U.P.B. Sci. Bull., Series C, Vol. 79, Iss. 1, 2017 ISSN 2286-3540

STREAMER - A TOOL FOR CLUSTERING CONVERSATIONS IN SOCIAL NETWORKS

Andrei OPREA[1], Costin-Gabriel CHIRU[2]

In this paper, we present Streamer, a search application running over streams of Twitter messages. As opposed to most services that only do simple text search over conversations, Streamer aims to cluster messages together in order to simplify analyzing a large number of messages from similar topics. The novelty of Streamer is that, unlike most applications that use fixed corpus or categories when clustering, it works with streaming data that may debate about any number of topics: Twitter messages are continuously retrieved and the clusters are updated as more data comes in. The running time and clustering quality of the application were evaluated using purity and Silhouette coefficient.

Keywords: clustering, social media, information extraction

1. Introduction

Social media platforms are no longer an emerging field, they have become well established and millions of messages are exchanged daily by people globally. A variety of application clients and services try to keep up with this surge of information by offering users information about popular topics or highlights about the events that are happening around them. They are promoting popular content, determined either by number of clicks, views, favorites or other metrics, and although this is effective for controversial topics, it does little to highlight other subjects of conversation.

In this paper, we present a study on the clustering of messages from the Twitter platform, also known as *tweets*, inspired by the website's *Trends* category, which presents the most popular subjects either worldwide or in a certain geographical region. The aim of our paper is to facilitate the exploration of less popular subjects with the same easiness as exploring the hot ones. The developed application is equipped with a query option to allow searching for conversations on different topics and based on this option, it extracts and cluster the matching discussions. The messages provided by the website's API already provide a filtering option: one can specify keywords that he/she wants to be part of the

[1] Eng., Dept. of Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: victor.andrei.oprea@gmail.com

[2] Lecturer, Dept. of Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: costin.chiru@cs.pub.ro

messages he/she receives. This might provide some indication on the conversation topic, but getting an overview of the different conversations on the same subject is not a trivial task because messages have no obvious order. The purpose of the clustering is to offer a detailed view of the different conversations taking place on the same topic, trending or not, and to be able to gain a high-level overview of the conversations taking place around a certain subject.

1.1. Twitter Overview

Before getting into application's details, we will first offer a short description of Twitter - an online micro blogging platform and social networking website. On Twitter, users communicate through short messages (having at most 140 characters) called *tweets*. To ease the communication, people use *mentions*: the @ character followed by a person's name. This is a way to involve another user into the conversation. Another feature is the *hashtag* (the # sign) followed by a word. This is used to highlight key parts of the message. The most frequent hashtags are included in the *Trends*, popular subjects automatically extracted from tweets taking place worldwide or in a certain region. Exploring a topic reveals a very large number of messages with just as more coming in every second.

Users can also *favorite* a tweet - a public way of adding a particular message written by another user to a personal list of favorites -, and *retweet* it, meaning that they share it with the people who follow them while still attributing the message to the original author. These two options contribute to the overall popularity of a tweet: according to Twitter website, there are 500 million tweets sent daily by its 320 million monthly active users with a record of 143,199 tweets per second [1]. People usually turn to Twitter during major natural events, sporting events, award ceremonies and so on. With such a high amount of information coming in every second it is almost impossible to keep track of everything that is discussed.

The purpose of Streamer is to provide close to real-time clustering of conversations that take place on Twitter and to offer an overview for conversations spanning over the topics the user provided as input.

The paper continues with the discussion of the related work in the field, implemented solutions that deal with Twitter conversations or scientific papers written on this subject. We then provide an overview of the system architecture. The next section goes into details regarding testing and the obtained results, while the last part will present the discussions and conclusions that can be drawn based on these results.

2. Related Work

There are many services and applications that are in some way dependent on Twitter data, whether they simply archive or actually parse and analyze public users' streams. In the same time a lot of academic papers are concerned with the content of tweets because of the large number of messages exchanged daily. This section will focus on providing insights in the current state of both existing client applications and academic papers regarding the subject of Twitter conversations.

2.1. Existing Solutions

A big category of services that provide information regarding Twitter data are analytics services. Most of them are offering information about the engagement of followers with the created content. Their goal is to help increase the visibility of tweets for different companies, and therefore the metrics are related to the followers and focus less on exploring the content. This is the solution offered by the Twitter analytics. Another such example is SproutSocial [2], which allows one to publish content from their application to Twitter, and then monitors it for engagement and offers analytics on the users who interacted with this content. An application that tries to solve a similar problem to Streamer is TweetArchivist [3]. Using it, one is able to query specific time frames and see top users and words related to certain search terms, as well as the most shared URLs and the most influential users that have send messages. Influential users are the users with a large number of followers, their messages reaching a large number of people, get viewed, re-tweeted and shared by thousands of other users.

TweetMotif [4] retrieves tweets using the Twitter API based on a user provided query. It uses n-grams to extract a certain number of topics and groups the messages under those topics, therefore giving an overview of what people are saying. N-grams are continuous sequences of *n* items from a document (they can be syllables, letters or words).

Another class of applications related to Twitter data are 3rd party clients that allow filtering the content based on a particular hashtag and popularity (rated according to favorites and number of re-tweets). This is a good alternative for finding popular opinions: one can judge them by how popular those tweets are.

Finally, the last class of applications is represented by Twitter Trends, which extracts a list of frequent keywords that are present in the tweets from a certain region or worldwide. This allows real time browsing of the tweets having those keywords. The user does not have any control over the data, so exploring it means going through each tweet and reading it, which may be impossible.

2.2. Related Research Topics

Similar work in the field of data mining related to Twitter conversations and topic trends are pushing the limits regarding the information and insights one can obtain from analyzing tweets and also how to achieve this most efficiently.

Sudhof's work [5] aimed to cluster Twitter users into groups based on the opinions they expressed regarding a political controversy. The corpus was fixed and contained tweets from the time when the events occurred, that were selected based on their hashtag. Different methods were used to cluster the users. One of them used *TF-IDF* [6] and *term-weighting* [7] to extract relevant keywords from messages, as multiple keywords shared between tweets are an indication of how similar they are and thus link the users together. Another method was to use *Mentions* (referencing one or more users in your tweet), as this means that they are somehow involved and relevant to your opinion. Finally, *hashtags* were considered, the idea behind it being that users who send out messages using the same hashtags share similar opinions [8]. Again, the assumption is that the more hashtags users share, the more similar they must be.

Rosa et al. [8] tried to classify tweets into different categories that were predefined and the corpus was fixed, composed of selected tweets that covered the predefined topics. They considered hashtags to be an approximate indication of the message's topic and used it to improve the clustering results. First of all, the messages were pre-processed: tokenization, removal of rare terms, conversion to lowercase, etc. Both unsupervised (K-Means [9] in combination with TF-IDF) and supervised (Rocchio classifier [6]) methods were used for clustering. Their observations based on the obtained results were that simple normalization rules such as removing terms with very low frequency did not improve clustering but did help reduce dimensionality; TF-IDF similarity based ranking proved effective in discovering representative tweets; and training sets quickly become obsolete if attempting to use supervised clustering for real time data.

Similar approaches were tried in order to extract and assign tweets to different categories using a number of predefined templates that were updated through automatic learning [10] or by creating clusters defined by their topic [11].

Compared to these approaches, our application doesn't have predefined topics or corpora, trying to cluster the tweets as they are posted (attempting online clustering), without any prior knowledge about their topics.

O'Connor, Krieger and Ahn's TweetMotif [4] fetches tweets from Twitter Search API, generates 2-3 words topics from the newly formed corpus and associates messages to these topics. Its interface allows for a recursive detailing of the topics, the goal being to offer a concise summary of the generated topics. Topic generation is achieved using n-grams with certain heuristics such as disregarding unigrams that are function words, or bigrams, trigrams that cross syntactic boundaries. Topics are merged and their sets of messages are combined. Our research is different than this one as although we want to offer an overview for different conversations, we don't aim to summarize them.

Another paper by Sahami and Heilman [12] handles the issue of measuring similarity for short sentences (e.g. search queries), which might not share any terms (e.g. A.I. and Artificial Intelligence, refer to the same thing but it is not trivial to find this). The proposed solution was to use the terms to query the web for documents and therefore provide context on it. These are called *context vectors*. Thus, even though the terms *AI* and *Artificial Intelligence* would yield a cosine similarity of 0, gathering a number of documents on these terms would help improve the context and compute a much better similarity for the two. This paper was helpful for the evaluation part of our research, when we had to assess the quality of clustering. Thus, instead of the context vectors we used the most frequent words from a cluster to provide its context and to decide whether a tweet is correctly placed in that cluster or not.

Chen, Nikolov, Shah [13] proposed a method for an accurate estimation of the topics that will become trending on the Twitter social platform, before they are declared trending by the website itself. The proposed method classifies topics into trending or not trending by using a nearest neighbor classifier employed over the concepts time series. The results of their experiments yielded a 79% success rate in detecting trends faster than Twitter, with a true positive rate of 95% and a false positive rate of 4%.

Finally, other researches did sentiment analysis on tweets, evaluating the stock market fluctuation [14], the impact of different events [15], or the elections' outcome [16]. However, sentiment analysis is out of the current research's scope.

3. Application's Details

At a high level, Streamer's architecture is built up of two logical parts:

1. A **backend** consisting of several services that communicate with the Twitter API, retrieve tweets that match the query provided by the user, and perform parsing and clustering. The result is a list of messages annotated with the cluster id they belong to. This information is made available via an HTTP server with a public endpoint. We have chosen JSON as the format for publishing the data via the API.
2. A **frontend**, which is responsible for receiving the JSON and for rendering clusters of tweets as well as providing an interface for the user to explore the conversations. The user can see the related messages, view the original message to get the context and the replies, and visit the user's profile.

The backend is in turn composed by 3 main modules: *data acquisition, data processing and data clustering. Data acquisition* is responsible for fetching

Twitter messages (tweets) using the Twitter4j library and stores them in a queue provided by Apache Kafka. The reason for using queues between intermediary steps is that they allow for the modules to operate at different frequencies, completely decoupling them. As the Twitter API gives access to public data either from individual users or a percentage of all data going through the platform, we opted out to use the public message streams, which return real time tweets.

Data processing parses the raw tweets and converts them into shorter messages containing key terms. Messages are read from the queue filled in by the previous module and then all the non-alphanumerical characters are removed. The next step is to parse the tweets using StanfordNLP library, thus each word being annotated with its own part-of-speech tag. Afterwards, the words are filtered out based on these tags, some of them (personal pronouns, possessive pronouns, prepositions, conjunctions) being removed as they are non-informative and might decrease the accuracy of the clustering algorithm. Also, the extra information would increase the amount of time required for clustering. The processed tweets are written to a new queue, thus decoupling this module from the next one.

Data clustering is responsible for clustering the messages based on the keywords generated in the previous step. A clustering algorithm groups a set of documents (tweets, in our case) into subsets called clusters. The goal is to generate clusters with high intra-cluster similarity (having a sufficient number of coherent documents) and low inter-cluster similarity (being given any two distinct clusters, they should be different enough from each other). Usually in clustering algorithms the key metric is the distance between points (e.g. the Euclidian distance for points in a two-dimensional space). The algorithm used for this step was K-Means (MacQueen [17]) clustering and the chosen distance function was the cosine similarity between tweets.

The K-Means algorithm works by starting with a seed consisting of randomly selected points (in our case messages that constitute different centroids). The algorithm then proceeds to assign the remaining messages to the existing centroids using the distance function (cosine similarity). After each step, the centroid is re-computed as the average of the messages that have been assigned to the cluster it defined. These steps are repeated until a stopping criterion is reached.

The algorithm's objective is to minimize the average distance between a point and its centroid. Thus, if we define *residual sum of squares (RSS)* as in (1) (where K is the number of clusters, x is a point from cluster w_k and $\mu(w_k)$ is the centroid of cluster w_k), the purpose of K-Means is to minimize this value.

$$\text{RSS} = \sum_{k=1}^{K} \sum_{x \in w_K} \left| \mathrm{x} - \mu(w_K) \right|^2 \tag{1}$$

Starting from this purpose, a couple of termination criteria could be used: when the RSS value falls below a certain value ε; when the centroids remain the same between 2 consecutive iterations; when the assignments of points to clusters do not change over iterations or once a fixed number of iterations have been. This upper bound is added to ensure that the algorithm halts if it fails to reach convergence, but it might also affect the quality of the obtained results.

Due to the fact that K-Means is a greedy algorithm, it might end up in a local minima instead of converging to the global minima. Thus, to avoid this problem, the algorithm is usually run several times with different seeds and then, the results generating the smaller RSS are returned. However, this affects the overall running time of the application, so a trade-off has to be made. Depending on our goals (either speed or precision), we can choose to just rely on the first run of the algorithm or to try multiple runs and retain the best one.

Another issue that might influence the results quality is related to choosing the correct value of K - the number of clusters. Most papers suggest having domain knowledge over the data that is being clustered, but in our case this is impossible due to the fact that data is coming in real time and it is unknown what their topic is about. Using RSS function and choosing a K value that minimizes it cannot be done because RSS will reach its minimum for K = N (a cluster for every tweet in the dataset). A solution to this problem was presented in [6]: start with K = 1, compute the value of RSS and then increment K's value at each step and compute the new values for the RSS until the optimum K is found (see (2)).

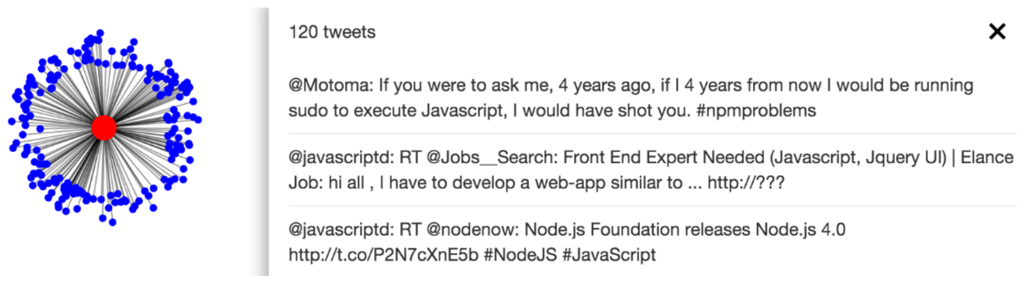
$$K = \min\left(RSS_{\min}(K) + \lambda * K\right) \tag{2}$$

This is a generalized approach to the previous solution, as we can see that setting λ to 0 will yield the best solution for *K* = *N*. However, choosing the value of K this way involves multiple runs of the K-Means algorithm which, in our case, represents an impediment as the growing runtime makes real-time clustering impossible. Thus, the value of K was chosen to be K = sqrt(total no. of tweets / 2), the value of RSS being used only for finding the best distribution of points in the K clusters.

The result of the clustering algorithm as well as the tweet message and its author are combined and converted to a JSON data structure that is saved on the disk to be consumed by the endpoint accessing the server.

The application's frontend is responsible for data visualization, which is rendered in the browser by reading the JSON files provided by the backend. When new data becomes available, the frontend renders the clusters and its adjacent nodes. The polling process continues in the background afterwards and additional information that might appear is appended to the output.

From the user interface, one can observe the clusters and quickly identify the interesting ones. The user is able to see all the messages that belong to a cluster either by hovering over the nodes or by clicking the cluster and getting an expanded view with all the messages from that cluster (see Fig. 1, where the tweets are the blue dots, while the red on is the cluster's centroid).


120 tweets ✕

@Motoma: If you were to ask me, 4 years ago, if I 4 years from now I would be running sudo to execute Javascript, I would have shot you. #npmproblems

@javascriptd: RT @Jobs__Search: Front End Expert Needed (Javascript, Jquery UI) | Elance Job: hi all , I have to develop a web-app similar to ... http://???

@javascriptd: RT @nodenow: Node.js Foundation releases Node.js 4.0 http://t.co/P2N7cXnE5b #NodeJS #JavaScript

Fig. 1. Example of a cluster (red dot) with its associated tweets (blue dots) and with the menu displaying the content of these tweets (right side of the figure)

The centroid is only used as a visual cue, making it obvious which tweets belong together. There is no relationship or hierarchy between neighboring clusters, but only between a centroid and the tweets associated with it. We tried to emphasize this by drawing lines between the tweet and its centroid and by placing centroids equally distanced from each other (see Fig. 2).

The interface presents the user a more meaningful representation of the data: tweets that belong to the same cluster are drawn together and it is also possible to explore the clusters and see all the messages that compose them.

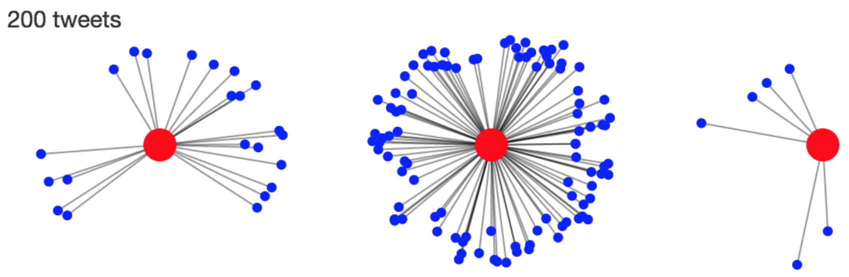


Fig. 2. Example of a cluster with associated menu displaying the content

4. Testing and Evaluation

Much of the testing involved in the development of the application was done on DigitalOcean (https://www.digitalocean.com/), which is a Platform as a Service (PaaS) that offers virtual private servers. One of the most useful feature they provide is the ability to start up, at the same time, several machines that are capable of running the system, each with custom performance capabilities. The servers used were equipped with 4 CPUs Intel(R) Xeon(R) CPUs and 8GB of RAM. Using virtual machines was an ideal setup for benchmarking because even though we used several different instances, we were able to use the same snapshot across all of them, thus ensuring a uniform testing environment.

The first test that we have made to benchmark Streamer was related to the runtime performance improvements generated by its parallel implementation. In order to do this, we have used several datasets containing from 100 to 1000 tweets. The obtained results are highlighted in Fig. 3. For the final set of data, containing 1000 messages, the performance improvement was 588s (9,8 minutes).

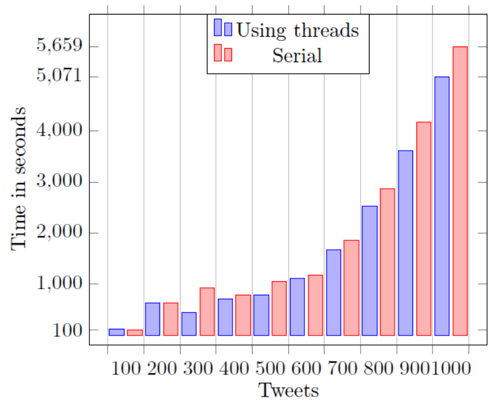


Fig. 3. Results comparison to highlight the influence of using parallelization

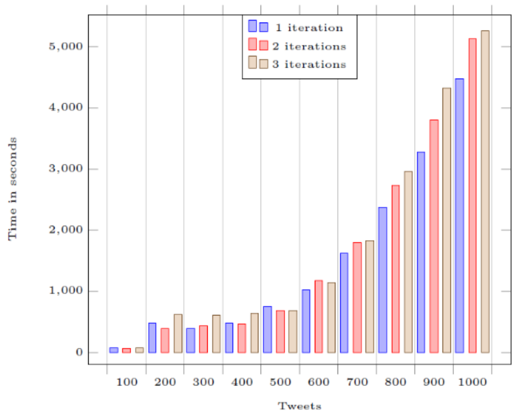


Fig. 4. The impact of increasing the number of times the K-Means algorithm is run

As mentioned in the previous section, each cluster generated by the K-Means algorithm has a different RSS value and running the algorithm multiple times can yield better results, at the cost of the running time. In Fig. 4 we present how using 1, 2 or 3 iterations of the K-Means algorithm impacts the running time.

In order to also evaluate the quality of clustering, we have used two different metrics: *Purity* and *Silhouette coefficient*. To compute *Purity* [6], we first determined the valid messages from each cluster. For each of our clusters we compiled a list with the most frequent words from the cluster and then we retained from this list only the top ε. Next, the messages were investigated in order to see if they contain all the words from the list and those who did not where considered to be incorrectly clustered. ε was empirically determined based on the obtained results to have the value 5. The number of correctly clustered messages is divided by the total number of tweets in all clusters and then these values are summed. The output of this computation is the Purity (see (3)).

$$\text{purity(C)} = \frac{1}{N} * \sum_k \max_i (w_k \cap c_i) \tag{3}$$

For evaluation, we considered the 3 scenarios, when K-Means algorithm is run once, twice or 3 times using datasets of different sizes (from 100 to 1000 tweets). The results are presented in Table 1.

Table 1

Purity Scores

Number or runs	Average Purity score
1	0.5890248
2	0.6177107
3	0.6280095

The Silhouette coefficient is computed using (4). Intuitively, this coefficient evaluates the clustering by both evaluating the clusters cohesion and the separation between clusters. The best value for this coefficient is 1 and the worst is -1, while values close to 0 indicate overlapping clusters. Thus, a negative value for this coefficient indicates that the message should be placed in a different cluster.

$$\text{Silhouette score} = \frac{\text{b - a}}{\max(\text{a, b})} \tag{4}$$

Similar to computing the Purity, for evaluating the Silhouette coefficient we have used 1, 2, or 3 runs of K-Means algorithm and datasets having between 100 and 1000 tweets. The results are presented in Table 2. They show that multiple runs of K-Means algorithm lead to better scores, although some tweets are still placed in different clusters than they should.

Table 2

Silhouette Coefficient Scores

Number or runs	Average Silhouette coefficient score
1	0.3605939
2	0.55
3	0.6068415

5. Discussion and Conclusions

Several observations can be drawn from the results. First of all, as it can be seen from Table 1 and 2, increasing the number of runs of K-Means algorithm lead to obtaining better results compared to the case when only a single run was employed, both in terms of Purity and of Silhouette score. Moreover, the clustering obtained a high purity coefficient, regardless of the number of iterations, its value being most of the time above 0.5 showing a strong cohesion between the elements of a cluster. At the same time, the Silhouette coefficient, although shows some strong clusters, also contains values close or equal to -1 which corresponds to a poor clustering. Knowing that the purity function had positive results, the main reason for the poor values of this coefficient is the inter cluster distance (which in this case is too small). One of the reasons for this small distance is the fact that the data is not homogenously distributed over topics: one or more topics requested by the user are far more popular than the others, and thus they will have more tweets. Because the initial centroids are chosen at random, it is likely that two or more initial centroids to belong to the same topic. During the algorithm iterations, the clusters will compete for messages related to the same topic and their inter-cluster distance will always be small.

Other issue that affected the results, as well as the cluster rendering in the graphical interface, is represented by the clusters that have no tweets assigned to them (except for the initial centroid). Since the number of clusters is chosen based on the total number of tweets, sometimes a higher value than necessary is considered. This affects the computation of the Silhouette coefficient because a cluster with no messages assigned will have a silhouette coefficient of -1.

In conclusion, better results can be obtained if the total number of clusters is adjusted. One possible way of doing this is to simply decrement the total number of clusters if at the end of an iteration a cluster has 0 messages assigned. In turn, the number of clusters could be increased if the purity function falls under a certain value. This would signal that some messages than do not belong to the current cluster and would rather either belong to another existing cluster (this would be signaled by the cluster distance component of the Silhouette coefficient) or would rather form a new cluster.

However, even without these adjustments, the application proved to be successful in clustering tweets, no matter of the topics they were debating on - some of the topics we have experimented with were movies, which received a great deal of reviews on Twitter and programming languages, in this case the tweets being divided in two clusters: one containing links to different tutorials and another one with job offers.

Acknowledgments

The work has been funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Ministry of European Funds through the Financial Agreement POSDRU/159/1.5/S/ 132395.

R E F E R E N C E S

[1] *R. Kirkorian*, "New Tweets per second record, and how!", 16 August 2013, [online] https://blog.twitter.com/2013/new-tweets-per-second-record-and-how.
[2] *Sprout Social - Social Media Management Software*, 2015, [online] http://sproutsocial.com/
[3] *Tweet Archivist- Simple Powerful Affordable Twitter Analytics*, 2014, [online] http://www.tweetarchivist.com/
[4] *B. O'Connor, M. Krieger and D. Ahn*, "TweetMotif: Exploratory search and topic summarization for Twitter", in Proc. of AAAI Conference on Weblogs and Social Media, 2010.
[5] *M. Sudhof*, "Politics, Twitter, and information discovery: Using content and link structures to cluster users based on issue framing", in The Stanford Undergraduate Research J., 2012, pp. 67-76.
[6] *C. D. Manning, P. Raghavan and H.Schütze*, Introduction to Information Retrieval, Cambridge University Press, 2008.
[7] *G. Salton and C. Buckley*, "Term-weighting approaches in automatic text retrieval", in International Journal of Information Processing and Management, vol.24, no.5, 1988, pp.513-523.
[8] *K. D. Rosa, R. Shah, B. Lin, A, Gershman and R. Frederking,* "Topical clustering of tweets", in Proceedings of SIGIR Workshop on Social Web Search and Mining, 2011, pp. 1304-1311.
[9] *C.C. Aggarwal and C. Zhai*, "A survey of text clustering algorithms", in Mining Text Data, 2012, pp. 77–128, Springer: Berlin.
[10] *A.Ritter, Mausam, O. Etzioni and S. Clark*, "Open Domain Event Extraction from Twitter", in Proceedings of KDD ’12, 2012, pp. 1104–1112.
[11] *H. Becker, M. Naaman and L. Gravano*, "Beyond trending topics: Real-world event identification on twitter", in ICWSM 2011, 2011, pp. 438–441.
[12] *M. Sahami and T. Heilman,* "A web-based kernel function for measuring the similarity of short text snippets", in WWW’06, 2006, pp. 377-386, ACM Press.
[13] *G. H.Chen, S. Nikolov and D. Shah,* "A latent source model for nonparametric time series classification", in Advances in Neural Information Processing Systems, 2013, pp. 1088–1096.
[14] *B. O’Connor, R. Balasubramanyan, B. Routledge and N. Smith,* "From tweets to polls: Linking text sentiment to public opinion time series", in ICWSM 2010, 2010, pp. 122–129.
[15] *A.-M. Popescu, M. Pennacchiotti and D. A. Paranjpe*, "Extracting Events and Event Descriptions from Twitter", in WWW'11, 2011, pp. 105-106.
[16] *J. Bollen, H. Mao and X.–J. Zeng*, "Twitter mood predicts the stock market", in Journal of Computational Science 2(1), 2011, pp. 1–8.
[17] *J. MacQueen*, "Some methods for classification and analysis of multivariate observations", in Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1: Statistics, 1967, pp. 281-297.