International Journal of Intelligent Computing and Information Science

REAL TIME FIRST STORY DETECTION IN TWITTER USING A MODIFIED TF-IDF ALGORITHM

S. Elbedwehy M. Alrahmawy T. Hamza

Computer Science Department Faculty of Computer and Information Sciences, Mansoura University - Egypt
samarelbedwehy@mans.edu.eg mrahmawy@mans.edu.eg taher_hamza@yahoo.com

Abstract: *Twitter is a social micro blogging, it has its own feature that it enables to tweet only a maximum of 140 characters per tweet. Even with this small number of characters per tweet, analyzing the tweets for billions of users faces the challenges of real-time data processing. One of the important aspects of social behavior is that we can detect the significance of the events and the way the people reacted to them. In this paper, we focus on First Story Detection (FSD) that means we can detect bursts of tweets that refer to a particular topic. First story is defined as the first document from a given series of documents to discuss a specific event, which occurred at a particular time and place. TF-IDF denotes to term frequency–inverse document frequency is an algorithm traditionally used in most of Text similarity applications like FSD. In this paper, we embedded a modified version of TF-IDF algorithm to enhance the accuracy of a pre-implemented open source for FSD that uses Storm platform to benefit from its scalability, efficiency and robustness in analyzing the tweets in real time. The empirical results show significant enhancements in the accuracy of the detection without noticeable effect on the performance.*

Keywords: *Real Time, Similarity Algorithms, Social Media, Information Retrieval, Big Data.*

1. Introduction

The rise in mobile devices usage along with many applications available for Twitter allows users to publish updates easily and frequently. Twitter has huge information posts every second, this means huge events. The data of these events need to be processed and analyzed. Twitter is one of the popular micro-blogging websites. It has a private property for posting tweets; each tweet can only have a maximum of 140 characters; with these characters people can share their own stories. In Twitter, people can follow each other and re-tweet their stories. Twitter has 310 million monthly active users, *one billion* unique visits monthly to sites with embedded Tweets [1], which mean that the saved data within the twitter blogs is huge. It generates more than 400 million tweets (events) each day [2]. These events need to be processed. Using algorithms that not just have high accuracy, but also they have to be efficient to do their tasks with high performance on the huge volumes of data in real-time.

FSD means determining the first story to discuss a particular event, by monitoring the streams of arriving new tweets (stories), or determining which tweet (event) is a first story. FSD was firstly defined by [3] in terms of topic detection and tracking. First stories (First Tweets) examples can be the 8.8 magnitude earthquake which occurred in Chile on 27 February 2010 or Egyptian revolution on 25 January 2011. One Recent example is the Egypt Air *flight crash* in Mediterranean on May 2016 or

Suicide attacks which hit Madinah on July 2016; Figure 1 shows samples of tweets or events. Such events spread widely over Twitter showing the noticeable impact they have on the real world. Needing to get more accurate data will help in finding FSD and this problem needs research. Most existing FSD searches uses TF-IDF [4] algorithm. In this paper, we introduce modifications on this algorithm to improve its accuracy results without affecting its performance.

Figure 1: Sample of tweets speaking about Suicide attacks hit Madinah

In our work, we modified an FSD open source that is based on Storm. Storm is an open source for processing events in real time in distributed systems. It is fast, scalable, fault-tolerant and enables processing the tweets in parallel.This paper is organized as follows: next section presents previous work and the basic algorithms for FSD; we discuss in section 3 the basic FSD model including Local Sensitive Hashing (LSH), STORM and DRPC. We discuss the TF-IDF algorithm that is commonly used in FSD systems and its enhanced version mTF-IDF that we used in our work in section 4; with an example to illustrate how it works. Evaluation is presented in section 5, where we compare the results of the proposed modification with the original. Finally a conclusion of the work is presented in Section 6.

2. Previous Work

In the last few years, micro-blogs have taken much of people attention. Most people use social network daily. Due to this daily usage and with huge number of users, data has become huge too. This attracted many researchers to investigate the processing methods of this type of data, like how to analyses it. They focused on the properties of Twitter as it is considered a rich data source [5, 6].

First stories are extracted from tweets [7] was the initial search made in FSD. Researchers used Local Sensitive Hashing (LSH) to build FSD system for first time in [8]. This approach aims at reducing the number of comparisons that need to find nearest neighbor among number of tweets. Recently, authors in [9] performed a first story detection on streaming data with application on Twitter using LSH with a proposed a variance reduction strategy which imposes that the query is compared with only a given number of previous points. Most researchers focused on using TF-IDF algorithm in finding match stories in [9, 10, 11, and 12]. In 2012,Vogiatzis [13] worked on improving the scalability and the speed of FSD as previous approaches focused only on the accuracy; he used a distributed approach based on

using Storm open source, which can process an incredible amount of input data (millions of tweets) and demands high computational power to process them in real time. In 2013 Mikolov [14] presents a method for efficient, unsupervised computation of continuous vector representations of words, and the result is the open-source toolkit named Word2vec. Latest search in 2016 used it also to enhance FSD effectiveness [15].In 2014 Leand Mikolov [16] propose a method they call Paragraph Vector or Doc2vec where the Word2vec model is adopted to create an unsupervised algorithm for learning a fixed-length feature representation from variable-length pieces of text. In 2013 and 2014, BM25 algorithm used to improve results to be more accuracy in [17, 18]. In 2016 Vuurens [19] propose method for news summarization based on 3-nearest neighbor clustering which is effective than a baseline that uses dissimilarity of an individual document from its nearest neighbor.

Most approaches based on TF-IDF convert each tweet to a vector and make LSH after that. TF-IDF is a simple approach that tries to get accurate results; but the highest TF-IDF words of a document may not make sense with the topic of the document. We focused in this paper to improve converting words to vector by using another algorithm called mTF-IDF [20] which is an enhanced modification of TF-IDF that makes results more accurate and in the same time, to enhance the accuracy, we embedded this algorithm in an FSD that works on the distributed open source Storm; as explained in section 3.3.

3. First Story Detection

The purpose of FSD is determining "a first story" from arriving streaming of tweets. First story means a new topic that happened at specific time and place [9]. We can get benefits from detecting new events like adding social component, so we can know how people can react with it, and also know if there is a Rumor, who can tweet the first story for it. Algorithm (1) shows a pseudo code used by the UMass system [9]. This is a general approach based on nearest-neighbor search; it is used in UMass and CMU [9]. It represents documents as vectors, where each new document is compared to previous ones. If the similarity to the closest document is less than a certain threshold, the new document is considered the first story, where $dis_{(min)}(d)$ is the novelty score assigned to document d.

The structure of an FSD system is based on a nearest neighbor search methodology that follows the following steps: first; a stream of tweets is acquired; then, each tweet is split into words, where whitespace and special character like #, $, etc. are removed; then, a vector space for each word is computed by calculating TF-IDF weight for each word. LSH is applied on each word by calculating hash value for each word. Then results are stored in buckets as discussed later in section 3.1

Table 1: General Approach to FSD

Algorithm 1: Pseudo code for the General Approach of FSD

```
Data: corpus
foreach document d in corpus do
    foreach term t in d do
        foreach previously seen document d' that contains t do
            update distance (d, d')
        end
    end
    dis_(min)(d)=min_d' = {distance(d, d')}
    Add d to inverted index
End
```

Then incoming tweets are compared with tweets residing in the buckets that have the same hash value. Finally, the distance between each tweet and the nearest neighbor is computed, e.g. using cosine similarity, where the decision on the distance is based on using a threshold, if the distance < threshold, it's not First Story; otherwise, it is considered as First Story [9]. As shown in Figure 2 that shows a detailed block diagram of the steps of FSD.

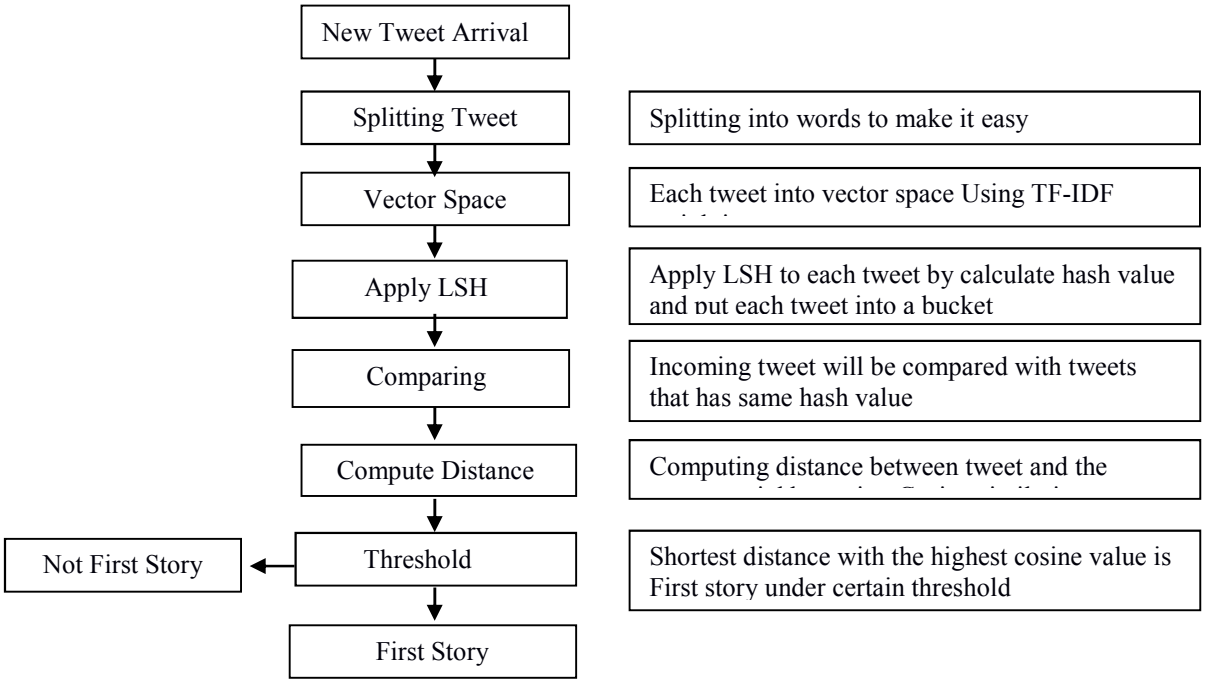


Figure 2: Block Diagram of FSD steps

3.1 L S H Overview

LSH denotes to Local Sensitive Hashing, sometimes called near neighbor search. The aim of LSH is to reduce the probabilities to find the approximate neighbor among documents or tweets. When new tweet arrives, it is split into words so that the partial counting can be applied, we use normalization like OOV (out of vocabulary) which is a method for reducing lexical variation that occurs in different documents instead of using porter stemming as it doesn't give high results in accuracy [21].

For example: a word like *u* can be replaced with *you*, *sureeee* can be replaced with *sure* and so on. Also special characters; like @, $, %, &... etc., are removed. LSH algorithm continues by representing each tweet in the vector space by the traditional algorithm TF-IDF weighting after we converted the tweet into vector, we apply LSH to find the nearest neighbor candidates.

Figure (3) shows LSH logic, where a *b* hash functions are applied to map N database examples into a hash table, where similar items are likely to share a bucket. After hashing a query Q, one must only evaluate the similarity between Q and the database examples with which it collides to obtain the approximate nearest-neighbors [22].

LSH uses buckets called hash tables for similar tweets, by this method; once a new tweet arrives, its hash value is computed and it is put into one of several buckets. Within the same bucket, the probability

of collision with similar tweets is much higher, because tweets that have identical hash with the arriving tweets are nearest neighbor candidates. After collecting the nearest neighbor from all buckets, we compare the distance between the tweet and the nearest neighbors using the cosine similarity measure [13] by applying equation (1).

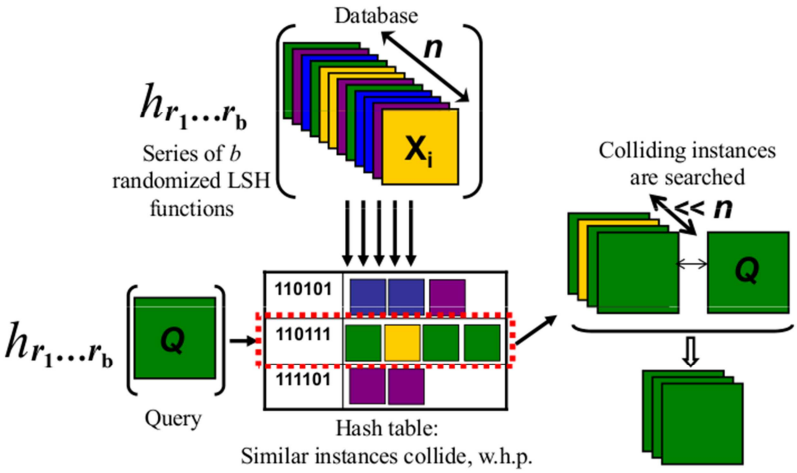


Figure 3: LSH logic [22]

The neighbor with the highest cosine score see equation (1) is assigned as the nearest and the distance represents the *score* of the tweet. If a tweet has score above a certain threshold, it is identified as a *First Story*.

$$cos(\theta)\frac{u \cdot v}{\|u\|.\|v\|} = \frac{\sum_{k=1}^{n} u_i x v_i}{\sqrt{\sum_{k=1}^{n}(u_i^2)} x \sqrt{\sum_{k=1}^{n}(v_i^2)}} \qquad (1)$$

3.2 Storm

Storm is a distributed real time event processing system which distributes executors and workers among the cluster nodes, we discuss the meaning and the jobs of each in the following: *topology, bolt, spout, tuple* and *stream grouping,* see Table (2). *Tuple* is a list of values, where each value can be any type of object, a *Stream Grouping* is unlimited sequence of tuples, *Spout* is the source of streams which can receive messages from external sources or can produce tuples by itself, it is used to get the tweets submitted by people. *Bolt* is acting as logical units of the application emitting new streams. *Topology* is the network of bolts, spouts and streams, which is the top-level abstraction we submit to Storm as shown in Figure (4).

A topology never terminates unless we kill it explicitly and this is a good advantage for Storm. A stream cluster as in Figure 5 consists of two kind of nodes; master node and worker node. The master node is called “Nimbus” which is responsible for distributing code around the cluster; monitoring any failures happened in the cluster and can assign tasks to machines. The worker node is called “supervisor”, it is responsible for starting and stopping worker process which is responsible for the topology execution.

Topology can be executed by several worker processes and every worker process can run one or more executors [23]. Storm can manage the cluster by Apache Zookeeper, which is an open source in Apache project that allows distributed processes in large systems to synchronize with each other. It makes clients able to receive consistent data.

One similar framework to Storm is Hadoop. Hadoop supports *batch processing*. Batch processing occurs when all inputs are already received. This makes such systems slow [24] and its speed is dependent on both the size of the data being processed and the computational power of the system, this may cause the output to be delayed significantly, which is not acceptable in real time system.

Table 2: Main Elements in Storm

Tuple	A list of values
Stream Grouping	Unlimited sequence of tuples
Spout	The source of streams which can receive messages from external sources or can produce tuples by itself
Bolt	Acting as logical units of the application like emitting new streams
Topology	The network of bolts, spouts and streams, which is the top-level abstraction we submit to storm

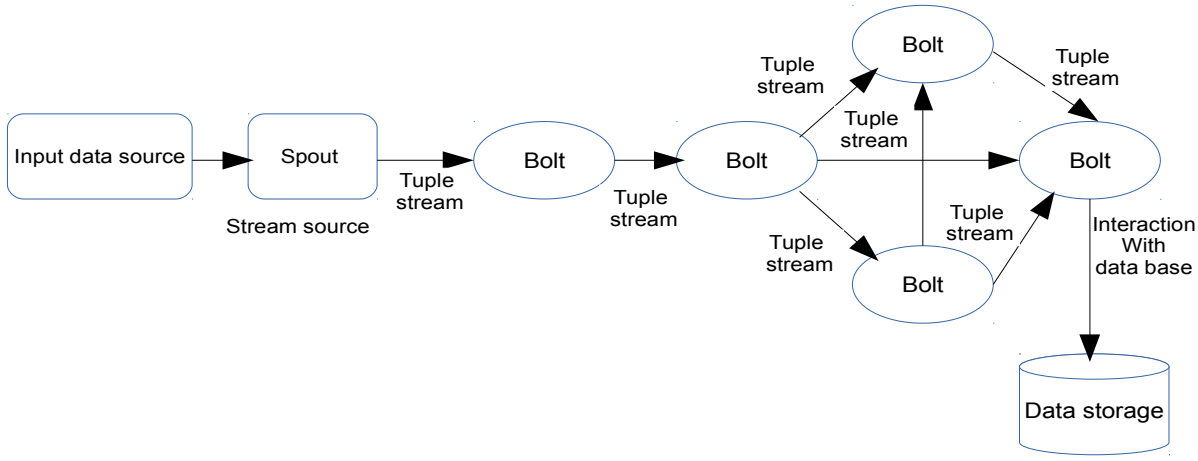


Figure 4: Core concept of Apache Storm [22]

Storm is an open source since 19th September 2011; it is coded in Java and Cloujre. Many companies uses Storm; including Twitter, Alibaba and Groupon. Storm can provide data processing and high fault tolerance to large amounts of data as it is a distributed real time system. In October 2012, Storm moved from Git-hub to Apache to ensure the long-term success of Storm.

Another distributed platform called S4 [26] can spread computations over network machines as it is distributed processing system, partially fault tolerant platform written in Java, can process unbounded data streams as in Yahoo! Inc. The Two platforms are different in some points but it can be similar in another. Storm was released after S4 but has fast and active apache hosted group, there are developers and users of Storm that can help in many problems. A Storm topology is defined using simple XML file in contrast to the complexity in code definition for S4. Storm displays easy interface for monitoring the cluster by Nimbus. Storm gives its users responsibility for their choices. S4 can lose data through communications so it can be less fault tolerance than Storm. Storm can be selected as main system but S4 can be alternative system.

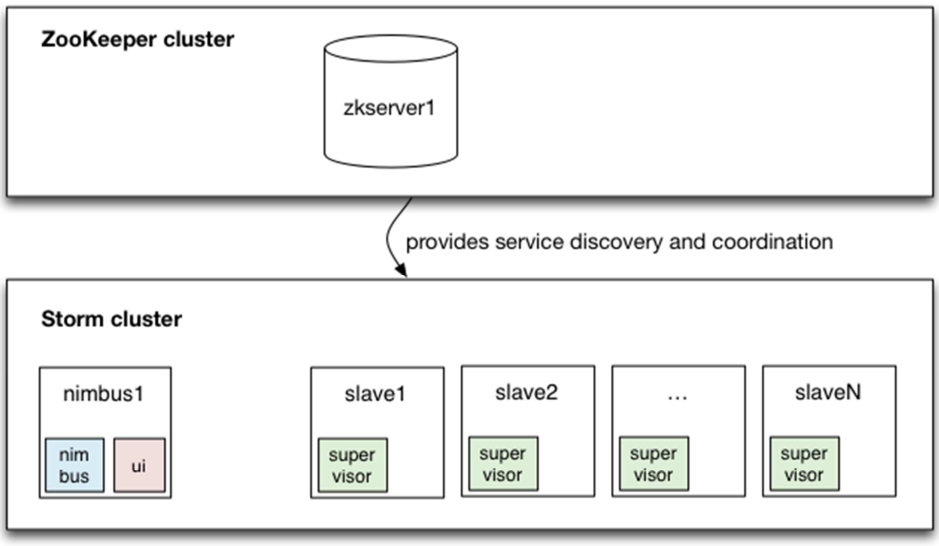


Figure 5: Simple Storm Cluster [25]

3.3 DRPC

The usage of DRPC adds benefit of scalability and speed to Storm. Storm uses a DRPC Server to handle requests which are already implemented in Storm. As in Figure 6, we can see that DRPC Server receives the name of the function that is needed to execute and its arguments from the client. The streams that come from a spout can be read by the topology, and then the topology computes the function and emits the results to a special bolt that has the request-id. Then, the bolt connects to the DRPC Server to send the results as a reply to this particular request. Finally, DRPC Server sends the results to the host and port that the client defined it in the first stage. Once the result arrives to the client, DRPC Server unblocks the waiting client. An important feature of DRPC is the usage of a bolt interface called "IbatchBolt". The role of IbatchBolt is allowing tuples to be emitted in batches and to wait before a tuple is fully processed by all tasks inside a bolt before emission [27].

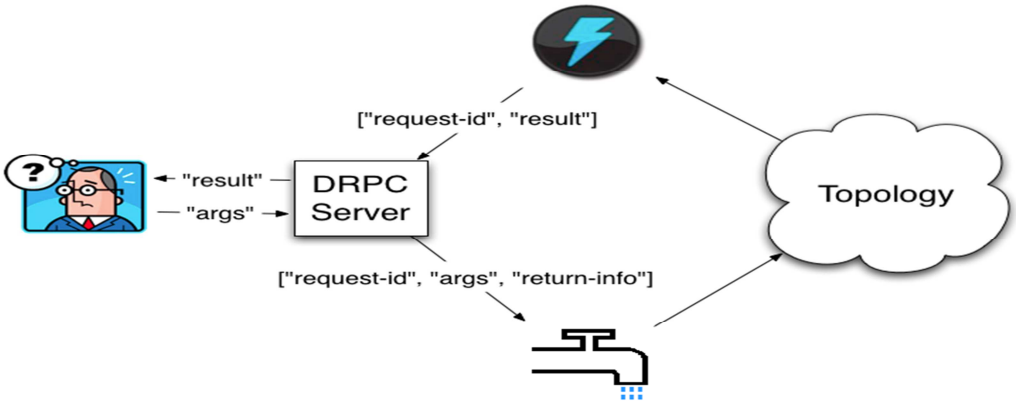


Figure 6: DRPC Data Flow [27]

4. Proposed Enhanced FSD

We mentioned in section 2 that most of the existing FSD use TF-IDF, here in the next section we

overview algorithm; then we present the modified TF-IDF (mTF-IDF) and show how we employed it as an efficient replacement of the traditional TF_IDF in FSD implementation. Figure 7 represents the general stages of applying TF-IDF/m TF-IDF.

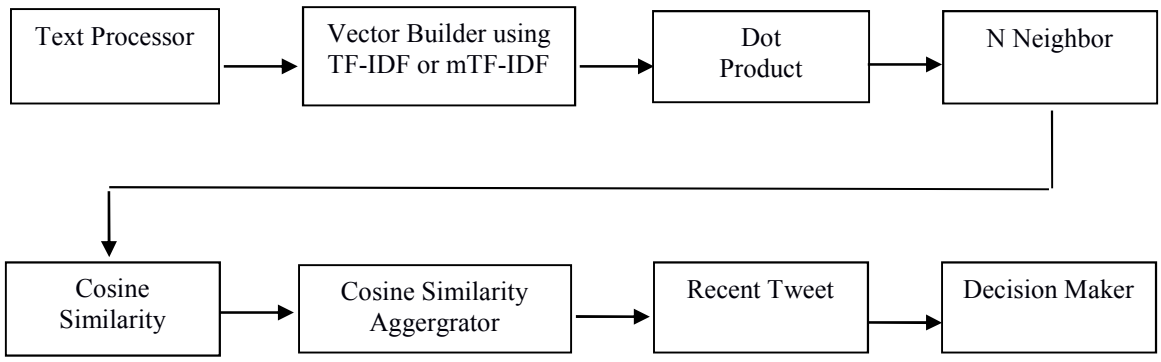


Figure 7: Scalable Distributed Topology with TF-IDF or mTF-IDF

In our proposed model, we adopted the FSD model implemented upon Storm in [13]. This model provides a topology of Bolts for extracting the first story; these bolts and their jobs are presented here.

- Text processor Bolt

Once a tweet arrives at this bolt, it splits it into words after removing any special character like @, #, $, % ... etc., reducing lexical variations by using OOV and removing URLs in the tweet.

- Vector Builder Bolt

The base Vector Builder bolt is used to convert each tweet to a vector. The vector builder bolt works with single task, it receives tweet identifier with tweet body "text"; then, it emits a tweet vector with number of new words present in the tweet.

- Dot Product Bolt

This bolt runs several identical tasks in parallel on several buckets. This bolt consists of hash tables "Buckets" that store hash values of previous tweets. Each bucket has a *k* number of random vectors each of which is a single dimensional dense matrix of size *k*. multiplication of a tweet vector by a random vector produces a dot product. Based on the LSH logic, similar tweets have high probability to hash to the same value. Therefore, tweets with the same hash in each bucket are possible near neighbors. Hence, this bolt produces a list of near neighbors from all buckets and returns it as a result.

- Nearest Neighbors Bolt

This bolt is responsible for a single task which sorts the tweets according to the frequency of collision between stored frequencies with the incoming input tweet. Since tweets are hashed into the buckets; the probability of colliding is high.

- Cosine Similarity Bolt

This bolt is responsible for measuring the similarity distance between each tweet's vector and vectors of nearest neighbors by using Cosine similarity measure. This bolt emits its value into Cosine Similarity Aggregator Bolt.

- Cosine Similarity Aggregator Bolt

It makes comparison between the received nearest neighbors from previous bolt. This step uses a threshold to determine if the tweet is old story or not; if the nearest neighbor distance is more than a predefined threshold value (0.3 in the implemented model), then it is a candidate of a new-story tweet and this bolt emits a tuple of <tweet, recent tweet> to the next bolt; otherwise, the tweet is classified as old story.

- Recent Tweets Bolt

This bolt contains the tweets that have a distance larger than the threshold value; if the neighbor distance is less than a given threshold. This bolt compares the recent 500 tweets with the input tweet.

- Decision Maker Bolt

This bolt emits tweet and score. Score is computed as$s = 1 - d$, where $d$is the minimum measured distance. If the score s is greater than threshold value, then tweet is an old story. Otherwise, it is classified as a new story

4.1 Proposed Distributed Topology

The main aim of a Storm is to distribute work among different machines to process tasks in parallel in real-time. In our proposed topology on Storm, shown in Figure 7, we distribute tasks to each component; each bolt receives its assigned job and processes it before it emits its results to another bolt. Bolts are the logical units of the application and can perform jobs such as running functions, filtering, streaming aggregations, streaming joins and updating databases.

In our work, we group tweets by their tweet_ids to make sure that we deal with tweets that collide only with this input tweet and to perform a count on them. Then, we use a Count aggregator which keeps the input fields in the output. The aggregate operation is responsible for removing any fields the tuple had before the aggregation except the grouping fields. The current computation gives us a tuple for each colliding tweet and a number of how many times this tweet was found in all buckets. Then we sort the tuples using a FirstNAggregator aggregator, see Figure 8.

This aggregator takes as input the max number of tuples to emit the sorting field and the sorting order (true if reversed). Aggregator requires that the output fields must correspond to the input fields and the sorting field not have the same name as before. Compute Distance is the function which computes the cosine similarity between the tweets' vectors and emits it as a value. Then a grouping by tweet id follows before sorting. FirstNAggregator is used also to sort the tweets on cosine similarity and only pick the closest colliding tweet to the input tweet.

Also we have Decider aggregator which is deciding the results; it uses aggregator of type combiner for implementation. Combiner aggregates the tuples in each partition before transferring them over the network to maximize efficiency. According to the values in each tuple, it decides whether the nearest tweet comes from the buckets or from the most recently seen tweets queue.

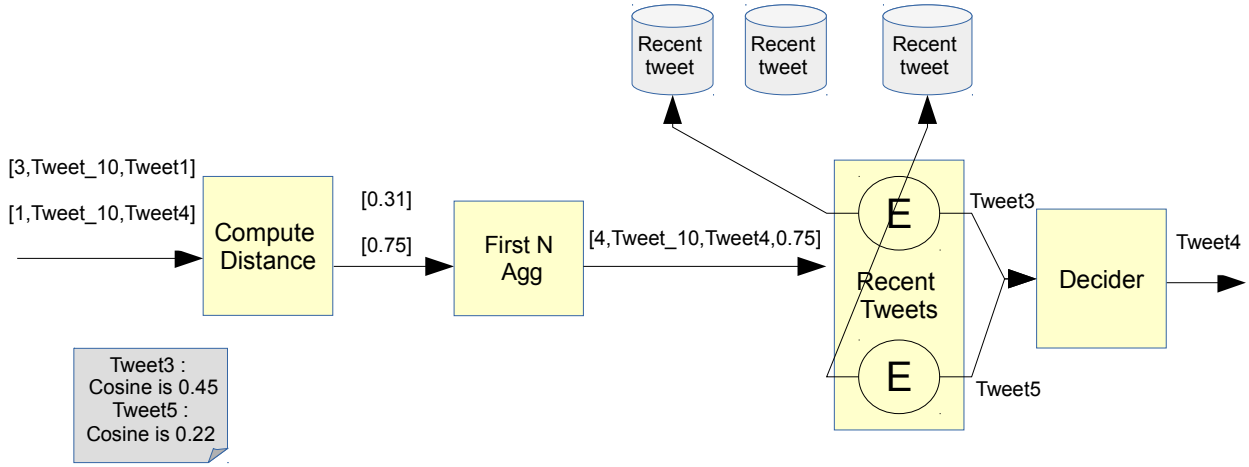


Figure 8: Example for using Aggregator function adaptive from [22]

4.2 Traditional method (TF-IDF)

There are several text mining algorithms commonly used in Information retrieval and web classifications like TF-IDF, Glasgow [20], and Entropy [28]. TF-IDF is the most one in use; it is efficient and simple algorithm [29]. TF-IDF can be seen as a combination of two algorithms; Term frequency (TF) and Inverse Document Frequency (IDF). TF works, as seen in equation (2), by computing the normalized frequency of a certain term that exist in one or a small set of documents; this ensures that higher TF-IDF values are computed for common words in these documents. In Twitter-oriented FSD systems, tweets can be seen as short documents, and each individual word in a tweet represent a term which its frequency can be calculated using equation (2).

$$TF_{t,d} = \frac{tf_{t,d}}{\sqrt{\sum_{t=1}^{n} tf_{t,d}^2}} \qquad (2)$$

Hence, the nominator in equation (2) contains $tf_{t,d}$,which represents the number of occurrences of a word t in a tweet d, while the denominator is known as the Euclidean norm of the tweets in which $tf_{t,d}$ is a frequency of t^{th} word in tweet d, and n is a number of distinctive words in tweet d.
On the other side, IDF_t, see equation (3), is very important for statistical measure that supports the assumption that a more frequent word in the collection is considered less important.

$$IDF_t = log\left(\frac{N}{DF_t}\right) \qquad (3)$$

In Twitter-oriented FSD systems, N represents the total number of tweets in the collection, $DF_t$is a number of tweets within the collection that contain the word.
TF-IDF algorithm Combines both TF and IDF by multiplying them together (i.e., TF-IDF=TF*IDF) as seen in equation (4)

$$TF - IDF_{t,d} = \frac{tf_{t,d}}{\sqrt{\sum_{t=1}^{n} tf_{t,d}^2}} * log\left(\frac{N}{DF_t}\right) \qquad (4)$$

It is seen from equation (4) that the calculations of TF-IDF are based on the importance of each word in each tweet, so accuracy here is not high because the deduced relation here depends only on the number

of word occurrence in each tweet. Next, we present an example that explains how the similarity among tweets is calculated by TF-IDF.

Example 1: On this example, we assume that there are two old tweets that we need to calculate cosine similarity value for each one of them with a new input tweet entered to the FSD system, as the calculated values are used to decide which one is the closest neighbor to it. The two old tweets are:

Old Tweet-1: "hate when some people can't let go of their phone for five minutes"
Old Tweet-2: "I hate my job but I love getting my check"

And the new tweet is: **New Tweet:** "I hate forget my phone at job office"

Step 1: Here we split each tweet into words and count the occurrences $tf_{t,1}$, $tf_{t,2}$, $tf_{t,1}$,of each tweet ***t*** of the three tweets, see the counted values for each word in Table (3)

Step 2: In this step, IDF is calculated for each word ***t*** by using equation (3), see the results in Table (4).

Step 3: is calculated in this step for each word in tweets, this value is calculated by using equation (4) of TF-IDF without the dominator, see the values in in Table (5).

In this example: in this step, the word phone occurs two times. The total number of tweets is 3, so IDF will equal log (3/2) =0.176 and so on.

Step 4: In this step, we calculate the length normalization of each tweet *t*, i.e. normalize the vector using dominator of equation (4), as a vector can be (length-) normalized by dividing each of its components by its length L_t. This is done for the three tweets as follows:

L_1 = sqrt [(0^2) + (0.477^2) + (0.477^2) + (0.477^2) + (0.477^2) + (0.477^2) + (0.477^2) + (0.477^2) + (0.477^2) + (0.176^2) + (0.477^2) + (0.477^2) + (0.477^2)] =1.589

L_2 = sqrt [(0.176^2) + (0.477^2) + (0.477^2) + (0.477^2) + (0.477^2)] = 0.970

L_3 = sqrt [(0.176^2) + (0.176^2) + (0.477^2) + (0.477^2) + (0.477^2)] =0.862

Step 5: After the last step. In this step, each tweet can be represented by a vector containing the TF-IDF values of its words. These vectors are used to calculate the cosine similarity among the tweets. Here we use equation (1) to compute cosine similarity values between each two tweets as follows: This method uses the dot product between each two vectors as follows:

cosSim(t_1,t_3) = [(0.176*0.176)/(1.518*0.862]=0.023

cosSim(t_2,t_3) = [(0.176*0.176)/(0.970*0.862)]=0.037

According to the similarity values above t_3 is closer tot_2. So we can conclude that new Tweet (t_3) is closer in similarity to the old Tweet (t_2); hence, we can put it in the same bucket witht_2. Any new tweet comes later will be processed in the same way.

Table 3: tf for Each Word

Word (t)	$tf_{t,1}$	$tf_{t,2}$	$tf_{t,3}$
hate	0	0	0
when	0.477	0	0
some	0.477	0	0
people	0.477	0	0
can't	0.477	0	0
let	0.477	0	0
go	0.477	0	0
of	0.477	0	0
their	0.477	0	0
phone	0.176	0	0.176
for	0.477	0	0
five	0.477	0	0
minutes	0.477	0	0
I	0	0	0
my	0	0	0
job	0	0.176	0.176
but	0	0.477	0
love	0	0.477	0
getting	0	0.477	0
check	0	0.477	0
forget	0	0	0.477
at	0	0	0.477
office	0	0	0.477

Table 4: IDF for Each Word

Word (t)	IDF_t
hate	0
when	0.477
some	0.477
people	0.477
can't	0.477
let	0.477
go	0.477
of	0.477
their	0.477
phone	0.176
for	0.477
five	0.477
minutes	0.477
I	0
my	0
job	0.176
but	0.477
love	0.477
getting	0.477
check	0.477
forget	0.477
at	0.477
office	0.477

Table 5: TF-IDF for Each Word

Word (t)	TF_{IDF}	TF_{IDF}	$TF - IDF_{t,3}$
hate	0	0	0
when	0.477	0	0
some	0.477	0	0
people	0.477	0	0
can't	0.477	0	0
let	0.477	0	0
go	0.477	0	0
of	0.477	0	0
their	0.477	0	0
phone	0.176	0	0.176
for	0.477	0	0
five	0.477	0	0
minutes	0.477	0	0
I	0	0	0
my	0	0	0
job	0	0.176	0.176
but	0	0.477	0
love	0	0.477	0
getting	0	0.477	0
check	0	0.477	0
forget	0	0	0.477
At	0	0	0.477
office	0	0	0.477

4.3 Modified TF-IDF Algorithm (mTF-IDF)

The mTF-IDF was first proposed as modification of TF-IDF to enhance its accuracy in [20]. The main idea in mTF-IDF algorithm is not only consider the number of occurrence of a word in a tweet, but also it takes in consideration the proportion of the total number of word occurrences in the collection's tweets to the total number of distinctive words in the same collection. Figure (5) shows how this is made in this algorithm.

$$mtf_{t,d} = tf_{t,d} \times \left(\frac{T_t}{T_c}\right) \qquad (5)$$

Where $tf_{t,d}$ is the standard formula we mentioned it in the previous section, T_c is total number of distinctive words in the collection, T_t is total number of word occurrences in all collection's tweets and it can be calculated as shown in equation (6).

$$T_t = \sum_{d=1}^{D} tf_{t,d} \qquad (6)$$

The modification made to the TF equation leads to another modification in the formula for TF-IDF leading to the new general modified formula of TF-IDF shown in equation (7) and its detailed formula

shown in equation (8). In equation (8), IDF is the same as defined earlier in equation (3), while the dominator denotes to normalization here, which is not Euclidean norm as it was for TF-IDF. The fraction $\frac{length_d}{T_c}$ denotes to the number of missing words if the documents relatively to the total number of terms in the collection. So that the length of the document here is considered as the number of the distinctive terms in the document, where $length_d$ is number of distinctive words in the tweets. In the nominator the fraction $\frac{T_t}{T_c}$ will grow up by the increment of word occurrence on the collection level. The next example shows how this algorithm works.

$$mTF - IDF_{t,d} = mTF_{t,d}.IDF_t \quad (7)$$

$$mTF - IDF_{t,d} = \frac{tf_{t,d} * \left(\frac{T_t}{T_c}\right)}{log\left[\left(\sum_{t=1}^{n} tf_{t,d}^2\right) * \left(\frac{length_d}{T_c}\right)\right]} * log\left(\frac{N}{DF_t}\right) \quad (8)$$

Example 2: In this example, we run the mTF-IDF using the same tweets used in Example (1) to show how it works and compare its results with TF-IDF.

Step 1: In this step, we apply equation (5) to get its values for each tweet as shown in Table 7. We have already values of tf in Table 3. We can easily see find that, for the tweets in this example, the distinct words are defined in the set: Dist-words = {hate, when, some, people, can't, let, go, of, their, phone, for, five, minutes, I, my, job, but, love, getting, check, forget, at, office}

Hence, we can conclude that the number of distinct words $T_c$in these tweets equal 23.
For the same tweets, the set that Total-Words include all words and is defined by:

Total-Words = {hate, when, some, people, can't, let, go, of, their, phone, for, five, minutes, I, hate, my, job, but, I, love, getting, my, check, I, hate, forget, my, phone, at, job, office}

Hence, the length of this set $T_t$can be calculated from equation (6) and equals 31.

Step 2: We calculate IDF value for each tweet; the results are identical to the values in Table (4).

Step 3: In this step, is calculated, it is defined by the numerator of the mTF-IDF shown in equation (8), it is calculated for each word in the tweets and its results are shown in Table (8).

Step 4: Here, the dominator of equation (8) is calculated for each tweet in order to normalize the calculated values. In this example $length_d$ for Tweet-1 equals 13, $length_d$for Tweet-2 equals 8 and $length_d$for Tweet-3 equal 8. Hence; Dominator values for each tweet can be calculated as follows:

Dominator for Tweet-1 = log [[(0^2) + (1.347^2) + (1.347^2) + (1.347^2) + (1.347^2) + (1.347^2) + (1.347^2) + (1.347 ^2) + (1.347^2) + (2.695^2) + (1.347^2) + (1.347 ^2) + (1.347 ^2)] * (13/23)] = 15.38

Dominator of Tweet-2 = log [[(2.695^2) + (1.347 ^2) + (1.347 ^2) + (1.347^2) + (1.347^2)] * (8/23)] = 5.05

Dominator of Tweet-3 = log [[(2.695^2) + (2.695^2) + (1.347 ^2) + (1.347 ^2) + (1.347^2)] * (8/23)] = 6.94

Step 5: Results from the previous two steps are used to calculate the values of mTF-IDF for each tweet using equation (8) in this step.

Table.7 Modified TF Values

Word(t)	$mtf_{t,1}$	$mtf_{t,2}$	$mtf_{t,3}$
hate	1.347	1.347	1.347
when	1.347	0	0
some	1.347	0	0
people	1.347	0	0
can't	1.347	0	0
let	1.347	0	0
go	1.347	0	0
of	1.347	0	0
their	1.347	0	0
phone	1.347	0	1.347
for	1.347	0	0
five	1.347	0	0
minutes	1.347	0	0
I	0	2.695	1.347
my	0	2.695	1.347
job	0	1.347	1.347
but	0	1.347	0
love	0	1.347	0
getting	0	1.347	0
check	0	1.347	0
forget	0	0	1.347
at	0	0	1.347
office	0	0	1.347

Table.8 mTF-IDF Values

Word(t)	$mtf_{t,1}$	$mtf_{t,2}$	$mtf_{t,3}$
hate	0	0	0
when	0.642	0	0
some	0.642	0	0
people	0.642	0	0
can't	0.642	0	0
let	0.642	0	0
go	0.642	0	0
of	0.642	0	0
their	0.642	0	0
phone	0.237	0	0.237
for	0.642	0	0
five	0.642	0	0
minutes	0.642	0	0
I	0	0	0
my	0	0	0
job	0	0.237	0.237
but	0	0.642	0
love	0	0.642	0
getting	0	0.642	0
check	0	0.642	0
forget	0	0	0.642
at	0	0	0.642
office	0	0	0.642

Table.9 mTF-IDF Values

Word(t)	$mTF_{t,1}$	$mTF_{t,2}$	$mTF_{t,3}$
hate	0.041	0	0
when	0.041	0	0
some	0.041	0	0
people	0.041	0	0
can't	0.041	0	0
let	0.041	0	0
go	0.041	0	0
of	0.041	0	0
their	0.041	0	0
phone	0.015	0	0.034
for	0.041	0	0
five	0.041	0	0
minutes	0.041	0	0
I	0	0	0
my	0	0	0
job	0	0.046	0.034
but	0	0.012	0
love	0	0.012	0
getting	0	0.012	0
check	0	0.012	0
forget	0	0	0.092
at	0	0	0.092
office	0	0	0.092

Step 6: Similar to example (1), for each tweet, the Cosine Similarity is calculated; where first get the length of each tweet as follows:

L1 = sqrt[(0^2) + (0.041^2) + (0.041^2) + (0.041^2) + (0.041^2) + (0.041^2) + (0.041^2) + (0.041^2) + (0.041^2) +(0.015^2) + (0.041^2) + (0.041^2) + (0.041^2)]=0.13

L2 = sqrt[(0.046^2) + (0.127^2) + (0.046^2) + (0.046^2) + (0.046^2)]=0.25

L3 = sqrt[(0.034^2) + (0.034^2) + (0.092^2) + (0.092^2) + (0.092^2)]=0.16

Using Cosine similarity, we get the following similarity values of Tweet-1, Tweet-2 with the input tweet Tweet-3:

cosSim(Tweet-1, Tweet-3) = [(0.015*0.034)/(0.13*0.16)] = 0.024

cosSim(Tweet-2, Tweet-3) = [(0.046*0.034)/(0.25*0.16)] = 0.107

According to the similarity values Tweet-2 is more close to Tweet-3 than Tweet-1.

5. Evaluation

In this section, to evaluate the modified FSD model presented in this paper, we first evaluate its accuracy theoretically by analyzing the results of Examples 1 and 2 mentioned earlier; then we provide an empirical evaluation through a set of experiments.

5.1 Theoretical Evaluation of Accuracy

In order to evaluate the accuracy our proposed implementation of FSD, we compare it with the original FSD implementation in [13]. Both implementations use the cosine similarity values to direct incoming tweets to the bucket which has the closest tweets, i.e. classifying tweets into the correct topic/story class; based on the TF-IDF in the original implementation and mTF-IDF in our modified implementation, which are based. So, for comparing the efficiency, we need to define metrics that indicates the classification correctness. We propose here Cosine Similarity Gap (CSGAP metric, which can be

defined for a certain algorithm *Alg* as:

$$CSGAP_{Alg}\,[(ENTITY_A, ENTITY_B), ENTITY_C] = Abs\,[CosSim\,(ENTITY_A, ENTITY_C) - CosSim\,(ENTITY_B, ENTITY_C)]$$

This metric requires measuring the cosine similarity values of both entity $ENTITY_A$ with entity $ENTITY_C$, and entity $ENTITY_B$ with entity $ENTITY_C$. Then the absolute difference between the two values is calculated. This metric can be used to compare the efficiency of similarity detection algorithms by computing this value for these algorithms. If both $ENTITY_A$ and $ENTITY_B$ belong to the same or overlapping class; then the cosine similarity values of $ENTITY_C$ with each of them are very close; hence, CSGAP becomes very small. On contrary, if both $ENTITY_A$ and $ENTITY_B$ belong to non-overlapping classes, it is expected that their cosine similarity values are not close to each other; i.e. the CSGAP value is high. Therefore, we conclude that the higher the CSGAP value of a certain algorithm, the better is its efficiency in classification.

In FSD system implemented in [13].TF-IDF is responsible for directing any tweet arrives to the system to the bucket of tweets which is the closest to it, i.e. it classifies the bucket holding the same topic or story. In order to enhance the accuracy of this model, we replaced TF-IDF in our implementation by mTF-IDF. The similarity values computed for both methods using the results of processing the tweets defined in examples 1 and 2 mentioned earlier, these values are shown in Table 10. It is clear that both algorithms indicate that tweets Tweet-1 is much closer to Tweet-3 than Tweet-2. To decide which algorithm is better, we compare here the CSGAP metric defined above of both algorithms as follows.
For TF-IDF, the gap in the cosine similarity values is given by:
$CSGAP_{TF\text{-}IDF}$ [(Tweet-1, Tweet-2), Tweet-3] = 0.037-0.023 = 0.014,
For mTF-IDF, the gap in the cosine similarity values is given by:
$CSGAP_{mTF\text{-}IDF}$ [(Tweet-1, Tweet-2), Tweet-3] = 0.107-.024 = 0.083,

It is clear that $CSGAP_{mTF\text{-}IDF}$ is much higher than $CSGAP_{TF\text{-}IDF}$. This gives the indication that classifying the tweets using mTF-IDF gives better and more accurate classification/detection of the topics/stories to which the incoming tweets belong.

Table 10 Comparison between old, new Cosine Similarity Values of both TF-IDF and mTF-IDF

Used Method	(Tweet1,Tweet3)	(Tweet2,Tweet3)
TF-IDF	0.023	0.037
mTF-IDF	0.024	0.107

5.1 Empirical Evaluation

In order to run experiments on the implementation of our FSD mode, we used Twitter4j to collect the tweets as it is an unofficial open source Java library [30], which provides a Java based module to easily access the "Twitter Streaming API" which is a web service tool to get the tweets submitted by people in real time provided by Twitter. It can be accessed in any programming language. To access Twitter API, we need to sign in for Twitter developer account and get the authentication details like Customer key, Customer Secret, Access Token, and Access Token Secret. Also, we used a dataset from twitter4j. This dataset includes tweets in different languages like English, Arabic, Chinese, etc. We used around 215 MB of tweets, i.e. Big Data. To analyze the tweets, we built our FSD model over Storm. Storm works in two modes; local and distributed. Once the tasks are tried and run in local mode and it works fine without any problems, the mode can be changed to be distributed with few changes in settings. Storm

runs through "storm-ui", which is a site we can access from the Web browser by navigating to http://{ui-host}:8080. This web site gives some measurements on the cluster and running topologies like process latency, capacity, execute latency. We used Storm with specific configurations like number of workers equal 1 as each worker process runs on a single JVM as we run only one Java process for detecting the first story. We used 13 bits as the number of bits that define the length of the hash value calculated for each tweet, where we used 36 buckets/hash tables. A threshold is used to compare the incoming tweet to a fixed number of most recently seen tweets; the value of this threshold value was calculated heuristically, we used 0.3 for threshold. We have run the one worker over a cluster of 2 machines; machine 1 runs the Nimbus, it has core i5, 8GB RAM, and Machine 2 that act acts as a supervisor and works through a virtual machine with 4 GB RAM and 4-core processor.

The following metrics were used to measure performance and accuracy of results. Ack is number of Tuples acknowledged by this Bolt. Process Latency is the average time it takes to Ack a Tuple after it is first received. Bolts that join, aggregate or batch may not Ack a tuple until a number of other Tuples have been received. Capacity means if this is around 1.0, the corresponding Bolt is running as fast as it can. Execution Latency is the average time a Tuple spends in the execute method [31].

We ran both the original FSD presented in [13] that uses TF-IDF and our FSD model that use mTF-IDF. Table 11 shows some real sample results that we got after submitting the topology to storm UI to both implementations and running them on a corpus retrieved on 21st March 2016 during the early stages of the USA elections. The table shows a comparison between similarities values and timings of some stored tweets in the corpus with the tweets detected as first stories by our modified implementations that use the mTF-IDF and the original implementation that use the TF-IDF .

As we see from the results, the use of mTF-IDF algorithm enabled the system to detect tweets in the case of T1, T2 and T4 in the same topic with earlier timings than those detected by using TF-IDF; i.e. they are more accurately detected as first stories. We note that in the case of tweets T3, T5 and T6, both systems detected the same tweets as first story earlier. Also, in the case of T2 and T4 the tweet identified by the original FSD is completely out of the topic, while the modified system identified an earlier tweet in the same topic of T2 and T4. From the above, we see that the use of mTF-IDF is promising as it gives better (or at least equal) accuracy than the traditional TF-IDF.

Tables 12, 13 show the results we got after submitting the topology to storm UI for both FSD with TF-IDF and FSD with mTF-IDF. We can see in Table 12 and 13 Complete Latency for spout and bolts, Capacity and Process Latency for bolts which we defined it in section 5. Window column refers to the past period of time for which the statistics apply.

Complete Latency for spout in mTF-IDF, see Table 12, is slightly higher than it in TF-IDF in Table 13. This is expected as the mTF-IDF calculation is a bit more complex than TF-IDF, but the increased time is very small and does not affect the performence that the time of the system to finish its work. We used apache-storm version 0.9.6; it enables to display a visual representation of the running topology and find the data bottleneck in the running Storm application by click *Show Visualization* button in the Storm UI, see the visual representation of the FSD topology in original and modified implementation of in Figures 9 and 10 respectively. Thicker lines between components denote larger data flows. A blue component represents the first component in the topology, such as the spout in Figure 9 and Figure 10. The color of the other topology components indicates whether the component is exceeding cluster capacity: red components denote a data bottleneck and green components indicate components operating within capacity. Each Bolt has a time with milliseconds.

Table 11 Comparison between Similarity result for old and new algorithm

	Input Tweet	FS Detected by (TF-IDF)	FS Detected by mTF-IDF
T1	@FoxNews @realDonaldTrump Trump is killing it at AIPAC!	TRUMP LIVE AT AIPAC NOW !!!	"Trump at AIPAC \"Israel is willing to deal\" #ccctrail16"
Time	Mar 21 22:51:29	Mar 21 22:50:54	**Mar 21 22:48:38**
S.Score	X	0.5546665680165584	0.3815019004347936
InTopic	X	Yes	Yes
T2	@HillaryClinton the United States props itself on human rights and yet turns around and supports Israel???? □□□What's wrong w/neutrality?	Nice riding, Sore def subsides. And it all turns to power. See you on the bike!	"Raul Castro Slams the United States: Our Stance on Human Rights Will Not Change https:\/\/t.co\/GkvaoU0IUo"
Time	Mar 21 22:53:42	Mar 21 22:49:25	**Mar 21 22:48:30**
S.Score	X	0. 11060296569693867	0. 30550904297883547
InTopic	X	No	Yes
T3	#KKK Leader Endorses This Democrat For President,-@HillaryClinton Media Is Silent https://t.co/ZcmQl2KdTa	KKK Leader Endorses This Democrat For President, Media Is Silent https:\/\/t.co\/E4nbPvi4Ni	KKK Leader Endorses This Democrat For President, Media Is Silent https:\/\/t.co\/E4nbPvi4Ni
Time	Mar 21 23:07:49	**Mar 21 23:01:49**	**Mar 21 23:01:49**
S.Score	X	0. 8070830995336103	0. 8070830995336106
InTopic	X	Yes	Yes
T4	RT @WayneDupreeShow: Trump: Obama's deal with Iran test firing missiles...we will, we will, we will https://t.co/xlb0YN5BOq https://t.co/D…	□□□□ we'd be all set	"\"We will, we will, I promise we will\" is a pretty unenthusiastic Trump pander. #AIPAC2016"
Time	Mar 21 22:47:20	Mar 21 22:46:59	**Mar 21 22:41:46**
S.Score	X	0. 24506804124088716	0. 48366796025169323
InTopic	X	No	Yes
T5	RT @EmekaGift: @leoraisrael,Free #BIAFRA: OPEN LETTER TO ISRAEL https://t.co/2qSiWu0nIo	RT #BIAFRA: OPEN LETTER TO ISRAEL https:\/\/t.co\/2qSiWu0nIo	RT #BIAFRA: OPEN LETTER TO ISRAEL https:\/\/t.co\/2qSiWu0nIo
Time	Mar 21 23:15:27	**Mar 21 23:09:00**	**Mar 21 23:09:00**
S.Score	X	0. 9999753614766369	0. 9999753614766371
InTopic	X	Yes	Yes
T6	RT DanScavino: .@realDonaldTrump speaking @AIPAC in Washington, D.C. #AIPAC2016 #StandWithIsrael #Trump2016 https://t.co/HQAHeDTKGc	RT Donald Trump just hit it out of the park at #AIPAC2016! #Trump2016 #StandWithIsrael	RT Donald Trump just hit it out of the park at #AIPAC2016! #Trump2016 #StandWithIsrael
Time	Mar 21 23:16:11	**Mar 21 23:20:58**	**Mar 21 23:20:58**
S.Score	X	0. 3904767051320428	0. 4073151510671118
InTopic	X	Yes	Yes

Table 12 Stats Results for topology with TF-IDF

Topology Stats			
Window	**Complete Latency**		
10m 0s	2027.356		
3h 0m 0s	1804.816		
1d 0m 0s	1045.652		
All time	1045.652		
Spouts(All Time)			
Id	**Complete Latency**		
spout0	1045.652		
Bolts(All Time)			
Id	**Capacity**	**Execute Latency**	**Process Latency**
b-0	0.017	0.100	0.972
b-1	0.245	117.022	128.868
b-2	0.002	0.091	5.725
b-3	0.013	0.139	263.373
b-4	0.023	1.028	1.617
b-5	0.013	6.435	6.850
b-6	0.014	0.219	2.448
b-7	0.008	0.206	0.164
b-8	0.342	175.18	193.298
b-9	0.000	0.172	0.129

Table 13 Stats Results for topology with mTF-IDF

Topology Stats			
Window	**Complete Latency**		
10m 0s	2441.961		
3h 0m 0s	2150.187		
1d 0m 0s	1150.956		
All time	1150.956		
Spouts(All Time)			
Id	**Complete Latency**		
spout0	1150.956		
Bolts(All Time)			
Id	**Capacity**	**Execute Latency**	**Process Latency**
b-0	0.005	0.131	6.013
b-1	0.008	0.229	0.171
b-2	0.250	135.952	150.580
b-3	0.021	0.157	1.087
b-4	0.000	0.422	0.363
b-5	0.209	183.335	204.229
b-6	0.004	2.240	266.606
b-7	0.025	13.500	14.487
b-8	0.017	1.857	1.822
b-9	0.010	0.265	2.761

Figure 11, 12 and 13 shows the difference between Bolts and Capacity, Execute Latency and Process Latency respectively for (TF-IDF) and (mTF-IDF). Figure 14 shows the difference between Topology state and Complete Latency for (TF-IDF) and (mTF-IDF).

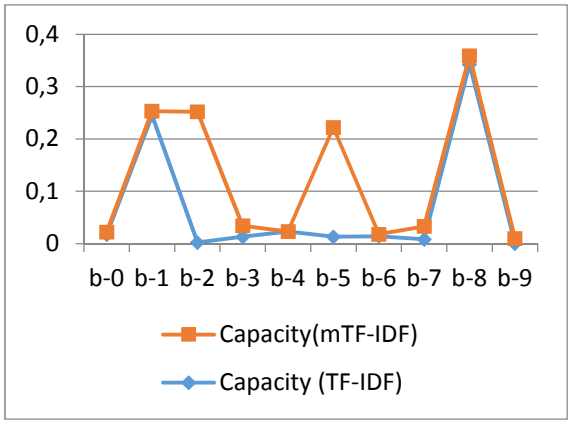


Figure 11 : Bolts and Capacity for both Algorithms

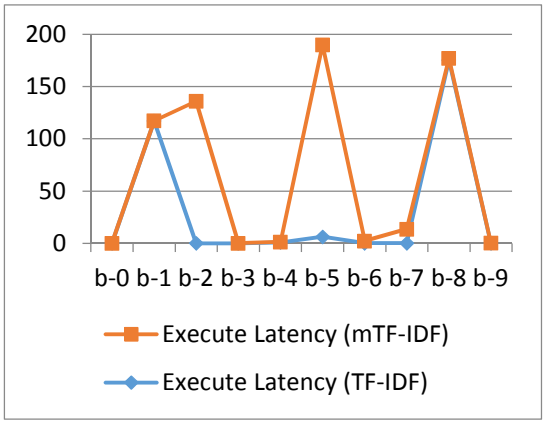
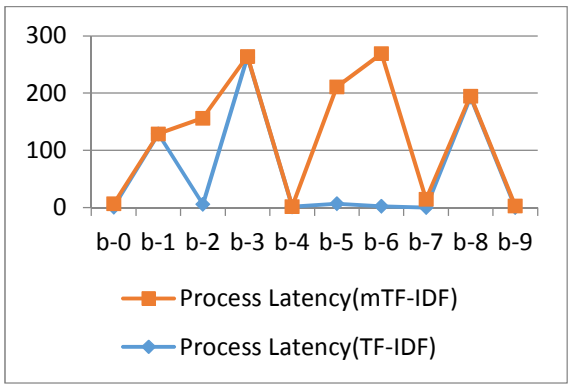


Figure 12 : Bolts and Execute Latency for both Algorithms



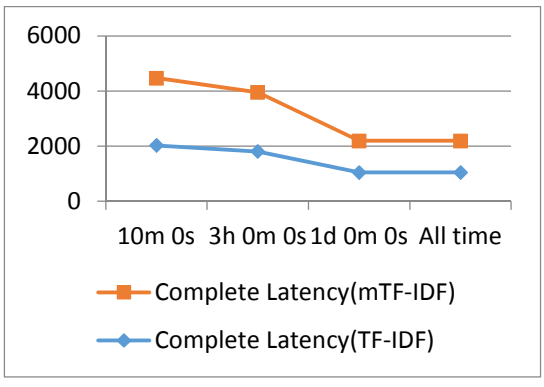


Figure 14: Topology state and Complete Latency

Figure 13 : Bolts and Process Latency

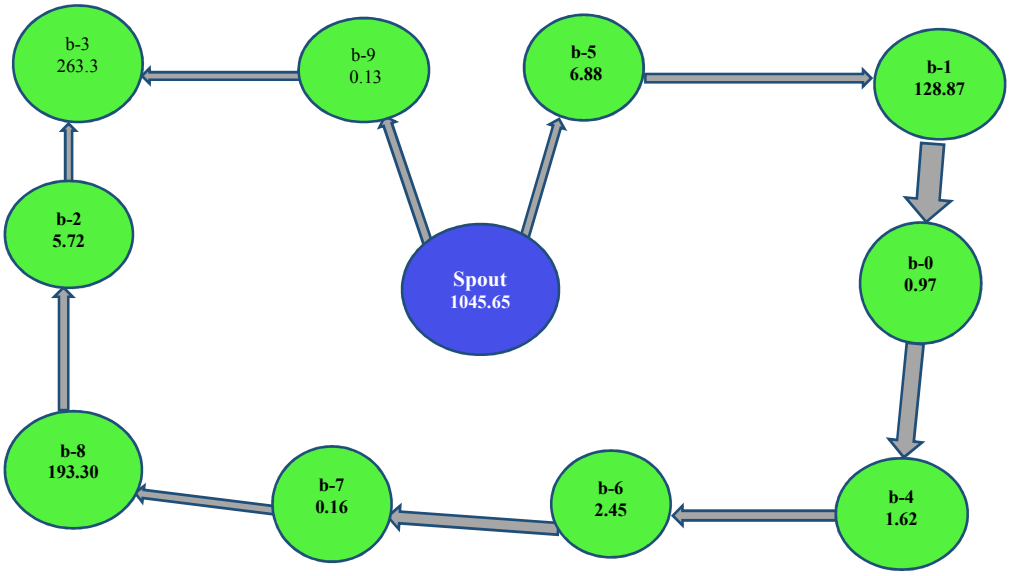


Figure 9 visualize Results for Topology with TF-IDF

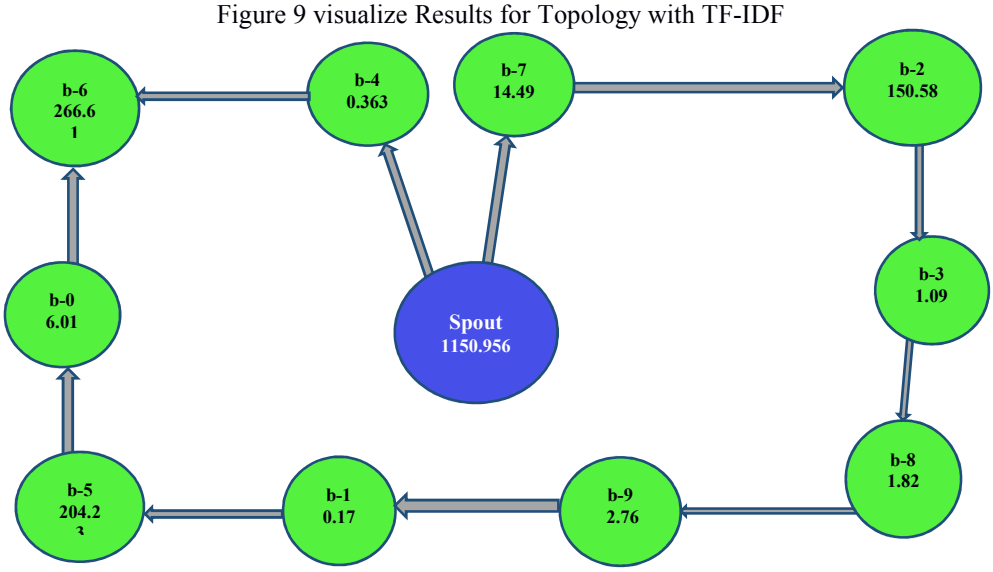


Figure 10 visualized Results for Topology with mTF-IDF

6. Conclusion

The process of analyzing Tweets to extract useful information in real time is a big data problem, due to the huge number of tweets produced continuously by the huge number of users. This process faces many challenges to get the required results with high accuracy in real time. FSD is one of these processes as it aims at detection of the tweet that included the first story in a certain topic in the tweets. FSD systems require distributed real-time platform to gain the benefits of high degree of parallelism and guarantee real-time execution. Storm is an open source for distributed real-time stream processing; hence, it provides a flexible, scalable platform to implement high performance FSD systems. Some existing FSD systems are built over Storm; most of these systems measure the similarities among tweets using the traditional TF-IDF; however this algorithm has its limitations. In order to enhance accuracy of such systems, we replace the TF-IDF by an alternative algorithm called mTF-IDF, which is an enhanced

version of the TF-IDF. Our empirical results show that mTF-IDF makes significance enhancements in the accuracy of detection results without affecting the performance noticeably. In the future, we intend to add enhancements to our system to enhance its accuracy by incorporating other algorithms for sub-text similarities in addition to exact text matching, Also, in addition to the unsupervised clustering method presented used in this paper, we aim to build a set of parallel supervised detectors followed by a final decision maker to enhance the overall performance. One possible extension of this work is using sub-word based distance measurement method instead of word based methods. After that, we plan to build a customized version of the system proposed here for the Arabic tweets in which the specific features of the Arabic language could be embedded in the pre and post processing.

References

1. Retrieved from https://about.twitter.com/company.
2. McCreadie, Richard, Craig Macdonald, Iadh Ounis, Miles Osborne, and Sasa Petrovic. "Scalable distributed event detection for twitter." In *Big Data, 2013 IEEE International Conference on*, pp. 543-549. IEEE, 2013.
3. Allan, James, ed. *Topic detection and tracking: event-based information organization*. Vol. 12. Springer Science & Business Media, 2012.
4. Fu, Xinyu, Eugene Ch'ng, Uwe Aickelin, and Lanyun Zhang. "An improved system for sentence-level novelty detection in textual streams." In *2015 International Conference on Smart and Sustainable City and Big Data (ICSSC)*, pp. 1-6. IET, 2015.
5. Luo, Gang, Chunqiang Tang, and Philip S. Yu. "Resource-adaptive real-time new event detection." In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pp. 497-508. ACM, 2007.
6. Allan, James, Victor Lavrenko, Daniella Malin, and Russell Swan. "Detections, bounds, and timelines: Umass and tdt-3." In *Proceedings of topic detection and tracking workshop*, pp. 167-174. 2000.
7. Yang, Yiming, Tom Pierce, and Jaime Carbonell. "A study of retrospective and on-line event detection." In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 28-36. ACM, 1998.
8. Indyk, Piotr, and Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality." In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604-613. ACM, 1998.
9. Petrović, Saša, Miles Osborne, and Victor Lavrenko. "Streaming first story detection with application to twitter." In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 181-189. Association for Computational Linguistics, 2010.
10. Benhardus, James, and Jugal Kalita. "Streaming trend detection in twitter." *International Journal of Web Based Communities* 9, no. 1 (2013): 122-139.
11. Huddar, Mahesh G., Manjula M. Ramannavar, and Nandini S. Sidnal. "Scalable distributed first story detection using storm for twitter data." In *Advances in Engineering and Technology Research (ICAETR), 2014 International Conference on*, pp. 1-5. IEEE, 2014.
12. Allan, James, Victor Lavrenko, and Hubert Jin. "First story detection in TDT is hard." In *Proceedings of the ninth international conference on Information and knowledge management*, pp. 374-381. ACM, 2000.
13. Michail Vogiatzis. (2012). Using Storm for Real-Time First Story Detection (Master's thesis,UniversityofEdinburgh). Retrieved from https://micvog.com/2013/09/08/storm-first-story-detection/#more-125

14. Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).
15. Moran, Sean, Richard McCreadie, Craig Macdonald, and Iadh Ounis. "Enhancing First Story Detection using Word Embeddings." (2016).
16. Godin, Fréderic, Baptist Vandersmissen, Wesley De Neve, and Rik Van de Walle. "Multimedia Lab@ ACL W-NUT NER Shared Task: Named Entity Recognition for Twitter Microposts using Distributed Word Representations." ACL-IJCNLP 2015 (2015): 146.
17. Kali, Margarita, François Rousseau, Alexandros Ntoulas, and Michalis Vazirgiannis. "Efficient online novelty detection in news streams." In International Conference on Web Information Systems Engineering, pp. 57-71. Springer Berlin Heidelberg, 2013.
18. Brigadir, Igor, Derek Greene, and Pádraig Cunningham. "Adaptive representations for tracking breaking news on twitter." arXiv preprint arXiv:1403.2923 (2014).
19. Vuurens, Jeroen BP, and Arjen P. de Vries. "First Story Detection using Multiple Nearest Neighbors." In Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval, pp. 845-848. ACM, 2016.
20. Sabbah, Thabit, and Ali Selamat. "Modified Frequency-Based Term Weighting Scheme for Accurate Dark Web Content Classification." In *Asia Information Retrieval Symposium*, pp. 184-196. Springer International Publishing, 2014.
21. Creutz, Mathias, Teemu Hirsimäki, Mikko Kurimo, Antti Puurula, Janne Pylkkönen, Vesa Siivola, Matti Varjokallio, Ebru Arisoy, Murat Saraçlar, and Andreas Stolcke. "Morph-based speech recognition and modeling of out-of-vocabulary words across languages." *ACM Transactions on Speech and Language Processing (TSLP)* 5, no. 1 (2007): 3.
22. Vogiatzis,Michael.(2013,Sep 8). How to spot first stories on Twitter using Storm [Web log post]. Retrieved Dec 12 , 2016, from https://micvog.com/2013/09/08/storm-first-story-detection/#more-125
23. Long, Shaohang, Ruonan Rao, Wangsheng Miao, and Xin Zhang. "An improved topology schedule algorithm for storm system." In *Computer Science and Applications: Proceedings of the 2014 Asia-Pacific Conference on Computer Science and Applications (CSAC 2014), Shanghai, China, 27-28 December 2014*, p. 187. CRC Press, 2015.
24. Shahrivari, Saeed. "Beyond batch processing: towards real-time and streaming big data." *Computers* 3, no. 4 (2014): 117-129.
25. Noll,Michael.G. (2013, May 5). Running a Multi-Node Storm Cluster. Retrieved from http://www.michael-noll.com/tutorials/running-multi-node-storm-cluster/ .
26. Neumeyer, Leonardo, Bruce Robbins, Anish Nair, and Anand Kesari. "S4: Distributed stream computing platform." In *2010 IEEE International Conference on Data Mining Workshops*, pp. 170-177. IEEE, 2010.
27. Distributed RPC. Retrieved from http://storm.apache.org/releases/current/Distributed-RPC.html.
28. Babcock, Brian, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. "Models and issues in data stream systems." In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 1-16. ACM, 2002.
29. Ramos, Juan. "Using TF-IDF to determine word relevance in document queries." In *Proceedings of the first instructional conference on machine learning*. 2003.
30. TWITTER4J. Retrieved from http://twitter4j.org/en/.
31. Perera, Malinga.Romesh. (2016, Apr 8). Reading and Understanding the Storm UI [Storm UI explained. Retrieved Dec 12, 2016, from http://www.malinga.me/reading-and-understanding-the-storm-ui-storm-ui-explained/ .