NLP Assignment 1

Mohammad Aflah Khan (2020082) | Neemesh Yadav (2020529)

I. REGULAR EXPRESSION

A.

a. Average Number of

- Sentences:
  - Label 0 - 1.960500
  - Label 1 - 1.970704
- Tokens
  - Label 0 - 14.801000
  - Label 1 - 14.116747

b. Total Number of Words starting with

- Consonants
  - Label 0 - 17995
  - Label 1 - 19201
- Vowels
  - Label 0 - 6989
  - Label 1 - 7188

c. Unique Tokens

- Before Lowercasing
  - Label 0 - 6987
  - Label 1 - 8617
- After Lowercasing
  - Label 0 - 6210
  - Label 1 - 7610

d. Total Usernames

- Label 0 - 803
- Label 1 - 1305

e. Total URLs

- Label 0 - 60
- Label 1 - 145

f. Tweets for Each Day of the Week

- Label 0
  - Sun 565
  - Mon 391
  - Tue 154
  - Wed 127
  - Thu 171

    - Fri 473
    - Sat 119
- Label 1
    - Sun 763
    - Mon 481
    - Tue 132
    - Wed 172
    - Thu 50
    - Fri 391
    - Sat 298

Methodologies:

a. Sentence Splitting:

1. Split on ! and ? delimiters assuming they denote end of sentence
2. Then Split on . followed by spaces and a capital alphabet
3. The second step can potentially cause loss of information in cases such as Mr. Bean and Mrs. Bean hence for such cases we use a abbreviation matching regex to check if the end of a sentence is a abbreviation and if it is we merge the sentence with the next sentence

Token Splitting:

1. First naively split on whitespaces
2. Iterate over all the tokens and:
    a. If the token has a URL: Add it directly to the list (Method explained in URL Matching Explanation)
    b. Else If the token has a Username: Add it directly to the list (Method explained in Username Matching Explanation)
    c. Else Split on . ! and ? to create new tokens and add them
3. Trim white spaces on all tokens
4. Remove Empty Tokens If any

b. Words Starting with Vowels

1. Words starting with a Vowel followed by either an alphabet, an accented alphabet, an apostrophe or a hyphen.
2. This was done to ensure no cases of say spanish text which creep in twitter data from countries like the U.S. are not missed

Words Starting with Consonants

1. Words starting with a Consonant followed by either an alphabet, an accented alphabet, an apostrophe or a hyphen.
2. This was done to ensure no cases of say spanish text which creep in twitter data from countries like the U.S. are not missed

c. Lowercasing was done using a regex sub inside a for loop where for each iteration a capital alphabet was replaced by it’s lowercase counterpart

d. Counting Usernames

According to Twitter Guidelines

1. Your username cannot be longer than 15 characters. Your name can be longer (50 characters) or shorter than 4 characters, but usernames are kept shorter for the sake of ease.
2. A username can only contain alphanumeric characters (letters A-Z, numbers 0-9) with the exception of underscores, as noted above. Check to make sure your desired username doesn't contain any symbols, dashes, or spaces.
3. Optional Rule to Spot Users* - Usernames containing the words Twitter or Admin cannot be claimed. No account names can contain Twitter or Admin unless they are official Twitter accounts.

Some experimentation with our Twitter Handle helped us reach the following conclusions:

1. @UserName can't be placed with other alphanumeric characters so abc@user 9@user are not valid and won't tag the user
2. X@UserName where X is a punctuation is valid but the following cases also don't allow tagging -
    a. @@xyz
    b. _@xyz

Hence we propose regexes which actually find real tag matches instead of say a user just writing an @ somewhere in the tweet and it getting falsely matched as a tagged user when it really isn't

e. Counting URLs

1. `http`, `https`, `www` common starters for URLs usually
2. Manually investigating all sentences which contain `http` shows no false positives that is all occurrences of `http` correspond to links
3. Manually investigating all sentences which contain `www` shows lots of false positives due to words like `aww`
4. On manual inspection we find no sentences which contain `https`
5. According to Twitter's official Blog, Twitter uses a URL-Shortener which converts links to the form `t.co` however there are no positive matches for this in our dataset. The only matches that arise are spurious matches in links like `blogspot.com`

6. A URL maybe as simple as `www.xyz.abc` and as complex as `http://www.xyz.abc/efg` and can get even more complex by adding / to index more indepth into pages
7. There are false positives that we encounter such as `Gi.don` or `worry.we` but they are both valid URLs as well as there can be custom domains by those names. We assume the domain will be atleast 2 characters long to account for `.me`, `.uk` etc.
8. Numbers have been allowed as only numbers can form valid URLs

f. Counting the number of tweets for each day of the week: Since the date times are already strings we use a simple regex match for the words Mon, Tue, and so on to match and procure the dates

B.

a. A simple regex lookup for the word applied to dataframe column gives us the word count followed by a filter query to get the required counts and sentences of the word
b. We use a similar strategy as compared to a. however we also account for the fact the there might be preceding white spaces and use ^ to match beginning of sentence
c. We use a similar strategy as compared to a. however we also account for the fact the there might be trailing white spaces and use $ to match end of sentence

II. TEXT PREPROCESSING

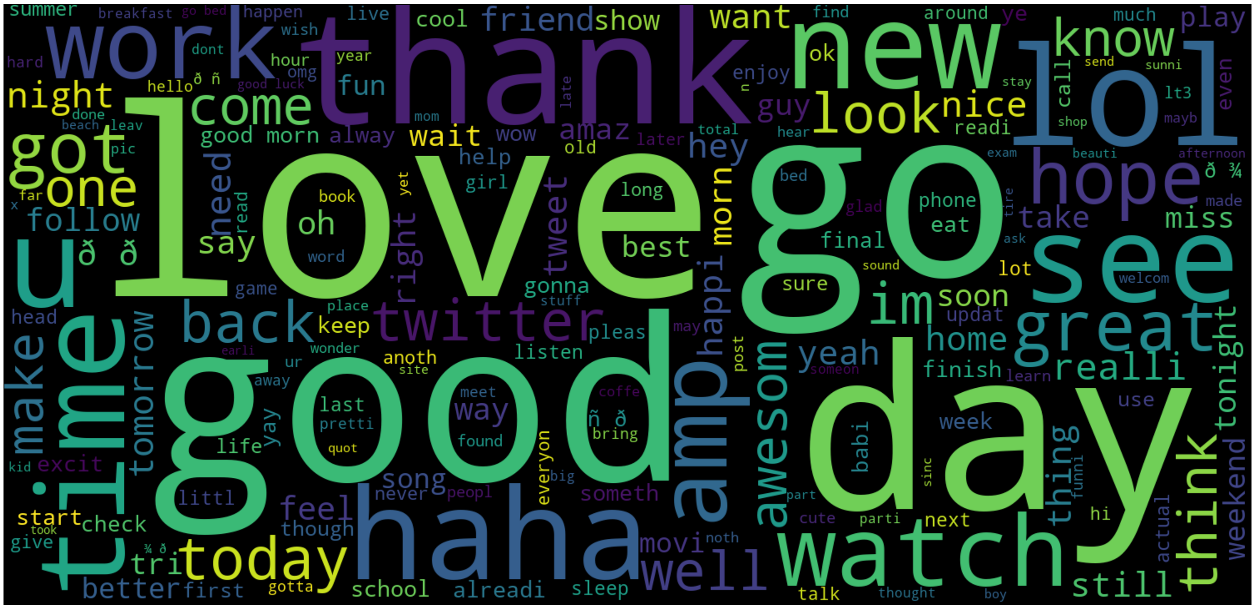
Firstly, before applying any further preprocessing we convert the text to lowercase which makes the text much easier to deal with. After which we perform the preprocessing in the following manner:

a. Using regex remove URL and HTML tag: Here we have used the same regex as mentioned above, in the previous part to find the URLs in the text, and we then replace those URLs with an empty string. For the HTML tags, we have used regex to remove tags like ‘&quot’, ‘&amp’, and ‘&lt;’ symbols which tend to escape and stay in the text. This is performed before any other operation mainly because the other operations like removing punctuations might hinder the detection of URLs.
b. Removing usernames: We also perform the removal of usernames, just after we remove the URLs for which again we use the same regex as mentioned above for finding the usernames and then replacing them with an empty string. As the data is taken straight from twitter, there are a few guidelines and rules we can follow for this task.
c. Punctuations: We use the punctuations provided by string.punctuation, which are !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~. Simply not adding the character if it exists in the punctuation string does the job.]
   This is performed after removing the URLs, so that it doesn’t pose any difficulty to the regex for finding URLs.
d. Using regex remove extra whitespaces: We use the ‘\s*\s’ as the regex string for this task.
   This operation should usually be performed before tokenization, but has no other specific ordering so can even be performed after removing URLs, or before.
e. Tokenization: By utilizing the information given to us that the dataset was curated from Twitter, we preferred going for the TweetTokenizer which is readily available using the nltk library.
f. Spelling Correction: We went for 4 approaches, 2 of them were provided by the nltk library namely, Jaccard Distance and Edit Distance, the other ones were performed using autocorrect and textblob libraries which offer spelling correction. The autocorrect library offers context driven corrections and hence we tried those as well along with naively doing corrections on each token. However we couldn’t use edit_distance for our experiments as it was too slow even when restricting edit distance and gave an ETA of onwards of 6 hours as we compared each new word to every word in the wordnet word corpus which is very large. In the end we also don’t report Jaccard scores as on manual inspection it was distorting the text as it matched single character alphabets to maximize similarity in several cases.
g. Using regex remove stopword: Here we use the stopwords corpus provided by the nltk library. For this task the framing of our regex string comprises of separating each stopword with a “|” separator (or separator) inside parentheses surrounded by the word boundaries. We then replace each stopword found using the regex string with an empty string.

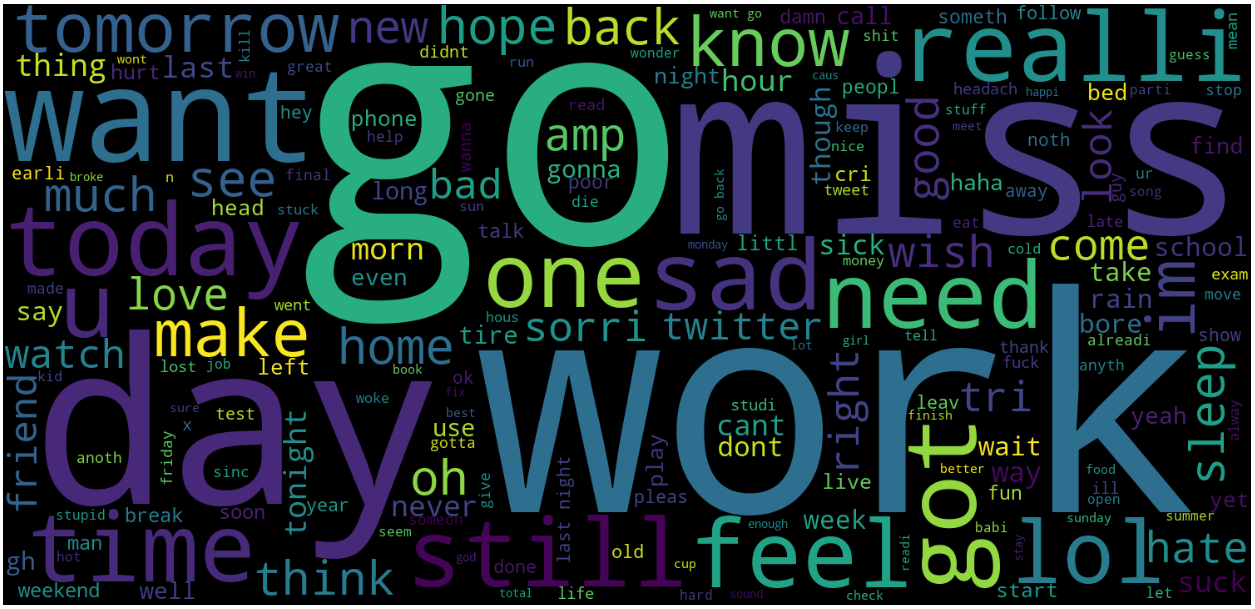
h. Lemmatization/Stemming: We went for lemmatization here. The reason being that after stemming a word tends to lose its inherent meaning, and we don’t want that for a task like Sentiment Analysis which is quite sensitive and reliant on the context.

III. VISUALIZATION

For Positive Preprocessed Text:

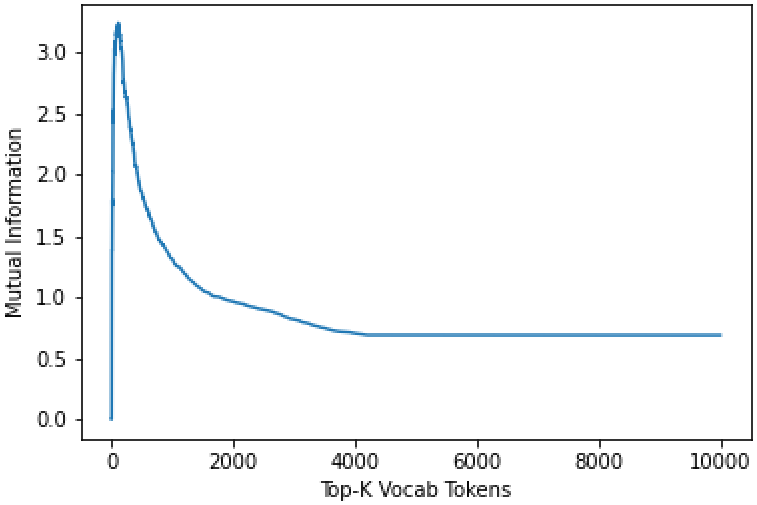


For Negative Preprocessed Text:



On seeing the vocabularies it is clear that in the most frequent words there are some common words which repeat in both classes however there are some distinctive words such as 'love', 'good' for positive and 'sad' for negative. But overall we see a clear lack of characterizing lexicon between the two and this adds to the difficulty of the task. The common words include words like ‘go’, ‘work’ etc. which do not give any information about the sentiment

If jaccard spelling checker is used to build word cloud then there are lots of single characters as they maximize jaccard similarity in many cases



A mutual information for the top k tokens plots shows that for the first 150-200 tokens there is high mutual information implying there is high similarity in most frequent words

IV. RULE-BASED SENTIMENT ANALYSIS

Since our data does not have neutral label however VADER also returns a neutral score we use different configurations to use that score or merge it with other classes

The configurations are:

1. **compound_with_neutral**: Here we use the compound score and keep the neutral label. Hence any score above 0.05 is marked positive, any below 0.05 is considered negative and anything in between is neutral.
2. **compound_without_neutral**: Here we use the compound score and do not keep the neutral label. Hence any score above 0 is marked positive, any below 0 is considered negative.
3. **neutral_mapped_to_pos_scores**: Here we use the neutral, negative and neutral scores which are given separately and do not keep the neutral label. Hence neutral and positive are mapped to positive while negative is mapped to negative.
4. **neutral_mapped_to_neg_scores**: Here we use the neutral, negative and neutral scores which are given separately and do not keep the neutral label. Hence neutral and negative are mapped to negative while positive is mapped to positive
5. **pos_neg_scores**: Here we don't use the neutral label at all and only allot classes based on positive and negative

Preprocessing Components	**Compound with neutral**	**Compound without neutral**	**Neutral mapped to pos scores**	**Neutral mapped to neg scores**	**Pos neg scores**
Raw	53.230697457429	68.34616281782	55.283414975507	51.55120130627478	68.4627944949848
Autocorrect	50.9913692558899	65.36039188243	58.222533240027	58.08257522743177	65.4536972241660
Textblob	51.3879169582458	66.20013995801	58.339164917191	58.10590156286447	66.2701189643107

Through multiple evaluations we came to the conclusion that our mapping of labels and spelling correction mechanism heavily influences our results.

This led us to the following conclusions

- In any case where we map neutral we get degraded performance due to one extra class which is not present in our data.
- Compound score is poorer metric than simply comparing positive and negative
- Spelling Correctors used heavily influence the results and weaker ones such as Jaccard show very poor performance in many places
- Spelling Correctors which can use context do not fare any better in our examples if we compare them by using them on per token basis
- Spelling Correctors can ruin results by taking away information contained in native lingo of social media platforms

INDIVIDUAL CONTRIBUTIONS

We divided the parts equally amongst ourselves. One of us worked on the 1st and 4th parts, while the other worked on the other two parts. We also tried to help each other out for both the parts so that we have an understanding of the working of all the tasks.