FINAL YEAR PROJECT REPORT ON

"SENTIMENT ANALYSIS & EVENT SUMMARIZATION WITH CROWDSOURCED SOCIAL MEDIA DATA"

Submitted in partial fulfilment of the requirements

For the award of the degree

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

SUBMITTED BY:

PAPUL GHOSH

UNDER THE SUPERVISION OF:

DR. SRINKA BASU

DEPARTMENT OF ENGINEERING & TECHNOLOGICAL STUDIES

UNIVERSITY OF KALYANI

YEAR: 2019

CERTIFICATE OF APPROVAL

DEPARTMENT OF ENGINEERING AND TECHNOLOGICAL STUDIES

UNIVERSITY OF KALYANI

This is to certify that the project report on **"Sentiment Analysis & Event Summarization With Crowdsourced Social Media Data"** has been submitted for the partial fulfilment of the required degree of Bachelor of Technology in Information Technology is an original work carried by **PAPUL GHOSH** (Registration No-**100044** of **2015-2016)** of the **DEPARTMENT OF ENGINEERING AND TECHNOLOGICAL STUDIES** of **UNIVERSITY OF KALYANI** under the supervision of the undersigned.

The aforesaid project is hereby approved as a credible study of Engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite for the degree of which it has been submitted. It is understood by this approval that the undersigned does not necessarily endorse or approve any statement made opinion expressed or opinion drawn there in, but approves the project report only for the purpose for which it is submitted.

_____

Head of the Department

Department of Engineering
& Technological Studies

University of Kalyani

_____

Dr. Srinka Basu

Assistant Professor

Department of Engineering
& Technological Studies

University of Kalyani

PROJECT REPORT

ON

SENTIMENT ANALYSIS & EVENT SUMMARIZATION WITH CROWDSOURCED SOCIAL MEDIA DATA

FOR PARTIAL FULFILMENT FOR THE DEGREE OF

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

SUBMITTED BY

_____

PAPUL GHOSH (100044 of 2015-2016)

DATE _____

PLACE _____

ACKNOWLEDGEMENT

I would like to convey my heartfelt thanks of gratitude to our project supervisor Dr. Srinka Basu for the kind of support I have got throughout the project work. I am highly indebted to her for the valuable suggestions and guidance she provided without which this project could not be completed successfully. I would like to extend our sincere thanks to all who helped us in justifying our effort in this project.

Abstract

Social media sites such as Twitter and Facebook are rich sources of text examples expressing positive and negative sentiment. Many public and private sectors are interested in extracting information regarding opinions that consist of subjective expressions across a variety of products or services. Microblogging sites have millions of people sharing their thoughts daily because of its characteristic short and simple manner of expression. During the sudden onset of a crisis situation, affected people post useful information on Twitter that can be used for situational awareness and other humanitarian disaster response efforts, if processed timely and effectively. Processing social media information pose multiple challenges such as learning information categories from the incoming stream of messages and classifying them into different classes among others. In this project we applied some of the machine learning algorithms to the collected twitter dataset on the unfortunate Nepal Earthquake incident to analyse the sentiment of each tweet and classify its polarity as positive, negative or neutral. Based on the relative similarities present in the tweets and the calculated sentiments we have clustered the tweets in multiple groups having their own specific categories of objectives. Thus, summarised information can be extracted from huge number of blogs much efficiently in a faster way that in turn may be fruitful in disaster management.

Table of Contents

Chapter 1: Introduction

Social media– mobile and web-based applications that allow people to communicate and share information across multiple platforms–is experiencing rapid growth and is being adopted by many. How and why such technology diffuses is a question of current importance, as it is adding new dimensions to human interaction. My research addresses how social media is being used in emergency and mass convergence situations, where time frames are often compressed and routine life is disrupted or changed in some fashion. My interest is in understanding the summarised information from huge number of tweets posted during these non-routine situations.

Twitter[1] is a micro-blogging service that allows its users to share short messages up to 280 characters in length with each other. These short messages are referred to as *tweets* and can be sent and retrieved across a wide variety of media including email, text messaging, instant messaging, the Internet, and other third-party applications. Users may choose to share their tweets publicly with anyone, or restrict access to their tweets so that only users they give permission may view them.

Launched in October 2006, Twitter is estimated to now have over 261 million user accounts[2]. It is also ranked number 11 in global internet engagement, with it being ranked the most popular micro-blogging service[3].

This project focuses on clustering tweets that might help in Disaster Management in multiple groups having their own specific categories of objectives based on the relative similarities present among the tweets and also based on the calculated sentiments. Each group of tweets have some common motive or intention regarding the event.

The scope of this project is:

[1] https://twitter.com
[2] https://www.omnicoreagency.com/twitter-statistics/. Retrieved on 7th September, 2019
[3] https://www.alexa.com/siteinfo/twitter.com. Retrieved on 7th September, 2019

i. To analyse the sentiments from a given set of tweet data.
ii. To summarise the sentiments of the tweet data to extract significant summarised information.

1.1. Sentiment Analysis

Sentiment analysis is a type of data mining that measures the inclination of people's opinions through natural language processing (NLP), computational linguistics and text analysis, which are used to extract and analyze subjective information from the Web - mostly social media and similar sources. The analyzed data quantifies the general public's sentiments or reactions toward certain products, people or ideas and reveal the contextual polarity of the information. Sentiment analysis is also known as opinion mining.

Many public and private sectors are interested in extracting information regarding opinions that consist of subjective expressions across a variety of products or services. This helps companies determine strategies for improving the quality of their products or to assist decision makers. The main purpose of sentiment analysis is to conclude positive sentiments or negative sentiments from given opinionated text. We explore a number of questions in relation to the sentiment analysis problem. First, we examine dataset preprocessing specific to the natural language domain of tweets. We then evaluate a number of baseline linear models for sentiment analysis. Finally, we attempt to improve on the performance of our baseline models using annotated dictionary initialized with linear model weights. All the algorithms we consider in this project are supervised methods over unigram and bigram features.

In marketing field companies use it to develop their strategies, to understand customers' feelings towards products or brand, how people respond to their campaigns or product launches and why consumers don't buy some products.

Sentiment analysis also is used to monitor and analyse social phenomena, for the spotting of potentially dangerous situations and determining the general mood of the blogosphere.

1.2. Sentiment Summarization

With Twitter's ability to send messages with mobile devices and easily broadcast those messages to a wide audience, it would seem to be a natural fit for use during mass convergence and crisis events (provided that the service is available). During the onset of a crisis, a variety of information is posted in real-time by affected people; by people who are in need of help (e.g., food, shelter, medical assistance, etc.) or by people who are willing to donate or offer volunteering services. Moreover, humanitarian and formal crisis response organizations such as government agencies, public health care NGOs, and military are tasked with responsibilities to save lives, reach people who need help, etc. Situation-sensitive requirements arise during such events and formal disaster response agencies look for actionable and tactical information in real-time to effectively estimate early damage assessment, and to launch relief efforts accordingly.

On April 25th 2015, just before noon, Nepal experienced an earthquake of magnitude 7.8 on the moment magnitude scale. The earthquake ripped through Kathmandu valley, and a series of aftershocks levelled entire villages.

Immediately after the earthquake, volunteers from around the world were instrumental in guiding emergency operations, using satellite imagery to identify infrastructure destruction throughout the region.

However, people on the ground in Nepal were also generating tremendous amounts of information which could be of use to rescue operations, albeit less directly: on twitter. Between April 25th and May 28th, 33610 tweets were tweeted by people in Nepal. These tweets were full of useful information, but 33,610 tweets is simply too many for a rescue operation to comb through. This project concentrates on extracting summarised information on from the dataset by evaluating their TF-IDF scores and sentiments.

Chapter 2: Literature Review

This section provides an overview of terminology, methods, and limitations related to the existing systems.

2.1. Sentiment Analysis:

Sentiment analysis is the automated process of understanding an opinion about a given subject from written or spoken language.

A sentiment analysis system for text analysis combines natural language processing (NLP) and machine learning techniques to assign weighted sentiment scores to the entities, topics, themes and categories within a sentence or phrase. Sentiment analysis helps data analysts within large enterprises gauge public opinion, conduct nuanced market research, monitor brand and product reputation, and understand customer experiences.

Basic sentiment analysis of text documents follows a straightforward process:

1. Break each text document down into its component parts (sentences, phrases, tokens and parts of speech)
2. Identify each sentiment-bearing phrase and component
3. Assign a sentiment score to each phrase and component (-1 to +1)
4. Optional: Combine scores for multi-layered sentiment analysis

for a simple explanation of sentiment analysis, consider these sentences:

→Terrible pitching and awful hitting led to another crushing loss.

→Bad pitching and mediocre hitting cost us another close game.

Both sentences discuss a similar subject, the loss of a baseball game. But we reading them, can clearly see that first sentence's tone is much more negative.

Our brain figures this out by looking for and interpreting sentiment-bearing phrases – that is, words and phrases that carry a tone or opinion. These usually appear as adjective-noun combinations. In the examples above, the sentiment-bearing phrases are:

→ Terrible pitching | awful hitting | crushing loss

→ Bad pitching | mediocre hitting | close game

Our have encountered words like these many thousands of times over our lifetime across a range of contexts. And from these experiences, we've learned to understand the strength of each adjective, receiving input and feedback along the way from teachers and peers.

There are various classifier algorithms available to analyse the sentiments.

a. Naïve Bayes Classifier:

Naive Bayes is a popular algorithm for classifying text. Although it is fairly simple, it often performs as well as much more complicated solutions.

Given the dependent feature vector $(x_1, \ldots, x_n)$ and the class C_k. Bayes' theorem is stated mathematically as the following relationship:

$$P(C_k \mid x_1, \ldots, x_n) = \frac{P(C_k)P(x_1, \ldots, x_n \mid C_k)}{P(x_1, \ldots, x_n)}$$

$$P(C_k \mid x_1, \ldots, x_n) = \frac{P(C_k)P(x_1, \ldots, x_n \mid C_k)}{P(x_1, \ldots, x_n)}$$

According to the "naive" conditional independence assumptions, for the given class C_k each feature of vector x_i is conditionally independent of every other feature x_j for $i \neq j$..

b. K Nearest Neighbors:

KNN algorithm is used to classify by finding the **K** nearest matches in training data and then using the label of closest matches to predict. Traditionally, distance such as euclidean is used to find the closest match. We'll identify the **K** nearest neighbors which has the highest similarity score among the training corpus.

The **Euclidean distance** between points **p** and **q** is the length of the line segment connecting pq.

In Cartesian coordinates, if **p** = (p_1, p_2,..., p_n) and **q** = (q_1, q_2,..., q_n) are two points in Euclidean n-space, then the distance (d) from **p** to **q**, or from **q** to **p** is given by the Pythagorean formula:

$$d(p,q) = d(q,p) = \sqrt{(p_1 - q_1)^2 + (p_1 - q_1)^2 + \cdots + (p_n - q_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

K-means clustering is one of the most widely used unsupervised machine learning algorithms that forms clusters of data based on the similarity between data instances. For this particular algorithm to work, the number of clusters has to be defined beforehand. The K in the K-means refers to the number of clusters.

The K-means algorithm starts by randomly choosing a centroid value for each cluster. After that the algorithm iteratively performs three steps: (i) Find the Euclidean distance between each data instance and centroids of all the clusters; (ii) Assign the data instances to the cluster of the centroid with nearest distance; (iii) Calculate new centroid values based on the mean values of the coordinates of all the data instances from the corresponding cluster.

c. Support Vector Machines (or SVM):

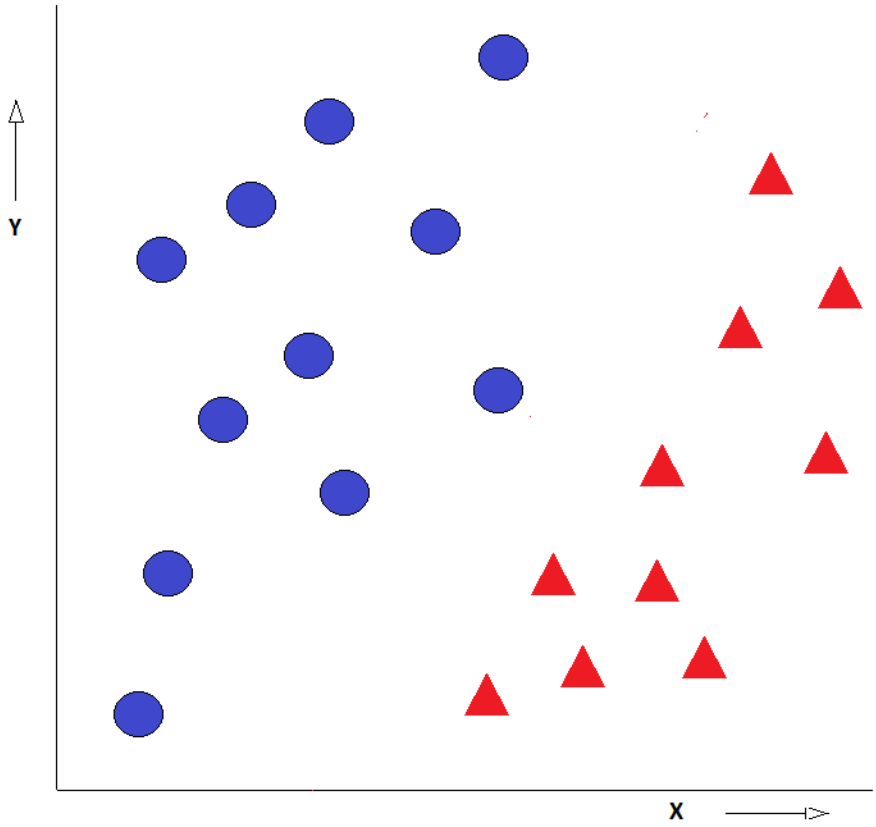
Support vector machines is an algorithm that determines the best decision boundary between vectors that belong to a given group (or category) and vectors that do not belong to it. That's it. It can be applied to any kind of vectors which encode any kind of data. This means that in order to

leverage the power of SVM text classification, texts have to be transformed into vectors.

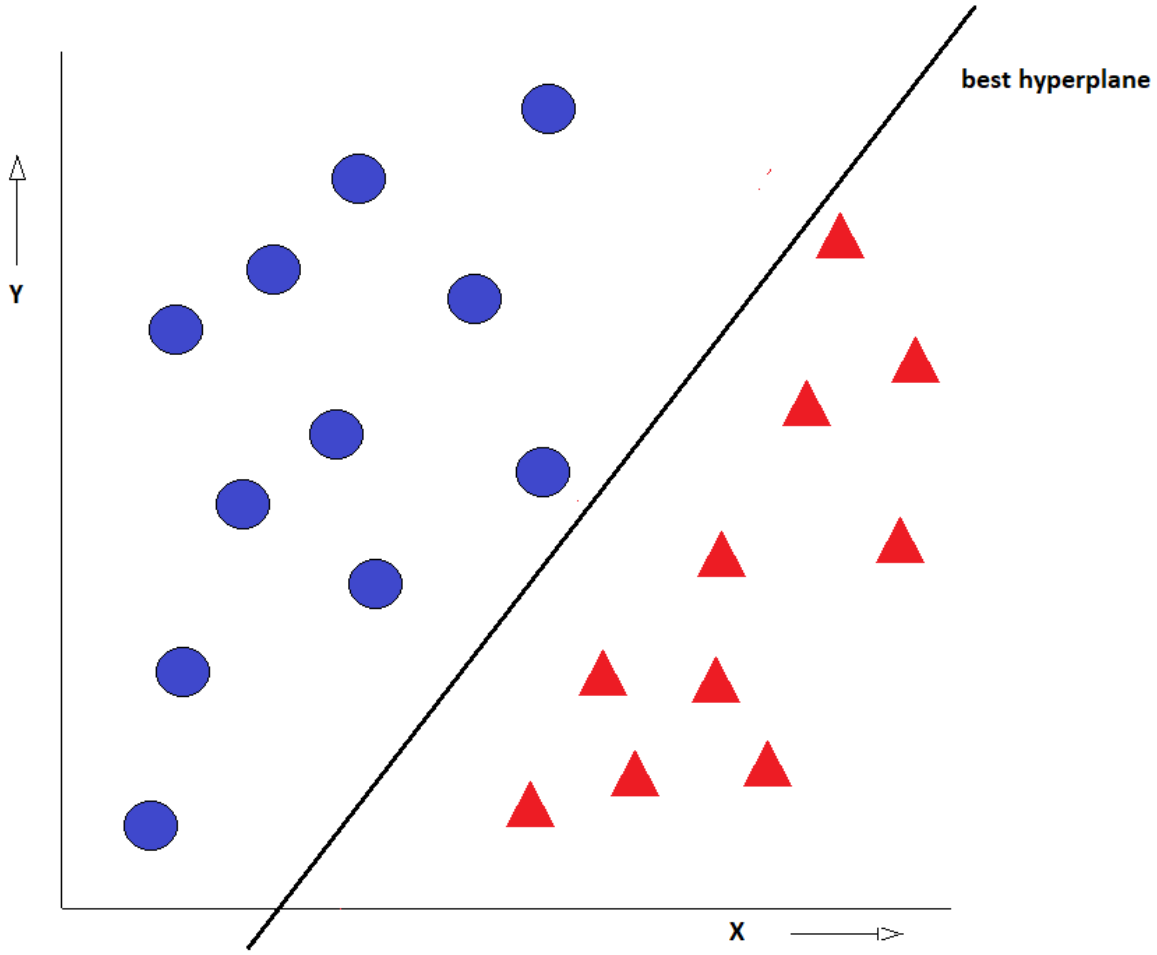
Now, what are vectors? Vectors are (sometimes huge) lists of numbers which represent a set of coordinates in some space. So, when SVM determines the decision boundary we mentioned above, SVM decides where to draw the best "line" (or the best hyperplane) that divides the space into two subspaces: one for the vectors which belong to the given category and one for the vectors which do not belong to it.

So, provided we can find vector representations which encode as much information from our texts as possible, we will be able to apply the SVM algorithm to text classification problems and obtain very good results.

Say, for example, the blue circles in the graph below are representations of training texts which talk about the *Pricing* of a SaaS Product and the red triangles are representations of training texts which do not talk about that. What would the decision boundary for the *Pricing* category look like?



best decision boundary would look like this:



Now that the algorithm has determined the decision boundary for the category we want to analyze, we only have to obtain the representations of all of the texts we would like to classify and check what side of the boundary those representations fall into.

2.2 Sentiment Summarisation:

2.2.1. Crowdsourcing:

In some domains, the process of sentiment analysis has more-or-less already been solved. If we want to know if a given Twitter message (tweet) is angry or happy, there's probably a blanket solution to make that determination (so long as the tweets are in English). Many companies have already trained systems on basic social media sentiment detection in English.

Therefore crowdsourcing becomes necessary when:

- A language is being analyzed that hasn't previously been analyzed at scale (for example, a company aiming to create a social media sentiment analysis solution for Greek or Hebrew)
- A topic being analyzed is unique (maybe a company wants to analyze the sentiment of very complicated and robust articles and essays about politics in the Middle East)

In the cases above, unique terms and entities and words need to be labelled and analysed en masse in order to get a machine to make sense of them well. This often requires native speakers of the language in question, and it could also involve some degree of contextual knowledge about the domain (in the case of middle eastern politics).

2.2.2. TF-IDF Score calculation:

TF-IDF stands for *term frequency-inverse document frequency*, and the tf-idf weight is a weight often used in information retrieval and text mining. This weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. Variations of the tf-idf weighting scheme are often used by search engines as a central tool in scoring and ranking a document's relevance given a user query.

One of the simplest ranking functions is computed by summing the tf-idf for each query term; many more sophisticated ranking functions are variants of this simple model.

Tf-idf can be successfully used for stop-words filtering in various subject fields including text summarization and classification.

How to Compute:

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), aka. the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

- **TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

  TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

- **IDF: Inverse Document Frequency**, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

  IDF(t) = log_e (Total number of documents / Number of documents with term t in it).

Finally,

$$\text{TF-IDF(t)} = \text{TF(t)} * \text{IDF(t)}$$

See below for a simple example.

Example:

Consider a document containing 100 words wherein the word *cat* appears 3 times. The term frequency (i.e., tf) for *cat* is then (3 / 100) = 0.03. Now, assume we have 10 million documents and the word *cat* appears in one

thousand of these. Then, the inverse document frequency (i.e., idf) is calculated as log(10,000,000 / 1,000) = 4. Thus, the Tf-idf weight is the product of these quantities: 0.03 * 4 = 0.12.

2.2.3. cosine Similarity Metric:

The similarity is the very basic building block for activities such as Recommendation engines, clustering, classification and anomaly detection. The similarity measure is the measure of how much alike two data objects are. Similarity measure in a data mining context is a distance with dimensions representing features of the objects. If this distance is small, it will be the high degree of similarity where large distance will be the low degree of similarity.

The similarity is subjective and is highly dependent on the domain and application. For example, two fruits are similar because of color or size or taste. Care should be taken when calculating distance across dimensions/features that are unrelated. The relative values of each element must be normalized, or one feature could end up dominating the distance calculation. Similarity scores ranges from 0 to 1 [0,1].

Cosine similarity metric finds the normalized dot product of the two attributes. By determining the cosine similarity, we would effectively try to find the cosine of the angle between the two objects. The cosine of 0° is 1, and it is less than 1 for any other angle.

It is thus a judgement of orientation and not magnitude: two vectors with the same orientation have a cosine similarity of 1, two vectors at 90° have a similarity of 0, and two vectors diametrically opposed have a similarity of -1, independent of their magnitude.

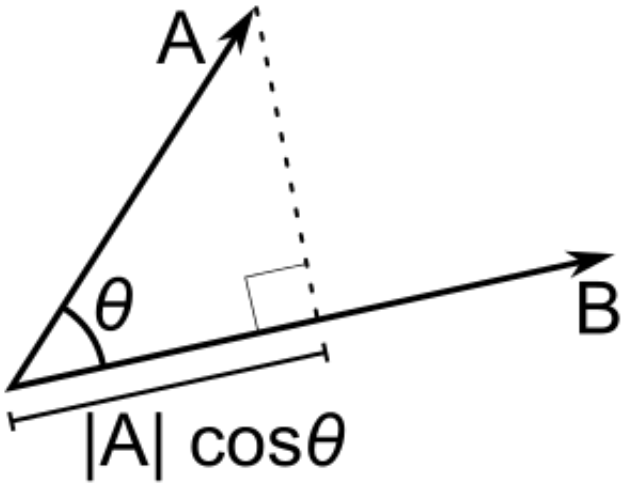
Cosine similarity is particularly used in positive space, where the outcome is neatly bounded in [0,1]. One of the reasons for the popularity of cosine similarity is that it is very efficient to evaluate, especially for sparse vectors.

$$\text{Sim}(A,B)=\cos\theta = \frac{\vec{A}.\vec{B}}{|\vec{A}|.|\vec{B}|}$$

Where the geometric definition of the dot product:

$$\vec{A}.\vec{B} = |\vec{A}|.|\vec{B}|.\cos\theta$$

This term $|\vec{B}|.\cos\theta$ is the projection of the vector $\vec{a}$ into the vector $\vec{b}$ as shown on the image below:



*source: https://www.lexalytics.com

2.2.4. Clustering:

Cluster is a group of objects that belongs to the same class. In other words, similar objects are grouped in one cluster and dissimilar objects are grouped in another cluster.

Clustering-based sentiment analysis is a novel approach for analyzing opinions expressed in reviews, comments or blogs. In contrast to the two traditional mainstream approaches (supervised learning and symbolic techniques), the clustering-based approach is able to produce basically accurate analysis results without any human participation, linguist knowledge or training time.

- A cluster of data objects can be treated as one group.
- While doing cluster analysis, we first partition the set of data into groups based on data similarity and then assign the labels to the groups.
- The main advantage of clustering over classification is that, it is adaptable to changes and helps single out useful features that distinguish different groups.
- Clustering analysis is broadly used in many applications such as market research, pattern recognition, data analysis, and image processing.

Chapter 3: Sentiment Analysis

3.1. Dataset

We use the dataset on a Disney movie, which is publicly available on Kaggle.com. Here is a description of the data, provided by Kaggle:

> The labelled data set consists of 25,000 IMDB movie reviews, specially selected for sentiment analysis. Each of these rows contain a review on the movie followed by a label which I either 0 or 1. The sentiment of reviews is binary, meaning the IMDB negative results in a sentiment score of 0, and positive rating have a sentiment score of 1. The 25,000 review set consists of equal number of positive and negative sentiment reviews that is, both counts 12500. Furthermore for assisting the dictionary based classifier we are using two annotated positive and negative wordlist. The positive wordlist contains 2007 distinct positive words and the count for negative word is 4783.

3.2. Proposed Method

The steps to be followed for analysing the sentiments are as shown here:

- **Preprocessing:**

  Before start processing the tweets, the initial step is to preprocess each tweet to get rid of the useless terms present. The data is cleaned by first creating a function clean() which receives each tweet from the main program and pre-process the words.

  Stopwords (using the Python Natural Language Toolkit), whitespaces, punctuations (using the Python package Regular- Expression), URL and

tag keywords were then removed. For this first we tokenize each tweet and then remove the words containing undesired characters. Tokenization is the process of splitting the given text into smaller pieces called tokens. Words, numbers, punctuation marks, and others can be considered as tokens. Conversion from upper to lowercase form was also performed. Moreover, all the words of two words or less are removed from the corpus.

"Stop words" are the most common words in a language like "the", "a", "on", "is", "all". These words do not carry important meaning and are usually removed from texts. It is possible to remove stop words using Natural Language Toolkit (NLTK), a suite of libraries and programs for symbolic and statistical natural language processing.

- **Applying Bigram:**

Some English words occur together more frequently. For example - Sky High, do or die, best performance, heavy rain etc. So, in a text document we may need to identify such pair of words which will help in sentiment analysis. First, we need to generate such word pairs from the existing sentence maintain their current sequences. Such pairs are called bigrams. The python definition getwords splits each review into separate words and applies bigram approach to improve the sentiment analysis.

- **Trainset-Testset splitting:**

The very next step after cleaning the reviews is to split the dataset into training and testing dataset. After shuffling 25000 reviews the negative and positive labelled reviews are stored in separated in two different lists. Then each of the lists are splitted in 3:2 ratio. The first 60% of both the positive and negative list are combined to generate trainset and the rest 40% constitutes the test set.

Dataset	No. of Tweets
Trainset	15000
Testset	10000

- **Applying Models:**

After preprocessing the dataset we applied 3 sentiment analysis method to predict the sentiments as positive or negative. Firstly, we applied K Nearest Neighbour classifier, then Naïve Bayes Classifier and finally we tried to improve the Naïve Bayes model by assisting it with a positive and a negative wordlist.

a. K Nearest Neighbor Classifier (KNN):

The basic idea of this algorithm is as follows: if we calculate the similarity between the document to be classified and each of the training documents, the one among the k that is more similar will be indicating to which class or category the document to be classified should be assigned. The one that accumulates more points, will be the suitable candidate.
Here we are counting the frequency of each word in the training set, and assigning it to a python dictionary named freq. Next, we compare test reviews with each train reviews and counted the total frequency of the common words of each test and train review. To scale down the total frequency we have implemented logarithm tool (using the Python logarithm Toolkit) to calculate this score. Finally we have sorted the score list in descending order and considered the 5 (ie. we have considered K as 5) maximum values in the list. Depending on the labels of the 5 most common training reviews of the corresponding test review we have calculated the predicted label.

b. **Naïve Bayes Classifier:**

The Naïve Bayes algorithm is one of the easiest to implement but still the most effective. This model is based on probabilistic theory, particularly in the Bayes theorem, which allows us to estimate the probability of an event out of the probability that another event will occur, on which the first depends. Here we are creating two python dictionary poswords and negwords that store the frequencies of each word in the positive and negative reviews present in the training dataset. Although we are removing the words with very high (we the minimum limit as 10) as well as very low (we the maximum limit as 2000) frequencies respectively as they are

considered to be valueless in the sentiment prediction. Thus we are predicting the overall sentiment of each review by considering the positiveness or negativeness of each word in the test reviews.

c. Naïve Bayes Classifier with annotated wordlist:

This method used two annotated dictionary containing dedicated positive and negative wordlists. These wordlists played substitute role in the classification algorithm assisting the Naïve Bayes classifier. If a test review word be present either in the positive training wordlist (that is the poswords list) or in the negative training wordlist (that is the negwords list), then the prediction will be performed based on the typical Naïve Bayes algorithm. But if the word be present in both the poswords and negwords list, then the label prediction will depend on the presence of that particular word in the assisting annotated dictionaries. This approach will definitely improve the accuracy of the sentiment prediction.

d. **Support Vector Machine (SVM):**

To apply Support Vector Machine we have imported scikit learn (sklearn) and TfidfVectorizer from sklearn to that helps to calculate the TF-IDF scores. After splitting the dataset in 3:2 ratio TfidfVectorizer is applied on the trainset. 2 methods are applied to to fulfil the step: fit_transform() and transform(). Thus after the completion of the learning method, we process the testset and finally classify it using the method svm.LinearSVC().

- **Calculate Accuracy:**

Finally, after implementing various classifier models we compare each element of the actual and predicted label list and thus calculate the accuracy percentage. The accuracy calculating formula is expressed as,

$$\text{accuracy} = \frac{\mathbf{tp + tn}}{\boldsymbol{N}} * \mathbf{100\%}$$

where,
tp is the count of correctly predicted positive test reviews,
tn is the count of correctly predicted negative test reviews,
and N is the total count of reviews in test dataset.

3.3. Results and Summary

Applying various machine learning models as discussed in section 3.2.4 we have analysed their accuracies as displayed in the following table:

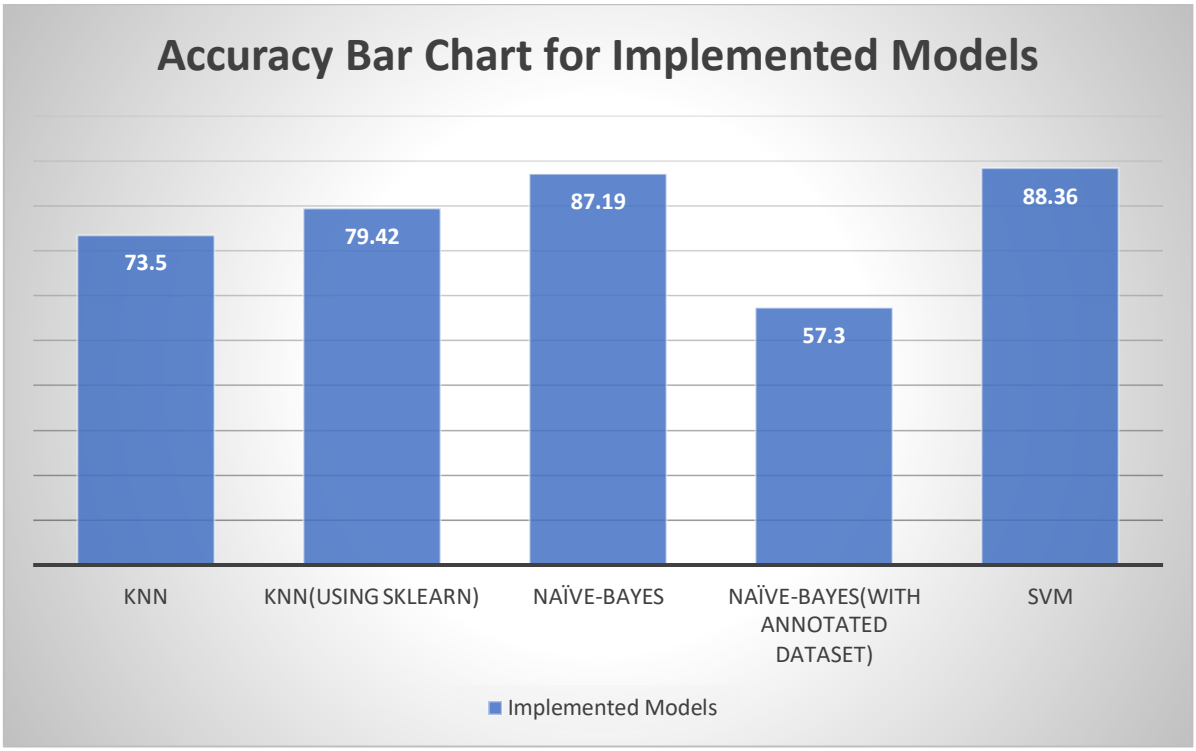
Algorithm	Accuracy (%)	True positive (%)	True Negative (%)	False positive (%)	False Negative (%)
KNN	73.5	43	30.5	19.5	7
KNN (using sklearn)	79.42	42.84	36.58	13.42	7.16
Naïve-Bayes	87.19	44.96	42.23	7.77	5.04
Naïve-Bayes (with annotated dataset)	57.30	40.60	16.70	33.3	9.4
SVM	88.36	43.84	44.52	6.14	5.5

Here we have calculated the percentage of correct predictions and incorrect predictions for both positive and negative tweets in the last 4 columns.

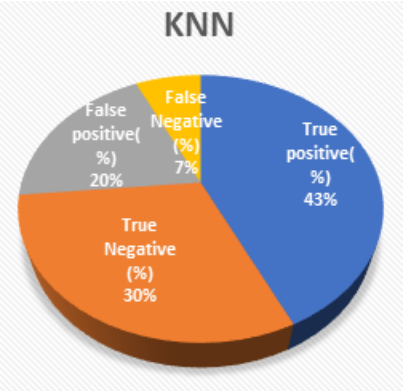
From the table we can conclude that:

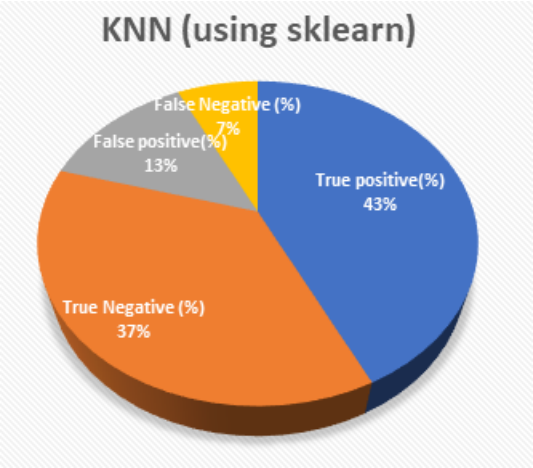
i. The applied models except Naïve-Bayes with annotated dataset) works exceptionally well for the dataset, especially Naïve-Bayes algorithm and Support Vector Machine classifier.
ii. We have to work harder on all the models to deal with the negative tweets to improve our obtained accuracy as predictions on the negative tweets are little poor.

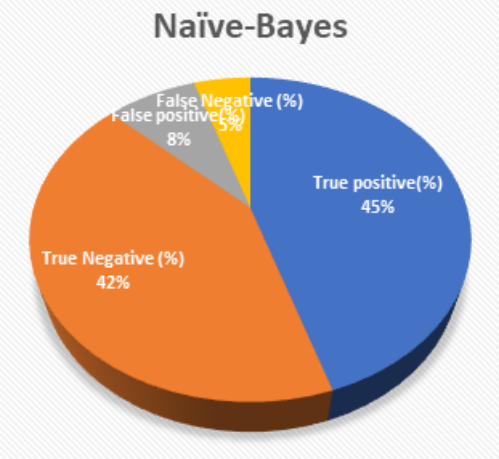
The accuracies for all the implemented models are plotted on the following Bar Chart. It depicts the superior performance of Naïve-Bayes classifier and SVM over the others.

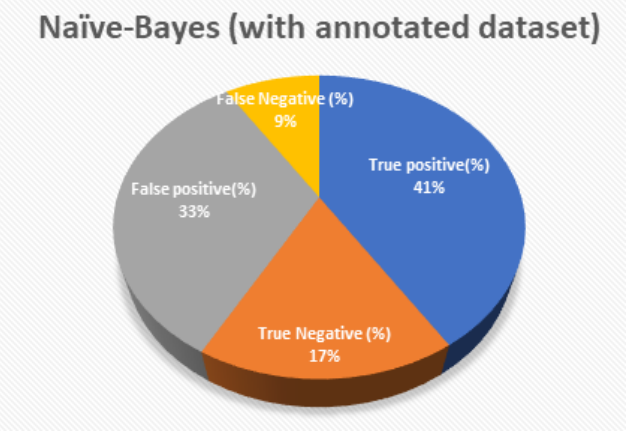


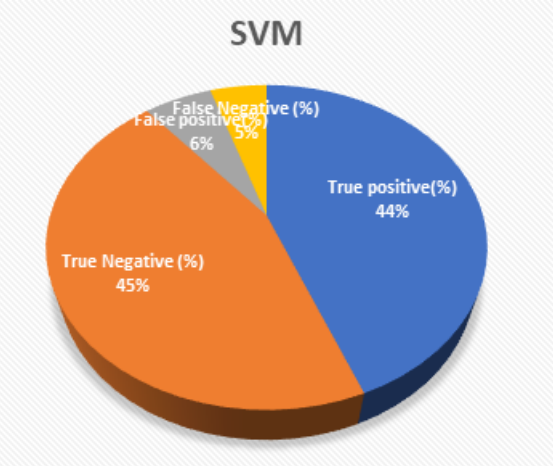
Pie Charts depicting accuracies for all implemented models











Chapter 4: Sentiment Summarisation

4.1. Dataset

We use the dataset on the disastrous earthquake in Nepal in the year 2015, which is publicly available on crisisnlp.qcri.org [7]. Here is a description of the data, provided by the website:

> The set consists of 3000 tweets on the unfortunate Nepal earthquake, specially selected for sentiment analysis between the period 2015-04-25 12:53:13 to 2015-05-19 06:35:49. Each of these rows contains tweet date, tweet category, confidence value of the tweet category, tweet id followed by the tweet itself. Furthermore, the retweets present in the dataset is mentioned with 'RT' keyword at the beginning. As per Twitter norms each tweet has maximum length of words.

4.2. Proposed Method

1. Preprocessing:

Before start processing the tweets the initial step is to preprocess each tweets to get rid of the useless terms present. The data is cleaned by first creating a function clean() which receives each tweet from the main program and pre-process the words.

Stop words, whitespaces, punctuations, URL and tag keywords were then removed. For this first we tokenize each tweet and then remove the words containing undesired characters. Tokenization is the process of splitting

the given text into smaller pieces called tokens. Words, numbers, punctuation marks, and others can be considered as tokens.

"Stop words" are the most common words in a language like "the", "a", "on", "is", "all". These words do not carry important meaning and are usually removed from texts. It is possible to remove stop words using Natural Language Toolkit (NLTK), a suite of libraries and programs for symbolic and statistical natural language processing.

Next, we lemmatize the words by importing WordNetLemmatizer from the NLTK package. Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is similar to stemming but it brings context to the words. So, it links words with similar meaning to one word. Text preprocessing includes both Stemming as well as Lemmatization. Many times, people find these two terms confusing. Some treat these two as same. Actually, lemmatization is preferred over Stemming because lemmatization does morphological analysis of the words.

Examples of lemmatization:

→ rocks: rock

→ corpora: corpus

→ better: good

Moreover, conversion from upper to lowercase form was also performed. Finally, we join the remaining useful words by whitespaces.

One example of preprocessing tweets is shown here:

I. ***Actual tweet:*** "RT @arjunk26: My friend @RohanShrestha is heading int o Nepal to him his fellow people show ur support anyway u can http://t.co/yJ xzPVMNke"

→ *After preprocessing*: "My friend heading Nepal fellow people show ur su pport anyway u"

II. *Actual tweet*: "RT @Harry_Styles: To help those affected by the devastat ing earthquake in Nepal, text: DONATE5 to 70008 and give Ã¥Â£5. Thank you @savechildreâ€°Ã›_"

→ *After preprocessing*: "RT To help affected devastating earthquake Nepal text DONATE5 70008 give Thank"

2. TF-IDF Score Calculation:

Without going into the math, TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents.
The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow us to encode new documents. Alternately, if we already have a learned CountVectorizer, we can use it with a TfidfTransformer to just calculate the inverse document frequencies and start encoding documents.

A vocabulary of 5748 words is learned from the documents and each word is assigned a unique integer index in the output vector.

The inverse document frequencies are calculated for each word in the vocabulary, assigning the lowest score of 1.0 to the most frequently observed words.

Finally, the document is encoded as a sparse array of shape (3000, 5748). The scores are normalized to values between 0 and 1 and the encoded document vectors can then be used directly with most machine learning algorithms.

3. Cosine Similarity Computation:

To compute cosine similarity we define get_cosine_sim() function that process the each of the tweets. The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

We can use it as follows:

- Create an instance of the *CountVectorizer* class.
- Call the *fit()* function in order to learn a vocabulary from one or more documents.
- Call the *transform()* function on one or more documents as needed to encode each as a vector.

An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document.

Because these vectors will contain a lot of zeros, we call them sparse. Python provides an efficient way of handling sparse vectors in the scipy.sparse package.

The vectors returned from a call to transform() will be sparse vectors. All sparse matrix representations in scipy have a todense() method which converts the matrix to a standard numpy matrix. (Again, the traditional definition of sparse matrix is in conflict with the conventional definition—todense() just changes the representation. It does not fill the zeros in with nonzero values.).

Finally, we apply hierarchy.linkage imported from the scipy package on the 1d condensed distance matrix to calculate the cosine similarity metric for each pair of tweets. For n tweets a (n−1) by 4 matrix Z is returned. At the i-th iteration, clusters with indices Z[i, 0] and Z[i, 1] are combined to form cluster n+i. A cluster with an index less than n corresponds to one of the n original observations. The distance between

clusters Z[i, 0] and Z[i, 1] is given by Z[i, 2]. The fourth value Z[i, 3] represents the number of original observations in the newly formed cluster.

4. Clustering based on Cosine Similarity Values:

Clusterization is performed based on the cosine similarity values between each pair of tweets. All the pair of tweets having more than a certain threshold cosine similarity score are formed a single group. In this implementation the threshold value is specified as 0.85. We have used hierarchy.fcluster() on the cosine similarity output matrix to cluster the tweets on the specified threshold value. The output formed is a numpy.ndarray of length 3000, each containing a cluster id. Applying the method we have successfully clustered 3000 tweets to 242 clusters.

Now we obtain the tweet information with maximum TF-IDF score from each cluster. For the remaining assessment we will consider only these shortlisted rows as each of these 242 tweets are considered to be the most valuable among each clusters created from the dataset of 3000 tweets.

5. Sentiment Analysis:

We need to load the SentimentIntensityAnalyser object in from the VADER package and as it's a bit long, we'll assign it to another name, sid, to make it a bit easier to use. Finally, we'll use the polarity_scores() method to get the sentiment metrics for a piece of text.

VADER produces four sentiment metrics from these word ratings, which we can see below. In the first example, the first three, positive, neutral and negative, represent the proportion of the text that falls into those categories. As we can see, our first example sentence was rated as 82% positive, 17% neutral and 0% negative. when VADER analyses a piece of text it checks to see if any of the words in the text are present in the lexicon. The final metric, the compound score, is the sum of all of the lexicon ratings have been standardised to range between -1 and 1. In this case, our example sentence has a rating of 0.9231, which is strongly positive.

Our second example sentence was rated as 0% positive, 25% neutral and 74% negative. The final metric in our second example sentence has a rating of -0.7003 which is pretty strongly negative.

The third and the last example sentence was rated as 0% positive, 100% neutral and 0% negative. The final metric has a rating of 0.00 which is overall neutral.

We illustrate the whole scenario:

I. ***Tweet 1:*** "Thank God that my dearest friend's family and friends in Nepal are safe and doing good. Let's Pray‰Û_ https://t.co/YqRABF5pj2"

→ Sentiment: {'neg': 0.0, 'neu': 0.173, 'pos': 0.827, 'compound': 0.9231}

II. ***Tweet 2***: "RT @TheMahiraKhan: Heartbreaking to see the devastation in Nepal.."

→ Sentiment: {'neg': 0.744, 'neu': 0.256, 'pos': 0.0, 'compound': -0.7003}

II. ***Tweet 3***: "RT @ABCNews24: #NepalEarthquake update: The Indian Army says 18 bodies have been found on #MountEverest after an #avalanche buried part of ‰Û_"

→ Sentiment: {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}

6. Interim Data Representation:

In this step we represent each row of the dataset as vectors considering only the essential features for performing kmean clustering. Thus we take the TF-IDF vectors and the sentiment vectors into one combination for each tweet.

Here we use numpy.hstack() method for this task. numpy.hstack() function is used to stack the sequence of input arrays horizontally (i.e. column wise) to make a single array. The arrays must have the same shape along all but the second axis, except 1-D arrays which can be any length. sparse format of the result (e.g. "csr") by default an appropriate sparse matrix format is returned. Here we have changed the scipy.sparse.coo.coo_matrix to simple numpy array.

7. K-Mean Clustering:

For this particular algorithm to work, the number of clusters has to be defined beforehand. The K in the K-means refers to the number of clusters. To run the following script we need the scikit-learn library.

It requires just two lines of code to implement the algorithm. In the first line, we create a KMean object and pass it k as value for n_clusters parameter. Next, we simply have to call the fit method on kmean and pass the data that we want to cluster, which in this case is the vector array that we created earlier. By varying values of k (e.g. 10, 20, 30) we got k number of clusters each have their specific range of sentiment values.

Implemented Algorithm:

→ Step 1: The data is cleaned by first creating a method clean() which receives each tweet from the main program and pre-process the words.

- Stop words, whitespaces, punctuations, URL and tag keywords were then removed. For this first we tokenize each tweet and then remove the words containing undesired characters.
- Next, we lemmatize the words by importing WordNetLemmatizer from the NLTK package.

→ Step 2: The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow us to encode new documents. A vocabulary of 5748 words is learned from the documents and each word is assigned a unique integer index in the output vector. The inverse document frequencies are calculated for each word in the vocabulary, assigning the lowest score of 1.0 to the most frequently observed words. Finally, the document is encoded as a sparse array of shape (3000, 5748).

→ Step 3: To compute cosine similarity we define get_cosine_sim() function that process the each of the tweets. We can use it as follows:

- Create an instance of the *CountVectorizer* class.
- Call the *fit()* function in order to learn a vocabulary from one or more documents.
- Call the *transform()* function on one or more documents as needed to encode each as a vector.
- The vectors returned from a call to transform() will be sparse vectors. We make use of the todense() method which converts the matrix to a standard numpy matrix.
- Finally, we apply hierarchy.linkage imported from the scipy package on the 1d condensed distance matrix to calculate the cosine similarity metric for each pair of tweets.

→ Step 4:

- We have used hierarchy.fcluster() on the cosine similarity output matrix to cluster the tweets having cosine similarity score more than a threshold value 0.85. Applying the method we have successfully clustered 3000 tweets to 242 clusters.

- Now we obtain the tweet information with maximum TF-IDF score from each cluster. For the remaining assessment we will consider only these shortlisted rows as each of these 242 tweets are considered to be the most valuable among each clusters created from the dataset of 3000 tweets.

→ Step 5: We load the SentimentIntensityAnalyser object from the VADER package and as it's a bit long, we'll assign it to another name, sid, to make it a bit easier to use. Finally, we'll use the polarity_scores() method to get the sentiment metrics for a piece of text.

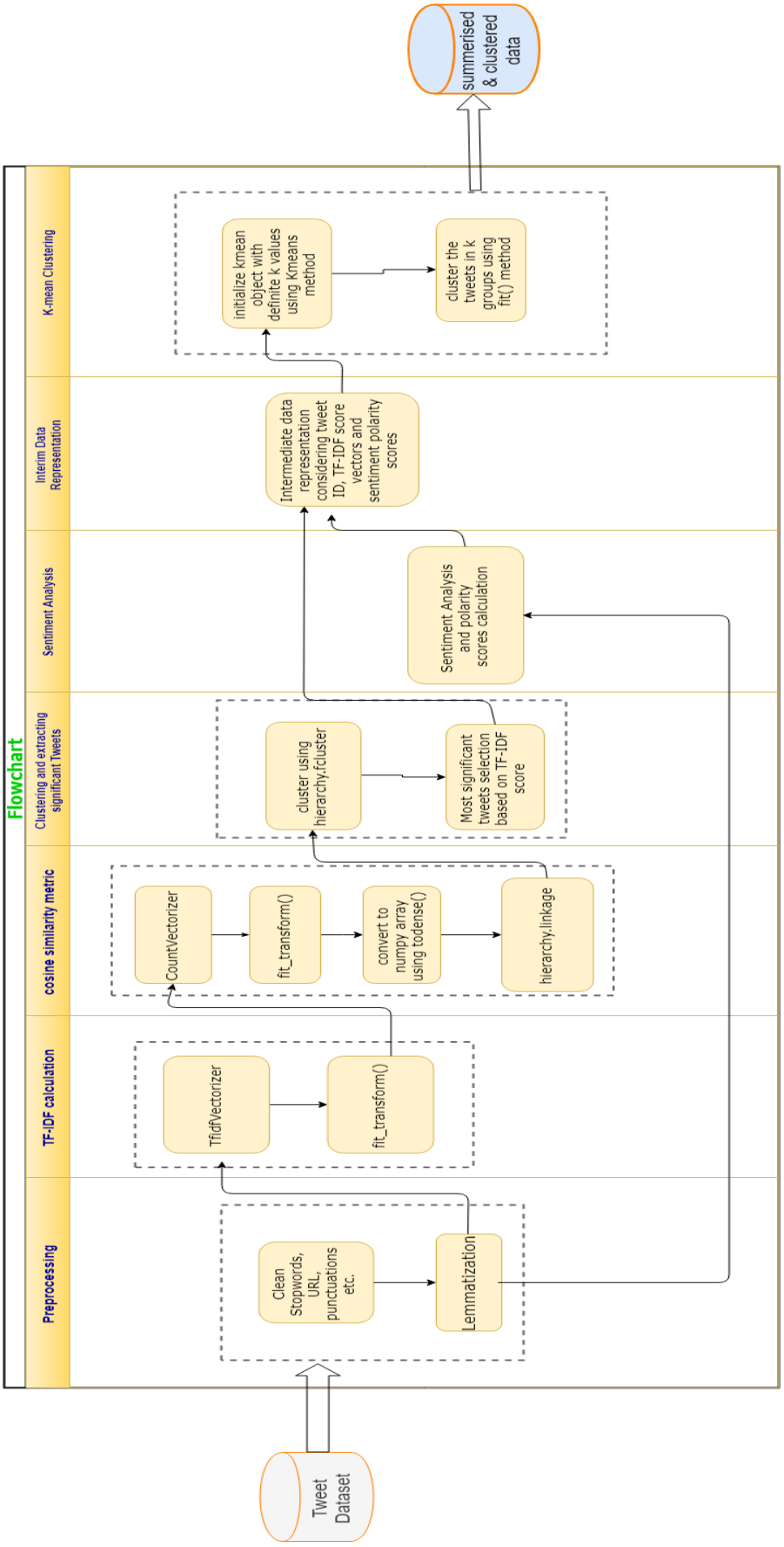
→ Step 6: In this step we represent each row of the dataset as vectors considering only the essential features for performing kmean clustering. Thus we take the TF-IDF vectors and the sentiment vectors into one combination for each tweet. Here we have changed the scipy.sparse.coo.coo_matrix to simple numpy array and used numpy.hstack() method for this task.

→ Step 7: The K in the K-mean refers to the number of clusters. To run the following script we need the scikit-learn library. It requires just two lines of code to implement the algorithm:

- In the first line, we create a KMean object and pass it k as value for n_clusters parameter.

- Next, we simply have to call the fit method on kmean and pass the data that we want to cluster, which in this case is the vector array that we created earlier. By varying values of k (e.g. 10, 20, 30) we got k number of clusters each have their specific range of sentiment values.

Flowchart:


Flowchart
Preprocessing
TF-IDF calculation
cosine similarity metric
Clustering and extracting significant Tweets
Sentiment Analysis
Interim Data Representation
K-mean Clustering
Tweet Dataset
Clean Stopwords, URL, punctuations etc.
Lemmatization
TfidfVectorizer
fit_transform()
CountVectorizer
fit_transform()
convert to numpy array using todense()
hierarchy.linkage
cluster using hierarchy.fcluster
Most significant tweets selection based on TF-IDF score
Sentiment Analysis and polarity scores calculation
Intermediate data representation considering tweet ID, TF-IDF score vectors and sentiment polarity scores
initialize kmean object with definite k values using Kmeans method
cluster the tweets in k groups using fit() method
summerised & clustered data

4.3. Result & Summary:

By clustering the dataset of 3000 tweets using cosine similarity we have obtained 242 distinct clusters. From each cluster we have extracted the most significant one (which have highest TF-IDF score) for further processing. Finally applying KMean clustering on the 242 intermediate tweets based on the TF-IDF metrics and sentiment scores we have grouped the corpus into different number of clusters (i.e. k= 10, 20, 30).

→ The following table depicts Average Sentiment, Minimum Sentiment, Maximum Sentiment and Standard Deviation of each cluster for k=10.

Cluster id	Average Sentiment	Minimum Sentiment	Maximum Sentiment	Standard Deviation	Tweet
1	0.000046	-0.1531	0.2023	0.041568	PlanGlobal: Watch PlanGlobal davtox on CNN in ten minutes, discussing #NepalEarthquake and Plan's response from the stranded plane to Kathmâ€°Ã›_
2	-0.11442	-0.3612	0.0258	0.122432	#RG fishing for votes politicking on #LAB when world is helping #NepalEarthquake victims! This completes Congis' disconnect! #RGPunjabVisit
3	0.8221	0.5859	0.9001	0.096844	RT @PatOndabak: IRONY ALERT: Harper Govt claims credit for actions of charities like Red Cross & World Vision... as they audit charities #Nâ€°Ã›_
4	0.583206	0.4767	0.7184	0.068625	#Nepal It is time to thank the TV news channels -all the editors and reporters (on ground zero &desk) continuously giving courage. I salute
5	-0.63517	-0.8527	-0.4767	0.111956	RT @SiliconAngel: Nepal desperate for supplies as death toll soars past 3,700: Shelter, fuel, food, medicine, power, news, worke... http://â€°Ã›_
6	0.41013	0.2263	0.4939	0.06875	@mallabipin Hi Bipin,BMO has made donation to the Nepal relief efforts and clients can donate to the Red Cross at their local BMO branch.^TO
7	0.067513	-0.25	0.34	0.178658	RT @RobTatumMMA: It's easy to get caught in the #MMA bubble, but the shit that's happening in Nepal and Baltimore is way more important thaâ€°Ã›_

8	-0.3348	-0.5574	-0.0572	0.115528	I have latest photos of damaged area of World Heritage Site #Nepal. Any media or organization want those?Plz contact #earthquake #Nepalquake
9	0.719558	0.4215	0.8625	0.09544	@Northeastern stop by the Curry Bookstore/On the Go tables for @NUutsav's Henna 4 Hope fundraiser! Proceeds to #NepalQuake relief efforts
10	-0.63206	-0.836	-0.3818	0.149498	Our teams report that the most needed items are food, shelter, pain killers, antibiotics and stretchers.Ã¥ÃŠ#NepalQuake http://t.co/5neAEBCf5p

→ The following table depicts Average Sentiment, Minimum Sentiment, Maximum Sentiment and Standard Deviation of each cluster for k=20.

Cluster id	Average Sentiment	Minimum Sentiment	Maximum Sentiment	Standard Deviation	Tweet
1	-0.73967	-0.8442	-0.6124	0.076456	RT @501Awani: [LATEST] Nepal Quake: Avalanche killed 22 so far and 167 mountaineers still reported missing at Mount Everest - Indian Army Oâ€°Ã›_
2	0	0	0	0	RT @IsraelandStufff: Photo Album: #Israelâ€°Ã›Âªs Medical Assistance in #Nepal Begins First humanitarian mission aircraft has landed in... http:â€°Ã›_
3	0.631409	0.5106	0.7845	0.07566	RT @hstapanghosh: US Pastor Tony Miano suggestS Nepalis should not rebuild their 'pagan shrines' http://t.co/VrrGV7lnp3 BRIGHT EXAMPLE OF Pâ€°Ã›_
4	0.493775	0.3818	0.5859	0.107604	Dear Pakistan, are out of your mind? http://t.co/gMmjrE4n3b
5	-0.32905	-0.4404	-0.1779	0.11782	Choppers ferry injured in Nepal; new mudslide hits village: GORKHA, Nepal (AP) â€°Ã›Ã“ Helicopters cri... http://t.co/cwncd7EXic <--Full Story
6	0.80835	0.6369	0.8979	0.120363	Yes I agree "a selfless service". a learning for others. https://t.co/j7yEY4nU5I
7	0.655422	0.34	0.8176	0.141107	Shared via NBC News for Windows Phone 8 http://t.co/kaRVoDpEYb
8	0.500238	0.296	0.7003	0.1119	There are miracles everyday! #proudofindianforces https://t.co/9gIPGyY8Rx
9	-0.48553	-0.5574	-0.4404	0.062915	RT @Ac_in_quest: Massive block of Earthâ€°Ã›Âªs crust, 75 miles long & 37

					miles wide, lurched 10 feet to south Saturday over course of 30 secondsâ€°Ã›_
10	-0.21652	-0.3182	0	0.111656	@timesofindia @meaindia :Not only Indians but who ever wants shelter from Nepal migrants to be kept in rehabilitation camps.
11	-0.58597	-0.7003	-0.4939	0.056195	Americans Desperately Wait for Word From Nepal: The catastrophic earthquake over the weekend took out power and phone service for who...
12	0.249286	0.0258	0.3612	0.117247	Dhungana family moved from Nepal 7 mnth ago. Family are okay but an orphan who lived with them is not. #cbcwdr http://t.co/izx7kUHCP2
13	-0.47406	-0.6492	-0.3818	0.089912	Fatboy asked why pray for #Nepal yet God who could have stopped the catastrophe didn't do nothing about it? it's a good Qn huh!!
14	0.832738	0.7506	0.9001	0.050586	RT @PatOndabak: IRONY ALERT: Harper Govt claims credit for actions of charities like Red Cross & World Vision... as they audit charities #Nâ€°Ã›_
15	-0.29781	-0.4939	-0.1027	0.128142	@RanvirShorey location of Vipul in nepal. Vipul died in Nepal. Need help to get his body back to mumbai http://t.co/sS3QsjV50z
16	-0.00315	-0.1531	0.1027	0.047489	@Babu_Bhaiyaa @Dec3rd_q Act as frnd in time of trouble. frnd can wipe tear. But India isn't mother of Nepal. Sovereign cntrs r equal.
17	0.463768	0.3612	0.7003	0.071597	Everyone let's help out @CrawfordCollins â€°Ã‚Åœâ€°Ã‚Åœâ€°Ã‚Åœâ€°Ã‚Åœ â€°_Ã‚ https://t.co/BNWueSjqaa â€°Ã‚‰ To donate â€°_Ã‚â€°_Ã‚â€°_Ã‚â€°_Ã‚â€°_Ã‚â€°_Ã‚ https://t.co/xivUtSzZZs â€¡__C
18	-0.76086	-0.8527	-0.6124	0.076055	Anyone else find it obscene that two guys punching each other in the head attracts a $300million pay day while the people of Nepal starve?
19	0.169025	-0.0258	0.3818	0.136615	RT @RobTatumMMA: It's easy to get caught in the #MMA bubble, but the shit that's happening in Nepal and Baltimore is way more important thaâ€°Ã›_
20	-0.01501	-0.1531	0	0.04075	PlanGlobal: Watch PlanGlobal davtox on CNN in ten minutes, discussing #NepalEarthquake and Plan's response from the stranded plane to Kathmâ€°Ã›_

→ The following table depicts Average Sentiment, Minimum Sentiment, Maximum Sentiment and Standard Deviation of each cluster for k=30.

Cluster id	Average Sentiment	Minimum Sentiment	Maximum Sentiment	Standard Deviation	Tweet
1	0	0	0	0	RT @eoiktmnp: Buses leaving for Gorakhpur carrying stranded Indians from Tilganga bridge near Kathmandu Intl Airport @MEAIndia http://t.co/â€°Ã›_
2	0.457711	0.2263	0.5859	0.10635	RT @hstapanghosh: US Pastor Tony Miano suggestS Nepalis should not rebuild their 'pagan shrines' http://t.co/VrrGV7lnp3 BRIGHT EXAMPLE OF Pâ€°Ã›_
3	-0.7604	-0.8527	-0.5574	0.08482	#NepalQuakeRelief Serious logistical problem. Ineffectiveness of the government and dirty politics C K Lal next http://t.co/G5ZAaLJrZs
4	-0.5567	-0.6597	-0.4767	0.093643	RT @one_by_two: I dont know if Modi will win next general elections in India - but he will definitely win in Nepal's next elections
5	0.7675	0.7096	0.8176	0.032906	RT @narendramodi: Received a call from Pujya Morari Bapu. He has donated Rs. 51 lakh for relief work in Nepal. My deepest gratitude to him.
6	0.786445	0.5859	0.9001	0.095468	There are miracles everyday! #proudofindianforces https://t.co/9gIPGyY8Rx
7	-0.34486	-0.4767	-0.25	0.083769	I have latest photos of damaged area of World Heritage Site #Nepal. Any media or organization want those?Plz contact #earthquake #Nepalquake
8	-0.74923	-0.8442	-0.6739	0.065907	((Noticias SIN)) The Latest on Nepal Quake: Thousands Try to Leave Kathmandu: Thousands line up at bus stations in Kathmandu in bid t...
9	0.49273	0.296	0.7003	0.126315	RT @BBCSanjoyM: Indian Air Force C130 Hercules and Globemaster aircraft on their way to Nepal carrying relief material says MoD #NepalEarthâ€°Ã›_
10	-0.09534	-0.296	0.1027	0.123567	RT @RobTatumMMA: It's easy to get caught in the #MMA bubble, but the

					shit that's happening in Nepal and Baltimore is way more important thaâ€°Ã›_
11	-0.04229	-0.296	0	0.111877	â€°Âª Ã‡ Just Announced: Ko Phangan, Thailand - Apr 30 at Nepal Fundraiser @ Leela Beach http://t.co/Fq3xj4lpqm
12	-0.01284	-0.3182	0.2023	0.129343	RT @CMOfficeUP: '@UPGovt is sending 25 buses today for evacuation in Nepal.More buses will be sent as demand arises'- #UPCM @yadavakhilesh
13	-0.50483	-0.6249	-0.4019	0.079033	Choppers ferry injured in Nepal; new mudslide hits village: GORKHA, Nepal (AP) â€°Ã›Ã“ Helicopters cri... http://t.co/cwncd7EXic <--Full Story
14	-0.05077	-0.2732	0.0258	0.11235	#RG fishing for votes politicking on #LAB when world is helping #NepalEarthquake victims! This completes Congis' disconnect! #RGPunjabVisit
15	-0.45152	-0.6249	-0.2263	0.142963	@BritishRedCross I've donated financially to the #NepalQuake Disaster Fund but also have clothes & blankets to give. Are u accepting those?
16	0.241171	0.0772	0.4404	0.122252	Gay families retrieved from Nepal quake zone; Northern Ireland rejects gay marriage again; Supreme Court begins... http://t.co/Ed9VK5UhhL
17	0.36643	0.25	0.4767	0.071902	@mallabipin Hi Bipin,BMO has made donation to the Nepal relief efforts and clients can donate to the Red Cross at their local BMO branch.^TO
18	-0.14546	-0.296	0	0.102685	#NepalQuake At this stage Dirco says it has no confirmed reports - but is working round the clock to find any affected South Africans. EC
19	0	0	0	0	@Libishski front page in all the states news papers, even trending on social media everywhere above Nepal. What a world huh?
20	0.46118	0.34	0.6486	0.11829	Phones jammed in Nepal. Thankfully for Whatsapp for communicating messages!

21	-0.50291	-0.802	-0.2023	0.190359	Big B expresses shock over Nepal quake, reminisces â€°Ã›Ã·Mahaanâ€°Ã›Âª shoot http://t.co/BsWEi98sdr http://t.co/prVRIPMAE3 #OnlySalmanKhan1
22	-0.605	-0.7003	-0.5267	0.05295	Americans Desperately Wait for Word From Nepal: The catastrophic earthquake over the weekend took out power and phone service for who...
23	-0.00469	-0.0516	0	0.015558	General from Bharat with Nepal... Heartening to see Vikramjit Sahney & Manjit Singh come out and contribute to... http://t.co/Bvbp3wJVLf
24	0.89195	0.886	0.8979	0.008415	Heartfelt prayers, support and healing love being sent to all, in Nepal. http://t.co/Iki2a96yiJ
25	0	0	0	0	â€°Ã›Ãï¿½I came to know about Earthquake through Modi twitter account while in Bangkok airport. We called Nepal and got updates regularly" Koirala
26	0.49864	0.4404	0.5994	0.060227	Earthquakes are soooo funny! Nepal quake reveals our â€°Ã›Ã·mean girlâ€°Ã›Âª social media avatars Sitting in a cab in Mumbai, â€°Ã›_ http://t.co/jQGM0IAd6o
27	0.53132	0.3818	0.7351	0.109217	RT @LiaqatAli2015: 32/48 everybody will become Jesus Christ. In this scenario of the Awaited One Gohar Shahi blessing humanity, I wanted toâ€°Ã›_
28	0.642082	0.4404	0.8126	0.10647	@SundusRasheed I have sent you a message on Facebook along with Devina Shrestha, my contact in Nepal. Kindly respond. They are waiting..
29	-0.44348	-0.4939	-0.3818	0.051424	I cried yesterday â€¡_Â¢D https://t.co/Hi8768fqUg
30	-0.61817	-0.8126	-0.4019	0.130888	Fatboy asked why pray for #Nepal yet God who could have stopped the catastrophe didn't do nothing about it? it's a good Qn huh!!

For different values of k the average **Standard Deviation (SD) is as shown here:**

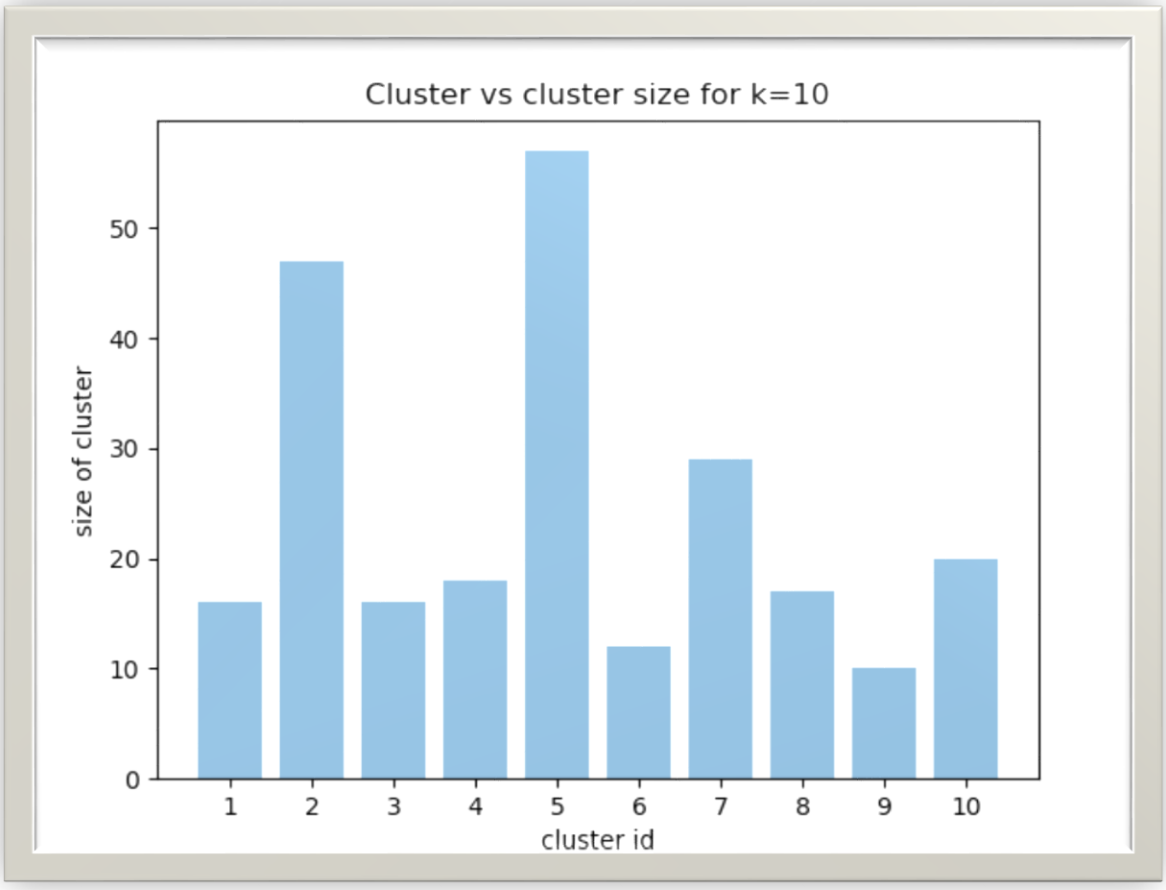
Clustering for different k values	Average Standard Deviation
k=10	0.09906
k=20	0.08700
k=30	0.08429

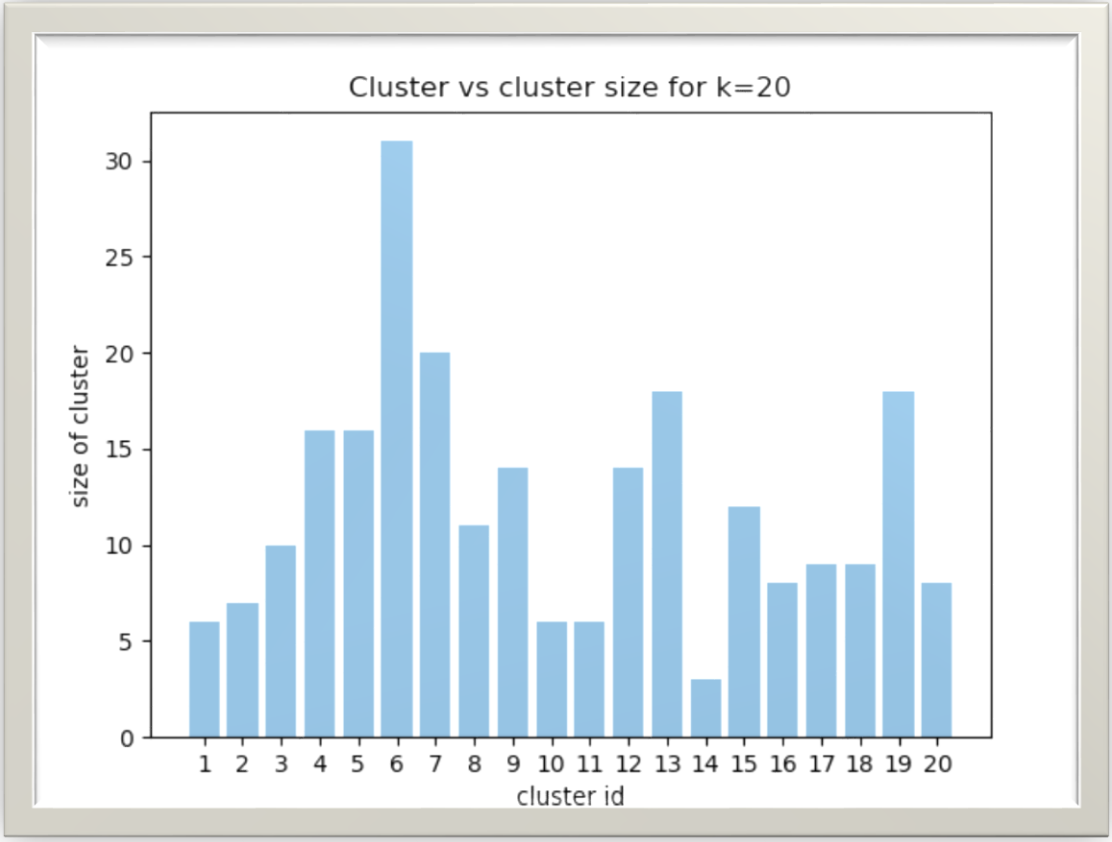
For the table it is evident that:

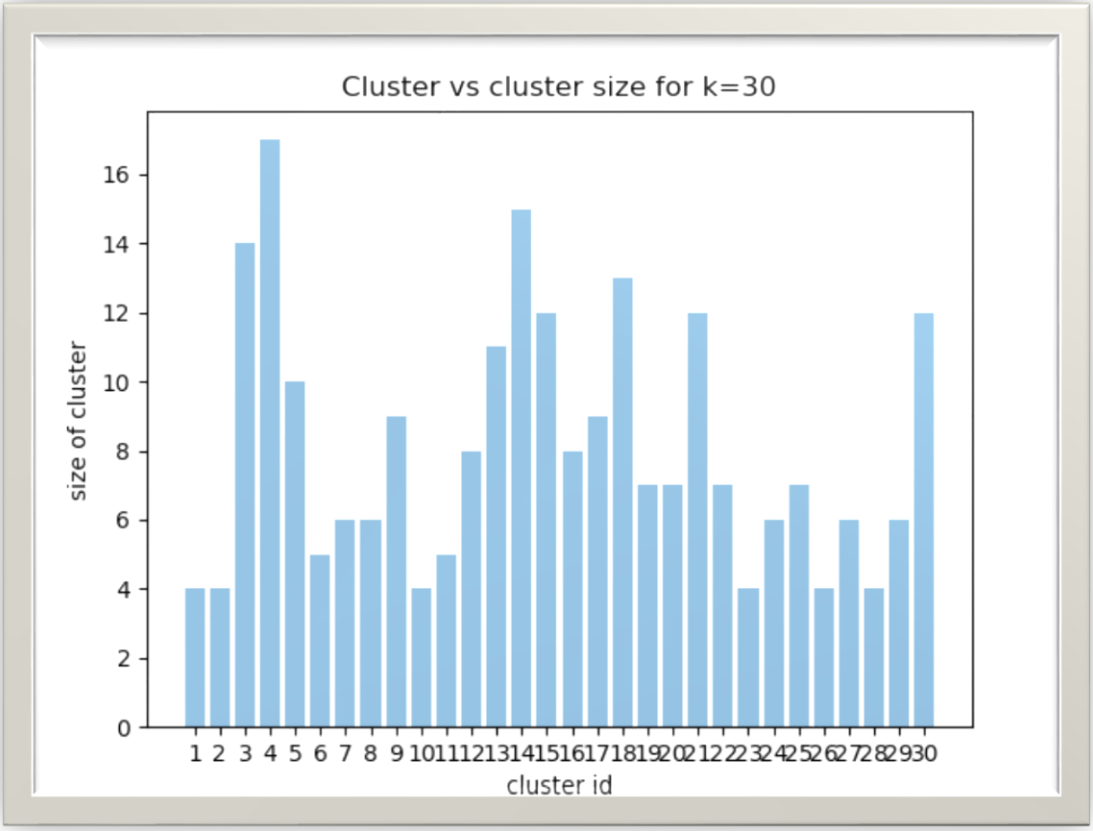
i. Average Standard Deviation is less than 0.1 which is quite satisfactory.

ii. With the increase in number of clusters (i.e. k=10, 20, 30) Standard Deviation decreases which leads to the conclusion that clusters are becoming to be more prominent.

→ The size of each cluster with varying number of clusters k = 10, 20 and 30 respectively are as follows:




Cluster vs cluster size for k=20
size of cluster
cluster id
0
5
10
15
20
25
30
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


Cluster vs cluster size for k=30
size of cluster
cluster id
0
2
4
6
8
10
12
14
16
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

Thus it is evident that the tweets are more or less uniformly distributed for among all the clusters for all the 3 values of k (i.e. k= 10, 20 and 30 respectively).

→ The following table displays average TF-IDF scores and their Standard Deviation for each cluster for cluster size k=10.

Cluster id	TF-IDF Average	TF-IDF Standard Deviation
1	3.231894	0.500833
2	3.111041	0.562306
3	2.781956	0.663716
4	2.968597	0.421903
5	3.011343	0.53075
6	3.231601	0.656738
7	3.224936	0.519289
8	2.970761	0.460675
9	3.082569	0.410459
10	3.400881	0.369813

→ The following table displays average TF-IDF scores and their Standard Deviation for each cluster for cluster size k=10.

Cluster id	TF-IDF Average	TF-IDF Standard Deviation
1	3.283229	0.357935
2	3.484617	0.293668
3	3.162057	0.406101
4	2.962673	0.416041
5	2.793439	0.671586
6	2.984308	0.545503
7	3.147556	0.592826
8	3.192253	0.466806
9	3.309276	0.572145
10	3.425736	0.336935
11	3.273146	0.258681
12	3.364776	0.583898
13	3.271875	0.325542
14	2.705854	1.19427
15	3.024761	0.25009
16	2.321444	0.612468
17	3.387841	0.425157
18	3.000655	0.288253
19	3.015365	0.560343
20	2.937237	0.379879

→ The following table displays average TF-IDF scores and their Standard Deviation for each cluster for cluster size k=10.

Cluster id	TF-IDF Average	TF-IDF Standard Deviation
1	3.120768	0.458078
2	2.7926	0.539653
3	2.98251	0.573846
4	3.46794	0.356529
5	2.975649	0.53133
6	3.324947	0.234972
7	3.376461	0.227894
8	3.307131	0.652305
9	2.848001	0.481214
10	3.266301	0.256041
11	3.199219	0.328138
12	3.283393	0.370965
13	3.275069	0.44808
14	2.837165	0.542762
15	3.085034	0.444818
16	2.988439	0.384568
17	3.350073	0.615809
18	2.830888	0.707647
19	2.179915	0.658388
20	3.505899	0.648913
21	3.097635	0.554036
22	3.095392	0.303842
23	3.099485	0.419919
24	2.792057	0.12053
25	2.935619	0.446764
26	3.247632	0.336404
27	3.032759	0.524346
28	3.260899	0.437341
29	3.377399	0.274384
30	3.172128	0.607857

Evidently, we can conclude that distribution of TF-IDF values among the clusters are quite satisfactory for all k values.

Chapter 5: Conclusion & Future Work

With the large amount of data driving business decisions, accuracy is imperative. Here our task was divided into two parts. The first part consists preprocessing of data and splitting the single dataset in training and testing set and then implementing sentiment classifier models and finally comparing the predicted labels with the actual sentiments and thus calculating the accuracy. We have achieved a satisfactory accuracy of 88% in Naïve Bayes classifier. Unfortunately, the overall performance of predicting the negative reviews is deficient. In the later period of time we will focus on improving this issue as well as will try to implement some more efficient approach that will improve the model performance. We will also focus on identifying and altering the misspelled words in the comments to get an effective sentiment analysis. This will lead to a better understanding of human language opinions and better processing of human thought by a machine.

In the second part we have clustered the dataset based on the cosine similarity values between each pair of tweets and have implemented k-means clustering and thus extracted the most significant tweets. We have achieved a satisfactory classification with less standard deviation. Unfortunately, we faced little difficulties dealing with retweets. Retweets will lead to a better understanding of human language opinions and better processing of human thought by a machine.

Chapter 4: Appendix

‘‘‘ ---------- **SENTIMENT ANALYSIS** ---------- ’’’

NAÏVE-BAYES CLASSIFIER

```
#Importing libraries
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import string
import re
import numpy as np
import random
from collections import Counter


stop = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()

# Cleaning the text sentences
def clean(doc):
    doc=doc.replace('<br />',' ')
    stop_free = " ".join([i for i in doc.lower().split() if i not in stop])
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    normalized = " ".join(lemma.lemmatize(word) for word in punc_free.split())
    processed = re.sub(r"\d+","",normalized)
    y = processed.split()
    for word in y:
        if len(word)<=2:
            del y[y.index(word)]
        word.lower()
    return y


#applying bigram
def getwords(sentence):
    w = sentence.split(" ")
    w= w + [w[i]+' '+w[i+1] for i in range(len(w)-1)]
    w= list(set(w))
    return w
```

```
path = "movie.txt"

train_clean_sentences = []
y_train=np.array([])
test_clean_sentences = []
y_test=np.array([])
fp = open(path,'r')

ds=[]
for row in fp:
    ds.append([row[:-2],int(row[-2])])

for i in range(len(ds)):
    cleaned= clean(ds[i][0])
    cleaned = ' '.join(cleaned)
    ds[i][0]=cleaned

random.shuffle(ds)

poslines=[]
neglines=[]
for i in ds:
    if i[1]==1:
        poslines.append(i[0])
    else:
        neglines.append(i[0])

#train-test splitting
possplit=int(len(poslines)*0.6)
negsplit=int(len(neglines)*0.6)

train_clean_sentences= [(x,1) for x in poslines[:possplit]] + [(x,0) for x in neglines[:negsplit]]
y_train=np.append(y_train,[[1]*possplit + [0]*negsplit])

test_clean_sentences= [(x,1) for x in poslines[possplit:]] + [(x,0) for x in neglines[negsplit:]]
y_test=np.append(y_test,[[1]*(len(poslines)-possplit) + [0]*(len(neglines)-negsplit)])

poswords={}
negwords={}


for line,label in train_clean_sentences:
    words= getwords(line)
    for word in words:
        if label==1: poswords[word]= poswords.get(word, 0) + 1
        if label==0: negwords[word]= negwords.get(word, 0) + 1

poswords = { k : v for k,v in poswords.items() if (v>=10 & v<=2000)}
negwords = { k : v for k,v in negwords.items() if (v>=10 & v<=2000)}
```

```
predicted_labels_NB=np.array([])

for testline,testlabel in test_clean_sentences:
    testwords= getwords(testline)
    totpos, totneg= 0.0, 0.0
    for word in testwords:
        a= poswords.get(word,0.0)# + 1.0
        b= negwords.get(word,0.0)# + 1.0
        if ((a!=0.0)|(b!=0.0)):
            totpos+= a/(a+b)
            totneg+= b/(a+b)
    if (totpos>totneg):
        predicted_labels_NB=np.append(predicted_labels_NB,1)
    else:
        predicted_labels_NB=np.append(predicted_labels_NB,0)


#accuracy calculation
print ("\n----------------PREDICTIONS BY NAIVE-BAYES----------------")

ap_pp=0
ap_pn=0
an_pp=0
an_pn=0
acc=0

for i in range(len(y_test)):
            if predicted_labels_NB[i]==y_test[i]:
                acc+=1
                if(y_test[i]==1):
                    ap_pp+=1
                else:
                     an_pn+=1
            else:
                 if(y_test[i]==1):
                    ap_pn+=1
                 else:
                     an_pp+=1
pct= 100/len(y_test)
accuracy= acc* pct
print('   Pos     Neg')
print('  ',ap_pp*pct,'   ',an_pp*pct)
print('  ',ap_pn*pct,'   ',an_pn*pct)
print('Accuracy: ', accuracy,'%')
```

 # NAÏVE-BAYES CLASSIFIER with annotated Dictionary

```
#Importing libraries
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import string
import re
import numpy as np
import random
from collections import Counter

stop = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()

# Cleaning the text sentences so that punctuation marks, stop words & digits are removed
def clean(doc):
    doc=doc.replace('<br />',' ')
    stop_free = " ".join([i for i in doc.lower().split() if i not in stop])
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    normalized = " ".join(lemma.lemmatize(word) for word in punc_free.split())
    processed = re.sub(r"\d+","",normalized)
    y = processed.split()
    for word in y:
        if len(word)<=2:
            del y[y.index(word)]
        word.lower()
    return y


#this method returns important words from a sentence as list
def getwords(sentence):
    w = sentence.split(" ")
    return w

path = "movie.txt"

fp = open('positive_words.txt','r')
posText = fp.read()
posTokens = posText.split("\n")
posTokens[-1] = []


fn = open('negative_words.txt','r')
negText = fn.read()
negTokens = negText.split("\n")
negTokens[-1] = []

train_clean_sentences = []
```

```
y_train=np.array([])
test_clean_sentences = []
y_test=np.array([])
fp = open(path,'r')

ds=[]
for row in fp:
    ds.append([row[:-2],int(row[-2])])

for i in range(len(ds)):
    cleaned= clean(ds[i][0])
    cleaned = ' '.join(cleaned)
    ds[i][0]=cleaned

random.shuffle(ds)

poslines=[]
neglines=[]
for i in ds:
    if i[1]==1:
        poslines.append(i[0])
    else:
        neglines.append(i[0])

possplit=int(len(poslines)*0.8)
negsplit=int(len(neglines)*0.8)

train_clean_sentences= [(x,1) for x in poslines[:possplit]] + [(x,0) for x in neglines[:negsplit]]
y_train=np.append(y_train,[[1]*possplit + [0]*negsplit])

test_clean_sentences= [(x,1) for x in poslines[possplit:]] + [(x,0) for x in neglines[negsplit:]]
y_test=np.append(y_test,[[1]*(len(poslines)-possplit) + [0]*(len(neglines)-negsplit)])

poswords={}
negwords={}


for line,label in train_clean_sentences:
    words= getwords(line)
    for word in words:
        if label==1: poswords[word]= poswords.get(word, 0) + 1
        if label==0: negwords[word]= negwords.get(word, 0) + 1

poswords = { k : v for k,v in poswords.items() if (v>=10 &
v<=max(poswords.values())*2*3)}
negwords = { k : v for k,v in negwords.items() if (v>=10 &
v<=max(negwords.values())*2/3)}

predicted_labels_NB=np.array([])
```

```
for testline,testlabel in test_clean_sentences:
    testwords= getwords(testline)
    totpos, totneg= 0.0, 0.0
    a,b=0.0,0.0
    for word in testwords:
        if word in poswords and word in negwords:
            if word in posTokens:
                a= poswords.get(word)
            elif word in negTokens:
                b= negwords.get(word)
        elif word in poswords:
            a= poswords.get(word)
        elif word in negwords:
            b= negwords.get(word)

        if ((a!=0.0)|(b!=0.0)):
            totpos+= a/(a+b)
            totneg+= b/(a+b)
    if (totpos>totneg):
        predicted_labels_NB=np.append(predicted_labels_NB,1)
    else:
        predicted_labels_NB=np.append(predicted_labels_NB,0)

print ("\n----------------PREDICTIONS BY Dictionary approach----------------")

ap_pp=0
ap_pn=0
an_pp=0
an_pn=0
acc=0

for i in range(len(y_test)):
            if predicted_labels_NB[i]==y_test[i]:
                acc+=1
                if(y_test[i]==1):
                    ap_pp+=1
                else:
                     an_pn+=1
            else:
                 if(y_test[i]==1):
                    ap_pn+=1
                 else:
                     an_pp+=1
pct= 100/len(y_test)
accuracy= acc* pct
print('   Pos     Neg')
print('  ',ap_pp*pct,'   ',an_pp*pct)
print('  ',ap_pn*pct,'   ',an_pn*pct)
print('Accuracy: ', accuracy,'%')
```

 # K NEAREST NEIGHBOR CLASSIFIER

```
#Importing libraries
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import string
import re
import numpy as np
import random
from collections import Counter
from math import log

stop = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()

# Cleaning the text sentences so that punctuation marks, stop words & digits are removed
def clean(doc):
    doc=doc.replace('<br />',' ')
    stop_free = " ".join([i for i in doc.lower().split() if i not in stop])
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    normalized = " ".join(lemma.lemmatize(word) for word in punc_free.split())
    processed = re.sub(r"\d+","",normalized)
    y = processed.split()
    for word in y:
        if len(word)<=2:
            del y[y.index(word)]
    return y

#this method returns important words from a sentence as list
def getwords(sentence):
    w = sentence.split(" ")
    #add bigrams
    w= w + [w[i]+' '+w[i+1] for i in range(len(w)-1)]
    w= list(set(w))
    return w

path = "movie.txt"
train_clean_sentences = []
y_train=np.array([])
test_clean_sentences = []
y_test=np.array([])
fp = open(path,'r')

ds=[]
for row in fp:
    ds.append([row[:-2],int(row[-2])])

for i in range(len(ds)):
    cleaned= clean(ds[i][0])
```

```
    cleaned = ' '.join(cleaned)
    ds[i][0]=cleaned

random.shuffle(ds)
poslines=[]
neglines=[]
for i in ds:
    if i[1]==1:
        poslines.append(i[0])
    else:
        neglines.append(i[0])

possplit=int(len(poslines)*0.8)
negsplit=int(len(neglines)*0.8)

train_clean_sentences= [(x,1) for x in poslines[:possplit]] + [(x,0) for x in neglines[:negsplit]]
#y_train= [1]*possplit + [0]*negsplit
y_train=np.append(y_train,[[1]*possplit + [0]*negsplit])

test_clean_sentences= [(x,1) for x in poslines[possplit:]] + [(x,0) for x in neglines[negsplit:]]
#y_test= [1]*(len(poslines)-possplit) + [0]*(len(neglines)-negsplit)
y_test=np.append(y_test,[[1]*(len(poslines)-possplit) + [0]*(len(neglines)-negsplit)])

train_len= len(train_clean_sentences)
freq={}
trainfeatures= []
for line,label in train_clean_sentences:
    words= getwords(line)
    for word in words:
        freq[word]= freq.get(word, 0) + 1
    trainfeatures.append((words,label))

predicted_labels_knn=np.array([])

for testline,testlabel in test_clean_sentences:
    testwords= getwords(testline)
    results=[]
    for trainwords, trainlabel in trainfeatures:
        #find all words in common between these two sentences
        commonwords= [x for x in trainwords if x in testwords]
        score= 0.0
        for word in commonwords:
            score += log(train_len/freq[word])
        results.append((score, trainlabel))
    results.sort(reverse=True)
    toplab= [x[1] for x in results[:5]]
    numones= toplab.count(1)
    numnegones= toplab.count(0)

    if numnegones> numones:
```

```
        predicted_labels_knn=np.append(predicted_labels_knn,0)
    else:
        predicted_labels_knn=np.append(predicted_labels_knn,1)

print ("\n-------------------------------PREDICTIONS BY KNN-------------------------------------
---")
ap_pp=0
ap_pn=0
an_pp=0
an_pn=0
acc=0

for i in range(len(y_test)):
            if predicted_labels_knn[i]==y_test[i]:
                acc+=1
                if(y_test[i]==1):
                    ap_pp+=1
                else:
                     an_pn+=1
            else:
                 if(y_test[i]==1):
                    ap_pn+=1
                 else:
                     an_pp+=1
pct= 100/len(y_test)
accuracy= acc* pct
print('   Pos     Neg')
print('  ',ap_pp*pct,'   ',an_pp*pct)
print('  ',ap_pn*pct,'   ',an_pn*pct)
print('Accuracy: ', accuracy,'%')
```

#K NEAREST NEIGHBOR using SKLEARN

```
# Importing libraries
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
from nltk.corpus import stopwords
```

```
from nltk.stem.wordnet import WordNetLemmatizer
import string
import re
import numpy as np
import random
from collections import Counter

stop = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()

# Cleaning the text sentences so that punctuation marks, stop words & digits are removed
def clean(doc):
    doc=doc.replace('<br />',' ')
    stop_free = " ".join([i for i in doc.lower().split() if i not in stop])
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    normalized = " ".join(lemma.lemmatize(word) for word in punc_free.split())
    processed = re.sub(r"\d+","",normalized)
    y = processed.split()
    for word in y:
        if len(word)<=2:
            del y[y.index(word)]
    return y


path = "movie.txt"

train_clean_sentences = []
y_train=[]
test_clean_sentences = []
y_test=[]
fp = open(path,'r')

ds=[]
for row in fp:
    ds.append([row[:-2],int(row[-2])])

for i in range(len(ds)):
    #ds[i][0]=' '.join(clean(ds[i][0]))
    cleaned= clean(ds[i][0])
    cleaned = ' '.join(cleaned)
    ds[i][0]=cleaned

random.shuffle(ds)
split=int(len(ds)*0.8)
for i in range(split):
    train_clean_sentences.append(ds[i][0])
    y_train.append(ds[i][1])

for i in range(split,len(ds)):
```

```
    test_clean_sentences.append(ds[i][0])
    y_test.append(ds[i][1])

vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(train_clean_sentences)

# Clustering the document with KNN classifier
modelknn = KNeighborsClassifier(n_neighbors=17)
modelknn.fit(X,y_train)



Test = vectorizer.transform(test_clean_sentences)

#true_test_labels = ['Cricket','AI','Chemistry']
predicted_labels_knn = modelknn.predict(Test)

print ("\n-------------------------------PREDICTIONS BY KNN-------------------------------------
---")

ap_pp=0
ap_pn=0
an_pp=0
an_pn=0
acc=0

for i in range(len(y_test)):
            if predicted_labels_knn[i]==y_test[i]:
                acc+=1
                if(y_test[i]==1):
                    ap_pp+=1
                else:
                     an_pn+=1
            else:
                 if(y_test[i]==1):
                    ap_pn+=1
                 else:
                     an_pp+=1
pct= 100/len(y_test)
accuracy= acc* pct
print('   Pos     Neg')
print('  ',ap_pp*pct,'   ',an_pp*pct)
print('  ',ap_pn*pct,'   ',an_pn*pct)
print('Accuracy: ', accuracy,'%')
```

 # SUPPORT VECTOR MACHINE

```
# Importing libraries
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report,accuracy_score
from sklearn import neighbors, svm
from nltk.corpus import stopwords
from nltk.stem.wordnet import WordNetLemmatizer
import string
import re
import numpy as np
import random
from collections import Counter

stop = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()

# Cleaning the text sentences so that punctuation marks, stop words & digits are removed
def clean(doc):
    doc=doc.replace('<br />',' ')
    stop_free = " ".join([i for i in doc.lower().split() if i not in stop])
    punc_free = ''.join(ch for ch in stop_free if ch not in exclude)
    normalized = " ".join(lemma.lemmatize(word) for word in punc_free.split())
    processed = re.sub(r"\d+","",normalized)
    y = processed.split()
    for word in y:
        if len(word)<=2:
            del y[y.index(word)]
    return y


path = "movie.txt"

train_clean_sentences = []
y_train=[]
test_clean_sentences = []
y_test=[]
predicted_labels_svm=[]
fp = open(path,'r')

ds=[]
for row in fp:
    ds.append([row[:-2],int(row[-2])])

for i in range(len(ds)):
    #ds[i][0]=' '.join(clean(ds[i][0]))
    cleaned= clean(ds[i][0])
    cleaned = ' '.join(cleaned)
    ds[i][0]=cleaned
```

```
random.shuffle(ds)
split=int(len(ds)*0.8)
for i in range(split):
   train_clean_sentences.append(ds[i][0])
   y_train.append(ds[i][1])

for i in range(split,len(ds)):
   test_clean_sentences.append(ds[i][0])
   y_test.append(ds[i][1])

vectorizer = TfidfVectorizer(min_df=5, max_df=0.8,
                        sublinear_tf=True, use_idf=True)

X = vectorizer.fit_transform(train_clean_sentences)
Test = vectorizer.transform(test_clean_sentences)

# Clustering the document with KNN classifier
clf=svm.LinearSVC()#(gamma='auto')
clf.fit(X,y_train)

predicted_labels_svm= clf.predict(Test)

print ("\n-------------------------------PREDICTIONS BY SVM_sklearn-----------------------------
------------")
ap_pp=0
ap_pn=0
an_pp=0
an_pn=0
acc=0

for i in range(len(y_test)):
            if predicted_labels_svm[i]==y_test[i]:
               acc+=1
               if(y_test[i]==1):
                  ap_pp+=1
               else:
                   an_pn+=1
            else:
                if(y_test[i]==1):
                  ap_pn+=1
                else:
                   an_pp+=1
pct= 100/len(y_test)
accuracy= acc* pct
print('   Pos     Neg')
print('  ',ap_pp*pct,'   ',an_pp*pct)
print('  ',ap_pn*pct,'   ',an_pn*pct)
print('Accuracy: ', accuracy,'%')
```

‘‘‘ ---------- **SENTIMENT SUMMARISATION** ---------- ’’’

```
# Importing libraries
import csv
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
from nltk.stem.wordnet import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from scipy.cluster import  hierarchy
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.cluster import KMeans
from scipy.sparse import hstack
import matplotlib.pyplot as plt
import numpy as np
import scipy
import statistics


stopword = set(stopwords.words('english'))
exclude = set(string.punctuation)
lemma = WordNetLemmatizer()


#step 1: Cleaning the text sentences
def clean(doc):
    url_free =' '.join([ch for ch in doc.split() if '.' not in ch and '@' not in ch and """ not in ch
and "" not in ch and "`" not in ch])
    tokenized_word=word_tokenize(url_free)
    stop_free =[i for i in tokenized_word if i not in stopword if i not in exclude]
    normalized =[lemma.lemmatize(word) for word in stop_free]
    return ' '.join(normalized)

def get_cosine_sim(strs):
    vectors = get_vectors(strs)
    vectors = vectors.todense()
    threshold = 0.85     #step 4: clustering on cosine similarity
    Z = hierarchy.linkage(vectors,"average", metric="cosine")
    C = hierarchy.fcluster(Z, threshold, criterion="distance") #step 4: clustering on cosine
                                                               similarity
    return C

def get_vectors(text):
    X = CountVectorizer().fit_transform(text)
```

```
    return(X)


def delete_rows_csr(mat, indices):    #remove the rows denoted by indices form the CSR
sparse matrix mat
    if not isinstance(mat, scipy.sparse.csr_matrix):
        raise ValueError("works only for CSR format -- use .tocsr() first")
    indices = list(indices)
    mask = np.zeros(mat.shape[0], dtype=bool)
    mask[indices] = True
    return mat[mask]
path = 'nepal_eq_2015.csv'
tweetlist= []
dataset=[]
with open(path,'r',encoding='utf8') as csv_file:
    csv_reader = csv.reader(csv_file)
    dataset= list(csv_reader)
del dataset[0]      #delete header row

for row in dataset[:3000]:
    tweetlist.append(clean(row[-1]))        #step 1: extract tweet column

vectorizer = TfidfVectorizer(stop_words='english')
tfidf = vectorizer.fit_transform(tweetlist)     #step 2: tfidf calculation



cos_sim_cluster=get_cosine_sim(tweetlist)      #step 3: cosine similarity calculation

cos_sim_groups=[]
sorted_tweet_index_list=[]

for i in range(1,max(cos_sim_cluster)+1):       #creating groups with same cos_sim_cluster id
    cos_sim_groups.append([j for j,val in enumerate(cos_sim_cluster) if val==i])

for group in cos_sim_groups:            #extracting index of max tfidf score from each group
    tfidf_score= [tfidf[k].sum() for k in group]
    max_tfidf_id= group[tfidf_score.index(max(tfidf_score))]
    sorted_tweet_index_list.append(max_tfidf_id)



#step 5: sentiment analysis
sentiment=[]
sid = SentimentIntensityAnalyzer()
for sentence in tweetlist:
    polarity = sid.polarity_scores(sentence)
    sentiment.append(list(polarity.values()))
```

```
#step 6: represent as vectors
temp=[]
sorted_tfidf=delete_rows_csr(tfidf, sorted_tweet_index_list)
vector=[]
tweet_id=[]
sorted_sentiment=[]
for row in sorted_tweet_index_list:
    tweet_id.append([int(dataset[row][-2][1:-1])])     #tweet id except "
    sorted_sentiment.append(sentiment[row])


#step 7: cluster using kmeans
vector= hstack([sorted_tfidf, sorted_sentiment]).toarray()      #combine tfidf scores with tweet
id
for k in[10,20,30]:
    kmeans = KMeans(n_clusters=k).fit(vector)



    #output tweets
    csvData = [[' index ',' Tweet_id ', ' Sentiment ', ' Cluster id ', ' TFIDF score ', ' Group
number ',' Tweet ']]
    for index in range(len(sorted_tweet_index_list)):
        row=[]
        row.append(index+1)
        row.append(tweet_id[index][0])
        row.append(sorted_sentiment[index][0])
        row.append(kmeans.labels_[index])
        row.append(sorted_tfidf[index].sum())
        row.append(cos_sim_cluster[sorted_tweet_index_list[index]])
        row.append(dataset[sorted_tweet_index_list[index]][-1])
        csvData.append(row)

    with open('Nepal_eq.csv', 'w', encoding="utf-8") as csvFile:
        writer = csv.writer(csvFile)
        writer.writerows(csvData)
        print('Data printed')
    csvFile.close()


    #Statistical Table on sentiment
    kmean_cluster=[]
    for id in range(max(kmeans.labels_)+1):
        kmean_cluster.append([(rindex,sentiment[rindex][-1]) for rindex in
sorted_tweet_index_list if kmeans.labels_[sorted_tweet_index_list.index(rindex)]==id])

    csvData = [[ ' Cluster id ', ' Average ', ' Min ',' Max ', ' SD ', ' Tweet ']]
    for r in kmean_cluster:
        id= kmean_cluster.index(r)+1
        sent_cluster=[t[1] for t in r]
```

```
        mean= sum(sent_cluster)/len(sent_cluster)
        SD= statistics.stdev(sent_cluster)
        minm= min(sent_cluster)
        maxm= max(sent_cluster)
        mid_index=int(len(r)/2)
        final_tweet_index=sorted(r)[mid_index][0]
        tweet=dataset[final_tweet_index][-1]
        csvData.append([id, mean, minm, maxm, SD, tweet])


    with open('final_table.csv', 'a', encoding="utf-8", newline='') as csvFile:
        writer = csv.writer(csvFile)
        #writer.writerows("#considering",k,"Clusters:")
        writer.writerows(csvData)
        writer.writerows("")
        csvFile.close()

#Statistical Table on TF-IDF
    csv_tfidf = [[ '  Cluster id  ', '  TF-IDF Average  ', '  TF-IDF SD  ']]
    for r in kmean_cluster:
        id= kmean_cluster.index(r)+1
        tfidf_cluster=[tfidf[t[0]].sum() for t in r]
        mean= sum(tfidf_cluster)/len(tfidf_cluster)
        SD= statistics.stdev(tfidf_cluster)
        csv_tfidf.append([id, mean, SD])

    with open('tfidf_table.csv', 'a', encoding="utf-8", newline='') as tfidf_File:
        writer = csv.writer(tfidf_File)
        #writer.writerows("#considering",k,"Clusters:")
        writer.writerows(csv_tfidf)
        writer.writerows("")
        tfidf_File.close()
    print('Data printed',k)


#draw bar chart
    cluster_id= list(range(1,k+1))
    cluster_size = [len(groups) for groups in kmean_cluster]

    plt.bar(cluster_id, cluster_size, align='center', alpha=0.5)
    plt.xticks(cluster_id)
    plt.xlabel('cluster id')
    plt.ylabel('size of cluster')
    plt.title('Cluster vs cluster size for k='+str(k))
    plt.show()
```

IX. References

[1] Kim S-M, Hovy E (2004) Determining the sentiment of opinions In: Proceedings of the 20th international conference on Computational Linguistics, page 1367. Association for Computational Linguistics, USA.

[2] Liu B (2010) Sentiment analysis and subjectivity In: Handbook of Natural Language Processing, Second Edition. Taylor and Francis Group, Boca.

[3] Liu B, Hu M, Cheng J (2005) Opinion observer: Analysing and comparing opinions on the web In:

Proceedings of the 14th International Conference on World Wide Web, WWW "05, 342–351. ACM, New York, NY, USA.

[4] Pouransari, H., & Ghili, S. (2014). *Deep learning for sentiment analysis of movie reviews*. Technical report, Stanford University.

[5] Tai, K. S. (2013). Sentiment Analysis of Tweets: Baselines and Neural Network Models. *CS229 Final Project December*, *13*.

[6] Jaitly, A., & Ahuja, S. (2018). IMPROVING THE ACCURACY FOR SENTENCE LEVEL SENTIMENT ANALYSIS. *International Journal of Advanced Research in Computer Science*, *9*(4).

[7] https://crisisnlp.qcri.org/lrec2016/content/2015_nepal_eq.html Accessed on March 19, 2019

[8] Hughes, A. L. and Palen, L. (2009). Twitter adoption and use in mass convergence and emergency events. International Journal of Emergency Management, 6(3-4):248– 260.

[9] Vieweg, S., Castillo, C., and Imran, M. (2014). Integrating social media communications into the rapid assessment of sudden onset disasters. In Social Informatics, pages 444–461. Springer.

[10] https://towardsdatascience.com/summarizing-tweets-in-a-disaster-e6b355a41732 Accessed on 3rd March, 2019

[11] https://www.lexalytics.com/technology/sentiment-analysis Accessed on 6th June, 2019

[12] https://monkeylearn.com/text-classification-support-vector-machines-svm/ Accessed on 3rd March, 2019

[13] https://emerj.com/partner-content/crowdsourced-sentiment-analysis-applications-social-media-customer-service/ Accessed on 13rd February, 2019

[14] http://www.tfidf.com/ Accessed on 5th March, 2018

[15] https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908 Accessed on 3rd January, 2019

[16] https://machinelearningmastery.com/prepare-text-data-machine-learning-scikit-learn/ Accessed on 3rd March, 2019

[17] http://heydenberk.com/blog/posts/sparse-matrix-representations-in-scipy/ Accessed on 13th September, 2018

[18]https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html#scipy.cluster.hierarchy.linkage Accessed on 13th July, 2018

[19] http://t-redactyl.io/blog/2017/04/using-vader-to-handle-sentiment-analysis-with-social-media-text.html Accessed on 7th November, 2018

[20] https://stackabuse.com/k-means-clustering-with-scikit-learn/ Accessed on 23rd August, 2018

[21] https://www.geeksforgeeks.org/numpy-hstack-in-python/ Accessed on 3rd September, 2018

[22] https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html Accessed on 5th March, 2019