22/07/15 18:14:58	670	96.04
Tweet20	user16	2.747	Sun 08/02/15 20:25:54	645	322.50
Tweet21	user17	50.234	Sun 28/06/15 12:30:48	644	201.60
Tweet22	user18	166.810	Sun 22/11/15 09:24:10	641	98.89
Tweet23	user19	316.294	Mon 20/07/15 11:41:33	609	186.20

Tweet24	user20	190.091	Mon 20/07/15 18:39:19	600	102.86
Tweet25	user21	23.553	Thu 23/07/15 14:43:11	590	125.30
Tweet26	user22	535.194	Sun 14/06/15 21:39:45	568	28.57
Tweet27	user23	446.603	Mon 29/06/15 18:22:27	559	73.14
Tweet28	user24	166.660	Sun 28/06/15 06:38:20	530	56.06
Tweet29	user25	111.404	Thu 09/06/15 21:35:27	529	86.09
Tweet30	user26	2.345	Wed 01/07/15 17:32:13	517	63.35
Tweet31	user27	85.476	Sun 22/02/15 14:51:54	515	36.59
Tweet32	user8	1.521.326	Tue 12/01/16 14:27:35	511	74.03
Tweet33	user28	2.520.567	Sun 22/02/15 05:48:26	503	35.91
Tweet34	user29	84.693	Wed 08/07/15 09:51:00	485	81.34
Tweet35	user11	50.877	Fri 13/11/15 22:33:04	474	26.69
Tweet36	user30	300.507	Sun 28/06/15 22:23:51	466	51.50
Tweet37	user31	1.556.018	Sun 22/02/15 10:28:58	445	90.50
Tweet38	user32	44.427	Thu 02/07/15 19:49:44	442	25.83
Tweet39	user33	264.823	Wed 17/02/16 23:55:25	439	218.00
Tweet40	user34	40.102	Sat 28/11/15 00:11:09	439	26.74
Tweet41	user35	57.454	Wed 24/02/16 19:01:11	432	108.90
Tweet42	user6	694.295	Mon 18/01/16 10:15:28	424	41.88
Tweet43	user36	62.246	Sun 22/02/15 10:26:10	418	209.00
Tweet44	user37	17.110	Thu 23/07/15 14:51:34	412	101.70
Tweet45	user38	4.220	Tue 12/01/16 11:51:05	412	125.00
Tweet46	user39	106.591	Thu 10/09/15 14:37:51	410	204.00
Tweet47	user40	302.911	Sun 15/02/15 16:14:51	407	62.94
Tweet48	user41	2.987.212	Fri 25/1215 23:55:36	404	31.72
Tweet49	user42	96.658	Wed 22/07/15 07:32:08	402	401.00
Tweet50	user41	2.686.033	Tue 30/06/15 01:07:47	400	58.74

Table 4.2: Details of Highly Retweeted Top 50 Tweets and Posting Accounts

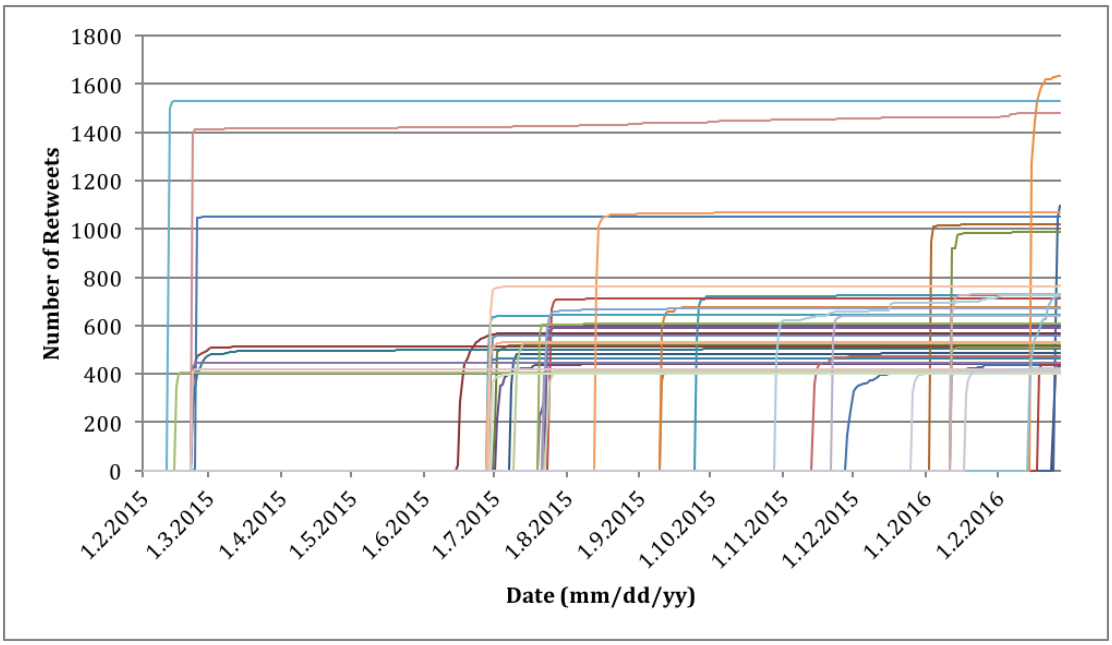


Figure 4.4: Spread and Fade out Patterns of Top 50 Retweeted Tweets

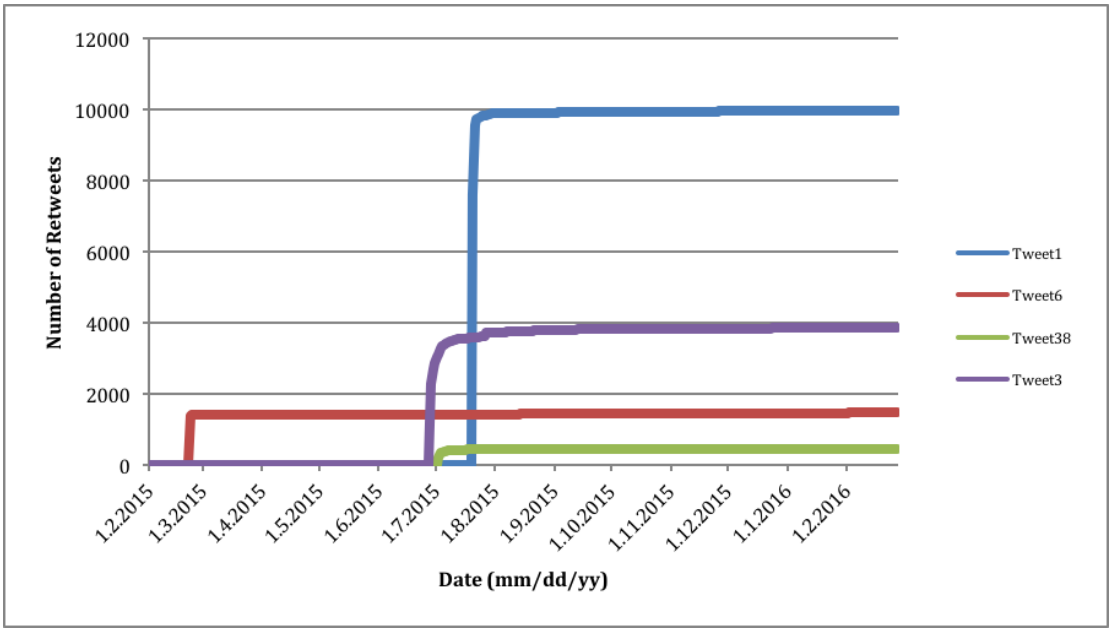


Figure 4.5: A Sample Spread and Fade out Patterns of Some Highly Retweeted Tweets

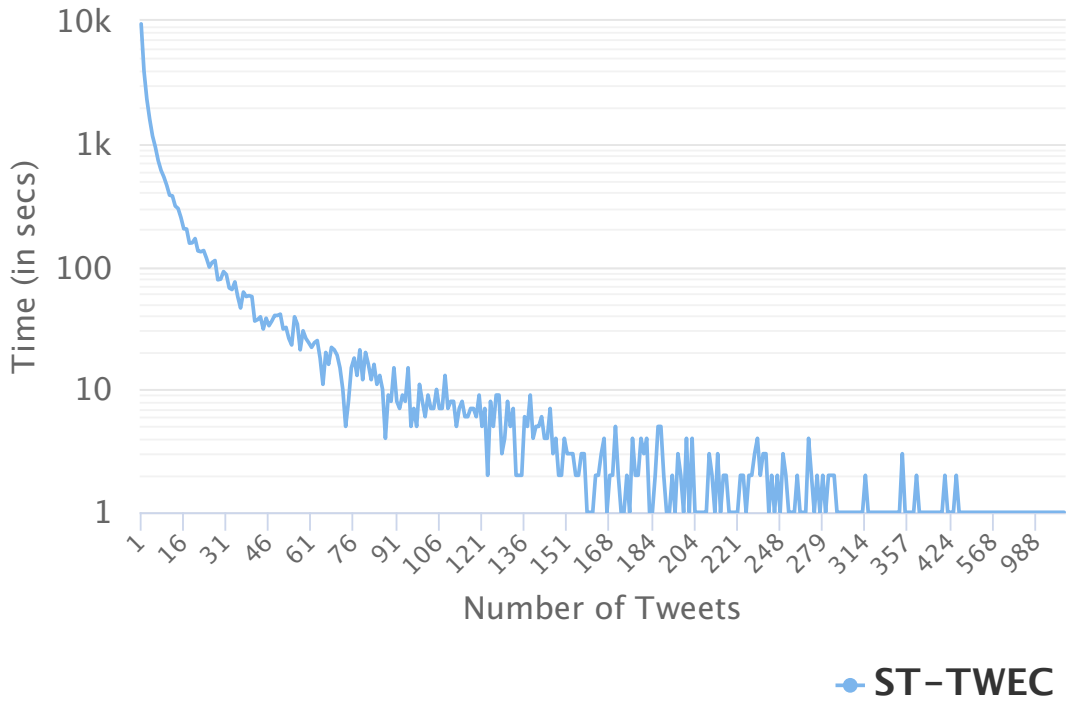


Figure 4.6: Distribution of Different RT Numbers

The distribution of different retweet numbers is given in Figure 4.6. In the next section, further analysis is conducted with a wider sample containing the top 1000 most retweeted tweets.

4.2.4 Conditions and Features Leading to High Retweet

All the points we have observed until this section are based on statistical features of the Syria tweet data in Turkish. These observations showed us the Syria keyword had contagiously been mentioned by different people on Twitter. After this point, we decided to make a deeper analysis on these tweets based on their content characteristics. We started to extract some content-based features from these tweets to construct a learning model by using some data mining techniques. The reason behind extracting content-based features is to test whether we can make good predictions about the acceleration and total retweet number of tweets by using these features. Therefore, we started with the content-features given in Table 4.3 for the mostly retweeted 1000 tweets. All the features given in Table 4.3

are the features where they will be used as regular attributes of a learning model.

Attribute	Content-feature
containsHashtag	It is a binomial attribute. It is true for the tweets that contain a hashtag, false otherwise.
containsLink	It is a binomial attribute. It is true for the tweets that contain a link, false otherwise.
numberOfFollowers	The follower number of the user at the time when the tweet had been sent.
numberOfCapitalLetters	The number of capital letters in the tweet text.
containsPoliticWords	It is a binomial attribute. We have generated a list of politic words, which consist of political party names, politician names, some politic keywords etc. This attribute is true if the tweet contains any of these politic words, it is false otherwise.

Table 4.3: Regular Attributes of the Learning Model

We defined 3 special attributes (labels) to be predicted which are *slope*, *numberOfTotalRTs*, *numberOfDaysForSaturation*. *numberOfDaysForSaturation* indicates the number of days spent until the related tweet starts not getting any retweets. As it can be noticed, although we have 3 different labels, all labels are derived from the "number of retweets" information.

And 5 regular attributes to be used as feature for learning

- *containsHashtag*
- *containsLink*
- *numberOfFollowers*
- *numberOfCapitalLetters*

- *containsPoliticWords*

1. ***slope:*** After we calculated slope of the each top 1000 retweeted tweets, then in order to differentiate them into two classes (high slope, low slope) we applied The 68-95-99.7% Rule. As this rule implies in a normal distribution, approximately 68% of the observations fall within 1 standard deviation (σ) of the mean (μ) (See Figure 4.7[5]) [55].

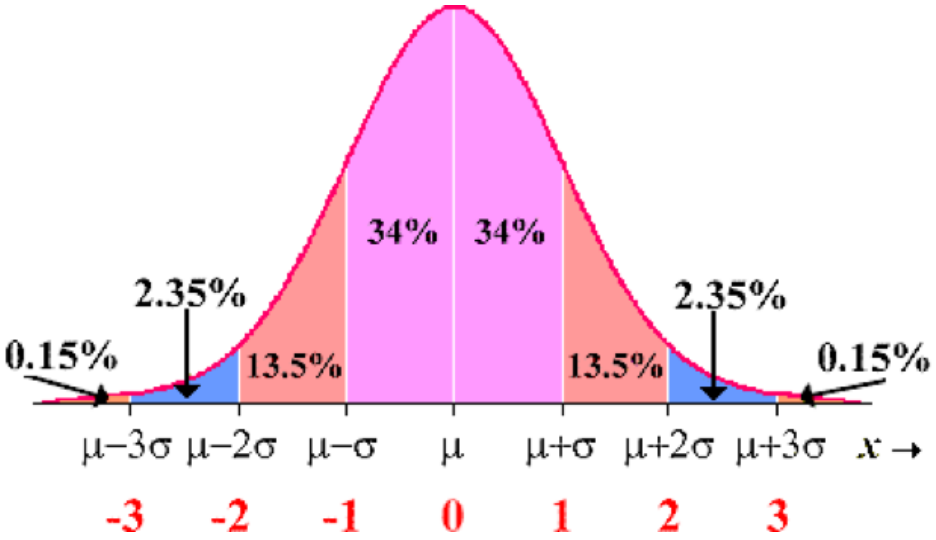


Figure 4.7: Distribution of the Observations wrt μ and σ

Therefore, we treated the tweets whose slope is higher than $\sigma + \mu$ as *high* class (there were 91 of them), while the other ones are *low* class. Finally, we came up with a data with 1000 samples, and this data has 5 regular attributes (as explained in Table 4.3) and 1 class attribute (slope as *high* or *low*). According to 68-95-99.7% Rule, we got 91 *high*, 909 *low* values among total of 1000 tweets. Thus we selected all 91 *high* tweets and randomly selected 91 *low* tweets for balancing the size of each class. Otherwise, there could be a bias on the label which has bigger data size. By using the 5 regular attributes mentioned above, we operated cross validation technique in RapidMiner[6] with two different classifiers to see the results

[5]Figure 4.7 retrieved from `http://www.oswego.edu/~srp/stats/images/normal_34.gif` on May 15, 2016

[6]An open source data science software platform `https://rapidminer.com/`

(Table 4.4).

	Naive Bayes			K-NN (k = 22)		
	True High	**True Low**	**Class Precision**	**True High**	**True Low**	**Class Precision**
Predicted High	39	31	55.71%	61	31	66.30%
Predicted Low	52	60	53.57%	30	60	66.67%
Predicted Total	91	91		91	91	
Class Recall	42.86%	65.93%		67.03%	65.93%	
Accuracy	54.36 % +/- 8.97% (mikro: 54.40%)			66.52% +/- 8.53% (mikro: 66.48%)		

Table 4.4: Confusion Matrix on Predicting *low* and *high* Labels for *slope*

Results indicate that K-NN performed better than Naive Bayes, with 66.52% accuracy.

2. ***numberOfTotalRTs:*** We managed to obtain 26 *high* and 26 *low* tweets as balanced dataset after 68-95-99.7% Rule. Once again we used the same classifiers and the results are given in Table 4.5:

	Naive Bayes			K-NN (k = 22)		
	True High	**True Low**	**Class Precision**	**True High**	**True Low**	**Class Precision**
Predicted High	14	2	87.50%	12	0	100.00%
Predicted Low	12	24	66.67%	14	26	65.00%
Predicted Total	26	26		26	26	
Class Recall	53.85%	92.31 %		46.15%	100.00%	
Accuracy	72.00% +/- 18.33% (mikro: 73.08%)			72.00% +/- 16.00% (mikro: 73.08%)		

Table 4.5: Confusion Matrix on Predicting *low* and *high* Labels for *numberOfTotalRTs*

They both produced same accuracy result that is 72% for *numberOfTotalRTs*.

3. ***numberOfDaysForSaturation:*** We managed to obtain 134 *high* and 134 *low* tweets as balanced dataset after 68-95-99.7% Rule.

   At this time, K-NN performs slightly better than Naive Bayes with almost 4% overall accuracy difference (Table 4.6). The k values for K-NN classifier had been

	Naive Bayes			K-NN (k = 50)		
	True High	**True Low**	**Class Precision**	**True High**	**True Low**	**Class Precision**
Predicted High	25	11	69.44%	69	44	61.06%
Predicted Low	109	123	53.02%	65	90	58.06%
Predicted Total	134	134		134	134	
Class Recall	18.66%	91.79%		51.49%	67.16%	
Accuracy	55.24% +/- 5.07% (mikro: 73.08%)			59.27% +/- 5.54% (mikro: 59.33%)		

Table 4.6: Confusion Matrix on Predicting *low* and *high* Labels for *numberOfDaysFor-Saturation*

selected depending on some number of experiments such that it will produce the best accuracy results.

4.2.5 Cluster Analysis

Different techniques can be used to specify the class labels (*high* and *low*) for special attributes. For instance, we just analyzed distribution of the values in the special attribute. Then, we applied 68-95-99.7% Rule after calculating the mean and the standard deviation of the distribution to distinguish the tweets as *low* and *high*. Instead of doing this, a clustering technique can be used to separate them into two different groups by calculating some similarity measure between each of them.

Remember that we applied Linear Regression Method for each tweet to analyze the dispersion of the tweet by fitting it into a linear line. These lines can be thought as vectors that actually represent the tweets. As it is stated in [75], the angle between two vectors is used as a measure to find similarity between the vectors, and cosine of the angle can be used to calculate the numeric similarity where it becomes 1.0 for identical vectors and 0.0 for orthogonal vectors (Figure 4.8).

Agglomerative clustering is a type of hierarchical way that has a “bottom up approach”. In this technique, each data point composes one cluster at the beginning and then pairs of clusters are merged as moving up hierarchy depending on their distance metrics [46]. We set the mode of the distance measure as “Single Linkage” which means the

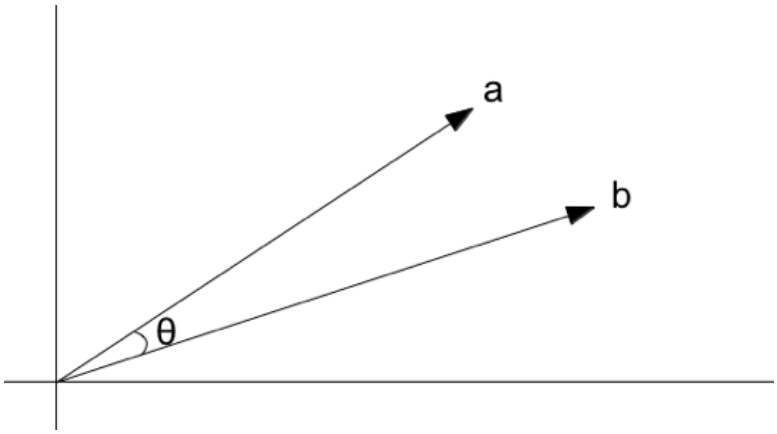


Figure 4.8: Cosine of the angle between two vectors

closest pair of elements that belong to different clusters is taken into consideration while merging clusters. Moreover, "cosine similarity" was used as similarity metric. However, after we had obtained the results, when we looked into two top clusters (as we want to group them into two clusters), we realized that they had 4 and 996 observations respectively, which is quite unbalanced.

For that reason, we switched to k-means (where k is 2) clustering to group them into two different clusters. Again "cosine similarity" metric had been applied as distance metrics among vectors. At that time, we manage to have more balanced clusters as the first one (cluster_0) has 654 items and the other one (cluster_1) has 436 items. Now, we have 5 regular attributes as explained below and 1 special attribute which is cluster_0 or cluster_1. The overall accuracy of the cross validation with k-NN model (where k is 50) is 61.9% (Table 4.7).

4.2.6 Modifying *low* and *high* Labels

Although we could not manage to distinguish each classes depending on their regular attributes with good accuracies, maybe differentiating mostly retweeted tweets can be questioned. They all already drew attention of people, had lots of retweets, thus there might not be major characteristic differences between each of them, even if some of them had been retweeted little bit more than the others. Additionally, since our main purpose

	True Cluster_1	True Cluster_0	Class Precision
Predicted Cluster_1	160	105	60.38%
Predicted Cluster_0	276	459	62.45%
Predicted Total	436	564	
Class Recall	36.70%	81.38%	
Accuracy	61.90% +/- 3.96% (mikro: 61.90%)		

Table 4.7: Confusion matrix on predicting clusters

is to understand the content-based characteristic differences of highly retweeted tweets, it would be better to include the tweets with low retweet numbers to our learning model.

Consequently, we composed our training dataset in a way that it contains mostly retweeted 1000 tweets where each of them has 60 retweets at least and 9973 retweets at most. The dataset also contains additional 1000 random tweets where each of them has 5 retweets at most. In other words, we now have total of 2000 tweets and it has equal size of bins for highly retweeted ($>$ 60 RT, we set their label as *high*) and lowly retweeted ($\leq$ 5 RT, we set their label as *low*) tweets. Again, by using our 5 regular attributes we composed our learning model by using k-NN (k is 20) and tested this model with cross validation method as explained before. The overall accuracy of the model becomes 83.8% with this dataset as shown in Table 4.8.

	True High	True Low	Class Precision
Predicted High	863	187	82.19%
Predicted Low	137	813	85.58%
Predicted Total	1000	1000	
Class Recall	86.30%	81.30%	
Accuracy	83.80% +/- 2.14% (mikro: 83.80%)		

Table 4.8: Confusion Matrix on Predicting *low* and *high* Labels with 2000 Tweets

It can be noticed that, not the only overall accuracy is quite good, but also precision

and recall values for each class are also satisfactory.

If we increase the RT difference between *high* and *low* classes in a way that we selected the mostly retweeted 576 tweets as *high* (each of them has 105 retweets at least) and lowly retweeted 548 tweet as *low* (each of them has 2 retweets at most), then we executed k-NN (k is again 20) classifier on this data. The overall accuracy reached to 85.7% as given in Table 4.9.

	True High	**True Low**	**Class Precision**
Predicted High	495	79	86.24%
Predicted Low	81	469	85.27%
Predicted Total	576	548	
Class Recall	85.94%	85.58%	
Accuracy	85.76% +/- 3.95% (mikro: 85.77%)		

Table 4.9: Confusion Matrix on Predicting *low* and *high* Labels for More Polarized Classes

As a result, it is possible to say that as long as the retweet difference increases among the tweets; they can be more accurately distinguished by looking 5 attributes we have obtained (*containsHashtag*, *containsLink*, *numberOfFollowers*, *numberOfCapitalLetters*, *containsPoliticWords*).

4.2.7 Identifying Class-Specific Terms

In information retrieval, if text data is being analyzed as we do here, some specific tokens (terms) give more information regarding their class. As it is stated in [63], topics (classes) are typically identified by finding the special words that characterize that class.

We thought that if we determine class-specific terms in our dataset and use them as another feature; then it may give us some information to make our prediction results better. In other words, we extracted important terms for *high* and *low* classes respectively based on *tf-idf* scoring measure; then add two extra features to the tweets which are

- *containsHighRelatedTerms*: it is a boolean value *true* or *false*
- *containsLowRelatedTerms*: it is a boolean value *true* or *false*

After adding these two features to the training set that has 2000 samples (1000 *low* - 1000 *high*), we used same classifier k-NN (k is 20) again. We observed that our accuracy almost did not change and stayed as 83.8% (as in Table 4.8). However, when we added these two features to the other training set (548 *low* - 576 *high*), there was an increase in the overall accuracy that climbs to 86% (Table 4.10) from 85.7% (Table 4.9).

	True High	**True Low**	**Class Precision**
Predicted High	488	68	87.77%
Predicted Low	88	480	84.51%
Predicted Total	576	548	
Class Recall	84.72%	87.59%	
Accuracy	86.12% +/- 2.57% (mikro: 86.12%)		

Table 4.10: Confusion Matrix After Adding *tf-idf* Related Features

In the previous experiment, we had tweets that have more than 60 retweets and less than 5 retweets. However, there are other tweets that stays in the middle where their retweet numbers between 5 and 60. We decided to add 1000 random tweets that have retweet numbers between 5 and 60 to our training set. We also set their class label as *low* (we treated them same as the lowly retweeted tweets). As a result, we had a training set with size 3000 (2000 of them were *low* and 1000 of them were *high*). Actually, the number of low retweeted tweets is much higher than high retweeted tweets in real life, thus it became a more realistic training data. In other words, this is also reflective of the real tweet ecosystem. The confusion matrix on this dataset with k-NN (k is 20) is given in Table 4.11.

Overall accuracy of 75% on this dataset seems good, we are still able to predict whether a tweet will get high number of retweets in good accuracy. One may say that *numberOfFollowers* is the dominant regular attribute that is directly related to retweet

	True High	True Low	Class Precision
Predicted High	612	359	63.03%
Predicted Low	388	1 641	80.88%
Predicted Total	1 000	2 000	
Class Recall	61.20%	82.05%	
Accuracy	75.10% +/- 2.38% (mikro: 75.10%)		

Table 4.11: Confusion Matrix After Adding *tf-idf* Related Features with 3000 Tweets

number. Actually, 800 of 2000 *low* labeled tweets have the *numberOfFollowers* value as greater than 10000 (but of course they did not get high number of retweets) and we were still able to predict them in a correct way. In other words, our learning model does not say that a tweet with high *numberOfFollowers* will directly have high number of retweets. In fact, this is one important point which is quite good for the results.

We observed that some characteristics of tweets lead to extensive retweeting. Accordingly, we can conclude that tweets containing political words are more likely to amass highly retweeting when compared to tweets without political words. Our findings indicate that 32% of the highly retweeted tweets contained political words, while only 27% of the lesser retweeted tweets contained political words. Therefore, determining whether a tweet contains political token (word) is identified as a distinctive feature in predicting the likelihood for high retweeting. Note that we labeled 1000 tweets as *high* which had more than 60 retweets (more specifically, number of retweets for each tweet we labeled *high* ranged from 60 to 9,973). Additionally, tweets containing a hyperlink tend to be highly retweeted by Turkish Twitter users. Moreover, containing class-specific terms (*containsHighRelatedTerms* and *containsLowRelatedTerms*) are other features in predicting the likelihood for highly retweeting.

This work presents an analysis of tweets in exploring the reflections on Syria conflict among the Turkish public. Unlike most of the Twitter research focusing on digital activists who are mostly a narrow group of people engaging online for social change, we contribute to the literature by providing a wider perspective within public.

Our analysis reveals that armed fighting, religious and political sensitivities within the Turkish public inflate the volume of posted Tweet during the Syrian conflict. In addition, tweets containing Syria have not been only posted by a small specific group of people, but indeed attracted new individuals to post. The topic of Syria is reflected upon by a wider range of individuals.

A predictive model with 86.12% (Table 4.10) accuracy is built to classify *high* and *low* tweets based on the number of retweets received via seven features, namely *numberOfFollowers*, *containsHighRelatedTerms*, *containsLowRelatedTerms*, *containsPoliticWords*, *containsLink*, *numberOfCapitalLetters*, and *containsHashtag*.

Our retrieved data by Twitter Stream API has some limitations. Twitter does not permit to retrieve of all tweets, but instead allows streaming of only a small fraction of the total volume of tweets at any given moment. Therefore, our findings are only based on the tweets within this limitation. Another limitation that comes with tweet data is related with the bot accounts on Twitter. A twitter bot is an account, which is organized to post automatic tweets by using a software program, is not a real user. In our analysis, we checked top 500 most retweeted tweets' users and confirmed that real individuals, celebrities, media organizations and/or state institutions posted them. Hence the general limitation of bot accounts does not apply to our findings. At least, it does not apply to tweets which were labeled as *high* in our experiments.

4.3 Impact Prediction with CovNets

In section 4.2, we converted our data into a structured form by extracting some number of features. These features consist of tweet based features (*containsLink*, *numberOfCapitalLetters*, *containsHashtag*, etc), also contains user based feature like *numberOfFollowers*. We also wanted to analyze the same problem (predicting whether a tweet will have high number of retweets in a specific domain) with a different approach. This approach is text categorization with Convolutional Neural Networks which was explained in subsection 3.4.7. For these experiments, we collected another dataset about Soma mining dis-

aster, and details about the data and the methodology are given in the following sections. We want to know whether we can correctly predict the labels by just using tweets' text fields with CovNets.

4.3.1 Overview of the Data

Soma mining disaster, which killed 301 workers, happened on 13^{th} May 2014 [56]. We started to collect publicly available tweets provided by Twitter Streaming API that contain "soma" keyword from 13^{th} May 2014 to 23^{rd} March 2015. During this period of time, we managed to retrieve 6.329.038 tweets in our database. According to our observations, when we look at the daily number of tweets sent by the users in Figure 4.9, Soma disaster did not have a continuous effect on Twitter.

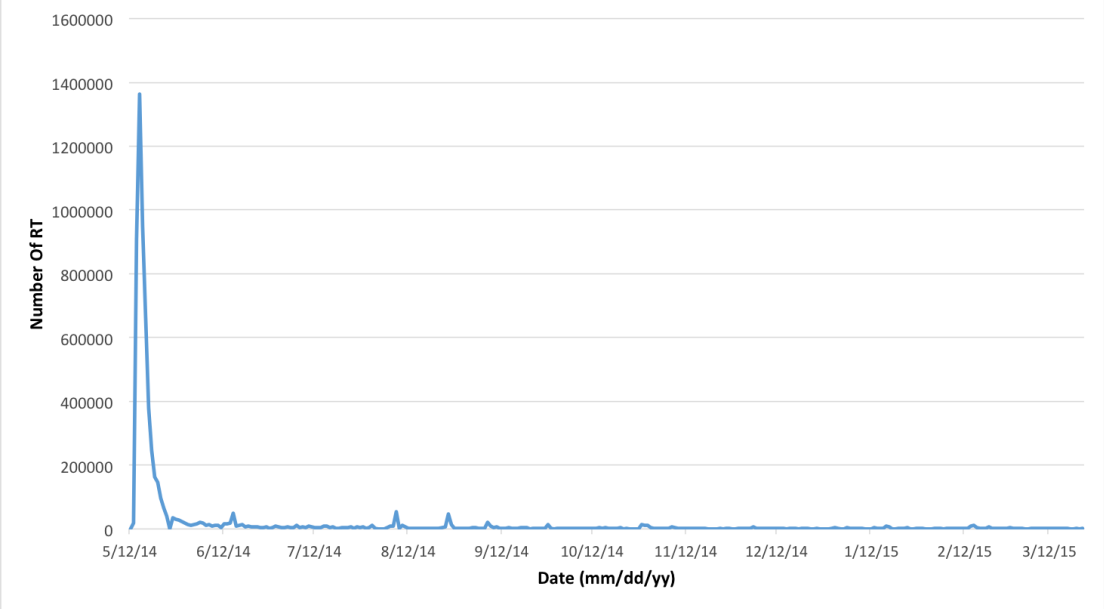


Figure 4.9: Daily Number of Tweets (13 May 2014 – 23 March 2015)

A similar characteristic can also be observed for the daily number of distinct users who sent tweets about Soma in Figure 4.10. It has almost the same characteristic with daily number tweets with their peak times, decreasing/increasing trends, etc.

Investigating the cumulative number of unique users during the specified time range in Figure 4.11 might also give some insights about the data. There is a dramatic increase

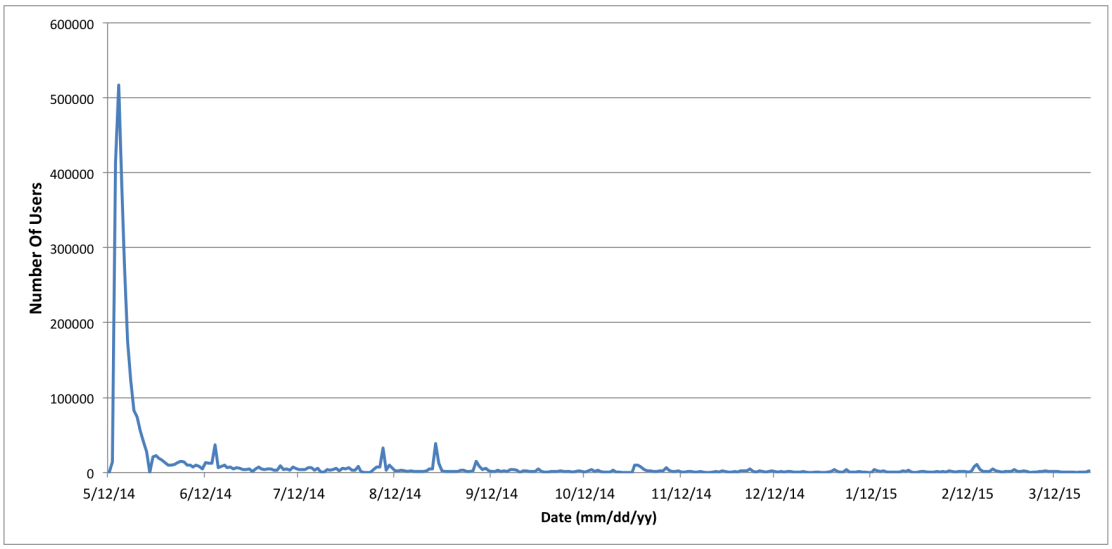


Figure 4.10: Daily Number of Users (13 May 2014 – 23 March 2015)

for cumulative user number in the first week and then this acceleration slows down as the number for tweets sent in this time period decreases. In other words, we can say that all these tweets especially in the first week were not sent by same group of people. Instead, different people got involved in for the spreading the topic over Twitter, which means this disaster had an important impact on huge number of people.

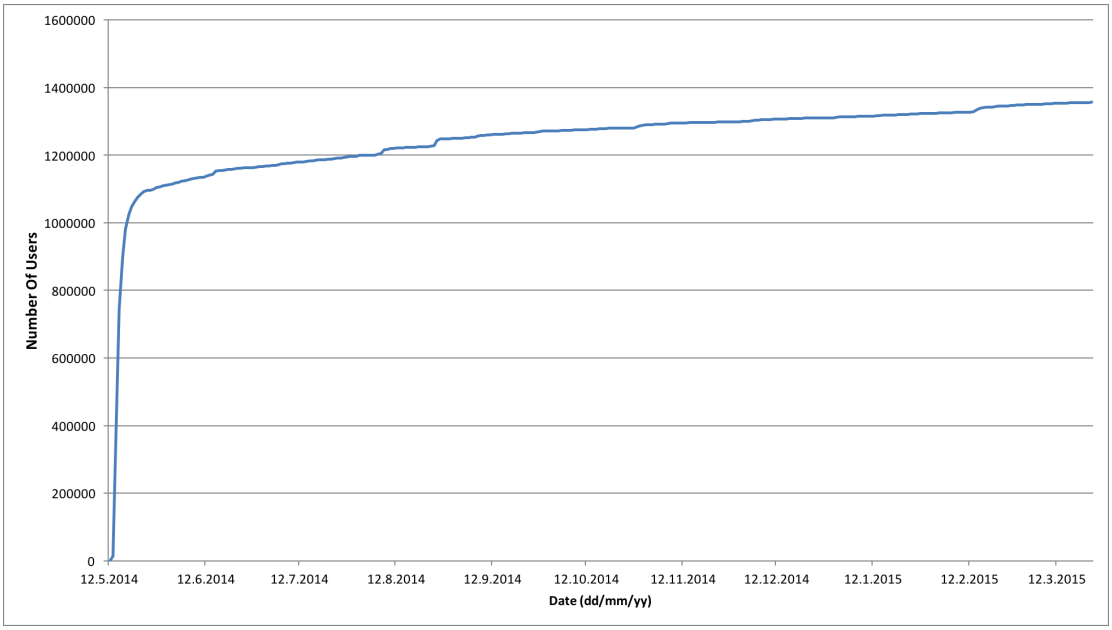


Figure 4.11: Cumulative Number of Users (13 May 2014 – 23 March 2015)

Lastly, pattern of tweet numbers also affected the spread of the most retweeted tweets. Differently from Syrian data characteristics shown in Figure 4.4, the top tweets mostly spread in the very beginning of the time range (Figure 4.12).

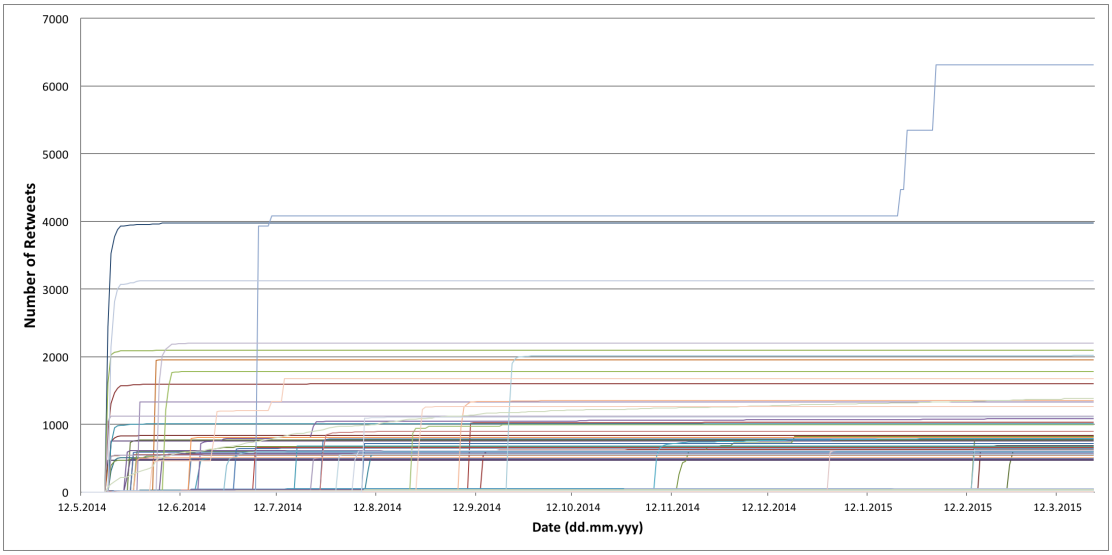


Figure 4.12: Soma Spread and Fade out Patterns of Top Tweets

4.3.2 CovNets Experiments

We have used 30K tweets which contains the tweets with the most number of retweets and the tweet with the least number of retweets in our experiments. We labeled the data as *high* and *low* in the same way we explained before. Then, we randomly split our data as training and test data that their sizes are 25K and 5K respectively. The CovNet model we created in TensorFlow[7] is given in Figure 4.13.

To give the more detail about the schema in Figure 4.13, we first convert our tweets into the sequence of word embeddings. We have specified the maximum sequence length as 25. It means that each tweet document will contain 25 word embedding vectors. Then, we apply 3 different convolution operations with different filter sizes. The filter sizes are 3, 4, and 5 respectively. As non-linear function, we preferred ReLUs and also applied L2

[7]An open source software library for Deep Neural Networks by Google `https://www.tensorflow.org/`

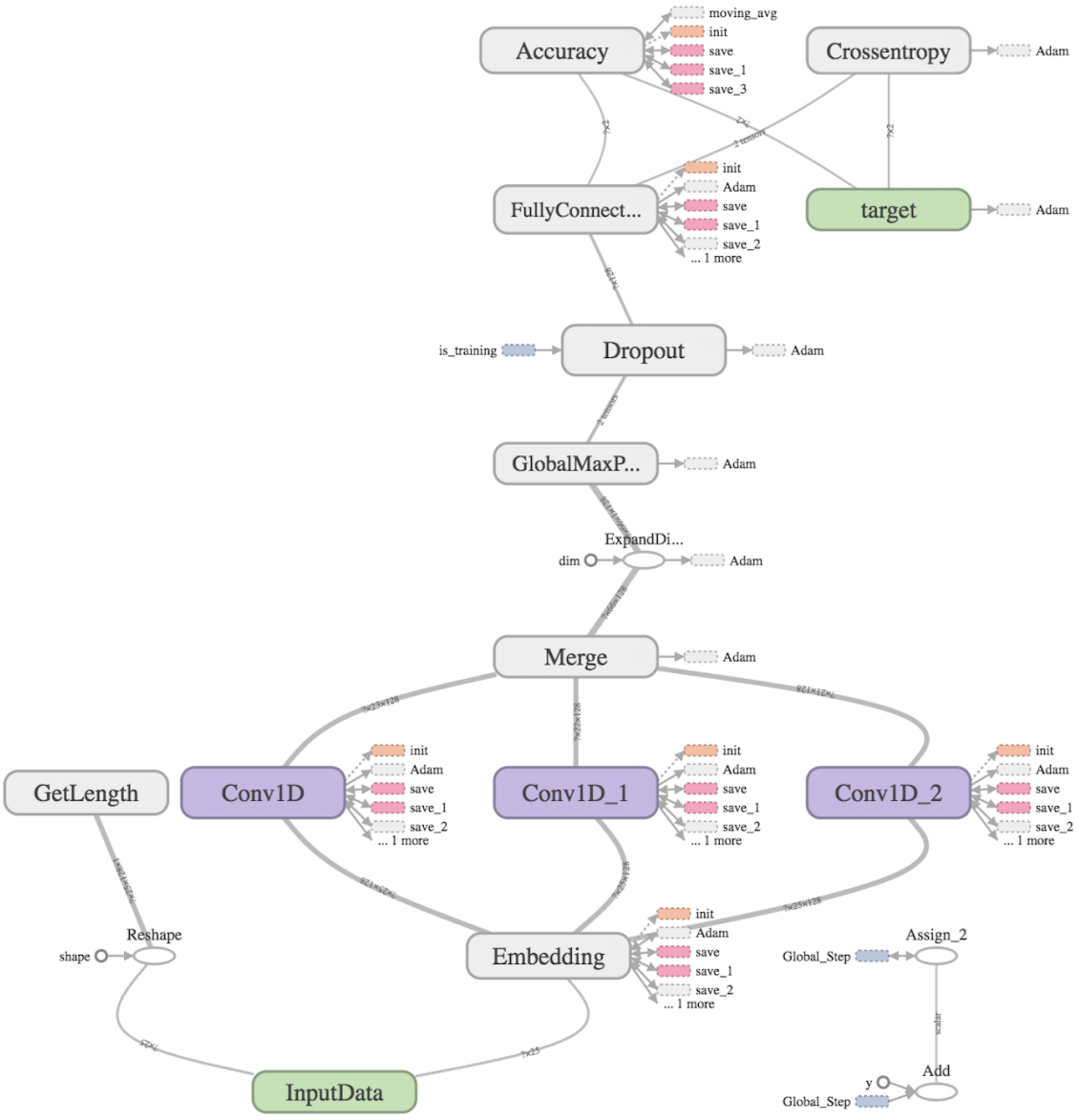


Figure 4.13: CovNet Model Architecture

regularizer. Our choice is max pooling method for the pooling operation, and parameter for the dropout operator was 0.5. That means, half of the units is dropped between each layer. In the last part, we have a fully connected layer that uses "Adam" optimizer and specifies 0.001 as learning rate. The batch size for optimizer is 32 which means that every step of the calculation of the loss function uses randomly selected 32 examples. Surely, we have two outputs *high* and *low* to be predicted, therefore our problem is a binary classification.

Our results were demonstrated in TensorBoard[8] which is a visualization tool that makes it easier to interpret TensorFlow programs. In Figure 4.14, the accuracy of the training data is presented. As it was expected, the learning model tends to learn the training data really well, with over 95% accuracy after some point. When we look at the results of the test data, we see big oscillations at the beginning. But after 16Kth example, the oscillations start getting smaller and the accuracy fits in range between 65% and 73%.

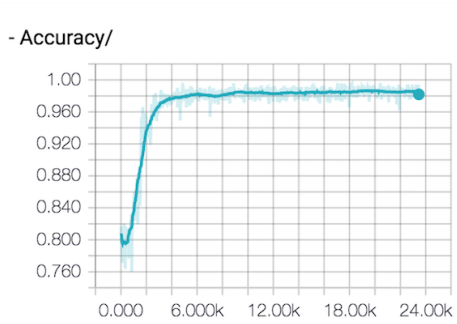


Figure 4.14: Accuracy of Training Data

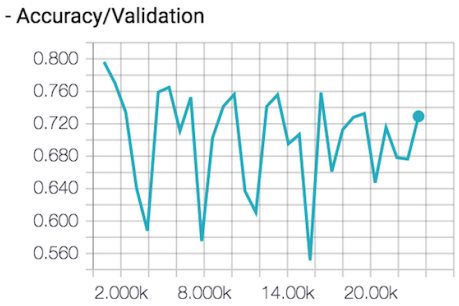


Figure 4.15: Accuracy of Test Data

4.3.3 Generalization of the Proposed Approach

In this work, we collected almost 450K tweets related to "Syria" topic and then conducted some experiments on this data. As we pointed out, *containsPoliticWords* is a binomial attribute that completely depends on whether a tweet contains any pre-determined political words. Since "Syria" is a political topic, it is reasonable to use *containsPoliticWords* attribute. However, if we work on sports domain, then this attribute will not have an good

[8] `https://www.tensorflow.org/get_started/summaries_and_tensorboard`

impact on the accuracy. Therefore, we may need to change this feature as *containsDomainRelatedWords* such that "domain" may become politics, sports, entertainment, art, etc.

One other important point is that to use time information. For instance, if we want to make a prediction on a tweet related to OWS (Occupy Wall Street) movement, then the tweets sent in September 2011 and the tweets sent today might not have the same output. We collected the tweet related to "Syria" in the time range that was quite hot for Syria topic. Thus, above-mentioned features and classifiers performs good for these tweets. However, they might not have the similar effect on any tweet contains "Syria" keyword which was sent before the Syrian conflict.

Chapter 5

Impact Assessment of Tweets

In chapter 1, we stated that identification of *Hidden Retweets* is quite important to have a more accurate impact knowledge of tweets. We also defined this problem as document clustering since we try to group similar tweets whose similarity score is above a threshold. Note that, hidden retweets are not necessarily exact identical tweets as they might have small modifications or extra comments. In order to group these similar tweets, we use lexical clustering where textual data items are grouped together based on a lexical similarity metric. By clustering tweets, it is possible to obtain a cleaner dataset containing only singular tweets representing a large group of tweets with similar content; which also reduces the time for more complex data processing algorithms. However, standard document clustering algorithms cannot be directly applied to tweets, because tweets have two distinct characteristics which differentiate them from standard documents such as blogs, news etc. First, tweets are very short, therefore standard document clustering algorithms which use word-based similarity metrics will not work well with tweets. Second, Twitter has no writing format, people can use informal language, emoticons, abbreviations in their tweets and there might be misspellings in the tweets as well. As a result, Twitter needs a specific clustering methodology based on lexical clustering to identify similar tweets in terms of content.

In this chapter, we first compare two lexical tweet clustering techniques. The first clustering technique is lexical threshold based clustering using Longest Common Sub-

sequence (LCS) similarity metric, which we call *LCS-Lex*). We consider *LCS-Lex* as a benchmark for constructing high quality clusters of tweets since we observe that *LCS-Lex* clustering is quite effective in constructing high quality clusters. However the high computational complexity of *LCS-Lex* makes it ineffective to deal with a large number of tweets. The second technique we propose (*ST-TWEC*) is based on generalized suffix trees. In order to show the benefits of suffix tree based clustering algorithm, we have experimented with Twitter data collected on different topics. We compared these two algorithms and results show that *ST-TWEC* performs well in terms of time complexity. Additionally, we conducted further experiments with a state of the art k-means based document clustering algorithm in terms of time performance and cluster qualities.

After showing *ST-TWEC* performs much better than *LCS-LEX* and traditional k-means based document clustering algorithm in section 5.2, we also developed three different versions of DBSCAN algorithm. First algorithm is classical DBSCAN version with the distance measure as LCS. In the second algorithm we integrated suffix tree, and in the third algorithm we integrated LSH; as suffix tree and LSH are efficient nearest neighbour finding methods that dramatically affect time performance of DBSCAN.

5.1 Methodology

We propose several different methods for tweet clustering in this work. In both methods, our aim is to create a set of clusters $C = \{c_1, c_2, ..., c_m\}$ for given a set of tweets $T = \{t_1, t_2, ..., t_n\}$ where $m \leq n$. Proposed methods are explained in the following subsections.

5.1.1 LCS-Lex: Longest Common Subsequence Based Lexical Clustering of Tweets

LCS-Lex is based on Longest Common Subsequence (LCS), and we use the normalized LCS score defined in Equation 5.1 for similarity calculation between two tweets.

$$NormalizedScore(t_i, t_j) = \frac{2 * LCS(t_i, t_j)}{Length(t_i) + Length(t_j)} \tag{5.1}$$

Algorithm 2: LCS based tweet clustering algorithm

```
C = {};
for i ← 1 to n do
    t_i.isClustered ← false;   ▷ Initially, all tweets are marked as unclustered;
end
for i ← 1 to n do
    if t_i.isClustered = false then
        c = {i};
        for j ← (i + 1) to n do
            if t_j.isClustered = false and NormalizedScore(t_i, t_j) ≥ threshold
             then
                c ← c ∪ {j};                    ▷ Find first unclustered similar tweet
            end
        end
        if |c| ≥ k then
                                                ▷ If cluster size is big enough
            C ← C ∪ c;                          ▷ Add this cluster to set of clusters
            foreach index ∈ c do
                t_index.isClustered ← true;     ▷ Mark these tweets as clustered
            end
        end
    end
end
```

A naive algorithm for LCS based tweet clustering process is given in Algorithm 2. In this algorithm, for each unclustered tweet (referred as tweet i), we pass over all other following unclustered tweets (referred as tweet j). If the normalized score that we defined above, between tweet i and tweet j is higher than the threshold, then we include indexes

of these tweets (which are i and j) in the same cluster c. However, not all clusters are included into the set of clusters C; instead only the clusters whose sizes are greater than or equal to k are included into C where k is the minimum number of tweets a cluster must have. In this case, the first tweet (t_i) in c becomes the representative tweet of cluster c. Once we include the cluster c into the set C, then we remove the tweets included in cluster c from the dataset. This algorithm ensures that the cluster sizes are greater than or equal to k.

For any $c_m \in C$, let's call the first index in c_m as c_{m_0}. We guarantee that

$$\forall c_{m_i} \in c_m, NormalizedScore(t_{c_{m_0}}, t_{c_{m_i}}) \geq threshold$$

This means that every tweet which belongs to a cluster is similar to the representative tweet of that cluster more than the threshold. We should also note that there is no guarantee that the similarity between any two tweets in a cluster is above the threshold. However, experiments show that tweets belonging to even very large clusters are similar to each other in content as well as to the representative tweet.

There are two different parameters (*threshold* and *k*) in *LCS-Lex* to be defined by the user. In this work, we assumed that at least 2 tweets are required to compose a cluster. In other words, we define a cluster as group of tweets where the number of tweets in this group is at least 2. For that reason, we selected *k* as 2 in our experiments in section 5.2. For *threshold* parameter, we did not define any specific threshold, instead we tested performance (both time and cluster quality) of *LCS-Lex* with different *threshold* values.

Although LCS based clustering algorithm (*LCS-Lex*) performs well in terms of cluster qualities, the time performance is prohibitive for large number of tweets (even for tens of thousands). The complexity of Algorithm 2 is $O(N^2 * L^2)$ where N is the number of tweets ($|T|$), and L is the maximum tweet length which is 140 due to the characteristics of Twitter. Additionally, LCS-LEX is not a deterministic algorithm which means that the formation of the clusters is the depend on the order of the tweets in the dataset. If we shuffle the order of tweets, then the clusters can change.

5.1.2 ST-TWEC: Suffix Tree Based Tweet Clustering Method

ST-TWEC, which was firstly used in a thesis by Erpam [17], utilizes the generalized suffix trees in order to compose clusters. Since generalized suffix tree (GST) is able to find longest common substring between two strings very efficiently (in $O(m+n)$ time where n and m length of the strings). Differently from normal GSTs, for each node we keep the length of the total path from root to that node (say *nodeLength*). It is an important point of the algorithm which will be explained later. Additionally, we keep the ids of the tweets that contains the corresponding substring in the leaf nodes. We keep these information only in the leaf nodes due to the memory issues. In order to retrieve the ids belongs to a node which is a non-leaf node, we apply a bottom-up approach by merging ids. For this reason, we add a reserve link from each child node to its parent node that makes us able to reach any node from the bottom side. In other words, our GST becomes bidirectional.

Before constructing GST, we remove the punctuation marks, links, hashtags and usernames. We also transform all letters into lowercase and eliminate the tweets shorter than 5 characters. Then, we construct our GST with Ukkonen's algorithm in linear time. In this GST, every node (implicitly or explicitly) contains ids of the tweets that contains the related substring. We say implicitly or explicitly, since the leaf nodes explicitly contains these ids while other nodes will implicitly have this information with bottom-up approach. Note that same tweet id might occur in different leaf nodes in the tree. Every node v in our GST represents a cluster c. We check every single tweet t that belongs to v and add t into c if $score$ value which is defined below is greater than the user based threshold.

$$score = nodeLength_v \,/\, \mid t \mid$$

By this way, we guarantee that every tweet in the same cluster shares a substring and ratio of this substring to the length of each tweet is above the threshold. As we mentioned before, we apply a bottom-up approach starting from the deepest level of the tree. At the beginning, we keep ids in the leaf nodes; as we keep going to the upper levels, we merge these ids in the new level and remove them from the previous (lower) level.

Since the same tweet id might exist in different leaf nodes, it can also join different

clusters at the same time. In this case, we prefer to keep this tweet in the cluster with the largest size. In order to do that, we sort clusters according to their sizes in descendant manner. Then starting from the largest cluster, we label each unlabelled tweet (all the tweets are initially unlabelled). If we encounter a labeled tweet in any cluster, it means that it also exists one of the larger clusters so we remove it from this cluster. Any cluster remains with the size smaller than 2 is removed, since it is not a cluster anymore. Note that we define a cluster as group which contains at least two tweets.

All these processes perform really fast with GST. All the preprocessing steps are linear time works. Then, the construction of GST is accomplished with Ukkonen's algorithm again in linear time. Total number of characters cannot exceed $140 * N$ where N is the total number of tweets, therefore the complexity of construction of GST is $O(140N)$. In the cluster creation process for a specific node n, we calculate the $score$ value for each tweet which takes $O(\mid n \mid)$ time ($\mid n \mid$ refers to number of ids in that node n). Identifying ids for node n is also linear since this information is being merged from one lower level. After we sort all the clusters by their size, we pass through all tweets which takes $O(N)$ time where N is the total number of tweets. Only non-linear operation in the algorithm is sorting the clusters which takes $O(\mid C \mid log(\mid C \mid))$ time where $\mid C \mid$ is the total number of clusters. However, since $\mid C \mid << N$ (our experiments show that $\mid C \mid log(\mid C \mid) \cong N$) so we can also say that sorting process also takes linear time.

For the space requirements, since the total number of characters is at most $140 * N$; there are $O(2 * 140 * N)$ nodes. As we keep all ids in the leaf nodes at the beginning, and one specific tweet can have 140 suffixes at most. In other words, id of a particular tweet can exist in 140 leaves at most, therefore the space complexity for ids is $O(140N)$. We will also need some space for the clusters since they will be keeping ids of the tweets. In the worst case, each node creates a cluster and all ids in that node join to the cluster. We said that there can be $140 * N$ ids at most in the leaf level, and we know that as we reach to the top level in the tree which is root there will be exactly N ids. In the middle levels, there will be merges; however in the worst case there will be no merge until the root as in Figure 5.1. Note that the depth of the GST, say h, cannot be larger than the length of the

longest tweet which is 140. Therefore, the total space needed to keep ids in the cluster is $O((h-1)*140*N+N)$ where h is 140 in the worst case; and it makes a linear time complexity with a high coefficient, $O(19461N)$.

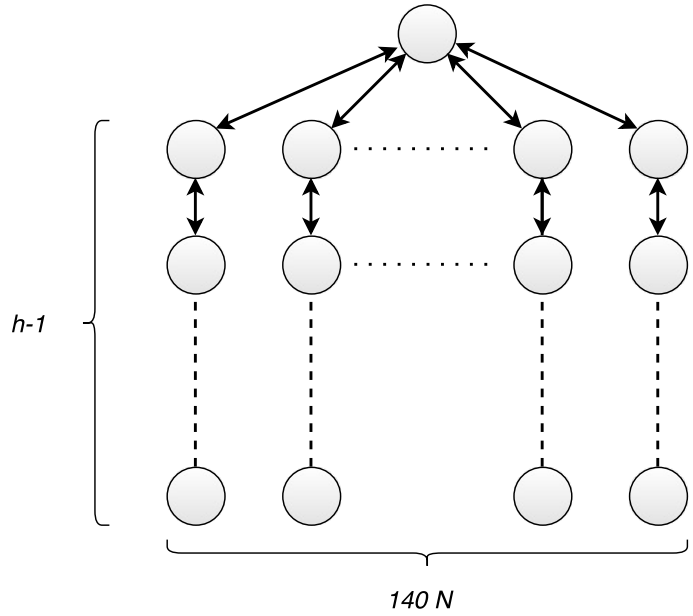


Figure 5.1: Worst Case Space Complexity of GST

.

5.1.3 K-means Document Clustering Method

We implemented the very basic k-means document clustering method where we represent documents with bag-of-words model and construct the feature vectors with tf-idf values. Then, we applied k-means clustering algorithm to compose clusters. Cosine similarity was used as distance function to calculate similarity between vectors.

5.1.4 LCS-DBSCAN: Classic DBSCAN with LCS

In this method, as we stated in subsection 3.6.2, we utilize classic DBSCAN algorithm and modified *regionQuery* function that finds similar tweets to a specific tweet t. For a given tweet t, we calculate the similarity between this tweet and all tweets; then retrieve the ones whose similarity score with t is greater than the threshold (see Algorithm 3).

LCS-DBSCAN (and all other DBSCAN based algorithms) are deterministic for core and outlier points. In other words, core points will be assigned to the same cluster regardless of the order of the data points. Although, this is not the case for border points and border points might be assigned to the different clusters depending on the order of the data; fortunately this situation does not occur frequently.

Algorithm 3: *regionQuery* function in *LCS-DBSCAN*

Parameters: $Tweet\ \ t, Double\ \ threshold$

$result = \{\}$;

for $i \leftarrow 1$ ***to*** N **do**

  if $NormalizedScore(t_i, t) \geq threshold$ **then**

    $result \leftarrow result \cup t_i$;

  end

end

return $result$;

5.1.5 ST-DBSCAN: DBSCAN Integrated with Suffix Tree

Again in this method, we modified *regionQuery* function and used GST to find similar tweets above a threshold (see Algorithm 4). Remember that, we have used a bottom-up approach in *ST-TWEC* and we keep ids of tweets only in the leaf nodes in the beginning. Then, as we climb towards upper levels, we merged the ids from lower lever to upper level and then removed them from lower level. This approach worked well in *ST-TWEC* since this operation is done only once. However, in *ST-DBCAN* we will have lots of queries to get the nearest neighbours (i.e to find similar tweets), we decided to keep ids of tweets in each level to reach them very fast. Differently from subsection 5.1.4, we have an additional suffix tree construction time which will be done only once (remember that this takes linear time with Ukkonen's algorithm), and then have a DBSCAN complexity with modified *regionQuery*.

Algorithm 4: *regionQuery* function in *ST-DBSCAN*

Parameters: $Tweet\ t, Double\ threshold$

$result = getSimilarTweetsFromGST(GST, t, threshold)$;

return $result$;

5.1.6 LSH-DBSCAN: DBSCAN Integrated with LSH

This method is quite similar to *ST-DBSCAN* method, but instead using GST in *regionQuery* function, we use Locality Sensitive Hashing to get similar documents. Remember from subsection 3.7.3 that we expect similar documents to produce the same hash codes; unfortunately this is not the case all the time. Although exact same documents always produce the same hash code, very similar (not exact same) documents may produce different hash codes. For that reason, we repeat the same process with L different hash tables to decrease the probability of missing a similar document. Surely, very different documents can end up with the same hash code as well (i.e they can end up in the same bucket). So that we need to compare tweet t with each tweet in the same bucket to check whether they are really similar in terms of their LCS (Longest Common Subsequence) similarity (see Algorithm 5). If we assume that the total number of buckets is $bucketNumber$, then $N/bucketNumber$ tweets in average occur in a bucket. In other words, we need to compare tweet t with $N/bucketNumber$ tweets in average. We need to repeat this operation for L hash tables, then the total number of comparisons becomes $L * N/bucketNumber$ in average (In *LCS-DBSCAN* we need N comparisons to find similar tweets). Note that, $bucketNumber$ changes depending on the length of the hash codes. If we use longer hash codes, it will cause to have more buckets which means faster processing. However, there is a trade off here. As we increase the number of buckets, we improve the time performance; but we also increase the probability of missing a similar document.

Algorithm 5: *regionQuery* function in *LSH-DBSCAN*

Parameters: $Tweet\ \ t, Double\ \ threshold$

$result = \{\}$;

for $i \leftarrow 1$ ***to*** L **do**

  $preResult\ =\ getSimilarTweetsFromMaps(HashTable_i, t.hashCode)$;

  for $j \leftarrow 1$ ***to*** $length(preResult)$ **do**

    if $NormalizedScore(preResult_j, t) \geq threshold$ **then**

      $result \leftarrow result \cup t_i$;

    end

  end

end

return $result$;

5.2 Experimental Evaluation

Experimental evaluation in this chapter consists of 3 main parts. In the first part (subsection 5.2.1), we compare *ST-TWEC*, *LCS-LEX*, and k-means document clustering methods and show *ST-TWEC* outperforms the others. In the second part (subsection 5.2.2), we compare density based approaches (*LCS-DBSCAN*, *ST-DBSCAN*, *LSH-DBSCAN*). One of the reasons why we separated the second part from the first part is that *LCS-DBSCAN* performs very poorly in terms of time (even worse than *LCS-LEX*), therefore we evaluated them with a smaller dataset. Additionally, it is very hard to demonstrate the results of so many methods in the figures. Finally, in the third part (subsection 5.2.3) we compare *LSH-DBSCAN*, *ST-DBSCAN* and *ST-TWEC* in terms of time performance with the same bigger dataset used in subsection 5.2.1.

5.2.1 Comparison of *ST-TWEC*, *LCS-LEX*, k-means

We evaluated both *LCS-Lex* and *ST-TWEC* using a combination of four datasets collected from Twitter Streaming API using hashtags. Our dataset consists of tweets related to Charlie Hebdo event (#jesuisCharlie), Christmas in 2016 (#christmas), NBA organisation

(#nba) and US President Trump (#trump). We limited the number of tweets of each dataset to 15K. Thus we have a total of 60K tweets for 4 different domains and before starting evaluations we applied some preprocessing on tweets like eliminating hashtags and transforming letters into lowercase. We experimented with these 60K tweets in order to compare the two tweet clustering algorithms (*LCS-Lex*, and *ST-TWEC*), however we also performed scalability experiments for *ST-TWEC* with much higher number of tweets which are reported in this section.

Given the dataset $D = \{t_1, t_2, ..., t_n\}$, we create a set of clusters $C = \{c_1, c_2, ..., c_m\}$ such that every cluster c_i contains at least k tweets. In the experimental evaluation, we set k to 2 and measure the intra-cluster similarity and cluster purity. Because we are focusing on the textual representations of tweets, we define the intra-cluster similarity measure based on LCS. Using Equation 5.1, we calculate intra-cluster similarity of a cluster by calculating the pairwise similarity of each tweet inside the cluster as:

$$intraCSim(c) = \frac{2}{|c| * (|c| - 1)} * \sum_{i=0}^{|c|} \sum_{j=i+1}^{|c|} NormalizedScore(t_i, t_j) \tag{5.2}$$

Equation 5.2 allows us to find the intra-cluster similarity of a cluster. Using the Equation 5.2, we find the average intra-cluster similarity in Equation 5.3 and the weighted average intra-cluster similarity of the cluster set in Equation 5.4. For the weighted average intra-cluster similarity measure, the weight of each cluster is proportional to the size of the cluster:

$$avgISim(C) = \frac{1}{m} * \sum_{i=0}^{m} intraCSim(c_i) \tag{5.3}$$

$$wAvgISim(C) = \frac{1}{\sum_{i=0}^{m} |c_i|} * \sum_{i=0}^{m} |c_i| * intraCSim(c_i) \tag{5.4}$$

Since we have collected four different datasets using four hashtags, we assumed that the tweets have four possible categories corresponding to each of the four hashtags. Therefore we assigned a label to each tweet depending on its hashtag. In total, we have

four labels represented by hashtags: #christmas, #nba, #trump, and #jesuischarlie. We use these labels as a gold standard to calculate the purity of clusters with Equation 5.5:

$$purity(C) = \frac{1}{\sum_{i=0}^{m} |c_i|} * \sum_{i=0}^{m} |max_{label}\ in\ c_i| \tag{5.5}$$

Time performance results with different similarity thresholds in terms of time performance, number of constructed clusters, number of unclustered tweets, average intra-cluster similarity, weighted average intra-cluster similarity, and purity are shown in Figure 5.2, Figure 5.3, Figure 5.5, Figure 5.6, Figure 5.7, Figure 5.8, and Figure 5.9 respectively.

LCS-LEX uses common subsequence and *ST-TWEC* uses common substring to determine cluster membership. Because of that, these two algorithms cannot be compared directly by using same thresholds. However, because a substring is a consecutive subsequence, *ST-TWEC* is expected to produce better clusters at lower thresholds compared to *LCS-LEX* which we have also observed in our experiments. We have verified this observation by experimenting with different datasets (in different domains) with different thresholds. For that reason, *LCS-Lex* had been experimented with the following thresholds: *0.5*, *0.6*, *0.7* and *0.8* and *ST-TWEC* had been experimented with the following thresholds: *0.3*, *0.4*, *0.5*, *0.6* and *0.7*.

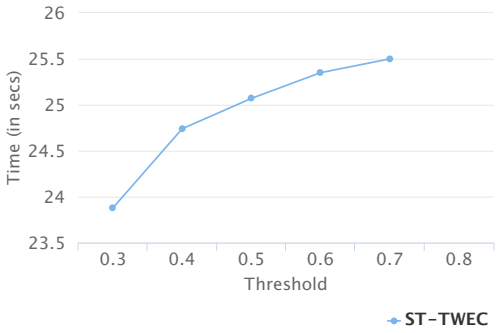


Figure 5.2: Time performance of *ST-TWEC* for 60K tweets with different thresholds

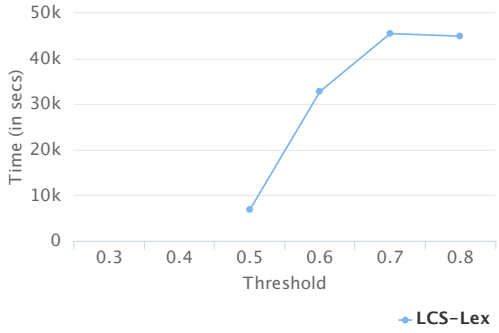


Figure 5.3: Time performance of *LCS-Lex* for 60K tweets with different thresholds

From the experiments we can easily observe that there is a dramatic time improvement

with *ST-TWEC*. The clustering time with *ST-TWEC* ranges from 23.8 to 25.5 seconds. On the other hand, the process with *LCS-Lex* ranges form 6825 to 45472 seconds (all the experiments were tested on a system with 32 processor, model name *Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz*, and 128 GB RAM). These two time performance results shown in Figure 5.2 and Figure 5.3 represent the time it takes for clustering processes with different methods in separate charts. It is worth noting that *ST-TWEC* is able to cluster 1 million tweets in about 1500 seconds. Figure 5.4 shows the time performance of *ST-TWEC* with increasing dataset sizes with a threshold of 0.4. We need to look at average intra-cluster similarity values to understand why we selected 0.4 as threshold in ST-TWEC. The average intra-cluster similarity values are 0.79, 0.84, 0.87, 0.89, 0.91 for thresholds 0.3, 0.4, 0.5, 0.6 and 0.7 respectively as it can be seen in Figure 5.7. Although greater threshold value means greater the average intra-cluster similarity, changing threshold from 0.3 to 0.4 made the biggest improvement. Note that increasing threshold value also increases number of unclustered tweets as it can be observed in Figure 5.6 and we want to keep this number low as much as possible. For that reason, 0.4 is reasonable value for threshold in our experiments.

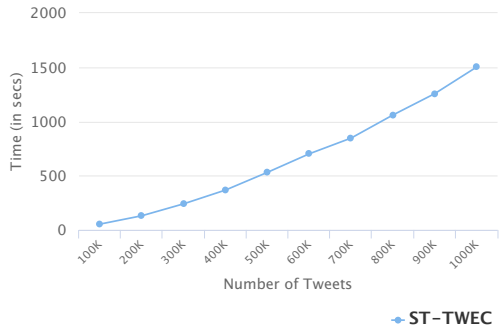


Figure 5.4: Time Performance of *ST-TWEC* with threshold 0.4

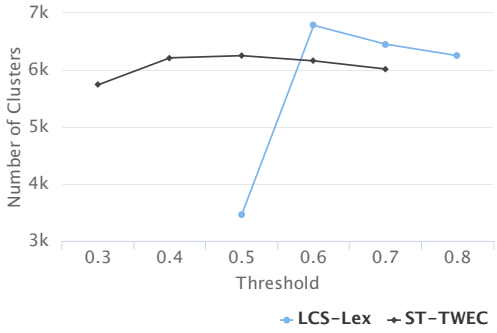


Figure 5.5: Number of clusters for 60K tweets with different thresholds

Other results given in Figure 5.5, Figure 5.6, Figure 5.7, Figure 5.8, Figure 5.9 vary for different thresholds, however we can observe that the difference in terms of clustering quality and the number of clusters produced is very low. In other words, similar number of clusters with roughly same intra-cluster similarities can be obtained by different

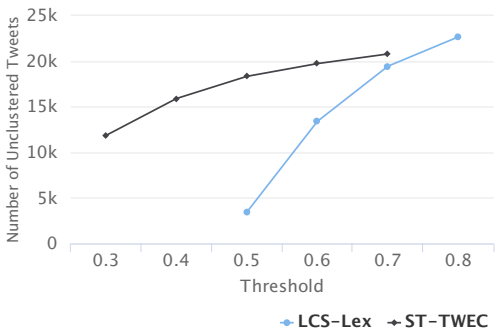


Figure 5.6: Number of unclustered tweets for 60K tweets with different thresholds

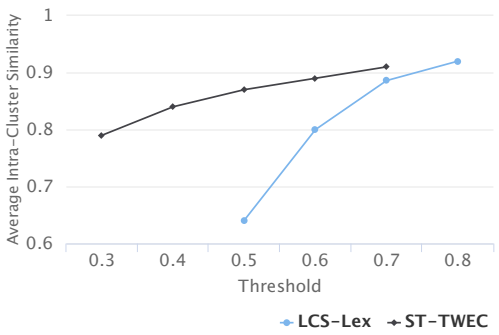


Figure 5.7: Average intra-cluster similarity for 60K tweets with different thresholds

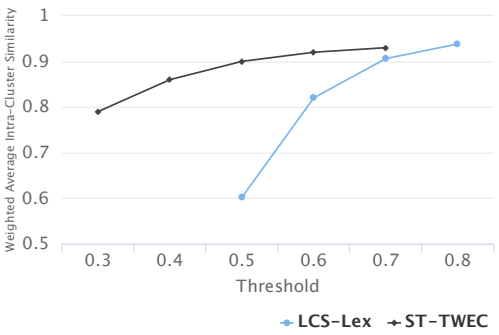


Figure 5.8: Weighted average intra-cluster similarity for 60K tweets with different thresholds

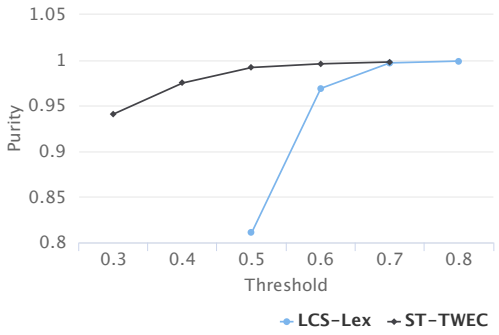


Figure 5.9: Purity for 60K tweets with different thresholds

thresholds of these two clustering methods. However, clustering time is the major factor that distinguishes these methods which proves the effectiveness of *ST-TWEC* for large datasets. Figure 5.6 shows that the number of unclustered tweets is between 10K and 20K which can be considered as a little bit high. However, we have used Twitter Stream API while collecting tweets regarding specific hashtags. Twitter gives a random sample of tweets, thus some of the tweets are not related to others (or have important difference) although they contain the same hashtag. For that reason, it is normal that some tweets are not grouped in any cluster. We have also inspected unclustered tweets and observed that unclustered tweets are not related to the composed clusters.

We have also conducted experiments to compare *ST-TWEC* and *LCS-LEX* with *k-means document clustering* in terms of performance and cluster qualities using Precision, Recall and F-Score. In order to apply k-means algorithm, we represented all tweets in vector space model with tf-idf values and used cosine similarity measure to find similarity between vectors. Since our data was collected using 4 different hashtags, we specified *k* value of k-means as 4. We have 60K tweets in our data set that resulted in a large high dimensional vector space representation causing poor time performance for k-means. Using k-means took 72013 seconds to complete clustering. Remember that *LCS-Lex* spends 6825 to 45472 seconds for different thresholds and *ST-TWEC* spends 23.8-25.5 seconds for different thresholds to cluster same data. Time performance of k-means can be improved by reducing vector dimensions, however this will affect the cluster qualities in a negative way. As we mentioned before, we compare cluster qualities with Precision, Recall and F-Score values as in Equation 5.6, Equation 5.7, and Equation 5.8 where *tp* represents "true positive", *tn* represents "true negative", *fp* represents "false positive", and *fn* represents "false negative".

$$Precision = \frac{tp}{tp + fp} \tag{5.6}$$

$$Recall = \frac{tp}{tp + fn} \tag{5.7}$$

$$F\text{-}Score = \frac{2tp}{2tp + fp + fn} \tag{5.8}$$

While analyzing k-means results, we assume that the most frequent real class label (hashtag) in any cluster is *true positive* for that cluster. Remember that we had many (much more than 4) clusters from *LCS-Lex* and *ST-TWEC* as stated in Figure 5.5. In order to make a good comparison with k-means, let's assume that there are 4 big clusters (referring to #jesuisCharlie, #christmas, #nba and #trump) as k-means has and each of these clusters belongs to one of the 4 big clusters depending on the most frequent real class label again. Comparisons of k-means, *LCS-Lex* (with different thresholds), and *ST-TWEC* (with different thresholds again) for each big cluster are given in Figure 5.10, Figure 5.11, Figure 5.12 and Figure 5.13 respectively. In these charts, for instance "*LCS-Lex (0.5)*" means *LCS-Lex* has been applied with threshold *0.5*.

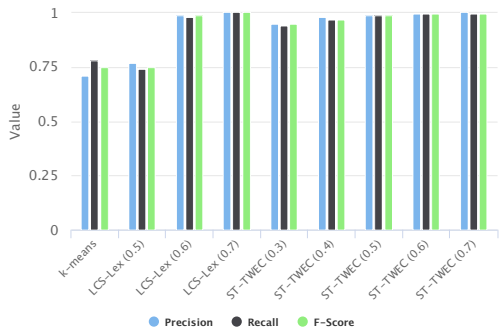


Figure 5.10: Precision, Recall and F-Score results for "#charlie" cluster

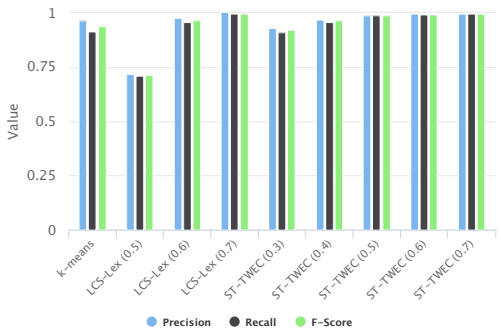


Figure 5.11: Precision, Recall and F-Score results for "#christmas" cluster

As it can be seen from Figure 5.10, Figure 5.11, Figure 5.12 and Figure 5.13; *LCS-Lex* and *ST-TWEC* mostly outperforms k-means document clustering algorithm in terms of Precision, Recall and F-Score values (except from *LCS-Lex (0.5)*). It is also worth to note that there are some unclustered tweets in *LCS-Lex* and *ST-TWEC* as shown in Figure 5.6 before, however k-means clusters all tweets in one of the 4 groups.

Spina et al. [78] stated that links, hashtags and named entities carry semantic content; and it might be a good idea to consider them as well for similarity analysis. Actually, it makes sense for their approach. However, in our work, we use a lexical approach for tweet

Figure 5.12: Precision, Recall and F-Score results for "#nba" cluster

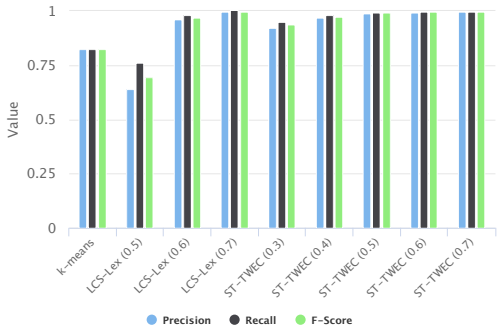


Figure 5.13: Precision, Recall and F-Score results for "#trump" cluster

clustering. In Twitter, links are shortened using t.co service and it creates lexically similar links with different contents. Using links increase lexical similarity between tweets which have different content and decreases the intra-cluster similarity overall. We agree that named entities are significant, however in Twitter, hashtags and also usernames may also be used out of context, especially in trend hashtags and popular usernames; therefore we decided to focus on purely on content. We also use hashtags for evaluation and pruning them removes any bias the selected hashtags may create in our work. We also examined ST-TWEC for the same data without removing links, usernames and hashtags in terms of Precision, Recall and F-Score; but results just slightly got better for lower thresholds or didn't change at all for higher thresholds. And the time performance went down to 42 seconds in average (remember that Figure 5.2 shows that when we prune links, usernames and hashtags, it takes 24-25 seconds in average).

The biggest limitation for the *ST-TWEC* is the memory requirement of the constructed suffix tree. As the tweet size grows up, the memory size consumed by suffix tree also increases. Our suffix tree consumed 475 MB memory for 60K tweets and its size proportionally changes with the total number of characters in the tweet data set.

5.2.2 Comparison of *LCS-DBSCAN*, *ST-DBSCAN*, *LSH-DBSCAN*

In this subsection of the experimental evaluation, we have used a smaller dataset as we stated before. This dataset contains 10K tweets which are randomly selected from the

dataset used in chapter 4. Since *LCS-DBSCAN* performs very poorly in terms of time, we preferred to use this smaller dataset for comparison. In this part, all the experiments were tested on a system with 2.8 GHz Inter Core i7 processor and 8 GB 1600 MHz DDR3 ram. Similar to the previous part, since *LCS-DBSCAN* and *LSH-DBSCAN* are based on Longest Common Subsequence, they had been experimented with the following thresholds: 0.5, 0.6, 0.7 and 0.8. On the other hand, *ST-DBSCAN* had been experimented with the following thresholds: 0.4, 0.5, 0.6, 0.7 and 0.8. One important note is that we have tested *LSH-DBSCAN* with different K and L values; and called all these with different names. For example, *LSH-DBSCAN-K4-L5* refers to "*LSH-DBSCAN* with K as 4 and L (number of hash tables) as 5". Shingle size in *LSH-DBSCAN* is set to 3 in all experiments (we have also tested all the experiments with shingle size as 2, but there was no dramatic difference). Remember that in DBSCAN algorithm, there are 2 parameters $minPts$ and ε. In all algorithms derived from DBSCAN, we set $minPts$ to 10 (we will explain why $minPts$ is selected as 10 at the end of this subsection) and ε value is equal to the *threshold* value in the experiments.

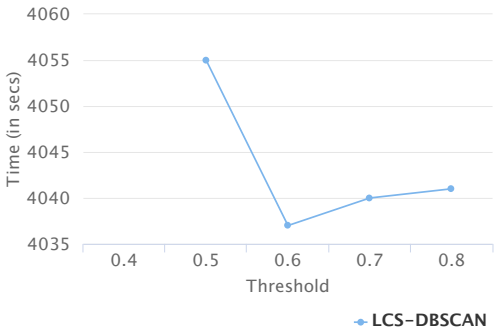


Figure 5.14: Time performance of *LCS-DBSCAN* for 10K tweets with different thresholds

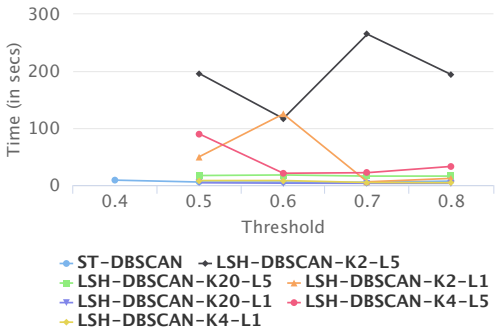


Figure 5.15: Time performance of other methods for 10K tweets with different thresholds

We demonstrated the time performance of *LCS-DBSAN* in Figure 5.14, others in Figure 5.15 since *LCS-DBSAN* performs poorly and increases the timing results such a way that results cannot be interpreted from a single figure. We observe that as the K increases the time performance of *LSH-DBSAN* increases as expected. If we increase K, there will

be more buckets (less collisions among tweets) so that the total number of comparisons will be lower which increases the time performance. The number of buckets in a hash table wrt K value in our experiments is shown in Table 5.1. Naturally, increasing L value means repeating the same process more times, so time performance decreases as well. Results in Figure 5.15 show that, *ST-DBSCAN*, *LSH-DBSCAN-K20-L5*, *LSH-DBSCAN-K20-L1*, *LSH-DBSCAN-K4-L1* perform good for each threshold. *LSH-DBSCAN-K4-L5* starts to perform fairly good after the threshold becomes 0.6 and *LSH-DBSCAN-K2-L1* starts to perform good after the threshold becomes 0.7.

K	**Number of Buckets**
2	[2100, 4200]
4	[5000, 5600]
20	[6000, 6250]

Table 5.1: Number of Buckets wrt. K value

In Figure 5.16, we showed a zoom in version of Figure 5.15 by selecting a subset of the methods. In Figure 5.17, the number of clusters wrt. different thresholds are shown. In high thresholds, all methods create almost the same number of clusters, but results vary with low thresholds. The reason behind results vary with lower thresholds is the effect of K and L parameters. *LCS-DBSCAN* does not miss any similar document so the number of unclustered tweets is lower and the number of clusters is higher. When K is 2, there are more tweets in a buckets that allows us to get more similar tweets. That's why when K is 2, there are more clusters with low thresholds. If we compare the different L values when K is constant (when K is 2 for example), higher L value causes more clusters as expected.

We have quite expected results in Figure 5.18. We observe that *LSH-DBSCAN* tends to cluster less number of tweets than *ST-DBSCAN* and *LCS-DBSCAN*. Especially, when K is 20 (*LSH-DBSCAN-K20-L1* and *LSH-DBSCAN-K20-L5*), the total number of unclustered tweets is quite high for each threshold. Actually, it makes sense since when K is 20, we have more buckets and the candidates for the similar tweets are exact same or nearly the

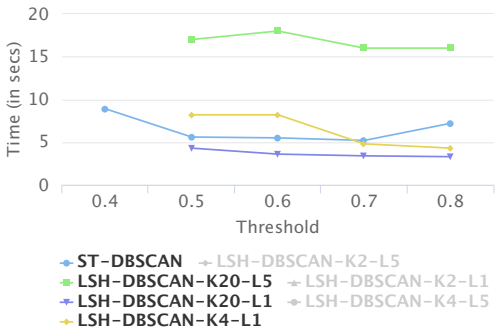


Figure 5.16: Zoom in version of Figure 5.15

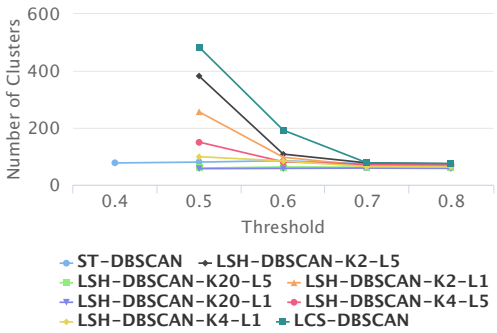


Figure 5.17: Number of clusters for 10K tweets with different thresholds

same tweets. That's why these methods tend to cluster less number of tweets. It shows that depending on our purpose, if we want to capture very similar (exact or nearly the same) tweets, then it is better to choose *LSH-DBSCAN-K20-L1* or *LSH-DBSCAN-K20-L5*. We can also prefer to choose *ST-DBSCAN* with a high threshold for the same purpose and their timing performances do not have big differences as shown in Figure 5.15 and Figure 5.16.

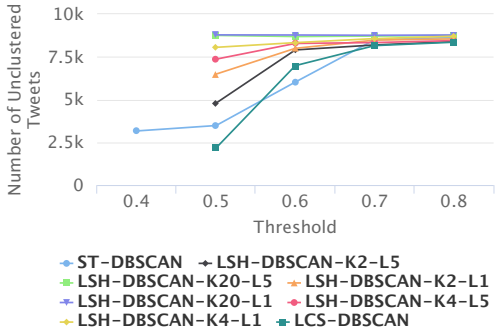


Figure 5.18: Number of unclustered tweets for 10K tweets with different thresholds

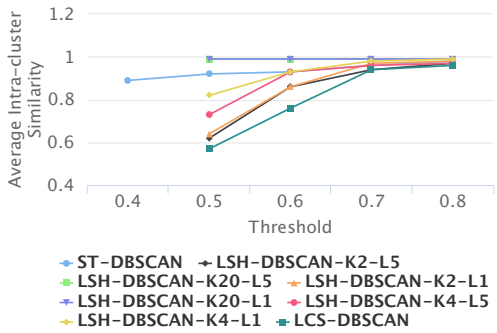


Figure 5.19: Average intra-cluster similarity for 10K tweets with different thresholds

When we look at the average intra-cluster similarity results in Figure 5.19, again *LSH-DBSCAN-K20-L1* and *LSH-DBSCAN-K20-L5* have the highest similarity values. However, *ST-DBSCAN* also shines here, because although it clusters more tweets than both *LSH-DBSCAN-K20-L1* and *LSH-DBSCAN-K20-L5*, the average intra-cluster similarity

for *ST-DBSCAN* is still quite high. The remaining methods showed almost the same pattern for this metric.

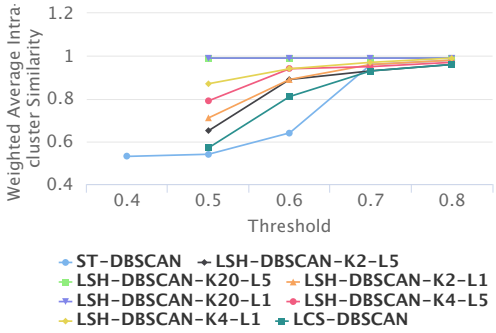


Figure 5.20: Weighted average intra-cluster similarity for 10K tweets with different thresholds

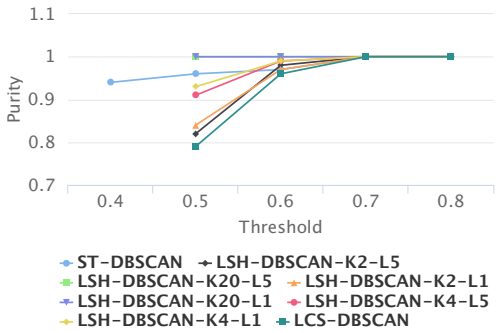


Figure 5.21: Purity for 10K tweets with different thresholds

We have an interesting result in Figure 5.20 which is for weighted average intra-cluster similarity. *ST-DBSCAN* in the only method that shows a different pattern with its average intra-cluster similarity in Figure 5.19. For smaller thresholds, weighted average intra-cluster similarity is much lower than the average intra-cluster similarity for *ST-DBSCAN*. It shows that bigger clusters in *ST-DBSCAN* have much lower intra-cluster similarities than the smaller clusters have. Purity results in Figure 5.21 have almost same output with average intra-cluster similarity results in Figure 5.19.

In Figure 5.22, Figure 5.23, Figure 5.24, Figure 5.25, we give precision, recall and F-score results for *LCS-DBSCAN*, *ST-DBSCAN* and *LSH-DBSCAN* methods with different thresholds. We selected *LSH-DBSCAN-K20-L1* to represent *LSH-DBSCAN* due to its good results in the previous examples. Again we assume that there are 4 big clusters (referring to #jesuisCharlie, #christmas, #nba and #trump) and each of the relatively smaller clusters produced by these methods belongs to one of the 4 big clusters depending on the most frequent real class label.

Results show that *ST-DBSCAN* with high thresholds and *LSH-DBSCAN* produce clusters with the same or better quality than *LCS-DBSCAN* produce. However, their time performances are quite different, and it proves that it is worth to choose *LSH-DBSCAN* or

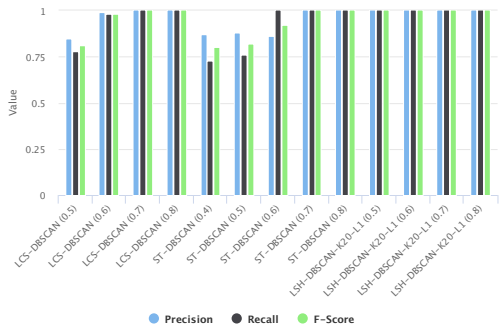


Figure 5.22: Precision, Recall and F-Score results for "#charlie" cluster

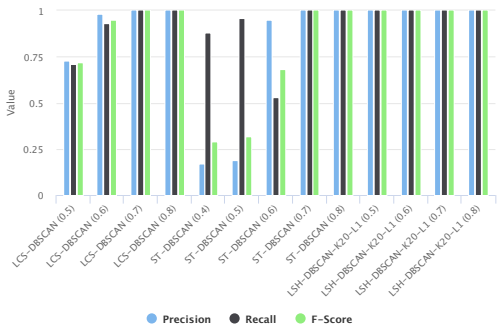


Figure 5.23: Precision, Recall and F-Score results for "#christmas" cluster

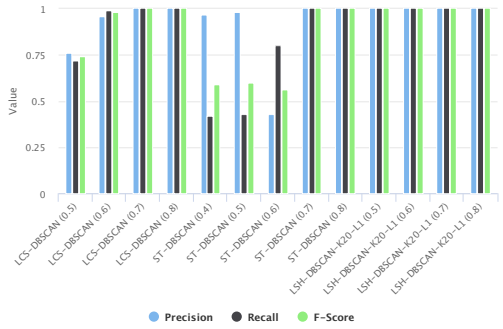


Figure 5.24: Precision, Recall and F-Score results for "#nba" cluster

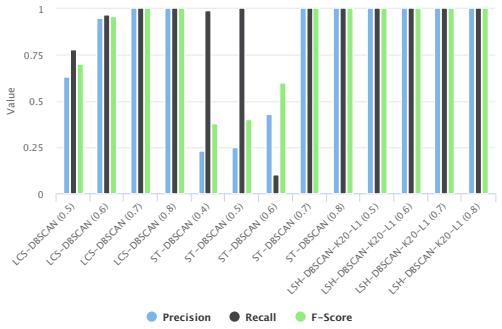


Figure 5.25: Precision, Recall and F-Score results for "#trump" cluster

ST-DBSCAN to make a lexical clustering on tweets.

Remember that we set $minPts$ to 10 in this subsection and conducted all of the experiments with different ε (threshold) values for density based methods. We now conducted some other experiments in order to see how the results change when $minPts$ is 5, 10, 50 and 100. We selected *LSH-DBSCAN-K20-L1* and *ST-DBSCAN*; and set their thresholds to 0.5 and 0.4 respectively.

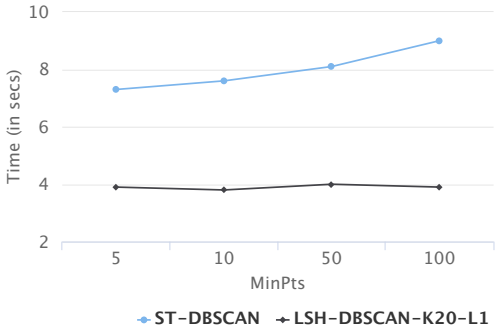


Figure 5.26: Time performance with different $minPts$ values

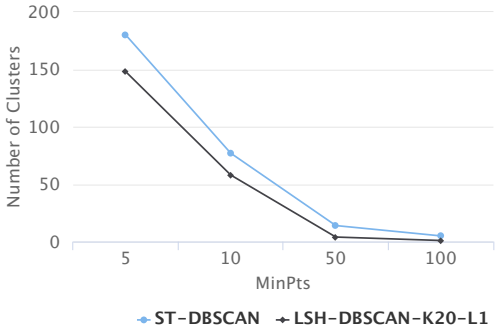


Figure 5.27: Number of clusters with different $minPts$ values

Figure 5.26 shows that the time performance increases only a little bit when $minPts$ increases for *ST-DBSCAN*; but it is almost constant for *LSH-DBSCAN-K20-L1*. However, the number of clusters in Figure 5.27 decreases significantly as the $minPts$ increases. There are only 5 and 1 cluster(s) for *ST-DBSCAN* and *LSH-DBSCAN-K20-L1* respectively when $minPts$ is 100. Actually it is a quite expected result, since it is harder to compose a cluster in DBSCAN with higher $minPts$ values.

When we look at Figure 5.28, we see that the number of unclustered tweets increases for both methods as $minPts$ increases. It is also expected with less clusters in higher $minPts$ values. However, *LSH-DBSCAN-K20-L1* clusters less tweets than *ST-DBSCAN* does, because *LSH-DBSCAN-K20-L1* only retrieves exact the same or nearly the same tweets as the nearest neighbours which was already mentioned before. This fact also inclines the result in Figure 5.29 and Figure 5.30 where the average and weighted average intra-cluster similarities are almost 1.0 for *LSH-DBSCAN-K20-L1*.

Consequently, we observed that the number of clusters is more reasonable when

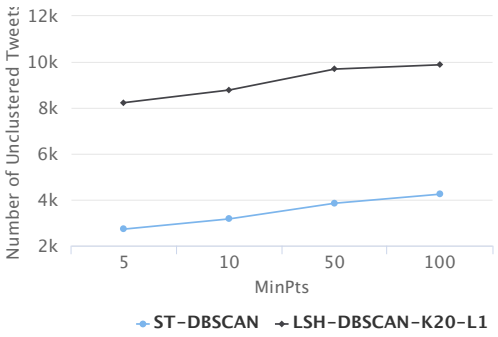


Figure 5.28: Number of unclustered tweets with different $minPts$ values

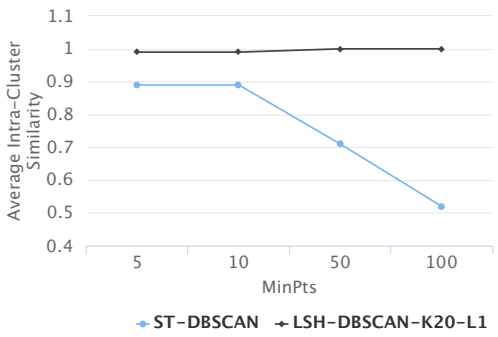


Figure 5.29: Average intra-cluster similarity with different $minPts$ values

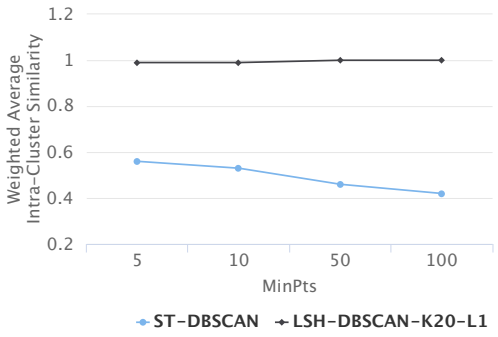


Figure 5.30: Weighted average intra-cluster similarity with different $minPts$ values

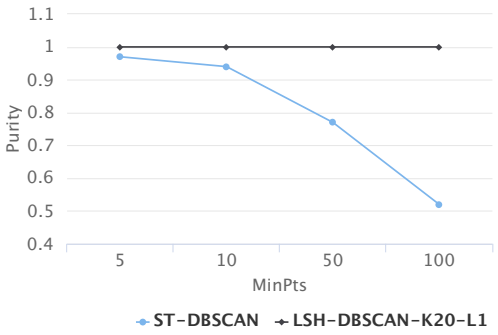


Figure 5.31: Purity with different $minPts$ values

$minPts$ is 5 and 10. The average intra-cluster similarity is quite good for these values as well. As Schubert et al. [73] stated that it may improve the cluster results to increase $minPts$ for the datasets that are very large or have many duplicates. Therefore, we selected $minPts$ as 10 among the candidates 5 and 10.

5.2.3 Comparison of *LSH-DBSCAN*, *ST-DBSCAN*, *ST-TWEC*

In this part, we will compare *LSH-DBSCAN*, *ST-DBSCAN* and *ST-TWEC* methods in terms of the time performance. We observe that these three methods performed quite good in terms of the time performance and cluster qualities in the previous subsections. Now, we will compare them with the bigger dataset used in subsection 5.2.1 in terms of time performance. Again we selected *LSH-DBSCAN-K20-L1* to represent *LSH-DBSCAN* since its good timing results. We conducted our experiments with the same machine used in subsection 5.2.1 (a system with 32 processor, model name *Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90GHz*, and 128 GB RAM). Results regarding these experiments are given in Figure 5.32. Although *LSH-DBSCAN-K20-L1* performed slightly better than *ST-DBSCAN* with 10K dataset in terms of the time performance in subsection 5.2.2, as the data size grows *ST-DBSCAN* starts to perform better. And we see that there is not a big difference between *ST-DBSCAN* and *ST-TWEC* in Figure 5.32.

It is also worth to note memory consumption of each method. Remember that suffix tree *ST-TWEC* consumed 475 MB in subsection 5.2.1. We noted in subsection 5.1.5 that we will have lots of queries in *ST-DBCAN* to get nearest neighbours (i.e to find similar tweets), we decided to keep ids of the tweets in each level to reach them very fast. So that, suffix tree in *ST-DBCAN* consumes much more memory which is 6060 MB. On the other hand, *LSH-DBSCAN-K20-L1* consumes 290 MB for the hash table in the algorithm. Note that there is only 1 hash table in *LSH-DBSCAN-K20-L1*. If we increase the value of L, then the memory consumption will increase too.

Lastly, we want to state than these three methods are scalable. We have already shown that how *ST-TWEC* performs with different datasets which contain from 100K to 1M tweets. We also show and compare these methods in the same range as shown in

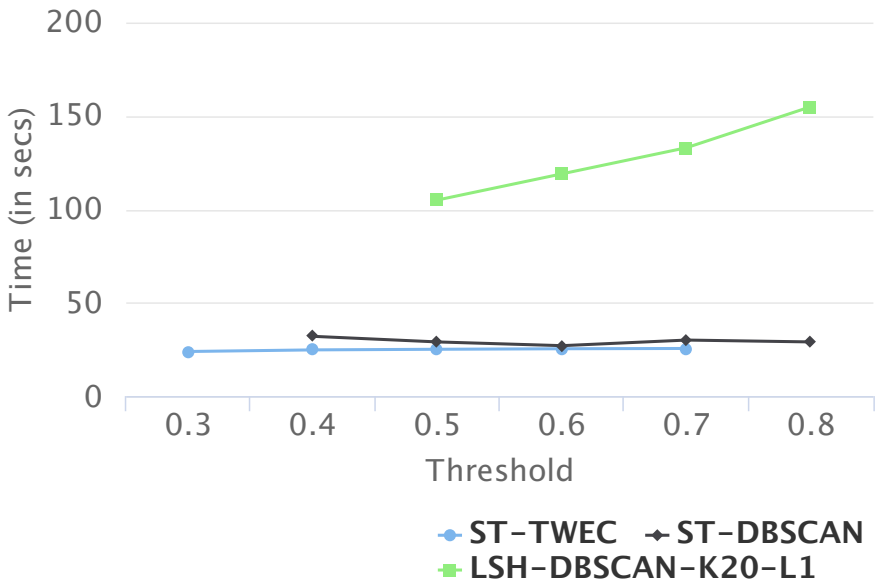


Figure 5.32: Compare *LSH-DBSCAN-K20-L1*, *ST-DBSCAN* and *ST-TWEC* with 60K dataset in terms of the time performance

Figure 5.33. In this experiment, the thresholds were selected as 0.4 for *ST-TWEC* and *ST-DBSCAn*; and selected as 0.5 for *LSH-DBSCAN-K20-L1*.

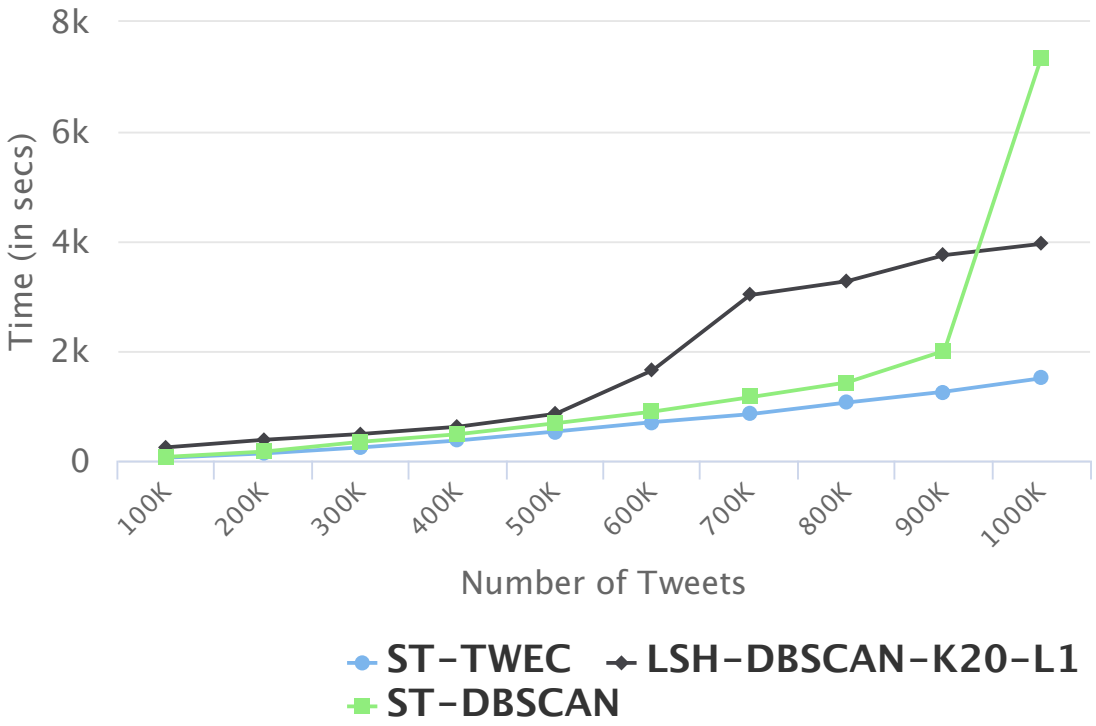


Figure 5.33: Compare *LSH-DBSCAN-K20-L1*, *ST-DBSCAN* and *ST-TWEC* with different data sizes

Chapter 6

Conclusion and Future Work

Twitter is one of the most popular platforms where users share their opinions, reactions and feelings towards the events and daily life. For that reason, it is very important to measure the impact of the tweets and topics in Twitter for researchers and policy makers to understand the trends and the reactions of the society in a better way. In this work, we focused on the ways to measure the impact of tweets or topics. We firstly emphasized the role of retweets to make the information transmitted in all over the Twitter. Therefore, in the first part of the thesis, we focused on predicting whether a tweet will take high number of retweets as it had been also studied by some other works in the literature. However, our work contains deeper analysis. We grouped tweets in different classes depending on the number of retweets they received. We have end up with seven features of tweets, namely *numberOfFollowers*, *containsHighRelatedTerms*, *containsLowRelatedTerms*, *containsPoliticWords*, *containsLink*, *numberOfCapitalLetters*, and *containsHashtag* to create a good learning model and we obtained satisfying results in terms of the accuracy. In the second part of the chapter 4, we more concentrated on extracting the content based features of tweets by using Convolutional Neural Networks.

In the second part of the thesis, we introduced the concept of *hidden retweets*. We observed that people tend to re-post tweets by adding some extra comments to the beginning or to the end of these tweets or they also tend to re-post the same tweets without retweeting them; and these are not part of the original retweets. It doesn't matter whether these

tweets are supportive or against the original tweet. Our focus is that users talk about this tweet/topic to increase the awareness in some way (positive or negative). So that hidden retweets are quite important for measuring the real impact of the tweet.

In this work, capturing hidden retweets is defined as a document clustering problem, since we try to group similar tweets whose similarity value is above a threshold in the same cluster. Although there are lots of works on document clustering methods in the literature, clustering tweets needs some kind of adaptive solutions due to Twitter's characteristic features. As it is mentioned in the thesis, tweets are short documents (at most 140 characters), users intentionally or unintentionally use the misspelled words or abbreviations; therefore standard text clustering methods do not work well for short and informal textual data generated in Twitter. In the first part of the chapter 5, we presented a suffix tree based tweet clustering algorithm, *ST-TWEC*, which is able to efficiently cluster the tweets in large scales. In order to prove its quality, we compared *ST-TWEC* with a benchmark algorithm (*LCS-Lex*) which is a lexical tweet clustering method based on Longest Common Subsequence (LCS) similarity metric. In fact, LCS is a good similarity metric for comparing tweets, however, it has a high time complexity. Our experiments revealed that *ST-TWEC* is capable of constructing high quality clusters as *LCS-Lex* constructs in terms of avgISim, wAvgISim, Purity, Precision, Recall and F-Score evaluation metrics. We also showed that while constructing same quality clusters, *ST-TWEC* dramatically outperforms *LCS-Lex* in terms of the time performance. This outcome enables us to cluster tweets in a more efficient way and to work in scales of million tweets which is the case in real life. Apart from that, we also showed that *ST-TWEC* runs more efficiently than traditional document clustering methods (k-means document clustering) in terms of the time performance and cluster qualities. In the second part of the chapter 5, we were more focused on density based clustering approaches. First, we have developed a naive implementation of DBSCAN (*LCS-DBSCAN*) which uses LCS to find similar tweets. Then, we integrated DBSCAN with generalized suffix tree data structure (*ST-DBSCAN*) and locality sensitive hashing method (*LSH-DBSCAN*) as the second and third method respectively to improve the time performance of retrieving the nearest neighbour tweets. Especially,

LSH-DBSCAN has two different parameter which are the length of the hash code and the number of the hash tables. We compared these methods with different parameters, and showed how *ST-DBSCAN* and *LSH-DBSCAN* performs effectively to cluster similar tweets. In the last part, since *ST-TWEC*, *ST-DBSCAN* and *LSH-DBSCAN* outperformed the naive implementations of their competitors, we also made a comparison among them with different data sizes.

As feature work, "Wide and Deep Neural Networks" [11] might be performed to predict whether a tweet will get high number of retweets. Note that the biggest limitation for the suffix tree based methods is the memory requirement of the constructed generalized suffix tree. As the tweet size grows up, the memory size consumed by the generalized suffix tree also increases. In order to process a higher number of tweets, we plan to extend these methods with batch clustering and store a portion of the generated suffix tree in the secondary storage as future work.

Last but not least, remember that we mentioned about five different reasons why people retweet in chapter 1. We observe that these reasons are directly related to the number of hidden retweets. For example, if users want to promote some specific people or an account, they tend to retweet the original tweets of these people/accounts. Similarly, if users want to increase the awareness on a specific event/tweet then they tend to retweet the original tweet since the high number of retweets, that is explicitly seen by everyone, might affect other people too. These two reasons are the factors that decrease the number of hidden retweets. On the other hand, users also share jokes and humorous contents. In this case, they might copy/paste and send this tweet as their own post to seem more funny and increase their popularity in the social network. Or when they want to criticize, protest or insult an event/opinion, they copy the original tweet and add extra comment that express their idea. These reason are the factors that increase number of hidden retweets. As we said that, we believe the number of hidden retweets is directly related to these concepts, however we did not make any experiment to prove this idea which might be done as future work.

Appendices

.1 Additional Figures

Tweet	# of RT
rt @omgadamsaleh: we got kicked out of a @delta airplane because i spoke arabic to my mom on the phone and with my friend slim... wtf...	14715
rt @omgadamsaleh: we got kicked out of a @delta airplane because i spoke arabic to my mom on the (cont) https://t.co/ttoakdczsc	1
damn rt @omgadamsaleh: we got kicked out of a @delta airplane because i spoke arabic to my mom on the ... https://t.co/b1mnkidebf	1
rt @eric_of_1691: we got kicked out of a @delta airplane because i spoke arabic to my mom on the phone and... by #gasuza_tiziano	1
rt @ftoon_noor: @i_9mile we got kicked out of a @delta airplane because i spoke arabic to my mom on the phone and with my wtf https://t.c...	1
rt @20x99: we got kicked out of a @delta airplane because i spoke arabic to my mom on the phone and with my friend slim... wtff https://t.c...	4
woooow rt @omgadamsaleh: we got kicked out of a @delta airplane because i spoke arabic to my mom on (cont) https://t.co/bqvoslehlc	1
smh rt @omgadamsaleh: we got kicked out of a @delta airplane because i spoke arabic to my mom on (cont) https://t.co/7i3owr8c4j	1

Figure 1: Label: we got kicked out of a airplane because i spoke arabic to my mom on the .

Tweet	# of RT
rt @therealtymula: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/flp2jswz11	1248
rt @blackpplvines: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/tgmzmirdmf	289
rt @niggativities: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/oa4yvcfvo5	38
rt @worldstarvlne: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/o7s4jgdpda	121
rt @wshhfans: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/2n3hjwyed6	399
rt @thastonernation: snoop dogg and wiz had a smoke off in the middle of a concert 😤 https://t.co/hh0yv892hu	185
rt @freestyleraps: snoop & wiz had a smoke off in the middle of a concert lmao https://t.co/higmmzomks	82
rt @cloudn9nesyrup: snoop dogg and wiz had a smoke off in the middle of a concert 😤 https://t.co/slrngd25rb	356
rt @worldstarcomedy: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/spdagm8xee	183
rt @hotfreestyle: snoop and wiz had a smoke off in the middle of a concert https://t.co/jdejhfnqqg	133
rt @weloverobdyrdek: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/zt7bj220ft	156
rt @therealtymula: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠 https://t.co/flp2jswz11	106
rt @hippy: snoop dogg and wiz had a smoke off in the middle of a concert 😤 https://t.co/tsgjcifrgt	64
rt @guycodes: snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/3uxf9bx6gp	50
rt @iibreaknecks: uncle snoop and wiz had a smoke off in the middle of a concert 😠 https://t.co/nvc5k1evgn	212
rt @thehoodvines: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/mjsqkd04f6	61
rt @thehighfessions: snoop & wiz had a smoke off in the middle of a concert lmao https://t.co/wbojsbhwzm	51
rt @worldstarlaugh: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/4ggc1t9gca	30
rt @worldstarnow: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/njogvebent	8
rt @litfreestyles: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/01kl7cqjok	258
rt @populus678: snoop and wiz really had a smoke off in the middle of a concert. 😠 https://t.co/ql6hjcnadh	87
rt @comfade: uncle snoop and wiz had a smoke off in the middle of a concert 💀😠 https://t.co/p4zthgazdo	36
rt @brologics: snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/gxg7x9qxqa	4
rt @somexlcan: snoop and wiz had a smoke off in the middle of a concert https://t.co/aljsuciuwi	14
rt @tim_nordahl: uncle snoop and wiz had a smoke off in the middle of a concert 😠 https://t.co/xyswcay2vr	21
rt @themeninist: snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/9bcfpatjbe	11
rt @niggacommentary: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/okqsuncga8	133
rt @ratchet: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/igweaabeu9	11
rt @trrvisxx: uncle snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/xrgo7rmz95	6
rt @causewereguys: snoop and wiz had a smoke off in the middle of a concert 😠😠😠 https://t.co/ugw9bgjbez	25

Figure 2: Label: had a smoke off in the middle of a concert

.

Tweet	# of RT
rt @mowgli6ix: how i sleep at night knowing i'm a disappointment to my family https://t.co/fh9zjfauev	1056
rt @girlposts: how i sleep at night knowing i'm a disappointment to my family https://t.co/wqiz5sgmcm	586
rt @wshhfans: how i sleep at night knowing i'm a disappointment to my family https://t.co/hupjpxtlw1	136
rt @w0rldstarhlphop: how i sleep at night knowing i'm a disappointment to my family https://t.co/rjpp7123i9	18
rt @mariopalush: how i sleep at night knowing i'm a disappointment to my family https://t.co/3yovcpoybz	27
rt @chanelpuke: how i sleep at night knowing i'm a disappointment to my family https://t.co/7mmxgdhbii	119
rt @mlkeekhourry: how i sleep at night knowing i'm a disappointment to my family https://t.co/ekg5u0nbou	69
rt @powerful: how i sleep at night knowing i'm a disappointment to my family https://t.co/gtautpjdcx	49
rt @freddyamazin: how i sleep at night knowing i'm a disappointment to my family https://t.co/xqcgmy9hi8	175
rt @comedyortruth: how i sleep at night knowing i'm a disappointment to my family https://t.co/lhbm7pm4j1	60
rt @worldshiphop: how i sleep at night knowing i'm a disappointment to my family https://t.co/i56ycwnimf	6
rt @fiirtationship: how i sleep at night knowing i'm a disappointment to my family https://t.co/jh18r3idzm	59
rt @femalebook: how i sleep at night knowing i'm a disappointment to my family https://t.co/cjufvbo9rw	14
rt @femaletexts: how i sleep at night knowing i'm a disappointment to my family https://t.co/9vagidokvo	37
rt @girlhoodposts: how i sleep at night knowing i'm a disappointment to my family https://t.co/g1nd4rudok	9
rt @woridstarlaugh: how i sleep at night knowing i'm a disappointment to my family https://t.co/isalp6xcku	14
rt @bubblestbh: how i sleep at night knowing i'm a disappointment to my family https://t.co/3mbckkztv0	14
how i sleep at night knowing i'm a disappointment to my family https://t.co/hpx2sofljc	1
rt @girivibe: how i sleep at night knowing i'm a disappointment to my family https://t.co/zsueknztwf	3
rt @aianhangover: how i sleep at night knowing i'm a disappointment to my family https://t.co/vgurbuwlse	3
how i sleep at night knowing i'm a disappointment to my family https://t.co/isalp6xcku	1
rt @thedurttyboyz: how i sleep at night knowing i'm a disappointment to my family https://t.co/apaewzfocv #durttyboyz	1
rt @comedyposts: how i sleep at night knowing i'm a disappointment to my family https://t.co/qqxvgkzcul	9
rt @teengirllyf: how i sleep at night knowing i'm a disappointment to my family https://t.co/hpx2sofljc	1
rt @relatibie: how i sleep at night knowing i'm a disappointment to my family https://t.co/amdkyrhf2g	1
how i sleep at night knowing i'm a disappointment to my family https://t.co/wqiz5sgmcm	1
how i sleep at night knowing i'm a disappointment to my family https://t.co/i56ycwnimf	1
how i sleep at night knowing i'm a disappointment to my family https://t.co/i2ns3z5obs	1
rt @prymetymej3: rt @mowgli6ix: how i sleep at night knowing i'm a disappointment to my family https://t.co/yanizqzvwq	1
how i sleep at night knowing i'm a disappointment to my family https://t.co/xxcissuta7	1
rt @gossipgriii: how i sleep at night knowing i'm a disappointment to my parents https://t.co/whefxj931h	1
rt @mowgli6ix: how i sleep at night knowing i'm a disappointment to my family https://t.co/guhdlsqx0m	1

Figure 3: Label: how i sleep at night knowing i m a disappointment to my

.

Bibliography

[1] Alexa. Top sites in turkey. http://www.alexa.com/topsites/countries/TR, 2016. Accessed 2016.08.02.

[2] N. Ela Gökalp Aras and Zeynep Şahin Mencütek. The international migration and foreign policy nexus: the case of syrian refugee crisis and turkey. *Migration Letters*, 12(3), 2015. ISSN 1741-8992. URL http://www.tplondon.com/journal/index.php/ml/article/view/502.

[3] Farzindar Atefeh and Wael Khreich. A survey of techniques for event detection in twitter. *Comput. Intell.*, 31(1):132–164, February 2015. ISSN 0824-7935. doi: 10.1111/coin.12017. URL http://dx.doi.org/10.1111/coin.12017.

[4] Zeynep Aycan. Human resource management in Turkey - Current issues and future challenges. *International Journal of Manpower*, 22(3):252–260, 2001. doi: 10.1108/01437720110398347. URL https://doi.org/10.1108/01437720110398347.

[5] H. Becker, M. Naaman, and L. Gravano. Beyond trending topics: Real-world event identification on twitter. In *Fifth International AAAI Conference on Weblogs and Social Media*, 2011.

[6] beevolve. An exhaustive study of twitter users across the world, 2012. URL http://www.beevolve.com/twitter-statistics/.

[7] Tuba Bircan and Ulaş Sunata. Educational assessment of syrian refugees in

turkey. *Migration Letters*, 12(3), 2015. ISSN 1741-8992. URL http://www.tplondon.com/journal/index.php/ml/article/view/509.

[8] Johan Bollen, Huina Mao, and Xiao-Jun Zeng. Twitter mood predicts the stock market. *CoRR*, abs/1010.3003, 2010. URL http://arxiv.org/abs/1010.3003.

[9] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[10] Dawn Chatty. The syrian humanitarian disaster: Disparities in perceptions, aspirations, and behaviour in jordan, lebanon and turkey. *IDS Bulletin*, 47(3), 2016. ISSN 1759-5436. doi: 10.19088/1968-2016.142. URL http://bulletin.ids.ac.uk/idsbo/article/view/2728.

[11] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. *CoRR*, abs/1606.07792, 2016. URL http://arxiv.org/abs/1606.07792.

[12] M. Cheong and V. Lee. A study on detecting patterns in twitter intra-topic user and message clustering. In *2010 20th International Conference on Pattern Recognition*, pages 3125–3128, Aug 2010. doi: 10.1109/ICPR.2010.765.

[13] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7(Mar):551–585, 2006.

[14] Rianne Dekker and Godfried Engbersen. How social media transform migrant networks and facilitate migration. *Global Networks*, 14(4):401–418, 2014. ISSN 1471-0374. doi: 10.1111/glob.12040. URL http://dx.doi.org/10.1111/glob.12040.

[15] Daniel Rodriguez Dominguez, Rebeca P. Diaz Redondo, Ana Fernandez Vilas, and Mohamed Ben Khalifa. Sensing the city with instagram: Clustering geolocated data for outlier detection. *Expert Systems with Applications*, 78:319 – 333, 2017. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2017.02.018. URL http://www.sciencedirect.com/science/article/pii/S0957417417300994.

[16] Elastic. The heart of the elastic stack, 2016. URL https://www.elastic.co/products/elasticsearch.

[17] M. K. Erpam. Tweets on a tree: Index-based clustering of tweets. Technical report, Sabanci University, Istanbul, Turkey, April 2017. URL http://research.sabanciuniv.edu/31274/.

[18] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[19] Yixiang Fang, Haijun Zhang, Yunming Ye, and Xutao Li. Detecting hot topics from twitter: A multiview approach. *Journal of Information Science*, 40(5):578–593, 2014.

[20] G Fowler, P Vo, and LC Noll. Fvn hash, 1991.

[21] Nell Gabiam. Humanitarianism, development, and security in the 21st century: Lessons from the syrian refugee crisis. *International Journal of Middle East Studies*, 48(2):382–386, 2016. doi: 10.1017/S0020743816000131.

[22] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[23] GLOBE. Global leadership & organizational behavior effectiveness. http://globe.bus.sfu.ca/results/countries/TUR?menu=list, 2016. Accessed 2016.09.30.

[24] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press, New York, NY, USA, 1997. ISBN 0-521-58519-8.

[25] Donald O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, New York, June 1949. ISBN 0-8058-4300-0.

[26] François Heisbourg. The Strategic Implications of the Syrian Refugee Crisis. *Survival*, 57(6):7–20, 2015. doi: 10.1080/00396338.2015.1116144. URL http://dx.doi.org/10.1080/00396338.2015.1116144.

[27] Geert H. Hofstede. *Culture's consequences: International differences in work-related values*. Sage Publications, Beverly Hills, CA, 1980.

[28] Robert J House, Global Leadership Program., and Organizational Behavior Effectiveness Research. *Culture, leadership, and organizations : the GLOBE study of 62 societies*. Sage Publications, Thousand Oaks, Calif., 2004. ISBN 0761924019 9780761924012.

[29] Bill Howard. Analyzing online social networks. *Commun. ACM*, 51(11):14–16, November 2008. ISSN 0001-0782. doi: 10.1145/1400214.1400220. URL http://doi.acm.org/10.1145/1400214.1400220.

[30] Philip Issa. Syrian civil war: Five ways the conflict has changed the world. http://www.independent.co.uk/news/world/middle-east/syrian-civil-war-isis-how-it-changed-the-world-refugee-crisis-a6html, 2016. Accessed 2016.09.30.

[31] Adam Jacobs. The pathologies of big data. *Commun. ACM*, 52(8):36–44, August 2009. ISSN 0001-0782. doi: 10.1145/1536616.1536632. URL http://doi.acm.org/10.1145/1536616.1536632.

[32] Fieke Jansen. Digital activism in the Middle East: mapping issue networks in Egypt, Iran, Syria and Tunisia. *Knowledge Management for Development Journal*,

6(1):37–52, 2010. doi: 10.1080/19474199.2010.493854. URL http://dx.doi.org/10.1080/19474199.2010.493854.

[33] Jonah Berger and Katherine L Milkman. What Makes Online Content Viral? *Journal of Marketing Research*, 49(2):192–205, 2012. doi: 10.1509/jmr.10.0353. URL https://doi.org/10.1509/jmr.10.0353.

[34] Sunghae Jun, Sang-Sung Park, and Dong-Sik Jang. Document clustering method using dimension reduction and support vector clustering to overcome sparseness. *Expert Systems with Applications*, 41(7):3204 – 3212, 2014. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2013.11.018. URL http://www.sciencedirect.com/science/article/pii/S0957417413009305.

[35] Hayat Kabasakal and Muzaffer Bodur. Leadership, values and institutions: The case of turkey, 1998. Unpublished Manuscript. Istanbul, Turkey.

[36] Omer M. Karasapan. The impact of syrian businesses in turkey. https://www.brookings.edu/blog/future-development/2016/03/16/the-impact-of-syrian-businesses-in-turkey/, 2016. Accessed 2016.08.02.

[37] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[38] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[40] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th International Con-*

ference on World Wide Web, WWW '10, pages 591–600, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772751. URL http://doi.acm.org/10.1145/1772690.1772751.

[41] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

[42] Sarah Van Leuven, Ansgard Heinrich, and Annelore Deprez. Foreign reporting and sourcing practices in the network sphere: A quantitative content analysis of the Arab Spring in Belgian news media. *New Media & Society*, 17(4):573–591, 2015. doi: 10.1177/1461444813506973. URL http://dx.doi.org/10.1177/1461444813506973.

[43] Chenliang Li, Aixin Sun, and Anwitaman Datta. Twevent: Segment-based event detection from tweets. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 155–164, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1156-4. doi: 10.1145/2396761.2396785. URL http://doi.acm.org/10.1145/2396761.2396785.

[44] Yinglong Ma, Yao Wang, and Beihong Jin. A three-phase approach to document clustering based on topic significance degree. *Expert Systems with Applications*, 41(18):8203 – 8210, 2014. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2014.07.014. URL http://www.sciencedirect.com/science/article/pii/S0957417414004126.

[45] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[46] Oded Maimon and Lior Rokach. *Data Mining and Knowledge Discovery Handbook*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 0387244352, 9780387244358.

[47] Juan Martinez-Romo and Lourdes Araujo. Detecting malicious tweets in trending topics using a statistical analysis of language. *Expert Systems with Applications*, 40(8):2992 – 3000, 2013. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2012.12.015. URL http://www.sciencedirect.com/science/article/pii/S0957417412012560.

[48] MEE. Economic effect of syrian war at $35bn: World bank. http://www.middleeasteye.net/news/economic-effect-syrian-war-35bn-world-bank-1953497162, 2016. Accessed 2016.08.02.

[49] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. URL http://arxiv.org/abs/1301.3781.

[50] Zachary Miller, Brian Dickinson, William Deitrick, Wei Hu, and Alex Hai Wang. Twitter spammer detection using data stream clustering. *Inf. Sci.*, 260:64–73, March 2014. ISSN 0020-0255. doi: 10.1016/j.ins.2013.11.016. URL http://dx.doi.org/10.1016/j.ins.2013.11.016.

[51] Hamideh Molaei. Discursive opportunity structure and the contribution of social media to the success of social movements in Indonesia. *Information, Communication & Society*, 18(1):94–108, 2015. doi: 10.1080/1369118X.2014.934388. URL http://dx.doi.org/10.1080/1369118X.2014.934388.

[52] Kyosuke Nishida, Ryohei Banno, Ko Fujimura, and Takahide Hoshide. Tweet classification by data compression. In *Proceedings of the 2011 international workshop on DETecting and Exploiting Cultural diversiTy on the social web*, DETECT '11, pages 29–34, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0962-2. doi: http://doi.acm.org/10.1145/2064448.2064473. URL http://doi.acm.org/10.1145/2064448.2064473.

[53] MARGARITA NORIEGA. Why we retweet, 2014. URL http://www.dailydot.com/debug/why-we-retweet/.

[54] Oytun Orhan. Effects of the syrian refugees on turkey, 2015. TESEV.

[55] Oswego. The 68-95-99.7 rule for normal distributions. http://www.oswego.edu/~srp/stats/6895997.htm, 2016. Accessed 2016.05.15.

[56] Humeyra Pamuk. Turkish mine disaster town under lockdown as death toll rises to 301, 2014. URL http://www.reuters.com/article/us-turkey-mine-idUSBREA4C0KO20140518.

[57] Joel Penney and Caroline Dadas. (Re)Tweeting in the service of protest: Digital composition and circulation in the Occupy Wall Street movement. *New Media & Society*, 16(1):74–90, 2014. doi: 10.1177/1461444813479593. URL http://dx.doi.org/10.1177/1461444813479593.

[58] Kay Peters, Yubo Chen, Andreas M Kaplan, Björn Ognibeni, and Koen Pauwels. Social Media Metrics — A Framework and Guidelines for Managing Social Media. *Journal of Interactive Marketing*, 27(4):281–298, 2013. ISSN 1094-9968. doi: http://dx.doi.org/10.1016/j.intmar.2013.09.007. URL http://www.sciencedirect.com/science/article/pii/S109499681300042X.

[59] Sasa Petrovic, Miles Osborne, and Victor Lavrenko. Rt to win! predicting message propagation in twitter. *ICWSM*, 11:586–589, 2011.

[60] C. Phillips. The impact of syrian refugees on turkey and jordan, 2012. The World Today.

[61] Shanmugam Poomagal, Palanisamy Visalakshi, and Thiagarajan Hamsapriya. A novel method for clustering tweets in twitter. *International Journal of Web Based Communities*, 11(2):170–187, 2015.

[62] Donatella Della Porta. Comment on organizing in the crowd. *Information, Communication & Society*, 17(2):269–271, 2014. doi: 10.1080/1369118X.2013.868503. URL http://dx.doi.org/10.1080/1369118X.2013.868503.

[63] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011. ISBN 1107015359, 9781107015357.

[64] Aniket Rangrej, Sayali Kulkarni, and Ashish V. Tendulkar. Comparative study of clustering techniques for short text documents. In *Proceedings of the 20th International Conference Companion on World Wide Web*, WWW '11, pages 111–112, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0637-9. doi: 10.1145/1963192.1963249. URL http://doi.acm.org/10.1145/1963192.1963249.

[65] Felix Richter. Twitter's top 5 markets account for 50% of active users. https://www.statista.com/chart/1642/regional-breakdown-of-twitter-users/, 2013. Accessed 2016.08.02.

[66] Huaxia Rui, Yizao Liu, and Andrew Whinston. Whose and what chatter matters? the effect of tweets on movie sales. *Decis. Support Syst.*, 55(4):863–870, November 2013. ISSN 0167-9236. doi: 10.1016/j.dss.2012.12.022. URL http://dx.doi.org/10.1016/j.dss.2012.12.022.

[67] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: Real-time event detection by social sensors. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 851–860, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8. doi: 10.1145/1772690.1772777. URL http://doi.acm.org/10.1145/1772690.1772777.

[68] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Tweet Analysis for Real-Time Event Detection and Earthquake Reporting System Development. *IEEE*

Transactions on Knowledge and Data Engineering, 25(4):919–931, apr 2013. ISSN 1041-4347. doi: 10.1109/TKDE.2012.29. URL http://ieeexplore.ieee.org/document/6152108/.

[69] Rodrigo Sandoval-Almazan and J Ramon Gil-Garcia. Towards cyberactivism 2.0? Understanding the use of social media and other information technologies for political activism and social movements. *Government Information Quarterly*, 31 (3):365–378, 2014. ISSN 0740-624X. doi: http://dx.doi.org/10.1016/j.giq.2013.10.016. URL http://www.sciencedirect.com/science/article/pii/S0740624X14000902.

[70] Rıdvan Saraçoğlu, Kemal Tutuncu, and Novruz Allahverdi. A new approach on search for similar documents with multiple categories using fuzzy clustering. *Expert Systems with Applications*, 34(4):2545 – 2554, 2008. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2007.04.003. URL http://www.sciencedirect.com/science/article/pii/S0957417407001467.

[71] Neil Savage. Twitter as medium and message. *Commun. ACM*, 54(3):18–20, March 2011. ISSN 0001-0782. doi: 10.1145/1897852.1897860. URL http://doi.acm.org/10.1145/1897852.1897860.

[72] David Sayce. Number of tweets per day?, 2016. URL http://www.dsayce.com/social-media/tweets-day/.

[73] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Trans. Database Syst.*, 42(3):19:1–19:21, July 2017. ISSN 0362-5915. doi: 10.1145/3068335. URL http://doi.acm.org/10.1145/3068335.

[74] Zhan Shi, Huaxia Rui, and Andrew B. Whinston. Content sharing in a social broadcasting environment: Evidence from twitter. *MIS Q.*, 38(1):123–142, March

2014. ISSN 0276-7783. URL http://dl.acm.org/citation.cfm?id=2600518.2600525.

[75] Amit Singhal. Modern information retrieval: A brief overview. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 24(4)::35–43, 2001.

[76] Web Smith. Our truth is why twitter is struggling, 2015. URL https://medium.com/@web/our-truth-is-why-twitter-is-struggling-ffc6a4e02bcd#.b2v4908pi.

[77] Wei Song, Yingying Qiao, Soon Cheol Park, and Xuezhong Qian. A hybrid evolutionary computation approach with its application for optimizing text document clustering. *Expert Systems with Applications*, 42(5):2517 – 2524, 2015. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2014.11.003. URL http://www.sciencedirect.com/science/article/pii/S0957417414006861.

[78] Damiano Spina, Julio Gonzalo, and Enrique Amigó. Learning similarity functions for topic detection in online reputation monitoring. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 527–536, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2257-7. doi: 10.1145/2600428.2609621. URL http://doi.acm.org/10.1145/2600428.2609621.

[79] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[80] Kate Starbird and Leysia Palen. (how) will the revolution be retweeted?: Information diffusion and the 2011 egyptian uprising. In *Proceedings of the*

ACM 2012 Conference on Computer Supported Cooperative Work, CSCW '12, pages 7–16, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1086-4. doi: 10.1145/2145204.2145212. URL http://doi.acm.org/10.1145/2145204.2145212.

[81] Internet Live Stats. Twitter usage statistics, 2016. URL http://www.internetlivestats.com/twitter-statistics/.

[82] Stefan Stieglitz and Linh Dang-Xuan. Emotions and Information Diffusion in Social Media—Sentiment of Microblogs and Sharing Behavior. *Journal of Management Information Systems*, 29(4):217–248, apr 2013. ISSN 0742-1222. doi: 10.2753/MIS0742-1222290408. URL http://www.tandfonline.com/doi/full/10.2753/MIS0742-1222290408.

[83] Mani R. Subramani and Balaji Rajagopalan. Knowledge-sharing and influence in online social networks via viral marketing. *Commun. ACM*, 46(12):300–307, December 2003. ISSN 0001-0782. doi: 10.1145/953460.953514. URL http://doi.acm.org/10.1145/953460.953514.

[84] Guoyu Tang, Yunqing Xia, Weizhi Wang, Raymond Lau, and Fang Zheng. Clustering tweets using wikipedia concepts. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland, may 2014. European Language Resources Association (ELRA). ISBN 978-2-9517408-8-4.

[85] Santipong Thaiprayoon, Alisa Kongthon, Pornpimon Palingoon, and Choochart Haruechaiyasak. Search result clustering for thai twitter based on suffix tree clustering. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2012 9th International Conference on*, pages 1–4. IEEE, 2012.

[86] Yannis Theocharis, Will Lowe, Jan W van Deth, and Gema García-Albacete. Using Twitter to mobilize protest action: online mobilization patterns and action repertoires in the Occupy Wall Street, Indignados, and Aganaktismenoi movements. *Information, Communication & Society*, 18(2):202–220, 2015. doi: 10.1080/1369118X.2014.948035. URL http://dx.doi.org/10.1080/1369118X.2014.948035.

[87] Mark Tremayne. Anatomy of Protest in the Digital Era: A Network Analysis of Twitter and Occupy Wall Street. *Social Movement Studies*, 13(1):110–126, 2014. doi: 10.1080/14742837.2013.830969. URL http://dx.doi.org/10.1080/14742837.2013.830969.

[88] H. Tu and J. Ding. An efficient clustering algorithm for microblogging hot topic detection. In *2012 International Conference on Computer Science and Service System*, pages 738–741, Aug 2012. doi: 10.1109/CSSS.2012.189.

[89] Zeynep Tufekci. "Not This One". *American Behavioral Scientist*, 57(7):848–870, 2013. doi: 10.1177/0002764213479369. URL http://dx.doi.org/10.1177/0002764213479369.

[90] A. Tumasjan, T.O. Sprenger, P.G. Sandner, and I.M. Welpe. Predicting elections with twitter: What 140 characters reveal about political sentiment. In *Proceedings of the Fourth International AAAI Conference on Weblogs and Social Media*, pages 178–185, 2010. URL http://scholar.google.de/scholar.bib?q=info:mc319eHjea8J:scholar.google.com/&output=citation&hl=de&as_sdt=0&ct=citation&cd=28.

[91] Twitter. #numbers, 2011. URL https://blog.twitter.com/2011/numbers.

[92] Twitter. Faqs about retweets, 2016. URL https://support.twitter.com/articles/77606.

[93] Twitter. Tweet activity dashboard, 2016. URL https://support.twitter.com/articles/20171990.

[94] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[95] UNHCR. Syria regional refugee response. http://data.unhcr.org/syrianrefugees/country.php?id=224, 2016. Accessed 2016.08.02.

[96] John Vanderzyden. What is elasticsearch, and how can i use it?, 2015. URL https://qbox.io/blog/what-is-elasticsearch.

[97] Vincent Vanhoucke. Deep learning, 2017. URL https://classroom.udacity.com/courses/ud730.

[98] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[99] David H. Wolpert. Off-training set error and a priori distinctions between learning algorithms. Technical report, The Santa Fe Institute, 1995.

[100] Worldometers. Syria regional refugee response. http://www.worldometers.info/world-population/turkey-population/, 2016. Accessed 2016.08.02.

[101] R S Wyer, C Chiu, and Y Hong. *Understanding Culture: Theory, Research, and Application*. Psychology Press, 2009. ISBN 9781848728080. URL https://books.google.com.tr/books?id=W5{_}ZAAAAMAAJ.

[102] Xin Yan and Xiao Gang Su. *Linear Regression Analysis: Theory and Computing*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2009. ISBN 9789812834102, 9812834109.

[103] Jiang Yang and Scott Counts. Predicting the speed, scale, and range of information diffusion in twitter. *ICWSM*, 10:355–358, 2010.

[104] Tauhid R Zaman, Ralf Herbrich, Jurgen Van Gael, and David Stern. Predicting information spreading in twitter. In *Workshop on computational social science and the wisdom of crowds, nips*, volume 104, pages 17599–601. Citeseer, 2010.

[105] Oren Eli Zamir and Oren Etzioni. *Clustering web documents: a phrase-based method for grouping search engine results*. University of Washington, 1999.

[106] Juan Zamora, Marcelo Mendoza, and Héctor Allende. Hashing-based clustering in high dimensional data. *Expert Systems with Applications*, 62:202 – 211, 2016. ISSN 0957-4174. doi: http://dx.doi.org/10.1016/j.eswa.2016.06.008. URL http://www.sciencedirect.com/science/article/pii/S0957417416302895.

[107] Qi Zhang, Yeyun Gong, Jindou Wu, Haoran Huang, and Xuanjing Huang. Retweet prediction with attention-based deep neural network. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 75–84, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4073-1. doi: 10.1145/2983323.2983809. URL http://doi.acm.org/10.1145/2983323.2983809.

[108] Xue Zhang, Hauke Fuehres, and Peter A Gloor. Predicting Stock Market Indicators Through Twitter "I hope it is not as bad as I fear". *Procedia - Social and Behavioral Sciences*, 26:55–62, 2011. ISSN 1877-0428. doi: http://dx.doi.org/10.1016/j.sbspro.2011.10.562. URL http://www.sciencedirect.com/science/article/pii/S1877042811023895.

[109] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[110] Arkaitz Zubiaga, Damiano Spina, Raquel Martínez, and Víctor Fresno. Real-time classification of twitter trends. *Journal of the Association for Information Science*

and Technology, 66(3):462–473, 2015. ISSN 2330-1643. doi: 10.1002/asi.23186. URL http://dx.doi.org/10.1002/asi.23186.