UNIVERSITY OF CALIFORNIA, SAN DIEGO

A Real-Time Temporal Clustering Algorithm for Short Text, and its Applications

A Thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science

by

Nishant Agarwal

Committee in charge:

Julian John McAuley, Chair
Amarnath Gupta
Arun Kumar Kumar

2017

The thesis of Nishant Agarwal is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

Chair

University of California, San Diego

2017

DEDICATION

I dedicate this to my parents.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my advisor Dr. Amarnath Gupta for the continuous support of my thesis and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for the research I did here.

Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Julian McAuley and Prof. Arun Kumar, for their encouragement, insightful comments, and support.

My sincere thanks also goes to Informatica, for offering me the summer internship opportunities in their Data Incubation Lab, where I worked on exciting research projects.

I thank my fellow classmates in UCSD: Priyanka Dighe and Shivani Agrawal, for the stimulating discussions and suggestions along the course of development of this research. Also I thank Dr. Subhasis Dasgupta, for collaborating in this research and helping me extend this in other domains.

Last but not the least, I would like to thank my family: my parents Mr Satish Agarwal and Mrs. Poonam Agarwal, for making me capable to reach such a prestigious institution as UCSD and supporting me spiritually throughout my life.

VITA

2013	B. E.(Hons.) in Computer Science *cum laude*, BITS Pilani University, Goa
2013-2015	Software Developer, Oracle Server Technologies, Data integration group, Bangalore
Sept 2015- Dec 2015	Teaching Assistant for Data Integration and ETL Course, UCSD
Jan 2016- June 2016	Research Assistant , San Diego Supercomputer Center
Sept 2015- Dec 2015	Software Engineer Intern, Data Incubation Lab, Informatica LLC, Redwood City

PUBLICATIONS

Nishant Agarwal, Reza Karimpour, Guenther Ruhe, "Theme-based Release Planning: An Analytical Approach", *HICSS '14 Proceedings of the 2014 47th Hawaii International Conference on System Sciences*, Pages 4739-4748, 2014.

ABSTRACT OF THE THESIS

A Real-Time Temporal Clustering Algorithm for Short Text, and its Applications

by

Nishant Agarwal

Master of Science in Computer Science

University of California, San Diego, 2017

Julian John McAuley, Chair

Real-world events of general interest trigger engaging discussions among people for short bursts in time. These events then either evolve, merge or slowly lose public interest as time goes by. With the advent of social media like Twitter, people have a platform to voice opinions, share news and make announcements, some of which gain mass popularity. Leveraging social media for early detection of these events, has thus become a problem of immense practical value for news media companies and people who want to be aware about the world in general.

In this thesis, we present the foundations of a system that runs on a real-time, dynamic, temporal clustering algorithm, that is capable of detecting bursts in streaming

data, as they happen in time. By using efficient data structures, we not only detect these bursty topics, but also store them efficiently, to facilitate many independent applications such as hashtag recommendations and community detection, which would requires a completely different approaches, if handled separately. An efficient storage mechanism also enables us to perform evolutionary queries on the discovered themes, to get a greater insight. We also extend this research on social media, to user behavioral analysis, using their database access logs, to find bursty patterns, anomaly detection and related tasks. Lastly, we perform an in-depth evaluation to show that our model outperforms many popular approaches in terms of topic quality and hashtag precision by more than 15%.

Chapter 1

Introduction

1.1 Overview

Consider a scenario where you are walking down the street, and you overhear a couple discussing something about Donald Trump. Your curious mind doesn't allow you to let go and it really wants to know what are the topics of discussion related to Mr. Trump happening currently. In your quest for enlightenment, you turn to the existing technologies which are popular for trend analysis. Let's start with Google Trends. Figure 1.1 shows the results one might get when they query "Trump" on Google Trends. While it gives an estimate of the changing interests of people on the topic, and related queries, it still does not give me any information about what are the key discussions going around this keyword, and insights into those topics. If they put a similar query to Twitter, the results contain a bunch of tweets by some users, and some news articles, most of which talk about the same incident. It's hard to make sense out of user tweets without context, as these have a short character limit. We are still not able to get answers to queries such as, "when did this event start gaining popularity", "what are the key users involved in the discussion", "what are the emerging hashtags about this topic", "which of these are

gaining popularity at the fastest rate", etc. Same is the problem with Google News. We get a list of news articles, but its hard to gain insights about these events without actually reading them and aggregating information from different sources. Later we show with examples that sometimes, when news originates from Twitter, there is a significant delay in Google search to detect and show it as news in its search results.

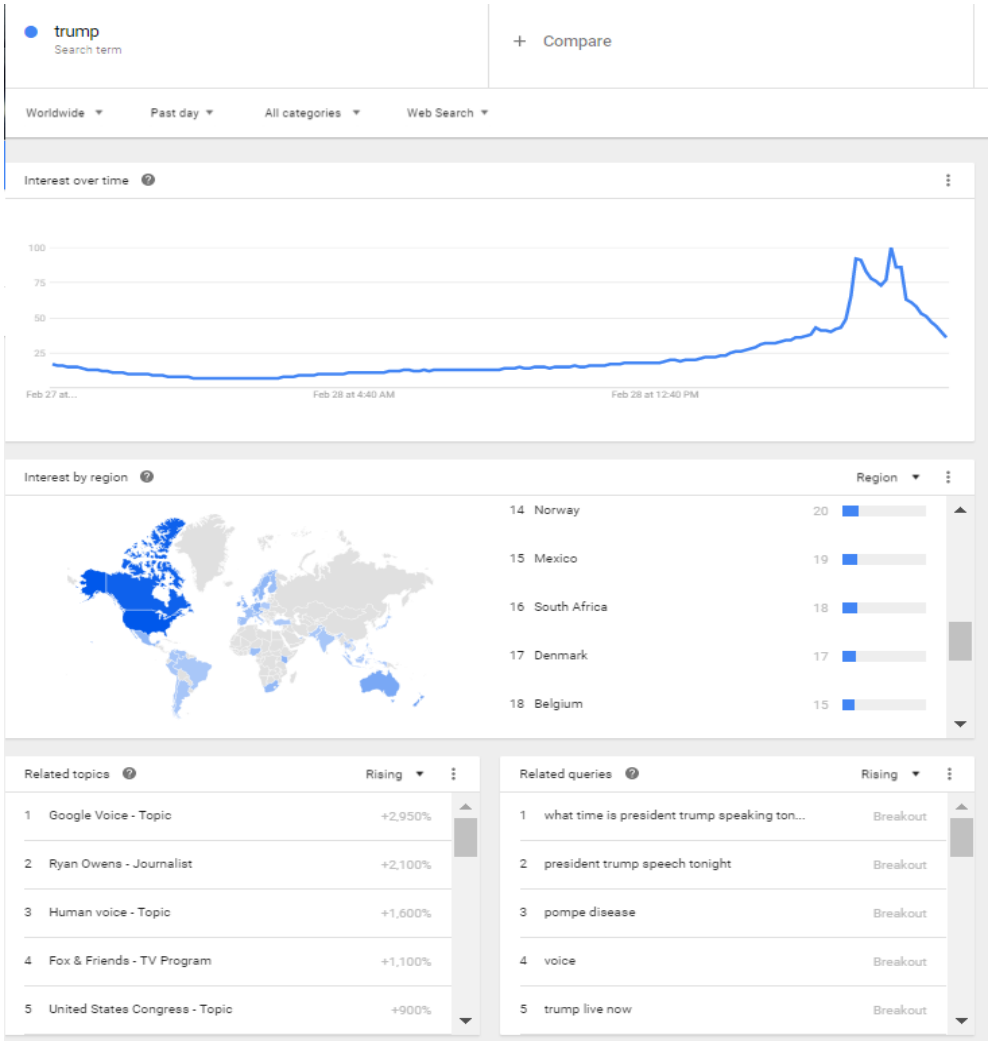


Figure 1.1: Results of a search query about Trump on Google Trends

Now envision a system where when you put a search query about entities like "Trump", you get a group of "themes", each depicting a topic of discussion about Mr. Trump as it is happening in real-time. Figure 1.2 shows a snapshot of a theme. The

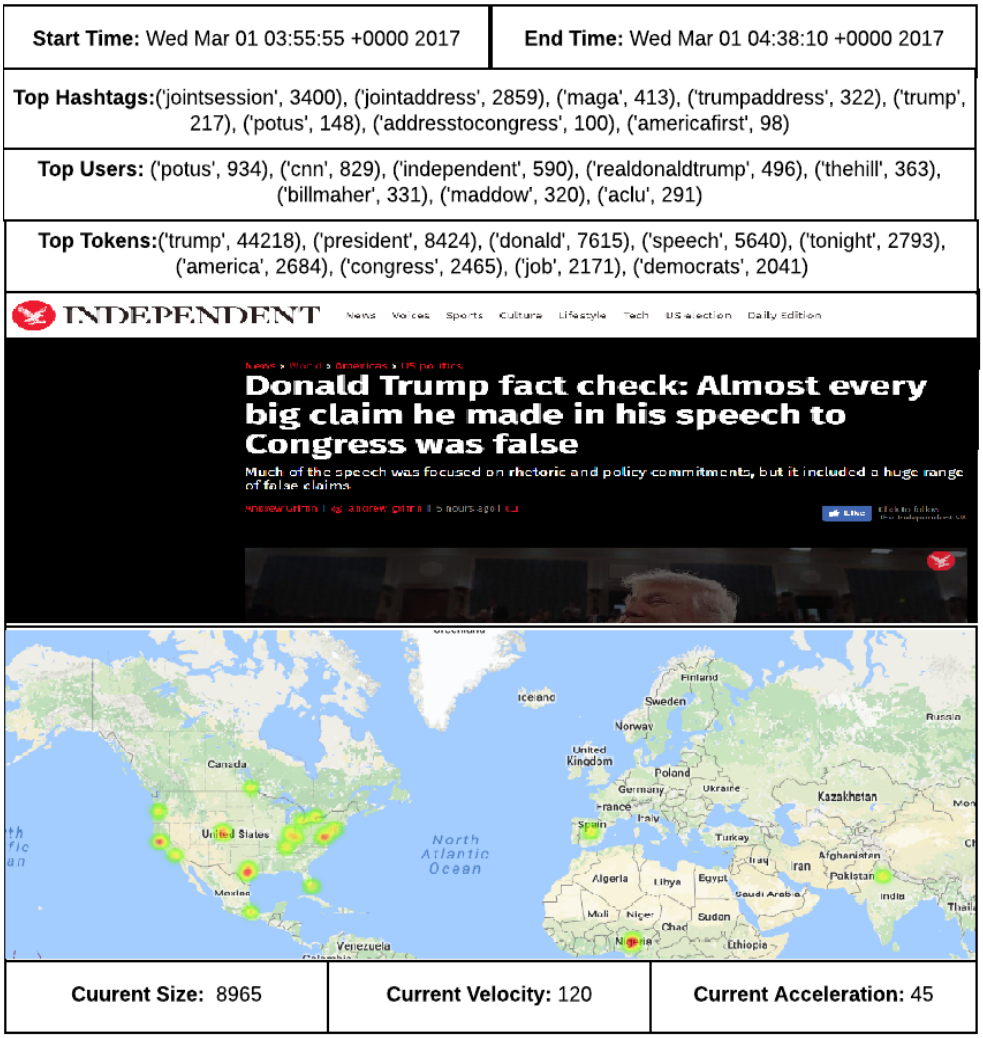


Figure 1.2: Sample Detected theme

theme evolves in real-time as new tokens arrive, users discussing it change and so on. It specifies the time window when it was detected, and also shows important links to news articles about that discussion, and statistics about rate of growth. It also provides additional functionality of performing queries on it, in order to get deeper insight, which is not provided by news articles. This is the system we build as a part of this thesis. We also show that such a system can be extended to perform various complex research tasks such as hashtag recommendations, community detection, etc. simultaneously in real-time. We do so by building a temporal clustering algorithm, that is tuned to detect bursts in

real-time and persist their snapshots in efficient data structures, which allow advanced queries to be performed on them.

Thus,the main contributions of this research can be summarized as follows-

- We present a real-time, online theme detection system that monitors these themes as they evolve, and report it as an event when their growth rate exceeds the average.
- The system is also capable of performing advanced queries on the groups of clusters, which facilitates deeper understanding of the event.
- We show that the same system can be used to perform many relatively complex and independent tasks like hashtag recommendation and community detection for Twitter, simultaneously.
- We also extend the algorithm to user log analysis and discuss its applications.

Note that significant research has been performed to detect topic bursts in data but only a few of these studies throw light on real-time detection [AMRW12]. Moreover, a system that is capable of doing this, along with providing support for advanced queries and other applications simultaneously has not been built yet.

The thesis is structured as follows- in Chapter 2 we perform a background study of existing research that have attempted to solve similar problems. Chapter 3 and 4 describe the problem statement, design decisions, and the theme detection algorithm that performs clustering of bursty topics. Chapter 5 discusses a theoretical underpinning of the approach and shows its relationship to a point process, which tries to model a topic burst. Chapters 6 and 7 discuss research problems like early event detection and hashtag recommendation and how these can be solved with the current algorithm. In Chapter 8, we discuss a completely new research problem and show that the algorithm can be tweaked to perform various complex tasks in user behavior analysis as well. In Chapter

9, we introduce some analytic operators that enable the users to perform complex queries on the collected snapshots of themes. In Chapter 10, we evaluate the model with known approaches and finally, we conclude the thesis in Chapter 11.

Chapter 2

Background Study

Short, multi-dimensional, temporally varying data is found in abundance these days. Be it tweets from twitter, or user logs from different applications, reviews of products from e-commerce websites, etc. all these kinds of data have a common intrinsic property. Each data point is concise, it captures a diverse set of attributes in different dimensions, they exhibit temporal dynamics, i.e. they span over some period of time and encode changing patterns of user behavior/ events.

With streaming data, leveraging time and content information simultaneously creates improved clusters. In the case of micro-blog data, where people discuss global topics, incidents, etc. many clusters evolve around real-life events. Temporal dynamics of different types of clusters may vary differently. Clusters or themes relating to background topics show a steady rate in popularity, while sudden events show a sharp burst in their popularity and they ultimately die out. These bursts attract a lot of research as it creates room for evolutionary queries such as, “when did it start becoming popular”, “what are the emerging entities within each topic”, etc. In this section, we discuss the approaches people have tried for similar problems. In order to gain clarity on existing solutions, we investigate the research along two axes. The first one indicates the way in which

the topics are defined, which leads us into a discussion on Clustering vs. Topic Model Based approaches. The second axis is the data processing technique, where we discuss differences between batch-based vs. online/real-time processing.

2.1 Clustering vs. Topic Modeling

Most clustering based systems define a topic as a collection of related documents. Using some distance measures, they compute the proximity of documents with one another to construct clusters. Yang et al. [YPC98] use refined hierarchical and online document clustering algorithms to detect events from a news stream. In [APL98a] each document is represented as a TF-IDF vector. Each new incoming document is compared against earlier vectors. If the new point is close enough to its nearest neighbor, it is absorbed by its nearest neighbor. Otherwise, this new document is labeled as a new event. Brants et al. [APL98b] extend [APL98a] by using incremental TF-IDF model and a sophisticated similarity score. However, this approach has many drawbacks. Finding k-nearest neighbors is a costly operation, and it is not suitable to handle huge data loads as of Twitter. Moreover, TF-IDF measure is not a good representation for tweets, as these are short documents with a low probability of having repeated words. In many studies, a topic is defined as a coherent set (or cluster) of key words [CCS13], [HJ10] or hashtags [AMRW12], [FZZ$^+$15]. In such works, usually a collection of bursty terms are detected from the document stream based on some criteria. Possibly later, these bursty terms are grouped into several clusters that represent the bursty topics. However, these methods fail to utilize the co-occurrence patterns in keywords.

LDA[BNJ03] is considered as one of the basic topic model that was initially designed for document corpora. Topic modeling for temporally changing micro-blogs has more complications as compared to performing the same on a corpora of documents.

1). They have a diverse and dynamically shifting vocabulary that is created due to wide variations in abbreviations, misspellings and proper nouns. Hence, any technique relying on a fixed dictionary based feature vector is not likely to perform well in the case of micro-blogs. 2). As the length of a tweet is short, assuming that it contains a distribution of k topics is not accurate. 3). Many temporal topic models try to find topics by dividing time in epochs and running topic modeling on each epoch. However, deciding the size of each epoch is a non-trivial problem, and we may get different results for different cases. 4). LDA and its variants assume a fixed number topics in the beginning, which is not true in this scenario because new topics constantly emerge and older ones are discarded.

Variants of LDA: Significant research has been published with respect to topic modeling on micro-blogging sites, document classification, tag recommendation for a corpora, etc. Many studies have tried to configure LDA to handle micro-blog data. Algorithms like [AMS16] tried to increase the length of documents by aggregating tweets in conversation groups for effective application of topic models on short-texts. To prevent assignment of a tweet to a topic distribution, Twitter-LDA algorithm [ZJW$^+$11] assigned one topic per tweet. However it could not be extended to do online inference or provide recommendations. Evolution of topics over time was introduced by Blei in [BL06]. Many variants like [KTU$^+$13] have been developed using this concept for Twitter. We construct a similar dynamic topic model with stochastic Gibbs sampling to compare our results later. Other related work includes [WZHS07], which builds a topic model to find correlated bursty topics from text streams, and [YCL$^+$13], which creates a unified model to distinguish temporal topics from background ones. In recent works [HAG$^+$12], [YCM$^+$13], topic models are built to discover geographical topics from Twitter. Although the direct focuses of these works are not on bursty topics, using a similar way as above, geographical bursty topics could be found. Wei et al. [XZJ$^+$16] were among the few who were able to perform topic modeling for bursty topic detection in a real-time setting. However, their

system could not be extended to perform evolutionary queries, community detection, etc as they collected all these features as a single list. Considering different arrival rates of these features, we might not have enough user-names and hashtags to recommend.

2.2 Retrospective vs. Real-time processing

In early work Yang et al. [YPC98] propose methods for both retrospective and online event detection. The main difference between retrospective view and online processing is that, retrospective view assumes the entire corpora of data to be present for evaluation at once. On the other hand, in the case of online event detection, the system processes current document before looking at any subsequent documents. In many cases, [YPC98] the results of retrospective detection results can be significantly better than the online one, as more information is available from a retrospective way. Therefore, most topic model based approaches fall into this category. The complexity of their models makes them good at learning topics from the data in a retrospective way, but at the same time lose the flexibility to respond to any new incoming data. In contrast, online methods only need to maintain a statistic for each term, and report the terms when their statistics exceed the significance level. However, the detected topic which consists of few keywords is far less informative than the topic learned from topic modelling, which is a distribution over all the words. The subtle difference between real-time and online processing is that in real-time detection, time is crucial, so much so that no fixed time window for detection should be assumed. By incrementally updating the statistic in an efficient way, SigniTrend [SWK14] can detect bursty keywords in real-time, but before it aggregates keywords into larger topics, it needs to wait until the end-of-day (or a fixed time period).

Chapter 3

Problem Formulation

As mentioned above, the central idea of the thesis is to detect specific patterns, called "themes" in streaming data in real time. More formally, a *theme* is a latent representation of a popular discussion topic, extracted from a stream of tweets. A single stream can have multiple themes existing at once, evolving at different rates. In our definition, we use most recent hashtags, users and tokens as features that define a theme and to decide whether an incoming tweet should be a part of the theme. Other features like web links and geotags are not used for comparison, but to get deeper insight into the content of the theme.

3.1 Problem statement

Given a stream s of incoming short-text, find a set of themes T that evolve with time, and represent potentially different patterns in the incoming data. Using these themes, monitor their growth and eventually detect bursts as they happen.

Most of the thesis discusses temporal theme detection and evolution over Twitter data. Hence, later portions of the thesis use Twitter terminology of hashtags, user-mentions and tokens(which are essentially keywords in a tweet), heavily. The process

of theme discovery should incorporate temporal similarity in addition to contextual similarity, in order to generate more relevant clusters. Also, hashtags loosely describe the topic of the tweet. Hence, tweets with co-occurring hashtags should have a higher probability of being assigned to the same cluster.

3.2 Preprocessing and Feature Selection

Currently, the data we are using is live Twitter feeds related to a variety of subjects like the US elections, presidential candidates, India-Pakistan tension over terrorism, Apple, etc. A tweet is only considered for evaluation if it is a valid JSON object. Tweets with less than three words in the text field are ignored. For every incoming tweet, we remove all punctuation (hashtags and user-mentions are handled separately). All text is converted to lower case and each word is lemmatized so that different variations of a word are mapped to the same root word. Stop words like articles and conjunctions that do not have a special meaning to them are removed and only useful string tokens are considered for evaluation.

A feature vector is constructed from a tweet in the following manner. After initial pre-processing we divide the tweet into three lists representing hashtags, user-mentions and important tokens, respectively. A similarity function compares this data structure against current themes in the system to identify the best fit.

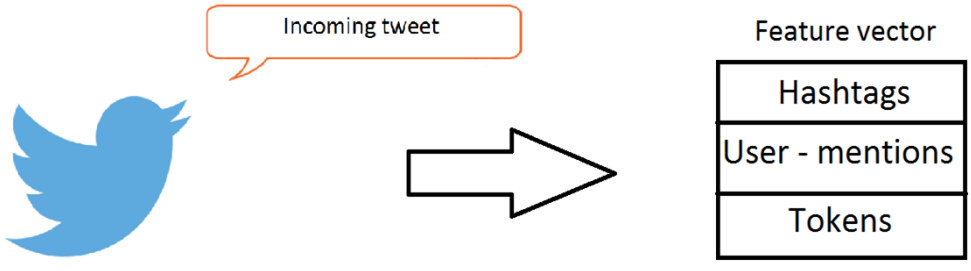


Figure 3.1: Converting a tweet to a feature vector.

3.3 Theme Representation

In our setting, a theme is defined as a collection of features, specifically hashtags, user-mentions and tokens, represented by a list of *attribute-value pairs* for each feature. The *attribute* represents the string value of the feature (e.g., #IamWithHer) and the *value* represents the number of occurrences of this attribute within this feature within this theme. In the algorithm, we define a theme as a class with the following attributes:

- *Theme Size*: This attribute stores the number of tweets that have been assigned to this theme.

- *Least Recently Used(LRU) caches* : An LRU cache [lrub, lrua] is a well-known data structure that has a finite capacity and when it gets full, it chooses the least recently used key to be discarded. We have 3 LRU caches for hashtags, user-mentions and important tokens receptively – each stores key-value pairs where a key is an entity and value, their frequency.

Whenever a tweet is assigned to a theme, the size attribute is incremented by 1. Other useful metadata like geolocation of the tweet, links shared in the tweet, etc are also stored as a part of the theme, to facilitate advanced queries on them. Tweets have a dynamic vocabulary, with acronyms, slang, pronouns, etc. A cache is better than a fixed word vector because in a cache, any new word can be added as a key value pair. Also, as time goes by, any hashtag that is popular globally will tend to show up more frequently in tweets, therefore, stay in the cache, while the infrequent ones will be sieved out. This results in better quality of hashtag recommendations for a target tweet t. When the target tweet t is matched to a theme T_i, we output its most frequent hashtags as recommendations.

3.4 LRU vs. Sliding Windows vs. LFU

As mentioned above, we use 3 LRU caches for hashtags, user-mentions and string tokens. This is a better design as compared to using a single sliding window to store multiple tweets or a Least Frequently Used(LFU) caches because of the following reasons.

- Since tweets follow a mixed model of stochastic temporal process [ZS14, RXS$^+$16], hashtags, user-mentions and tokens, despite their co-temporality, have very different arrival rates. Decoupling them from each other allow them to update independently and store past information proportional to the length of the individual LRU caches.

- An LRU scheme helps us to calculate the weights for the recency of overlapping terms between the cache and incoming tweet, in the similarity function. An LFU cache does not store this information.

- A LFU strategy might hinder the detection of theme evolution. Suppose a particular term was very frequent in the past but currently it is not popular. It will still continue to remain at the top of the cache until all other members attain a frequency higher than it, which might take a long time. Thus it reduces the effective utilizable length of the cache. On the other hand, in an LRU strategy, it will be removed if it is not observed for some time, as elements are arranged in the order of their recency, thus allowing themes to evolve into new ones.

- An LRU cache stores information as key value pairs. Saving individual tweets with a sliding window requires more space. Thus, an LRU cache provides efficient utilization of memory which is required for real-time evaluation.

Chapter 4

Theme detection Algorithm

In this section, we describe the temporal clustering algorithm. We perform

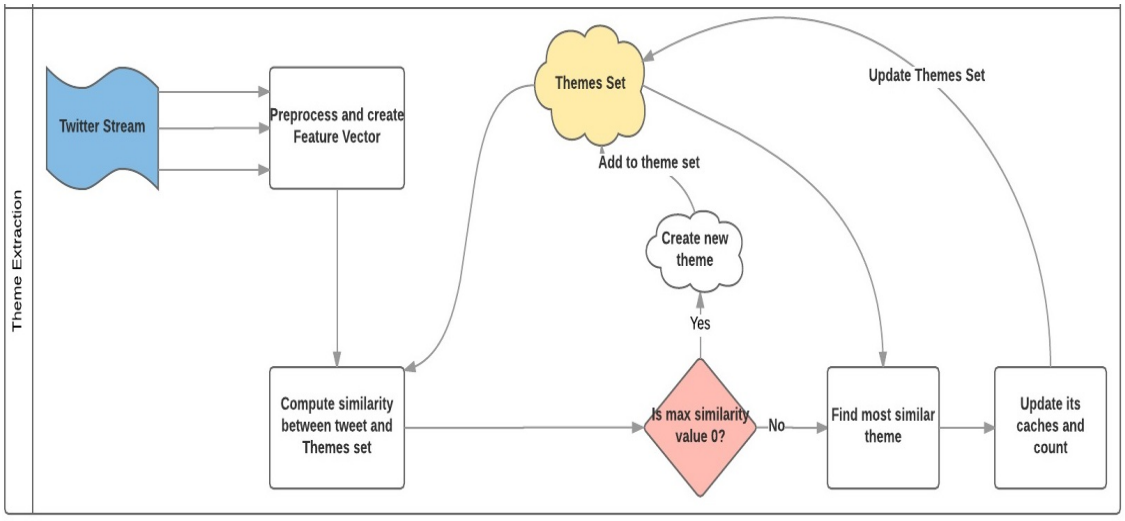


Figure 4.1: A flowchart outlining the primary steps of our temporal clustering algorithm

Figure 4.1 shows a pictorial representation of Theme Detection.

4.1 Similarity function S(t, T_i)

The similarity function computes the similarity between an incoming tweet t with a theme T_i. The tweet t is assigned to the theme with which it has a maximum similarity. If there is no match with any of the current themes(similarity is 0), we create a new theme for this tweet. It is defined as:

$$S(t,T_i) = \lambda_1 H + \lambda_2 U + \lambda_3 W + c \tag{4.1}$$

where λ_i are the weights proportional to the recency of the overlapping members of the attribute. Let $K_{t_j \cap T_{ij}}$ be the set of common keys between tweet t's jth list(e.g., hashtag list, users list etc.) and corresponding cache j in Theme T_i, $j \in \{1,2,3\}$. Let $Idx(K,T_{ij})$ be the sum of indices of keys K in cache T_{ij}, the index of more recent key being higher than the older one. Hence $\lambda_j = \frac{Idx(K_{t_j \cap T_{ij}}, T_{ij})}{length(T_{ij})}$. Let $Value(k,T_{ij})$ be the frequency value of a key k in the cache T_{ij}. Also, let $Csm(j)$ be the contextual similarity of an incoming tweet with an entity j(Hashtags, users and words) in the Theme T_i. We define it to be-

$$Csm(j) = \frac{\sum\limits_{k \in K_{t_j \cap T_{ij}}} Value(k,T_{ij})}{\sum\limits_{k \in Keyset(T_{ij})} Value(k,T_{ij})} \tag{4.2}$$

where, $H = Csm(j=1)$, $U = Csm(j=2)$ and $W = Csm(j=3)$. Hence, the similarity function is a normalized weighted measure of the degree of overlap between each tweet attribute with a corresponding theme. λ_j gets a high value if all the matching keys are recent in that cache. The underlying assumption is, in a short time interval, if a tweet is recently assigned to a theme, then another similar tweet coming shortly after it might be assigned to the same theme.

The term c denotes the hashtag co-occurrence bias which is given by the formula

$c_i = h_{matched(i)} - 1$. $h_{matched(i)}$ is the number of common hashtags between the incoming tweet and the theme T_i. So if there is only 1 hashtag that has matched, then c=0, and there is no co-occurrence bias, and the similarity is calculated based on other terms. If more than one hashtags are common between the tweet and the theme, then c is non-zero and there is a greater probability of the tweet being assigned to the theme with more hashtag matches. This is done under the assumption that a hashtag is a key descriptor of a theme, and a tweet having a greater overlap of hashtags with a theme, belongs to that theme. If the overlap is same, then other factors govern the assignment.

4.2 Theme Extraction Algorithm

The pseudo-code of our algorithm is described below.

Algorithm 1 Theme Extraction algorithm

```
1:  procedure CONSTRUCT MODEL(tweet, themes)
2:      for T_i in themes do
3:          similarity_i ← S(t, T_i)
4:          max_s ← max(max_s, similarity_i)
5:      end for
6:      if max_s = 0 then
7:          T_j ← newTheme()
8:          themes ← themes.append(T_j)
9:          idx ← themes.length
10:     else
11:         idx ← argmax(similarity)
12:     end if
13:     themes ← themes.assign(idx, tweet)
14:     return idx                          ▷ return the index of theme
15: end procedure
```

The Construct Model method is invoked every time a new tweet arrives. We find the similarity of every tweet with current themes based on the similarity function. If the maximum similarity(max_s) from all current themes is zero, then we create a new theme

and assign the tweet to it, else we assign it to the most similar theme. Assignment here refers to updating individual caches of the theme based on the keys in the tweet. Here, *idx* is the index of the theme where assignment should take place.

Whenever a tweet is assigned to a theme, we update its size count. Its metadata is also stored in a list/cache to construct heatmaps for events, etc. Let $count_a$ be the counts vector containing size of each theme at time=a. We define a velocity vector as:

$$velocity_a = count_a - count_{a-1} \tag{4.3}$$

$$acceleration_a = velocity_a - velocity_{a-1} \tag{4.4}$$

where the velocity at time=a is the difference of the count vectors between current and previous time slice. After fixed time intervals, the slowest growing themes are removed, to prevent an outburst of total number of themes. In the actual implementation, we limit the maximum number of themes to a constant. Space for new evolving themes is continuously created by the evacuation scheme described above.Note in the theme creation algorithm that our model is completely data driven and makes no assumption about the distributions from which the themes arise. In Section 6, we show one example of how this non-assumption helps the quality and speed of our results.

4.3 Runtime Analysis

Let the number of terms present in a tweet be n and let K be the number of themes present in the system. The Construct Model method is invoked for each tweet as it arrives. The similarity function $S(t, T_i)$ compares the tweet t with a theme T_i to compute a similarity value. The denominator term, in Contextual Similarity $Csm(j)$, which is the running sum of all values in the cache j is updated everytime the matching keys are

updated in cache j. The update and insert operations on the cache take $O(1)$ time. As the similarity computation is done for each term in the tweet t, it takes $O(n)$ time to compute the Contextual Similarity between a tweet and a theme, n being number of terms in a tweet. Similarly, to compute λ_i terms, which determine the indices of each matching term in tweet t in its respective cache, take $O(n)$ time. If there are K themes in the system, the total time to compute the similarity values against all themes, and finding the maximum similarity is $O(nK)$.

The length of each cache is fixed, which makes the model scalable. As new keys arrive in the cache, older ones are removed, limiting the size of the data which is stored as themes. At regular intervals, the theme caches are flushed to avoid overflow of frequency values.

4.4 Theme Recovery

As topics are incrementally added if incoming tweets are not similar to the existing ones, we cannot allow total number of themes to arbitrarily increase in number. Hence there are two ways which we use to limit the number of themes.

- **Velocity Check:** To detect stale events, we monitor the slowest growing clusters based on their velocities and acceleration. After fixed intervals (some k times the polling intervals), such themes are removed so that new ones can be allocated in that space.

- **Inter-cluster distance:** Theme discovery on temporal data leads to instances where many such themes start as different ones, but eventually merge into one after sometime. We define a inter-cluster distance measure to monitor the distance between current themes. At fixed intervals, closest clusters are merged together,

according to the KMeans algorithm. As this is an expensive operation, it is done on persisted data in a separate thread.

Chapter 5

Relationship with Point Processes

A point process N is a random process in which any realization consists of a collection of points typically representing the times and locations of events [CCMS10]. One such process is the Hawkes process, which is a mathematical tool for modeling recurrent patterns and continuous-time nature of real world events.

5.1 Hawkes Process

A temporal point process is a random process whose realization consists of a list of discrete events localized in time, $\{t_i\}$, with $t_i \in \mathbb{R}^+$. Considering the temporal nature of events in social networks data, they can be modeled as a temporal point process. A temporal point process can therefore be represented as a counting process, $N(t)$, which records the number of events before time t. Let the history τ be the list of event time $\{t_1, t_2, t_3, t_4, ...t_n\}$ upto but not including time t. Then in a small time window dt between [0, t), the number of observed event is

$$dN(t) = \sum_{t_i \in \tau} \delta(t - t_i)dt, \tag{5.1}$$

and hence $N(t) = \int_0^t dN(s)$, where $\delta(t)$ is a Dirac delta function. It is often assumed that only one event can happen in a small window of size dt, and hence $dN(t) \in \{0,1\}$

An important way to characterize temporal point processes is via the conditional intensity function – the stochastic model for the next event time given all previous events. Within a small window $[t, t+dt)$, $\lambda(t)dt$ is the probability for the occurrence of a new event given the history τ :

$$\lambda(t)dt = P\{event \in [t, t+dt)|\tau\} \tag{5.2}$$

The functional form of the intensity $\lambda(t)$ is often designed to capture the phenomena of interests. For instance, in a **homogeneous Poisson process**, the intensity is assumed to be independent of the history τ and constant over time, i.e., $\lambda(t) = \lambda_0 \geq 0$. In an **inhomogeneous Poisson process** , the intensity is also assumed to be independent of the history τ but it can be a function varying over time, i.e., $\lambda(t) = g(t) \geq 0$. In both case, we will use notation Poisson(λ(t)) to denote a Poisson process. A Hawkes process captures the mutual excitation phenomena between events, and its intensity is defined as

$$\lambda(t) = \gamma_0 + \alpha \sum_{t_i \in \tau} \gamma(t, t_i) \tag{5.3}$$

where $\gamma(t, ti) \geq 0$ is the triggering kernel capturing temporal dependencies , $\gamma_0 \geq 0$ is a baseline intensity independent of the history. The summation of kernel terms is history dependent and a stochastic process by itself. The kernel function can be chosen in advance, e.g., $\gamma(t, ti) = \exp(-|t - t_i|)$ or directly learned from data.

A key feature of Hawkes Process is the occurrence of each historical event contributes to the increment in intensity by a certain amount. Since the intensity function depends on the history τ upto time t, the Hawkes Process is essentially a conditional Poisson Process (or a doubly stochastic Poisson process) in the sense that conditioned on

the history τ , the Hawkes process is a Poisson process formed by the superposition of a background homogeneous Poisson process with the intensity 0 and a set of inhomogeneous Poisson processes with the intensity $\gamma(t,t_i)$. However, because the events in a past interval can affect the occurrence of the events in later intervals, the Hawkes process in general is more expressive than a Poisson process. Hawkes process is particularly good for modeling repeated activities, such as social interactions [ZZS13], search behaviors [LDD+14], or infectious diseases that do not convey immunity.

Given a time $t' \geq t$, we can also characterize the conditional probability that no event happens during $[t,t_0)$ and the conditional density that an event occurs at time t_0, using the intensity $\lambda(t)$, as

$$S(t') = \exp(-\int_t^{t'} \lambda(\tau)d\tau), \tag{5.4}$$

$$f(t') = \lambda(t').S(t') \tag{5.5}$$

Using these two quantities, the likelihood of the list of event times can be expressed. However, further details are out of scope of this thesis.

5.2 Connecting Hawkes process with our model

In the previous section, we described a temporal point process, called the Hawkes Process. Many researchers have used the Hawkes process in the context of event detection from news articles and micro-blogs. Nan Du et al. [DFA+15] use a Dirichlet process with a Hawkes prior on social media streams to detect topics in the temporal context. Eric Lai et al. [tim] use non-negative matrix factorization and Hawkes process to analyse time series in micro-blogs. All these methods build probabilistic models using these

temporal point processes and generate results based on it. However, as stated in prior work, most of these models struggle with changing vocabularies, hyper-parameter tuning, etc. Instead, we use temporal excitation to compute the best fit for a tweet among current set of clusters in real time, using a deterministic approach. We now prove how our cache-based scheme can be seen as a light-weight implementation of the temporal point process.

Consider the Equation 5.3. Here, in the mutual excitation intensity function, γ_0 is a constant while $\gamma(t,t_i)$ is a kernel function non-increasing in $|t-t_i|$. Thus, $\gamma(t,t_i) \geq \gamma(t,t_j)$ if $event_i$ is more recent than $event_j$. If we only record the last occurrence of an event, instead of considering its last n occurrences, then mutual excitation formula becomes-

$$\lambda(t) = \gamma_0 + \alpha.\gamma(t,t_i) \tag{5.6}$$

where t_i is the last occurrence of the event before time t.

In our approach, let us define an event as the assignment of a token to its respective cache. Now let us observe our similarity function-

$$S(t,T_i) = \lambda_1 H + \lambda_2 U + \lambda_3 W + c \tag{5.7}$$

Observe the contribution from a single cache, say the Hashtag cache. the Similarity contribution for this will be $S(t,T_i)_H = c + \lambda_1 H$. The λ_1 term is a normalized sum of the indexes of matching tokens in the LRU cache, with more recent index having a higher value. These indexes denote the last time that token was assigned to the particular cache. Therefore, between $event_i$ and $event_j$, if $event_i$ attributes to more recent tokens in the cache, the value of $\lambda_1 i > \lambda_1 j$. This is the exact behavior shown by the $\gamma(t,t_i)$ term in the mutual excitation equation.

The c term in the similarity expression can be assumed to be similar to the γ_0

term in Equation 5.6. The Hawkes process treats α to be a constant. However, in our method, we multiply the temporal term (triggering kernel), with the contextual similarity H. Thus, our similarity method can be seen as a superimposition of mutual excitation intensities of different features incorporated in the calculation. In the case of tweets, the features are hashtags, user-names, and string tokens.

In the next chapter we will discuss show how this theme detection mechanism can solve three important social network problems, namely, event detection, community discovery and hashtag recommendation, simultaneously.

Chapter 6

Application: Real-time Event detection

In this chapter, we will apply the theme detection algorithm to identify events in real-time. Since, the data is directly pulled from twitter, and Twitter serves as a platform for many personalities and agencies to make important announcements, this method is also capable of early detection of events which are strong candidates for becoming news items in the future. We will demonstrate one such example, which we observed during one of our experiments.

6.1 Overview

Twitter has risen to become a popular social networking website . People are using Twitter as a platform to voice their opinion, increase awareness, report events and conduct discussions. Thus we can exploit this platform by analyzing streams of tweets to detect events and predict real world outcomes. Event detection aims at finding real-world occurrences that unfold over space and time. Messages posted on Twitter currently exceed 400 million tweets per day, and they could reveal information about real-world events as they unfold. However, event detection from Twitter data must efficiently and accurately uncover relevant information about events of general or specific interest, which

is buried within a large amount of mundane information (e.g., meaningless, polluted, and rumor messages)In this paper, we aim to identify temporal topics and differentiate them from persistent ones. While extensive research has been done on event detection, the nature of Twitter data makes it particularly challenging to extract meaningful information. The vocabulary is also not fixed due to various acronyms and informal language used in tweets. Additionally, there is a huge percentage of tweets which that noisy and are not concerned with relevant topics.

6.2 Event Detection methodology

The clustering algorithm creates themes in real-time. Along with tracking the hashtags, users and tokens in separate caches, we also compute the velocities and acceleration for each theme by Equation 4.3, where velocity of a theme at time=a is is the difference of the theme sizes between current and previous time stamps. The topics with consistent high velocity (or acceleration) over consecutive fixed polling intervals are chosen as candidate topics for events. The demonstration of this method using the clustering algorithm is shown in Figure 6.1.

6.3 Event Definition

The similarity measure decides the assignment of a tweet to a theme. However, Twitter API exposes a lot of other attributes that can give valuable information about the event. Thus, as soon as a tweet is assigned to a theme, we record these attributes for the event to get an in-depth understanding of the event-

- **Time Statistics:** From the tweets being assigned to a theme, we update the time of creation of the theme and also the time at which the last tweet was assigned to

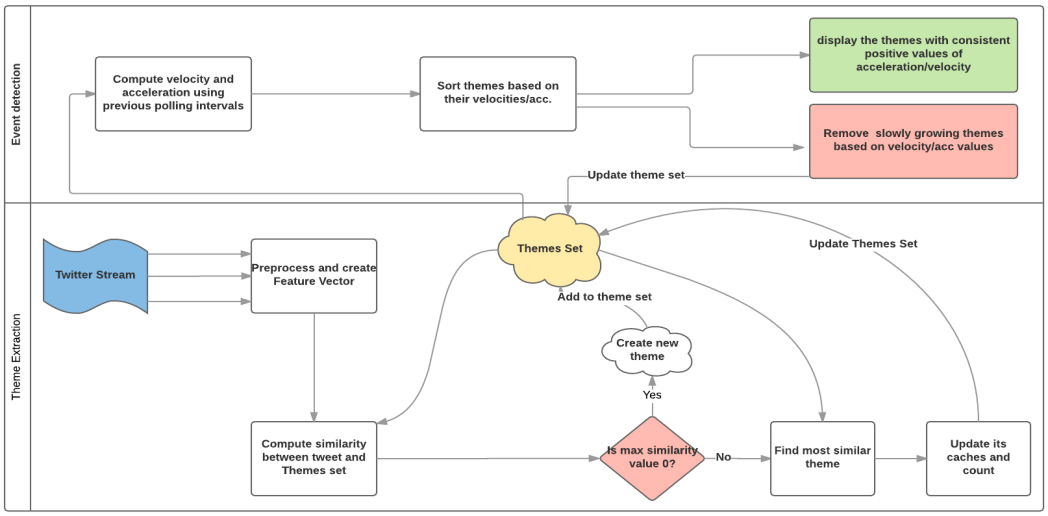


Figure 6.1: A flowchart for event detection

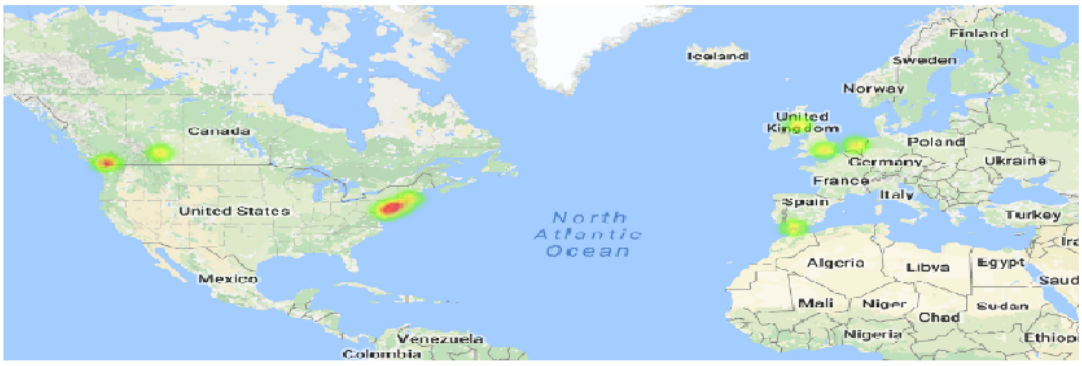
that theme. This helps us assess the time frame in which the theme is valid and helps in detecting stale ones. They are denoted as Start Time and End Time in the theme data structure.

- **Cache statistics:** Our three primary caches are polled for the current state. They are sorted based on the counts of their keys, and top 50 keys from each cache are displayed. The count values give a confidence value of the key being part of the current theme.

- **Popular Links:** Instead of stripping out links from tweet content, we use them as a source of validation for events. If the tweet assignment is correct, then the keywords in the caches should comply with the top links that are being shared in the tweets belonging to that cluster. Also, if is actually an event in real- world, then the top links will point to the actual news article or viral tweet that contains the keywords present in the caches.

- **Heat Map:** Among the tweets being assigned to a theme, latitude/longitude

Start Time: Sat Nov 19 22:32:27 +0000 2016	**End Time:** Sat Nov 19 23:00:29 +0000 2016
Top Hashtags: ('boycotthamilton', 1121), ('trump', 873), ('amjoy', 383), ('maga', 299), ('trumpuniversity', 155), ('hamilton', 148), ('notmypresident', 137)	
Top Users: ('goethelife', 1424), ('juddlegum', 1393), ('kurteichenwald', 1083), ('prisonplanet', 1013)	
Top Tokens: ('trump', 46573), ('hamilton', 5252), ('donald', 4600), ('cast', 3629), ('pence', 3610), ('supporters', 3001)	



Cuurent Size: 5287	**Current Velocity:** 101	**Current Acceleration:** -9

Figure 6.2: A detected event

information is extracted from the geo-tagged ones and they are mapped on Google Maps. This gives a good understanding of what regions of the world are discussing that event. This eventually helps in detecting communities, which will be discussed in the next section.

Such a sample event is shown in Figure 6.2. It corresponds to the news of a feud between

Hamilton cast and Donald Trump.

6.4 Some examples and observations

Here are some events that were detected using real-time twitter data, including topics like US Presidential elections, Brexit and Indian Politics.

6.4.1 Fast Event detection

Below are some figures that show that our approach is capable of detecting real-world events within minutes (or seconds) of them gaining popularity. To conserve space, we only show the sorted cache data and the corresponding news item with respect to the detected event.

There are many key observations that can be inferred from these examples.

- **Accurate detection of events:** It is evident from the results above that the algorithm can effectively find events. In Figure 6.3, if we observe the token list, we are able to find all the keywords in the headlines being present in the top keywords for the theme related to this event. Similar is the case with Fig. 6.4 - 6.6.

- **More elaborate than trending topics on Twitter:** Twitter provides a list of Top trending hashtags. However, hashtags alone do not give a good insight about the event. Our event representation provides not just the top hashtags being discussed in the event; it also gives user-names and important tokens being discussed. We also list the event creation times, geo-mapped tweets to specific regions in the world, and the velocities of events. Such an elaborate description of an event facilitates advanced analytics to discover key insights. For example, in Figure 6.6, apart from the information that the event is about brexit, we are also able to capture


```
Topic:  9
[('euref', 831), ('eurefresult', 16), ('eureferendum', 9), ('publictransport', 9), ('kent', 8), ('fmcg', 8)]
[('skynewsbreak', 160), ('tweediatrics', 158), ('telegraph', 114), ('carina3603', 70), ('gemma_chan1', 46)]
[('cameron', 418), ('david', 406), ('minister', 298), ('prime', 295), ('leave', 187), ('uks', 179), ('resigned', 169)]
```

Figure 6.3: An event of David Cameron resigning after Brexit

the top user who is tweeting about that topic, and key information like 180 billion Euro wipeout, etc.

- **Fast Detection:** As evident from the examples shown, the events are detected quite fast. For example, in Figure 6.5, the news item is detected within 2 minutes of it being published on a media website. The detection times are highlighted in other examples as well.

6.4.2 Early event detection

The key idea of this approach is that it consumes Twitter data in real-time to detect events. Celebrities, politicians and government agencies find it more convenient


```
Topic:  2
[('trump', 97), ('trump2016', 20), ('orlando', 13), ('prayforlilwayne', 10), ('p2', 10), ('uslatino', 10)]
[('washingtonpost', 78), ('sexuaitalk', 44), ('usatoday', 39), ('anncoulter', 37), ('realdonaldtrump', 36), ]
[('trump', 1949), ('donald', 585), ('washington', 213), ('post', 183), ('press', 171), ('credentials', 159), ]
```


```
Topic:  6
[('orlando', 44), ('rt', 32), ('follow', 30), ('topstories', 22), ('news', 13), ('worstpresidentever', 11), ('trump', 7)]
[('sheriffclarke', 62), ('abc', 26), ('realdonaldtrump', 17), ('writeintrump', 15), ('pincopallina93', 13), ('potus', 12)]
[('obama', 1005), ('orlando', 248), ('president', 188), ('victims', 138), ('terror', 131), ('shooting', 112), ('console', 69)]
```

Figure 6.4: Events of Donald trump and Obama


```
Topic:  4
[('euref', 652), ('brexit', 156), ('eurefresults', 101), ('uk', 47), ('leave', 34), ('health', 8), ('breakfastread', 6)]
[('timeshighered', 92), ('imgur', 77), ('mirrorpolitics', 44), ('bbcnewsbeat', 40), ('kp24', 34), ('markets', 27)]
[('uk', 263), ('level', 215), ('pound', 164), ('since', 139), ('right', 133), ('1985', 126), ('lowest', 118), ('brexit', 108)]
```


```
Topic:  1
[('brexit', 957), ('euref', 894), ('uk', 339), ('eurefresults', 336), ('leave', 218), ('eu', 192)]
[('skynewsbreak', 305), ('balloon_fish', 284), ('louisebrealey', 260), ('seimei1992', 214), ('emmakennedy', 198)]
[('uk', 11226), ('eu', 3885), ('leave', 2014), ('credit', 1106), ('rating', 1005), ('aaa', 990), ('lose', 987)]
```

Figure 6.5: Events related to Brexit

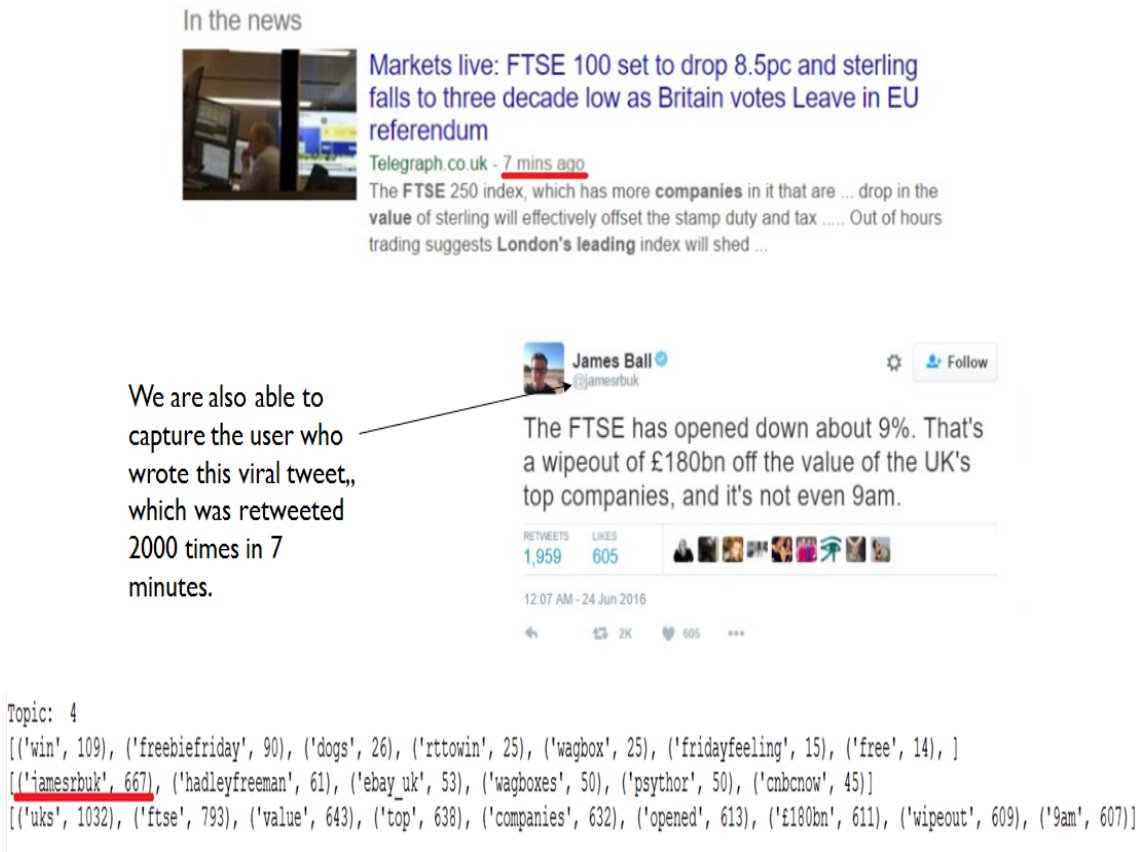


Figure 6.6: Some other Events related to Brexit

to make announcements via Twitter. Hence, there can be instances when we detect an event even before News articles have been published on them, just using Twitter data. One such incident is discussed in this section.

We detected an event as shown in Figure 6.7. The event is about how Government of India announced bilateral ties with Kenya. The event took place on January 4th, 2017. The event also shows regions in India where people are talking about it, and lists *mib_india* as the top user for the event, which is the Government agency that reported this.

To prove early detection works, we queried Google with the top keywords as soon as we detected the event from the algorithm. During detection, a Google search still

shows the old news from September 13th. However, after sometime, the same Google query shows the correct news from 4th January 2017. This proves the validity of the algorithm, as shown in Figure 6.8.

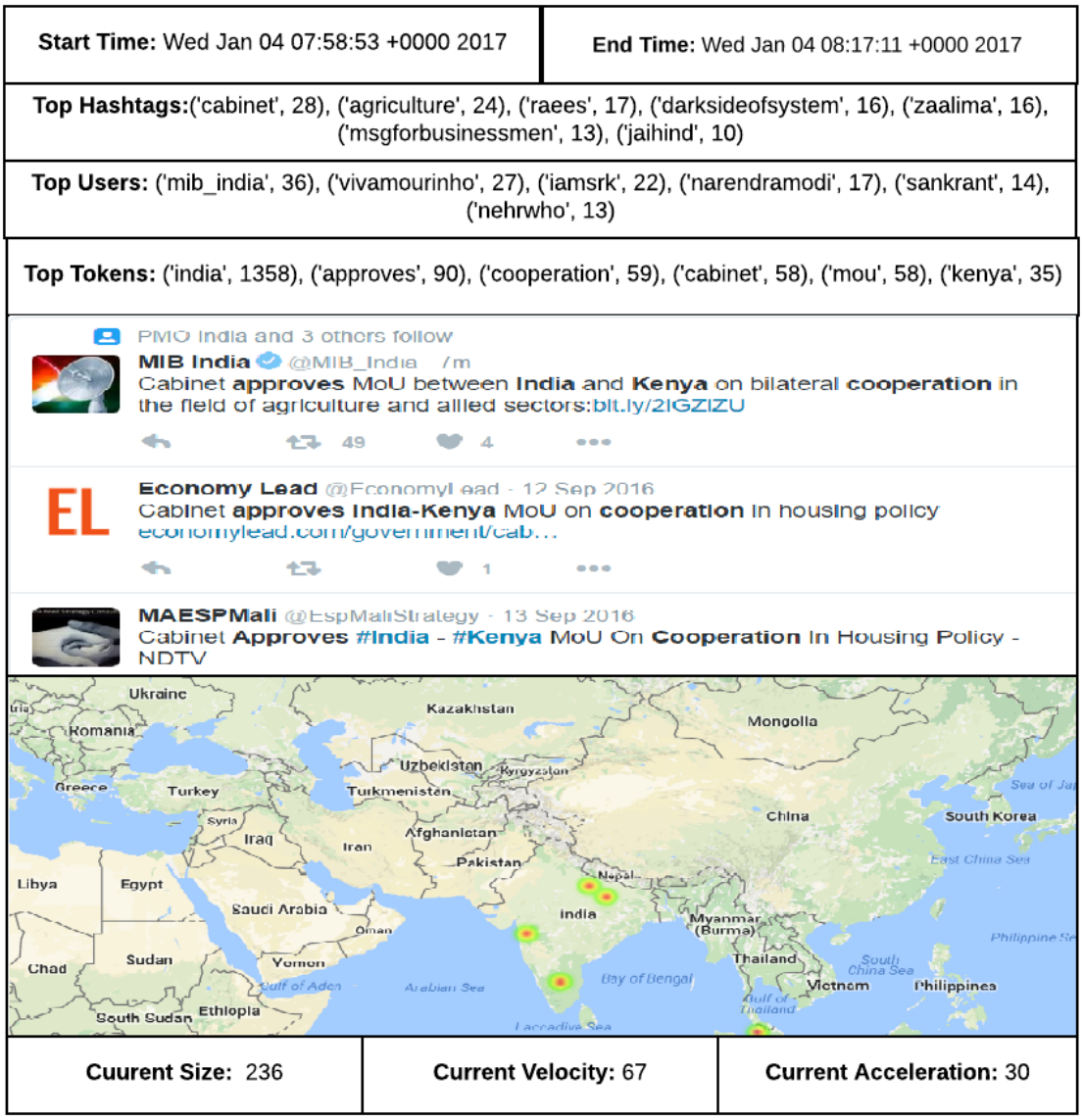


Figure 6.7: Early Event Detection

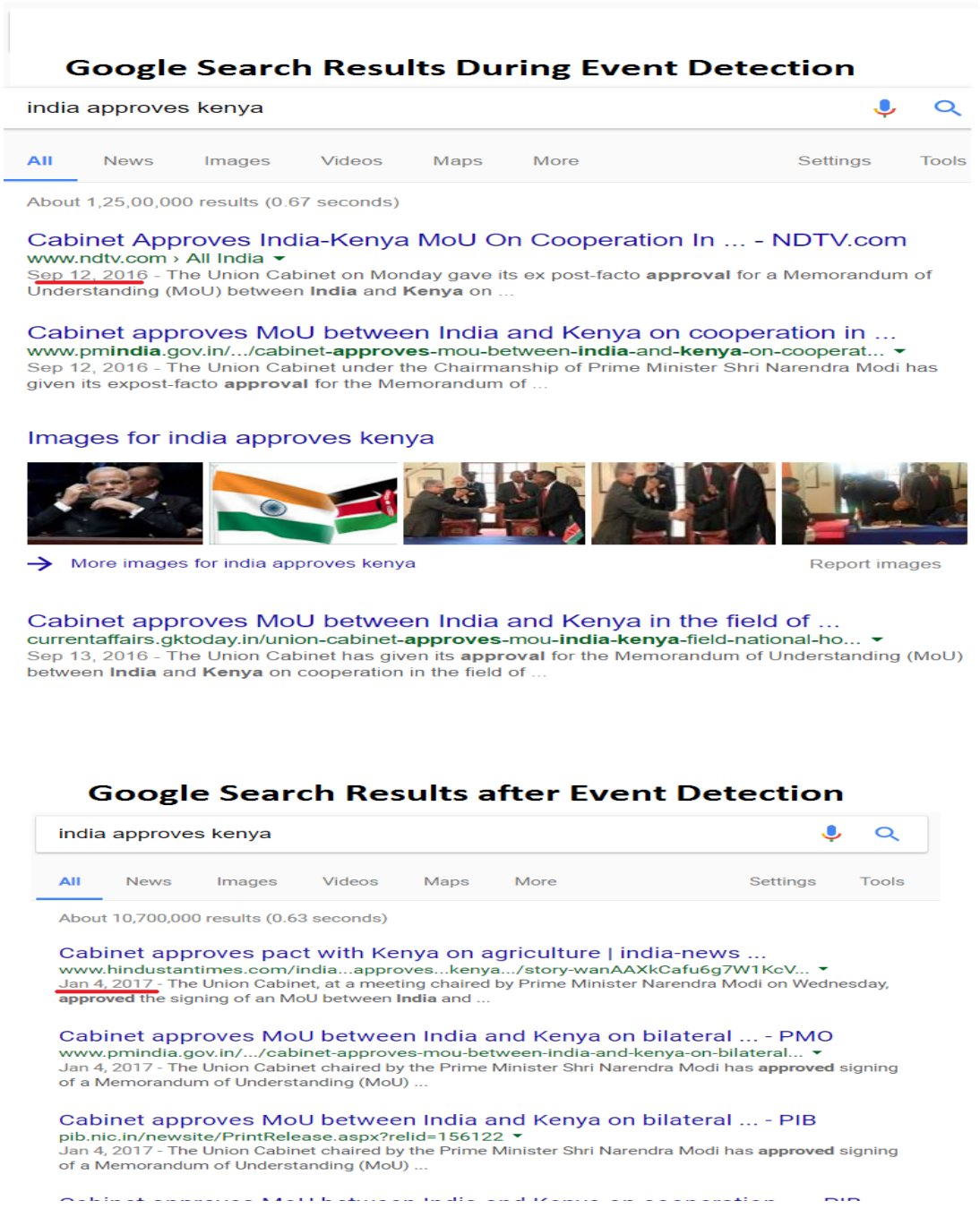


Figure 6.8: Early Event Detection

Chapter 7

Application: Hashtag recommendation and other applications

In this section we will briefly discuss about two other main applications of the technique in Twitter analytics, namely, hashtag recommendation and community detection.

7.1 Overview

It is estimated that around 500 million tweets are posted everyday on Twitter[twe], where people broadcast announcements, have online discussions and arguments, and express their emotions about real-world events. This makes it an excellent live platform to explore human online behavior at scale. However, the scale of data also presents the need to have a more principled way to organize the data, which would group tweets around themes and facilitate search operations based on similar tweets. To enable this thematic structuring of information, Twitter introduced hashtags which are user-defined strings preceded by the # symbol, to announce the topic of the tweet. In reality, however, only

around 10 percent of tweets contain hashtags [HCC11]. As Twitter does not offer any definitive guidelines for creating hashtags, a tweet creator is free to use any alphanumeric-combination as a hashtag. So, if a user chooses an unfamiliar hashtag for a tweet, it can probably lose visibility due to low usage. Further, indexing that tweet with hashtags is adversely affected.

This brings us to the problem of *real-time hashtag recommendation*, where, given a tweet text, a recommender system would suggest a few appropriate hashtags that would be consistent with the theme of that text in the current context.

7.2 Hashtag Recommender System

The figure below shows how we use the temporal clustering engine to suggest hashtags for an incoming tweet *t*

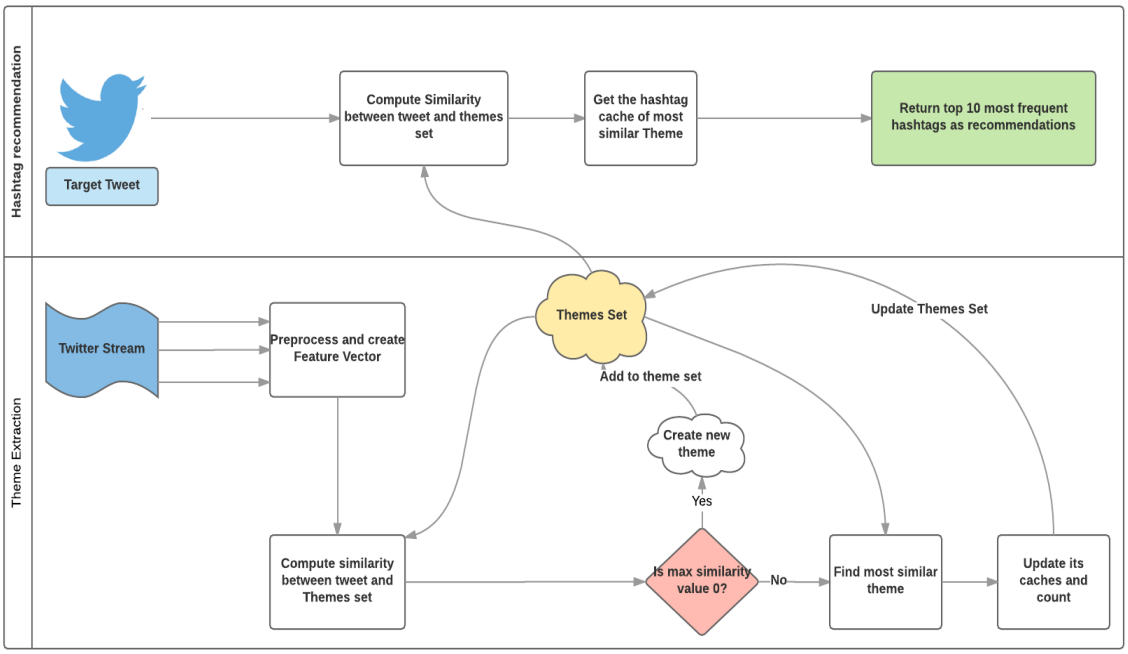


Figure 7.1: Early Event Detection

For hashtag recommendations, the theme extraction algorithm runs in background.

When a candidate tweet is given as input, it polls the modeling algorithm by calling the Construct Model method(discussed in Chapter 4) and gets the index of the most similar theme. The result is a list of top hashtags in the matched theme, sorted based on their frequencies. The frequency values give a confidence score. If the tweet does not match any theme, no recommendations are given.

7.3 Observations

7.3.1 Experimental setup

There are many global events that have been popular between September 2016-October 2017. Some of them include USA presidential elections, India-Pakistan border tensions, iphone 7 release and so on.

We use a python API called *tweepy* that connects to Twitter and gives tweets based on some filters in real-time. At startup, we let the theme extraction algorithm build the themes for sometime. After it processes more than 50,000 tweets, we poll it with our target tweets to receive hashtag recommendations. Table 7.1 shows some examples of tweets without hashtags and the system recommends a list of relevant hashtags for it. With each hashtag, we show the frequency values as well, which serves as a confidence score.

Table 7.1: Hashtag recommendation results from live data

Input tweet	**Hashtag recommendations**
does donald trump pay his taxes(t=t_1)	trump:76, nevertrump:65, trumptaxreturns:9, trumptaxes:8
lebron james never goes wrong	basementdwellers:123, hillary:52, , imwithher:43
pakistan is at unrest after india's reply	pok:21, pakistan:15, surgicalstrike:14, balochistan:11
margot robbie did an amazing job in snl	snl:89, snlpremiere:11, margotrobbie:6, theweeknd:4, alecbaldwin:4
hillary should not abuse bernie supporters	basementdwellers:517, basketofdeplorables:71, deplorables:64, neverhillary:38
pakistan terrorists attack once again	baramulla:25,baramullaattack:23, gurdaspur:20, uriattack:14
why did priest monson leave the conference	ldsconf:287, preseyring:52, presmonson:47, presuchtdorf:12, eldercook:6
does donald trump pay his taxes(t=t_1+20)	trump:361, lasttimetrumppaidtaxes:88, maga:76
apple store incident was sad	apple:173, iphone:81, flashfly:43,mac:32, iphone7:17
nobody likes hillarys rally	neverhillary:33, manheim:25, hillaryclinton:14, trump:11, clintonkaine:8

7.3.2 Discussion

The results of the recommendation system are described in Table 7.1. In each experiment, the initial model building time, i.e., the time between starting the tweet stream and the time when we start getting reasonable recommendation is between 4 and 5 minutes. Some key aspects inferred from the results are as follows.

- **Hashtags provide valuable information.** In the examples shown in Table 1, it is clearly observed that the recommended hashtags understand the context of the tweets and provide greater insight about the theme. For the tweet "Lebron James never goes wrong", the recommendations are related to Hillary Clinton, because he just endorsed her for President. Similarly, for "hillary should not abuse bernie supporters" gives us useful hashtags like basementDwellers, etc. which is the exact term she called them.

- **Hashtags evolve with time.** For the same tweet, we get different recommendations in different points in time. For example, for the same tweet "does donald trump pay his taxes", we get different different sets of recommendations at different times.

- **Same subject mapped to different hashtags.** The similarity measure evaluates every element of the feature vector before assigning it to a theme. Thus, in our examples, even though there are multiple tweets concerning Hillary Clinton, they are mapped to different hashtags because all are from different contexts.

- **Uncommon hashtags are filtered out.** One of our motivations for a hashtag recommender system was to improve the visibility and indexing of tweets by attaching them to globally recognized hashtags. Here, in the previous example tweet about trump's taxes, we observe that uncommon hashtags like #trumptaxes and #trumptaxreturns are replaced by lasttimetrumppaidtaxes, which was actually a trending hashtag on Twitter.

7.4 Community Detection

Community Detection is a key problem in social networks analysis. As illustrated above, a themes captures similar tweets talking about a common subject as separate lists of hashtags, users, etc. A user list in a theme can be seen as a community of users, who are discussants of a common topic of interest. Also, the heatmaps help us visualize the different areas on Earth, where the tweets about a certain theme are originating from. This gives a complete picture of specific users discussing common themes and their geographical locations. The Figure7.2 below shows one such instance.

The figure shows an event captured which talks about sports event played in Michigan. Notice the top users are primary sports channels, which are discussing the sports event aggressively, and hence, get featured in the list. Moreover, the heat map depicts the expected accurate scenario as it shows a lot of traffic coming from Michigan, and particularly, Ann Arbor area, where the event was held.

Thus our methodology gives a wholesome community with primary discussants and their locations as opposed to just the trending topics on twitter, which only showed “Goblue” as a trending hashtag for the event.

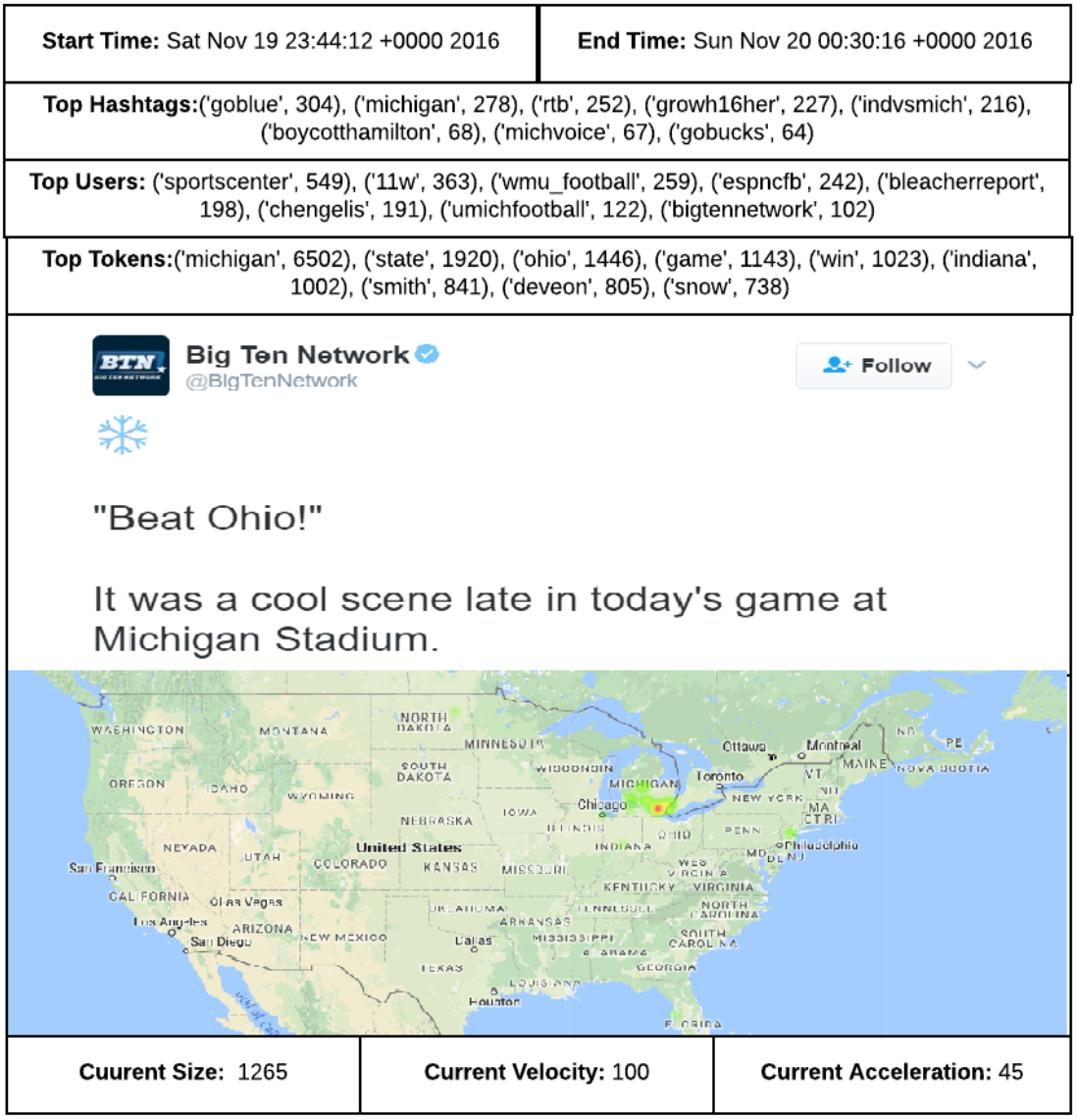


Figure 7.2: Community detection

Chapter 8

Application: User Behavior Analysis

8.1 Overview

Recently, some approaches using data mining algorithms are applied to log analysis in the intrusion detection community. Those algorithms are based on supervised learning. That is to say, they are trained, instead of programmed, on data sets with labels indicating whether the instances are pre-classified as attacks or not. However, manually labeling the large volumes of network data (It is not unusual to see a network log data set larger than 1 gigabytes) is difficult and extremely expensive, due to prohibitive human labor costs. This renders supervised learning hard to apply or its performance badly impaired due to the lack of well labeled training data.

Most error log analysis studies perform a statistical fit to the data assuming a single underlying error process.

8.2 Type of data

Due to the heavy use of software versioning systems, databases, and employee attendance systems like Workday, companies have good sources of logs to determine user behavior, popular trends in the company and possible anomalies or insider threats. The following figure shows a sample user database access log generated by after a database event is performed by an employee. It can be seen in Figure 8.1 that by aggregating information from different sources, a user-event log can tell us valuable information about that activity performed by the user. For example, this log shows the time and source IP of the user when the event occurred, the query executed, number of sensitive events generated, number of rows affected by the query, Domains accessed, etc.

```
In [16]: support_logs.take(1)

Out[16]: [{"EventCollector": "Imperva Inc.",
  "EventDescription": {'BindVariables': '',
   'DataStore': 'COE_PCRS_951/SDBU_ORA3',
   'EventAffectedCount': '52',
   'EventApplicationName': 'Default Oracle Application',
   'IsSensitive': True,
   'Query': 'SELECT "ORDER_NO", "FIRST_NAME", "SSN" FROM "SDBU"."CUSTOMER_ORDERS", "CI_STG1"."PATIENTS", "CI_STG5"."USERS"',
   'SensitiveDataDomains': [{'columns': [{'columnName': 'ORDER_NO',
       'schemaName': 'SDBU',
       'tableName': 'CUSTOMER_ORDERS'}],
     'dataDomainName': 'IBAN'}],
   'UserGroup': 'Default oracle group'},
  "EventMessageType": 'CEF',
  "EventUid": '-2144764649',
  "RawEventMessage": '<13>Apr 23 10:06:18 quickstart cloudera: CEF: 0|Imperva Inc.|SecureSphere|10.5.0.4_0|Audit|Audit.DAM|Info
rmative|dst=10.17.40.25 dpt=1521 duser=Edwards,  Melissa F. src=10.17.0.102 spt=43570 proto=TCP rt=01 June 2015 08:20:01, cat=A
udit Infa DB Audit cs1Label=Policy cs2=Database Server Group cs2Label=ServerGroup cs3=Oracle Services cs3Label=ServiceName cs4=D
efault Oracle Application cs4Label=ApplicationName cs5=1086628793948 cs5Label=EventId cs6=Query cs6Label=EventType cs7=Default o
racle group cs7Label=UserGroup cs8=True cs8Label=UserAuthenticated cs9= cs9Label=ApplicationUser cs10=jdbc thin client cs10Labe
l=SourceApplication cs11=oracle cs11Label=OSUser cs12=pbox-montreal cs12Label=HostName cs13=COE_PCRS_951/SDBU_ORA3 cs13Label=Da
tabase cs14=hr cs14Label=Schema cs15= cs15Label=RawQuery cs16=SELECT "ORDER_NO", "FIRST_NAME", "SSN" FROM "SDBU"."CUSTOMER_ORDE
RS", "CI_STG1"."PATIENTS", "CI_STG5"."USERS" cs16Label=ParsedQuery cs17= cs17Label=BindVariables cs18= cs18Label=SQLError cs19=
0 cs19Label=ResponseSize cs20=57 cs20Label=ResponseTime cs21=52 cs21Label=AffectedRows\n',
  "SourceEmail": '',
  "SourceHostname": '',
  "SourceIP": '10.77.1.161',
  "TargetEmail": '',
  "TargetHostname": '',
  "TargetIP": '10.17.40.25',
  "TimeOfEvent": '2015-06-01-08-20-01',
  "TimeOfEventUTC": '2015-06-01-02-50-01',
  "User": {"Department": 'Support',
   'Location': 'Hyderab',
   'User': 'Edwards,  Melissa F.',
   'UserGroup': 'Operation'}}]
```

Figure 8.1: A sample user-activity log

8.3 Problem Description

Problem statement: *Given a corpora of user-activity logs c, updated after fixed intervals of time T, construct themes t_i which serve as the latent representation of types of different tasks performed by the users, and monitor how they evolve with time.*

As mentioned above, the basic task is to perform unsupervised learning to extract meaningful information from unstructured logs, which are generated in large quantities by applications, and which are key reflectors of user behavior in those applications. In later subsections, we describe how this meaningful information can be used to perform key knowledge discovery tasks which may prove beneficial in designing strategies for business development.

Some of the applications discussed in the later subsections include near-real time monitoring of user activity, broken log user recommendations, domain discovery(in the case of database access logs) etc. Applications of this methodology are dependent on the content of logs and the design of the feature vector. Each dimension in the feature vector can potentially have an application centered around it.

8.4 Our Method

The underlying algorithm described in Chapter 4 can be tweaked to discover patterns in logs as well. Prior to this section, our discussion was mostly centered around tweets, and our methodology is shown to give good results for them. Logs, which are json objects, similar to tweets, share the same structure. However, there are many key challenges in generic log analysis, as compared to tweets-

- User logs are stored in batches of fixed time intervals. A streamed analysis on one user log at a time can be very slow, computation intensive and require too much memory. In many of such applications, some latency in discovery in acceptable,

as we need not know the exact second when an anomaly occurred. Even knowing hourly/ daily statistics is acceptable.

- The feature vector should be much more detailed as compared to the social network task. As mentioned above, user logs give us information like the time and source location/ IP of the user generating the event, domains accessed, rows affected, etc. Such information should be a part of the feature vector as it provides key insights in user behavior.

- The similarity functions mentioned in Chapter 4 needs to be changed depending on the new feature vector.

The big data problem caused us to change the basic design which we had for Twitter. The subsequent sections discuss the changes made in the feature vector and the algorithm for this use-case.

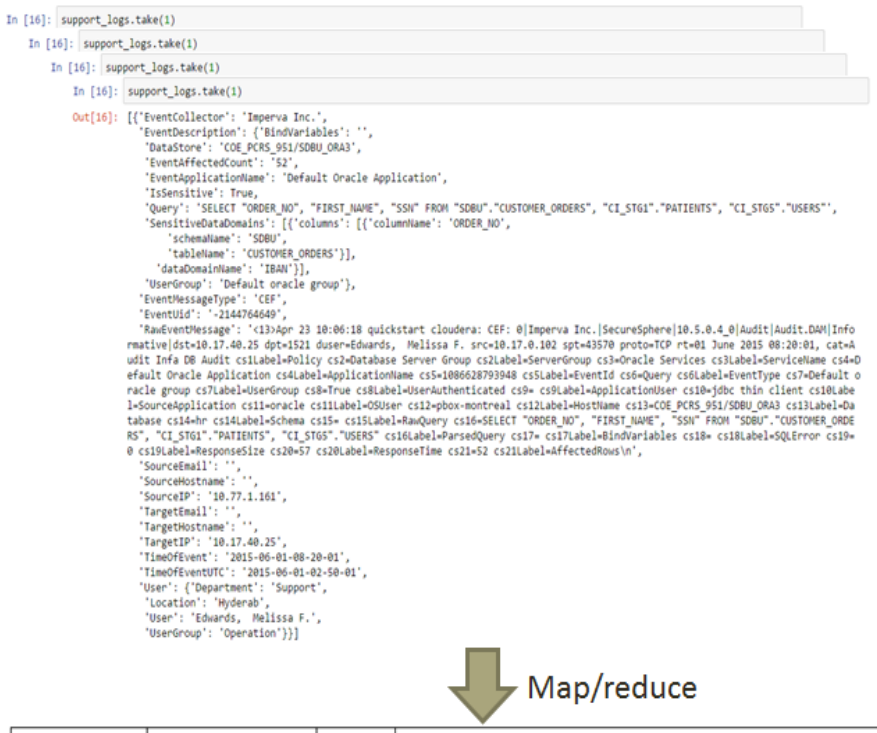
8.4.1 Feature Vector Design

As stated above, running an in-memory algorithm for logs, which range in terabytes in size everyday, even for a mid size organization is infeasible. Running a stream-based algorithm on one log at a time could also be slow, computationally intensive and often undesirable. Thus, we define our feature vector as-

Feature Vector: *Feature vector for log analysis, or a user profile, is a summary of all the events generated by that user in the fixed time interval, as per by batching scheme. In other words, its a componentwise aggregation of all the logs generated by the user in a batched file, which contains all the logs in a fixed time interval.*

The features we use for this data are- source IP of the user, domains accessed, time slot in a day of access, rows affected, and sensitive events generated. The data used is fabricated database access logs generated for an organization, with injected anomalies

to calculate the performance of their product. For aggregation of user logs into a user profile, we use Apache Spark [spa] to do a map reduce operation to map all the logs for one user into a single vector in that time slot. This causes a drastic reduction in the size of the data as we migrate from a size of O(no. of logs) to O(no. of users), which is much more manageable. Figure 8.2 shows the feature vector creation step.



UserName	Source Ip		Time Slots		Domains				Effectted Rows	Sensitive Events
Name1	10.77.6.51	10.77.6.11	3	0	Passport_India	Postcode	Passport_DEU_MR	DriversLicence_GBR		
Flores, Octavius P.	8	40	45	3	4	2	7	5	2000	56

Figure 8.2: A sample feature vector creation process

As seen in Figure 8.2, the map reduce step creates a heterogeneous vector which contains categorical components, like Domains, IPs, time slots, etc. and also continuous components, like affected rows, number of sensitive events generated, etc.

8.4.2 Applying Temporal Clustering

There are two alterations to the theme detection algorithm that need to be introduced to accomodate the changes introduced above.

- Due to heterogeneity of the feature vector, similarity function needs to be changed to handle categorical and continuous variables differently.

- Earlier, a new theme was created when an incoming data point showed zero similarity with the current themes. However, in this case, as most of the employees work in similar time slots in a day(a day is divided into 4 time slots of 6 hours each), and continuous components will have finite distances, achieving a zero similarity is hard. Hence, we need to define a splitting criteria, which if satisfied, leads to the creation of a new theme.

Splitting criteria: For this use-case, we consider if users differ substantially in their domain access behavior then they are probably showing different user behavior, and hence should belong to different theme. This is purely a design decision, and can vary based on domain knowledge. In our experiments, we decide to create a new theme(split) if the incoming user profile has less than α common domains with every other theme currently in the system.

The similarity function in Chapter 4 was defined as-

$$S(t, T_i) = \lambda_1 H + \lambda_2 U + \lambda_3 W + c \tag{8.1}$$

For log analysis, the recency terms, i.e. λ_i remain the same for each categorical component. the H, U and W terms are replaced by Domain overlap, time slot overlap and Ip overlap respectively. Similar update procedure and similarity comparison is used for these as for hashtags, users and tokens, as both contain lists of key-value pairs.

Whenever a user profile is matched to a theme and added to it, the categorical components are updates as before, in Twitter use-case. However, for continuous variable, they are just appended to their corresponding lists, which contains the values of that component for the last n assignemnts. When an incoming user profile needs to be compared with Theme T_j, we consider this list as a gaussian distribution and calculate the probability value of this gaussian at the corresponding continuous component. The total similarity is just the sum of each of these individual similarities.

$$S(u,T_j) = \lambda_1 IP + \lambda_2 Time + \lambda_3 Dom + P_1 + P_2 \tag{8.2}$$

Here, P_1 refers to $P(u_r \in T_j(Affectedrows))$ where u_r is the affected rows value in the incoming user profile and $T_j(AffectedRows)$ is the gaussian from the list of last n assignments of Affected rows component. $P2$ can be found similarly. A sample theme generated from this clustering procedure is shown in Figure 8.3

Theme No. 1	
Current Size	230
Current Velocity	25
Top Users	'Gupta , Rahul', 'Kumaresan, Bala', 'Gilliam, Lewis K.', 'Becker, Rae X.', 'Albert, Arden D.'
Most Frequent Time Slot of Access	('1', 623), ('0', 311)
Most Frequent Source Locations/IPs	('10.77.6.51', 85), ('10.77.6.11', 54), ('10.77.5.104', 39), ('10.77.5.246', 31)
Most Frequent Domains accessed	('PhoneNumber', 34), ('Passport_MachReadable', 26), ('Passport_India', 15), ('Postcode', 15)
Average Rows Accessed	456, 56, 342, 353, 322, 545, 223, 12
Average sensitive events generated	12, 34, 11, 1, 56, 78, 12, 9

Figure 8.3: A sample theme extracted from the clustering process

8.5 Applications

Let us discuss some of the possible applications of this approach.

8.5.1 Activity Monitoring and anomaly detection

As the themes evolve with time, they give an up-to-date description about the different types of user behavior shown by the users of the application. Some of the key insights include- what are the source IPs/locations users are generally accessing it from, at what time of the day are they most active, what are the most prominent users, what are the domains they are accessing the most, etc.

Such a near real-time monitoring tool can be used to detect anomalies as well. Consider a theme being formed which has been showing very low velocity and acceleration consistently for a very long time. This would mean that there was an activity, which was different from all the themes at that moment, which lead to the creation of that theme, but after that it showed negligible assignments. This could possibly be a rare event, and can be flagged as an anomaly.

8.5.2 Broken log Recommender System

In some cases, as the user log is aggregated from different sources, due to error in some pipeline (like authentication servers, etc.) the username information in the log might be missing while other activities might be retained. If in such a case we want recommendations for the usernames for that broken log, we can find its best fit to the current themes. As these themes are in sync with the current user behavior patterns, and they store the top users showing that particular behavior at that point in time, the top users in the matched theme can be shown as recommendations to fix the broken log. Even if another component is missing, as the similarity function is a normalized sum of

individual similarities, it can still calculate the value by relying on other components, thus producing recommendations.

8.5.3 Discovering Domain clusters

In many applications rules and policies are applied to different database domains. For larger applications, which have huge databases with many domains, similar ones are clustered together and common rules are applied to the cluster, instead of defining rules for individual tables. The most common way of finding similarity between database tables is to compare their schema metadata and compute scores. However, in many cases the meta data information is not available. This is specially true for new domains that get added to the system. In such a scenario finding clusters of domains with conventional methods is not possible. However, if we consider the top trending themes, their domains are actually the ones accessed by users exhibiting a common behavior. This collaborative filtering type approach can give us valuable recommendations for domain groupings when their metadata is unavailable.

Chapter 9

Results and Evaluation

In this section, we design multiple experiments and metrics to evaluate two main aspects of our approach:

- *Recommendation quality*: We test the quality of recommendations produced by the algorithm by comparing them against real-world evidence of the popularity of suggested hashtags, and also against some popular metrics. One such metric is Hashtag Precision which is given by-

$$\text{Hashtag Precision} = \frac{\#\ correct\ guesses}{\#\ tweets\ with\ hashtags}$$

  We collect tweets which contain hashtags and then pass it to the algorithm with their hashtags removed. The precision is calculated a ratio of the number of correct guesses to the total number of input tweets.

- *Theme quality*: We measure the quality of underlying themes that are generated from the model. We compare our results against a dynamic topic model with logistic normal priors and inference based on collapsed Gibbs sampling using the same dataset. We also prove the stability of the themes formed by splitting the dataset into two and running

the algorithm independently on each.

The details of the dataset used for these experiments are summarized in the Table 9.1. For privacy reasons, we cannot share the actual data used for computations. The algorithms and implementation is available online[1]. The remaining section describes the experiments and evaluation methods.

Table 9.1: Input Data Description

Total No. of tweets	892,050
Total number of tweets with hashtags	297,805
Total number of users	33,298
Total number of distinct hashtags	23,163

9.1 Evidence from the Internet

We collected tweets in real-time on October 2, 2016 and got recommendations as shown in Table 7.1. Some of the hashtags recommended by the system were actually trending on Twitter. Hashtags like basementDwellers and lastTimeTrumpPaidTaxes were featured in various news articles like Truthfeed[tru] and BBC UK[bbc] respectively. This shows that the algorithm tries to recommend hashtags that are popular globally.

9.2 Cluster Stability

To measure stability of clusters, we split the dataset described in Table 9.1 into two files, such that, each tweet in the original file has equal probability of going to one of the two files. We run the theme extraction algorithm independently on both of them to get themes. The results are summarized in Table 9.2.

[1] https://github.com/nishucsd/thesis/

The table shows themes generated from the two datasets. It is evident from the results that they are fairly similar. The top 3 themes in both cases are about US Presidential candidates Donald Trump, Hillary Clinton and Bernie Sanders. For each theme we show the top hashtags, users and tokens in the result.

Table 9.2: Cluster Stability results

	Set1	**Set2**
Theme 1	maga, nevertrump, trump2016	nevertrump, makedonald-drumfagain, alwaystrump
	realdonaldtrump, michellevii, youtube, marcorubio	realdonaldtrump, michellevii, marcorubio, thehill
	trump, donald, rally, john, vote, rubio	trump, donald, rubio, rally, kkk, support
Theme 2	supermonday, imwithher, hillary, feelthebern	imwithher, hillaryclinton, su-pertuesday, hillary2016
	hillaryclinton, berniesanders, people4bernie, thehill	hillaryclinton, the_rant_, tom-mychong, joenbc
	hillary, clinton, vote, win, south, carolina	hillary, clinton, south, win, carolina
Theme 3	feelthebern, berniesanders, su-permonday, whitepoeple	feelthebern, berniesanders, hi-lary, whitepeople
	killermike, newswirenet, youtube, berniesanders	1d33p, berniesanders, newswirenet, killermike
	bernie, sanders, vote, support, clinton	bernie, sanders, vote, clinton, support

9.3 Comparison with a Topic Model

We compared the performance of our algorithm with the well-known Dynamic Topic Model (DTM) developed by Blei [BL06] to capture topic evolution over discrete time stamps. In this model, at each timestamp, tweets are combined into multiple documents such that each document has a meaningful structure, with majority of tweets about similar topics. A logistic normal parameter β is used to express distribution over

topic proportions. The βs are sampled from normal distribution with mean α and variance Σ. The α parameters at any timestamp are obtained from the previous timestamp using Kalman filtering. Each document d is assigned a document-specific topic proportion θ which is calculated from β using following mapping:

$$\theta_t = \frac{\exp(\beta_t)}{\sum_{t'=1}^{T} \exp(\beta_{t'})} = \frac{\exp(\beta_t)}{1 + \sum_{t'=1}^{T-1} \exp(\beta_{t'})}$$

Every word in the document is drawn conditionally independently from a discrete(categorical) distribution with parameter vector $\theta^{(z(w))}$. The word distributions $\phi(c)$ have a symmetric Dirichlet prior, with each entry of the V-dimensional parameter vector being equal to η, denoted as Dirichlet(η). A logistic normal prior is used for parameter β to sequentially tie the tweet documents over multiple time stamps. Here is the generative process of each time slice t :

1. Draw $\alpha_t | \alpha_{t-1} \sim \mathcal{N}(\alpha_{t-1}, \sigma^2 I)$
2. Draw $\phi \sim Dirichlet(\eta)$.
3. For each document d:
   (a) Draw $\beta_d \sim \mathcal{N}(\alpha_t, \Sigma)$
   $\theta_d = \pi(\beta_d)$
   (b) For each word:
      i. Draw $z_{dn} \sim Mult(\pi(\theta_d))$.
      ii. Draw $w_{dn} \sim Mult(\phi^{z_{dn}})$.

The graphical model for this generative process is shown in Figure 9.1. For inference, we used Collapsed Gibbs Sampling as described by Mimno in [MWM08].

We divided the Twitter data into 50 time slices and ran the dynamic topic model on it. Number of topics were set to 15.

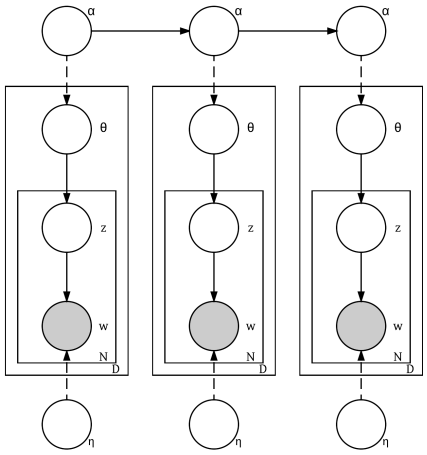


Figure 9.1: Dynamic Topic Model

Table 9.3: Comparison between our approach and DTM

	Our approach	**DTM**
Topic 1	trump, donald, vote, rally, support	cruztovictory, sick, heat, bernie, rubio
Topic 2	vote, bernie, sanders, today	hillary, idea, today, right, unelect
Topic 3	hillary, clinton, black, world, email, peace	people, union, nominee, hope, march
Topic 4	cruz, ted, look, christie, party, endorse	evangel, hillary, pro, trust, cannot

Table 9.3 shows results as top 4 topics extracted from both the approaches. We only show the top tokens retrieved from our algorithm as results, while the results of the dynamic topic model are the top 5 keywords extracted from the model after 50 time slices.

It is clear from the results that our model outperforms the DTM. As per the results from our approach, it is evident that each theme is about the 4 main candidates in US

Elections, namely, Trump, Hillary Clinton, Bernie Sanders and Ted Cruz. However, the results from DTM give incoherent topics and it is hard to infer what they relate to. There can be multiple reasons for the above observations. Firstly, DTM was not designed for short text, while our algorithm has been designed for micro-blogs specifically. Secondly, there are many hyper-parameters like no. of topics, etc that are required to be set a priori in DTM. Choosing wrong hyper-parameters can lead to erroneous results. However, as our model is data-driven, there are no hyper-parameters to be set. New themes are created as they emerge, thus creating coherent clusters.

9.4 Hashtag Precision

To evaluate the validity of recommendations produced by the algorithm, we calculate the Precision for our approach and compare it with two other baselines-

Naive Recency Based Approach

In this experiment we design a naive baseline hashtag recommender system to compare against our approach. In this experiment we assume that a user uses similar hashtags in a small time interval. Therefore, for a target tweet t by a user u, we recommend the hashtags he used in his last tweet. This approach does not consider the context of the tweet and is a purely recency based measure.

K-Nearest Neighbor based approach

In this experiment, we design a K-Nearest Neighbor based hashtag recommender. It uses Tanimoto Distance measure(number of common terms between two tweets) to compute K-nearest tweets in a sliding window and recommends its hashtags as output. This kind of an approach is purely context based and it does not consider the relative

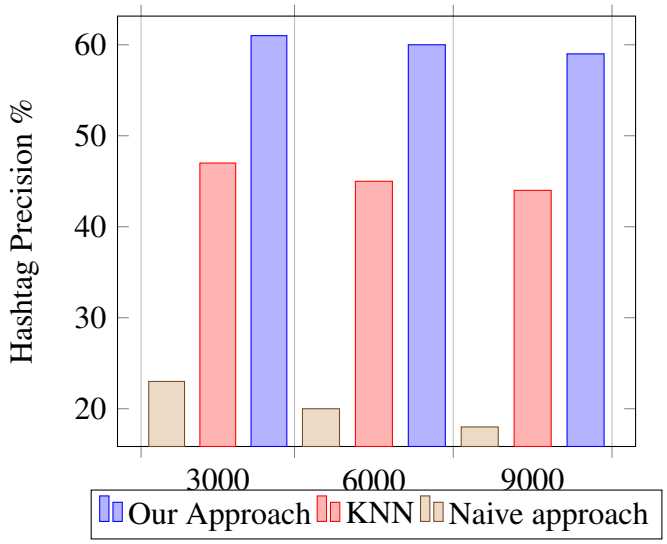


Figure 9.2: Plot showing Hashtag precision for 3000, 6000 and 9000 tweets

ordering of tweets within the sliding window.

These experiments thoroughly evaluate the similarity function. Precision is calculated for these two approaches against our algorithm and the results are shown in Figure 9.2. Here we use a training set of 100,000 tweets and vary the test set as per the figure.

The results show that our algorithm outperforms a naive recency based approach and K-nearest neighbor based approach. One of the reasons could be that our similarity measure incorporates both, the recency and context, to decide the candidate hashtags, while other approaches concentrate only on one of these two factors. Also, as shown above, the c term in the measure helps to fine-tune the results. In terms of performance, our LRU based scheme computes themes in background. When a test tweet arrives, it is just compared against current themes, which is far less than individual tweets. In the sliding window KNN approach, all computations of finding distances between individual tweets in the sliding window, retrieving the K-nearest ones, etc. are done at the time when a test tweet arrives. This is far more computationally intensive, and hence slower than the LRU-based scheme.

Chapter 10

Advanced Analytical Operations

At regular intervals, the themes set is polled to know the current state of the system. This current state is persisted in a JSON tree structure to allow queries to be performed on it. Figure 10.1 shows the structure for 2 time stamps with 2 themes per time stamp. As it is clear from the diagram, if the timestamp is known, we can go down the hierarchy to get useful information like - top hashtags at that time, popular discussants of the theme, important terms, links to articles shared within tweets in the theme, geocodes etc. about the particular theme. As we also monitor the size, velocity and acceleration information alongwith other attributes, we can answer queries like the top rising themes, top discussed themes,etc at any point in time.

10.1 Advanced queries

As mentioned above, answering queries pertaining to a single timestamp is easy, as we store all the information as JSON objects, each comprising of key value pairs. To compare the states of the system between timestamps, and to perform other complex operations, we need special operators which can do this. A couple of them are described below:

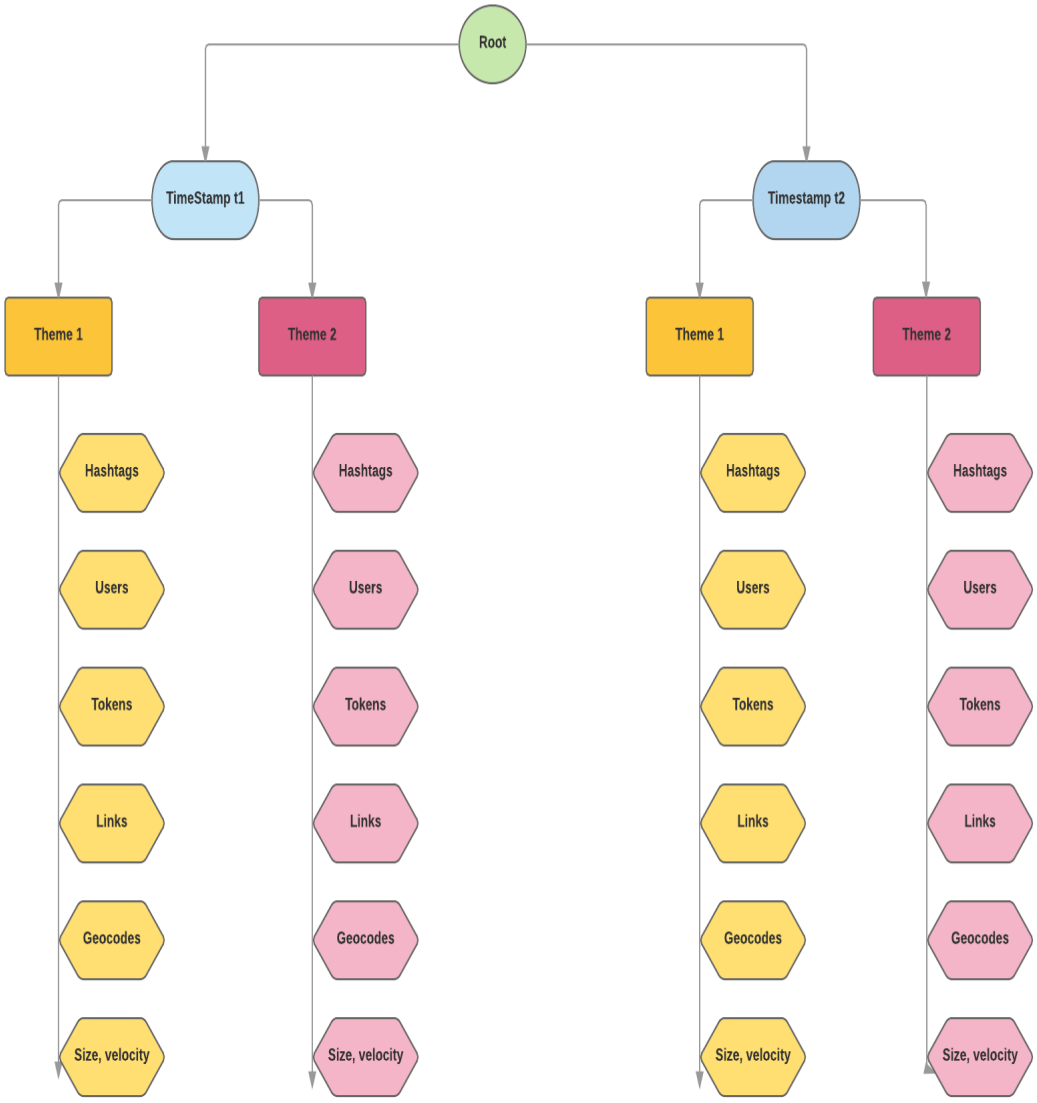


Figure 10.1: The data structure to store themes over time.

- **Emerging_Hashtags(Timestamp, theme no.):** A major class of evolutionary queries are concerned with analyzing emergent behavior of patterns as they evolve through time. For this reason, an operator like emerging hashtags calculates the difference between the hashtag lists of a particular theme, between current and the previous timestamp. Generally, the state of the theme detection algorithm is polled after every 10,000 tweets. Each of these time stamps are stored in a separate

linked list, so that knowing the previous timestamps from the current one is an O(1) operation. The result is then shown as a tag cloud. Figure 10.2 shows one such output of this operation on an event related to Mr. Trump.

Figure 10.2: Tag cloud of emerging hashtags for a Theme related to Mr. Trump

- **Merge(Num. of Themes):** As discussed before, there are two ways to reduce the total number of themes currently in the system. First way is to remove the slowest growing themes at regular intervals. The other operation is a merge which combines similar themes using a K-Means operation, into a fixed number of themes. As Themes are a collection of ranked lists of key-value pairs, we use *Spearman's footrule* as the distance metric, which is defined as follows-

  Spearman's footrule: Consider 2 ranked lists L_1 and L_2. Let $idx_1(i)$ and $idx_2(i)$ be the positions of an element i in lists L_1 and L_2 respectively. The displacement of an element i is defined as $\sigma_i = |idx_1(i) - idx_2(i)|$. Spearmans's footrule is therefore defined as

$$F(\sigma) = \sum_i \sigma_i \tag{10.1}$$

Chapter 11

Conclusion

In this thesis, we propose a novel approach to discover themes in streaming data in a real-time setting. We also show the theoretical motivation and explanation of design decisions by comparing its similarity to point processes. The unique aspect of this approach is that it lets the user perform relatively complex and independent tasks of early event detection, community detection, etc. simultaneously, alongwith providing advanced query capabilities on the themes. As themes evolve over time, we discuss data management options of storing these snapshots, monitoring their growth, and limiting the size of the collected data so that its valid in an online setting.

Future work involves designing a user interface that efficiently portrays these theme discoveries in a natural way. We also plan to refine the similarity function by experimenting on different choices of temporal and contextual proximity parameters. Other steps include experimenting with better cache designs that account for both, recency and frequency.Stronger baselines and better metrics need to be established for comparison with the approach. Finding clusters within themes based on the sentiments of tweets is another area we plan to look into.

Bibliography

[AMRW12] Foteini Alvanaki, Sebastian Michel, Krithi Ramamritham, and Gerhard Weikum. See what's enblogue: real-time emergent topic identification in social media. In *15th International Conference on Extending Database Technology, EDBT '12, Berlin, Germany, March 27-30, 2012, Proceedings*, pages 336–347, 2012.

[AMS16] David Alvarez-Melis and Martin Saveski. Topic modeling in twitter: Aggregating tweets by conversations. In *10th Int. AAAI Conf. on Web and Social Media*, 2016.

[APL98a] James Allan, Ron Papka, and Victor Lavrenko. On-line new event detection and tracking. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pages 37–45, 1998.

[APL98b] James Allan, Ron Papka, and Victor Lavrenko. On-line new event detection and tracking. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pages 37–45, 1998.

[bbc] BBC UK article on trending hashtags http://www.bbc.co.uk/newsbeat/article/37533666/la-sttimetrumppaidtaxes-trending-after-us-newspaper-reveals-trump-documents.

[BL06] David M Blei and John D Lafferty. Dynamic topic models. In *Proc. of the 23rd Int. Conf. on Machine Learning*, pages 113–120. ACM, 2006.

[BNJ03] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[CCMS10] Comas, Jorge Carles Mateu, and Aila Srkk. A third order point process characteristic for multi-type point processes. *Statistica neerlandica*, 64:19–44, 2010.

[CCS13] Mario Cataldi, Luigi Di Caro, and Claudio Schifanella. Personalized emerging topic detection based on a term aging model. *ACM TIST*, 5(1):7:1–7:27, 2013.

[DFA+15] Nan Du, Mehrdad Farajtabar, Amr Ahmed, Alexander J. Smola, and Le Song. Dirichlet-hawkes processes with applications to clustering continuous-time document streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 219–228, 2015.

[FZZ+15] Wei Feng, Chao Zhang, Wei Zhang, Jiawei Han, Jianyong Wang, Charu Aggarwal, and Jianbin Huang. STREAMCUBE: hierarchical spatio-temporal hashtag clustering for event exploration over the twitter stream. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1561–1572, 2015.

[HAG+12] Liangjie Hong, Amr Ahmed, Siva Gurumurthy, Alexander J. Smola, and Kostas Tsioutsiouliklis. Discovering geographical topics in the twitter stream. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 769–778, 2012.

[HCC11] Lichan Hong, Gregorio Convertino, and Ed H. Chi. Language matters in twitter: A large scale study. In *Proc. of the 5th Int. Conf. on Weblogs and Social Media, Barcelona, Catalonia*, July 2011.

[HJ10] Dan He and Douglas Stott Parker Jr. Topic dynamics: an alternative model of bursts in streams of topics. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 443–452, 2010.

[KTU+13] Daichi Koike, Yusuke Takahashi, Takehito Utsuro, Masaharu Yoshioka, and Noriko Kando. Time series topic modeling and bursty topic detection of correlated news and twitter. In *Proc. of Int. Joint Conf. on NLP*, pages 917–921, 2013.

[LDD+14] Liangda Li, Hongbo Deng, Anlei Dong, Yi Chang, and Hongyuan Zha. Identifying and labeling search tasks via query-based hawkes processes. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 731–740, 2014.

[lrua] LRU api used in implementation https://github.com/amitdev/lru-dict.

[lrub] LRU definition https://en.wikipedia.org/wiki/cache_replacement_pol-icies.

[MWM08] David Mimno, Hanna M Wallach, and Andrew McCallum. Gibbs sampling for logistic normal topic models with graph-based priors. In *Proc. of NIPS Workshop on Analyzing Graphs: Theory and Applications*, 2008.

[RXS+16] Marian-Andrei Rizoiu, Lexing Xie, Scott Sanner, Manuel Cebrián, Honglin Yu, and Pascal Van Hentenryck. Can this video be promoted? - endogenous and exogenous popularity processes in social media. *CoRR*, abs/1602.06033, 2016.

[spa] Apache spark http://spark.apache.org/.

[SWK14] Erich Schubert, Michael Weiler, and Hans-Peter Kriegel. Signitrend: scalable detection of emerging topics in textual streams by hashed significance thresholds. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 871–880, 2014.

[tim] ftp://ftp.math.ucla.edu/pub/camreport/cam14-76.pdf.

[tru] Truthfeed artice on trending hashtags http://truthfeed.com/hillary-gets-destroyed-on-social-media-in-response-to-attacking-bernie-supporters-with-trending-hashtag-basementdwellers/26858/.

[twe] Twitter usage statistics from http://www.internetlivestats.com/twitter-statistics/.

[WZHS07] Xuanhui Wang, ChengXiang Zhai, Xiao Hu, and Richard Sproat. Mining correlated bursty topic patterns from coordinated text streams. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Jose, California, USA, August 12-15, 2007*, pages 784–793, 2007.

[XZJ+16] Wei Xie, Feida Zhu, Jing Jiang, Ee-Peng Lim, and Ke Wang. Topicsketch: Real-time bursty topic detection from twitter. *IEEE Trans. Knowl. Data Eng.*, 28(8):2216–2229, 2016.

[YCL+13] Hongzhi Yin, Bin Cui, Hua Lu, Yuxin Huang, and Junjie Yao. A unified model for stable and temporal topic detection from social media data. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, pages 661–672, 2013.

[YCM+13] Quan Yuan, Gao Cong, Zongyang Ma, Aixin Sun, and Nadia Magnenat-Thalmann. Who, where, when and what: discover spatio-temporal topics for twitter users. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, pages 605–613, 2013.

[YPC98] Yiming Yang, Thomas Pierce, and Jaime G. Carbonell. A study of retrospective and on-line event detection. In *SIGIR '98: Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pages 28–36, 1998.

[ZJW+11] Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee-Peng Lim, Hongfei Yan, and Xiaoming Li. Comparing twitter and traditional media using topic models. In *European Conf. on Information Retrieval (ECIR)*, pages 338–349. Springer, 2011.

[ZS14] Amir Hassan Zadeh and Ramesh Sharda. Modeling brand post popularity dynamics in online social networks. *Decision Support Systems*, 65:59–68, 2014.

[ZZS13] Ke Zhou, Hongyuan Zha, and Le Song. Learning social infectivity in sparse low-rank networks using multi-dimensional hawkes processes. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2013, Scottsdale, AZ, USA, April 29 - May 1, 2013*, pages 641–649, 2013.