Sentiment Analysis on Indian General Elections 2019

Table of Contents

1. Introduction

Sentiment analysis is a category of natural language processing which is used to extricate, recognize, or portray opinions from different content structures, including news, audits, social media and articles and categorizes them as positive, negative or [1]. In this project we are going to analyze the general user tweets from the election point of view. Here we will study the user view of Indian election. Based on the users tweets the system analysis if there exists a pattern between the tweets and to analyse and draw meaningful inferences from the collection of these tweets collected over certain period. The proposed system finds a classification model to identify the political orientation of the twitter users based on the tweet content and other user based features like location of the tweet [2].

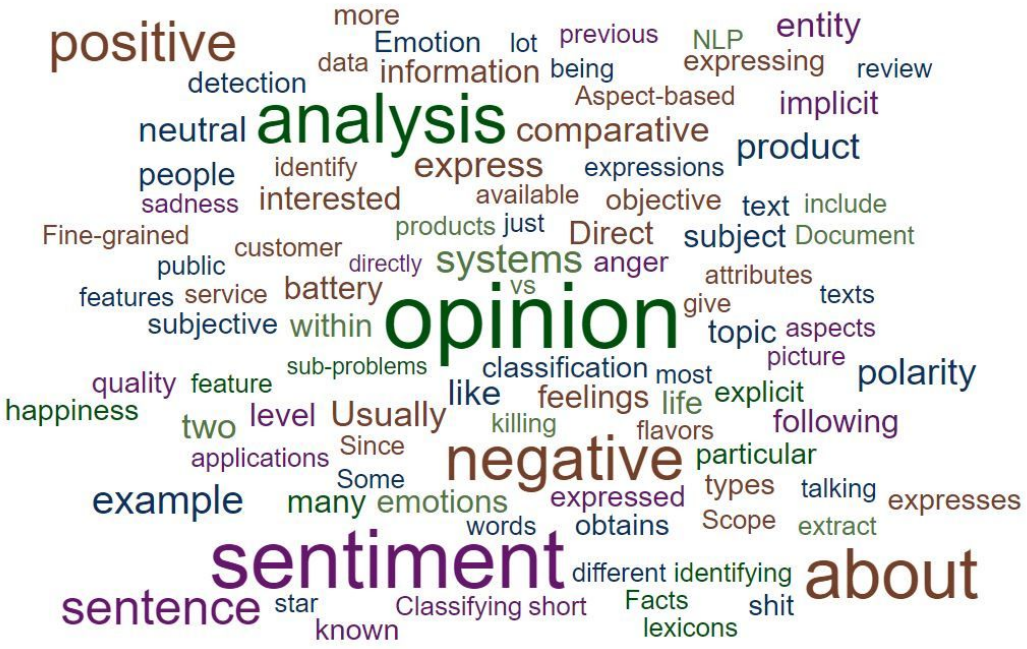

Fig 1. Word Cloud generated from paper [P. Sharma et al.]

2. Project Description

The purpose of this project is as follows:

To analyze the user sentiment on twitter for the political parties participating in Indian General Elections 2019 and help understand how it varies on location and to understand which party leads based on positive sentiment and compare them with opinion polls to come up with our prediction.

The description of the project is as follows:

1. Crawl twitter to find relevant tweets and clean the tweet to have only required information before saving it into a data store.
2. Various strategies would be discussed regarding to political leaning like user location, twitter specific features, user sentiment for the appropriate prediction of the elections.
3. Views related to several features as comparison using user's posting behaviour, linguistic content, reply would also be considered.
4. Finally, we would predict the results of Indian general elections based all the above analysis and plot a well granularized graph showing our inference

3. Method Overview

3.1 Extracting data from twitter

The first step is to gather the data from Twitter. By giving Twitter a list of keywords we can receive a live stream of all tweets that match those keywords. Before our system can analyze the tweet, it needs to be modified so that we can focus in on the most important features of the tweet. The words are stemmed, punctuation is removed, and we only keep adverbs, adjectives, and verbs because they have been most valuable in determining sentiment.

3.2 Performing Sentiment Analysis

Now that the tweet is ready, it is passed on to a series of machine learning classification algorithms that will each analyze and tell us whether it believes that tweet is speaking positively or negatively. Due to the complexity of spoken languages, especially with words having multiple meanings, these classifiers are not perfectly accurate. In order to improve our results, we use several of them, and each one votes on what it thinks the correct answer is. We then take whichever answer is most popular as the correct answer. That result is then stored for later use.

4. Sentiment Analysis after crawling tweets

4.1 Using the sentiment Corpora

We have used the words polarity Corpora from Stanford SNAP which already has pre-classified words to process them before feeding into the classifiers. For this we first tokenizer the words find the frequency distribution.
word_features, which contains the top 3,000 most common words. Next, we're going to build a quick function that will find these top 3,000 words in our positive and negative documents, marking their presence as either positive or negative.
fairly popular text classification task is to identify a body of text as either spam or not spam, for things like email filters. In our case, we're going to try to train a sentiment analysis model.
To do this, we're going to start by using the NLTK corpus. First, let's wrangle our data. Basically, in plain English, the above code is translated to: In each category (we have pos or neg), take all of the file IDs, then store the word_tokenized version (a list of words) for the file ID, followed by the positive or negative label in one big list.

Next, we use random to shuffle our documents. This is because we're going to be training and testing. If we left them in order, chances are we'd train on all of the negatives, some positives, and then test only against positives. We don't want that, so we shuffle the data.

4.2 Training Naive Bayes classifier

Now it is time to choose an algorithm, separate our data into training and testing sets, and press go! The algorithm that we're going to use first is the Naive Bayes classifier. This is a pretty popular algorithm used in text classification, so it is only fitting that we try it out first. Before we can train and test our algorithm, however, we need to go ahead and split up the data into a training set and a testing set.

You could train and test on the same dataset, but this would present you with some serious bias issues, so you should never train and test against the exact same data. To do this, since we've shuffled our data set, we'll assign the first 1,900 shuffled words, consisting of both positive and negative words, as the training set. Then, we can test against the last 100 to see how accurate we are.

Code output:

```
Number of features selected:  10664
Naive Bayes Classifier Accuracy :  72.59036144578313
Most Informative Features
                   flat = True              neg : pos    =     22.2 : 1.0
              engrossing = True              pos : neg    =     20.4 : 1.0
                  boring = True              neg : pos    =     19.3 : 1.0
                 generic = True              neg : pos    =     16.9 : 1.0
                mediocre = True              neg : pos    =     16.2 : 1.0
               inventive = True              pos : neg    =     15.1 : 1.0
              refreshing = True              pos : neg    =     14.4 : 1.0
                 routine = True              neg : pos    =     13.6 : 1.0
                    warm = True              pos : neg    =     12.7 : 1.0
                powerful = True              pos : neg    =     12.5 : 1.0
               wonderful = True              pos : neg    =     12.3 : 1.0
                mindless = True              neg : pos    =     11.6 : 1.0
               realistic = True              pos : neg    =     10.4 : 1.0
             mesmerizing = True              pos : neg    =     10.4 : 1.0
                    dull = True              neg : pos    =     10.3 : 1.0
```

4.3 Pickling data for speed

Python pickle module is used for serializing and de-serializing a Python object structure. Any object in Python can be pickled so that it can be saved on disk. What pickle does is that it “serializes” the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script. This saves us time and helps us not run the training models again and again[6].

So after we have trained our model then, we use pickle.dump() to dump the data. The first parameter to pickle.dump() is what are you dumping, the second parameter is where are you dumping it.
After that, we close the file as we're supposed to, and that is that, we now have a pickled, or serialized, object saved in our script's directory.

4.4 Training Additional Classification models

Now to make sure we get the best accuracy percentage we have decided to run a few more models and create a Invoking the fit method on the VotingClassifier which will fit clones of those original estimators that will be stored in a class attribute[7].n estimator can be set to None using set_params.

The output which we see from running our multiple models is below:
Original Naive Bayes Algo accuracy percent: 73.49397590361446
MNB_classifier accuracy percent: 72.59036144578313
BernoulliNB_classifier accuracy percent: 73.94578313253012
LogisticRegression_classifier accuracy percent: 71.23493975903614
LinearSVC_classifier accuracy percent: 69.27710843373494
SGDClassifier accuracy percent: 70.33132530120481

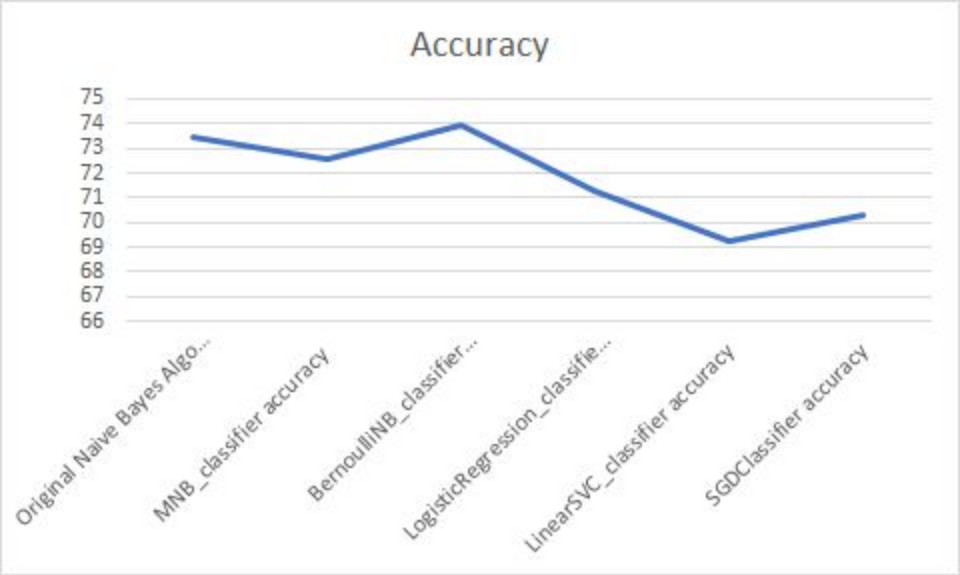

Fig 2. Accuracy of ML algorithms over sentiment dataset

4.5 Getting tweets

We get tweets from twitter using twitterstream function of tweepy to gather tweets and then run our pre trained models on those tweets to process them and assign negative and positive sentiment values along with confidence on them. This data we write into a MySQL database.

Code Output:

```
@Babu_Bhaiyaa We missed you @sardesairajdeep at Benaras today, come and cover one rally of Modi in person neg 1.0
Found keyword:  Modi  belongs to party:  Bharatiya Janata Party
It appears in TV debates that you are supporter of terrorism neg 1.0
@narendramodi Mein bhi ek chowkidar..I m proud of Indian chowkidar pos 0.8
This guy has lost his mental balance. neg 1.0
@JhaSanjay It's fear of Rahul Gandhi.
Modi is so scared of the Congress and it's superb strategic planning for RaGa. pos 1.0
Found keyword:  Gandhi  belongs to party:  Indian National Congress
```

4.6 MySQL db setup

We have chosen to dump the data into a database instead of a CSV because it gives us much more control over the data, since we can run queries to find how aligned the data was coming according to our understanding. The table looks like below image.

Info | Columns | Indexes | Triggers | Foreign keys | Partitions | Grants | DDL

Column	Type	Default Value	Nullable	Character Set	Collation	Privileges	Extra
id_t1	bigint(20)		NO			select,insert,update,references	auto_increment
tweetID	varchar(200)		NO	utf8mb4	utf8mb4_unicod...	select,insert,update,references	
party_name	varchar(200)		NO	utf8mb4	utf8mb4_unicod...	select,insert,update,references	
dateTime	datetime(6)		YES			select,insert,update,references	
tweet	varchar(200)		NO	utf8mb4	utf8mb4_unicod...	select,insert,update,references	
screen_name	varchar(200)		NO	utf8	utf8_general_ci	select,insert,update,references	
source	varchar(200)		NO	utf8mb4	utf8mb4_unicod...	select,insert,update,references	
country	varchar(200)		YES	utf8mb4	utf8mb4_unicod...	select,insert,update,references	
country_code	varchar(200)		YES	utf8mb4	utf8mb4_unicod...	select,insert,update,references	
full_name	varchar(200)		YES	utf8mb4	utf8mb4_unicod...	select,insert,update,references	
name	varchar(200)		YES	utf8mb4	utf8mb4_unicod...	select,insert,update,references	
place_type	varchar(200)		YES	utf8mb4	utf8mb4_unicod...	select,insert,update,references	
quote_count	varchar(200)		YES	utf8mb4	utf8mb4_unicod...	select,insert,update,references	
reply_count	int(11)		YES			select,insert,update,references	
retweet_count	int(11)		YES			select,insert,update,references	
favorite_count	int(11)		YES			select,insert,update,references	
sentiment	varchar(6)		YES	utf8	utf8_general_ci	select,insert,update,references	
confidence	float		YES			select,insert,update,references	
num_sentiment	int(11)		YES			select,insert,update,references	
followers_count	int(11)		YES			select,insert,update,references	
verified_bit	bit(1)		YES			select,insert,update,references	
friends_count	int(11)		YES			select,insert,update,references	

Fig 3. Party_sentiment table containing oru data extract from the twitter crawl

The data that we are using are:

Field	Explanation
tweetID	Numerical ID of the tweet.
party_name	Name of the party we are classifying the tweet as based on our set of keywords. Can be “Bharatiya Janata Party or “Indian National Congress”.
dateTime	Timestamp when the tweet has been pulled.
tweet	The text body of the tweet.
screen_name	Screen name of the tweeter.
source	The type of device the tweet has been generated from.
country	Country of origin of tweet.
country_code	Numerical ID of the country of origin.
full_name	Location string of the tweet, containing city and country both.
name	City name of the tweet origin.
place_type	Can be city or country.
quote_count	Count of tweet being quoted.
reply_count	Count of tweet being replied to.
retweet_count	Count of tweet being retweeted.
favorite_count	Count of tweet being favorited.
sentiment	Sentiment value being output from our classifier. Can be pos or neg.
confidence	Confidence of our classification. Can hold values in 1.0, 0.8 and 0.6.
num_sentiment	Numerical conversion of our sentiment. 1 is positive and 0 is negative.
followers_count	Count of followers of the tweeter.
friends_count	Count of friends of the tweeter.

verified_bit	Shows if tweeter is verified or not

4.7 TextBlob sentiment analysis

To get a more comprehensive sentiment score we are also using TextBlob sentiment analysis API of twitter. This categorizes the tweets into positive, negative and neutral based on polarity and subjectivity scores. We use both the model sentiment score and TextBlob score to get the most accurate result.

4.8 Code Summary

Below we look at a short summarized version of the pseudo code which is explained in the above sections.

Part 1 involves training our models on the sentiment keyword data after NTLK processing.

```
#0. datasets to train models based on classified words
short_pos = open("short_reviews/positive.txt", "r").read()
short_neg = open("short_reviews/negative.txt", "r").read()

#1. we select only adjectives for our analysis,
allowed_word_types = ["J"]

#2.  we create a document merging both words with labels
for p in short_pos.split('\n'):
    documents.append((p, "pos"))
    part_of_speech = pos_tag(word_tokenize(p))
    for w in part_of_speech:
        if w[1][0] in allowed_word_types:
            all_words.append(w[0].lower())

#3. tokenize the words and create our feature set of words
def find_features(document):
    words = word_tokenize(document)
    features = {}
    for w in word_features:
        features[w] = (w in words)
    return features

#4. create our features set from documents dictionary
featuresets = [(find_features(rev), category) \
               for (rev, category) in documents]

#5. split the data
train_data = featuresets[:10000]
test_data = featuresets[10000:]

#6. pass data through our classifier
NB_classifier = NaiveBayesClassifier.train(train_data)

#7. save the trained model
save_classifier = open("pickles/Naive_Bayes.pickle", "wb")
pickle.dump(NB_classifier, save_classifier)
save_classifier.close()
```

Part 2 involves crawling twitter for data using tweepy and applying our model on tweets to perform sentiment analysis and gather other features for further processing.

```
#8. load model
file = open("pickles/Naive_Bayes.pickle", "rb")
NB_classifier = pickle.load(file)
file.close()

#9. Setting up connection to database
conn = MySQLdb.connect(host,user,passwd,db)

#10. Insert query for the data
add_tweet = ("INSERT INTO party_sentiment (tweet")

#11. Cleaning tweets to remove "@" in tweets, remove punctuations
def clean_tweet(tweet):
    return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|\
                (\w+:\/\/\S+)", " ", tweet).split())

#12. Streaming function to pass through tweepy
class twitter_streaming(StreamListener):

    def on_data(self, data):
        all_data = json.loads(HTMLParser().unescape(data))
        tweet = all_data['text']
#13. Clean tweets and remove "RT" and links, pass throught analyzer
        c_tweet = clean_tweet(tweet)
            if not all_data['retweeted'] and not \
            tweet.startswith('RT') and 't.co' not in tweet:
                sentiment_value, confidence = sentiment(c_tweet)

#14. Call function using tweepy Stream
  twitterStream = Stream(auth, twitter_streaming())

```

The full comprehensive code has been attached to the zip.

4.9 Kibana dashboard

Because of the massive data we had extracted, SQL was proving to be an ineffective tool to perform our analysis. We looked for alternatives and finally we found an new stack. We used Elasticsearch for our database dump and used a powerful visualization tool called Kibana to perform analysis and generate the plots we shall see below. All of the below visualizations have been generated on a well organized dashboard. The code to migrate the SQL database to Elasticsearch has been attached in the zip.

6. Analysis on data extracted

6.1 “Hinglish” and native language tweets

Since majority of our tweets are concerned with India and we lack proper datasets for processing native languages we only take tweets marked as english from twitter. Another case we came across is the use of “Hinglish” confuses our sentiment analyzer and tags a few tweets incorrectly.

6.2 Bias based on “Congress” keyword

We see that use of word “congress” for “Indian National Congress” was very vital but this also pulled in tweets of US Congress. So we had tweets involving the american congress and on the muller report because the report was published in this timeframe. Running the below query we were able to select values and later drop values containing the US congress tweets.

```
SELECT
   count(party_name)
FROM
   smdm.party_sentiment
WHERE
   party_name = "Indian National Congress"
   AND
   (
      tweet LIKE '%trump%'
      OR tweet LIKE '%donald%'
      OR tweet LIKE '%u.s.%'
      OR tweet LIKE '%potus%'
      OR tweet LIKE '%repub%'
      OR tweet LIKE '%democrat%'
      OR tweet LIKE '%nancy%'
      OR tweet LIKE '%muller%'

   );
```

6.3 Identification of bot accounts from source

We were able to find the report of [8] finding that bots were used during the elections to boost tweet count. One way we used to tackle this was by parsing the tweet and removing all instance of "RT" to remove the retweets. This has been done using below line.

```
if not all_data['retweeted'] and not tweet.startswith('RT') and
't.co' not in tweet
```

Another method we have used to handle this was that the source field had shown us fields with the name of the bot sites in them and these were dropped from the database.

```
SELECT
   count(party_name)
FROM
   smdm.party_sentiment
WHERE
      source LIKE '%bot%'
 );
```

6.4 Significant accounts with 1000 followers

To find the most significant impacts on our search results we have only considered accounts with follower_count > 1000.

6. Results

All charts have been created after crawling twitter for three weeks of data just before the elections had started and till the election third phase was completed.

6.1 Word cloud of all tweets

Our words clouds show what are the most recurring words out of all tweet for the parties. We see how the most important people in these parties are Rahul Gandhi and Narendra Modi.

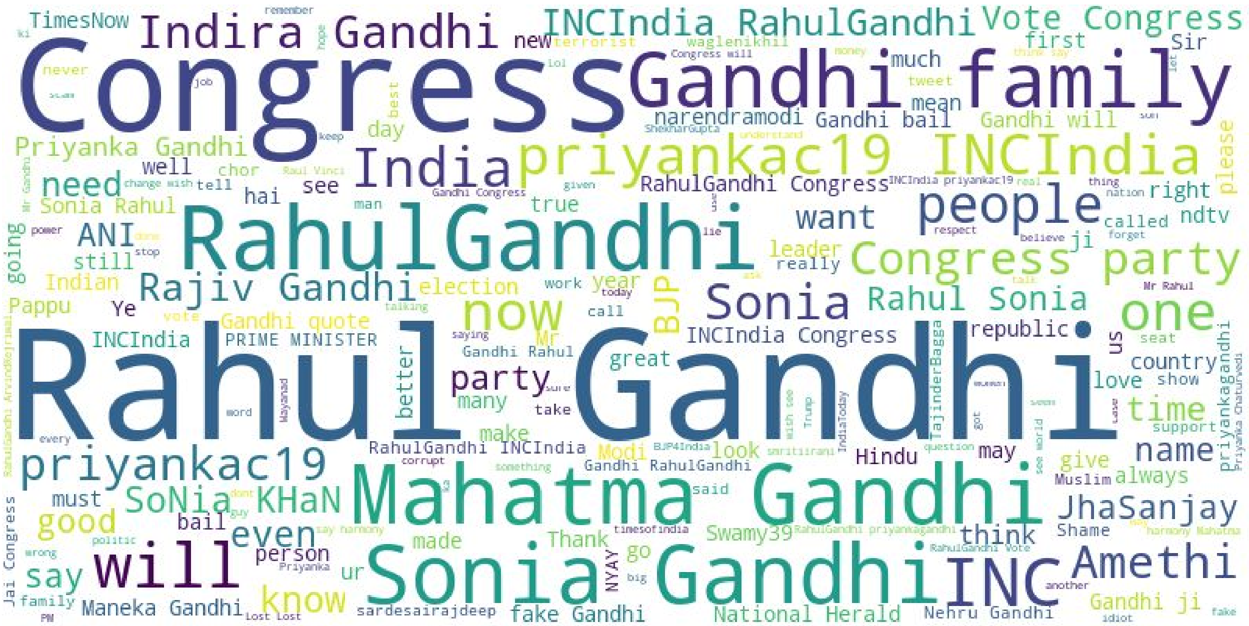

Fig 4. Wordcloud generated from Indian National Congress tweets

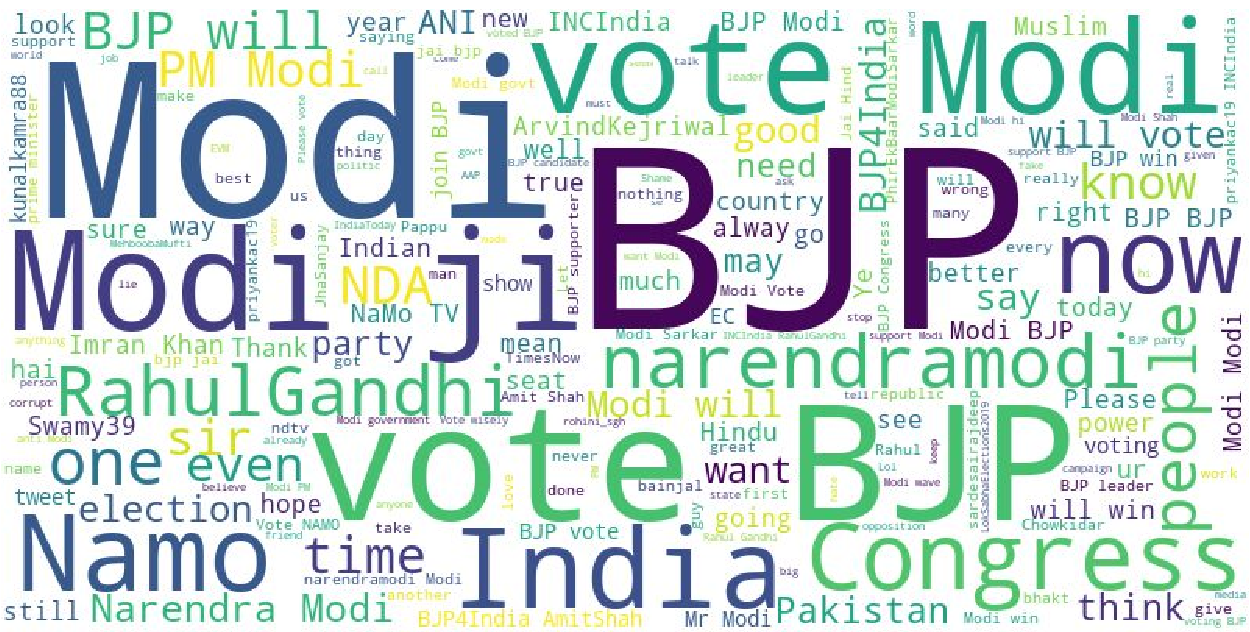

Fig 5. Wordcloud generated from Bharatiya Janata Party tweets

6.2 Total tweet count metrics

We perform a simple analysis on the number of tweets categorized for each party and we see that BJP has a much higher count than INC. Out of the total 105,257 tweets, BPJ owns 69,385 tweets and INC owns 35,872 tweets. We see that BJP has overwhelming popularity over INC.

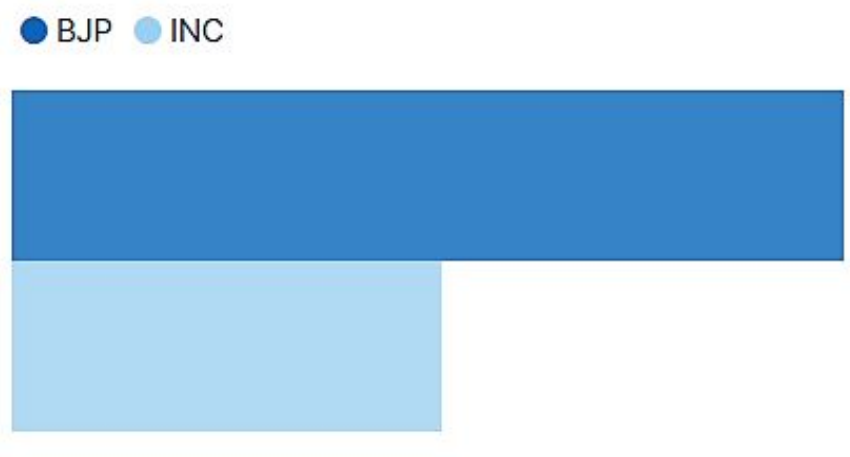

Fig 6. Total tweet count by parties

6.3 Sentiment trends based on parties

From the legend we see that violet tagged tweets are neutral, green are positive and red are negative tweets. We see how BJP has overall more positive sentiment than INC. We also see that the number of tweets were higher just before and at the beginning of the elections and as the election progresses midway the number of tweets keep getting reduced.

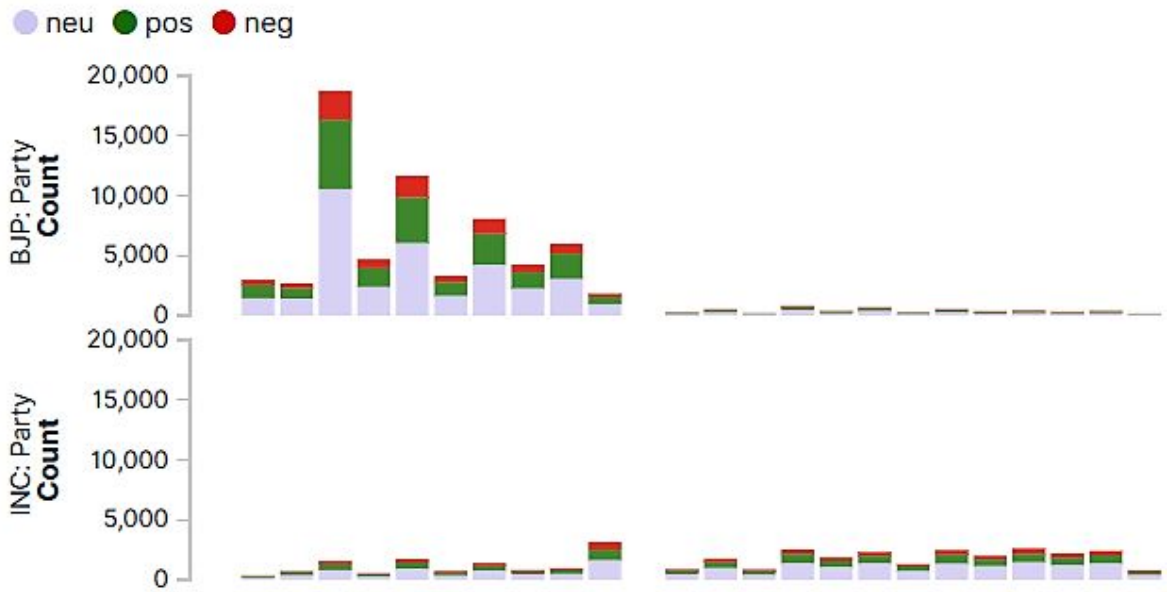

Fig 7. Sentiment trends by parties

6.3 Sentiment overview based on parties

We see here that the center of the circle shows the type of sentiment and the colored bars surrounding the sentiment shows the party share. We see how BJP has the major chunk of the positive tweets. We also see that because of "Hinglish" being used a lot of the tweets were classified as neutral. This is a problem we have faced because we were working on Indian elections.

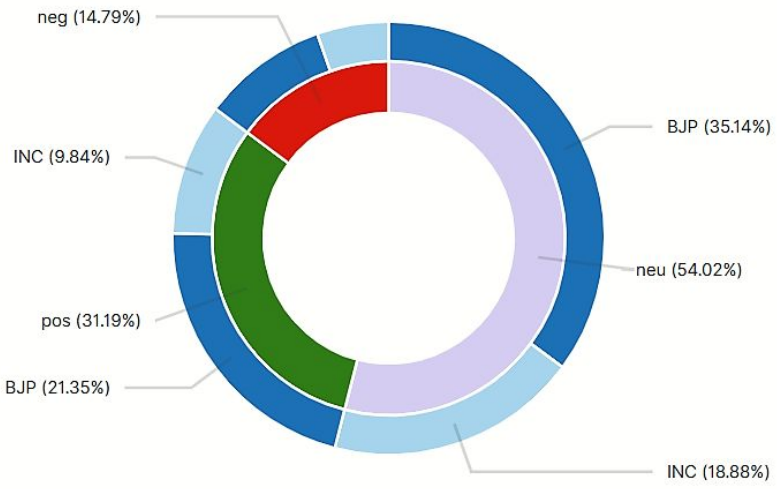

Fig 8. Sentiment Overview of parties

6.4 Global Location metrics based on parties

We see that the majority of the tweets originated from India. But we also see a significant number of tweets from around the world. This shows that Indian elections has global interest.

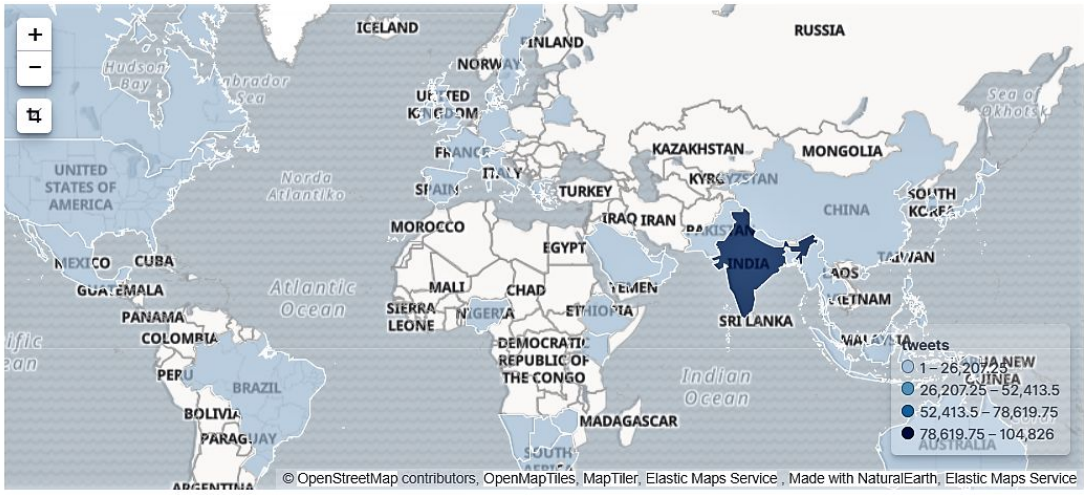

Fig 9. Location of origin of tweets

6.5 Indian state Location metrics based on parties

The top image shows BJP's distribution of tweets and bottom shows INC's. We see a high volume of tweets originating from the North East side of India. This aligns with the exit poll data from major media outlets. We also see the volume of tweets in the north of India being higher for BJP than for INC while in south of India a uniform division between both the parties is evident.

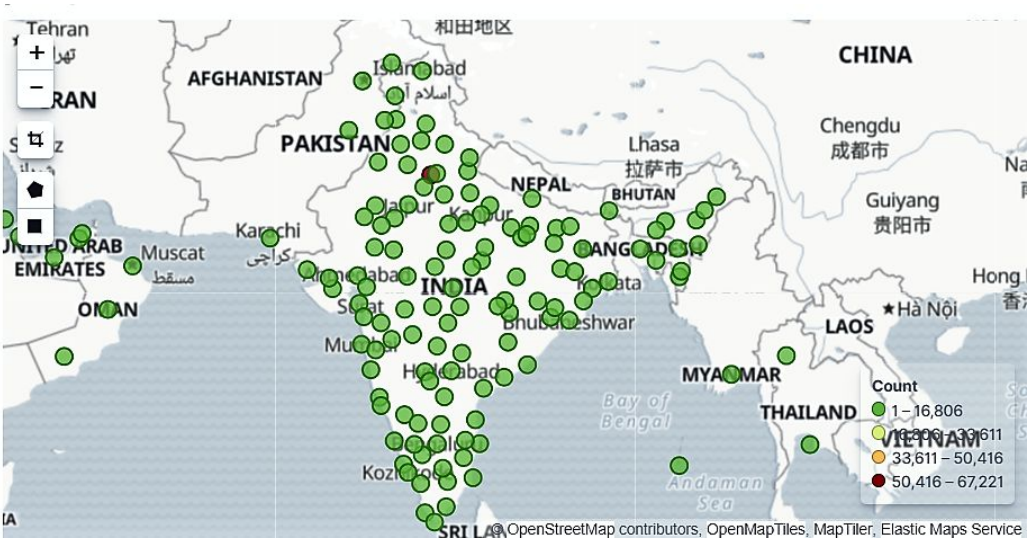

Fig 10. BJP tweet distribution

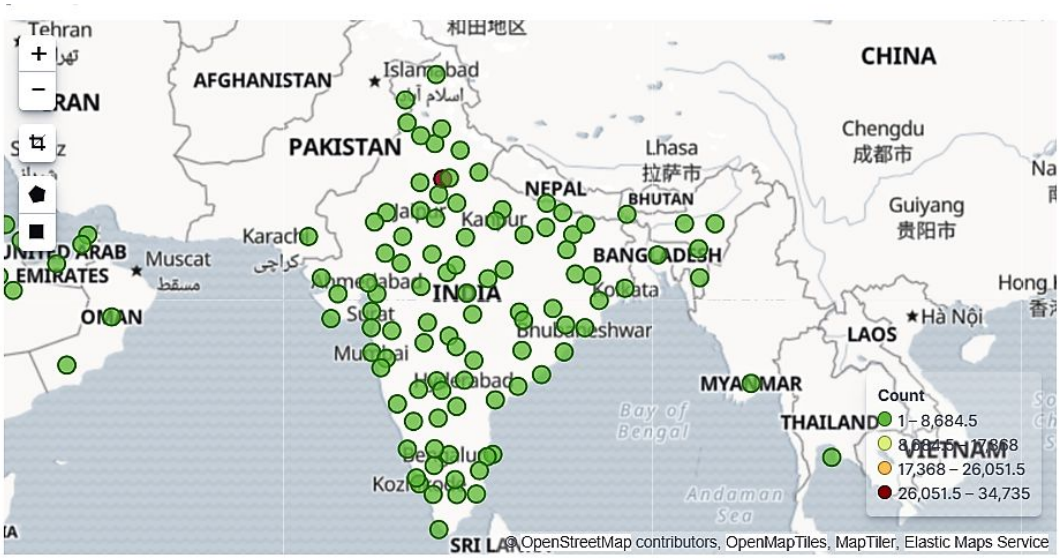

Fig 11. INC tweet distribution

6.6 Tweet Source metrics based on parties

One trend that we discover is that most of the tweets originate from mobile devices especially from Android phones. After that we see Twitter Web App and Web Client having a major chunk. We found the total number of client devices to be 236 from our crawled tweets showing a wide diversity. We also see a lot of bots being used for biasing

the tweet count something that was stated on “The Wire” media house. This is a similar trend we have seen in other elections across the world.

Fig 12. Wordcloud of Tweet sources

6.7 Tweet Source metrics based on parties

From the below image we that the most followed twitter accounts being the “influencers” belong to news outlets and also BJP4India and Swamy39 are the official BJP twitter handle and the handler of a BJP party member. We see that BJP has more social clout than INC does.

Twitter User	Followers
timesofindia	11,735,238
BJP4India	10,890,589
abpnewstv	8,250,776
Swamy39	7,833,540
the_hindu	5,254,305
IndiaToday	5,132,696
SkyNews	4,957,307
virsanghvi	4,353,958
ZeeNews	4,349,107
vikramchandra	3,002,285

Fig 13. Most followed accounts from tweets crawled

7. Conclusion

Our conclusion is Bharatiya Janata Party is going to have the majority in the election based on sentiment metrics and tweet counts generated. We also see that BJP has much social impact than INC, results which match with the previous general election results.

8. Future Work

From our project we come across as few tasks which can be performed to improve the accuracy of our prediction. Firstly we can create a dictionary of words containing translations for most recurring hinglish words. This would help us utilize the sentiment even more from the tweets that we extract reducing the neutral tweet count. Secondly, we can use existing dictionaries for local dialect translation which would let us use other regional language tweets like Hindi, Tamil, Telugu etc. This would make our analysis more comprehensive to the region.

9. Libraries And Resources Used

- This project was written in Python using python Natural Language Toolkit.
- Machine learning algorithms have been used from scikit-learn for Python.
- TextBlob has been used for additional sentiment research.
- Tweepy Twitter API library used for crawling tweets.
- GeoPy library was used for location information extraction.
- Python Programming Tutorials for NLTK for natural language processing were used for reference.
- Kibana has been used to design our dashboard and graphs.

10. References

[1] P. Sharma and T.-S. Moh, “Prediction of Indian election using sentiment analysis on Hindi Twitter,” 2016 IEEE International Conference on Big Data (Big Data), 2016.

[2] G. P.wani and N. V. Alone, “Analysis of Indian Election using Twitter,” International Journal of Computer Applications, vol. 121, no. 22, pp. 37–41, 2015.

[3] J. Ramteke, S. Shah, D. Godhia, and A. Shaikh, “Election result prediction using Twitter sentiment analysis,” 2016 International Conference on Inventive Computation Technologies (ICICT), 2016.

[4] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan, Thumbs up? Sentiment Classification using Machine Learning Techniques, Proceedings of EMNLP 2002

[5] pythonprogramming.net/pickle-data-analysis-python-pandas-tutorial/

[6] geeksforgeeks.org/understanding-python-pickling-example/

[7] scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier

[8]cnbc.com/2019/02/04/twitter-bots-were-more-active-than-previously-known-during-2018-midterms-study