Expert Systems With Applications 96 (2018) 1–13

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

I-TWEC: Interactive clustering tool for Twitter

İnanç Arın*, Mert Kemal Erpam, Yücel Saygın

Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 15 May 2017
Revised 28 November 2017
Accepted 29 November 2017
Available online 1 December 2017

Keywords:
Tweet clustering
Short text clustering
Suffix tree based clustering
LCS based clustering

ABSTRACT

Social media provides a medium for people to express themselves on different issues. Twitter has gained a lot of popularity in the past decade as a social media platform where people create micro-blogs providing a valuable data source to understand trends and public opinion. However, the volume of tweets even on specific topics may reach millions creating a big challenge for the analyst. Clustering is a technique which can be utilized to better understand such large volumes of data. The main idea of clustering is to group similar tweets into batches in order to find patterns, to summarize, and to compress a large dataset. Though clustering is a natural technique for the analysis of tweets, there is no clustering tool specifically designed for Twitter data that utilizes lexical and semantic similarities; and that can be readily used by non-technical experts such as social scientists. *I-TWEC* is a web based tweet clustering tool where users can upload their data and the resulting clusters are presented with different visualizations which further enable the user to interactively select and merge clusters based on their semantic similarity. *I-TWEC* has the lexical and semantic clustering components implemented as two consecutive phases. For the lexical clustering of tweets, Longest Common Subsequence is a widely accepted similarity metric, however it is also very costly, and therefore not applicable to large data sets such as the ones collected through Twitter. In order to overcome that challenge, we have implemented a suffix tree based index structure in *I-TWEC* to efficiently cluster tweets based on the Longest Common Substring similarity which is an approximation of the Longest Common Subsequence. Experiments we have conducted show that lexical clustering phase of *I-TWEC* can produce results with comparable clustering quality in a fraction of the time required by the baseline methods which use Longest Common Subsequence and Suffix Tree. We have also experimented with a k-means document clustering as well as a state-of-the-art word-based suffix tree clustering algorithm and the results show that I-TWEC outperforms the state-of-the-art in terms of time with comparable clustering quality.

© 2017 Elsevier Ltd. All rights reserved.

1. Introduction

Twitter was founded in 2006 and its creator Jack Dorsey sent the very first tweet on March 21st 2006 saying *"just setting up my twttr"*. After that day, Twitter continued to be used in an incredibly increasing manner. Three years after the first tweet was sent, Twitter reached one billionth tweet.[1] As of 2015, average number of tweets per second reached 6000 (Sayce, 2016), which means approximately 500 millions tweets are generated in a single day.

There are two different account types in Twitter: *public*, and *protected*. A user can follow any user with a public account without any permission needed which allows users to follow and share all the tweets from these accounts. Permission of the owner is needed in order to follow a protected account. However, only 11.84% of the accounts are protected on Twitter[2] which means that most of the tweets are visible and shareable. In fact, people are constantly sharing their feelings, opinions, and reactions towards events and life in general on Twitter, hence it may be considered as the digital reflection of our society. Therefore, analyzing tweets has a lot of benefits such as giving significant insights about society for researchers in social sciences and other related fields.

Tweets generated by people contain a variety of information. There are distinct tweets, while many other tweets are duplicates and therefore, they do not contribute much information in terms of content analysis. One way of eliminating duplicate or similar tweets and reducing the number of tweets for analysis is lexical clustering where textual data items are grouped based on a lex-

* Corresponding author.
E-mail addresses: inanc@sabanciuniv.edu (İ. Arın), merterpam@sabanciuniv.edu (M.K. Erpam), ysaygin@sabanciuniv.edu (Y. Saygın).

[1] Twitter. #numbers. (2011). https://blog.twitter.com/2011/numbers Accessed 2017.01.18.

[2] Beevolve. An Exhaustive Study of Twitter Users Across the World. (2012) http://www.beevolve.com/twitter-statistics/ Accessed 2017.01.18.

https://doi.org/10.1016/j.eswa.2017.11.055
0957-4174/© 2017 Elsevier Ltd. All rights reserved.

ical similarity metric. By clustering tweets, it is possible to obtain a cleaner dataset containing only singular tweets representing a group with similar content, which reduces the time for more complex data processing algorithms. However, standard document clustering algorithms cannot be directly applied to tweets, because they have two distinct characteristics which differentiate them from other documents such as blogs, and news: (1) Tweets are very short, therefore standard document clustering algorithms which use word-based similarity metrics will not work well for tweets; (2) Twitter has no writing format, and people can use informal language, emoticons, abbreviations, and misspellings. As a result, Twitter needs a specific clustering methodology based on lexical clustering to identify similar tweets in terms of content. Following the lexical clustering, more complex semantic clustering techniques can be employed as a further post-processing step. To the best of our knowledge, there is no clustering tool designed specifically for Twitter data which utilizes lexical and/or semantic similarity.

In this paper, we propose two tweet clustering techniques and describe an open source clustering tool we developed. The first clustering technique is lexical threshold based clustering using Longest Common Subsequence (LCS) similarity metric, which we call *LCS-Lex*. We consider *LCS-Lex* as a benchmark since we observe that *LCS-Lex* clustering is quite effective in constructing high quality clusters. However the high computational complexity of *LCS-Lex* makes it ineffective to deal with a large number of tweets. The second technique we propose (*ST-TWEC*) is based on suffix trees. We implemented a customizable and extendable *Interactive Tweet Clustering Tool (I-TWEC)* which can be used by researchers from social sciences as well as researchers from more technical areas. For this tool, we implemented the *ST-TWEC* clustering method with a heuristic function for optimization and merging of clusters for Twitter domain. We also utilize word embeddings as a semantic tool for guiding users in interactive clustering. Basic word embedding methods adapt well to short phrases as stated in Mikolov, Chen, Corrado, and Dean (2013) and this makes word embeddings compatible with our suffix tree clustering algorithm. *I-TWEC* exploits the semantic information provided by word embeddings as well as efficient lexical grouping of tweets by a suffix tree. In order to show the benefits of suffix tree based clustering algorithm, we have experimented with Twitter data collected on different topics. We compared these two algorithms and results show that *ST-TWEC* performs well in terms of time complexity. Additionally, we conducted further experiments with two state-of-the-art methods: (1) New suffix tree document clustering algorithm (NSTC) developed by Chim and Deng (2007), and (2) A k-means based document clustering algorithm. *I-TWEC* outperforms both methods in terms of time performance with comparable or superior cluster qualities.

I-TWEC has been developed and shared on our GitHub page[3] with the research community. Additionally, we deployed our tool to a web server[4] (note that this server is not the same machine where we run all the experiments in Section 6). We also provide a short video tutorial about using *I-TWEC* on YouTube[5] to demonstrate the visual and interactive components of our system.

The rest of the paper is organized as follows. We first discuss the related work for short text clustering in Section 2. Background information about traditional clustering algorithms and Longest Common Subsequence (LCS) are provided in Section 3. In Section 4, *LCS-Lex, ST-TWEC*, and *Interactive Merging* methodologies are explained in detail. We provide the interactive system design of *I-TWEC* in Section 5 and experimental results are presented in Section 6. Finally, Section 7 concludes our work with some future research directions.

2. Related work

There is existing work on clustering documents and analyzing the data collected from social networking platforms. However, most of these works use vector space model to represent textual documents which is then used for similarity calculation. For example, Ma, Wang, and Jin (2014) propose a topic based document clustering technique with three phases where conventional techniques are used in each phase which are LDA, k-means++, and k-means respectively. Similarly, Jun, Park, and Jang (2014) propose a model that converts the text data into vector space model. Their model works on this sparse data structure, reducing the number of dimensions and then performing the clustering task. The clustering method they use is k-means based on support vector clustering and the Silhouette measure. Rangrej, Kulkarni, and Tendulkar (2011) convert text documents into vector space format with tf-idf values and then use k-means clustering with cosine and jaccard distances in order to group short text documents. Tu and Ding (2012) and Li, Sun, and Datta (2012) represent tweets and event segments respectively with tf-idf weights and then use cosine similarity metric to calculate the distance between tweets. Tang, Xia, Wang, Lau, and Zheng (2014) represent tweets as word vectors but they enrich these vectors with Wikipedia concepts. They focus on tweet representation, which maps each tweet to a space of Wikipedia concepts. Similar to tf-idf values, they count cf-itf (concept frequency and inverse tweet frequency) to fill vector representations. Becker, Naaman, and Gravano (2011) focus on online identification of real-world events from Twitter and use an incremental clustering algorithm where the number of clusters is not pre-determined. They also represent tweets with tf-idf vectors and use cosine similarity approach. In this work, our aim is to group tweets which are very similar in content with small additions, deletions, and updates. Therefore, we do not convert tweets into vector representations, instead we utilize *longest common subsequence* and *longest common substring* methods to identify similar tweets.

Other related work assigns documents (or tweets) into a set of pre-defined categories. For instance, Miller, Dickinson, Deitrick, Hu, and Wang (2014) consider two categories: *spam* and *not spam* and assign each tweet to one of these two categories. Nishida, Banno, Fujimura, and Hoshide (2011) propose a new method for classifying an unseen tweet as being related to an interesting topic or not. Zubiaga, Spina, Martínez, and Fresno (2015) categorize tweets into 4 different classes that are news, ongoing events, memes, or commemoratives. Saraçoğlu, Tutuncu, and Allahverdi (2008) developed a tool for clustering documents; however, their task is to determine the documents which belong to more than one class using fuzzy clustering. In our work, we do not have a predefined set of categories.

It is worth mentioning related work on clustering long-text documents such as news articles. Among those, Song, Qiao, Park, and Qian (2015) propose a hybrid evolutionary computation approach to optimize text clustering. Their approach takes advantage of quantum-behaved particle swarm optimization (QPSO) and genetic algorithm (GA). Their experiments are conducted on 4 subsets of standard Reuter-21578 and 20Newsgroup datasets which are quite different than Twitter data. Zamora, Mendoza, and Allende (2016) propose an efficient document clustering method based on locality-sensitive hashing (LSH), but their experiments were only based on formal language, long texts like 20Newsgroup and DOE (Department of Energy) datasets which contain abstracts about energy documents. The methodol-

[3] https://github.com/merterpam/I-TWEC.
[4] http://sky.sabanciuniv.edu:8080/I-TWEC/.
[5] https://youtu.be/_QWP5t5zPGw.

ogy used in long-text clustering is different than tweet clustering which considers short text containing informal language.

There are some studies on clustering in social media platforms. For instance, Dominguez, Redondo, Vilas, and Khalifa (2017) propose a method for clustering geolocated data from Instagram for outlier detection. However, their focus is not textual data. Martinez-Romo and Araujo (2013) worked on Twitter text data to detect malicious tweets in trending topics. They split the data into two groups (spam and not spam) as in text categorization, and then predict whether the tweets are spam using statistical language analysis. Cheong and Lee (2010) studied patterns in Twitter, however their work is mainly based on clustering users who exhibit some patterns and they only use data sets of size 13K tweets in their experiments.

We propose ST-TWEC for lexical clustering and the underlying data structure of this method is suffix tree. In literature, there is existing work based on suffix trees for document clustering. STC is a popular algorithm (Zamir & Etzioni, 1999) which uses word-based suffix tree for clustering. Chim and Deng (2007) improved STC by developing a new suffix tree document clustering algorithm (NSTC) that employs a novel suffix tree similarity measure with Group-average Agglomerative Hierarchical Clustering (GAHC) technique. It is important to stress out differences between ST-TWEC and STC as most state-of-the-art suffix tree clustering algorithms are based on STC. STC uses a word-based suffix tree to create clusters and then merges clusters based on the overlap of their document sets. To achieve linearity, STC can only merge k clusters with other clusters, hence it returns only top-k clusters. On the other hand, ST-TWEC uses a character-based suffix tree and achieves linearity for datasets of fixed size documents such as tweets. ST-TWEC is able to return all clusters and it is also able to capture character variations when comparing two tweets.

In Twitter domain, currently there are three papers which use suffix trees for clustering. Thaiprayoon, Kongthon, Palingoon, and Haruechaiyasak (2012) use Carrot2 framework for its generalized suffix tree implementation.[6] They generate initial cluster labels based on frequently occurring substrings in a set of tweets using the generalized suffix tree, and then create a two level hierarchy of cluster labels. During the construction of first level, a label is discarded if it is a substring of another label, and the second level is formed by including those labels overlapping with a label at the first level. Poomagal, Visalakshi, and Hamsapriya (2015) use STC along with semantic similarity to cluster tweets and determine topics of interest. On the other hand, Fang, Zhang, Ye, and Li (2014) use suffix tree to detect the common phrases between tweets and use them as features to detect popular events. Although these methods use suffix tree to employ different clustering techniques, the main limitation of these methods is that they return top-k clusters/events, discarding the rest. Atefeh and Khreich (2015) compare event detection methods for Twitter. Authors explain both event detection methods in Twitter and in traditional media. One of the event detection methods explained in traditional media uses an n-gram approach for event detection in news and uses suffix tree to speed up the retrieval of n-gram words, however clustering was not considered.

One of our contributions is an interactive tool (*I-TWEC*), therefore it is also worth mentioning existing clustering tools. Bozkir and Sezer (2013) developed a desktop software, called FUAT, to analyze, explore and visualize aspects of clusters created by using fuzzy c-means algorithm. Argyrou and Andreev (2011) designed a semi-supervised tool which clusters accounting databases. Both of these tools are quite useful; however, none of them works on textual data. Our clustering tool has been designed to find out similar tweets from Twitter in an efficient way and helps analysts in quick assessment of the general trends in large collections of tweets.

[6] http://project.carrot2.org/index.html.

3. Preliminaries and background

Document clustering is the task of grouping documents based on a similarity measure. There are traditional clustering algorithms such as K-means, DBScan, and hierarchical clustering. These traditional clustering algorithms do not work well in large datasets which contain unknown number of clusters such as the case in Twitter where users may talk about similar topics, give similar responses or retweet each other. Therefore, clustering algorithms and tools tailored for Twitter are needed. In general, clustering algorithms can be categorized in many aspects such as methodology, complexity or cluster definition. In our work, we differentiate clustering algorithms based on the similarity metric they use and roughly divide them in two categories: (1) Clustering based on lexical similarity, and (2) Clustering based on semantic similarity. Calculation of semantic similarity between documents is usually a non-linear operation, which makes semantic clustering unsuitable for large amounts of data.

Lexical similarity between tweets is generally calculated by Named Entity Recognition (NER) based approaches as in Derczynski et al. (2015), Jung (2012), and Liu and Zhou (2013). Another way of calculating lexical similarity is by looking at the textual representation of tweets and finding the Longest Common Subsequence (LCS) between tweets. LCS is a long studied computer science problem. Given a set of sequences, LCS is the longest subsequence which exists in all the given sequences. It has been used before as a similarity measure for documents by Islam and Inkpen (2008) and Banerjee and Ghosh (2001).

Finding LCS of an arbitrary number of sequences is an NP-hard problem. However, LCS of two sequences can be found in $O(m^*n)$ time and space using dynamic programming where m and n are the length of the input sequences. The problem of finding the length of LCS can be divided into overlapping sub-problems. Let X and Y be two strings and $X_i = x_1x_2 \dots x_i$ be the prefix of X until ith character and $Y_j = y_1y_2 \dots y_j$ be the prefix of Y until jth character, then the length of the LCS for X and Y can be obtained using Eq. (1):

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

LCS is a significant measure for the detection of lexical similarity, however its non-linear complexity makes it unsuitable for large datasets such as Twitter. In our work, we use an approximation for LCS that uses the common substring notion instead of the subsequence between tweets as the similarity measure which enables us to develop a linear-time clustering algorithm through a suffix tree.

Suffix tree is a data structure which represents the suffixes of a given string. In a suffix tree, each edge represents a substring of the string and each path from the root to a leaf represents one suffix of the given string. The construction of a suffix tree has linear time complexity and the suffix tree itself has linear space complexity, enabling linear time and space string operations such as pattern, regular expression matching and finding the longest common substring.

4. Methodology

We propose two different methods for tweet clustering in this work. In both methods, our aim is to create a set of clusters $C =$

$\{c_1, c_2, \ldots, c_m\}$ for a given set of tweets $T = \{t_1, t_2, \ldots, t_n\}$ where $m \leq n$. Proposed methods are explained in the following subsections.

4.1. LCS-Lex: Longest Common Subsequence based lexical clustering of tweets

LCS-Lex is based on Longest Common Subsequence (LCS), and we use the normalized LCS score defined in Eq. (2) for similarity calculation between two tweets.

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

A naive algorithm for LCS based tweet clustering process is given in Algorithm 1. In this algorithm, for each unclustered tweet

Algorithm 1: LCS based tweet clustering algorithm.

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

(referred to as tweet i), we go through all remaining unclustered tweets (referred to as tweet j). If the normalized score defined above, between tweet i and tweet j is higher than the threshold, then we include indexes of these tweets (which are i and j) in the same cluster c. However, not all clusters are included into the set of clusters C; instead only the clusters whose sizes are greater than or equal to k are included into C where k is defined as the minimum number of tweets a cluster must have. The very first tweet (t_i) included in c becomes the representative tweet of cluster c. Once we include the cluster c into the set C, then we remove the tweets that are clustered in c from the dataset. This algorithm ensures that the cluster sizes are greater than or equal to k.

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

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

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

There are two different parameters (*threshold* and k) in *LCS-Lex* to be defined by the user. In this work, we assume that at least 2 tweets are required to compose a cluster. In other words, we define a cluster as a group of two or more tweets. For that reason, we selected k as 2 in our experiments in Section 6. For *threshold* parameter, we did not define any specific threshold, instead we tested the performance (both time and cluster quality) of *LCS-Lex* with different *threshold* values.

Although LCS based clustering algorithm (*LCS-Lex*) performs well in terms of cluster qualities, the time performance is prohibitive for large number of tweets (even for tens of thousands). The complexity of Algorithm 1 is $O(N^2 * L^2)$ where N is the number of tweets ($|T|$), and L is the maximum tweet length which is 140 due to the characteristics of Twitter.

4.2. ST-TWEC: Suffix tree based tweet clustering method

LCS-Lex is not a scalable algorithm. Therefore, we designed an alternative suffix tree based algorithm which we call *ST-TWEC*. We implemented *ST-TWEC* in a tool which we call *I-TWEC*, together with the semantic representations of tweets for clustering. We define a tweet, t, as a sequence of characters that someone wrote where $|t|$ is the length of t. Given two tweets t_i and t_j, we define a common substring of t_i and t_j as a string which occurs in both tweets.

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

With *ST-TWEC*, we would like to create a set of clusters $C = \{c_1, c_2, \ldots, c_m\}$ such that the length of the common substring of tweets inside a cluster is above a threshold. We define the common substring of all tweets in the cluster as the cluster label and this label represents the cluster. *ST-TWEC* has a linear space and time complexity for data points with fixed maximum length which suits well for Twitter.

As the first step of *ST-TWEC*, we preprocess tweets to standardize and remove components with no clear semantic context such as links, usernames, retweet tags and punctuation marks. We also transform all tweets to lower case and adjust white spaces. After the preprocessing phase, we remove tweets which contain less than 5 characters. Then, we construct a generalized suffix tree to detect the common substrings between tweets. A generalized suffix tree for a given set of tweets is a data structure which stores the suffixes in an efficient manner where a path from root to an internal node represents a unique substring while a path from root to each leaf represents a unique suffix of a tweet (or tweets). In our algorithm, we use each node as a basis for clusters and we define a cluster corresponding to each node. Given a node n and its corresponding cluster c_n, we define a tweet to be a member of the cluster c_n, iff:

1. Tweet contains the string which node n represents (denoted as *n.string*)
2. *length*(n.string) / *length*(tweet) > **thrCluster**, where **thrCluster** is a user-defined threshold.

These cluster membership rules allow us to create clusters which share common substrings. With the second rule, we ensure that the ratio of the common substring's length and the length of tweets inside a cluster is above a threshold.

The nodes of a character-based suffix tree, especially the nodes which have parental relations, show similarity in terms of the substring they represent and tweets they contain. Because of that, clusters that are created based on the nodes of a character-based

suffix tree have a high overlapping ratio. We aim to reduce this overlapping ratio by introducing an overlapping detection method before the cluster creation phase. We use nodes to approximate the overlapping and examine nodes with parental relationships. We mark a child node as a duplicate node and merge it with its parent, if it satisfies any of the following conditions:

- The ratio of the number of tweets in an ancestral node to the number of tweets in its child node is below 1.2
- The ratio of the length of the substring of an ancestral node to the length of the substring of its child is above 0.8

The rationale for choosing those ratios will be explained in Section 6. In the suffix tree, an ancestor node also contains all of the tweets its children contain; therefore, when we compare the number of tweets a parent and a child node contains, implicitly we look at the overlap of the tweet sets of these nodes and mark the node with smaller tweet set as duplicate if the difference is small. Similarly, when we compare the substring length between nodes with parental relationships, we actually compare the common substring between nodes. In the Twitter domain, we observe that there are many tweets which are similar in content, but appear on different nodes in the suffix tree due to small variations. The second condition for overlapping elimination method tries to capture similar tweets into one cluster by allowing small variations. At the end of the cluster creation phase, if a tweet appears in more than one cluster, then it is allowed to stay in the biggest cluster and it is removed from the rest of the smaller clusters. This way we favor large clusters to form.

At the end of the lexical clustering, we assign labels to each cluster. Remember that each cluster has a corresponding node in the suffix tree and each node represents a substring. We use these substrings as labels for clusters. If the substring contains partial words at the beginning or at the end making it incomprehensible, we complete the string based on the tweets inside the cluster by finding the position of the string inside a tweet and extending the label from the beginning and the end until we encounter a space. A more detailed explanation of *ST-TWEC*'s lexical clustering can be found in Erpam (2017) which is technical report.

4.3. Interactive merging

After lexical clustering, we obtain clusters of tweets which have similar contents, however there may be tweets which convey the same meaning with different words. These tweets may be assigned to different clusters, because in *ST-TWEC*, tweets are clustered based on their string similarity, not considering their semantics. If these clusters are semantically similar, they need to be merged. Therefore, we introduce the interactive merging phase after *ST-TWEC* where users can merge clusters based on semantic relatedness which measures how related are a given set of words. An example for semantic relatedness could be **"bus", "road"**, and **"driver"**.

Word embeddings is a technique used in natural language processing where words or phrases are represented as a vector of real numbers. The vector representations obtained by word embeddings retain their semantic relatedness and it is even possible to perform arithmetic operations on them. Mikolov et al. (2013) gives a famous example where the arithmetic operation of vector(King) - vector(Man) + vector(Woman) gives a vector which is closest to the vector representation of Queen.

To compute the relatedness score between labels, we first calculate the mean of word embeddings for each label and use cosine similarity. Given two clusters c_i and c_j, we define m_i and m_j as the mean word embeddings of their respective labels and calculate the relatedness score as:

$$rScore(c_i, c_j) = cosinesimilarity(m_i, m_j) = \frac{m_i \cdot m_j}{||m_i||_2 \; ||m_j||_2} \quad (3)$$

For word embeddings, we use Google's pre-trained model[7] which was trained over 3 million words and phrases using the data obtained by Google News. The dimensionality of each vector in the pre-trained model is 300.

Depending on the content of cluster labels and relatedness score, the user decides to merge a cluster or not. The process of presenting clusters for merging continues until the user stops or there are no further clusters for merging.

5. Interactive system design

I-TWEC is a hybrid tool for clustering large amounts of tweets based on string similarity. With *I-TWEC*, users are also able to merge similar clusters based on semantic relatedness scores. For clustering, we implemented the algorithm explained in Section 4 which uses a generalized suffix tree as the base data structure and has a linear time complexity for data points with fixed maximum length. For interactive merging of clusters, we use word embeddings to calculate the semantic relatedness score between clusters and guide the user through the merging process.

Using suffix tree and word embeddings as the base methodologies, *I-TWEC* is a web-based interactive system designed for the end-user. At the server side, resource intensive operations such as suffix tree and cluster creation are made; while at the client side, a visualization of the clusters is presented to the end-user and the end-user is able make adjustments to the threshold based on the information provided. We use Java Servlets at the server side and JavaScript for the client side operations. The communication between server and client is made through Ajax queries.

5.1. Server side implementation

At the back-end of our tool, we make the computations necessary for cluster creation, labelling, refinement, and semantic relatedness calculations. The pipeline starts when the end-user uploads the Twitter data for clustering. In order to parse the data correctly, the uploaded file should have a specific format with one tweet per line. *I-TWEC* also allows the user to evaluate the formed clusters based on intra-cluster similarity and cluster purity. To evaluate clusters on cluster purity, each tweet requires a pre-defined label. The user can optionally upload tweet labels by appending them at the end of each tweet in the uploaded file. The tweet and label should be separated by a tab character and therefore, tweets should not contain any tab character.

After the upload, tweets are stored at an in-memory list and preprocessed. After the preprocessing, tweets which are below a character limit are removed from the list. With the preprocessed tweets, a suffix tree and then clusters based on this suffix tree are created. Fig. 1 illustrates the complete pipeline on the server side.

The threshold for the cluster creation phase is determined by end-user and it can be changed after clusters are created, triggering the reconstruction of clusters. Suffix tree is constructed once, and the user is able to change the threshold to obtain different clusters in which case we use the already constructed suffix tree, and re-execute the cluster creation phase and the phases following it. Fig. 2 illustrates the mechanics of the re-clustering phase in the pipeline.

After the cluster labelling phase, semantic relatedness score between clusters need to be calculated for interactive merging. This

[7] Google Code. word2vec. (2013). https://code.google.com/archive/p/word2vec/ Accessed 2016.11.22.

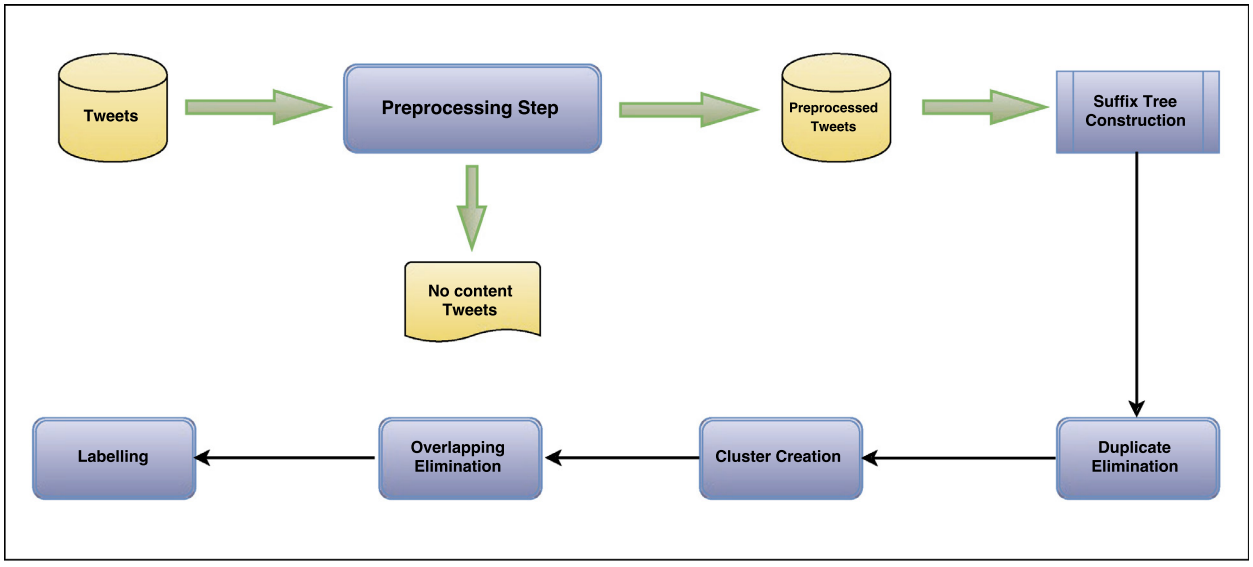

Fig. 1. Complete pipeline of clustering process.

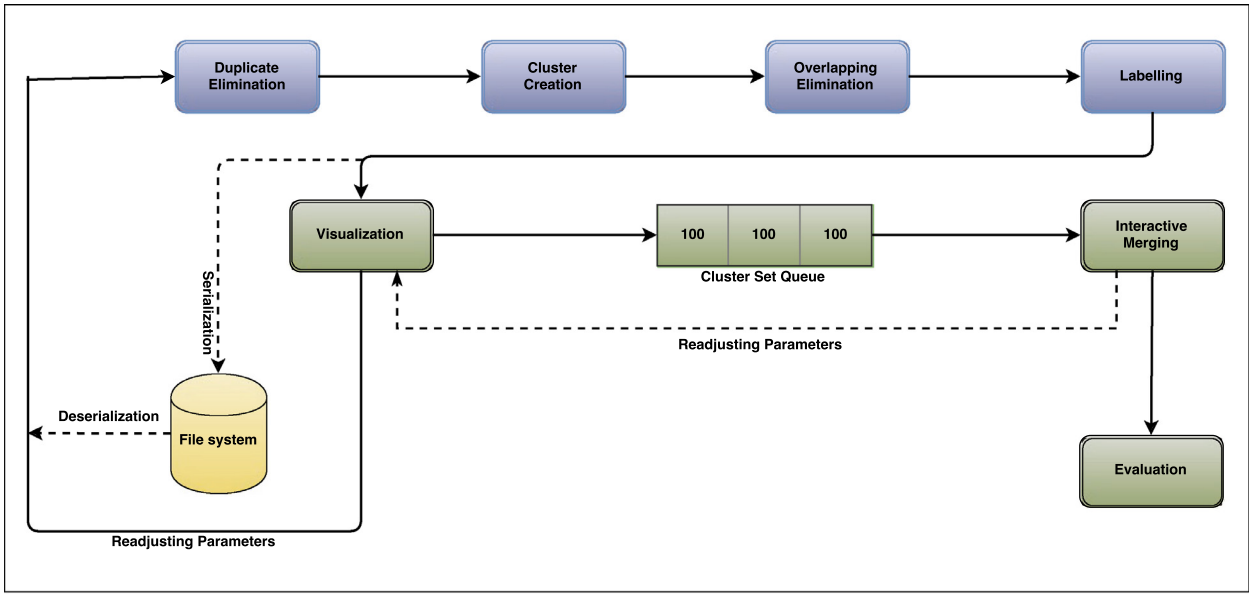

Fig. 2. Pipeline of visualization and re-clustering process.

calculation is also a resource intensive operation and it is done on the server side asynchronously. The result of the calculation is deserialized to the file system and serialized when the client side requests.

I-TWEC is a multi-user system. We isolate each operation inside the pipeline on user-level. When the end-user uploads the Twitter data, we create a unique session between the user and server. By using the session, we store the serialized suffix tree and semantic relatedness calculations in a directory unique to each user. When the session ends, we remove the serialized files unique to the user.

5.2. Client side implementation

The front-end of our tool has the functionality of uploading datasets for clustering, visualizing the clustering results, adjusting the clustering threshold, and merging clusters with the help of semantic relatedness. After uploading the data, the initial clustering is performed with a default threshold of 0.4 (we will explain why default threshold is 0.4 in Section 6) and then the user is able to interactively adjust the cluster threshold to obtain a better clustering scheme.

After clustering, the results are displayed by using a histogram and a bubble chart, as shown in Fig. 3. Each bar in the histogram represents a cluster and the histogram is sorted by cluster size. To obtain a better visualization, the histogram can be zoomed-in, zoomed-out and scrolled. In the bubble chart, each bubble represents a cluster and the size of each bubble is proportional to the size of a cluster. By hovering on a bubble, it is possible to obtain the label and size of the cluster it represents. In a cluster, tweets with the exact same content are grouped together and we sort the distinct tweets inside clusters by the size of these groups. When the user clicks a bubble in the bubble chart, the first 10 tweets of the cluster sorted by the size is displayed to the user.

By examining the resulting clusters, the end-user can adjust the clustering threshold, and re-cluster the dataset. This option is displayed on a dynamic dropdown menu and it invokes the re-clustering pipeline shown in Fig. 2. In the re-clustering pipeline,

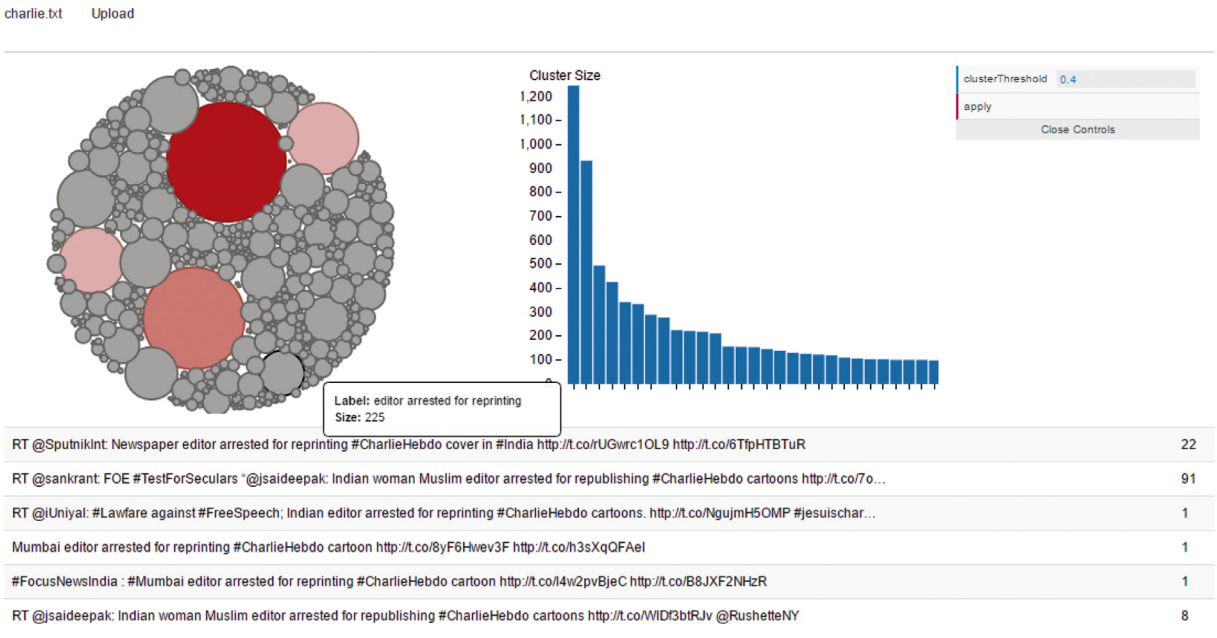

Fig. 3. Lexical clustering GUI.

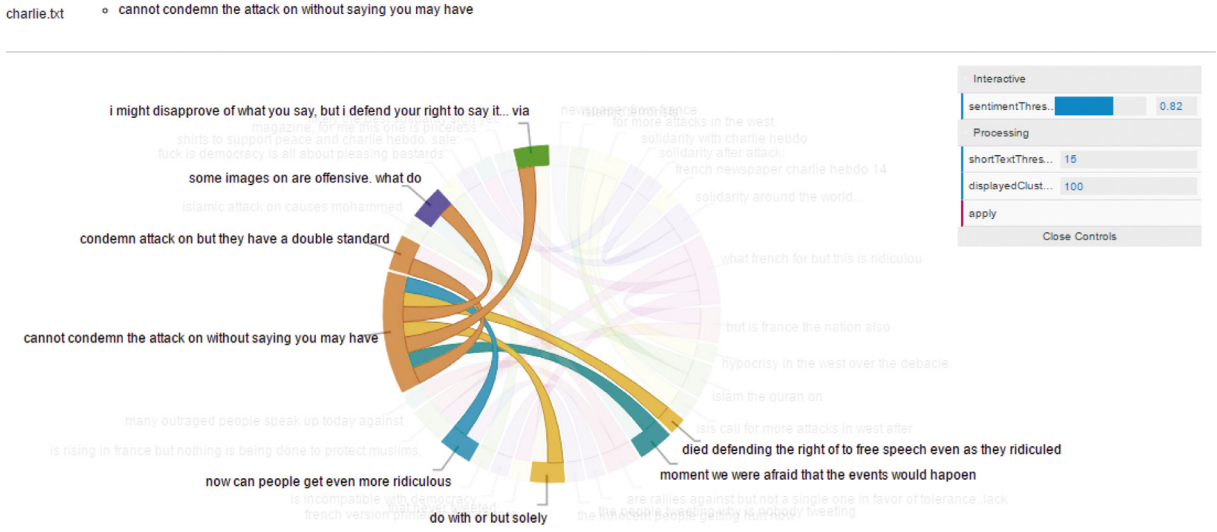

Fig. 4. Interactive merging GUI when merging a cluster.

the suffix tree is deserialized and new clusters are created with the new cluster threshold. Then, overlapping clusters are eliminated from new clusters and for the remaining clusters, their labels are assigned.

In the interactive merging part of *I-TWEC*, cluster labels are displayed on a wheel chart. Each label occupies a fixed space on the wheel chart and if the semantic relatedness score between two cluster labels is greater than a threshold, a path is drawn between labels. By selecting connected cluster labels, it is possible to merge them into one cluster. Fig. 4 illustrates the wheel chart and a merging example.

The threshold for semantic relatedness score can be changed dynamically on the drop down-menu. In addition to the threshold, parameters such as the length of the cluster labels and the maximum number of clusters shown on the wheel can be adjusted. Adjusting these two parameters require the reselection of clusters and recalculation of the semantic relatedness scores which are performed at the server-side.

6. Experimental evaluation

We evaluated both *LCS-Lex* and *ST-TWEC* using a combination of four datasets collected from Twitter Streaming API using hashtags. Our dataset consists of tweets related to Charlie Hebdo event (#jesuisCharlie), Christmas in 2016 (#christmas), NBA organisation (#nba), and US President Trump (#trump). We limited the number of tweets of each dataset to 15K. Thus we have a total of 60K tweets for 4 different domains and before starting evaluations we applied some preprocessing on tweets like eliminating hashtags and transforming letters into lowercase. We experimented with these 60K tweets in order to compare the two tweet clustering algorithms; *LCS-Lex*, and *ST-TWEC*, however we also performed scalability experiments for *ST-TWEC* with much higher number of tweets which are reported in this section.

Given the dataset $D = \{t_1, t_2, \ldots, t_n\}$, we create a set of clusters $C = \{c_1, c_2, \ldots, c_m\}$ such that every cluster c_i contains at least k tweets. In the experimental evaluation, we set k to 2 and mea-

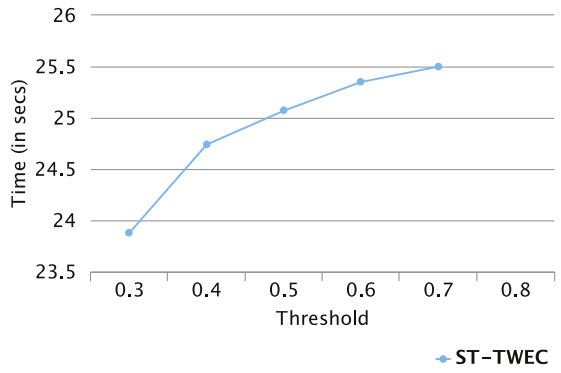

Fig. 5. Time performance of *ST-TWEC* for 60K tweets with different thresholds.

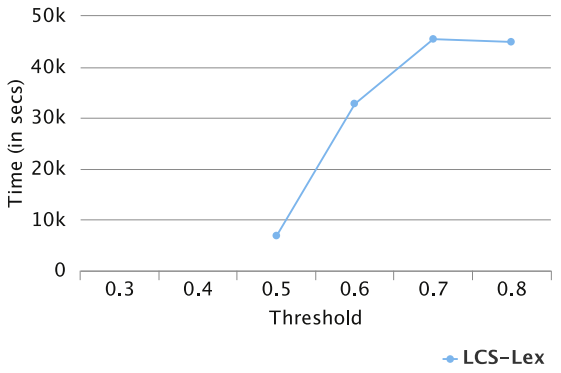

Fig. 6. Time performance of *LCS-Lex* for 60K tweets with different thresholds.

sure the intra-cluster similarity and cluster purity. Because we are focusing on the textual representations of tweets, we define the intra-cluster similarity measure based on LCS. Using Eq. (2), we calculate intra-cluster similarity of a cluster using the pairwise similarity of each tweet inside the cluster as:

$$intraCSim(c) = \frac{2}{|c| * (|c| - 1)} * \sum_{i=0}^{|c|} \sum_{j=i+1}^{|c|} NormalizedScore(t_i, t_j) \tag{4}$$

Eq. (4) allows us to find the intra-cluster similarity of a given cluster. Using the Eq. (4), we find the average intra-cluster similarity in Eq. (5) and the weighted average intra-cluster similarity of the cluster set in Eq. (6). We take the size of each cluster into consideration to calculate the weighted average intra-cluster similarity measure.

$$avgISim(C) = \frac{1}{m} * \sum_{i=0}^{m} intraCSim(c_i) \tag{5}$$

$$wAvgISim(C) = \frac{1}{\sum_{i=0}^{m} |c_i|} * \sum_{i=0}^{m} |c_i| * intraCSim(c_i) \tag{6}$$

Since we have collected four different datasets using four hashtags, we assume that the tweets have four possible categories corresponding to each of the four hashtags. Therefore we assigned a label to each tweet depending on its hashtag. In total, we have four labels: #christmas, #nba, #trump, and #jesuischarlie. We use these labels as a gold standard to calculate the purity of clusters with Eq. (7):

$$purity(C) = \frac{1}{\sum_{i=0}^{m} |c_i|} * \sum_{i=0}^{m} |max_{label} \ in \ c_i| \tag{7}$$

Experimental results with different similarity thresholds in terms of time performance, number of constructed clusters, number of unclustered tweets, average intra-cluster similarity, weighted average intra-cluster similarity, and purity are shown in Fig. 5 through Fig. 11, respectively.

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

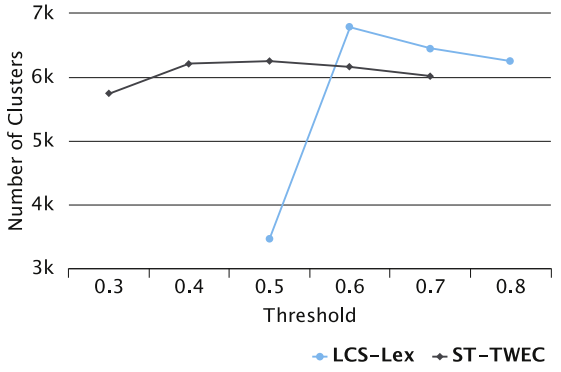

Fig. 7. Number of clusters for 60K tweets with different thresholds.

From the experiments we can easily observe that there is a dramatic time improvement with *ST-TWEC*. The clustering time with *ST-TWEC* ranges from 23.8 to 25.5 s in Fig. 5. On the other hand, the clustering process with *LCS-Lex* takes form 6825 to 45,472 s as can be seen in Fig. 6 (all the experiments were tested on a system with 32 processor, model name *Intel(R) Xeon(R) CPU E5-2690 0 @ 2.90 GHz*, and 128GB RAM). It is worth noting that *ST-TWEC* is able to cluster 1 million tweets in about 1500 s. Later in this section, we will mention about the time performance of *ST-TWEC* with increasing dataset sizes with a threshold of 0.4 in Fig. 22. We need to look at average intra-cluster similarity values to understand why we selected 0.4 as threshold in ST-TWEC. The average intra-cluster similarity values are 0.79, 0.84, 0.87, 0.89, 0.91 for thresholds 0.3, 0.4, 0.5, 0.6, and 0.7 respectively as it can be seen in Fig. 9. Although greater the threshold value means greater the average intra-cluster similarity, changing threshold from 0.3 to 0.4 made the biggest improvement. Note that increasing threshold value also increases number of unclustered tweets as it can be observed in Fig. 8, and we want to keep this number as low as possible. For that reason, we have chosen 0.4 as a reasonable threshold value for our experiments.

Other results given in Figs. 7–11 vary for different thresholds, however we can observe that the differences in terms of clustering quality and the number of clusters produced are very low. In other words, similar cluster qualities in terms of the number of clusters, the average intra-cluster similarity, and the purity can be obtained by *ST-TWEC*. However, clustering time is the major factor that distinguishes these methods which proves the effectiveness of *ST-TWEC* for large datasets. In order to show that the clusters composed by *ST-TWEC* and *LCS-LEX* are not significantly different, we have selected 100 random subsets of size 5K tweets from our original dataset of 60K tweets. We ran *ST-TWEC* and *LCS-LEX* on each subset and recorded the corresponding cluster qualities. For the statistical significance tests, the null hypothesis states that

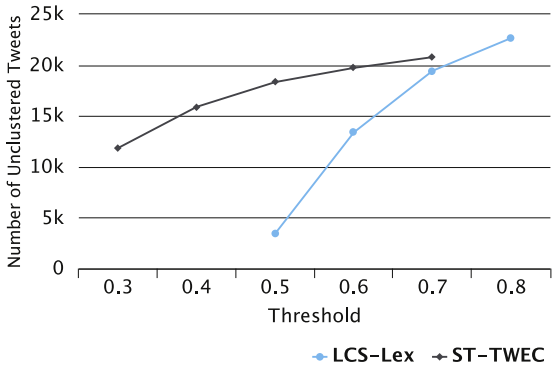

Fig. 8. Number of unclustered tweets for 60K tweets with different thresholds.

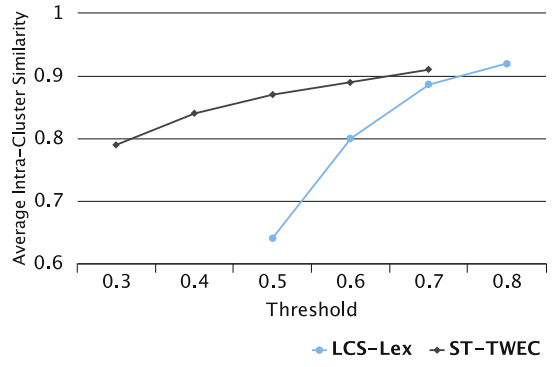

Fig. 9. Average intra-cluster similarity for 60K tweets with different thresholds.

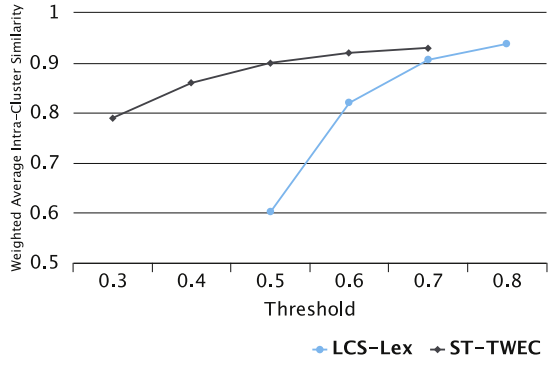

Fig. 10. Weighted average intra-cluster similarity for 60K tweets with different thresholds.

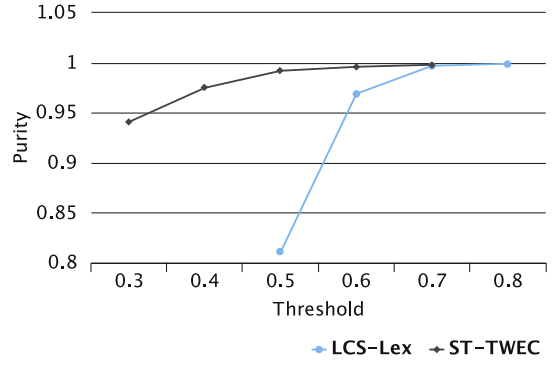

Fig. 11. Purity for 60K tweets with different thresholds.

there is no significant difference in cluster qualities produced by *LCS-LEX* and *ST-TWEC*. However, we cannot perform statistical significance tests on the cluster results generated by the exact same threshold values for the two algorithms. This is due to the fact that *LCS-LEX* and *ST-TWEC* use different similarity measures for clustering, therefore the threshold values are not comparable. For that reason, we have modified our null hypothesis to a more specific one. Our modified null hypothesis states that for a given threshold value for *LCS-LEX* and a cluster evaluation metric, we can find a threshold value for *ST-TWEC* where there is no significant difference in cluster quality results. We have chosen a fixed threshold value of 0.7 for *LCS-LEX* and experimented with varying threshold values for *ST-TWEC*. This was a practical choice since *ST-TWEC* produces results in a very short time. We have observed that for a threshold value of 0.55 for *ST-TWEC*, the cluster qualities are not significantly different in terms of the average intra-cluster similarity and number of clusters produced by *LCS-LEX* with threshold 0.7. We obtained $p = .07$ for the number of clusters, and $p = .68$ for the average intra-cluster similarity using a two-tailed paired t-test. These results favor our null hypothesis since the obtained p-values are greater than .05. In terms of the purity score, we also observed that the difference of clustering qualities produced by *ST-TWEC* with threshold of 0.68 is not statistically significant when the threshold is 0.7 for *LCS-LEX* since we obtained $p = .09$ using a two-tailed paired t-test. This result also supports our null hypothesis since the obtained p-value is greater than .05. We used a normal probability plot in order to show that the datasets follow normal distribution to ensure that t-tests can be performed. For instance, Figs. 13 and 14 show the number of cluster results versus the normal scores for *LCS-Lex* and *ST-TWEC* respectively. We observe that the data points lie more or less on a straight line; therefore, we can say that the datasets follow normal distribution. We observed that this is also valid for other datasets; however since we have too many datasets, we cannot show all of the normal probability plots for the sake of readability.

We mentioned about two ratios for *ST-TWEC* in Section 4.2 which are the ratio of the number of tweets in an ancestral node to the number of tweets in its child node and the ratio of the length of the substring of an ancestral node to the length of the substring of its child. Those ratios were intuitively specified as 1.2 and 0.8 respectively. In order to understand the impact of those ratio values, we conducted experiments on the dataset of size 60K tweets when threshold is 0.6 for *ST-TWEC*. Figs. 15 and 16 show the change in the cluster qualities measured by the number of clusters, the number of unclustered tweets, the average intra-cluster similarity, and purity score for different ratios of the length of the substring of an ancestral node to the length of the substring of its child. We detected an observable increase until 0.6–0.7 range for these metrics and then they were less sensitive after the ratio of 0.7. For that reason, we specified a value of 0.8 for this ratio. On the other hand, there was no observable change in the average intra-cluster similarity, and purity metrics for different ratios of the number of tweets in an ancestral node to the number of tweets in its child node as shown in Fig. 17. This was also the case for the number of clusters and the number of unclustered tweets. However, we observed a considerable improvement in the timing performance of *ST-TWEC* when we increase the ratio from 1.1 to 1.2. Therefore, 1.2 was chosen for this ratio. We repeated the timing experiment 10 times, and Fig. 18 shows the average of these timing results for different ratios.

Fig. 8 shows that the number of unclustered tweets is between 10K and 20K which can be considered a little high. However, we have used Twitter Stream API while collecting tweets regarding specific hashtags. Some of the tweets are not related to others (or have important difference) although they contain the same hashtag. Therefore, it is expected that some tweets are outliers. We

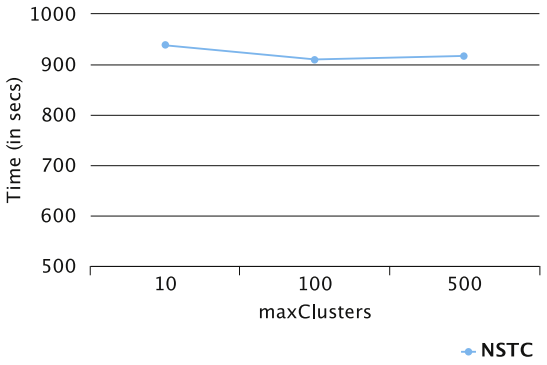

Fig. 12. Time performance for *NSTC* with different *maxClusters* values.

have also checked the similarity between each unclustered tweet and the composed clusters. For the similarity between an unclustered tweet t and a composed cluster c, we calculated the average similarity between t and each tweet in c. Thus, we are able to find the closest composed cluster to t. Finally, we checked the average of the similarities between each unclustered tweet and the corresponding closest cluster. Results show that this score is always smaller than the threshold. For instance for the threshold 0.7 of *LCS-Lex*, this score is 0.48; the same situation is valid for the other thresholds as well. We can conclude that unclustered tweets are really not related to the composed clusters.

In order to show the effectiveness of *ST-TWEC*, we also compared it with new suffix tree document clustering algorithm (*NSTC*) which was developed by Chim and Deng (2007). *NSTC* mainly utilizes word based STC, but then it maps all nodes in the suffix tree to M dimensional vector space model where M is the total number of nodes. In other words, each document d is represented as in Eq. (8).

$$d = \{w(1,d), w(2,d), w(3,d), \ldots, w(M,d)\} \tag{8}$$

In Eq. (8), $w(n, d)$ represents the weight of node n in document d and it is calculated by the $tfi - df$ formula as in Eq. (9) where $tf(n, d)$ refers to the total traversed times of document d through node n; and $df(n)$ refers to the number of the different documents that have traversed node n. Then *NSTC* uses Group-average Agglomerative Hierarchical Clustering (GAHC) and cosine similarity metric to calculate the similarity between documents.

$$tfidf(n,d) = (1 + \log(tf(n,d))) \cdot \log\left(1 + \frac{N}{df(n)}\right) \tag{9}$$

In order to apply *NSTC* to our dataset which contains 60K tweets, we used a publicly available implementation[8] of the algorithm. We used default values in the implementation for *clusterOverlapDegree* (the minimum overlapping degree for two clusters to be combined into a single one) and *minClusterWeight* (the minimum weight of a cluster to be considered) parameters which are 0.3 and 0.01 respectively. However, we used different values (10, 100, and 500) for the *maxClusters* parameter which is the maximum number of clusters (this value was selected as 500 in the original STC algorithm).

Fig. 12 shows that the time performance of *NSTC* varies between 910 and 938 s for different *maxClusters* values. Remember that *ST-TWEC* takes 23.8–25.5 s to cluster the same data. Limiting maximum number of clusters causes to have higher number of unclustered tweets in *NSTC* as shown in Fig. 19.

The average intra-cluster similarity for *NSTC* increases with higher *maxClusters* values as shown in Fig. 20; however, *ST-TWEC* has better average intra-cluster similarity results as shown in

[8] https://github.com/gratianlup/DocumentClustering.

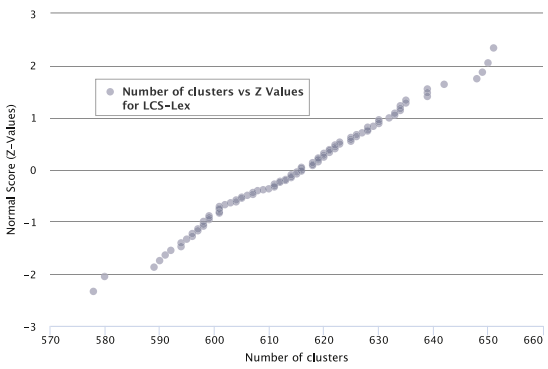

Fig. 13. Normal probability plot for the dataset of obtained cluster numbers from 100 subsets when the threshold is 0.7 for LCS-Lex.

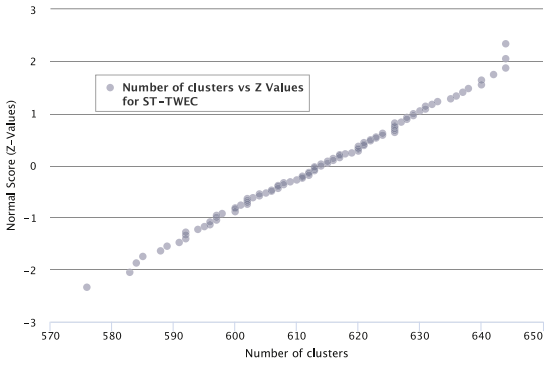

Fig. 14. Normal probability plot for the dataset of obtained cluster numbers from 100 subsets when the threshold is 0.55 for ST-TWEC.

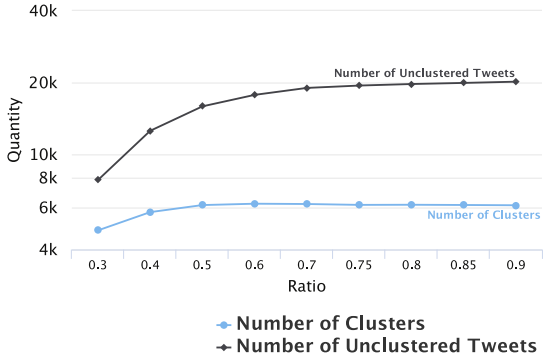

Fig. 15. Cluster number and unclustered tweets number results for different ratios of the length of the substring of an ancestral node to the length of the substring of its child when threshold is 0.6.

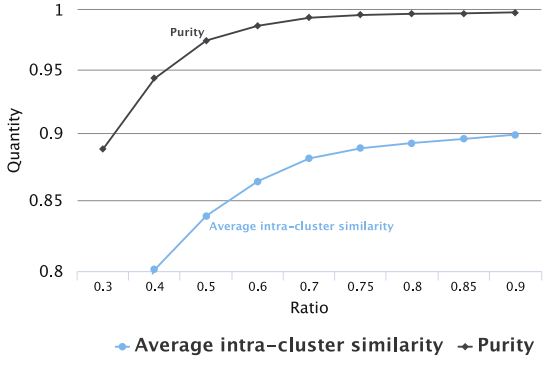

Fig. 16. Average intra-cluster similarity and purity results for different ratios of the length of the substring of an ancestral node to the length of the substring of its child when threshold is 0.6.

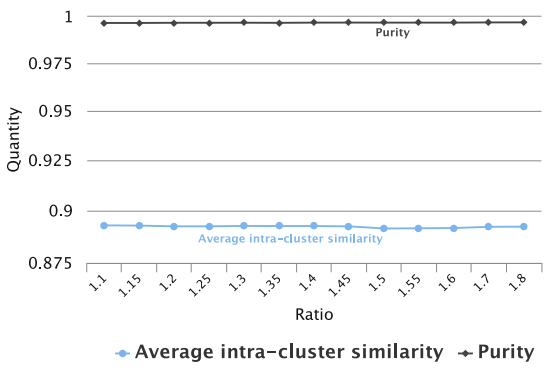

Fig. 17. Average intra-cluster similarity and purity results for different ratios of the number of tweets in an ancestral node to the number of tweets in its child node when threshold is 0.6.

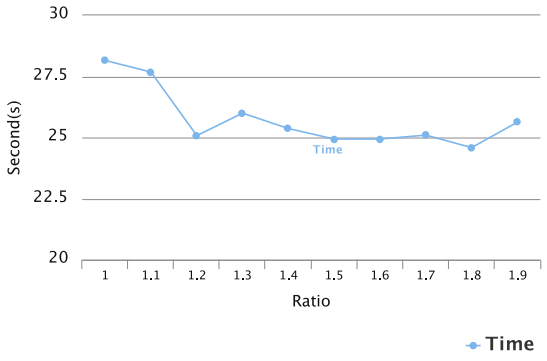

Fig. 18. Timing results for different ratios of the number of tweets in an ancestral node to the number of tweets in its child node when threshold is 0.6.

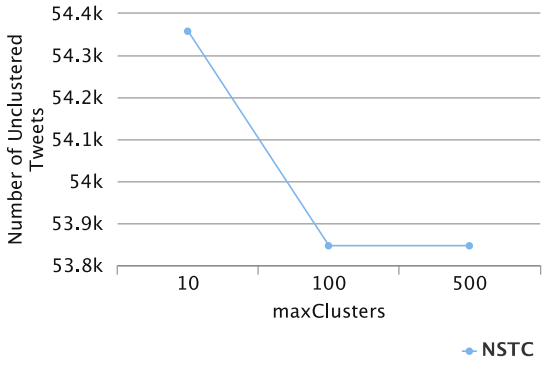

Fig. 19. Number of unclustered tweets for *NSTC* with different *maxClusters* values.

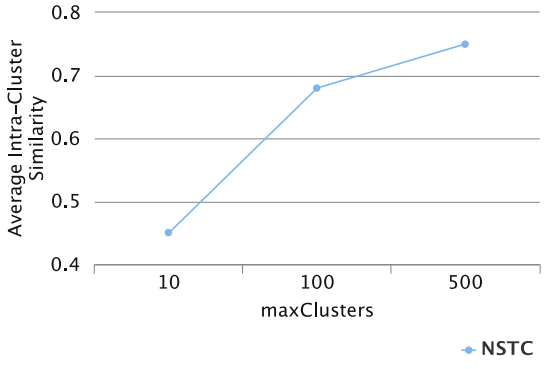

Fig. 20. Average intra-cluster similarity for *NSTC* with different *maxClusters* values.

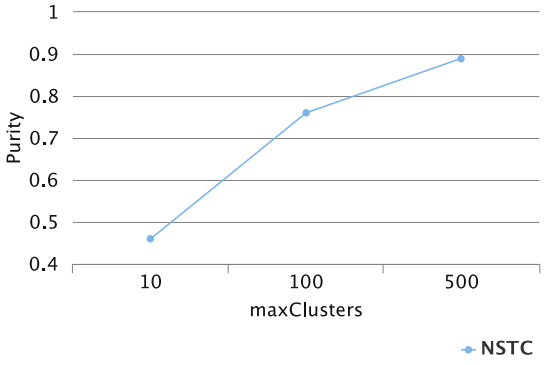

Fig. 21. Purity for *NSTC* with different *maxClusters* values.

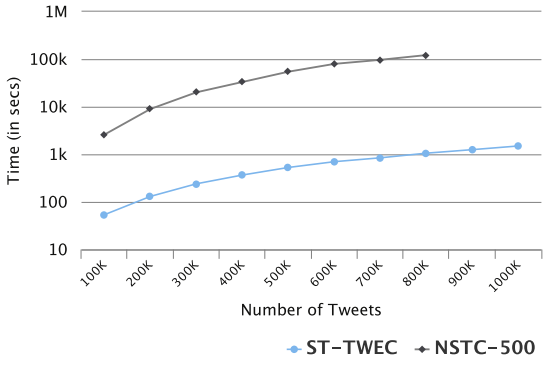

Fig. 22. Time performance for *ST-TWEC* with threshold 0.4 and *NSTC-500* with different number of tweets (y-axis was scaled logarithmically).

Fig. 9. Similar result is also observed in Fig. 21 for Purity scores. Additionally, we tested *NSTC* with increasing dataset sizes when *maxClusters* value is 500 and compared it with *ST-TWEC* when the threshold 0.4. We call *NSTC* method as *NSTC-500* when we set *maxClusters* value to 500. We observed that *NSTC-500* takes 2607, 9003, 20,165, 33,230, 54,684, 79,823, 97,128, and 122,272 s when there are 100K, 200K, 300K, 400K, 500K, 600K, 700K and 800K tweets respectively. Note that *ST-TWEC* takes 54, 131, 241, 368, 531, 702, 847, and 1060 s on the same datasets as shown in Fig. 22.

We have also conducted experiments to compare *ST-TWEC*, *LCS-LEX*, *NSTC-500*, and *k-means document clustering* in terms of performance and cluster qualities using Precision, Recall and F-Score. In order to apply k-means algorithm, we represented all tweets in vector space model with tf-idf values and used cosine similarity measure to find similarity between vectors. Since our data was collected using 4 different hashtags, we specified k value of k-means as 4. We have 60K tweets in our data set that resulted in a large high dimensional vector space representation causing poor time performance for k-means. Using k-means took 72,013 s to complete clustering. Remember that *LCS-Lex* spends 6825–45,472 s for different thresholds and *ST-TWEC* spends 23.8–25.5 s for different thresholds to cluster the same data. Time performance of k-means can be improved by reducing vector dimensions, however this will affect the cluster qualities in a negative way. As we mentioned before, we compare cluster qualities with Precision, Recall and F-Score values as in Eqs. (10)–(12) where *tp* represents "true positive", *tn* represents "true negative", *fp* represents "false positive", and *fn* represents "false negative".

$$Precision = \frac{tp}{tp + fp} \tag{10}$$

$$Recall = \frac{tp}{tp + fn} \tag{11}$$

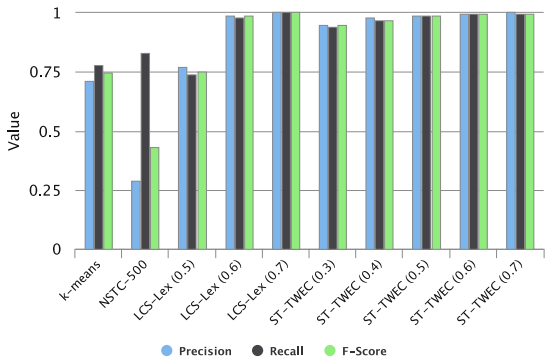

Fig. 23. Precision, Recall and F-Score results for "charlie" cluster.

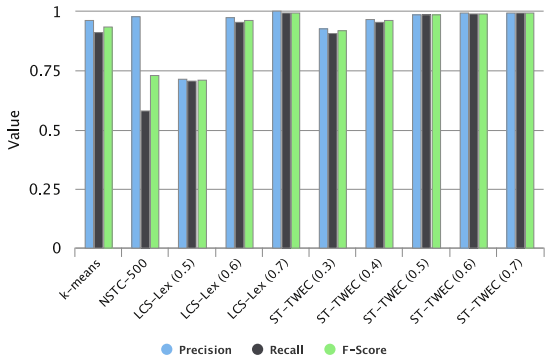

Fig. 24. Precision, Recall and F-Score results for "#christmas" cluster.

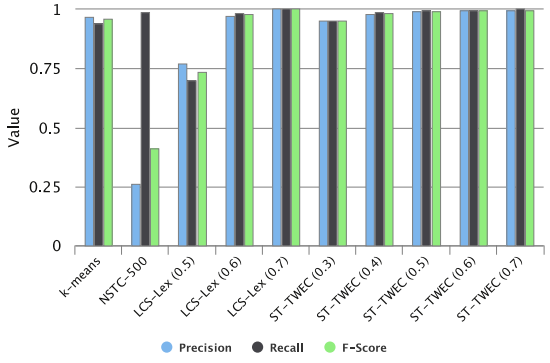

Fig. 25. Precision, Recall and F-Score results for "#nba" cluster.

$$F\text{-}Score = \frac{2tp}{2tp + fp + fn} \tag{12}$$

While analyzing k-means results, we assume that the most frequent real class label (hashtag) in any cluster is *true positive* for that cluster. Remember that we had many (much more than 4) clusters from *LCS-Lex, ST-TWEC* and *NSTC-500*. In order to make a good comparison with k-means, let's assume that there are 4 big clusters (referring to #jesuisCharlie, #christmas, #nba and #trump) as k-means has and each of these clusters belongs to one of the 4 big clusters depending on the most frequent real class label again. Comparisons of k-means, *NSTC-500, LCS-Lex* (with different thresholds), and *ST-TWEC* (with different thresholds again) for each big cluster are given in Figs. 23–26 respectively. In these charts, for instance "*LCS-Lex (0.5)*" means *LCS-Lex* has been applied with threshold *0.5*.

As it can be seen from Figs. 23–26; *LCS-Lex* and *ST-TWEC* mostly outperforms *NSTC-500* and k-means document clustering algorithm in terms of Precision, Recall and F-Score values (except from *LCS-Lex (0.5)*). It is also worth to note that there are unclustered tweets in *LCS-Lex, ST-TWEC* and *NSTC-500* as shown in Figs. 8 and 19 before, however k-means clusters all tweets in one of the 4 groups.

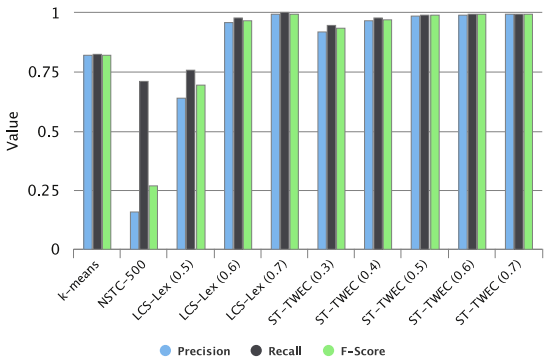

Fig. 26. Precision, Recall and F-Score results for "#trump" cluster.

We already mentioned that *LCS-Lex* is quite effective to compose good quality clusters, however its high complexity makes it ineffective to deal with large numbers of tweets. Thus, it is not a surprise that *LCS-Lex* achieves good results in Figs. 23–26. However, *ST-TWEC* also achieves the same or very similar (only 0.02-0.03 difference) results with *LCS-Lex* for the same thresholds. It may first seem that *LCS-Lex (0.7)* outperforms *ST-TWEC (0.3)*, however this is not a fair comparison since we only look at the Precision, Recall, F-Score in these figures; but the number of unclustered tweets is also an important factor. When we use 0.3 as threshold, the algorithms tend to cluster more number of tweets, assigning tweets from different classes to the same cluster which decreases the Precision, Recall, and F-Score results. Instead, it is more rational to compare higher thresholds of *LCS-Lex* with higher thresholds of *ST-TWEC*. On the other hand, k-means is only competitive with ST-TWEC when the threshold is 0.3 (and this is only valid for #christmas and #nba clusters). If we use any threshold higher than 0.3, *ST-TWEC* clearly outperforms k-means.

Spina, Gonzalo, and Amigó (2014) state that links, hashtags and named entities carry semantic content; and it might be useful to consider them for similarity analysis. We have experimented with the same data without removing links, usernames and hashtags in terms of Precision, Recall and F-Score; but results just slightly got better for lower thresholds or didn't change at all for higher thresholds, and the running time almost doubled to 42 s on average. In other words, using links and hashtags did not improve the cluster qualities so much, however it affected our timing performance in a negative way. Thus, we decided to filter links and hashtags in our work.

Other alternative algorithms to be used as baseline are density based algorithms. For density based clustering which was used in text clustering, we need to identify core points with epsilon and minimum points thresholds whose complexity is also very high. In fact, we implemented a density based clustering algorithm as well but the time performance was worse than *LCS-Lex*, therefore we have decided to continue using the *LCS-Lex* algorithm as the baseline.

7. Conclusion

Clustering is a widely used data mining method to understand trends and patterns in large collections of data. As a widely used social media platform, Twitter provides a vast data resource for researchers to detect events or to understand public opinion regarding various issues. However, the volume of the data is a challenge and standard text clustering tools do not work well for short-text data with informal language generated in Twitter. In this paper, we

presented a suffix tree based tweet clustering algorithm, *ST-TWEC*, which is able to efficiently cluster tweets in large scales. In order to prove its quality, we compared *ST-TWEC* with a benchmark algorithm (*LCS-Lex*) which is a lexical tweet clustering algorithm based on Longest Common Subsequence (LCS) similarity metric. In fact, LCS is a good similarity metric for comparing tweets, however, it has a high time complexity. Our experiments revealed that *ST-TWEC* is capable of constructing high quality clusters as *LCS-Lex* constructs in terms of avgISim, wAvgISim, and Purity clustering evaluation measures. We also show that while constructing same quality clusters, *ST-TWEC* dramatically outperforms *LCS-Lex* in terms of time performance. This outcome enables us to cluster tweets in a more efficient way and to work in scales of million tweets which experts need to handle in real life. Apart from that, we also showed that *ST-TWEC* runs more efficiently than state-of-the-art NSTC and k-means based document clustering methods in terms of time performance and cluster qualities.

We designed and implemented a customizable and expandable web based interactive tweet clustering tool (*I-TWEC*) where users can upload their tweet dataset, perform clustering, and see the constructed clusters in a graphical user interface. *I-TWEC* takes advantage of *ST-TWEC* and semantic similarities of tweets as two consecutive steps to construct high quality clusters. To the best of our knowledge, this is the only publicly available tweet clustering tool utilizing both lexical and semantic similarities which can be used by technical as well as non-technical experts through a user friendly interface. A limitation of *I-TWEC* is the memory requirement of the constructed suffix tree. As the tweet size grows up, the memory size consumed by suffix tree also increases. Our suffix tree consume 475 MB memory for 60K tweets and its size proportionally changes with the total number of characters in the tweet data set. Still, the memory requirement is comparable with state-of-the-art STC method but with superior time performance. However, in order to process a much higher number of tweets, we plan to extend I-TWEC with batch clustering and store a portion of the generated suffix tree in the secondary storage as future work.

In the current version of *I-TWEC*, the clustering threshold is adjusted by the end user. We plan to develop and implement automatic threshold adjustment in the future. Our tool (especially semantic similarity part) can also be extended by adding *Word Mover's Distance* distance function that is explained in Kusner, Sun, Kolkin, and Weinberger (2015).

We have performed experiments with some end-users about the usability of the tool, and had positive feedback. However, in the future we would like to conduct formal user studies. Last but not least, *I-TWEC* is publicly available, and developers may extend and/or modify it depending on their requirements.

References

Argyrou, A., & Andreev, A. (2011). A semi-supervised tool for clustering accounting databases with applications to internal controls. *Expert Systems with Applications, 38*(9), 11176–11181. doi:10.1016/j.eswa.2011.02.163.

Atefeh, F., & Khreich, W. (2015). A survey of techniques for event detection in Twitter. *Computational Intelligence, 31*(1), 132–164. doi:10.1111/coin.12017.

Banerjee, A., & Ghosh, J. (2001). Clickstream clustering using weighted longest common subsequences. In *Proceedings of the web mining workshop at the 1st SIAM conference on data mining: 143* (p. 144). Citeseer.

Becker, H., Naaman, M., & Gravano, L. (2011). Beyond trending topics: Real-world event identification on Twitter. *Fifth international AAAI conference on weblogs and social media.*

Bozkir, A. S., & Sezer, E. A. (2013). {FUAT} a fuzzy clustering analysis tool. *Expert Systems with Applications, 40*(3), 842–849. FUZZYSS11: 2nd International Fuzzy Systems Symposium 17–18 November 2011, Ankara, Turkey. 10.1016/j.eswa.2012.05.038.

Cheong, M., & Lee, V. (2010). A study on detecting patterns in Twitter intra-topic user and message clustering. In *2010 20th international conference on pattern recognition* (pp. 3125–3128). doi:10.1109/ICPR.2010.765.

Chim, H., & Deng, X. (2007). A new suffix tree similarity measure for document clustering. In *Proceedings of the 16th international conference on World Wide Web*. In *WWW '07* (pp. 121–130). New York, NY, USA: ACM. doi:10.1145/1242572.1242590.

Derczynski, L., Maynard, D., Rizzo, G., van Erp, M., Gorrell, G., Troncy, R., et al. (2015). Analysis of named entity recognition and linking for tweets. *Information Processing & Management, 51*(2), 32–49. doi:10.1016/j.ipm.2014.10.006.

Dominguez, D. R., Redondo, R. P. D., Vilas, A. F., & Khalifa, M. B. (2017). Sensing the city with Instagram: Clustering geolocated data for outlier detection. *Expert Systems with Applications, 78*, 319–333. doi:10.1016/j.eswa.2017.02.018.

Erpam, M. K. (2017). Tweets on a tree: Index-based clustering of tweets. *Technical Report*. Istanbul, Turkey: Sabanci University.

Fang, Y., Zhang, H., Ye, Y., & Li, X. (2014). Detecting hot topics from Twitter: A multiview approach. *Journal of Information Science, 40*(5), 578–593.

Islam, A., & Inkpen, D. (2008). Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data, 2*(2), 10:1–10:25. doi:10.1145/1376815.1376819.

Jun, S., Park, S.-S., & Jang, D.-S. (2014). Document clustering method using dimension reduction and support vector clustering to overcome sparseness. *Expert Systems with Applications, 41*(7), 3204–3212. doi:10.1016/j.eswa.2013.11.018.

Jung, J. J. (2012). Online named entity recognition method for microtexts in social networking services: A case study of Twitter. *Expert Systems with Applications, 39*(9), 8066–8070. doi:10.1016/j.eswa.2012.01.136.

Kusner, M., Sun, Y., Kolkin, N., & Weinberger, K. (2015). From word embeddings to document distances. In F. Bach, & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning*. In *Proceedings of Machine Learning Research: 37* (pp. 957–966). Lille, France: PMLR.

Li, C., Sun, A., & Datta, A. (2012). Twevent: Segment-based event detection from tweets. In *Proceedings of the 21st ACM international conference on information and knowledge management*. In *CIKM '12* (pp. 155–164). New York, NY, USA: ACM. doi:10.1145/2396761.2396785.

Liu, X., & Zhou, M. (2013). Two-stage {NER} for tweets with clustering. *Information Processing & Management, 49*(1), 264–273. doi:10.1016/j.ipm.2012.05.006.

Ma, Y., Wang, Y., & Jin, B. (2014). A three-phase approach to document clustering based on topic significance degree. *Expert Systems with Applications, 41*(18), 8203–8210. doi:10.1016/j.eswa.2014.07.014.

Martinez-Romo, J., & Araujo, L. (2013). Detecting malicious tweets in trending topics using a statistical analysis of language. *Expert Systems with Applications, 40*(8), 2992–3000. doi:10.1016/j.eswa.2012.12.015.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*. abs/1301.3781.

Miller, Z., Dickinson, B., Deitrick, W., Hu, W., & Wang, A. H. (2014). Twitter spammer detection using data stream clustering. *Information Sciences, 260*, 64–73. doi:10.1016/j.ins.2013.11.016.

Nishida, K., Banno, R., Fujimura, K., & Hoshide, T. (2011). Tweet classification by data compression. In *Proceedings of the 2011 international workshop on detecting and exploiting cultural diversity on the social web*. In *DETECT '11* (pp. 29–34). New York, NY, USA: ACM. doi:10.1145/2064448.2064473.

Poomagal, S., Visalakshi, P., & Hamsapriya, T. (2015). A novel method for clustering tweets in Twitter. *International Journal of Web Based Communities, 11*(2), 170–187.

Rangrej, A., Kulkarni, S., & Tendulkar, A. V. (2011). Comparative study of clustering techniques for short text documents. In *Proceedings of the 20th international conference companion on World Wide Web*. In *WWW '11* (pp. 111–112). New York, NY, USA: ACM. doi:10.1145/1963192.1963249.

Saraçoğlu, R., Tutuncu, K., & Allahverdi, N. (2008). A new approach on search for similar documents with multiple categories using fuzzy clustering. *Expert Systems with Applications, 34*(4), 2545–2554. doi:10.1016/j.eswa.2007.04.003.

Sayce, D. (2016). *Number of tweets per day?*. http://www.dsayce.com/social-media/tweets-day/. Accessed 2017.02.15.

Song, W., Qiao, Y., Park, S. C., & Qian, X. (2015). A hybrid evolutionary computation approach with its application for optimizing text document clustering. *Expert Systems with Applications, 42*(5), 2517–2524. doi:10.1016/j.eswa.2014.11.003.

Spina, D., Gonzalo, J., & Amigó, E. (2014). Learning similarity functions for topic detection in online reputation monitoring. In *Proceedings of the 37th international ACM SIGIR conference on research & development in information retrieval*. In *SIGIR '14* (pp. 527–536). New York, NY, USA: ACM. doi:10.1145/2600428.2609621.

Tang, G., Xia, Y., Wang, W., Lau, R., & Zheng, F. (2014). Clustering tweets using Wikipedia concepts. In N. C. C. Chair), K. Choukri, T. Declerck, H. Loftsson, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, & S. Piperidis (Eds.), *Proceedings of the ninth international conference on language resources and evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA).

Thaiprayoon, S., Kongthon, A., Palingoon, P., & Haruechaiyasak, C. (2012). Search result clustering for Thai Twitter based on suffix tree clustering. In *Electrical engineering/electronics, computer, telecommunications and information technology (ECTI-CON), 2012 9th international conference on* (pp. 1–4). IEEE.

Tu, H., & Ding, J. (2012). An efficient clustering algorithm for microblogging hot topic detection. In *2012 international conference on computer science and service system* (pp. 738–741). doi:10.1109/CSSS.2012.189.

Zamir, O. E., & Etzioni, O. (1999). *Clustering web documents: A phrase-based method for grouping search engine results*. University of Washington.

Zamora, J., Mendoza, M., & Allende, H. (2016). Hashing-based clustering in high dimensional data. *Expert Systems with Applications, 62*, 202–211. doi:10.1016/j.eswa.2016.06.008.

Zubiaga, A., Spina, D., Martnez, R., & Fresno, V. (2015). Real-time classification of Twitter trends. *Journal of the Association for Information Science and Technology, 66*(3), 462–473. doi:10.1002/asi.23186.