Graph-based Clustering Approach for Multi-Tweet Summarization Using a Hybrid Method

Author:

Seyed Hossein Shahsahebi

Tp053352@mail.apu.edu.my

Asia Pacific University of Technology & Innovation

Supervisor:

Dr. Selvakumar Samuel

Second Marker:

Dr. Nowshath Kadhar Batcha

Table of Contents

List of Figures

List of Tables

List of Equations

1. Introduction

1.1. Introduction

In 2004, Web 2.0 made more participation and contribution to web applications. Users involvement caused a significant impact on the development of many web applications. Applications such as Facebook, Twitter, Stack Exchange, and LinkedIn are the result of users participatory (Blank and Reisdorf, 2012; Fan *et al.*, 2018).

Social media as one of the Web 2.0 category of applications play an essential role in the human's daily life based on the accessibility of all users to share their feelings in any criteria (De and Dey, 2018). As many users share any news, events, concerns, and information on these social media and micro-blogging application, people prefer to seek for events, breaking news, and information through these networks (Jain and Guru, 2016; Kaur and Bansal, 2018).

There are too many social media applications, and each has its domain, specialty, target, and purpose (Sharma and Dyreson, 2018). Based on a report in early 2014, people spend one minute on social media applications every six minutes (Oh, Ozkaya and Larose, 2014). This fact shows the importance, impression, and impact of social media applications on human's life where improve users' motivation along with their functionality and efficiency (Mahan *et al.*, 2015).

Twitter is one of the social media applications built on top of the term Web 2.0 as a microblogging application in 2006 (Chae, 2015). People can share their thought in a short text which its size was limited to 140 characters earlier, and it increased to 280 characters recently. Twitter calls this short-expression a "Tweet". Having 500 million monthly active users and almost 500 million tweets per day in 2017, Twitter, along with news agencies, has become a significant source to follow daily news and events (Xu and Fang, 2018; Kursuncu *et al.,* 2019)

The diversity feature of microblogging system such as Twitter allows researchers to have access to an enormous source of user-generated content on different topics and subjects from different prospects (Paskevicius, Veletsianos and Kimmons, 2018). The fact of studying different users' contribution to specific criteria brings the knowledge and insight which the researcher can use to act. The process of extracting knowledge from different human perspectives is called the wisdom of the crowd (Bhatt *et al.*, 2018).

Based on the rich content of Twitter, there are many conducted research in different domains. As each user may be active in a specific domain, the wisdom of the crowd in any domain will be available through analyzing user-generated contents. Twitter has an application programming interface (API) which researcher can use to access the published content. Although diversity in Twitter helps to form the wisdom of the crowd and gain insight, there are too many spams, gossip, and false data, which lead to a challenging process of extracting valid data for further analyses (Kaur and Bansal, 2018). However, studies show that diversity in the processed data and published contents will result in the creation of crowd intelligence (Dana *et al.*, 2015; Bhatt *et al.*, 2017, 2018).

1.2. Project Background

In this section, Twitter analyses studies are mentioned, and their relation to the significance of this study has been explained. These state-of-the-art studies and methodologies are a great help to understand the current flow and use their novel ideas to boost the significance of our research.

A single tweet can consist of different aspects of the textual statement such as Emoji, URL, Emoticon, Hashtag, and Emoticon-words. These features can boost the quality of tweets processes by providing extra details to the main content. For instance, the existence of a URL in the tweet can simplify the process of tweet's knowledge extraction as the URL itself can add more detail to the short-published tweet. On the other hand, Hashtag is a singular or generated compound word which is defined using # symbol and introduced in 2007 on Twitter. Users are using Hashtag to categorize their tweets and specify them to the desired topic or category. Therefore, other users can monitor another user's contribution to a specific topic by searching the Hashtag. According to the Twitter report, users were sharing 125 million hashtags every day in 2018[1]. Emoji is a facial icon to express feelings. An Emoticon is like Emoji, except icons are not graphical and are shown using simple characters such as :). Emoticon-word is a combination of a Hashtag symbol and the feeling word such as #happy (Kaur and Bansal, 2018).

Researchers take advantage of the features described above to extract the sentiment out of the tweet, which is called sentiment analysis. An automatic text classification method to extract sentiment was provided in 2014 (Goel and Thareja, 2018) — similarly, another study conducted in 2015 to fetch tweet personality and emotions (Mohammad and Kiritchenko, 2015). Researchers in 2016 relied on the fact that the existence of emoticon-word in the tweet

[1] https://twitter.com/Twitter/status/1032676609902428160

can clearly specify the sentiment of that tweet. However, in another more complex scenario, relying on emoticon will not result in a correct answer as the emoticon in the tweet has different meaning such as #mad #man (Goel and Thareja, 2018).

In another recent study, the researcher proposes a predictive analysis method to analyze Twitter data. This method is implemented based on fine-grained and coarse-grained analysis levels. The fine-grained analysis applied to perform prediction on tweets and extract different knowledge such as emotions, topics, and sentiment. In the next step, they combine fine-grained analysis to create a coarse-grained method and predict real-world events outcome. In other words, they are using the primary tweet-level data such as publisher, sentiment, emotions, time, location, and hashtags to predict a real-world scenario by combining and processing these fine-grained details (Kursunc*u et al*., 2019).

The supply chain is another subject which is studied in 2015 by analyzing published tweets. They processed more than 20K tweets using network analytics, descriptive analytics, and content analytics method and were successful in extracting desired insight from tweets related to the supply chain (Chae, 2015).

Researchers trace tweets to detect the happened earthquake in its early stages. They were able to develop a tool to track users' tweets to recognize an earthquake quickly in Australia and Italy using burst detection. In this scenario, there will be a tweet storming when a disaster happens and enables Twitter watchers to get noticed (Avvenuti *et al.*, 2014; Robinson, Power and Cameron, 2016).

Another achievement of these studies was the calculation of the modified Mercalli intensity scale. The regular time needed to calculate the Mercalli is two hours after the earthquake using other types of tools where scientists were able to decrease it to half an hour by employing the tweets tracking system and Twitter data. Mercalli is a non-scientific measure to announce the scale of an earthquake where the researchers were able to reduce its calculation duration by ninety minutes using burst detection and tweet-level analysis (Mendoza, Poblete and Valderrama, 2018).

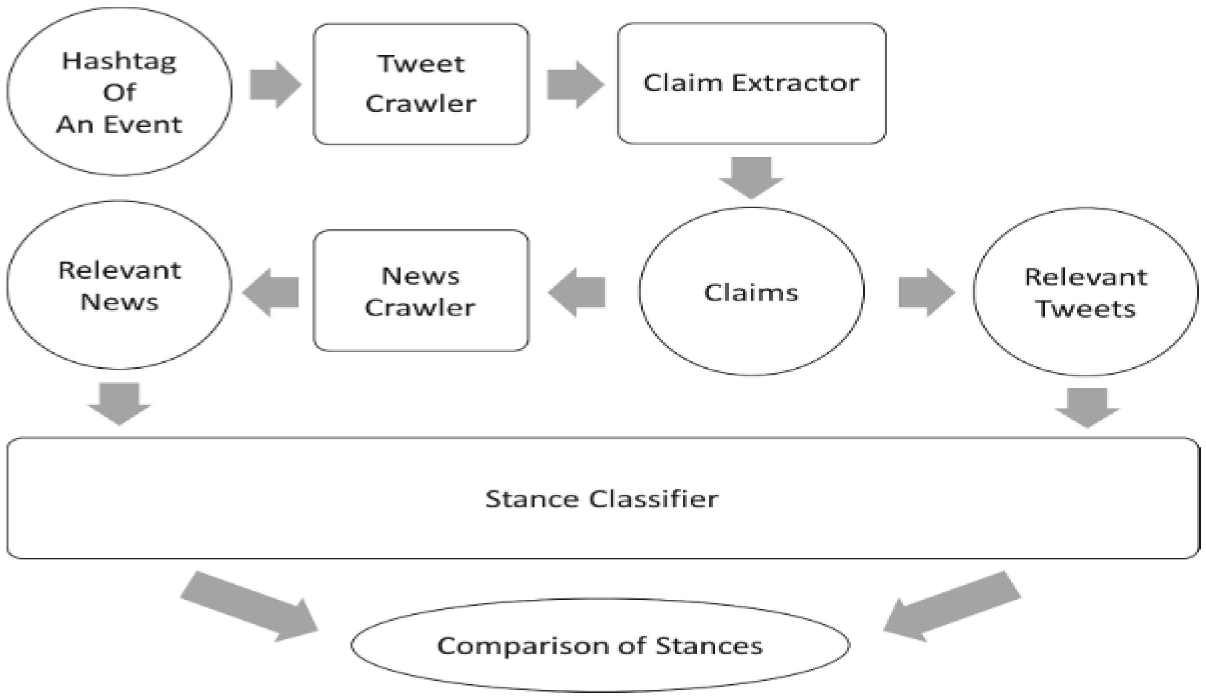


Figure 1.1: StanceComp rumor detector architecture

Another achievement of these studies was the calculation of the modified Mercalli intensity scale. The regular time needed to calculate the Mercalli is two hours after the earthquake using other types of tools where scientists were able to decrease it to half an hour by employing the tweets tracking system and Twitter data. Mercalli is a non-scientific measure to announce the scale of an earthquake where the researchers were able to reduce its calculation duration by ninety minutes using burst detection and tweet-level analysis (Mendoza, Poblete and Valderrama, 2018).

Alongside all positive and useful features and knowledge exist underneath of the Twitter microblogging network, due to the public nature of the system, every validated and non-validated contents will be published. Therefore, rumors can be generated easily on Twitter, and researchers will conduct a study based on the wrong source of information. StanceComp is a tool developed to detect rumors based on fed tweets. This tool starts functioning by providing a hashtag where it will fetch the hashtag's tweets and extract the main idea of each tweet. In the next step, the system tries to bring all similar claims from Twitter itself and news agencies. If the distribution in these analyzed contents is constant, we know the tweet is not a rumor. However, if the distribution is not reasonable, the tweet can be a rumor which needs more investigation. Figure 1.1 shows the scenario of rumor detection by StanceComp. Figure 1.2 illustrates how StanceComp works where the system analyzed a tweet. In this figure, the distribution of tweet based on other Twitter information is shown in the left pie chart, and the right chart is the distribution of the claim among the news. Both platforms are inconsistent, which lead to the fact that the tweet can be a rumor. Snopes[2] later announced that this claim was a rumor (Xu and Fang, 2018).

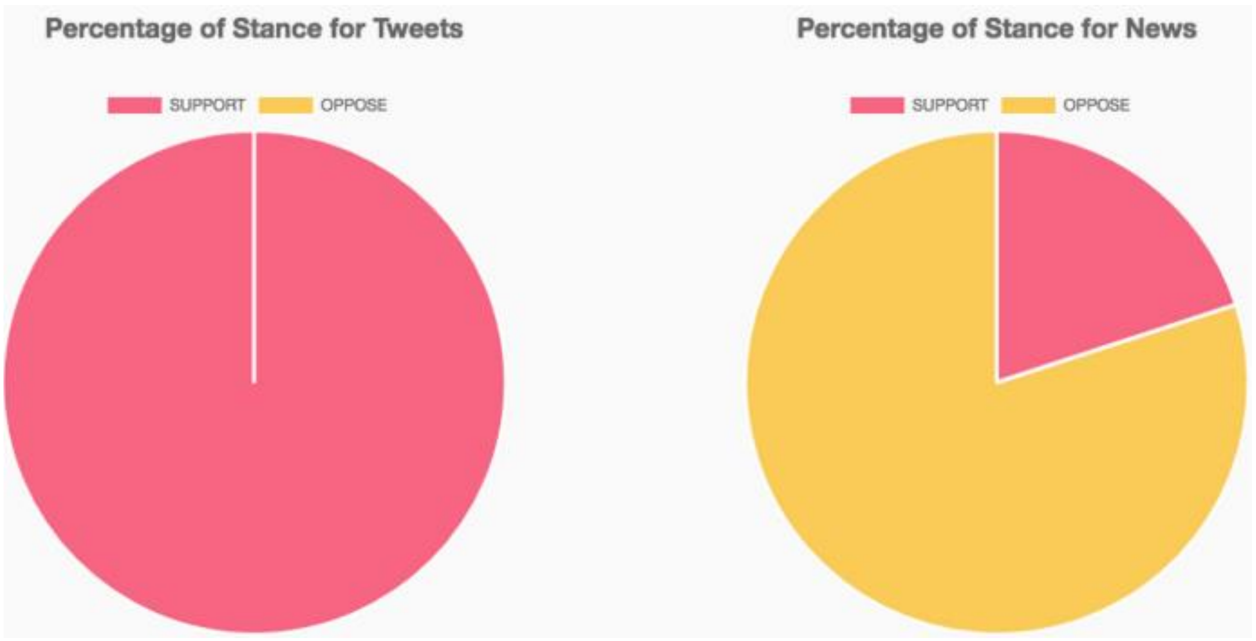


Figure 1.2: StanceComp functionality example for the tweet "Gabapentin may be new non-opioid drug of abuse"

A microblogging network such as Twitter has a natural crowd formation due to numerous and different active users. Most studies rely on this default feature and form their research. In other words, most studies rely on the bottom-up crowd formation, which starts from tweets level. Therefore, without applying any specific operation, and only by selecting tweets randomly, the crowd will be formed. However, researchers proved that creating a crowd based on a top-down method will lead to more diverse crowd formation and increase the accuracy and the quality of the result. In this scenario, after tweets had been collected, they will be categorized into different groups based on a binomial test. In the end, there are different categories where each category contains related tweets, and by selecting tweets from different types, the total diversity of the crowd will be enhanced. Researchers were able to increase crowd diversity by 63% using the top-down methodology (Bhatt *et al*., 2018).

[2] https://www.snopes.com/fact-check/gabapentin-newest-prescription-drug-killer/

1.3. Problem Statement

The fast pace of generating contents in a microblogging network such as Twitter, publish a large amount of news and data which overload users with extra information. Twitter has become an authoritative source of breaking news and daily events for its users. Generating a summary from published tweets in a specific domain enable users to have quick and easy access to the brief history of trend events by reading a single report instead of exploring every single tweet (Dehghani and Asadpour, 2019).

Knowledge extraction and improvement of public awareness are challenging actions. However, the presence of contributable network such as Twitter, overcome these concerns by facilitating the wisdom of the crowd.

Tweet summarization helps users to gain the main idea with less effort. Besides, summarizing tweets can be considered as a prerequisite to insight extraction processes. Tweet summarization helps users to gain the main idea with less effort. Besides, summarizing tweets can be considered as a prerequisite to insight extraction processes (XiaoHua *et al.*, 2012).

When users are exploring events and go through published tweets in that domain, they have to read many instances, including non-sense data or less related information. Therefore, tweets summarization empower them to track a popular topic with less distraction and direct to the point summarized report (Khan *et al.*, 2013; Shou *et al.*, 2013; Dutta *et al.*, 2015).

Many studies (Radev and Erkan, 2004; Sharifi, Hutton and Kalita, 2010; Shou *et al.*, 2013; Chin, Bhowmick and Jatowt, 2017; Dutta *et al.*, 2019) investigated the same problem and developed methods to provide a quick selection-based summarization of all available tweets in a particular keyword.

Based on the literature, many efficient approaches are considered in different studies, and combining these methods can boost up the accuracy and efficiency of the model.

1.4. Project Objective

The main objective of this study is to select the most representative tweets out of the available tweets in order to reduce the amount of time an explorer requires to get an insight on specific topic.

Explorers include all general readers, analysts, managers, and any group of people who want to grasp the wisdom of the crowd by reading a selection tweets out of thousands or millions tweets which has been published about a specific topic.

1.5. Project Significance

The outcome of this study can be used in Twitter search results. When a user searches for a keyword, related tweets will be listed in order of their published time which is not providing a comprehensive overview unless the user reviews all of them. Our approach will show the most representative tweets which help users to have an insight into the keyword they searched.

This scenario can benefit every person from a general news-seeker to business owners to find what crowd of wisdom is saying about a particular keyword.

2. Literature Review

Abstraction (Moawad and Aref, 2012; Shou *et al.*, 2013) and selection (Radev and Erkan, 2004; Shou *et al.*, 2013) are two methodologies for text summarization. Abstraction method summarizes a text by fetching the information, compressing sentences, and changing the structure of the text to transfer the meaning with less volume of text. On the other hand, the selection method computes the salient score of an individual portion of the text and only select sentences or words with the high salient score.

Checking the lexical chain of the language and maintain the correct grammar and format of the text are required in the abstraction method. However, implementation of the selection method is much simpler as there is no manipulation on the document.

The selection method could be categorized into four different methodologies. Cluster-based (Hardy *et al.*, 2002) methods divide the document into multiple groups and try to select the highest salient score from each group and aggregate all selected sentences to form the summarization text. The centroid-based (Radev *et al.*, 2004) technique considers the center sentence in document and computes the salience of each sentence based on the similarity of the phrase and central sentence. The graph-based (Radev and Erkan, 2004) method forms a graph where sentences are vertices and edges represent the similarity between two sentences that reside in connected vertices. Afterward, the salience score of each sentence computed by Markov Random Walk on the graph. Finally, machine learning methods (Neto, Freitas and Kaestner, 2002) treat the problem as a classification study of whether a sentence should be included in the summarized text or not.

Recall-Oriented Understudy for Gisting Evaluation (ROUGE) is an accessible measurement unit to measure the accuracy of the automatically summarized text. It will compute the precision and recall of automatically summarized text based on the manual version of the summary with the highest quality done by a human (gold-standard). The recall is a measure to see how many of the words exist in human review, kept in the system summary too. Similarly, precision measures the unnecessary words that the system summary has in comparison to the human-generated report. There are different versions of this method, such as ROUGE-2, which considers words two by two and compute the precision and recall based on the combination of each two words next to each other (Lin, 2004).

LexRank is a graph-based summarization method developed in 2004 to summarize news mostly (Radev and Erkan, 2004). Later, this system was used to test the summarization efficiency for tweets using the ROUGE-2 measurement unit. The accuracy dropped from 38.94% for the news to 28.71% for tweets summarization. The reduction in efficiency can be caused by noise, short length, or any other tweet-specific characteristics (XiaoHua *et al.*, 2012).

Another study (XiaoHua *et al.*, 2012) conducted to improve the summarization efficiency of the texts respecting the unique characteristics of a tweet. Based on the fact that a single tweet may contain words in an incorrect format or abnormal symbols, other attributes of a tweet such as the number of likes and retweets were included in the study to overcome the tweet salience score challenge. Readability of the tweets and preventing several tweets from the same person to be included in the summarized texts are other factors of this study, which makes it different in comparison to earlier studies. To satisfy these objectives, they used and alternative maximal marginal relevance (MMR) method, which selects increases the points of relevant sentences and reduces the points of redundant ones.

They considered the LexRank as their baseline and integrated the additional features mentioned in the previous paragraph to the method. By considering these factors, research was able to improve the ROUGH-2 score to 36.92%. In other words, considering other metadata in addition to the simple tweet text enhanced the efficiency of the text summarization and outperformed the base system.

With a step forward, another study (Shou *et al.*, 2013) aimed to create a live summarization stream from specific topic-related tweets. Published tweets on the registered keyword will be continuously monitored and will be added to the summarized text if it carries unique information in comparison to the tweets that are already monitored and processed. Two challenges were faced during the development of this research based on the redundancy and noise in tweets. The first challenge is the high volume of tweets where there is a need to process a high number of tweets real-time while improving the system's efficiency. The other difficulty is to identify sub-topics and discussions happen using the main topic.

The mentioned study wants to provide a time range filter to end-user, so he can filter summarized tweets based on the specific date or time, which introduce another challenge to the study. Sumblr is the name of the system which stands for summarization by stream clustering. To handle the stream clustering and store required information in each cluster, they use a stream clustering algorithm which has two data structures called Tweet Cluster Vector (TCV) and Pyramidal Time Frame (PTF). During the streaming process, TCV will be continuously kept in memory and will be updated with the tweet information whenever a new data arrives. PTF helps to store different snapshots from the TCV to facilitate the time range filtering process.

Tweets will be converted into a textual vector where TF-IDF specify each dimension score based on the importance of each word in the tweet. However, as other factors that affect the tweet importance as well, the tweet will be considered as a tuple, which contains the textual vector, the timestamp of the published tweet, and estimation of the tweet's author rank.

Similarly, tweets summarization is divided into categories of online (real-time), and historical. Each of these methods will use their specific data structure to fetch the data from TCV for real-time and PTF for historical summarization. The algorithm of creating the summarized text is TCV-Rank, which uses the centroid-based selection method we discussed earlier by finding the central tweet and calculating the distance for other tweets in the data structure.

TF-IDF, Term Frequency-Inverse Document Frequency, is a statistical scoring method which assigns a weight to each term based on the importance of the term. The TF component of this methodology assigns a higher score to the words that repeated more throughout the document, IDF component looks for the common words in the sentences which do not add value to the whole sentence and penalize them for reducing the total score of worthless words. TF-IDF created for document summarization, and a study tried to develop a hybrid TF-IDF to integrate the algorithm with short microblogging post. In the hybrid solution, TF component deals with the whole set of tweets in the study while the IDF component only applied to the single post. In this case, a normalization for changing the length of the tweet to a similar size had been done as TF-IDF will bias toward longer tweets.

One of the challenges in tweet summarization is choosing the optimal value for k, where k is the number of clusters or the number of selected tweets that need to be added to the summarized output. Developers of hybrid TF-IDF model tried to answer this question in a study (Inouye and Kalita, 2011) by monitoring the literature and conducting a survey. ROUGE is the most common measurement for summarization accuracy, and it highly sensitive about the summarized text length, which shows the importance of using the best value for the number of clusters. They could not find any particular method to identify k value, and they used a survey to see what will be proposed by people who summarize the tweets, which the value of

4 was selected. They were able to achieve a better result with F-measure of 25.24% in comparison to many models such as LexRank except SumBasic (Vanderwende *et al.*, 2007), which had a better measure of 25.44%.

Many popular topics will be continued over time on social media, which can include many valuable updates on a specific matter. For instance, in 2012, users posted over 237, 000 tweets per minutes regarding the presidential election of Obama. Nonetheless, when a user wants to search for such specific hashtag and review the overall abstraction of the topic, many tweets will be listed in a chronological order, which may include many redundant tweets as well. Another study (Khan *et al.*, 2013) conducted to address this issue by summarizing the tweets and shows the output to the user who searches for a particular topic. Like previous studies, researchers tried to cluster tweets into different categories based on the tweets' characteristics, and apply their selection method to select the most informational tweets from clusters and merge them to produce the outcome.

The study is based on the model which used a trained Hidden Markov Model (HMM) to identify sub-events of football matches (Chakrabarti and Punera, 2011). In the baseline model, TF-IDF with cosine similarity used to select tweets that are similar to other tweets. Football matches' related tweets are structured, and this fact is not true about many other real-time events. Therefore, to overcome the shortage of the baseline to handle worldwide events, researchers proposed this unsupervised study.

This model identifies words co-occurrence for scoring purposes like other some other summarization model (Sharifi, Hutton and Kalita, 2010; Nichols, Mahmud and Drews, 2012). Latent Dirichlet Allocation (LDA) had been used as a topic modeler to classify tweets to different clusters. LDA method needs to know the number of groups to proceed with the classification, where the cluster validation method is used to identify the best number for clusters in this study. The cluster validation method measures the efficiency of the classification algorithm result. After the classification step, a unigram lexical graph is formed for each tweet in different clusters, and undirected, weighted PageRank algorithms will be used to compute each tweet's score. Other than TF-IDF with cosine similarity model, they also used the Resonance model (Busch *et al.*, 2012), which considers factors such as re-tweet and likes count in addition to other indicators. This model was able to reach an F-score of up to 83% among all debate segments, which was more than the other two baseline models.

In another study (Dutta *et al.*, 2015) with the same objective of summarizing tweets, researchers benefited from graph-based and clustering method to summarize the text. First of all, similar tweets will be detected based on forming a graph between each tweet and counting their mutual common words in addition to semantical check, which will be conducted using WordNet (Fellbaum, 2010).

WordNet is an English lexical dictionary which is used very often when researchers try to detect words' synonyms. It represents the synonym of a specific word to facilitate us to identify similar tweets even if they have used different words with the same meaning. Another powerful option is the relation between a particular word with the more global word in a hierarchy architecture. For instance, the "tiger" is a specific word which relates to "animal", which is a universal concept.

Afterward, a community detection algorithm called InfoMap (Rosvall and Bergstrom, 2008) will be used to find a group of nodes/tweets in the graph. This algorithm will group nodes with high weight edges together and form the clusters based on the tweets’ similarity.

After clustering all tweets to different and separated groups, researchers tried to form six different models by changing the cluster representative tweet strategy. Total Summary is the first model, where InfoMap will select the cluster representative while forming the groups. Total Degree Summary is another model which select the node with the highest weight in the cluster as the representative. Total Length Summary focuses on the most extended tweet with the assumption that the longest tweet contains more information than the other tweets in the cluster.

Threshold Summary is the other developed model where a threshold will be computed using the mean value of all edges weight and edges with the weight of lower than the limit will be removed from the graph. The same selection strategies have been applied to the newly generated graph.

Among all implemented models, Total Length Summary had the best F value of 83% and also better than SumBasic model, which was introduced earlier in another study.

Another perspective of summarizing tweets is to collect the most related tweets to show to the user in their timeline. Twitter used to show timeline tweets based on the chronological order since 2016. After that, Twitter developed an algorithmic timeline approach where users will see tweets in the timeline based on their interests.

TOTEM (Chin, Bhowmick and Jatowt, 2017) is a model developed by researchers to summarize timeline latest tweets and provide an overview of the most recent activities in the timeline. The TOTEM model tends to fetch timeline tweets and store it on the user's mobile device. Then it attempts to create a summary report to show the highlights to the user where he can dig in any topic and explore it.

Before creating the report, TOTEM performs massive preprocessing actions on the data. In the first step, less critical words or indicators such as mentions, links, attached media, and non-printable ASCII characters are removed from tweets. The second step is responsible for eliminating stop words and converting abbreviations to the complete phrase by using the slang dictionary[3]. In the next level, two sibling words that have more graceful meaning will be compound together with an underscore. List of these collocations is borrowed from the chapter "Natural Language Corpus Data" of the book Beautiful Data (Norvig, 2009). The list is available online and contains more than 50,000 common bigram collocations[4]. In the final step of preprocessing, researchers use Cross-Sentence Informational Subsumption (CSIS) method to detect duplicate tweets or tweets with the same information and remove one of the similar tweets from the study. The mentioned processing method removes 13% of tweets in the database averagely.

After preprocessing actions, LDA method will be used to assign meaningfully similar tweets to the same group and to cluster the whole dataset. The default value for the number of clusters set as 8, but it will be dynamic and can be changed by the end-user.

Afterward, a vector of tweets in each cluster will be created, and each tweet will be assigned with a score using TF-IFD method. Several factors play a role in score calculation. For instance, the hashtag has the lowest weight, and popularity of the tweet has a more moderate effect than the salient score evaluated by TF-IDF with cosine similarity. In the next step, tweets with the highest calculated rank are selected as the topic representative.

To continue the study, researchers generate a topic for each cluster. All collected tweets in each group will be considered as a document. A weighted, undirected graph will be created

[3] https://www.noslang.com/dictionary/
[4] http://norvig.com/ngrams/count_2w.txt

between words of the document, where edges are the frequency of each two words co-occurrence. The process will be repeated for bigrams, and finally, the highest score words will be selected to generate the topic label. Eventually, they use Maximal Marginal Relevance (MMR) model to select the final tweets of each cluster for the summarization.

Dutta and his colleagues extended their work (Dutta *et al.*, 2019) by integrating different methodologies in addition to the methods they used in the earlier study. First of all, tweets will be preprocessed, and unnecessary sections such as mentions, non-textual, and special character will be removed. Then the similarity between tweets will be computed by considering several factors such as words count, URLs, hashtags, semantic, and cosine similarity. After calculating the analogy, a graph will be generated, where nodes are tweets and edges are the weights computed by the similarity between tweets. Similarly, several approaches used to select the most representative tweets for the summarized output.

Degree centrality summary refers to the nodes that have the highest number of connected edges, where these tweets (nodes) will be selected for the summarization. Closeness centrality method chooses tweets that have a lower distance to other tweets in the cluster. PageRank similarity counts the number of times a node appears while traversing from one node to another node. Nodes that appear the most in graph traversing will be selected for summarization. The last model is an ensembled model which is generated from six separate centrality models, including eigenvector, length, degree, PageRank, betweenness, closeness. After all, the ensemble summary has the highest accuracy with the F value of 93.1%.

In the end, Table 2.1 and Table 2.2 present a complete comparison between some of the models explored through the literature. The different preprocessing method, fine-tuning, and efficiency of different models are compared in these tables.

Reference	Alias	Sample	Preprocessing	Fine-Tuning	Method	Measure	Efficiency
XiaoHua et al., 2012	NewModel	10 million tweets	Tweets to words Remove 254 stop words Metadata Extraction Dictionary-Lookup for ill-formed words (eg. Loooove/love)	Social Network Features Tweet Readability User Diversity	Selection Graph-Based	ROUGE-1 ROUGE-2 ROUGE-5	45.62% 36.92% 16.32%
	LexRank			n/a	Selection Graph-Based		35.91% 28.71% 11.34%
Shou et al., 2013	Sumblr	20K tweets	n/a	TF-IDF to store words score Tweet's author rank Published timestamp	Clustering Centroid Selection Graph-Based	F-Score	~42%
	LexRank			n/a			~36%
Inouye and Kalita, 2011	Hybrid TF-IDF	5 Topics 1500 tweets for each	Converting unicode to ASCII Filter out embeded URLs Discarding spams using Naïve Bayes classifier	Number of clusters selected based on human summarization	Selection Cluster-Based Frequency-Based	F-Score	25.24%
	LexRank			n/a	Selection Graph-Based		20.27%
Khan et al., 2013	NewModel	270,337 tweets about presidential ellection in 2012	Removing URLs, user references, numerals, time expression, non printable characters, and stop words Tracked keywords (topic handler) will be considered as a stop word Removing duplicates and tweets with less than 10 terms	Cluster validation to identify k value in LDA. K value selected to be 10. Using unigram lexical graph	Selection Graph-Based	F-Score Up To	83.35%
	TF-IDF cosine similarity			n/a	Selection Frequency Based TF-IDF		76.10%
	Resonance Model			n/a	Selection Graph-Based		65.75%

Table 2.1: Complete comparison between multi-tweet summarization conducted in the literature - Part A

Reference	Alias	Sample	Preprocessing	Fine-Tuning	Method	Measure	Efficiency
Dutta et al., 2015	Total Summary	2921 tweets	n/a	InfoMap for clustering and representative, WordNet	Selection Graph-Based Clustering Comminity Detection WordNet	F-Score	81.80%
	Total Degree Summary			InfoMap for clustering, WordNet, Highest weight as representative			82.10%
	Total Length Summary			InfoMap for clustering, WordNet, Longest tweet as representative			83%
	Threshold Summary			InfoMap for clustering and representative, Low weights removed, WordNet			77.90%
	Threshold Degree Summary			InfoMap for clustering, Low weights removed, Highest weight as representative, WordNet			78.10%
	Threshold Length Summry			InfoMap for clustering, Low weights removed, Longest tweet as representative, WordNet			78.20%
	SumBasic			n/a	Selection Clustering Centroid		72.20%
Dutta et al., 2019	Degree Summary	2921 tweets	Stop words removal Special & non-textual characters removal Mentions removal	method-related tweet selection	Selection Graph-Based	F-Score	87.20%
	Closeness Summary						85.30%
	PageRank						83%
	Ensemble SUmmary						93.10%
	SumBasic				Selection, Clustering, Centroid		82.10%

Table 2.2: Complete comparison between multi-tweet summarization conducted in the literature - Part B

3. Methodology

In this chapter, we will discuss the methodology of performing each section of the project from start to the end. The method used to propose problem statement, objectives, and reviewing the literature is introduced. Moreover, the methodology of continuing the study, performing analysis, and presenting the result will be discussed. Figure 3.1 shows the overview of the whole study and what are the steps taken to conduct this research.

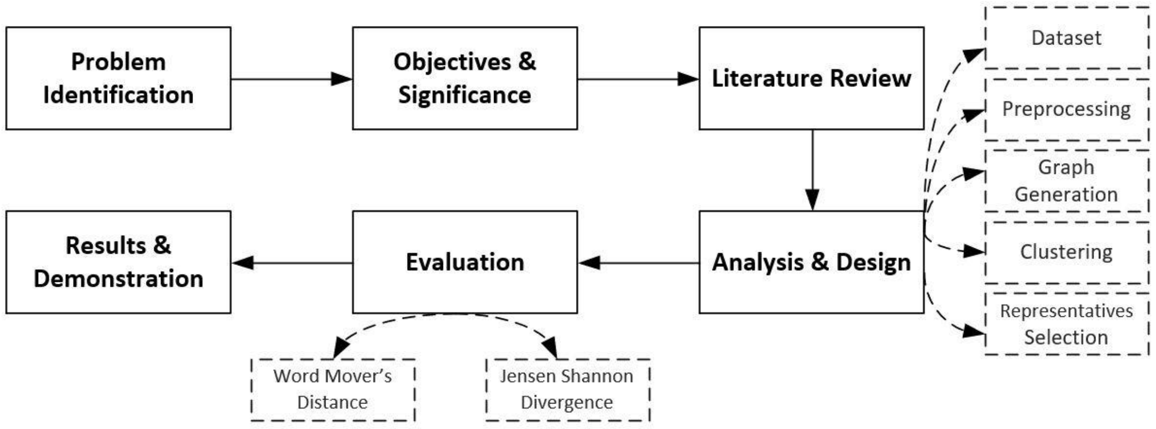


Figure 3.1: Reseach methodology steps of Graph-based clustering multi-tweet summarization

3.1. Problem Identification

As discussed earlier in section 1.3, when a Twitter user searches for a keyword to get updated, tweets will be shown in chronological order, which makes it impossible for the user to get the point from thousands of tweets that may contain redundant and noisy information.

Many studies (Dutta *et al.*, 2015, 2019; Chin, Bhowmick and Jatowt, 2017; Huang, Shen and Li, 2018; Phan, Nguyen and Hwang, 2018) had been conducted to address this issue, and each of them took a novel approach to provide multi-tweet summarization, or they used a hybrid method to achieve it. In this study, we will take advantage of conducted studies and set a hybrid plan based on the methods' results through the literature.

3.2. Setting Project Objective

The aim of this study had been discussed and elaborated in section **Error! Reference source not found.**, where the main objective is to summarize tweets to enable users to have access to the recent updates without reviewing all published tweets.

3.3. Literature Review

To study the literature, we reviewed several resources expanding from 2000 to2019. Several keywords had been used to extract resources in tweet analysis and summarization. Some keywords are listed below:

- tweet summarization
- multi-tweet summarization
- tweet analysis
- Selection-based summarization
- Graph-based summarization
- Wisdom of Crowds
- Social media analysis

Related studied are explored, and different aspect of their methods are fetched. Besides, a comprehensive comparison table is created to compare all methods in a glance based on their preprocessing, fine-tuning, and their accuracy.

3.4. Environmental Analysis

Many algorithms and strategies have been explored throughout the literature. A comparison between several methods of summarization has been prepared. Based on the knowledge achieved from previous studies, the analysis methodology for each section has been set, which we will go through each of them in the following sub-sections.

3.4.1. Dataset

In this study, we prepared a dataset including 12,393 tweets which are fetched using Twitter API[5]. These tweets are separated into two groups. The first group are the tweets which had #IranProtests in their contents. The second group contains tweets about #Brexit.

#IranProtests group contains 3,820 tweets which were published in 25 hours from December 2nd, 2019 to December 3rd, 2019. Similarly, #Brexit group contains 8,573 tweets which were

[5] http://developer.twitter.com/

published in almost 15 hours from December 2nd, 2019 to December 3rd, 2019. However, as there are Retweets and Quotes in these tweets, it will widen then range of available tweets.

These numbers show the statistics about the frequency of raw fetched tweets which are going to be pre-processed which will result in ignorance of some tweets to start the analysis and further assessments. For instance, in this study, we will ignore non-English tweets and will only analyze the tweets which were published in plain English.

3.4.2. Baseline

We chose two methods as our baseline, where our proposed method will be compared to these models. Multi-Tweet summarization of the real-time event (Khan *et al.*, 2013) and a graph-based clustering technique for tweet summarization (Dutta *et al.*, 2015) are the two baseline models of this study.

These methods, which are fully described in Chapter 2, performed extensive research and included many aspects of analysis. However, we aim to add more features and aspects into consideration to boost model accuracy.

3.4.3. Preprocessing

In this phase, the dataset will be monitored, and specified actions will be taken to remove noise. These actions are listed and described below:

Non-English Removal: All the tweets which are published in a language other than English will be ignored and removed from the future analysis. Although this study can be extended and applied to any language, for simplicity of implementation and avoiding some language barriers, only tweets in English will be processed.

Duplicate Removal: Duplicate tweets will be identified, and all instances will be deleted from the dataset except one (Khan *et al.*, 2013).

Username Removal: Some tweets may contain a username, which is not helpful for the analysis as it does not provide any further information. So, these usernames which are started with @ symbol will be removed for the analysis (Khan *et al.*, 2013; Dutta *et al.*, 2015).

Special Characters Removal: Special character such as # symbol or emoji-texts will be removed to limit the tweets to the words and phrases with actual meaning (Khan *et al.*, 2013; Dutta *et al.*, 2019).

Media Removal: Media files such as images and videos will be ignored as the analysis of these types are beyond the scope of this study, which is textual tweets summarization.

URL Removal: Embedded links in a tweet as they will not represent any meaningful information by themselves unless we analyze the contents of the destination, which is beyond the scope of this study (Inouye and Kalita, 2011; Khan *et al.*, 2013; Dutta *et al.*, 2019).

Stop Words Removal: To fetch the most out of a tweet, stop words such as in, the, and a will be removed from tweets as they do not add or change the meaning of a sentence (XiaoHua *et al.*, 2012; Khan *et al.*, 2013; Dutta *et al.*, 2019).

Short Tweets Removal: Based on the literature, short tweets generally do not convey meaningful information, and removing them from the analysis can speed up the process. In our study, we remove tweets with less than five words (Khan *et al.*, 2013).

Abbreviation and Dictation Correction: As there is a limitation for tweet's length, authors may use abbreviated words. Similarly, due to the quick typing, there is a possibility of spelling errors. We will use online tools to overcome these issues by transforming ill-formed words into their original format (XiaoHua *et al.*, 2012).

3.4.4. Graph Generation

When the data is prepared, we need to convert textual tweets into vectors, so to be able to process them and apply meaningful analysis. Word embedding is the process of transforming alphabetical words into a set of representing numbers. Afterward, these tweets which are transformed into vectors can be processed in order to find the similarity.

In our study, we will use the two state-of-the-art embedding models to vectorize tweets and apply cosine similarity to measure the interrelationship between every two tweets. In other words, a graph will be formed where nodes are vectorized tweets and edges represent the similarity weight between every two nodes. Word2Vec and GloVe are the two embedding method that we will use in our study, which will be mixed by cosine similarity later to form the similarity graph.

Word2Vec + Cosine Similarity: Word2Vec is a two-layer neural network model developed by researchers at Google in 2013 that converts a word to a vector by considering the corpus of text. Word2Vec can convert a word into a numerical vector based on the context and semantic. In other words, it considers the whole words in all tweets as a continuous bag of words (CBOW) and assigns a vector of weights to each word semantically and in a context-wise manner (Mikolov *et al.,* 2013). When the vectorization is done, cosine or Euclidean tries to calculate the cosine of the degree between two vectors. As we discussed, each tweet will have a vector representation, and the cosine of the degree between these vectors define the cosine similarity between those tweets (Nichols, Mahmud and Drews, 2012; Shou *et al.*, 2013; Chin, Bhowmick and Jatowt, 2017; Phan, Nguyen and Hwang, 2018).

GloVe + Cosine Similarity: GloVe is another similar method which is developed by Stanford in 2014. The goal is to vectorize words meaningfully by considering the context and similarity between words (Pennington, Socher and Manning, 2014). After vectorization, cosine similarity calculates the weight of edges.

3.4.5. Clustering

Once the graph is created among the involved tweets, we need to detect separate groups of tweets and categorize them in an isolated manner. These tweets formed an extensive network where nodes are connected with weighted edges. To make the summarization, we want to cluster similar tweets separately and pick the best cases out of each cluster to boost the comprehensiveness and inclusiveness. We will use two different methods for this step and create a model based on each of them.

InfoMap (Rosvall and Bergstrom, 2008) and Louvain (Clauset, Newman and Moore, 2004) methods are the most famous and efficient community detection algorithms. In the case of modularity, the Louvain method provides the highest rate in a benchmark conducted by a study (Sawhney, Prasetio and Paul, 2017). Modularity refers to the fact of how isolated are the edges inside a cluster. Silhouette score is another term of clustering measurement, which represents how similar are the nodes in a particular cluster. Based on the benchmark, InfoMap has the highest Silhouette score. However, Louvain mostly focuses on the modularity, although it keeps the Silhouette score at a reasonable value.

3.4.6. Select Clusters' Representatives

In this step, we have several clusters where each of them contains semantically and contextually similar tweets. Eventually, we form a Term Frequency-Inverse Document Frequency (TF-IDF) among the tweets in the same cluster. All the tweets in the cluster will be considered as a document, and the frequency of each word will be computed throughout the document. Therefore, each tweet will be converted into a vector, and each tweet will be assigned with a calculated weight. If a word is repeated too many times in the document, its weight will be lower.

Afterward, we apply each tweet's popularity level into the weight to generate the final score of each tweet in the cluster. In other words, the following characteristics of a tweet will be applied to the weight with lower influence than the current weight:

- Number of retweets without quotes
- Number of Faves
- Number of author's followers / Number of author's followings
- Whether the tweet author is a verified user on Twitter or not

After these calculations, each tweet will have a weight, which we can use and rank the tweets inside the cluster. Finally, K top tweets will be selected to be imported into the summarization output.

3.5. Methods

The result of the study is the output of a selection-based summarization. In other words, among a high number of tweets, the most representative and comprehensive tweets will be select and form the summarization output, which can cover the context of the whole tweets.

Based on the explained analysis, these results will be generated using the combination of multiple approaches. These approaches are forming four combinations which are listed below:

- Word2Vec + InfoMap + TF-IDF (WIT)
- Word2Vec + Louvain + TF-IDF (WOLT)
- GloVe + InfoMap + TF-IDF (GIT)
- GloVe + Louvain + TF-IDF (GLOT)

The result of this study is discussed and elaborated in this report. The output of the developed summarization system will be presented and the evaluation results will be shown in the next chapters. Also, the developed system and related codes for implementation will be publicly accessible through Github[6] repository.

3.6. Methods Validation

ROUGE is one of the most famous evaluation methods which will assess the similarity between automatic summarization and human summarization. The appendix number indicates the words' window size, where one means single word similarity will be counted, and two means a combination of each two words next to each other will be checked.

However, to use such evaluation method, we need to have a gold standard which requires a group of linguists' experts who explore the raw context and summarize the content based on two factors, including Pyramid and Responsiveness. Pyramid evaluates the quality of summary selection and responsiveness evaluate linguistic and content selection quality (Louis and Nenkova, 2013).

As mentioned above, creating a gold standard is time-consuming job and requires multiple experts. To avoid this barrier, we will use automatic evaluation methods such Jensen Shannon Divergence (Louis and Nenkova, 2013) and Word Mover's Distance (ShafieiBavani *et al.*, 2018) in our study. Researchers were able to compare ROUGE-1, ROUGE-2, and JSD methods based on the Pyramid and Responsiveness factors (Louis and Nenkova, 2013). Therefore, we will be able to calculate the ROUGE score roughly using the score we calculated based on the JSD.

[6] https://github.com/hshahsahebi/wigon

4. Data Preparation

To implement the scenario which has been elaborated in the previous chapters, we will divide the implementation into different section such as data collection, dataset preparation, pre-processing, embedding, and etc. to elaborate on every section in a distinct manner.

4.1. Data Collection

Finding a dataset of tweets including their related information such as number of favorites, re-tweets, and author information is not possible as every developer tries to create the dataset based on his research and requirements. Therefore, we set our goal to collect our own data using the Twitter Application Programming API in order to create the study's specific dataset.

Before starting to send queries to Twitter API, one needs to apply for API access and will be able to send requests and receive responds after the Twitter approved the application. API access divides to three version including standard, premium, and enterprise. Standard version is free and provide a limited information about tweets and also it trims the tweet version and doesn't let the developer to have access to all the information. Premium and Enterprise version are not free and after subscription, developer is able to send certain number and format of request based on the version he is subscribing for.

Upon account approval for API accessibility, Twitter allows developer to request for up to 5K tweets per month without any date limitation and 25K tweets per month which are published within the last thirty days. According to these options, we were able to collect almost 54,000 tweets from the Twitter API. However, we were unable to filter the request based on the language we need and these 54K tweets were published in multiple languages.

Each tweet will be provided in JSON format which includes basic information about the tweet, its author, whether it is a re-tweet or not, and whether it is a re-tweet with quote or not. In case it is a simple re-tweet, we will be able to fetch the main tweet and ignore its re-tweet, and if the tweet is a quote, we can keep the current tweet and also fetch the tweet which is quoted.

Based on the hot topics of the Twitter at the time of the research, we sent queries to collect tweets about latest Iran Protest and Brexit. Out of the ~54K collected tweets, 24K tweets were published about Iran protests and almost 30K tweets were about Brexit.

These raw tweets were stored in multiple files in JSON format without loss of information to be processed in the next phases of the implementation. Each file includes 100 tweets which makes the number of files 243 and 301 for Iran protests and Brexit respectively.

TweetCollector module is the code in python which is written to handle the data collection. The related code for this class is accessible in Tweet-Collector section of appendix. As the code shows, we have used TwitterAPI library in order to connect to the API and send queries using the API keys which are already assigned to our account by Twitter.

4.2. Dataset Creation

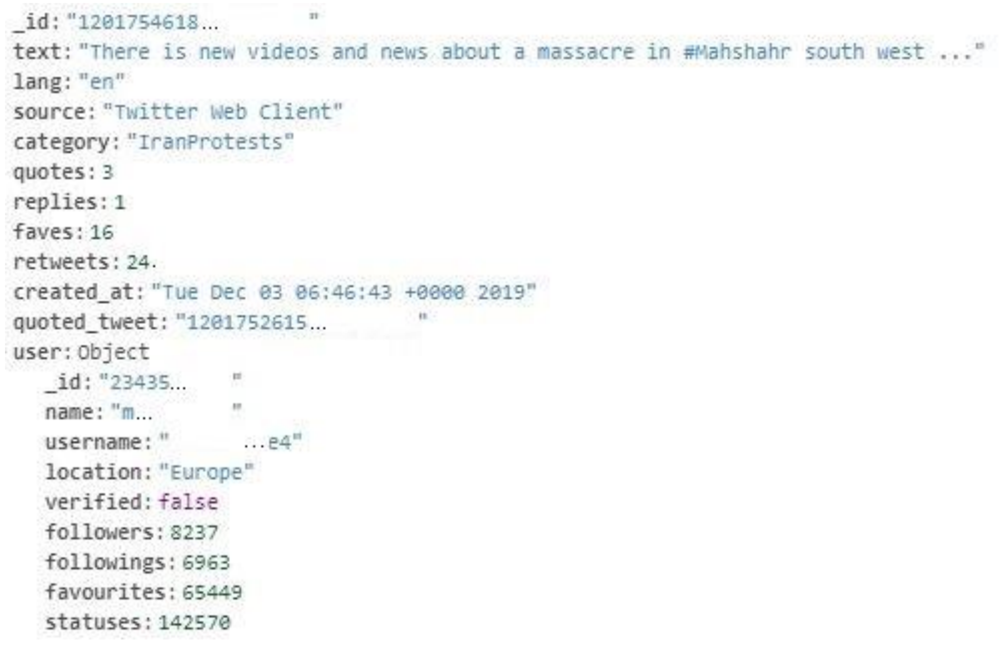
In order to create our dataset, we need to process all the collected tweets and store the tweets with the required information. We use MongoDB as our NoSQL database where we can read and write data in time- and memory-efficient manner and in JSON format.

There are three main operation in this step which we take to store the tweet in our database in order to prevent duplicate and unsupported tweets:

- If the tweet is a re-tweet, we will fetch the retweeted tweet and ignore the current tweet as it will not add any additional information.
- If the tweet is a re-tweet with a quote, we will fetch the quoted tweet and store along with the current tweet.
- We are using the Twitter language detection which is integrated in the tweet information to ignore all the tweets which are not in English. If the Twitter did not detect the language, we use langdetect python library to detect the tweet language.

After applying these operations, tweets will be store in our MongoDB database. We were able to store 12,393 tweets out of 54K based on the filtering we mentioned above. Out of 12,393 store tweet, 8,573 tweets belong to Brexit topic and the other 3,820 tweets are for IranProtests topic. Figure 4.1 shows a sample tweet format which is stored in JSON format in our MongoDB database.

DBI and DBWrapper are python modules created and written to interact with DB queries and connect to PyMongo python library to execute queries on MongoDB and perform operations on the data. The related code for these classes is accessible in Database section of appendix.



```
_id: "1201754618...        "
text: "There is new videos and news about a massacre in #Mahshahr south west ..."
lang: "en"
source: "Twitter Web Client"
category: "IranProtests"
quotes: 3
replies: 1
faves: 16
retweets: 24.
created_at: "Tue Dec 03 06:46:43 +0000 2019"
quoted_tweet: "1201752615...        "
user: Object
   _id: "23435...     "
   name: "m...        "
   username: "       ...e4"
   location: "Europe"
   verified: false
   followers: 8237
   followings: 6963
   favourites: 65449
   statuses: 142570
```

Figure 4.1: Format of a sample tweet which is stored in our database

4.3. Pre-Processing

In this section we loop through our all stored tweets and perform some pre-processing operation which are already mentioned in section 3.4.3. Afterward, some tweets may be ignored after applying these operations which we will elaborate on their implementation. The following steps are elaborated exactly based on their order of execution. However, before everything raw tweets are fetched from database using DBWrapper module. Then, we loop through all these tweets and apply the following operations:

- **URL Removal:** All URLs on Twitter will be shortened into a format which starts with t.co. We use this format to replace all these URLs with blank using regular expression and python RE package. After applying this step, there would be no URL in the tweet content.
- **Username Removal:** Usernames in the tweet are used to mention or refer to special account and does not add information to the content which is mentioned and discussed in the literature review. All username starts with @ symbol. Based on this fact, we use regular expression to replace all usernames with blank. Therefore, after this step, there would be no username in the tweet content.
- **Non-Latin Character Removal:** In this part, all the Unicode characters which are mostly emojis will be removed from the tweet content and any other character which

is not a part of ascii encoding, will be eliminated from the tweet content. By applying this operation, tweet content will only include Latin (Ascii Encoding) characters.

- **Stopword Removal:** In each language, there are some words which are required to create the sentence and keep the structure of the content but they do not add any meaning to the content. We use Spacy package in python to detect and remove stopwords from the tweet content.
- **Abbreviation Extension & Dictation Correction:** Based on the nature of the tweet and the limitation that each tweet should have up to 140 or 240 characters causes the text to have abbreviations and dictation error. In order to handle such words, we prepared a HashMap from several sources[7] to map abbreviations to their extended phrase. This HashMap is made publicly accessible[8] by the developer of this study. In addition, we use SpellChecker python package in order to correct possible errors in the tweet content. After applying this step, we expect the tweet content to be free of dictation error and abbreviations.
- **Stopword Removal:** After replacing abbreviations and correcting dictations, there may be some new stopwords that are introduced to the content. By re-applying this step, we ensure to eliminate any further stopwords if exist.
- **Special Character Removal:** In this step, any character other than alphabets, numbers, underscore, and space will be removed from the tweet content including line carriage character.
- **Final Pruning:** After applying all the above pre-processing operations, the content will be checked once again and if there is any word where all its characters are number, that word will be removed. In other words, a word like “2020” will be removed but it does not happen to a word like “Iran2019”. Also, if there is any multiple space between words, it will be replaced with one space.

When we applied all these steps on the tweet content, the preprocessed text will contain only meaningful words in lower case. After all, in this study, we ignore to process the tweet further if the number of its preprocessed words is less than five. In other words, for a tweet to be

[7] https://www.abbreviations.com/ and https://www.businessinsider.com/twitter-acronyms-2012-4 and https://www.noslang.com/dictionary/ and https://github.com/rishabhverma17/sms_slang_translator/blob/master/slang.txt

[8] https://github.com/hshahsahebi/abbreviations_json

considered in our analysis, it should have at least six meaningful words. This approach will cause to drop tweets with no useful content.

The output of the pre-processing steps has been shown in the following example. The main content of the tweet is:

"*The uprising in November showed that the regime is not capable of containing the roaring tides of protests and social rebellion, and that the Iranian people will never give in. The conditions will not revert back to the pre-uprising equilibrium. #IranProtests #Iran* https://t.co/94gMu3nvsW"

And the pre-processed content which is going to be processed throughout the analysis is:

"*uprising november showed regime capable containing roaring tides protests social rebellion iranian people conditions revert pre uprising equilibrium iranprotests iran*"

After removing tweets with less than five meaningful words, all the tweet which will be included in the further analysis reduced to 11,796. In other words, 597 tweets were ignored after pre-processing steps and will not be included in the analysis. Out of 11,796 tweets, 3,682 tweets are about Iran protests and the rest 8,114 tweets are in Brexit group.

Preprocessor module is written in python to handle these operations. Related code to this class is accessible in Pre-processor section of appendix.

5. Method Implementation

5.1. Word Embedding (Step 1)

Word embedding is the process of converting words into vectors which enable us to perform variety of computational and semantical comparison among them. In this study, we use the two most popular embedding methods which are Word2Vec and GloVe. Word2Vec developed by Google engineers in 2013 and GloVe is developed by Stanford University in 2014. Both methods are using two-layers Neural Network machine learning method to create words vector based on the training context. Embedding module is written to handle related operation in this section and the code is accessible in Embedding section of appendix.

5.1.1. Word2Vec

To implement Word2Vec model, we use Gensim python package. We provide all the tweets to train the model and will include every word in the training process regardless of the time it appeared in the content. Also, we set the configuration to create vectors with the size of 100. It means each word will be represented as a vector of 100 numbers. After training the model, we will have a vector for each and every word in the content.

5.1.2. GloVe

Similarly, to implement GloVe embedding model, we use glove-python package which enable us to train the model using our content. We provide all the tweets to the model in the training process and leave all the settings as default which is the learning rate of 0.05 and 200 epochs. Similar to Word2Vec, after finalizing the model, all the words will have an equivalent vector of 100 numbers.

5.2. Community Detection (Step 2)

After performing word embedding models, we are ready to perform community detection after constructing a graph where nodes are tweets. However, before detecting communities, we need to construct the graph of available tweets.

In order to create the tweet representative vector, we have each word's vector from the embedding section. Therefore, the tweet vector can be computed using Equation 5.1.

$$\vec{T} = \frac{\sum_i \vec{w}_i}{n}$$

$$where: 1 \leq i \ \leq n$$

$$\vec{w}_i \ is \ a \ vector \ of \ the \ word \ w_i \ in \ the \ tweet$$

$$n \ is \ number \ of \ the \ words \ in \ the \ tweet$$

Equation 5.1: Computing tweet vector based on the including words vector

Now that we have the representative vector of each tweet, we can find the similarity between each two tweets based on the cosine of their vector. Each tweet will have a node in our graph and for each two tweets we calculate the cosine similarity score. If the score is higher than 0.75 or 75%, we establish an edge between those two tweets, otherwise, there would be no edge between those tweets.

Upon construction of the graph, we provide the graph to our community detection algorithms which are Infomap and Louvain. Based on the provided graph and the schematic of the edges in the graph, these algorithms identify similar tweets in specific community or group. Therefore, we will be able to group similar tweets and perform the representative selection methods on each community. After this step, all the tweets will be categorized in their specific community and we will be ready to select the most representative tweets from each category.

All the related modules which are written in python to handle embedding and community detection are accessible in Embedding section of appendix.

5.3. TF-IDF & Representative Selection (Step 3)

When we reach this step, we have four combinations of communities to process. The first group of tweets are categorized using Word2Vec embedding model and Infomap community detection. Similarly, Word2Vec/Louvain, GloVe/Infomap, and GloVe/Louvain are the other combinations we used to separate tweets into modular groups.

Once we have prepared these processed and modular groups, we apply our representative selection method on each of them to select the representative tweets from the output of each approach. Finally, in the next chapter, we will evaluate the final output of each approach to select the best method.

However, to select the most representative tweets from each the output, we consider each tweet as a single document and all the tweets construct several documents. Afterward, TF-IDF will be applied to each document and each word in that document. The output of this application will be a score for each word which is computed by Equation 5.2. It is noteworthy that a word X which is appeared in document 1 and document 2, will have a different score as the document itself and all the vocabularies in all documents are taken into account to calculate this score.

As we have the score for each word in each document, we will be able to calculate the tweet score using Equation 5.3 where we will sum up the score of all words in that tweet. In this step, we have the TF-IDF score of each tweet, which we call it the base score. However, to increase the authenticity and the chance of selecting valid tweet, we apply other factors to the base score, which will be elaborated below:

- For each 500,000 faves and tweets, we add 1 to the base score
- We add 1 to the base score for each 50M difference between the followers and the following count. For instance, tweets that are published by Barack Obama will have almost 2.2 points from this option.
- Also, to bold longer tweets than short tweets for increasing comprehensiveness, each word in this study has 0.002 points. For instance, a tweet with 240 characters will receive almost 0.5 point from this step.
- If the tweet author is verified by Twitter, 1 will be added to the tweet base score

Therefore, based on the above elaboration and the base score we already calculated; we can compute the final score of each tweet using Equation 5.4.

$$TF - IDF(w_i) = TF(w_i) \times IDF(w_i)$$

Equation 5.2: Calculating TF-IDF score of each single word

$$TF - IDF(t_i) = \sum_{j=1}^{n} TF - IDF(w_j)$$

$$where\!: t_i \text{ } is\ a\ tweet$$
$$n\ is\ the\ number\ of\ words\ in\ tweet\ t_i$$
$$w_j\ is\ a\ word\ in\ the\ tweet\ content$$

Equation 5.3: Calculating TF-IDF score of a document or a tweet containing multiple words

$$\begin{aligned}Score_{Final} = {} & Score_{Base} + \big((Favorite_{Count} + Retweet_{Count}) \times FRC\big) \\ & + \big((Follower_{Count} - Following_{Count}) \times FFC\big) + (Word_{Count} \times WCC) \\ & + Verified\end{aligned}$$

$where: FRC\ is\ a\ constant\ which\ is\ considered\ as\ 0.000002\ in\ this\ study$

$FFC\ is\ a\ constant\ which\ is\ considered\ as\ 0.00000002\ in\ this\ study$

$WCC\ is\ a\ constant\ which\ is\ considered\ as\ 0.002\ in\ this\ study$

$Verified\ \in \{0, 1\}$

Equation 5.4: Calculating the final score of each tweet to select the most representative ones

In this level, we have the tweet score for every tweet in each community. Therefore, we are able to select the top K most representative tweets based on the developed approaches. In this study, based on the number of tweets in each community we choose the K based on the Equation 5.5 and after calculation, K tweets will be selected from each community as the representative tweets of that community. For instance, if we have 10 communities and each community has a thousand tweets, number of representative tweets can be calculated as shown in Equation 5.6.

After these operations, we have reached the study objective to select the most representative tweets out of thousands raw tweets we had selected earlier. Summarize is the python module we wrote to handle these operations, which is accessible in Summarization section of appendix.

This chapter elaborated and discussed in details on the approaches we took to achieve the research aim and selecting the top K representative tweets. However, in the next chapter, we aim to evaluate our methods and analyze our findings.

$$K = N_{C_i} * 0.001$$

$where: N_{C_i}\ is\ the\ number\ of\ tweets\ in\ the\ community$

Equation 5.5: Computing the number of representative tweets that need to be selected from each community

$$RT_{Count} = \sum_{i=1}^{10} 1000 \times 0.001 = 10$$

$where: RT_{count} is\ the\ number\ of\ total\ selected\ tweet\ for\ summary$

Equation 5.6: An example to calculate the number of representative tweets that will be selected for summary output

6. Methods Validation

So far, we had developed four methods for our base categories including IranProtests and Brexit. Each method's output contains selected representative tweets for each detected community. In this chapter, we will evaluate each method in depth and analyze the result of these methods.

As discussed earlier, Jensen Shannon Divergence (JSD) and Word Mover's Distance (WMD) are two methods that we selected from literature to evaluate our approaches. These methods are automatic ways to evaluate a summarization task and had compared to methods such as ROUGE in literature according to Pyramid and Responsiveness scores of the summary.

Therefore, based on the comparison between ROUGE-1 and JSD through literature, both methods have relatively the same measurement for Pyramid and Responsiveness factors of the text. In other words, we can assume that the JSD score that we compute for our approaches could be compared to ROUGE-1 score in order to compare two methods. So, we will use this fact to compare our selected model with other developed models we have selected as the baseline for this study.

In the next sections, we try to analyze and evaluate every approach we took for this study to learn about their output.

6.1. WIT

WIT stands for Word2Vec/Infomap/TF-IDF which is one of the approaches in our study. Based on the embedding which is done by Word2Vec, Infomap was able to find only one community for Brexit group.

It means that the processing of Brexit tweets has detected one community which contains all the available tweets for this category. At the end, TF-IDF has selected 10 representative tweets as the overall output of this approach for Brexit.

The situation for the other group, IranProtests, is similar in this approach. In other words, Infomap was able to detect only one community based on the Word2Vec embedding. This community contains all the tweets in IranProtests topic. Finally, TF-IDF selected 5 most representative out of this single community.

Table 6.1 shows the details statistics about the two categories that were processed using the WIT approach.

Category	Communities	All Tweets	Selected Tweets	WMD Score	JSD Score
Brexit	1	8110	10	0.2651	0.7608
IranProtests	1	3600	5	0.1497	0.7519
Average				0.2074	0.75635

Table 6.1: Statistic for the summarization of the approach Word2Vec/Infomap/TF-IDF for both categories and the final average score

6.2. WOLT

WOLT stands for Word2Vec/Louvain/TF-IDF approach. In other words, Word2Vec has been used to embed the tweets and Louvain used these vectors as a graph to detect the communities and finally, TF-IDF selected the most representative tweets from each community. Similar to WIT, this approach applied to both of our categories. Louvain method was also able to detect only one community for both categories as same as Infomap. Therefore, the schematic of the output for both approaches will be the same, and Table 6.2 shows the statistics for this method.

As shown in Table 6.2, Louvain was unable to detect the communities based on the embedding method of Word2Vec due the limitation in hardware capacity. Therefore, we were unable to calculate the scores for the Brexit category and will consider the IranProtests scores as the average scores of this method.

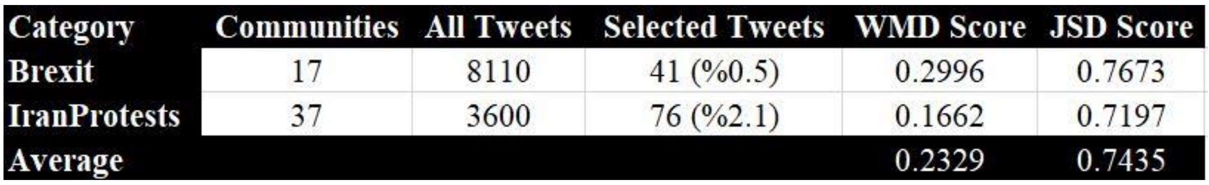
Category	Communities	All Tweets	Selected Tweets	WMD Score	JSD Score
Brexit	-	-	-	-	-
IranProtests	1	3600	5	0.1451	0.7519
Average				0.1451	0.7519

Table 6.2: Statistic for the summarization of the approach Word2Vec/Louvain/TF-IDF for both categories and the final average score

6.3. GIT

GIT stands for GloVe/Infomap/TF-IDF where we use the GloVe method by Stanford University to embed the words and create tweets vector. Afterward, Infomap uses the graph we created using GloVe output to detect the communities and categorizes tweets in a group of similar tweets. Finally, TF-IDF explore the tweets of each group and select the most representative tweets of each detected community.

Table 6.3 shows the scores and information of the GIT method on the both categories including Brexit and IranProtest.



Category	Communities	All Tweets	Selected Tweets	WMD Score	JSD Score
Brexit	17	8110	41 (%0.5)	0.2996	0.7673
IranProtests	37	3600	76 (%2.1)	0.1662	0.7197
Average				0.2329	0.7435

Table 6.3: Statistic for the summarization of the approach GloVe/Infomap/TF-IDF for both categories and the final average score

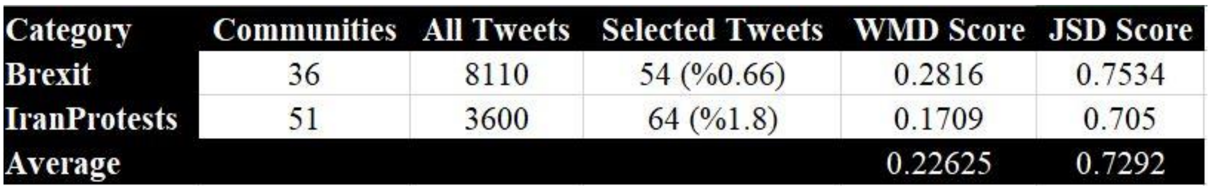
As illuminated in Table 6.3, GloVe embedded the tweets in totally different manner, where Infomap were able to detect 17 communities in Brexit and 37 communities in IranProtests groups. Number of selected tweets for each group is a sum of selected tweets in each community. The table shows that tweets in IranProtests category is more distinct than the tweets about Brexit. In other words, the similarity between the tweets about IranProtests is lower and results in a greater number of detected communities.

6.4. GLOT

GLOT is the last approach of this study which stands for GloVe/Louvain/TF-IDF. In the previous chapter we have noticed the difference in the output of GloVe and Word2Vec. In other words, the cosine similarity between nodes (tweets) were totally different in GloVe in comparison to Word2Vec which caused Infomap to detect more communities. This flow yields that the frequency of cosine similarity of lower than 0.75 was higher in GloVe embedding which resulted in a sparser graph with lower edges and finally more communities to find.

However, this section tries to apply Louvain community detection method on the output of GloVe embedding and graph generation. Table 6.4 shows this scenario and the scores achieved by executing this approach on our Brexit and IranProtests categories. GLOT is the final approach we have developed through this study.

In the next section, we will evaluate all the developed methods together to select one of our best approach, so we be able to analyze the selected method and compare it to our baseline models.



Category	Communities	All Tweets	Selected Tweets	WMD Score	JSD Score
Brexit	36	8110	54 (%0.66)	0.2816	0.7534
IranProtests	51	3600	64 (%1.8)	0.1709	0.705
Average				0.22625	0.7292

Table 6.4: Statistic for the summarization of the approach GloVe/Louvain/TF-IDF for both categories and the final average score

6.5. Aggregative Evaluation

We have explored the execution and application of four different methods to select the most representative tweets. Some methods select a greater number of tweets in comparison to others. However, the number of selected tweets does not bias the JSD score as this score is computed based on the average JSD score of each tweet and each community. Therefore, the only factor to increase the score is the quality of the selected tweets.

Also, we have explored that the GloVe embedding method makes the community detection algorithms to separate tweets into more isolated groups. In other words, GloVe works better than Word2Vec in case of distinct embedding. If the study objective is to grouping tweets in as many groups as possible, GloVe would be selected. However, in this study, the objective is the quality of the summarization, which will not be affected by how many communities are detected unless it increases the quality.

Jensen-Shannon Divergence refers to the difference between two vectors. The primary goal of our study, which we set the approaches for, yielded that we need to select the most representative tweets out of the collected raw tweets. We define the most representative as the tweet which introduce a unique information or provide an information which has the highest divergence in comparison to the other tweets in the repository. Based on this fact, we select the JSD as our main evaluation measure and compare the approaches we developed throughout this study. Based on the same fact, the higher score of JSD will be considered as higher quality regarding the objective of our study.

Table 6.5 shows the scores achieved for each approach. As we elaborated above, WIT method which used Word2Vec for embedding and creating tweets graph, Infomap to detect the communities in the formed graph, and TF-IDF to score each tweet in each community to select the most representative ones, has been selected as the leader method of our study which achieved %75.64 of JSD score. In the next section, this method will be compared to our baseline models.

Methods	WIT	WOLT	GIT	GLOT
Word Mover's Distance Score	20.74	14.51	23.29	22.63
Jensen-Shannon Divergence Score	75.64	75.19	74.35	72.92

Table 6.5: Comparing all the four approaches developed in this study, to select the method with highest quality.

WordMovers and JensenShannon are the modules written in python to handle evaluation of thse methods. The python code related to these modules is accessible in Evaluation section of appendix.

7. Findings & Conclusion

7.1. Analyze Findings

Based on the inter-methods comparison in the previous chapter, WIT method has been selected as the method with highest quality score in the summarization task. We can consider the JSD score as equivalent to the ROUGE-1 score based on the literature proof, although JSD has higher scores or Pyramid factor, but the difference and the average is not noticeable.

As we discussed in Section 3.4.2, we set two models in the literature as our baseline models. In this section, we will compare the output of this study model with the models in the baseline to measure the quality of the developed model. T shows the approximation of JSD/F-Score comparison between our method and the baseline models.

The measurement score of our model shows the lowest score among the all three models we compared. In the next report, we will try to produce the Gold-Standard report in order to have the exact ROUGE-1 Score and be able to compare our model more accurately. However, with considering the error in computation, our model does not seem to be able to superior the other two and in the best guess, it could have the equal accuracy to other models.

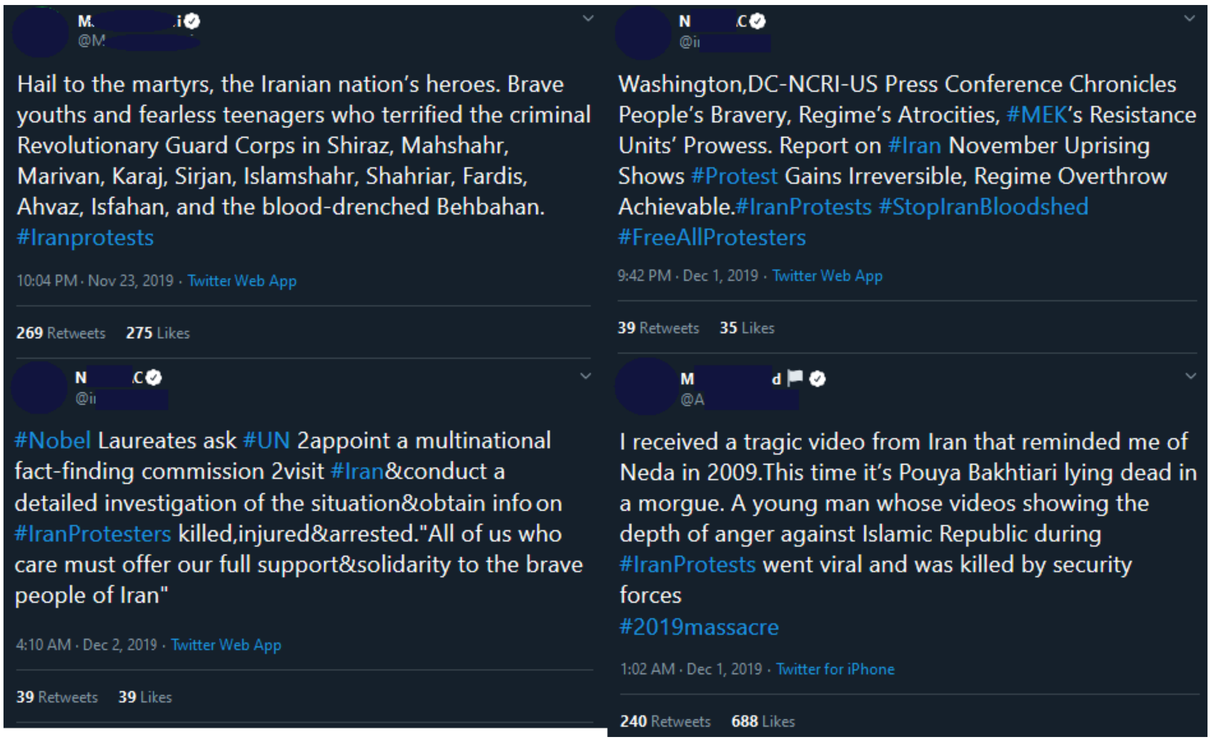
Based on this fact, we can conclude that computing the score of the representative tweets using TF-IDF and applying tweet and tweet's author popularity in the final score does not increase the summarization quality in comparison to when we simply select the longest tweets as representatives. Also, it seems that using LDA for clustering purposes, results in better grouping in comparison to Infomap and Louvain community detection methods.

After all, all the three methods compared in Table 7.1 have relatively the same level of quality in summarizing tweets. However, including tweet and tweet's popularity factors which has been done in our method, would produce a more understanding and reliable output. By applying these factors, we make sure that the author of the tweet has the popularity among the Twitter users and people validated the tweet by liking or re-tweeting it. The tweet will be selected based on factors such as the author's validity, the authenticity of the tweet, and the distinct information provided by the tweet aggregately.

Figure 7.1 shows the output of our model in summarizing IranProtests raw tweets using WIT method. This figure shows the four most representative tweets that were selected out of 3,600 raw tweets using the WIT method.

Models	Raw Tweets	Score Type	Score
WIGON - This Study	11,710	JSD	75.64%
Khan et al. Model	270,337	F-Score	83.35%
Duta et al. Model	2,921	F-Score	83%

Table 7.1: Comparison between the method developed by this study and the models selected as baseline



M[redacted]i
@M[redacted]

Hail to the martyrs, the Iranian nation's heroes. Brave youths and fearless teenagers who terrified the criminal Revolutionary Guard Corps in Shiraz, Mahshahr, Marivan, Karaj, Sirjan, Islamshahr, Shahriar, Fardis, Ahvaz, Isfahan, and the blood-drenched Behbahan. #Iranprotests

10:04 PM · Nov 23, 2019 · Twitter Web App

269 Retweets 275 Likes

N[redacted]C
@i[redacted]

Washington,DC-NCRI-US Press Conference Chronicles People's Bravery, Regime's Atrocities, #MEK's Resistance Units' Prowess. Report on #Iran November Uprising Shows #Protest Gains Irreversible, Regime Overthrow Achievable.#IranProtests #StopIranBloodshed #FreeAllProtesters

9:42 PM · Dec 1, 2019 · Twitter Web App

39 Retweets 35 Likes

N[redacted]C
@i[redacted]

#Nobel Laureates ask #UN 2appoint a multinational fact-finding commission 2visit #Iran&conduct a detailed investigation of the situation&obtain info on #IranProtesters killed,injured&arrested."All of us who care must offer our full support&solidarity to the brave people of Iran"

4:10 AM · Dec 2, 2019 · Twitter Web App

39 Retweets 39 Likes

M[redacted]d
@A[redacted]

I received a tragic video from Iran that reminded me of Neda in 2009.This time it's Pouya Bakhtiari lying dead in a morgue. A young man whose videos showing the depth of anger against Islamic Republic during #IranProtests went viral and was killed by security forces
#2019massacre

1:02 AM · Dec 1, 2019 · Twitter for iPhone

240 Retweets 688 Likes

Figure 7.1: Four most representative tweets of IranProtests topic fetched by WIT method

7.2. Conclusion

In this study, we introduced the project's aim and objective to facilitate event's explorers to be able to track the most important news on Twitter by monitoring only the group of important tweets. Due to the micro-blogging nature of the Twitter, many people including officials, provide breaking information on many events which result in thousands or millions of tweets per day on special trending topic. This study aimed to address this issue and enable explorers to follow the event by monitoring the most representative tweets instead of tracing all the tweets.

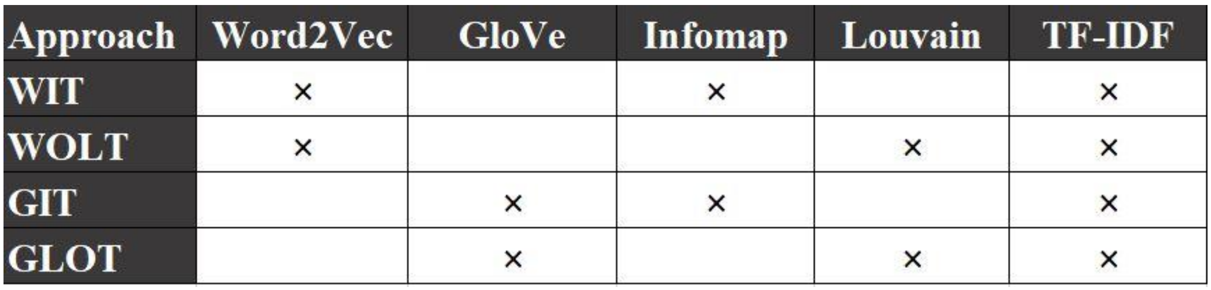
We have explored the studies in literature which worked on the same issue and developed models to address it. In this study, we tried to increase the quality of summarization by using different set of tools.

To achieve the objective of this study, we have developed four approaches to implement a model for summarizing tweets in selection-based manner. WIT is the first approach which is using Word2Vec embedding model to convert words to vectors, using Infomap community detection algorithm to detect isolated communities based on the mutual tweets' similarities. Finally, using TF-IDF to score the tweets and select the most representative ones. The second approach, WOLT, is completely the same as WIT, but using Louvain community detection instead of Infomap. GIT approach uses GloVe embedding method for vectorizing words and Infomap for community detection, where TF-IDF method does not change for scoring and selecting tweets for summarization. Finally, GLOT is similar to GIT and uses Louvain to select the communities instead of Infomap. Table 7.2 shows the same explanation in a simple table format.

In other words, we used embedding models, Word2Vec and GloVe, to embed the words in tweets content. By having word vectors, we are able to re-produce tweet vector based on its words. Afterward, each two tweets vectors will be compared together and their cosine similarity weight will be calculated. If the similarity is more than %75, and edge between two tweets will be formed and this process proceeds for every two tweets. Finally, we will have a graph, where tweets are the nodes and edges represent that the two nodes have more than %75 similarity based on cosine score.

After constructing the graph, we will feed it into community detection algorithms, Infomap and Louvain, which will monitor the graph and based on the edges detect communities to put similar tweets into their related groups. In this step, we have communities where each community contains multiple tweets that are relatively similar to each other.

Afterward, TF-IDF will be executed n each tweet in each community and the output score will be added up to some other factors such as tweet's and its author's popularity and authenticity weights. At the end, K highest scores tweets will be select from each community as the representatives and the output of the summarization task.



Approach	Word2Vec	GloVe	Infomap	Louvain	TF-IDF
WIT	×		×		×
WOLT	×			×	×
GIT		×	×		×
GLOT		×		×	×

Table 7.2: Schematic representation on the tools that were used for implementation of each approach

ROUGE-n is a standard method to evaluate summarization model. However, to achieve it, we need experts' resources to create the gold standard output where linguists and experts perform the summarization based on factors such as Pyramid and Responsiveness of the content. To simplify this process and remove the manual requirements for evaluation, researchers provided other equivalent algorithms which achieve relatively the same accuracy as ROUGE-1 recall score. Jensen-Shannon Divergence and Word Mover's Distance are two equivalent approaches provided by literature to replace ROUGE score and perform fully automatic evaluation.

Based on the JSD score, WIT approach achieved the highest score in comparison to other methods we practiced in this study. Therefore, WIT has been selected as the flagship of this study to be compared to the quality of other studies in the literature by having JSD score of 0.7564.

However, comparing our methods with the methods in the literature announced that there is no progress in the summarization process by applying our methodology in a manner of linguistic and semantic factors. ROUGE-n, JSD, and WMD evaluate the summarization in the case of the output semantic and linguistic factors based on the raw content. Therefore, comparing the models based on these measures, yielded that the model which is developed in this study does not introduce a significant increase in the quality of summarization, but also a small decrease in the measurement score. By applying tweet's popularity and authenticity along with its author's related factors, we introduced another aspect to the summarization other that content-level assessments.

7.3. Future Work

As we have mentioned earlier, to compare our model with literature models, it is better for us to prepare a gold standard to have an accurate and equal evaluation such as the compared models.

In addition, expanding the system in order to support tweets in any language and also multi-lingual tweets would be the next task. Meanwhile, we need to perform some fine-tuning operations in order to maximize the efficiency, accuracy, and the quality of summarization.

To make the system publicly accessible, we will develop a web application to enable users to query the system and explore the topics that they want to monitor to receive updates about the most important events which happen in that topic. However, before implementing this web application, we need to perform some efficiency boosting operations on the code to minimize the resource allocation and time consumption to generate the output.

References

Avvenuti, M. *et al.* (2014) 'EARS (earthquake alert and report system)', in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '14*. New York, New York, USA: ACM Press, pp. 1749–1758. doi: 10.1145/2623330.2623358.

Bhatt, S. *et al.* (2017) 'Enhancing crowd wisdom using measures of diversity computed from social media data', in *Proceedings of the International Conference on Web Intelligence - WI '17*. New York, New York, USA: ACM Press, pp. 907–913. doi: 10.1145/3106426.3106491.

Bhatt, S. *et al.* (2018) 'Enhancing Crowd Wisdom Using Explainable Diversity Inferred from Social Media', in *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE, pp. 293–300. doi: 10.1109/WI.2018.00-77.

Blank, G. and Reisdorf, B. C. (2012) 'THE PARTICIPATORY WEB', *Information, Communication & Society*, 15(4), pp. 537–554. doi: 10.1080/1369118X.2012.665935.

Busch, M. *et al.* (2012) 'Earlybird: Real-Time Search at Twitter', in *2012 IEEE 28th International Conference on Data Engineering*. IEEE, pp. 1360–1369. doi: 10.1109/ICDE.2012.149.

Chae, B. (Kevin) (2015) 'Insights from hashtag #supplychain and Twitter Analytics: Considering Twitter and Twitter data for supply chain practice and research', *International Journal of Production Economics*. Elsevier, 165, pp. 247–259. doi: 10.1016/j.ijpe.2014.12.037.

Chakrabarti, D. and Punera, K. (2011) 'Event Summarization Using Tweets', in *Weblogs and Social Media*. Barcelona, Catalonia, Spain: AAAI, pp. 66–73. Available at: https://linkinghub.elsevier.com/retrieve/pii/S1558767312001103.

Chin, J. Y., Bhowmick, S. S. and Jatowt, A. (2017) 'TOTEM: Personal tweets summarization on mobile devices', *SIGIR 2017 - Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1305–1308. doi: 10.1145/3077136.3084138.

Clauset, A., Newman, M. E. J. and Moore, C. (2004) 'Finding community structure in very large networks', *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related*

Interdisciplinary Topics, 70(6), p. 6. doi: 10.1103/PhysRevE.70.066111.

Dana, J. *et al.* (2015) 'The composition of optimally wise crowds', *Decision Analysis*, 12(3), pp. 130–143. doi: 10.1287/deca.2015.0315.

De, D. and Dey, B. C. (2018) 'TBFN: Trust Based Friend Network Design by Analyzing User's Voice Call and SMS Pattern', *Wireless Personal Communications*. Springer US, 99(2), pp. 737–763. doi: 10.1007/s11277-017-5150-y.

Dehghani, N. and Asadpour, M. (2019) 'SGSG: Semantic graph-based storyline generation in Twitter', *Journal of Information Science*, 45(3), pp. 304–321. doi: 10.1177/0165551518775304.

Dutta, S. *et al.* (2015) 'A graph based clustering technique for tweet summarization', *2015 4th International Conference on Reliability, Infocom Technologies and Optimization: Trends and Future Directions, ICRITO 2015*. doi: 10.1109/ICRITO.2015.7359276.

Dutta, S. *et al.* (2019) 'Community Detection Based Tweet Summarization', in Abraham, A. et al. (eds) *Emerging Technologies in Data Mining and Information Security*. Singapore: Springer Singapore (Advances in Intelligent Systems and Computing), pp. 797–808. doi: 10.1007/978-981-13-1498-8_70.

Fan, C. *et al.* (2018) 'Social Network Mining for Recommendation of Friends Based on Music Interests', in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. Barcelona: IEEE, pp. 833–840. doi: 10.1109/ASONAM.2018.8508262.

Fellbaum, C. (2010) 'WordNet', in *Theory and Applications of Ontology: Computer Applications*. Dordrecht: Springer Netherlands, pp. 231–243. doi: 10.1007/978-90-481-8847-5_10.

Goel, P. and Thareja, R. (2018) 'Emotion Analysis of Twitter Data Using Hashtag Emotions', in Deka, G. C. et al. (eds) *Applications of Computing and Communication Technologies*. Delhi: Springer Singapore, pp. 88–98. doi: 10.1007/978-981-13-2035-4_9.

Hardy, H. *et al.* (2002) 'Cross-document summarization by concept classification', in *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '02*. New York, New York, USA: ACM Press,

p. 121. doi: 10.1145/564376.564399.

Huang, Y., Shen, C. and Li, T. (2018) 'Event summarization for sports games using twitter streams', *World Wide Web*. World Wide Web, 21(3), pp. 609–627. doi: 10.1007/s11280-017-0477-6.

Inouye, D. and Kalita, J. K. (2011) 'Comparing Twitter Summarization Algorithms for Multiple Post Summaries', in *2011 IEEE Third Int'l Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third Int'l Conference on Social Computing*. IEEE, pp. 298–306. doi: 10.1109/PASSAT/SocialCom.2011.31.

Jain, P. and Guru, S. K. (2016) 'Profile based personalized web search using Greedy Algorithms', *International Journal of Innovative Research in Computer and Communication Engineering*, 4(5), pp. 9068–9075. doi: 10.15680/IJIRCCE.2016.

Kaur, K. and Bansal, D. (2018) 'Techniques to Extract Topical Experts in Twitter: A Survey', in Chandra, S. and Joshi, S. (eds) *Information and Communication Technology for Intelligent Systems*. Singapore: Springer Singapore, pp. 391–399. doi: 10.1007/978-981-13-1742-2_38.

Khan, M. A. H. *et al.* (2013) 'Multi-tweet Summarization of Real-Time Events', in *2013 International Conference on Social Computing*. IEEE, pp. 128–133. doi: 10.1109/SocialCom.2013.26.

Kursuncu, U. *et al.* (2019) 'Predictive Analysis on Twitter: Techniques and Applications', in Agarwal, N., Dokoohaki, N., and Tokdemir, S. (eds) *Emerging Research Challenges and Opportunities in Computational Social Network Analysis and Mining*. Cham: Springer, pp. 67–104. doi: 10.1007/978-3-319-94105-9_4.

Lin, C.-Y. (2004) 'ROUGE: A Package for Automatic Evaluation of Summaries', in *Workshop on Text Summarization Branches Out*. Barcelona, Spain: Association for Computational Linguistics, pp. 74–81.

Louis, A. and Nenkova, A. (2013) 'Automatically Assessing Machine Summary Content Without a Gold Standard', *Computational Linguistics*, 39(2), pp. 267–300. doi: 10.1162/COLI_a_00123.

Mahan, J. E. *et al.* (2015) 'Exploring the impact of social networking sites on running involvement, running behavior, and social life satisfaction', *Sport Management Review,* 18(2), pp. 182–192. doi: 10.1016/j.smr.2014.02.006.

Mendoza, M., Poblete, B. and Valderrama, I. (2018) 'Early Tracking of People's Reaction in Twitter for Fast Reporting of Damages in the Mercalli Scale', in Meiselwitz, G. (ed.) *Social Computing and Social Media. Technologies and Analytics*. Las Vegas: Springer, pp. 247–257. doi: 10.1007/978-3-319-91485-5_19.

Mikolov, T. *et al.* (2013) 'Distributed Representations of Words and Phrases and their Compositionality', in *Neural information processing systems*. Lake Tahoe, Nevada, pp. 3111–3119. doi: 10.1162/jmlr.2003.3.4-5.951.

Moawad, I. F. and Aref, M. (2012) 'Semantic graph reduction approach for abstractive Text Summarization', in *2012 Seventh International Conference on Computer Engineering & Systems (ICCES)*. IEEE, pp. 132–138. doi: 10.1109/ICCES.2012.6408498.

Mohammad, S. M. and Kiritchenko, S. (2015) 'Using Hashtags to Capture Fine Emotion Categories from Tweets', *Computational Intelligence,* 31(2), pp. 301–326. doi: 10.1111/coin.12024.

Neto, J. L., Freitas, A. A. and Kaestner, C. A. A. (2002) 'Automatic Text Summarization Using a Machine Learning Approach', in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* pp. 205–215. doi: 10.1007/3-540-36127-8_20.

Nichols, J., Mahmud, J. and Drews, C. (2012) 'Summarizing sporting events using Twitter', *International Conference on Intelligent User Interfaces, Proceedings IUI,* pp. 189–198. doi: 10.1145/2166966.2166999.

Norvig, P. (2009) 'Natural Language Corpus Data', in *Beautiful Data: The Stories Behind Elegant Data Solutions*, pp. 219–242.

Oh, H. J., Ozkaya, E. and Larose, R. (2014) 'How does online social networking enhance life satisfaction? the relationships among online supportive interaction, affect, perceived social support, sense of community, and life satisfaction', *Computers in Human Behavior*. Elsevier Ltd, 30, pp. 69–78. doi: 10.1016/j.chb.2013.07.053.

Paskevicius, M., Veletsianos, G. and Kimmons, R. (2018) 'Content is King: An analysis of how the twitter discourse surrounding open education unfolded from 2009 to 2016', *International Review of Research in Open and Distance Learning*, 19(1), pp. 116–137.

Pennington, J., Socher, R. and Manning, C. (2014) 'Glove: Global Vectors for Word Representation', in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Stroudsburg, PA, USA: Association for Computational Linguistics, pp. 1532–1543. doi: 10.3115/v1/D14-1162.

Phan, H. T., Nguyen, N. T. and Hwang, D. (2018) 'A Tweet Summarization Method Based on Maximal Association Rules', in *Computational Collective Intelligence*. Bristol, UK: Springer International Publishing, pp. 373–382. doi: 10.1007/978-3-319-98443-8_34.

Radev, D. R. *et al.* (2004) 'Centroid-based summarization of multiple documents', *Information Processing and Management*, 40(6), pp. 919–938. doi: 10.1016/j.ipm.2003.10.006.

Radev, D. R. and Erkan, G. (2004) 'LexRank : Graph-based Centrality as Salience in Text Summarization', *Journal of Artificial Intelligence Research*, 22(1), pp. 457–479. Available at: https://arxiv.org/abs/1109.2128.

Robinson, B., Power, R. and Cameron, M. (2016) 'A sensitive Twitter earthquake detector', (September), pp. 999–1002. doi: 10.1145/2487788.2488101.

Rosvall, M. and Bergstrom, C. T. (2008) 'Maps of random walks on complex networks reveal community structure', *Proceedings of the National Academy of Sciences of the United States of America*. National Academy of Sciences, 105(4), pp. 1118–1123. doi: 10.1073/pnas.0706851105.

Sawhney, K., Prasetio, M. C. and Paul, S. (2017) 'Community Detection Using Graph Structure and Semantic Understanding of Text'.

ShafieiBavani, E. *et al.* (2018) 'Summarization Evaluation in the Absence of Human Model Summaries Using the Compositionality of Word Embeddings', *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 905–914. Available at: https://www.aclweb.org/anthology/C18-1077.

Sharifi, B., Hutton, M. A. and Kalita, J. K. (2010) 'Experiments in microblog summarization', in *Proceedings - SocialCom 2010: 2nd IEEE International Conference on*

Social Computing, PASSAT 2010: 2nd IEEE International Conference on Privacy, Security, Risk and Trust, pp. 49–56. doi: 10.1109/SocialCom.2010.17.

Sharma, V. and Dyreson, C. (2018) 'LINKSOCIAL: Linking User Profiles Across Multiple Social Media Platforms', in *2018 IEEE International Conference on Big Knowledge (ICBK)*. IEEE, pp. 260–267. doi: 10.1109/ICBK.2018.00042.

Shou, L. *et al.* (2013) 'Sumblr: Continuous Summarization of Evolving Tweet Streams', in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval - SIGIR '13*. New York, New York, USA: ACM Press, p. 533. doi: 10.1145/2484028.2484045.

Vanderwende, L. *et al.* (2007) 'Beyond SumBasic: Task-focused summarization with sentence simplification and lexical expansion', *Information Processing and Management*, 43(6), pp. 1606–1618. doi: 10.1016/j.ipm.2007.01.023.

XiaoHua, L. *et al.* (2012) 'Graph-Based Multi-Tweet Summarization using Social Signals.', *Coling*, 2(December 2012), pp. 1699–1714. Available at: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Graph-based+Multi-tweet+Summarization+Using+Social+Signals#0.

Xu, H. and Fang, H. (2018) 'StanceComp: Aggregating Stances from Multiple Sources for Rumor Detection', in Tseng, Y.-H. et al. (eds) *Information Retrieval Technology*. Taipei: Springer International Publishing, pp. 29–35. doi: 10.1007/978-3-030-03520-4_3.

Appendix

Tweet Collector Module Python Code

```
import sys
sys.path.insert(1, '../')
from os import listdir, makedirs
from os.path import isfile, join, exists
from modules.helper import Helper
from TwitterAPI import TwitterAPI, TwitterPager, TwitterRequestError
from urllib import parse
import json
import datetime

class TweetCollector:

    COLLECT_MODE_30DAYS = 'endpoint_30day'
    COLLECT_MODE_ARCHIVE = 'endpoint_archive'

    def __init__(self, group_name, root_path = './', strict_mode = True, collect_m
ode = COLLECT_MODE_30DAYS):
        self.collect_mode = collect_mode
        self.root_path = root_path.rstrip('/') + '/'
        self.helper = Helper(self.root_path)
        self.strict_mode = strict_mode
        self.group_name = group_name
        self.__initiate_api()

        if not exists('{}tweets'.format(self.root_path)):
            makedirs('{}tweets'.format(self.root_path))

    def initiate_collection(self, hashtags, toDate = None):
        self.hashtags = hashtags
        self.toDate = toDate

        if self.strict_mode:
            tweet_files = "{}tweets/{}".format(self.root_path, self.group_name)

            if not exists(tweet_files):
                makedirs(tweet_files)

            files = [f for f in listdir(tweet_files) if isfile(join(tweet_files, f
))]

            if len(files) > 0:
                print("You're in strict mode. Either enter non strict mode or dele
te tweet files under ./tweets/{} directory.".format(self.group_name))
```

```
                return

        try:
            self.__collect_tweets()
        except TwitterRequestError as e:
            print("Failed to fetch tweets. Check your API limitation in Twitter da
shboard.\n")

    def __initiate_api(self):
        consumer_key = self.helper.config_item('twitter_config.consumer_key')
        consumer_secret = self.helper.config_item('twitter_config.consumer_secret'
)
        access_token = self.helper.config_item('twitter_config.access_token')
        access_token_secret = self.helper.config_item('twitter_config.access_token
_secret')

        self.api = TwitterAPI(consumer_key, consumer_secret, access_token, access_
token_secret)

    def __collect_tweets(self):
        self.__prepare_request()

        print("Starting to save tweets...\n")
        count = 0
        temp_repo = ""

        for item in self.pager.get_iterator():
            if 'text' in item:
                temp_repo += json.dumps(item)
                temp_repo += "\n"
                count += 1

                if count % 100 == 0:
                    print("{} tweets already stored in file...\n".format(count))

                    dt = datetime.datetime.now()
                    file_name = '{}tweets/{}/{}{}_{}{}{}.wtr'.format(self.root_pat
h,
                        self.group_name, dt.strftime('%b'), dt.strftime('%d'),
                        dt.strftime('%H'), dt.strftime('%M'), dt.strftime('%S'))

                    f = open(file_name, "a+")
                    f.write(temp_repo)
                    f.close()
                    temp_repo = ""
            elif 'message' in item:
                print("Process Stoped:\n")
                print("{}: {}".format(item['code'], item['message']))
                break
            else:
                print("No Text Entry Detected:\n")
                print(item)
                break

        if len(temp_repo) > 0:
            print("{} tweets already stored in file...\n".format(count))

            dt = datetime.datetime.now()
            file_name = '{}tweets/{}/{}{}_{}{}{}.wtr'.format(self.root_path,
                self.group_name, dt.strftime('%b'), dt.strftime('%d'),
                dt.strftime('%H'), dt.strftime('%M'), dt.strftime('%S'))

            f = open(file_name, "a+")
            f.write(temp_repo)
            f.close()

    def __prepare_request(self):
        hash_combine = self.hashtags if(type(self.hashtags) is str) else " OR ".jo
in(self.hashtags)
        query = "({}) lang:en".format(hash_combine)
        endpoint = self.helper.config_item('twitter_config.{}'.format(self.collect
_mode))

        request_config = {
            'query': query,
            'maxResults': 100
        }

        if self.toDate != None:
            if not self.__validate_parameter(self.toDate, 'toDate'):
                raise Exception(self.validation_error)
            else:
                request_config['toDate'] = self.toDate

        self.pager = TwitterPager(self.api, endpoint, request_config)

    def __validate_parameter(self, value, category):
        output = False

        if category == 'toDate' or category == 'fromDate':
            if type(value) is not str:
                self.validation_error = 'toDate must be in string format'
            elif len(value) != 12:
                self.validation_error = 'toDate must be in yyyyMMddHHmm format'
            else:
                output = True
        else:
            self.validation_error = 'Provided parameter is not supported'

        return output
```

Database Interface Modules Python Code

```
import sys
sys.path.insert(1, '../')
from modules.helper import Helper
from pymongo import MongoClient
from pymongo.errors import DuplicateKeyError, WriteError
from os.path import isdir, isfile, join
from os import listdir
import json
import re
from langdetect import detect
from nltk import word_tokenize

class DBI:

    MODE_NOT_CHANGED = 0
    MODE_UPDATED = 1
    MODE_INSERTED = 2

    def __init__(self, root_path = './'):
        self.root_path = root_path.rstrip('/') + '/'
        self.helper = Helper(self.root_path)
        self.conf_name = 'mongodb_config'

        dsn = self.helper.config_item('{}.dsn'.format(self.conf_name))

        if type(dsn) is str and len(dsn) > 0:
            dbh = MongoClient(dsn)
        else:
            dbh = MongoClient(host=self.helper.config_item('{}.host'.format(se
lf.conf_name)),
                                    port= self.helper.config_item('{}.port'.fo
rmat(self.conf_name)),
                                    username= self.helper.config_item('{}.user
name'.format(self.conf_name)),
                                    password= self.helper.config_item('{}.pass
word'.format(self.conf_name)))

        self.db = dbh[self.helper.config_item('{}.db_name'.format(self.conf_na
me))]
```

```
    def insert(self, collection, document):
        collection = self.db[collection]

        if type(document) is dict:
            result = collection.insert_one(document)
            many = False
        elif type(document) is list:
            result = collection.insert_many(document)
            many = True
        else:
            result = None

        if result is not None and result.acknowledged is True:
            inserted_ids = result.inserted_ids if many is True else result.ins
erted_id
        else:
            inserted_ids = None

        status = False if inserted_ids is None else True

        return status, inserted_ids

    def upsert(self, collection, filter, document):
        collection = self.db[collection]
        status = False
        mode = None
        instance_id = None

        try:
            result = collection.replace_one(filter, document, upsert=True)

            if result.acknowledged:
                status = True

                if result.matched_count > 0:
                    mode = self.MODE_NOT_CHANGED if result.modified_count == 0
 else self.MODE_UPDATED
                    instance_id = result.upserted_id
                else:
                    mode = self.MODE_INSERTED
                    instance_id = result.upserted_id
        except DuplicateKeyError:
            del document['_id']
            return self.upsert(filter, document)
        except WriteError:
            pass

        return status, mode, instance_id
    def row_exists(self, collection, filter):
        collection = self.db[collection]
        result = collection.count_documents(filter)

        return True if result > 0 else False

    def find_all(self, collection, where, custom_fields = None):
        collection = self.db[collection]
        return collection.find(where, custom_fields) if type(custom_fields) is
 dict else collection.find(where)

    def update_one(self, collection, query, update):
        collection = self.db[collection]
        return collection.update_one(query, update)


class DBWrapper:

    def __init__(self, root_path='./'):
        self.root_path = root_path.rstrip('/') + '/'
        self.dbi = DBI(root_path=root_path)

    def populate_tweets(self, group_name, initial_hashtags = [], flush_output
= True, verbosity = True):
        tweets_path = '{}tweets/{}'.format(self.root_path, group_name)

        if not isdir(tweets_path):
            raise Exception("Tweet group does not exist")

        files = [f for f in listdir(tweets_path) if isfile(join(tweets_path, f
))]

        if len(files) == 0:
            raise Exception("There is no tweet file in the provided group")

        if flush_output:
            print("Starting to process {} files...\n".format(len(files)))

        files_processed = 0
        tweets_processed = 0
        tweets_stored = 0

        for f in files:
            files_processed += 1
            file_path = join(tweets_path, f)

            if flush_output:
                print("-----------
\nProcessing file #{}: {}\n".format(files_processed, f))

            if self.dbi.row_exists('files', {'file_name': f, 'category': group
_name}):
                if flush_output:
                    print("\tAlready processed.\n")
                continue

            fh = open(file_path, 'r', encoding='utf-8')
            content = fh.read().split("\n")
            fh.close()

            if flush_output:
                print("\tStarting to process {} tweets...\n".format(len(conten
t)))

            for line in content:
                if len(line) < 10:
                    continue

                tweets_processed += 1

                if flush_output:
                    print("\t***\n\tProcessing tweet #{}...\n".format(tweets_p
rocessed))

                tweet = json.loads(line)

                # Replace the tweet with the source if the instnace is only a
retweet
                if 'retweeted_status' in tweet and tweet['retweeted_status']:
                    if flush_output and verbosity:
                        print("\t\tRT => Fall back to the source tweet\n")
                    tweet = tweet['retweeted_status']

                if self.__process_tweet(tweet, group_name):
                    tweets_stored += 1

                # Process additional tweet if it matches our input hashtags se
t
                if 'quoted_status' in tweet:
                    status, new_tweet = self.__validate_quoted_tweet(tweet['qu
oted_status'], initial_hashtags)
                    if status:
                        tweets_processed += 1
                        if flush_output:
                            print("\t***\n\tProcessing tweet #{}... -
- Additional\n".format(tweets_processed))

                        if self.__process_tweet(new_tweet, group_name):
                            tweets_stored += 1

            # Saving file as a processed file
            self.dbi.insert('files', {'file_name': f, 'category': group_name})

        output = {
            'processed_files': files_processed,
            'processed_tweets': tweets_processed,
            'stored_tweets': tweets_stored
        }

        if flush_output:
            print("\nProcess Finished:\n{}\n".format(output))

        return output

    def __process_tweet(self, tweet, group_name, flush_output = True, verbosit
y = True):
        # Fetch the full text of the tweet
        if 'extended_tweet' in tweet and tweet['extended_tweet']:
            if flush_output and verbosity:
                print("\t\t Extended tweet\n")
            text = tweet['extended_tweet']['full_text']
        else:
            text = tweet['text']

        text = text.encode('utf-16', 'surrogatepass').decode('utf-16')

        # Fetch the tweet source
        pattern = re.compile("(\>)(.+)(\<)")
        source = pattern.search(tweet['source']).group(2)
        if flush_output and verbosity:
            print("\t\tTweeted using {}\n".format(source))

        if tweet['lang']:
            lang = tweet['lang']
            print("\t\tTweet language is {}\n".format(lang))
        else:
            lang = detect(text)
            if flush_output and verbosity:
                print("\t\tLanguage detected as {}\n".format(lang))

        document = {
            '_id': tweet['id_str'],
            'text': text,
            'lang': lang,
            'source': source,
            'category': group_name,
            'quotes': tweet['quote_count'],
            'replies': tweet['reply_count'],
            'faves': tweet['favorite_count'],
            'retweets': tweet['retweet_count'],
            'created_at': tweet['created_at'],
            'quoted_tweet': tweet['quoted_status_id_str'] if 'quoted_status_id
_str' in tweet else None,
            'user': {
                '_id': tweet['user']['id_str'],
                'name': tweet['user']['name'],
                'username': tweet['user']['screen_name'],
                'location': tweet['user']['location'],
                'verified': tweet['user']['verified'],
                'followers': tweet['user']['followers_count'],
                'followings': tweet['user']['friends_count'],
                'favourites': tweet['user']['favourites_count'],
                'statuses': tweet['user']['statuses_count']
            }
        }

        if flush_output:
            print("\t\tSaving the tweet... ")

        status, mode, record_id = self.dbi.upsert('tweets', {'_id': tweet['id_
str']}, document)

        altered = False

        if status:
            if flush_output:
                print("Done")

            if mode == self.dbi.MODE_INSERTED:
                altered = True
                if flush_output:
                    print(" - Inserted")
            elif mode == self.dbi.MODE_UPDATED:
                altered = True
                if flush_output:
                    print(" - Updated")
            elif mode == self.dbi.MODE_NOT_CHANGED and flush_output:
                print(" - No Change")
        elif flush_output:
            print("Failed")
        if flush_output:
            print("\n")

        return altered

    def __validate_quoted_tweet(self, new_tweet, initial_hashtags):
        if 'retweeted_status' in new_tweet and new_tweet['retweeted_status']:
            new_tweet = new_tweet['retweeted_status']

        if 'extended_tweet' in new_tweet and new_tweet['extended_tweet']:
            tweet_hashtags = new_tweet['extended_tweet']['entities']['hashtags
']
        else:
            tweet_hashtags = new_tweet['entities']['hashtags']

        hashtag_matched = False

        # Check initial hashtags in the tweet hashtag entities
        for th in tweet_hashtags:
            for h in initial_hashtags:
                if h.replace('#', '').lower() == th['text'].replace('#', '').l
ower():
                    hashtag_matched = True
                    break

            if hashtag_matched:
                break

        # If previous step failed, look for initial hashtags in the plain twee
t text
        if not hashtag_matched:
            new_text = new_tweet['extended_tweet']['full_text'] if 'extended_t
weet' in new_tweet else new_tweet['text']
            new_text = new_text.encode('utf-16', 'surrogatepass').decode('utf-
16').lower()
            for h in initial_hashtags:
                ih = h.replace('#', '').lower()
                if ih in new_text:
                    hashtag_matched = True
                    break

        return hashtag_matched, new_tweet

    def fetch_preprocessing_tweets(self, category = None):
        condition = {
            "lang": "en",
            "$or": [{'preprocessed_text': None}, {'ignored': 1}]
```

```python
        }

        if category:
            condition['category'] = category

        return self.dbi.find_all('tweets', condition)

    def set_processed_tweet_info(self, id, ignored, processed_text):
        query = {"_id": id}
        update = dict()

        if ignored:
            update["$set"] = {"ignored": 1}
        else:
            update["$set"] = {"preprocessed_text": processed_text}

        return self.dbi.update_one('tweets', query, update)

    def fetch_processed_tweets(self, category = None):
        condition = dict()
        condition['preprocessed_text'] = {'$exists': True}

        if category:
            condition['category'] = category

        result = []
        alt_id = 0
        for tweet in self.dbi.find_all('tweets', condition):
            tweet['alt_id'] = alt_id
            alt_id += 1

            result.append(tweet)

        return result
```

Pre-processing Module Python Code

```python
import sys
sys.path.insert(1, '../')
from modules.helper import Helper
import re
from string import punctuation
import spacy
from spellchecker import SpellChecker
from nltk import word_tokenize

class Preprocessor:
    def __init__(self, hashtags_list, root_path = './'):
        self.spell = SpellChecker()
        self.helper = Helper(root_path)
        self.slangs = self.helper.slang_hashmap()
        self.hashtags = [ht.strip().lower().replace('#', '') for ht in hashtag
s_list]

    def preprocess_tweet(self, input):
        self.tweet = input
        self.__remove_urls()
        self.__remove_usernames()
        self.__remove_non_latin()
        self.__remove_stopwords()
        self.__prune_slang_dictation()
        self.__remove_stopwords()
        self.__remove_special_chars()
        self.__final_prunning()

        return self.tweet.lower(), self.__ignore_tweet()

    def __remove_usernames(self):
        self.tweet = re.sub(r"(?=[^\w])\@\w+(?=[^\w]|$)", r"", self.tweet)

    def __remove_non_latin(self):
        emoji_pattern = re.compile("["
                                u"\U0001F600-\U0001F64F"  # emoticons
                                u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                                u"\U0001F680-\U0001F6FF"  # transport & map symbols
                                u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                                u"\U00002702-\U000027B0"
                                u"\U000024C2-\U0001F251"
                                "]+", flags=re.UNICODE)
        self.tweet = emoji_pattern.sub(r"", self.tweet)
        self.tweet = self.tweet.encode('ascii', 'ignore').decode('ascii')
        self.tweet = self.tweet.replace("&amp;", "&")

    def __remove_special_chars(self):
        self.tweet = self.tweet.replace('\n', ' ').replace('\r', '')
        self.tweet = re.sub(r"[^\w\s]", r"", self.tweet)

    def __remove_urls(self):
        self.tweet = re.sub(r"https://t.co/\w*", r"", self.tweet)

    def __remove_stopwords(self):
        nlp = spacy.load("en_core_web_sm")
        self.tweet = " ".join([token.text for token in nlp(self.tweet) if not
token.is_stop])
```

```
    def __prune_slang_dictation(self):
        words = word_tokenize(self.tweet)
        new_words = []

        for word in words:
            change = ''
            if self.__should_be_chacked_for_slang(word) and word.upper() in self.slangs:
                change = "Abbr: {} => {}".format(word, self.slangs[word.upper()])
                new_words.append(self.slangs[word.upper()])
            elif self.__should_be_chacked_for_correction(word):
                correct_word = self.spell.correction(word)

                if not word == correct_word:
                    if correct_word.upper() in self.slangs:
                        change = "Abbr Correction: {} => {}".format(word, self.slangs[correct_word.upper()])
                        new_words.append(self.slangs[correct_word.upper()])
                    else:
                        change = "Correction: {} => {}".format(word, correct_word)
                        new_words.append(correct_word)
                else:
                    new_words.append(word)
            else:
                new_words.append(word)

        self.tweet = " ".join(new_words)

    def __ignore_tweet(self):
        words_threshold = self.helper.config_item('global.words_threshold')
        words = word_tokenize(re.sub(r"[^\w\s]", r"", self.tweet))

        return True if len(words) < words_threshold else False

    def __should_be_chacked_for_slang(self, word):
        result = True

        exceptions = self.helper.config_item('global.abbr_exceptions')
        exceptions = [ex.strip().lower() for ex in exceptions.split(',')]

        if word.lower() in exceptions:
            result = False
        elif word.lower() in self.hashtags:
            result = False
        #Possibly a name
        elif word[0].isupper() and word[1:].islower():
            result = False
        elif len(word) < 2:
            result = False

        return result

    def __should_be_chacked_for_correction(self, word):
        result = True

        exceptions = self.helper.config_item('global.correction_exceptions')
        exceptions = [ex.strip().lower() for ex in exceptions.split(',')]
        uppercase_chars = [ch for ch in word if ch.isupper()]

        #Ignore words that has other than A to Z characters
        if not re.match(r"^[A-Za-z]$", word):
            result = False
        elif word.lower() in exceptions:
            result = False
        elif word.lower() in self.hashtags:
            result = False
        #Possibly a name
        elif word[0].isupper and word[1:].islower():
            result = False
        #Ignore word if it has more than 1 uppercase letter
        elif len(uppercase_chars) > 1:
            result = False

        return result

    def __final_prunning(self):
        self.tweet = re.sub(r"\b[0-9]+\b", r"", self.tweet)
        self.tweet = re.sub(r"\s+", r" ", self.tweet.strip().lower())
```

Embedding Modules Python Code

```
import sys
sys.path.insert(1, '../')
from gensim.models import Word2Vec
from scipy import spatial
import numpy
from modules.db_interface import DBWrapper
from infomap import Infomap
import networkx as nx
import community
from glove import Glove
from glove import Corpus

class Embedding:
    def __init__(self, root_path = './', embedding = 'word2vec', community = '
infomap',
        category = None, verbose = False):
        self.verbose = verbose
        self.dbw = DBWrapper(root_path=root_path)
        self.tweets = numpy.array(self.dbw.fetch_processed_tweets(category=cat
egory))
        self.communities = dict()

        if self.verbose:
            print("    -
{} tweet have been fetched & are processing".format(self.tweets.shape[0]))

        if embedding.lower() == 'word2vec':
            self.embedding_handler = MyWord2Vec()
        elif embedding.lower() == 'glove':
            self.embedding_handler = MyGloVe()
        else:
            raise Exception("Embedding method is not supported. Either use wor
d2vec or glove")

        if community.lower() == 'infomap':
            self.community_handler = MyInfomap()
        elif community.lower() == 'louvain':
            self.community_handler = MyLouvain(self.tweets.shape[0])
        else:
            raise Exception("Community method is not supported. Either use inf
omap or louvain")

    def run(self):
        if self.verbose:
            print("    -Initiating embeding model")
        self.embedding_handler.initiate_model(self.__prepare_model_corpus())

        if self.verbose:
            print("    -Preparing data for community check")
        self.__prepare_community_data()

        if self.verbose:
            print("    -Detecting communities")
        communities = self.community_handler.detect_communities()

        if self.verbose:
            print("    -Preparing the output")
        self.__prepare_output_communities(communities)

        if self.verbose:
            print("    -
{} communities detected".format(len(self.communities)))
        return self.communities

    def __prepare_model_corpus(self):
        corpus = []

        for tweet in self.tweets:
            corpus.append(tweet['preprocessed_text'].split())

        return corpus

    def __prepare_community_data(self):
        counter = 0

        for first_key in range(self.tweets.shape[0] - 1):
            counter += 1
            second_counter = 0

            for second_key in range(first_key + 1, self.tweets.shape[0]):
                weight = self.embedding_handler.cosine_similarity(
                    self.tweets[first_key]['preprocessed_text'],
                    self.tweets[second_key]['preprocessed_text'])

                if(weight > 0.75):
                    self.community_handler.add_network_edge(first_key, second_
key, weight)

                second_counter += 1
                if self.verbose:
                    edges_left = self.tweets.shape[0] - 1 - first_key - second
_counter
                    print("        -
Tweet {}/{}: {} edges left to add".format(counter,
                        self.tweets.shape[0] - 1, edges_left),
                        end="\n" if counter == self.tweets.shape[0] - 1 else "
\r")

    def __prepare_output_communities(self, communities):
        for community in communities:
            for index in communities[community]:
                if community in self.communities:
                    self.communities[community].append(self.tweets[index])
                else:
                    self.communities[community] = [self.tweets[index]]

class MyWord2Vec:
    def initiate_model(self, input_corpus):
        self.model = Word2Vec(sentences=input_corpus, min_count=1, size=100)
        self.word_indexes = set(self.model.wv.index2word)

    def cosine_similarity(self, first_text, second_text):
        first = self.__average_feature_vector(first_text)
        second = self.__average_feature_vector(second_text)

        return 1 - spatial.distance.cosine(first, second)

    def __average_feature_vector(self, text):
        words = text.split()
        words_no = 0
        feature_vector = numpy.zeros((100, ), dtype="float32")

        for word in words:
            if word in self.word_indexes:
                words_no += 1
                feature_vector = numpy.add(feature_vector, self.model[word])

        if words_no > 0:
            feature_vector = numpy.divide(feature_vector, words_no)

        return feature_vector

class MyGloVe:
    def initiate_model(self, input_corpus):
        self.corpus_model = Corpus()
        self.corpus_model.fit(self.__read_corpus(input_corpus), window=10)

        self.glove = Glove(no_components=100, learning_rate=0.05)
        self.glove.fit(self.corpus_model.matrix, epochs=200)
        self.glove.add_dictionary(self.corpus_model.dictionary)

    def cosine_similarity(self, first_text, second_text):
        first = self.__average_feature_vector(first_text)
        second = self.__average_feature_vector(second_text)

        return 1 - spatial.distance.cosine(first, second)

    def __read_corpus(self, input_corpus):
        for line in input_corpus:
            yield line

    def __average_feature_vector(self, text):
        words = text.split()
        words_no = 0
        feature_vector = numpy.zeros((100, ), dtype="float32")
        for word in words:
            if word in self.glove.dictionary:
                word_idx = self.glove.dictionary[word]
                words_no += 1
                feature_vector = numpy.add(feature_vector, self.glove.word_vec
tors[word_idx])

        if words_no > 0:
            feature_vector = numpy.divide(feature_vector, words_no)

        return feature_vector

class MyInfomap:
    def __init__(self):
        self.handler = Infomap("--two-level")

    def add_network_edge(self, first_id, second_id, weight = 1.00):
        self.handler.addLink(first_id, second_id, weight)

    def detect_communities(self):
        self.handler.run()
        communities = {}

        for node in self.handler.iterTree():
            if node.isLeaf():
                if node.moduleIndex() in communities:
                    communities[node.moduleIndex()].append(node.physicalId)
                else:
                    communities[node.moduleIndex()] = [node.physicalId]

        return communities

class MyLouvain:
    def __init__(self, nodes_count):
        self.graph = nx.Graph()
        self.graph.add_nodes_from([n for n in range(nodes_count)])

    def add_network_edge(self, first_node, second_node, weight = None):
        self.graph.add_edge(first_node, second_node)

    def detect_communities(self):
        partition = community.best_partition(self.graph)
        communities = dict()

        for node in partition:
            group = partition[node]
            if group in communities:
                communities[group].append(node)
            else:
                communities[group] = [node]

        return communities
```

Summarizing Module Python Code

```
import sys
sys.path.insert(1, '../')
from sklearn.feature_extraction.text import TfidfVectorizer
from modules.helper import Helper
from math import ceil

class Summarize:
    def __init__(self, communities, root_path = './'):
        self.helper = Helper(root_path)
        self.communities = communities
        self.summarized = []

    def run(self):
        for community in self.communities:
            sentences = [tweet['preprocessed_text'] for tweet in self.communit
ies[community]]
            vectorize = TfidfVectorizer()
            tfidfs = vectorize.fit_transform(sentences)
            aggregate_tfidf = self.__populate_tweet_tfidf(tfidfs, len(sentence
s), self.communities[community])
            self.__select_most_representative(aggregate_tfidf, self.communitie
s[community])
        return self.summarized

    def __populate_tweet_tfidf(self, tfidfs, doc_length, tweets):
        result = dict()

        for doc in range(doc_length):
            score = 0
            feature_index = tfidfs[doc,:].nonzero()[1]
            tfidf_scores = zip(feature_index, [tfidfs[doc, x] for x in feature
_index])
            for s in [s for (i, s) in tfidf_scores]:
                score += s

            score += self.__compute_tweet_additional_score(tweets[doc])

            result[doc] = score
        result = {key: val for key, val in sorted(result.items(), key=lambda i
tem: item[1], reverse=True)}
        return result

    def __compute_tweet_additional_score(self, tweet):
        score = self.helper.config_item('scoring.verified', 1) if tweet['user'
]['verified'] else 0

        faves = tweet['faves']
        rt = tweet['retweets']
        fave_rt_const = self.helper.config_item('scoring.faves_rt_constant', 0
.0005)
        followings = tweet['user']['followings'] if tweet['user']['followings'
] > 0 else 1
        followers = tweet['user']['followers']
        popularity_const = self.helper.config_item('scoring.popularity_constan
t', 0.001)
        word_count = len(tweet['preprocessed_text'].split())
        word_count_constant = self.helper.config_item('scoring.tweet_length_co
nstant', 0.001)

        score += (faves + rt) * fave_rt_const
        score += (followers - followings) * popularity_const
        score += word_count * word_count_constant

        return score

    def __select_most_representative(self, scores, tweets):
        community_representatives = []
        selection_share = self.helper.config_item('global.representative_share
', 0.001)
        selection_threshold = ceil(len(tweets) * selection_share)
        counter = 0

        print("    -
Selecting {} tweet from community".format(selection_threshold))

        for chosen_index in scores:
            counter += 1
            community_representatives.append(tweets[chosen_index])

            if counter > selection_threshold:
                break

        self.summarized.append(community_representatives)
```

Evaluation Modules Python Code

```
import sys
sys.path.insert(1, '../')
from gensim.models import Word2Vec
from modules.db_interface import DBWrapper
from gensim.models import Word2Vec
from scipy.spatial.distance import jensenshannon
from nltk import FreqDist
from numbers import Number

class WordMovers:
    def __init__(self, root_path='./'):
        self.dbw = DBWrapper(root_path=root_path)
        self.category = None


    def run(self, summarized, category):
        self.summarized = summarized

        if category != self.category:
            print("New Category")
            self.category = category
            self.tweets = self.dbw.fetch_processed_tweets(category=category)

            sents = [tweet['preprocessed_text'].split() for tweet in self.twee
ts]

            print("Creating Embedding Model")
            self.embedding = Word2Vec(sentences=sents, min_count=1, size=100)
            self.w2v_vocab = set(self.embedding.wv.index2word)

        print("Calculating Summary Score")
        self.__calculate_summary_score()
        print("")

        return self.scores


    def __calculate_summary_score(self):
        self.scores = dict()
        summaries_score = 0
        summaries_count = 0

        community_counter = 0

        for community in self.summarized:
            community_counter += 1
            summary_counter = 0
            for summary in community:
                summary_counter += 1
                doc_count = 0
                temp_score = 0

                for tweet in self.tweets:
                    if tweet['_id'] != summary['_id']:
                        temp_score += self.embedding.wmdistance(
                            summary['preprocessed_text'], tweet['preprocessed_
text']
                        )
                        doc_count += 1

                        print("  Community {}/{} | Summary {}/{} | Checking {}
/{} | Summary Avg. Score {}".format(
                            community_counter,
                            len(self.summarized),
                            summary_counter,
                            len(community),
                            doc_count,
                            len(self.tweets),
                            temp_score/doc_count
                        ), end="\r")

                temp_score = 0 if doc_count == 0 else (temp_score / doc_count)
                summaries_count += 1
                summaries_score += temp_score
                self.scores[summary['_id']] = temp_score

        summaries_score = 0 if summaries_count == 0 else (summaries_score / su
mmaries_count)
        self.scores['total_score'] = summaries_score

class JensenShannon:
    def __init__(self, root_path='./'):
        self.dbw = DBWrapper(root_path=root_path)
        self.category = None
        self.freqs = dict()
        self.vocabs = 1
        self.scores = dict()

    def run(self, summarized, category):
        if category != self.category:
            print("New Category")
            self.category = category
            self.summarized = summarized
            self.tweets = self.dbw.fetch_processed_tweets(category=category)
            print("Calculate vocabularies and frequencies")
            self.__initiate_frequencies()

        print("Calculate Scores")
        self.__calculate_scores()
        print("")

        return self.scores

    def __initiate_frequencies(self):
        lines = [tweet['preprocessed_text'] for tweet in self.tweets]
        lines = " ".join(lines)

        self.freqs = FreqDist(lines.split())
        self.vocabs = len(self.freqs)

    def __calculate_scores(self):
        community_count = 0
        community_score = 0
        c_counter = 0

        for community in self.summarized:
            c_counter += 1
            s_counter = 0
            for summary in community:
                s_counter += 1
                summary_prob_dist = self.__probabbility_dist(summary['preproce
ssed_text'])
                summary_score = 0
                summary_count = 0
                t_counter = 0

                for tweet in self.tweets:
                    t_counter += 1
                    if tweet['_id'] != summary['_id']:
                        tweet_prob_dist = self.__probabbility_dist(tweet['prep
rocessed_text'])
                        summary_prob_dist, tweet_prob_dist = self.__balance_pr
obabilities(
                            summary_prob_dist, tweet_prob_dist
                        )
                        temp_score = jensenshannon(summary_prob_dist, tweet_pr
ob_dist)

                        if isinstance(temp_score, Number):
                            summary_score += temp_score
                            summary_count += 1
                    print("  Community {}/{} | Summary {}/{} | Tweet {}/{}".fo
rmat(
                        c_counter, len(self.summarized),
                        s_counter, len(community),
                        t_counter, len(self.tweets)
                    ), end="\r")

                if summary_count > 0:
                    self.scores[summary['_id']] = summary_score / summary_coun
t

                    community_score += (summary_score / summary_count)
                    community_count += 1
                else:
                    self.scores[summary['_id']] = 0

        if community_count > 0:
            self.scores['total_score'] = community_score / community_count
        else:
            self.scores['total_score'] = 0



    def __probabbility_dist(self, sentence):
        probability_dist = list()

        for word in sentence.split():
            if word in self.freqs:
                probability_dist.append(self.freqs[word]/self.vocabs)
            else:
                probability_dist.append(0)

        return probability_dist

    def __balance_probabilities(self, summary, tweet):
        summary_len = len(summary)
        tweet_len = len(tweet)

        if summary_len < tweet_len:
            for i in range(summary_len, tweet_len):
                summary.append(0)
        elif tweet_len < summary_len:
            for i in range(tweet_len, summary_len):
                tweet.append(0)

        return summary, tweet
```