ABSTRACTIVE TWEET STREAM SUMMARIZATION USING NATURAL LANGUAGE PROCESSING

S.Annamalai[1], R.Sudharsan[2], R.Thirumalai Raj[3]
[1,2,3]Dept of Computer Science & Enggineering, Thiagarajar College of Engineering, Madurai
Email:s.annamalaikarthick@gmail.com[1],sudharsan.r16@hotmail.com[2]
thiru.drake@gmail.com[3]

Abstract

Nowadays, the size of data in social media like twitter is growing in unprecedented rate. When we closely observe those data, redundancy is the main reason for the data explosion. Sometimes we also have unnecessary noise in those data. For data analysts and end-users, both the factors (noise and size) act as a hindrance in mining the data. So in this paper we introduce a novel approach to generate abstractive summaries of multi-topic based live tweet stream using Natural Language Processing. The live tweets are pre-processed and topic based Tweet Cluster Vectors are formed by incremental clustering. A transformation matrix is constructed using the TCV's centroids and the live tweets, then an abstractive summary is generated using the proposed summarization method. The results are evaluated based on the contextual significance of the summary.

Keywords: tweet stream, summary, abstractive summarization, incremental clustering

I. INTRODUCTION

Micro-blogging services such as Twitter and many others have been playing an important role in delivering news, opinions and more for quite a while. In fact, they have taken a predominant role in social media. Since the incline in their usage every day, it has resulted in a huge amount of data being handled. Twitter, for instance, has an average of around 6,000 tweets per second which corresponds to 500 million tweets per day[1]. Tweets themselves are short meaningful sentences which convey information, but there are also many such erratic tweets which deliver the same in a redundant and noise-full manner. Even while utilizing filters for understanding a particular topic, it also results in numerous tweets. Thus it has become tiresome for an onlooker to extract useful information and for analysts to understand the evolution of these tweets over a period of time.

One of the solutions to this problem is to summarize the texts in the tweets using the process of summarization. Automatic summarization, by definition, is the process of creating a summary that keeps the most important points of the source document which in turn reduces its size. In this way, when information about a particular topic is searched, there is a diversity maintained between other such topics and redundancy of tweets on single topic is reduced. An example of the use of summarization technology is search engines such as Google. Summarization is also used for summarizing text documents, image collections and videos. However, these traditional document summarization techniques are not efficient when considered for the large number of arriving tweets. Thus a capable summarization technique has to be devised which addresses all the above difficulties.

In this proposed system, tweets are filtered based on the "topics of interest" mentioned by an user for the required amount of time,

understand those tweets and produce multiple appropriate summaries which are abstractive in nature. Then a suitable summary is selected for the purpose of understanding the topic. Since the summaries generated about a topic are instantaneous and do not consider the history of tweets, the system should be triggered at intervals and the same should be done when summaries are required on multiple topics. Thus this method differs in the extensibility provided from the already existing tweet summarization techniques.

The paper is organized as follows:

In section 2, we do a literature survey of already existing summarization method that is used for the tweet summarization problem. In section 3, we provide brief definitions to the concepts used in the paper. In section 4, we describe the model and its application for the problem statement defined. Section 5 consists of the experimental results and in section 6 we conclude our work

II. LITERATURE REVIEW

A. *Summarization by Timeline Generation for Evolutionary Tweet Streams*

One effective tweet summarization introduced in [1] solves this problem by using *real-time timelines.* The framework proposed continuously monitors all the tweets on a specific topic and creates the timelines. When a user wants to explore the tweets in a period, the system comes up with *time stamped* summaries, which are the important tweets from which the topics/subtopics had evolved from. This in practice, is what we call as *extractive summarization.* In this paper, we have introduced a method which produces *abstractive summaries*.

B. *Natural Language Sentence Generation*

A corpus based natural language generation procedure was proposed in [2] which compared the sentences generated using a multi-tier N-gram-based method trained on the BEST2010, a publicly available text corpus with a corpus TELL-S, which contains simple sentences created using textbooks belonging to Thai schools. Test sets of 195 sentences were used to evaluate the results and the sentence generation accuracies were more in TELL-S than in BEST2010.

C. *Multi-Document Summarization*

News articles describing the same event in different ways were summarized earlier [3]. News can contain both similar and contradictory information. So this multi-document summarization method was used to identify which phrases should be included in the summary by comparing them and a sentence generation method was used to reformulate them as new texts

III. BASIC CONCEPTS

D. *Pre-processing*

Pre-processing is the first step in the text mining process. The three preliminary steps of pre-processing are stop words removal, stemming and TF/IDF vector construction [4].

Stop words are a part of natural language. The purpose that stop-words should be removed from a text is that they add to the weight of the text and make it less important for the analysts. Removing stop words reduces the dimensionality of term space. The most common words in text documents are articles, prepositions, and pronouns that does not give meaning to documents. These words are treated as stop words. Example of stop words are ‘the’, ‘in’, ‘a’, ‘an’, ‘with’, etc. Stop words are removed from documents because those words are not measured as keywords in text mining applications.

Stemming is used to identify the root/stem of a word. For example, the words play, played, playing, playful all can be stemmed to the word “play”. The purpose of this method is to remove various suffixes, to reduce the number of words, to have accurately matching stems, to save time and memory space.

Term Frequency–Inverse Document Frequency (tf-IDF) is a numeric quantity which reveals how important a word is in a document. The tf-IDF is often used as a weighing factor in information retrieval and text mining.tf-IDF of a word is directly proportional to the frequency of the word in a document and inversely proportional to the word’s frequency across document in the corpus. It is calculated as

$$\text{tf-IDF}\ (t,d) = \text{tf}\ (t,d)\ .\ \text{IDF}\ (t,d)$$

where tf (t,d) is the number of times the term t occurs in the document d and IDF (t,d) is the number of documents in the corpus that contains the term t.

tf–IDF measure can be used for stop-words filtering in various subject fields including text summarization and classification. tf–IDF is the product of two quantities which are term frequency and inverse document frequency.

E. Clustering Algorithms

Clustering is the process of grouping a set of objects in such a way that the measure of relation between the objects within a group (called a cluster) is always more than that measure between an object in one group to another group. There are many clustering algorithms based on the purpose of the cluster [5].

Central vectors are used for representing clusters which necessarily may not be elements of the dataset. K-means clustering provides an optimization problem when the number of clusters is fixed to k. The objects are assigned to their nearest cluster centers which total to a count k such that the squared distances between the clusters are minimized.

F. Bigram

A bigram is (an n-gram of n=2) which is typically a sequence of two adjacent letters, syllables or words contained in a string or document of tokens. The frequency distribution of all bigrams in a string or document is commonly used for simple quantitative analysis of text in many applications such as language technology, cryptography, speech recognition, and so on.

G. Transition Matrix

A transition matrix which is also termed as probability matrix, substitution matrix, or Markov matrix is a matrix used to define the transitions in a Markov chain.

In this transition matrix, the states of the Markov chain are replaced with words and each of its entries is a non-negative real number representing a count. This count is the frequency distribution of two adjacent words, thus forming the *bigram transition matrix.* Thus, the matrix is constructed from a training data and is useful for finding out the word (Markov output word) which has occurred the most consecutively to a particular word (Markov input word).

IV. ALGORITHM DESCRIPTION

H. Collecting Tweet data from Tweet Stream

Nowadays twitter has started showing privacy concern for its data. Therefore Twitter datasets are not available for public. In order to get the tweet stream we used the Twitter4J. Twitter4J is a Twitter API binding library for the Java language licensed under Apache License 2.0. By using the Twitter4J API[2] we can filter the live tweets by specific topics and pass it to our algorithm.

I. Pre-processing the tweet data

As mentioned in Section Ⅲ B pre-processing the raw tweets is important before we cluster the tweet data.

The first step of pre-processing is to remove stop words. Consider a tweet *t*, at first we delimited the tweet by special characters, to get the words array *wa*. Stop words collection *c* is available in weka jar. So for each word *w* in *wa*, if *w* is present in c then remove *w* from *wa.* By this way we can remove stop words from *t*.

Although stemming is an essential step in pre-processing as per natural language processing. In our scenario, we should not stem our tweet data because tweets contain words which are not found in regular English language but their correctness is important for our cluster formation and summarization steps. For example, tweets contain the word “iPhone” when they are filtered on the topic, say, “Apple”. If this tweet is stemmed after the removal of stop words, the word “iPhone” results in “iPhon”, which might be a proper result after stemming but the same has to be used for clustering and summary generation. This may lead to some meaningless but popular words appearing in our summary.

J. Calculating tf-idf matrix

As mentioned in Section Ⅲ B, both term frequency *tf* and inverse document frequency *idf* is required for calculating the word relevance to the tweet. *tf* of a word says about the importance of that word in the tweet. *idf* of the word says about the specificity of the word in that tweet.

Term Frequency *tf* on a word *w* can be calculated as the total number of occurrences of *w* in the tweet *t.*

Inverse Document Frequency *idf* of a word *w* can be calculated as the total number of tweets divided by the number of tweets containing *w*.
The tf-idf of a word *w* in a tweet *t* can be calculated as follows

$$tf\text{-}idf\,(w,t) = (1 + \log tf)\log idf$$

Hence the tf-idf matrix has be generated.

K. Training Data Collection and Initial Clustering of Training Data

For any machine learning algorithm to work, we need a training data from which our algorithm learns and trains itself. Since we are using incremental clustering, we need training data to perform the initial clustering using k-means.
We will basically have numeric value for *training_data_limit*. Initially we collect the first *training_data_limit* tweets as training data *td* and perform k-means clustering on the *td*. We use cosine distance formula for distance calculation.
Distance *d* between two tweet vectors t_1 and t_2 can be calculated as follows

$$Distance(t_1, t_2) = 1 - \frac{\sum_{i=1}^{n} t_1(i).t_2(i)}{\sqrt{\sum_{i=1}^{n} t_1(i)^2}\sqrt{\sum_{i=1}^{n} t_2(i)^2}}$$

Initially we create *k* clusters, where *k* is equal to the number of topics.

L. Incremental clustering of incoming tweets

Since we are dealing with live tweet streams, the number of incoming tweets at a particular time instant is unknown. In this case, incremental clustering of tweets is more appropriate.

Algorithm 1: Incremental Tweet Clustering Algorithm

```
Input: tweet stream ts
Output: tweet cluster array
while !ts.end() do
        Tweet t = ts.next();
        index = 0
        min = MAX_VALUE
        for cluster i in cluster_array C
                if(distance(t , i.centroid)<min)
                        min = distance(t , i.centroid)
                        index = i
        index.add(t)
        index.recalculateCentroid()
```

For each incoming tweet *t*, we calculate the distance between the *t* and the centroid of each cluster *i* in the cluster_array. Then we add the *t* to the cluster *i* whose distance is minimum and recalculate the centroid for that cluster *i*.

M. Transition Matrix Generation

We have a point called *summary_geneartion_point*.
When the number of tweets received is equal to the *summary_generation_point*, we generate the Transition Matrix *tm* and hence the summary *s*.

Algorithm 2: Transition Matrix Generation Algorithm

```
Input: preprocessed_tweet_stream(ts)
Output: transition matrix(tm)
headcopy = ts
create new bag_of_words(ts)
do
        tweet = ts.next
        for (word : tweet)
                if (!bag_of_words.contains (word))
                        bag_of_words.add (word)
        ts = ts.next
        while ( ts != null)
size = bag_of_words.size()
create transition_matrix [size][size]
ts = headcopy
do
        tweet = ts.next
        for (word : tweet)
                transition_matrix
[word][next_word]++
        ts = ts.next
        while ( ts != null)
```

At first we create the bag of words *bow* which contains the list of unique words present in all of the received tweets. Then we create a two dimensional transition matrix *tm* of dimension *bow.size()* x *bow.size()*. This matrix is nothing but the *bigram transition matrix* that we discussed in Section III E. This matrix acts as a reference for the count of the number of times a two set of words had occurred consecutively in the training data.

For example, if the matrix is constructed from the training data whose tweets were filtered on the topic "Apple", then the entries in the transition matrix would be as follows.
Table 1: Transition Matrix Entries
From table 1, it can be inferred that the word "reveal" follows the word "apple" significantly.

[row , column] words	Count
[“apple” , “reveal”]	4526
[“reveal” , “conference”]	32
[“apple” , “conference”]	657
[“reveal” , “MacBook”]	5298

Similarly, the word “reveal” is followed by the word “MacBook” more prominently than it is done by the word “conference”. Therefore, the set of two consecutive words are: [“apple”, “reveal”] and [“reveal”, “MacBook”]

N. Abstractive Summary Generation

After developing the transition matrix *tm*, we should generate the summary *s*.

Algorithm 3: Abstractive Summary Generation Algorithm

Input: *cluster_centroid* , *transition_matrix*
Output: summary *s*
Generate *max_heap* for each row of *transition_matrix*
for (*word* : max_tfidf (*cluster_centroid*))
    while (*next_word_found*)
        summary.add (*word*)

next_word=transition_matrix_heap[*word*].poll()
        if (*cluster_centroid*.contains (*next_word*))
            summary.add (*next_word*)

At first we develop max_heap for each row of *tm.* For each centroid we will run the abstractive summary generation algorithm individually. Let *c* be the centroid, *w* be the word with maximum *tf-idf* value. If *w* is not present in the summary *s*, the add *w* to *s.* The next word *nw* can be obtained by polling the max_heap corresponding to the *w* in *tm. nw* is also added to *s.* The above procedure goes on until, all the words in *c* are processed.

V. EXPERIMENTS & RESULTS

The implementation of the code can be found in this github repository[3].

The experiment was performed on *Intel i7* processor with *8GB* of RAM. The *training_data_limit* for the initial clustering of words in the tweets was set to 100. The point at which the clustering operation was stopped and summary was generated, i.e., *summary_generation_point* was set to 1000.

For our experiment, we filtered the streaming tweets by the topic “trump” and the summaries generated were

- Hillary-underperform - west – win
- trump - unprecedented – urge
- georgia… - trouble’ – trump
- they - put - products – publicity
- they - predicted - products – prayingfortrump
- i - order - obamacare - on kimmel
- i - notice - obamacare – nothing
- what - elections - duck - election2016
- nevada - nobodycares

But when we use extractive summarization, the results obtained were,

- Trump has trouble at the polls at Georgia state
- Hillary underperformed leading to winning only in the West America
- Trump says “I notice Obama cares nothing”

When we use extractive summarization the top N retweeted tweets are generated as summaries. The downside of this method is that, tweets other than top N are not considered for summary generation. This causes a major lack of information which can be overcome by abstractive summarization.

VI. CONCLUSION

Summarization in general is still one of the hard problems in language technology. While problems like spam detection are mostly solved and others like sentiment analysis, parsing and Information Extraction (IE) are all making good progress, text summarization seems harder to solve. Research on summarization using information provided has been emerging in recent years. Such information are represented as text using templates. Examples of such summarization include Weather Report generated from weather data and report templates. Since abstractive summarization involves understanding information, it is still an area of research. This reflects on the nature of results in this paper.

Future works on this paper include reconstructing the proposed algorithm so that it can run in a distributed environment. Another language technology like Parts-of-Speech (POS) tagging can be used for improving quality and grammatical correctness of the summaries generated.

ACKNOWLEDGMENT

The idea for this paper was guided and mentored by Mr.S.Karthick. The work was supported by the Dept. of Computer Science and Engineering, Thiagarajar College of Engineering, Madurai.

REFERENCES

[1] Wang, Zhenhua, et al. "On Summarization and Timeline Generation for Evolutionary Tweet Streams." *IEEE Transactions on Knowledge and Data Engineering* 27.5 (2015): 1301-1315.

[2] Limpanadusadee, Worasa, et al. "Text corpus for natural language story-telling sentence generation: A design and evaluation." *Computer Science and Software Engineering (JCSSE), 2014 11th International Joint Conference on*. IEEE, 2014.

[3] Barzilay, Regina, Kathleen R. McKeown, and Michael Elhadad. "Information fusion in the context of multi-document summarization." *Proceedings of the 37th annual meeting of the Association for Computational Linguistics on Computational Linguistics*. Association for Computational Linguistics, 1999.

[4] Atif, Khan, and Salim Naomie. "A review on abstractive summarization methods." *Journal of Theoretical and Applied Information Technology* 59.1 (2014).

[5] Berkhin, Pavel. "A survey of clustering data mining techniques." *Grouping multidimensional data*. Springer Berlin Heidelberg, 2006. 25-71.