# A Report on

**CLUSTER-DISCOVERY OF TWITTER MESSAGES FOR EVENT DETECTION AND TRENDING**

*In partial fulfillment of the course*

**CS F469 - Information Retrieval**



**SUBMITTED BY :**

|  |  |
| --- | --- |
| **NAME** | **ID NUMBER** |
| Meet Parikh | 2015A8PS0375P |
| Dwait Bhatt | 2015A3PS0284P |

## **Problem statement:**

Social media data carries abundant hidden occurrences of real-time events. In this paper, a novel methodology is proposed for detecting and trending events from tweet clusters that are discovered by using locality sensitive hashing (LSH) technique.

## **Background of the problem:**

There is an abundance of data available on social media sites regarding real-time events. The first systematic work concerning the topic and event detection in the newswire document corpus was conducted during the topic detection and tracking (TDT) research. TDT research focuses on finding techniques to identify event documents from a group of news documents.

There are five different tasks of a TDT system:

* Story segmentation: The process of dividing news into individual stories.
* First story detection (FSD) or new event detection (NED): The stories are monitored to detect the events which have not been seen before, also known as new event detection.
* Cluster detection: Similar stories are grouped under a specific topic.
* Tracking: News streams are monitored for additional stories which are related to previously existing stories.
* Story link detection: The link between stories discussing the same topic of interest is identified.

These tasks combined together can be used to detect new events occurring around the globe. Each task of TDT system has caused a separate research problem in the field of information retrieval. The focus of this paper iscluster detection.

**Why Twitter?**

Twitter, the microblogging site started in 2006, has become a social phenomenon. More than 340 million Tweets are sent out every day. While a majority of posts are conversational or not particularly meaningful, about 3.6% of the posts concern topics of mainstream news. Owing to its fixed post length of 140 characters, it is an invaluable resource for TDT research.

Twitter has also been credited with providing the most current news about many important events before traditional media, such as the attacks in Mumbai in November 2008. Twitter also played a prominent role in the unfolding of the troubles in Iran in 2009 subsequent to a disputed election, and the so-called Twitter Revolutions in Tunisia and Egypt in 2010-11.

Because of these reasons, there is a growing need for systems that can extract useful information from this amount of data.

**Technical issues**

Currently the spelling correction algo used is quite slow and primitive. First, it is not context aware, i.e. it is isolated spell correction. This in itself is a big drawback. Second, the model used for suggestions is not accurate. Words at a Levenshtein distance 1 are always preferred over those at distance 2 as possible candidates for the misspelled word. Also, words farther than an edit distance of 2 are never suggested. These issues can be rectified by using context aware spell correction, and finding ways to make the algo more efficient so as to incorporate word suggestions at edit distance of 3 or higher.

Abbreviations and emojis, which are used extensively in tweets have not been filtered out in the preprocessing. This might lead to certain common abbreviations or emojis to be falsely recognised as the representative features of a cluster.

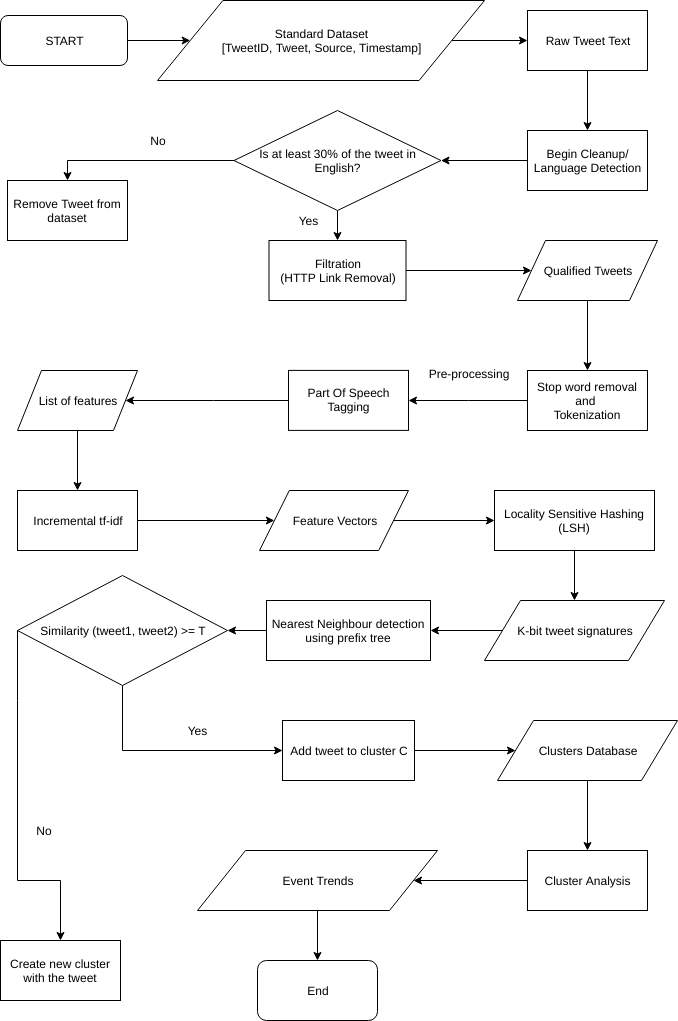
## **Literature Survey:**

The detection of events in Twitter has been the goal of many studies. It is mainly approached as a clustering problem, with burstiness as the most important characteristic to detect an event. The most salient dichotomy among approaches is called document-pivot clustering and term-pivot clustering: burstiness is either measured at the level of tweets that share common terms, or at the level of single terms that display a joint burstiness over time. This paper deals with a document-pivot clustering technique.

Parikh et al. developed an automatic scalable system for event detection from tweets, called the ET system [1], which employed a term-pivot clustering technique. Key components of their approach were: (a) extraction of keywords to represent events, (b) an efficient storage mechanism to store the patterns, and (c) a hierarchical clustering technique for event detection.

Locality sensitive hashing (LSH), a technique for approximate nearest neighbor search, was first articulated in [2,3]. Since then, LSH has become a state-of-the-art technique for similarity search in high dimension data [4-8]. Slaney et al. contributed an algorithm to calculate the optimum parameters for nearest-neighbor search using LSH [9]. Charikar used LSH in [10] to map high dimensional vectors to low dimensional space while preserving similarity between vectors in original space. While different types of approaches have been carried by researchers toward event detection in social media, applying LSH technique for event detection in social media [4,11] is fairly a recent development. Petrovic et al. proposed a method using LSH to speedup FSD, which could run on social streams (i.e. on incoming tweets). In the proposed methodology, LSH approach is used for discovering tweet clusters to identify the events.

## **System Description:**



**Methodology:**

A dataset of tweets posted between 23:29:01 and 23:59:59 on 21st January, 2017 (the day of the Women’s March in the USA) was taken from data.world as standard dataset for the event detection program. The dataset consists of tweetIDs, the tweet itself, the source of the tweet(Android, iPhone, Website etc) and the exact time of posting.

Tweets consist of a lot of noise pertaining to the fact that there are no specific rules on the kind of words, abbreviations or language to be used for a post. Also, there tend to be a lot of spelling mistakes. So, in order to make the dataset effective, we needed to clean up the data of these issues.

As can be seen in the block diagram, the first step in the process was language detection since our program is inclined to detect events from tweets written predominantly in the English language. All the tweets were processed through a language detection function and any tweet having less than 30% use of the English language were discarded. A link and tag removal function was taken into use to remove the HTTP URLs and the tags of other twitter users.

Now, the tweets were ready to be used for extraction of features. The features of a tweet are mainly the kind of words used in it. Now, if there is a word like ‘America’ in a tweet, we can say that it definitely carries more weight in knowing the meaning and sentiment of the tweet as compared to a word like ‘the’. Such unimportant and high frequency words are known as stopwords. To ensure that such words do not become a part of the features list, we removed these stopwords using the Natural Language Processing Toolkit (nltk) module. Next, for ease in feature extraction, we tokenized the tweet using a word tokenizer. Finally, we used nltk for POS(Part of Speech) tagging which would be helpful in knowing the important words in the tweet. In most cases, it is seen that the nouns and the verbs sum up the meaning of a statement. So, we extracted all the nouns and verbs as features from the tweet.

Finally, for the measure of similarity between two tweets, we used the cosine similarity algorithm using tf (Term Frequency) and idf (Inverse Document Frequency). Tf is given by the number of times a word occurs in a tweet. Idf is given by the df (Document Frequency) which is the number of tweets in which the word occurs. Idf(word) = log.

The similarity is given by tfidf. Here, since the dataset is huge, we use incremental tf-idf based similarity measure. Here, we distribute the dataset in different groups D1, D2, … Dn.

Then, we find tf idf for words in each of those datasets.

Now that we have the tweet feature vectors ready, we move on to forming clusters by finding similar tweets in this high dimensional space. The brute force method to add a tweet to a cluster would be comparing it with all other tweets and adding it to the cluster of the one which is closest to it. But the time complexity of this approach is O(n), which is pretty slow as our dataset is huge. So, the paper proposes to use Locality Sensitive Hashing (LSH), a technique which will reduce the complexity to O(log(n)) for adding a tweet to a cluster.

The basic idea behind LSH is that we first obtain K-bit ‘signatures’ for every tweet vector to reduce the dimensionality. This is done by comparing a tweet with K hyperplanes in the high dimensional space, and assigning either 0 or 1 depending on which side of the plane the tweet vector lies. Now tweets with the same signature or having a signature at a small hamming distance from the current tweet’s signature lie in a space close to the current tweet. So now instead of comparing a tweet with all other tweet vectors, we only need to compare with these closeby tweet vectors. This is what reduces the time complexity.

How this idea is usually implemented is as follows : The signature of the tweet is used as the index for a hashtable, so that all tweets with the same signature collide and fall in the same bucket. Now a similarity measure like cosine similarity is used to find nearest neighbor and the tweet is added to its cluster. This processes has been further improved in the paper by using prefix trees to store and find signatures at small hamming distances from the current one. The signature of a tweet is permuted in P ways and these permutations are stored in P different prefix trees. The nearest signature is found in each of the P trees and finally the one at the smallest hamming distance is picked as the nearest neighbor. Finally, the tweet is added to the cluster of this nearest neighbor.

After the clustering is done, all singleton clusters are removed and minimal cluster data is stored in a database. The id, label, and size of each cluster is stored in the database. The label of the cluster is obtained by taking the n most frequently occurring terms in the tweets in the cluster.

**Improvement by using hashtags :**

The paper made no use of hashtags in the clustering process. We propose an improvement by comparing the hashtags of tweets as well while calculating similarity. Instead of using just cosine similarity between tweet feature vectors, we used it along with Jaccard similarity between the hashtags in the tweets. This led to better clustering as the number of clusters reduced.

**An advanced design of the Spell-correct algorithm :**

Presently, the spell-correct algorithm used in the autocorrect module in Python is based on the Norwig implementation of the spell correct algorithm. These algorithm can be further modified to run faster and better. The major issues with the Norwig algorithm are as follows:

* It depends on a dictionary of words. If the word to be corrected is not in the dictionary, we won’t get the correct answer.
* Words at edit distance 1 are given priority over words at edit distance 2 even when the possibility of the correct word to be at edit distance 2 is significantly high.
* There’s a fixed maximum edit distance, in this case, 2. If the correct word is at an edit distance of 3, we won’t get the correct answer.
* The correction is not context-based. This can lead to getting a wrong answer even in some obvious cases.

To take in care some of these issues, we can implement a customized SymSpell (Symmetric Delete spelling correction) algorithm. This algorithm is 1 million times faster than the Norvig algorithm.

The idea behind the algorithm is that normally, spell-correct algorithms compute the Levenshtein distance on the basis of 4 operations (insert, delete, replace and transpose). Here, we would take only deletes into consideration. Transposes, replaces and inserts of the input term are transformed into deletes of the dictionary term. This delete-only philosophy is what gives the algorithm its speed.

Let us consider a maximum edit distance of 2. First of all, we would get a deletes list. Here, we would derive all the strings that can be obtained from a word with upto maximum edit distance number of deletions. This would give a large list of numbers. E.g. ‘meet’ would give [‘eet’, ‘met’, ‘mee’, ‘et’, ‘ee’, ‘mt’, ‘me’]. Next, the word and it’s deletions are added in the existing dictionary of words. If the word is already in the dictionary, it’s frequency is updated.

Now, a major function would be the Levenshtein distance calculator between the two words which would be later used for correcting the spelling. After calculating the Levenshtein distance, the suggestion function would be implemented which would decide whether the word is to be suggested as a possible correct spelling of the input word. First, all the words in the dictionary which are not deletions and don’t have lengths greater than a minimum suggest length are added in a suggest dictionary along with the length difference. Get the Levenshtein distance of the words in the suggest dictionary from the input word. The dictionary is then sorted by decreasing order of frequency and increasing order of edit distance simultaneously. The first entry in the dictionary is provided as the correct spelling suggestion of the input word.

The reason why this algorithm is so fast is that it only believes in deletion. As a result, during calculation of the Levenshtein distance, no attention has to be paid to insertion, replacement and transpose operations which makes the process quick.

## **Results and Evaluation:**

**Strategy:**

For preprocessing of the dataset, the removal of non-discriminatory terms is important. For evaluating the effectiveness of the algo, we compared the average tweet length before preprocessing and the average number of features per tweet after the process.

Some results :

* In the beginning, the dataset consisted of 29468 tweets. On cleanup, 24343 tweets remained.
* Originally, the average number of characters per tweet was 91.9 (92). After cleanup, this number fell down to 78.7(79) characters per tweet.
* The average number of features per tweet is 12.12 (12)

The effect of the preprocessing functions at every stage is shown below for a sample tweet:

|  |  |
| --- | --- |
| Original Tweet | I thank God that we have each other! #WomensMarch #WomensMarchLA https://t.co/kHAFzwsFhG |
| After Cleanup | I thank God that we have each other! #WomensMarch #WomensMarchLA |
| Features | {thank, god, womensmarch, womensmarchla} |

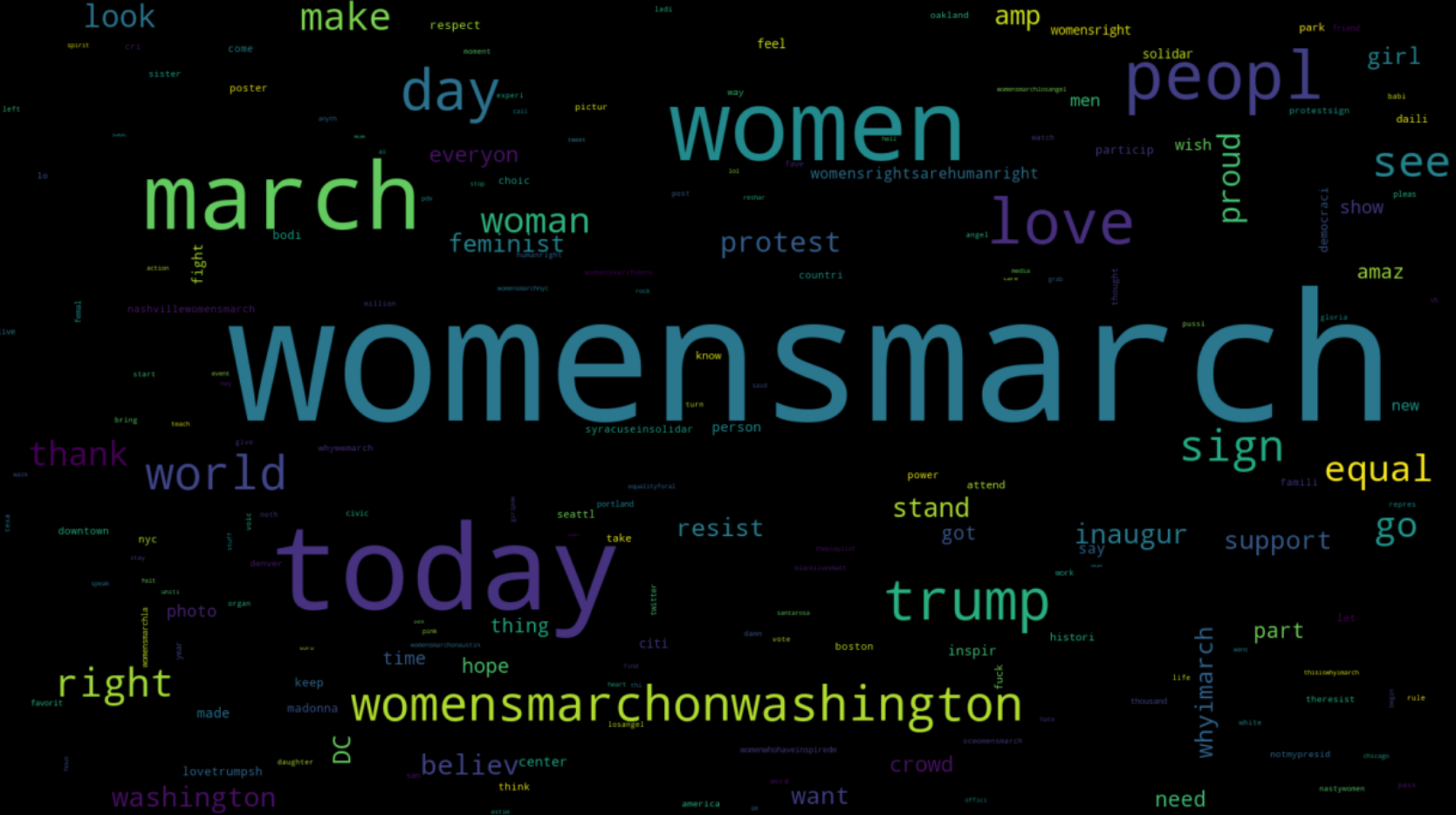
**Result of the hashtag improvement :**

Number of (non-singleton) clusters without using hashtags : 1625

Number of (non-singleton) clusters using hashtags : 1163

Number of (non-singleton) clusters after merging clusters with the same label : 307

## For visualization of the clustering, we used a wordcloud to show the top words in each cluster, with their font size indicating their frequency in the top words list.



## 

## **References:**

[1] - R. Parikh, K. Karlapalem, ET: events from tweets, in: Proceedings of the 22nd International Conference on World Wide Web Companion, Rio de Janeiro,

Brazil, 2013, pp. 613–620.

(<https://www.bibsonomy.org/bibtex/1dd1b4f4954b48b73dc4fcbddd9809aad>)

[2] - J. Allan, V. Lavrenko, D. Malin, R. Swan, Detections, bounds, and timelines: UMass and TDT-3, in: Proceedings of Topic Detection and Tracking Workshop,

2000, pp. 167–174.

(<https://pdfs.semanticscholar.org/340d/945572cd75b05258696886037002c5b9cb3b.pdf>)

[3] - J. Allan, in: J. Allan (Ed.), Topic Detection and Tracking, Kluwer Academic Publishers, Norwell, MA, USA, 2002, pp. 1–16.

(<http://www.springer.com/gp/book/9780792376644>)

[4] - S. Petrovic, M. Osborne, V. Lavrenko, Streaming first story detection with application to Twitter, in: Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational

Linguistics, Los Angeles, CA, 2010, pp. 181–189.

(<http://www.aclweb.org/anthology/N10-1021>)

[5] - P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: Proceedings of the 30th Annual ACM Symposium

on Theory of Computing, Dallas, TX, USA, 1998, pp. 604–613.

(<http://www.cs.princeton.edu/courses/archive/spr04/cos598B/bib/IndykM-curse.pdf>)

[6] - S. Petrovic, Real-Time Event Detection in Massive Streams, PhD Dissertation, Institute for Language, Cognition and Computation, School of Informatics, University of Edinburgh, UK, 2012.

(<http://homepages.inf.ed.ac.uk/s0894589/petrovic-thesis.pdf>)

[7] - P. Indyk, R. Motwani, P. Raghavan, S. Vempala, Locality-preserving hashing in multidimensional spaces, in: Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, USA, 1997, pp. 618–625.

(<http://www.cs.princeton.edu/courses/archive/spr04/cos598B/bib/IndykMRV-local.pdf>)

[8] - A. Dasgupta, R. Kumar, T. Sarlos, Fast locality-sensitive hashing, in: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and

Data Mining, San Diego, CA, USA, 2011, pp. 1073–1081.

(<https://dl.acm.org/citation.cfm?id=2020578>)

[9] - M. Slaney, M. Casey, Locality-sensitive hashing for finding nearest neighbors, IEEE Signal Process. Mag. 25 (March) (2008) 128–131.

(<http://ieeexplore.ieee.org/document/4472264/?reload=true>)

[10] - M.S. Charikar, Similarity estimation techniques from rounding algorithms, in: Proceedings of the 34th Annual ACM Symposium on Theory of Computing,

Montreal, Quebec, Canada, 2002, pp. 380–388.

(<https://www.cs.princeton.edu/courses/archive/spr04/cos598B/bib/CharikarEstim.pdf>)

[11] - B. Sharifi, M. Hutton, J.K. Kalita, Experiments in microblog summarization, in: Proceedings of the 2010 IEEE Second International Conference on Social Com-

puting, 2010, pp. 49–56.

(<http://www.cs.uccs.edu/~jkalita/papers/2013/SharifiBeauxComputerJournal2013.pdf>)