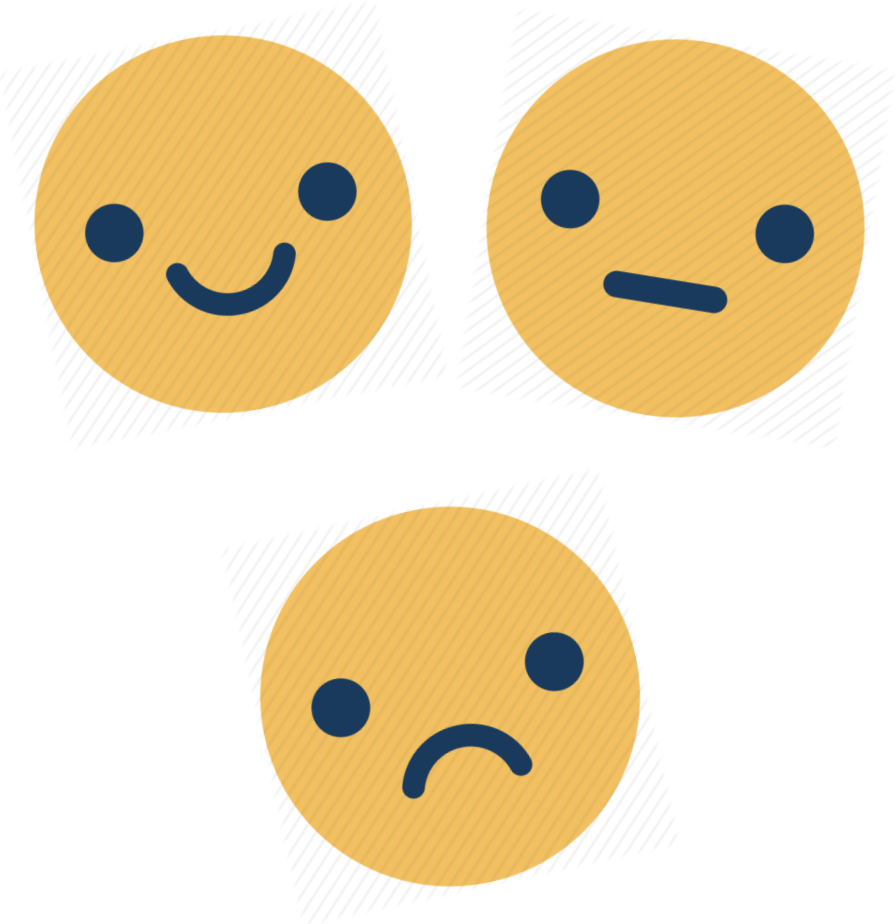
Twitter Sentiment Analysis on 2019 Lok Sabha Election Results



By Vinayak Sharma (Student, *GEC Ajmer*)

under the mentorship of Mr. Amit Dhawan (Scientist-F, *DRDO-ISSA*)

Table of Contents

Acknowledgement

Foremost, I would like to express my sincere gratitude to my Mentor *Mr Amit Dhawan* (Scientist-F DRDO, ISSA) for the continuous support of my Internship Project, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this report. I could not have imagined having a better mentor for my internship.

Twitter Sentiment Analysis on 2019 Lok Sabha Election Results

By Vinayak Sharma (Student, GEC Ajmer)
under the mentorship of Mr. Amit Dhawan (Scientist-F, DRDO-ISSA)

DRDO/ISSA Introduction

DRDO stands for Defence Research and Development Organization. It is an agency of the Government of India and works under the administrative control of Ministry of Defense. As the name suggests, it carries out R&D for Indian Military.

There are around 52 DRDO laboratories engaged in various fields like aeronautics, armaments, electronics, land combat engineering, simulation, life sciences, materials, missiles, and naval systems.

One such lab is *ISSA (Institute for Systems Studies & Analyses)* which is involved in System Analysis, Modeling & Simulation for various defence applications. Other than this, it also provides Software Technologies and develops WarGame for the Military.

Some of its technologies include.

- Land and Naval Wargaming Systems
- Land Weapon Evaluation
- Naval Weapon Evaluation
- No. of Rounds required to achieve desired level of casualties
- Algorithms for Estimation of Target Motion Parameter
- Integrated Air Defense Simulation Testbed
- Tactical tool for ECM mission planning

Problem Statement and Background

India conducted its 17th Lok Sabha (Lower House of Indian Parliament) election in seven phrases from 11th April 2019 to 19th May 2019. This election was particularly interesting because the Narendra Modi led BJP was contesting to get re-elected while Rahul Gandhi led Congress was contesting to redeem itself from the loss it faced in the previous Lok Sabha election.

The counting of votes began on 23rd May 2019 and final list of elected MPs was submitted to The President of India on 25th May 2019. Election was held on 543 seats. Of these, BJP won on 373 while congress won on 52.

Aim of this project was to obtain the high level sentiments (*positive, neutral, negative*) from Indian Tweets that surfaced during vote-counting period and ultimately contextualize those sentiments w.r.t two major political parties in India viz. BJP and Congress.
To achieve this, Machine Learning and various NLP (Natural Language Processing) concepts were used.
In order to understand the mined sentiments, tools like frequency v/s bigram graphs, wordclouds, maps etc. were used.

Such an analysis is very useful in understanding the perception of public towards a subject (political parties in this case).

Data Used:

-Lok Sabha election Tweets were collected via twitter api from 23rd May 2019 to 26th May 2019. After cleaning and filtering, it turned out to be a collection of 5,27,474 tweets in English language along with fields like tweet id, user name & id, creation time, location, followers count, favorite count, verified status etc.

-5,303 random tweets from above data were manually labelled into positive, neutral and negative classes to form training and test set (70-30 split)

-"GeoLite2-City" Database was used to obtain names of 2,034 Indian Cities (which were used to identify Indian tweets in case a user didn't mention country name or state name in their account location but instead just mentioned city name)

-"Indian Census 2011" based shapefile was used to plot a district-wise Indian map

Design/Approach

Following is the high level workflow of project. Each numbered block has been explained later in separate chapters

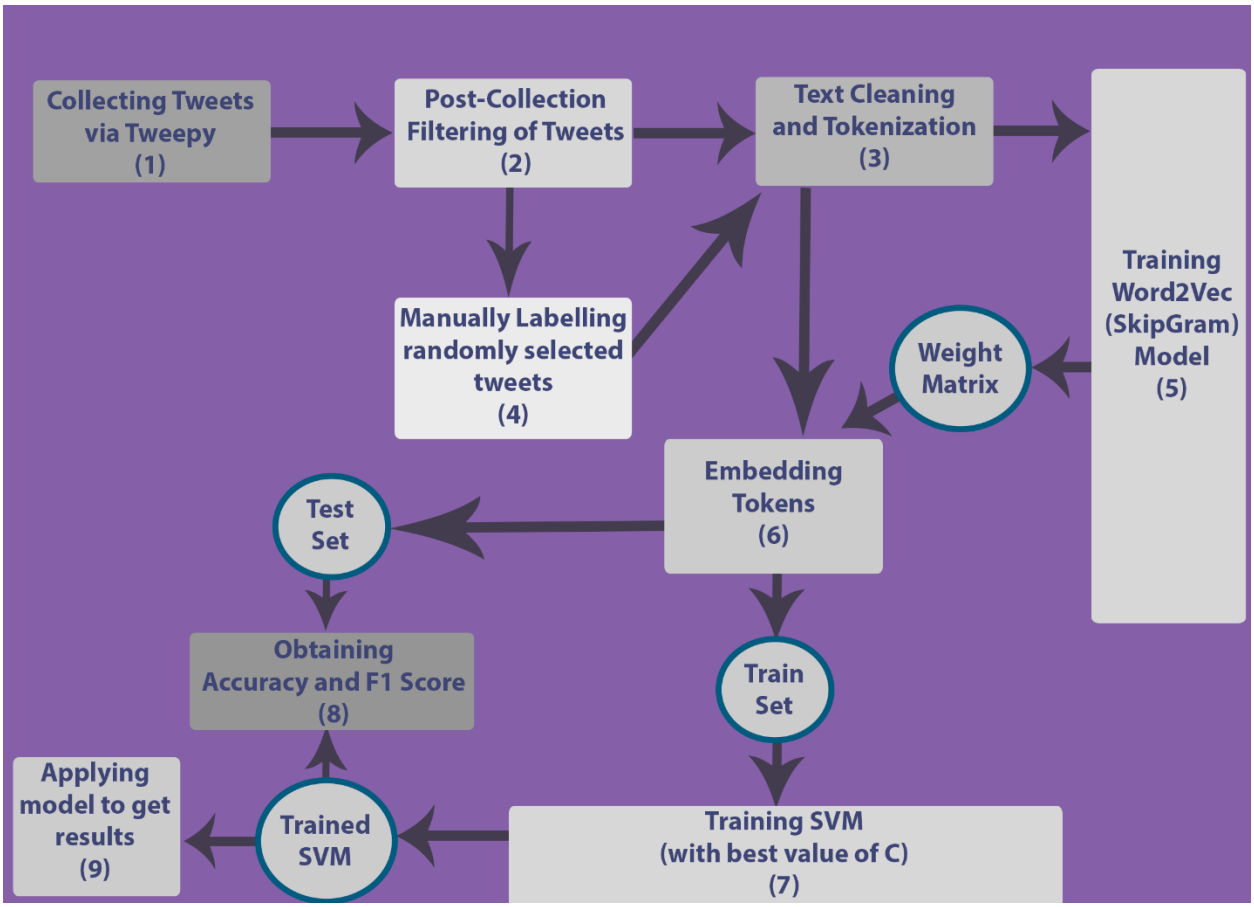


Fig.1

** 'C' in block 7 refers to "Penalty Parameter"*

Chapter 1: Tweet Collection

Overview:

In order to perform this sentiment analysis, tweets related to Indian Election that surfaced during the period of 23rd May to 26th May were needed. Now, there are two ways in which tweets can be mined from twitter, one is live collection of tweets and the other is collection of old tweets (not more than 7 days old). Latter method is not feasible for collecting a large number of tweets because of the twitter API limit (450 requests per 15 minutes).

That's why, it was required to implement live collection of tweets. Such an implementation was difficult on a local machine like laptop/desktop due to internet connectivity and HDD related issues. That's why, *a remote instance was created using* ***Google Cloud Platform*** for live collection of tweets. This was also helpful in later stages (training the ML models) due to the flexibility of modifying the no. of cores and the amount of RAM.

Other than that, a twitter developer account was created to obtain key and token credentials required for *Oauth protocol.* This protocol basically allows a third party app (client) to access details of a user on a server without the user having to give username and password to the third party app.

In this project, "application-only" authentication was used where the app (tweet scraping code) makes an API request on its own behalf to access publicly available data (here tweets) from the server (here twitter)

Pseudo-Code and its explanation:

```
1 key='*******'
2 key_sec='*******'
3 token='*******'
4 token_sec='*******'
5
6 auth=authenticate(key,key_sec,token,token_sec)
7 s=Stream(auth)
8 s.filter([list of keywords])
```

Code.1

In Code.1, lines 1 to 4 contain the key and token credentials required for OAuth protocol.

In line 6, request for authentication is made and the state of authentication is saved in the object "auth". In line 7, an instance "s" is created for streaming live tweets. In line 8, this instance is used to collect only those tweets which contain specific keywords.

Further Details:

- Retweets of a tweet were not collected (only the original tweet was collected)
- Entire json body of a tweet was not collected, only the relevant fields were collected. (This was done to lower the storage per tweet by a significant amount).

  Fields that were collected are *created_at, id, id_str, user_id, user_id_str , user_screen_name, text, extended_tweet, display_text_range, truncated, in_reply_to_status_id_str, in_reply_to_user_id_str , in_reply_to_screen_name , location, utc_offset, time_zone, protected,verified, followers_count, friends_count, listed_count, favourites_count, statuses_count and coordinates.*

- In order to collect tweets relevant to Indian Elections following keyword filters were used in api calls
  ElectionResults2019, Narendra, Election, AcceptTheVerdict , NotMyPM, Verdict2019, ABPresults2019, Modi, Gandhi, Rahul, BJP, Congress, mahagathbandhan, Namo, Pappu, Indian Election, Mamta, Mamata, AAP, SP, BSP, TMC , trinamool congres, NDA , UPA , Bhakt, chokidar, Pradhan

- Despite the need of restricting language to English and location to India, language and location filters of api were not used. This was done because

  1. English is not the first language of Indians and thus twitter's language detection might have missed some tweets with wrong spellings and incorrect grammar despite their context/sentiment being clear.

  2. Location filter only considers a tweet that is geo-tagged and ignore the user's account location. So using it in api request would have caused the loss of a lot of tweets with valid account location.

  Both these filtrations were carried out in the next stage (as explained in Chapter 2)

Chapter 2: Post-Collection Filtration of Tweets

Overview:

Tweets obtained after the collection step had a number of issues like

- Twitter api fetched results which matched even the translated versions. E.g. if the keyword 'Election' was used as a filter, then api would fetch even those tweets which had 'Election' written in Russian, Chinese, French etc.
- Corresponding to keywords 'Congress' and 'PM', api also fetched tweets related to Theresa May's resignation (Brexit) and USA's Congress (Parliament).
- A number of tweets were written in regional Indian languages.

In order to tackle these issues, it was required to further filter the tweets on the basis of their Location (weather they are from India or not) and Language (weather they are in English or not).

Reason for why these filters were not used during the collection step has already been covered in Chapter 1.

Getting location of tweets was particularly challenging because people usually don't mention their account's location in a predefined format (say "locality,city,state,country"). So, there was a need to develop steps to make sense of what's written in location. Also, a significant amount of people don't even mention the account's location at all. For this case, tweet's location (if it was get-tagged) was used. Although, if both account's location and tweet's location were absent then there was no ethical/legitimate way of extracting location of that tweet and thus such tweets were dropped.

Details and Pseudo-Code:

Filtration of tweets by location and by language was performed as explained below

- **By location:** Location of tweets was obtained from the "location" and "place" fields of the tweet. Former field contains the account's location (set by user), while latter field contains the geo-tagged location of the tweet.

  Geo-tagged locations are well defined and are usually in the form of "locality,city,state,country" and thus were used directly to check if the tweet is from India or not.

  Although, account's location are untidy and unpredictable and there is not guarantee that the user would mention state or country name and thus comparing it against a simple list of states or country name wouldn't have sufficed.

  Therefore, following things were done

1. A dataset called GeoLite2-City was used to obtain names of 2,034 Indian Cities (including some of the major cities).
   Then account's location of each tweet was compared against these names (using a regex search)
   This was definitely a slow task but it was a necessary one because it helped in reducing load from an even slower task which was to be done next(a server based location search)

2. In above step, comparison against some (not all) Indian cities was done.
   Also, sometimes people don't even mention city name but instead mention some famous landmark like some heritage site or some university name etc.

   In order to handle such locations and to cover more cities, a server based lookup of account locations was done. For this, geopy library and Nominatim servers were used.
   This was very slow ("Nominatim" servers being free, have a restriction of 1 api request per second).

3. Names of certain cities were frequently misspelled. Some examples of these are Ahm**e**dabad(or Ah**e**mdabad or Ah**e**m**e**dabad or Ahmdabad) and B**a**ngalore (or B**e**ngalore or Bengal**uru** or B**a**ngal**uru**)
   To deal with this, regex patterns having '*' at the erroneous positions of name were used instead of a fixed name.

```
obj=Nominatim()
string_states=[List of of 2,034 Indian Cities, 29 Indian states, 7 Union Territories]

def fun_state(arg):
    loc=arg['location']
    if search(string_states, loc )== True:
        return True
    else:
        return False

def fun_geo (arg):
    loc=arg['location']
    det_loc=obj.geocode(loc)

    if search("India" , det_loc) == True:
                return [True,det_loc]
    else:
                return [False,None]
```

Code.2

In Code.2, line 1 is creating an instance to interact with Nominatim Server.

Line 2 contains list of 2,034 Indian cities, 29 Indian States and 7 Union Territories (obtained from GeoLite2-City Dataset).

Lines 4 to 9 define a function which checks if its argument (a tweet) has a location field with a name that matches with a name in the list of line 2 and accordingly returns True and False.

Lines 12 to 19, define a function which subject its argument's location field to the Nominatim Server api to obtain the location in formal format (locality,city,state,country) . If country's name in the formal location is "India" then tuple containing True and the formal location is returned. Otherwise, tuple containing False and None is returned.

- **By language:** In order to detect language of the text "langdetect" was used. This is a pre-trained Naïve Bayes model coupled with noise filter and character normalization. This model was originally created by *Nakatani Shuyu* in Java.

  Text of tweet was subjected to this model to check if it was in English or not.

```
1  def fun_lang (arg):
2      s=arg['text']
3
4      d=lang_detect(s)
5
6      for el in d:
7              if el.lang=='en':
8                  if el.prob>0.4:
9                      return True
10
11     return False
```

Code.3

In Code.3, lines 1 to 11 define a function that checks if text in tweet is in English or not and accordingly returns True or False.

In line 4, langdetect classifier is being used to obtain a list of possible languages along with their probabilities. Then in lines 6 to 9, this list is traversed to search for English language. If English language with probability > 40% is found only then True is returned, otherwise False is returned.

Chapter 3: Text Cleaning and Tokenization

Overview:

Tweet's text in its original form can't be used directly. This is because in that form, text contain things which are sensible from grammatical point of view but don't really contribute anything from analytical point of view. Some of these things are *use of capital/small letters, stopwords, punctuation marks etc.*

Other than that, things like *URLs, hashtags, mentions and emoticons* also need special handling.

Pseudo-code and Details:

```
1  p='pattern string for matching hashtags,mentions, numbers and URLs'
2  p2='pattern string for matching non-alphabetic characters'
3  emoji_pattern = 'pattern string for matching emoji'
4  sw=[list of stopwords]
5  def f_clean(arg):
6      s=arg['text']
7      s=s.lower()
8      space=' '
9      s_=space.join(char for char in s )
10     l=findall(emoji_pattern , s_)
11     for e in l:
12         s=sub_name(e , s)
13
14     s=sub_blank(p , s)
15     s=space.join(s.split())
16
17     s=sub_blank(p2 , s)
18
19     s_=''
20     l=s.split()
21     for e in l:
22         if e not in sw:
23             s_=s_+' '+e
24     return s_
```

Code.4

```
1 df=read_csv('tweets.csv')
2 df['tokens']=df['text'].apply(tokenize)
3 phrases = Phrases(sentences=df['tokens'],min_count=10,threshold=5)
4 bigram = Phraser(phrases)
5 def f(arg):
6     return bigram[arg['tokens']]
7
8 df['tokens']=df.apply(f,axis=1)
9 df.to_csv('TokenizedTweets.csv')
```

Code.5

- **Lowercase:** Entire text was made lowercase. This was done so as to make form of words (/tokens) consistent across corpus. E.g. “Delhi” maybe written in various forms like “delhi”,”DELHI”, ”DelHI” etc. But since each of these forms mean the same thing, it is important to force a particular form.

  Line 7 of Code.4 turns text into lowercase.

- **Emoticons:** Emoticons are very helpful in identifying sentiment. Therefore they were retained. Although, they were not retained in their Unicode form but instead they were replaced with their names. E.g. a thumps up emoji was replaced with “thumpsxup” (all words in the name were concatenated with ‘x’ as joint) .This was done because later all the non- alphabetic characters were to be removed and that would have destroyed emoji’s Unicode form.
  In order to get names of emoji, “unicodedata” and “emoji” libraries were used.

  Lines 9 to 12 of Code.4 handle emoticons. In line 9, string is broken into characters separated by spaces, this is needed to separate adjacently joined emojis (otherwise they won’t be identifiable). Line 10 finds all the emoji characters in text and puts them in the list “l”. In lines 11 and 12, this list is traversed and all emoji characters of text are replaced with corresponding emoji names.

- **Urls, hashtags and mentions:** Urls were removed. Twitter mentions were also removed because they acted like neutral nouns. Hashtags on the other hand were retained (only the ‘#’ was dropped) because they contain relevant information about the trend/context.

  John Gruber’s prebuilt URL matching regex, was used to identify and remove URLs

  Lines 14 and 15 of Code.4 handle URLs, hashtags and mentions. Line 14 substitutes URLs, mention strings and ‘#’ with blank. Line 15 compresses multiple whitespaces to

single whitespace.

- **Non-alphabetic:** All non-alphabetic characters were removed.

  Line 17 of Code.4 removes all non-alphabetic characters.

- **Stopwords:** Stop words are those words in a language which don't convey any relevant sense e.g. "I", "you", "a", "the" etc. Some of the major stopwords of English language were removed from tweet's text (NLTK's English stopword list was used for this purpose)

  Although, stopwords related to the "negation" like "no", "didn't" , "can't" etc. were retained. This was done so that model could identify the difference between sentences like
  "I like it" and "I don't like it" (Although, model at the time of writing this report was not able to detect such differences due to lack of suitable data of this kind. But still retaining negation stopwords might useful for any future changes)

  Other than language specific stopwords, there are also some *context based stopwords.* In context of Indian election tweets, some of such stopwords are certain nouns like politician's name, journalist's name, political party's name, name of religion, name of state etc.
  Such names don't really contain any sentiment by themselves unless a name has been modified in a respectful or derogatory sense. e.g. names "Rahul" and "Modi" by themselves don't have sentiments. But names "Pappu" and "Feku" do have derogatory sentiments.

  So keeping all this in mind, neutral nouns were removed from text.

  Lines 19 to 23 of Code.4 reconstructs the text after ignoring stopwords.

- **Tokenization:** Tokenization is the task of breaking text into relevant entities. In NLP, these entities can be single characters, pair of characters, single words, pair of words, triplet of words etc.

  Here, for tokenization we used an implementation of "Phrases" which was first introduced in one of Word2Vec papers (called "*Distributed Representations of Words and Phrases and their Compositionality"*).
  This technique basically finds pair of words which are more likely to be used together (word-bigrams) like "New Delhi".
  This is achieved by using a simple equation. No. of times given words occur together and

the no of times they occur separately is fed to this equation to obtain a score. More is this score, more likely the words are to occur together.

For this project, "gensim" library's implementation of phrases was used.

Code.5 deals with tokenization. Line 1 reads a csv file containing all the tweets into a dataframe. Line 2 performs basic tokenization over text of each tweet in dataframe. This tokenization simply breaks sentence into single word tokens A new field "tokens" is created in dataframe to store list of all such tokens per tweet. Lines 2 and 3 use Phrases and Phraser methods of gensim library to get an object "bigram". In lines 5 and 6, this object is used to obtain bigram tokens from ordinary tokens.

Chapter 4: Labelling Data

Filtered and cleaned tweets were shuffled randomly and then a chunk of these were manually labelled into positive, neutral and negative classes. Overall, 5,303 tweets were labelled.

Following labelling scheme was followed

- **Positive:** a tweet was labelled positive if it had following traits
    1. **Honor**
    2. **Request**
    3. **Praise**
    4. **Wish**
    5. **Salutations**
    6. **Hope**
    7. **Victory**
    8. **Positive Sentiment Emojis**

- **Negative:** a tweet was labelled negative if it had following traits
    1. **Anger**
    2. **Threat/Dare**
    3. **Criticism**
    4. **Hate**
    5. **Sorrow**
    6. **Harsh**
    7. **Taunt**

8. **Loss**
9. **Negative Sentiment Emojis**

- **Neutral:** a tweet was labelled neural if it had following traits
    1. **Emotionless Commentary**
    2. **News (without praise / criticism)**
    3. **Vote count updates like trailing / leading**
    4. **Same amount of positive and negative traits**

If a tweet had both positive and negative traits then decision was made in favor of that class which was more prevalent in the text.

Tweets with unclear context were not labelled. Also tweets which had local languages written using English alphabets were also not labelled.

Chapter 5: Training Word2Vec Model

Overview:

Initial design choice was to use TF-IDF (Term Frequency and Inverse Document Frequency) technique to extract features from text and then use them with train data for training. *TF-IDF basically gives higher value to a token which occurs more within a document but not across documents. This helps highlighting more important/interesting tokens*. Although, while using TF-IDF with given amount of train data, accuracy of SVM classifier was at best 70%.

There could have been multiple problems leading to this accuracy, but one particular problem that was put under the microscope was that trained model wasn't able to understand words that weren't present in the training example despite the presence of other words in training example with similar meaning.

E.g. say, the word "happy" was present in the training set but the word "glad" wasn't then on encountering "glad", model won't be able to relate the two words and thus wouldn't identify the sense of "glad". To tackle this problem design choice was changed from use of TF-IDF to the use of Word2Vec embedding. This was done because, word2vec embedding assign similar values to tokens with similar meaning/sense and dissimilar values otherwise and this gives the machine a way to identify tokens with similar meaning/sense.

Details and Pseudocode:

Word2Vec is a neural network based model that was introduced by a team of researchers at Google. It consists of a shallow neural network (having single hidden layer). There are two variations of Word2Vec viz. CBOW (Continous Bag of Words) and Skipgram.

Word2Vec takes word at input layer and predict surrounding words (context) at output layer (if using Skipgram) or visa-versa (if using CBOW).

To obtain hidden layer, a *linear activation* is used while to obtain output layer, *softmax activation* is used.

Although, in practice, the content obtained in the output layer isn't really used. What's actually used is *the optimized weighs in the first weight matrix* (matrix which transforms input layer to hidden layer).

Rows or columns of this weight matrix (depending on how matrix was implemented) infact act as the "vectors" corresponding to "words" (hence the name "word to vec"). Said in a different way, *word2vec can be used to transform words or tokens in the corpus to vectors. And these vectors are such that, if two words have similar sense then values of their vectors will also be similar*. Intuitively, this is because, if two words have similar sense then they will be used in similar context and thus need to put similar things at output layer and for that they need similar values in hidden layer (which is linearly obtained from the first matrix itself)

An important thing to note is that word2vec doesn't require labelled data

In this project, entire collection of filtered and cleaned tweets (unlabelled) was used to train gensim's implementation of Word2Vec. Weight matrix obtained from this was used to embed features as explained in Chapter 6.

```
1 df=pd.read_csv('TokenizedTweets.csv')
2
3 cores = cpu_count()
4 tokens=df['tokens']
5
6 obj_w2v= Word2Vec(sentences=tokens, min_count=20,window=2,size=300,workers=cores-1,iter=30,sg=1)
```

Code.6

In line 6 of Code.6, training of word2vec happens using settings as suggested by parameters to the method "Word2Vec" (of genism library)

These parameters and their meaning are as below

min_count: ignore all words with frequency less than min_count
window: maximum distance between the current and predicted word within a sentence
size: length of resulting vector per word
workers: no of worker threads to be used to train model

sg: using SkipGram model(if sg==1) otherwise CBOW model
iter: no of iterations over corpus

Chapter 6: Embedding Tokens (/Features)

Overview:

Weight matrix obtained after training Word2Vec was used to embed the tokens of tweet. Embedding is the process of converting an entity into a suitable form so that machine can understand it.

What such Word2Vec based embedding intuitively does, is assign similar embedded values to tokens with similar meaning/sense and dissimilar values otherwise.

Mathematically, it puts similar tokens (/features) along axis with smaller angle between them instead of putting all tokens along equally spaced axis.

Practically, what it does, is solve the problem where trained model can't understand a word because it wasn't present in the training example despite the presence of another word in training example with similar meaning.

Usually, for embedding, word2vec models pre-trained on a very large corpus (like some Google or Amazon's corpus) are used so that a large number of words can be sampled. Such pre-tained models were also tried for this project but they were not giving the desired accuracy (e.g. Google-News model gave max accuracy of 72.62%). This was mainly because a lot of local words and references that are used in India were missing from them. Thus a separate word2vec model was trained on the collected election tweets corpus (as already covered in Chapter 5)

Details and Pseudocode:

Following steps were performed for embedding

1. For each token of every tweet, it was checked if token was present in the word2vec vocabulary or not.
2. If it was present in vocabulary, then "word-vector" of that token was retrieved from the weight matrix and an average of all values in that vector was taken. (This average value is called embedded value and is used as a representative of the token for the machine)

3. If token wasn't present in vocabulary then, 0 was used as embedded value

```
1  def get_average(tokens_list, w2v, k):
2      if len(tokens_list)==0:
3          return zeros(k)
4
5       v = [w2v[word] if word in w2v else zeros(k) for word in tokens_list]
6       length = len(v)
7
8      summed = sum(vectorized, axis=0)
9      averaged = divide(summed, length)
10
11     return averaged
```

Code.7

In Code.7, lines 1 to 11 define a function which computes embedded value of each token in the token list. Line 5 checks if the token is present in word2vec vocabulary or not and accordingly retrieve word vector from weight matrix. Lines 8 and 9 takes average along word vector of each token.

Chapter 7: Training SVM

Overview:

Support Vector Machine (SVM) is a supervised learning algorithm.
At a very high level,
- for linearly separable data, SVMs tries to fit straight line boundary such that margin between that line and closest training examples (called support vectors) is max.
-for data that isn't linearly separable, SVMs use kernels which are the functions that provide similarity score between different training examples.

In this project, SVM with linear kernel was used. This was done, because no. of training examples were smaller than no. of features, so using nonlinear kernels would have caused the issue of overfitting as there was not enough data to fit complex hyper-plane properly.

Pseudocode:

```
1  df=pd.read_csv('TokenizedLabelledTweets.csv')
2
3  X_train, X_test, y_train, y_test = train_test_split(df['embed'],
4                                                      df['labels'],
5                                                      test_size=0.3)
6  learn_obj=svm(C=0.05,kernel='linear')
```

Code.8

In Code.8, line 3 partitions the labelled tweets into a 70-30 train-test split.

In line 6, a linear SVM classifier is being trained with best possible value of C (explained below)

Further Details:

Methods of "sklearn" library were used for training SVM.

Before performing the final training, a suitable value of penalty parameter (C) was selected. This was necessary because C decides the trade-off between *bias and variance*.
By varying C from a large value to a small value, bias and variance change from (Low Bias, High Variance) to (High bias, Low Variance).

To select a suitable value of C, it was one by one varied in the following intervals (using grid search)

[0.000001,0.000003,0.00001,0.00003,0.0001,0.0003,0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, 3],

[0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09]

and [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]

For testing models trained with different Cs, 10-fold cross-validation scheme was used (with 0.4 being the fraction of cross validation part) *instead of using the test set*.

Best value of C was 0.05 with an average cross-validation score of 75.23%

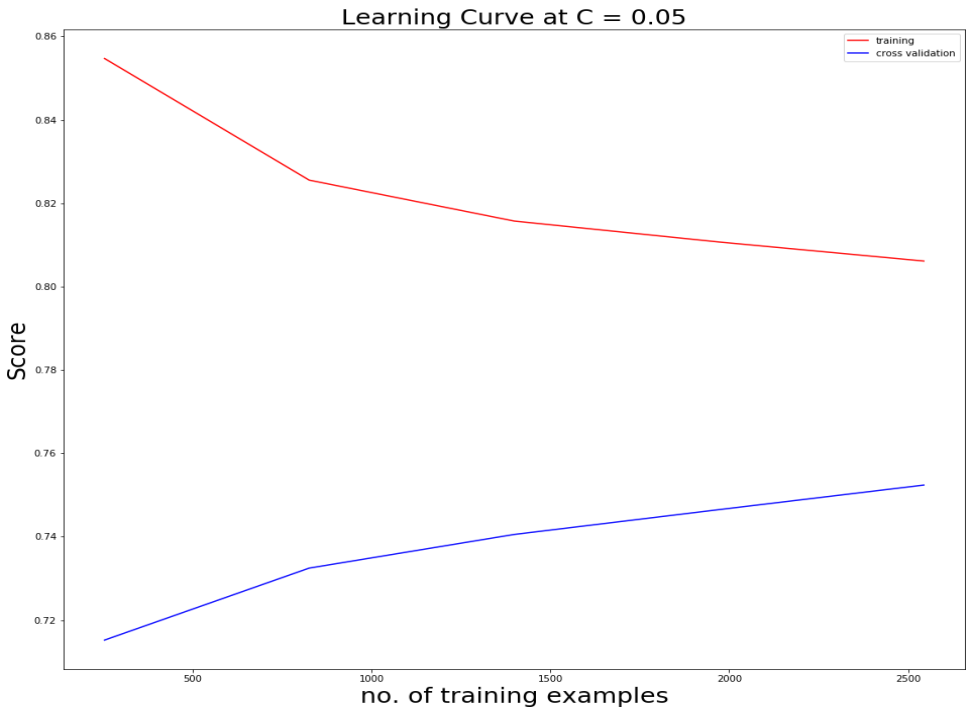


Fig. 2

Chapter 8: Accuracy and F1 scores

	precision	recall	f1-score	support
Rneu	0.77	0.67	0.72	333
neg	0.74	0.82	0.78	634
pos	0.77	0.75	0.76	624
accuracy			0.76	1591
macro avg	0.76	0.75	0.75	1591
weighted avg	0.76	0.76	0.76	1591

Fig.3

**Rneu above means neutral*

Against test set, trained SVM model gave an accuracy of 76.05%

Same test set was subjected to Vader (a lexical text analyzer) and its best accuracy was 58.6%

Chapter 9: Results

For better quality of results, *only those tweets were used for whom predictions had confidence values >= 60% (for negative and positive classes) and >= 50% (for neutral classl).*

Doing this, trimmed the number of usable tweets from 5,27,474 to 3,98,749.
This drop is quite consistent with our model's accuracy of 76% {5,27,474 * 0.76 == 4,00,880}
Additional loss of 4,00,880-3,98,749 == 2131, can be attributed to the fact that while labelling, tweets with unclear context and tweets with local language (written using English alphabets) were skipped. So confidence values will be low for such tweets.

Following things were plotted to understand result.

a) WordCloud (Fig-A)
b) Bigram Frequency Curves (Fig-B)
c) Sentiment v/s Date, Bar Graphs (Fig-C)
d) Geo-Tagged Sentiments (Fig-D)

(Each of these results will be explained at the end of their plots)

Of these (b), (c) and (d) were also plotted w.r.t two major Indian political Parties viz. BJP and Congress.

In order to plot content for BJP and Congress, *tweets that exclusively contained keywords related to one party (and no keyword related to other party) were used.*
This was done, because for tweets which had keywords of both the parties, it would be difficult to find if sentiment in the tweet was meant for one party or the other.

e.g. consider a statement "I Hate A, A is the worst . A should quit politics. But I like B" . This statement will be classified as negative because a large part of statement has negative sentiment. Now this statement contains name of both A and B so weather we pick this statement by considering A as keyword or by considering B as keyword, it will always be a negative statement despite the fact that writer had negative sentiments for A but positive sentiments for B.
So for this reason, tweets exclusively meant for BJP and those exclusively meant for Congress were identified.
For this purpose certain keywords were decided for BJP and Congress.

Out of 3,98,749 tweets, there were 1,69,705 tweets that exclusively contained the BJP keywords and 78,975 tweets that exclusively contained the Congress keywords and 1,50,069 tweets that either contained both BJP and Congress keywords or contained neither.

Difference between no. of exclusive BJP and Congress tweets can be understood by the fact that BJP was getting a lot of congratulatory tweets (it being the party that won on 303 out of 554 Lok Sabha seats) and such tweets usually contain a single subject (entity that is getting congratulated).
This point can be validated by looking at Fig. A.1 and Fig B.3.1 where congratulatory terms have exceptionally high frequencies.

Note 1: Map figures (Fig. D) were plotted using only those tweets which had district names in their location. *There were 2,08,624 such tweets in total. Of which 97,345 were exclusively BJP related (and thus used for plotting BJP maps) and 45,108 were exclusively Congress related (and thus used for plotting Congress maps).*
Moreover, in Maps only highly polarized sentiment intensity (defined later under "Get-Tagged

Sentiments" subsection) was plotted.

Note 2: Plots which are not wrt BJP or Congress (like Fig. A, Fig. C.1 etc.) were infact plotted considering all the usable tweets (3,98,749 tweets) with no restrain on them being related to any party or subject.

Note 3: In plots below, pos means positive, neg means negative, and neu means neutral

Note 4: All the following plots should be observed comprehensively with reference to each other (instead of jumping to conclusion after observing just one or two)

Wordclouds

Wordclouds are a good way of visualizing the words in corpus on basis of their frequency. Larger the frequency, bigger is the word. Keep in mind, wordclouds below are not wrt an exclusive subject (say a party), so we can't make any confident assertions just by looking at them (as already explained)

Positive WordCloud

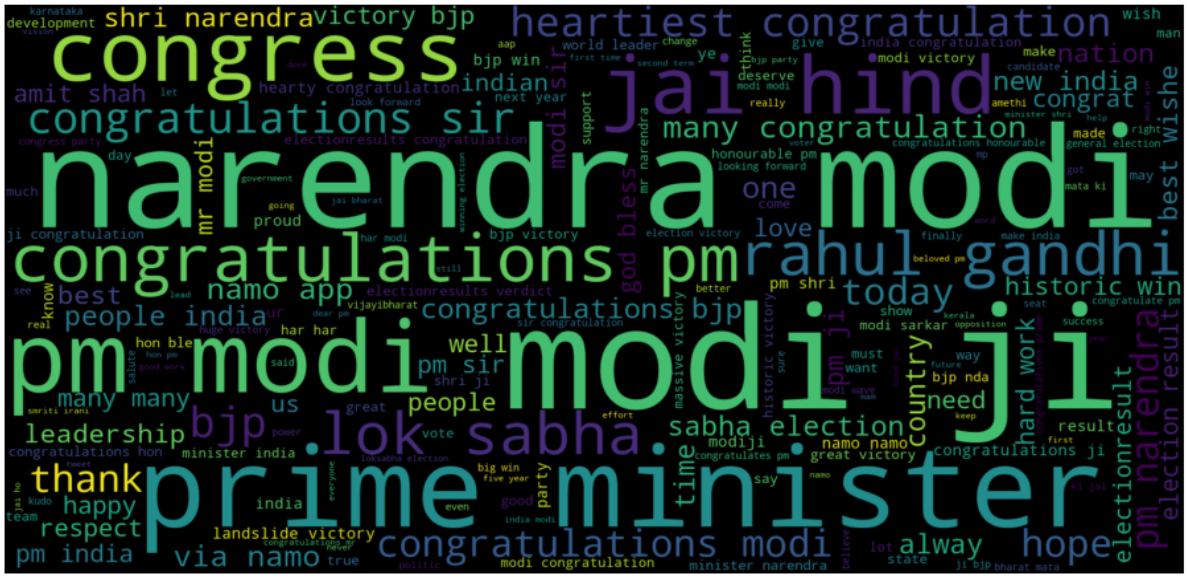


Fig. A.1

From positive wordcloud, it is obvious that a large number of positive tweets were of congratulatory nature (most were targeted to "Narendra Modi" or "Modi Ji" or "prime minister")

Negative WordCloud

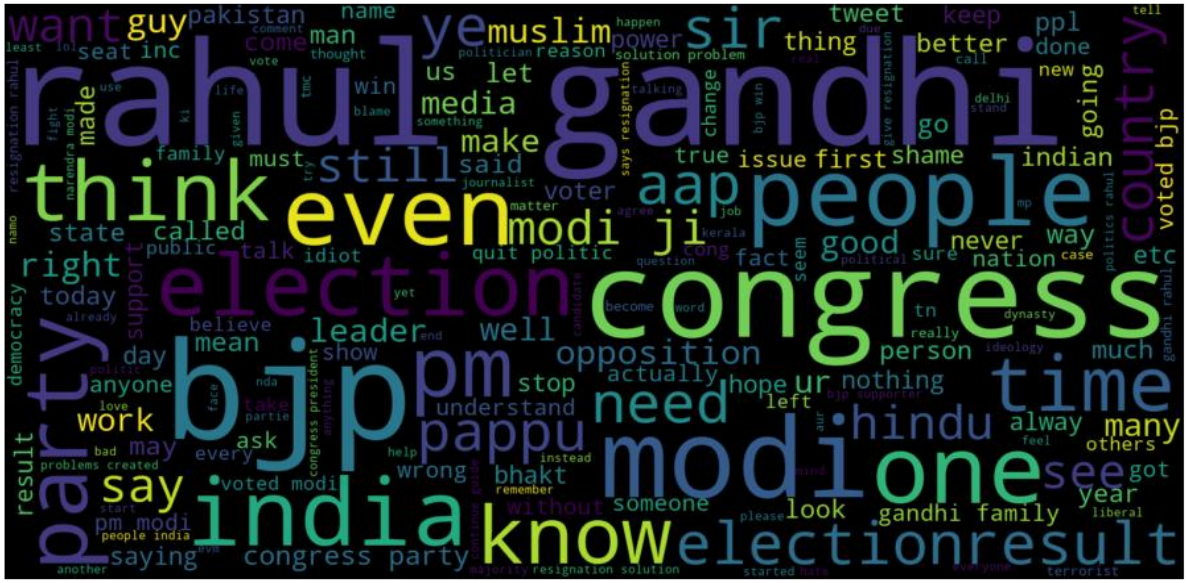


Fig. A.2

Negative wordcloud contains much more diverse terms.
Name “Rahul Gandhi” (former party president of Congress) seems to have a very large no. of occurrences in negative tweets. After that the names of parties “BJP”,”Congress” and after that name “Modi” (Main leader of BJP) have high number of occurrences in negative tweets.
Two of the largest religion (in terms of no. of followers) of India, viz. “Hindu” and “Muslim” also made it to the most frequent words in negative tweets

Neutral WordCloud

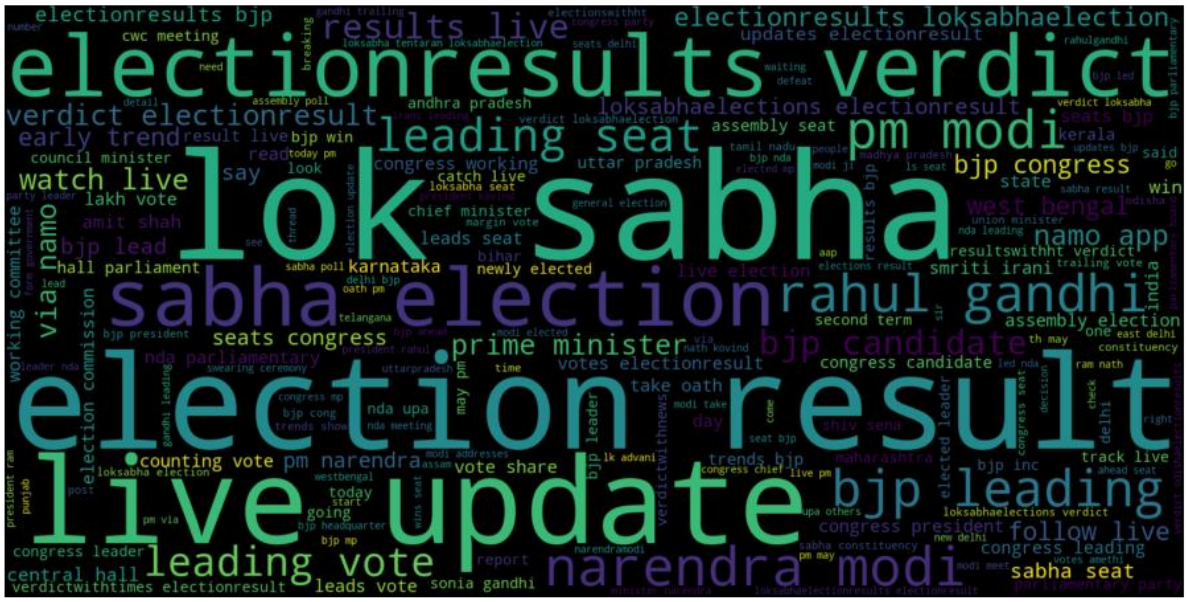


Fig. A.3

From Neutral wordcloud, it is obvious that a large number of neutral tweets contained stuff like vote-count/status update or any other news related to election.

Frequency v/s Bigram Curves

This plot is very helpful in giving a good insight about the context. Reason for this is that word bigrams give a much better idea of context that single words. E.g “bad” and “person” by themselves don’t provide a clear context but the bigram “bad person” provides a much clear context. So by looking at highest frequency bigrams we can contextualize sentiments a lot better.

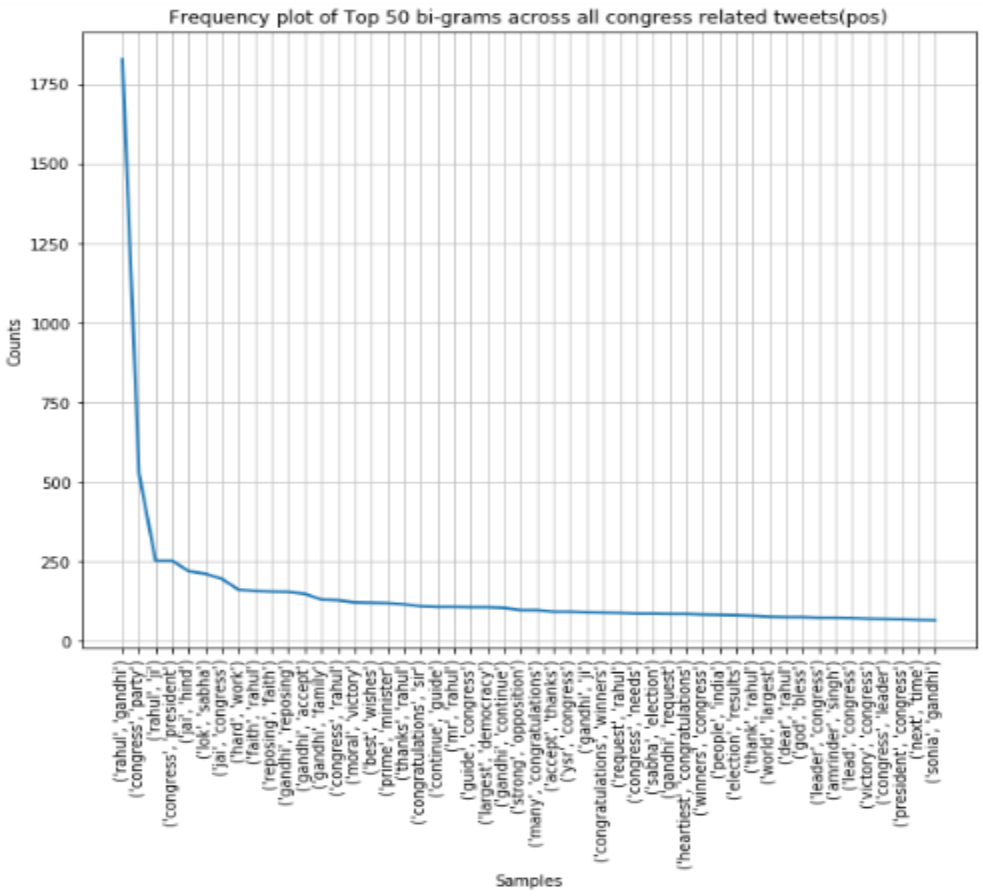


Fig. B.2.1

- Positive Congress Bigrams
  1. Bigrams like (moral,victory) suggest tweets about moral victory of congress
  2. Bigrams like (faith,rahul), (repose,faith), (continue,gandhi), (lead,congress),(congress,leader),(congress,preseident) suggest tweets about having faith in rahul (external context here is that after Congress's poor performance, Rahul offered his resignation from Party president position but congress turned the resignation down saying it has faith in Rahul' leadership)
  3. Bigrams like (request, rahul),(dear, rahul), (congress, needs) suggest tweets where people are giving Rahul some suggestions for improvements in congress
  4. Bigram (amarinder, singh) suggest positive tweets for Amarinder Singh (a seasoned Congress leader from Punjab under whose leadership congress won on larger no of seats in Punjab)
  5. Bigrams like (heartiest, congratulations), suggest that Congress also got congratulatory tweets (not as much as BJP though)

6. Bigrams (next,time) suggest tweets hoping for better performance of congress next time
7. Bigram (strong, opposition) suggest tweets wanting congress to improve to become strong opposition.

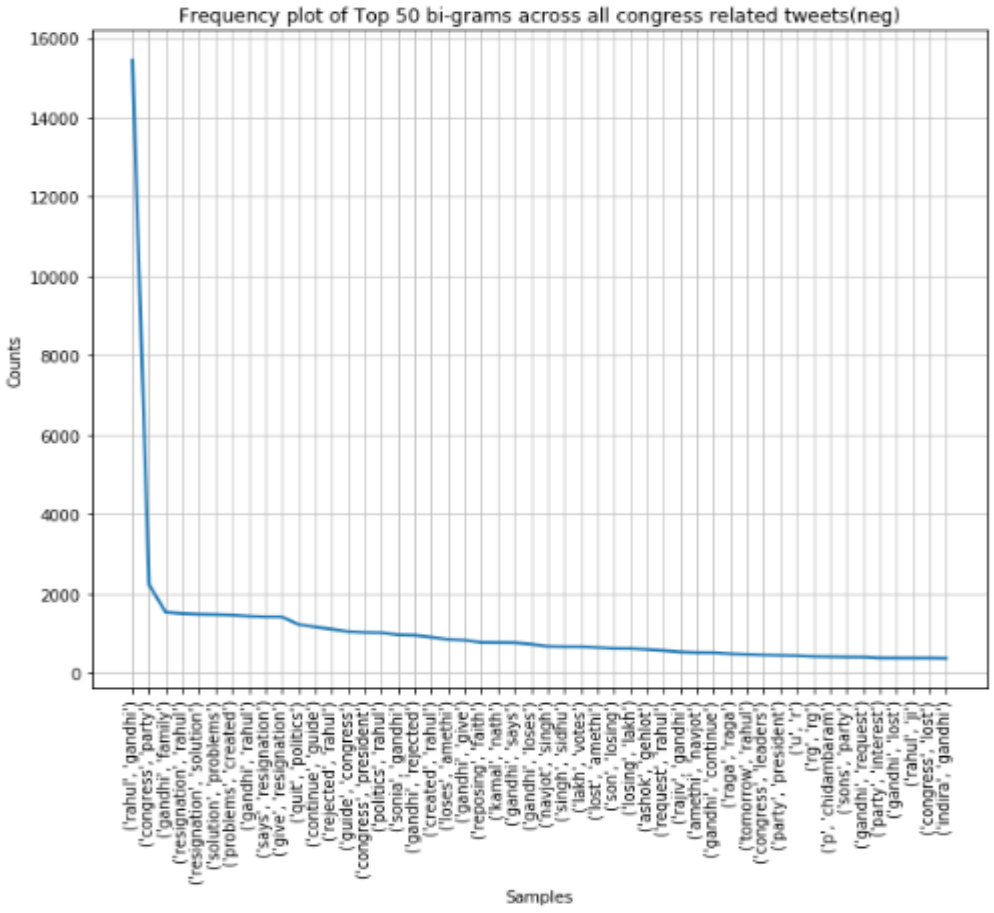


Fig. B.2.2

- Negative Congress Bigrams
    1. Bigrams like (gandhi,family), (sons,party) suggest tweets lashing out at Gandhi family or more practically at the concept of family politics or hierarchical politics.
    2. Bigrams like (quit,politics) ,(resignation,rahul),(give,resignation), (congress,president) suggest tweets asking Rahul Gandhi to resign from post of congress president or from politics in general

    3. Some congress politicians like Navjot Singh Siddhu, P.Chidambram, Ashok Gehlot and Kamal Nath got negative tweets directly targeted at them.

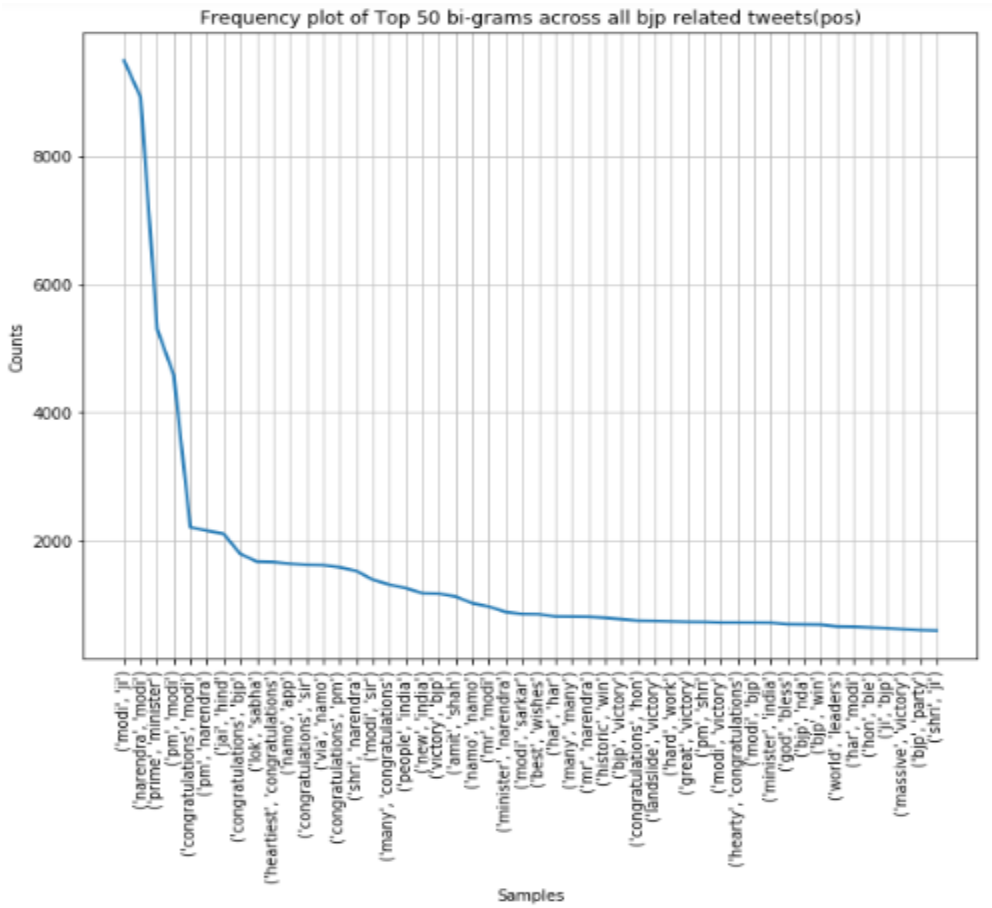


Fig. B.3.1

- Positive BJP Bigrams
    1. As already observed with wordcloud, almost all positive BJP tweets are about congratulating BJP or Modi.
    2. Bigrams like (new,india) suggest tweets where people believe that BJP's victory will bring positive changes to India
    3. Bigram (hard,work) also exist in positive BJP tweets and suggest tweets complementing BJP on its hard work

    4. There are a large number of positive tweets about another BJP leader Amit Shah as well

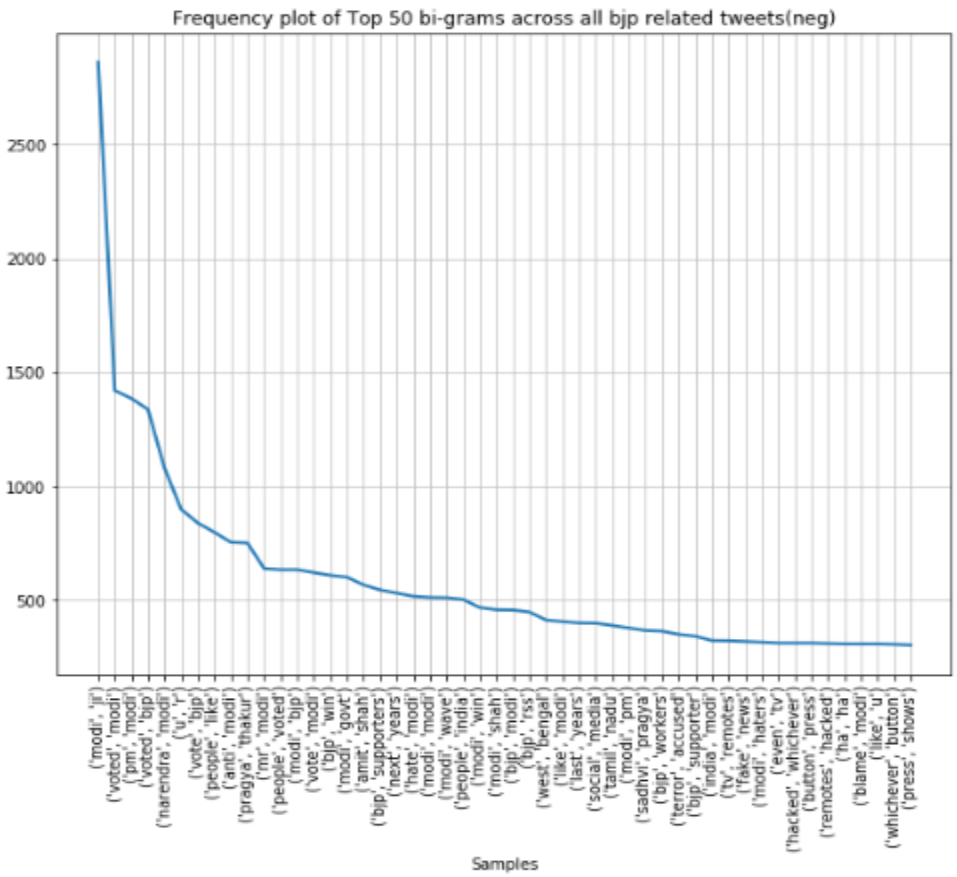


Fig. B.3.2

- Negative BJP Bigrams
    1. Bigrams like (voted, modi), (voted,bjp) ,(bjp, supporters) suggest tweets against people who voted for bjp or modi
    2. Bigram (anti, modi) , (modi,haters) suggest tweets against people who oppose Modi
    3. Bigrams like (tamil,nadu) and (west,bengal) suggest negative tweets related to states of Tamil Nadu and West Bengal
    4. Bigrams like (pragya,thakur),(sadhvi,pragya) , (terror,accused) suggest tweets against BJP candidate Pragya Thakur (who is an accused in a terror attack case)
    5. Bigrams like (social,media), (fake news) suggest tweets criticizing fake news on social media.

Sentiment v/s Date Bargraphs

In events like vote counting, which span over multiple days, plotting content against date is very useful because it can tell us how the trends change with date and with that we can find what triggered that change by looking at what happened on that date.

In bar graphs plotted below, Sentiment Intensity is defined as

a) *If plotting without subject*:
Sentiment Intensity == (Total number of tweets belonging to given sentiment on given date) / (Total no of tweets on given date)

b) *If plotting with a subject:*
Sentiment Intensity == (Total number of tweets belonging to given sentiment in given partial_set on a given date) / (Total no of tweets in union_set on a given date)

where, partial_set may mean “set of all BJP related tweets” or “set of all Congress related tweets”
while, union_set is union of “set of all BJP related tweets” and “set of all Congress related tweets”.

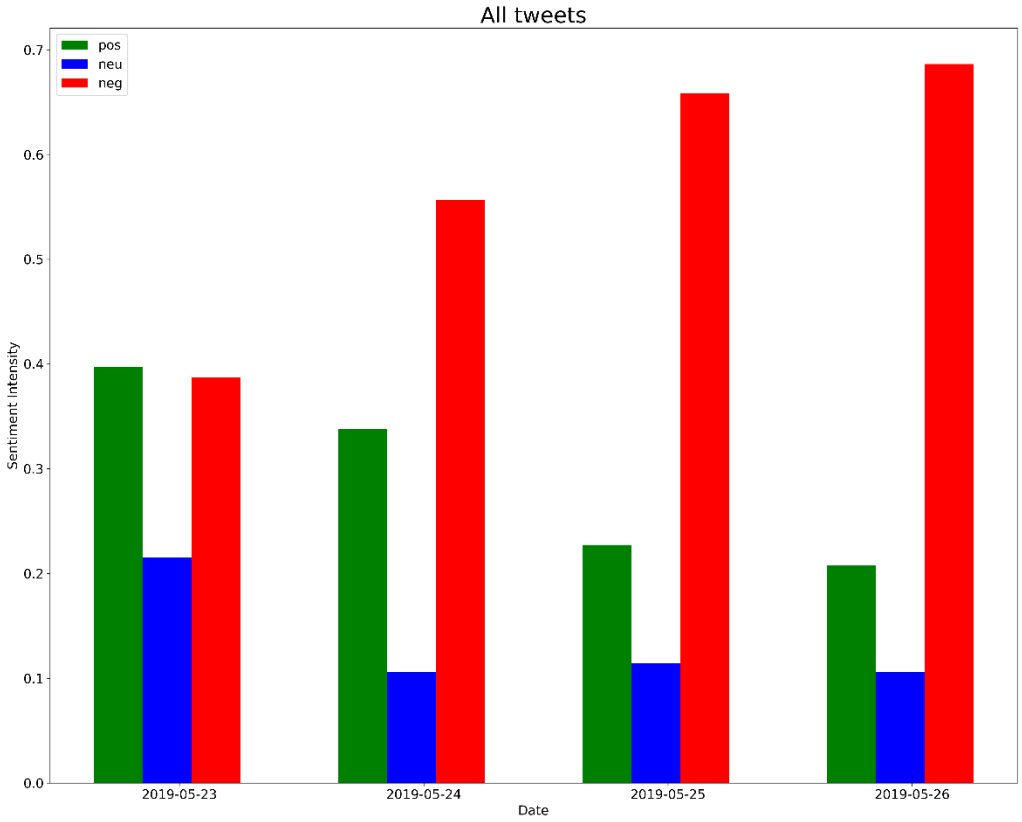


Fig. C.1

<u>All tweets:</u>

Variation in sentiments was observed only during 23rd and 24th May, after that sentiments more or less saturated to same values. This is mainly because fate of election was clear by the end of 24th May.

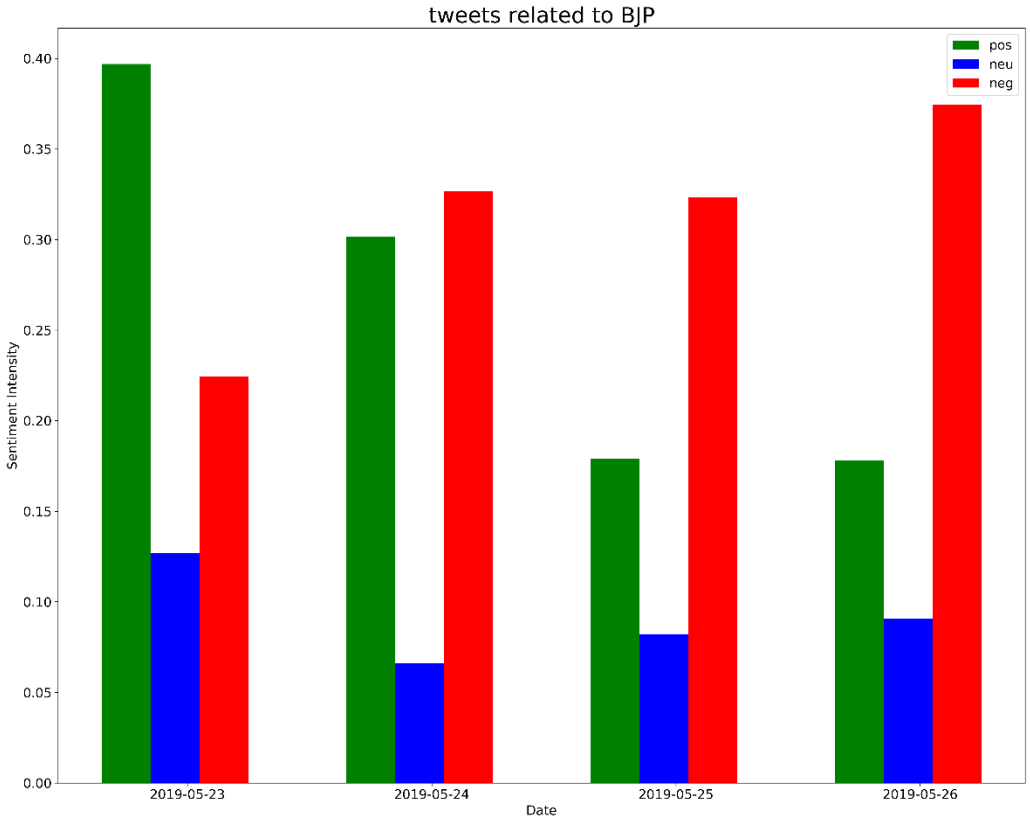


Fig. C.2

BJP related Tweets:

BJP got most positive tweets on 23rd May, after that amounts of positive tweets decreased on 24th and 25th May, after that it saturated. This is obvious from the fact that most of the BJP's positive tweets were of congratulatory nature and such tweets subsided once counting got over.

Negative tweets on the other hand were least on 23rd, increased on 24th, stayed similar on 25th and became highest on 26th

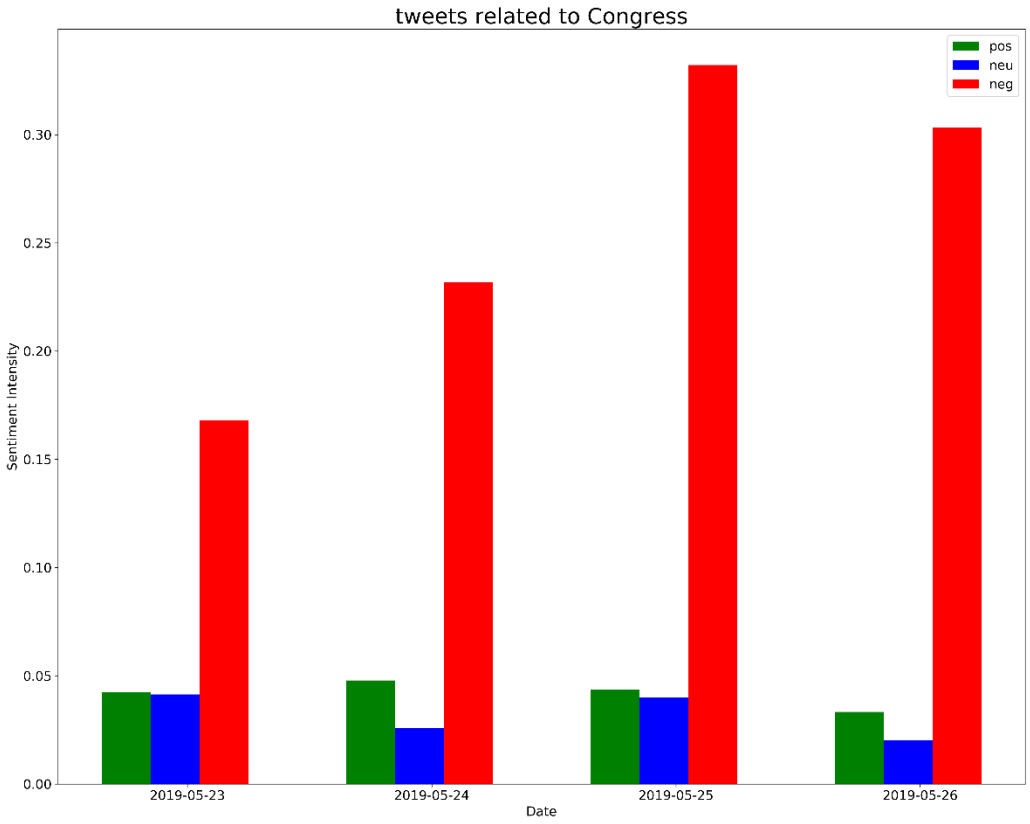


Fig. C.3

Congress related Tweets:

Congress got very small amount of positive tweets and this amount stayed constant across 4 days.

Negative tweets related to Congress were least on 23rd May but kept increasing on 24th and 25th May. On 26th May this amount dropped a little.

Geo-Tagged Sentiments

In maps below, darkness of colors indicate the intensity of given sentiment i.e. darker the color's shade, more intense is the sentiment.

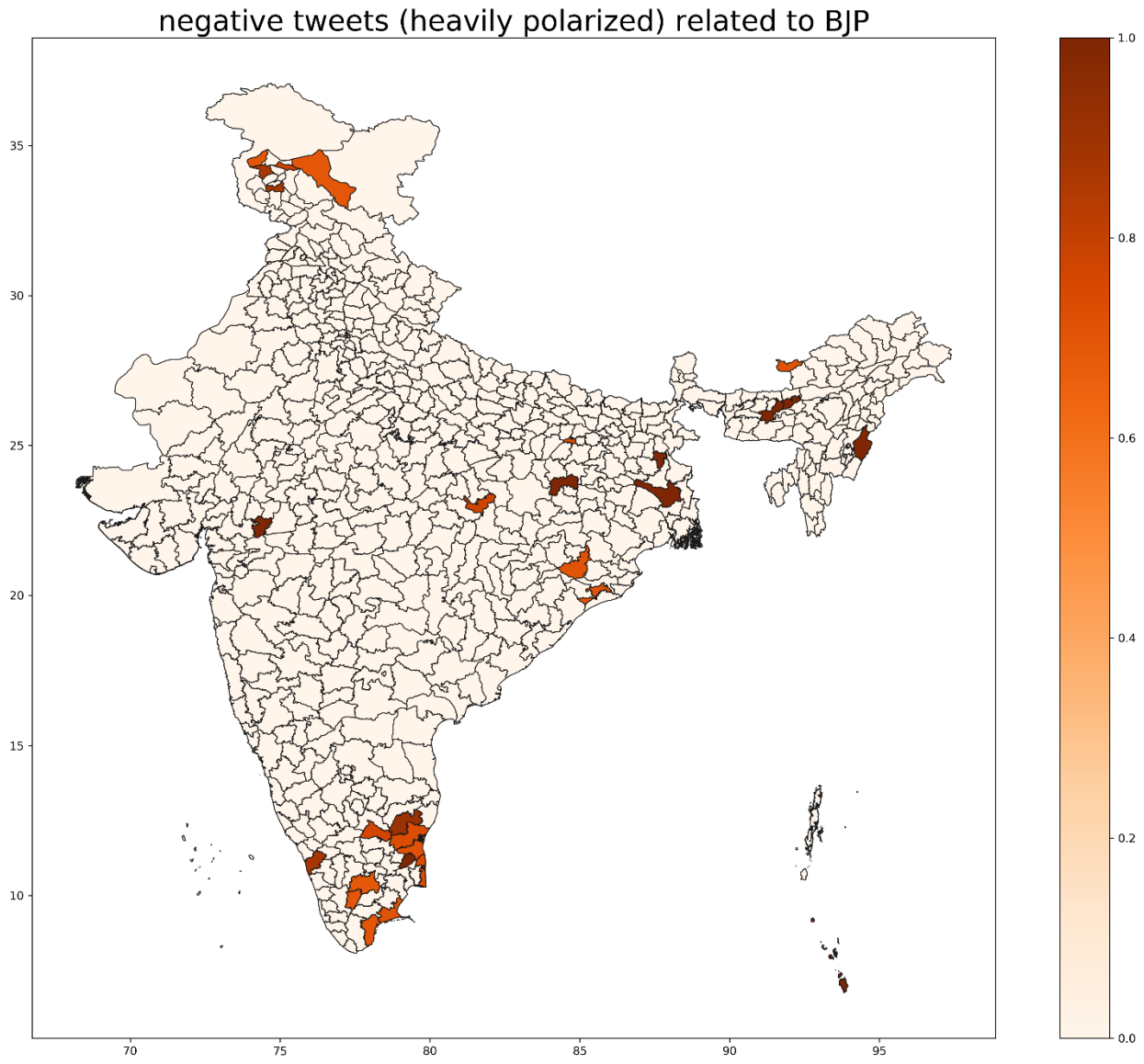
Intensity of sentiment in turn is defined as (in context of district level map)

a) *If plotting without subject*:
Sentiment Intensity == (Total number of tweets belonging to given sentiment at given district) / (Total no of tweets at given district)

b) *If plotting with a subject:*
Sentiment Intensity == (Total number of tweets belonging to given sentiment in given partial_set at given district) / (Total no of tweets in union_set on at given district)

where, partial_set may mean "set of all BJP related tweets" or "set of all Congress related tweets"
while, union_set is union of "set of all BJP related tweets" and "set of all Congress related tweets".

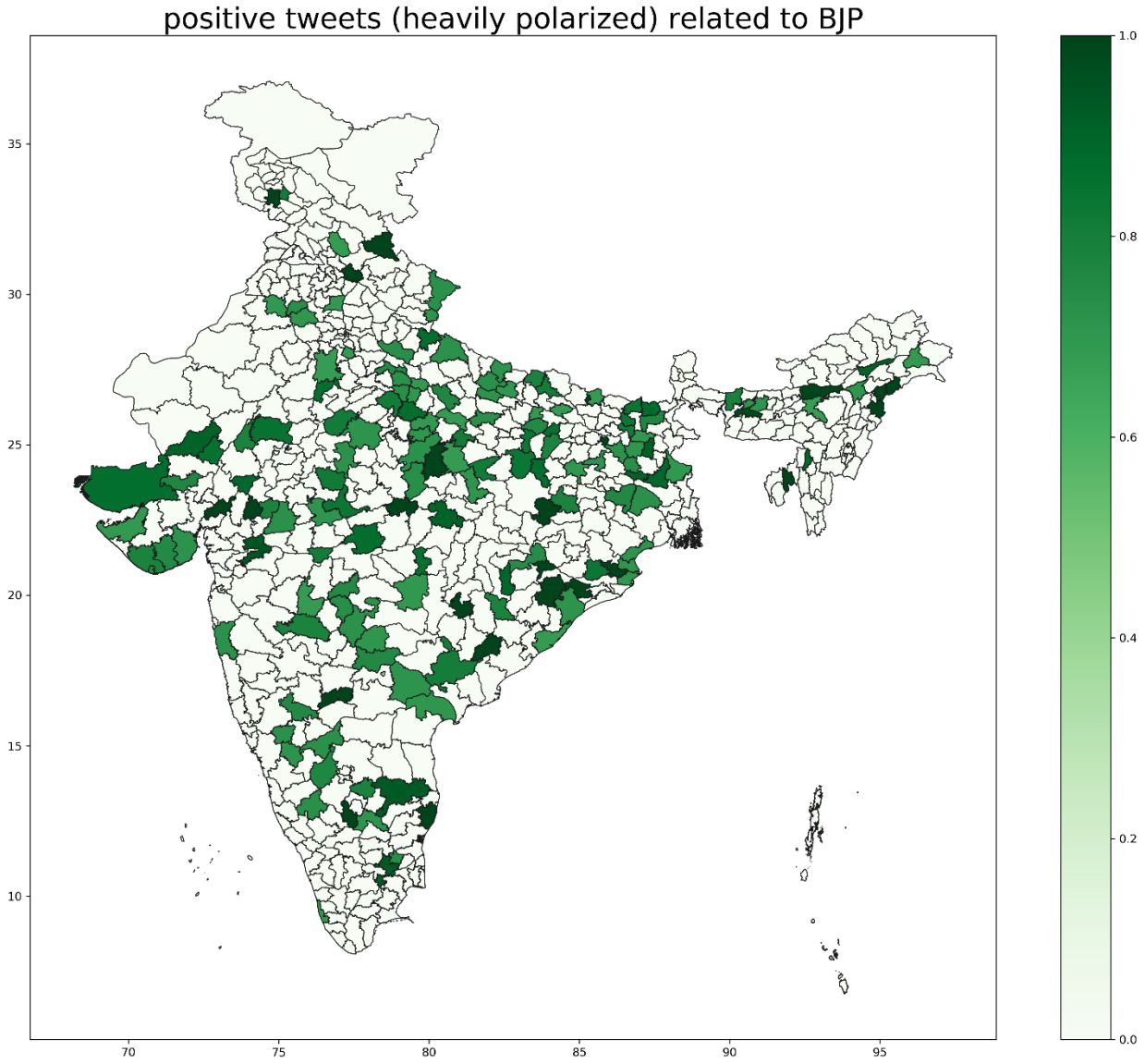
Note: *Unlike previous plots, this plot is composed of only heavily polarized sentiments (having sentiment intensity >0.5)*



D.1.1

<u>BJP related Negative tweets:</u>

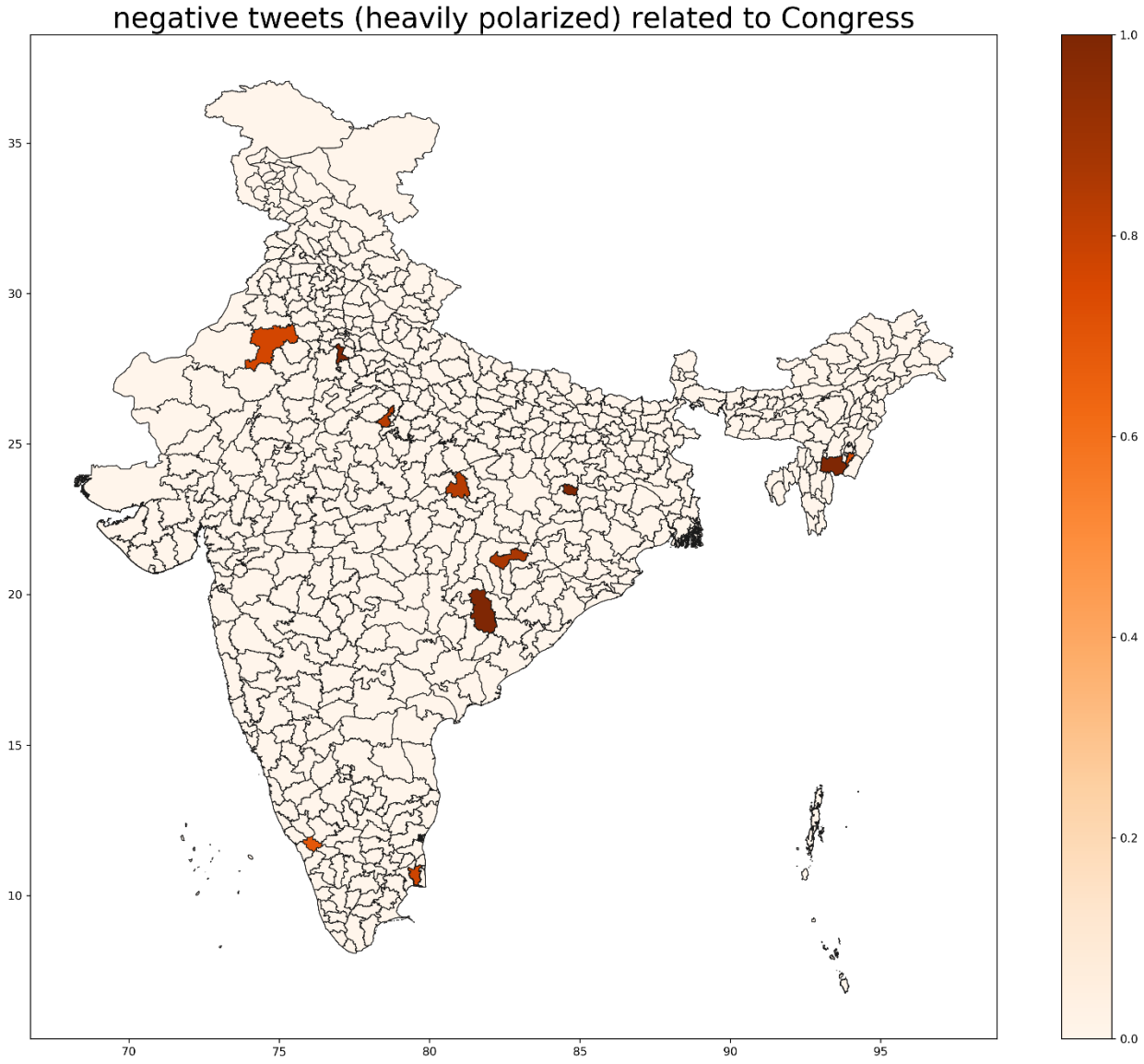
BJP got a lot of negative tweets from a lot of Tamil Nadu districts
(which is consistent with the fact that BJP performed poorly in Tamil Nadu)
Other than that ,districts like Bardhaman (West Bengal), Lateher (Jharkhand),Pakur (Jharkhand), Alirajpur (Madhya Pradesh), Kargil (Jammu and Kashmir), Nicobar (Andaman And Nicobar) and Ukhrul (Manipur) contributed a lot negative tweets towards BJP



D.1.2

<u>BJP related Positive tweets:</u>

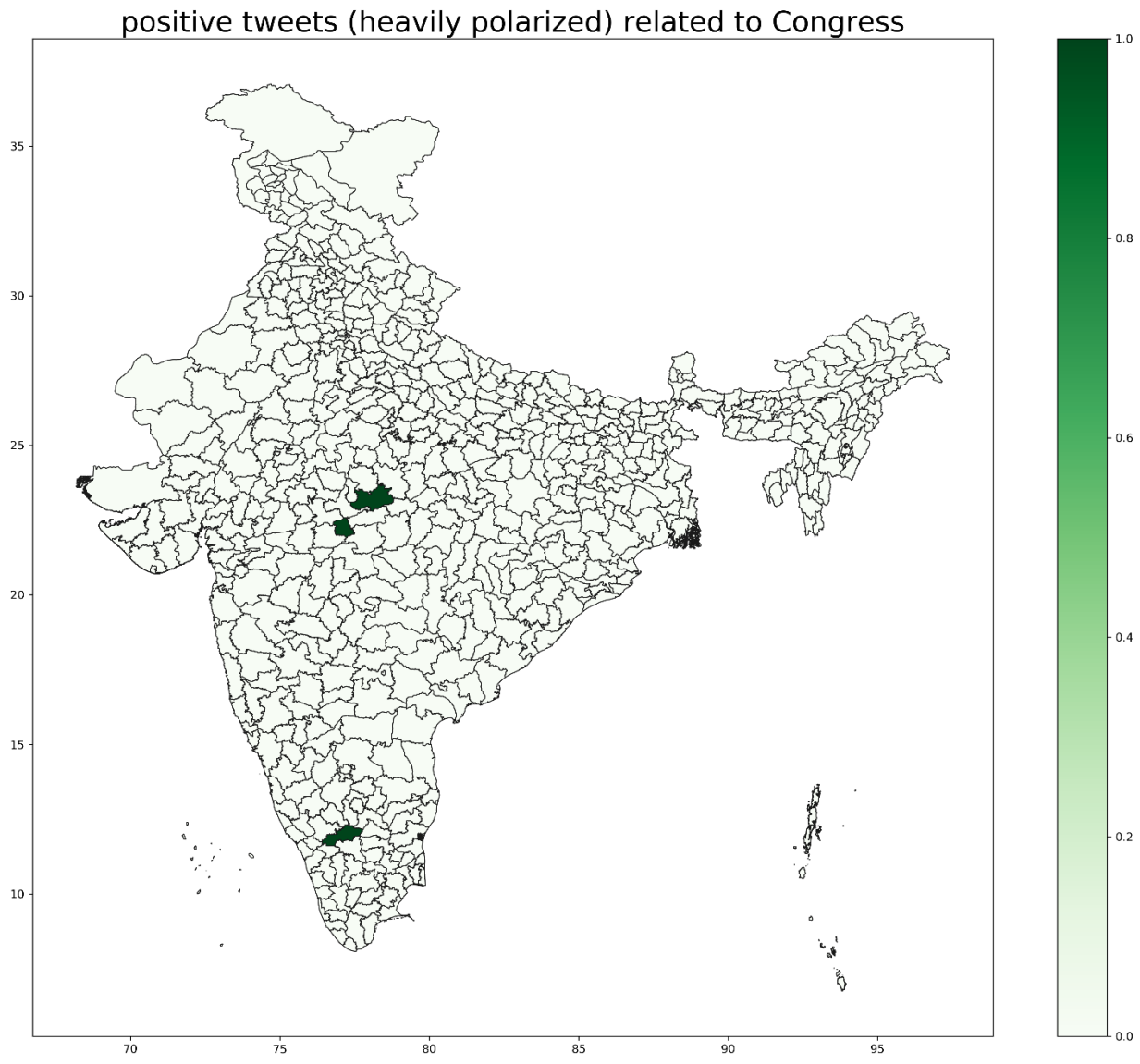
Districts like Kaach (Gujrat), Chittor (Andhra Pradesh),Kanchipuram (Tamil Nadu) , Tiruchirappalli (Tamil Nadu), Tirap (Arunachal Pradesh), Ambassa (Tripura), Kinnaur (Himachal Pradesh),Nayagarh (Orrisa), Phulbani (Orrisa) contributed a lot of positive tweets towards BJP



D.2.1

Congresss related Negative tweets:

Districts like Churu (Rajasthan), Uttarkashi (Uttarakhand), Churachandpur (Manipur), Tiruvarur (Tamil Nadu), Bageshwar (Uttarakhand) , Bastar (Chhattisgarh) contributed a lot of negative tweets towards congress



D.2.2

Congress related Positive tweets:

As already observed, no of positive tweets for congress are less and thus its positive tweets map is not that heavely shaded.

Still, districts like Raisen(Madhya Pradesh), Harda(Madhya Pradesh), Chamarajnagar(Karnataka) contributed a lot of positive tweets towards Congress

Challenges and scope for improvements

- Model can't distinguish between statement and its negation like "I love it" and "I don't love it"

  Possible Solution: Providing more data having such statements

- All tweets in regional languages were isolated. There were 1,98,261 such tweets. (These tweets were not thrown away but stored seperately for any future improvements)

  Possible Solution: Google's Translation API (with its improved Neural Network based translation) can be used to translate such tweets instead of ignoring them. This was infact considered during the project but was later discarded because of high api charges.

- As can be observed from Learning Curve (Fig.2), accuracy may be improved further by providing more labelled data for training.

- Deep Learning Algorithms like LSTM (Long Short Term Memory) can also be tried to improve accuracy.

Bibliography

- "*Distributed Representations of Words and Phrases and their Compositionality", Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean*
  https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

- *"Language detection library for Java", Nakatani Shuyu*
  https://www.slideshare.net/shuyo/language-detection-library-for-java
- *John Gruber's prebuilt URL matching regex*
  https://daringfireball.net/2010/07/improved_regex_for_matching_urls
- *"GeoLite2 Free Downloadable Databases"*
  https://dev.maxmind.com/geoip/geoip2/geolite2/
- *"Datameet/Maps"(India District wise shapefile based on Census 2011)*
  https://github.com/datameet/maps/tree/master/Districts