Project Proposal: Sentiment Analysis

Course : Natural Language Processing

Fall 2021

Contents

Project Name & Description: **2**

Participants & Roles: **2**

Workflow: **2**

Communication: 2

Google drive: 2

Github Repository: 2

Introduction: **3**

Goal and Objectives: **3**

Motivation: 3

Significance: 3

Milestones: 4

Objectives: 4

Background **5**

Model **6**

Architecture: 6

Dataset **9**

Features 10

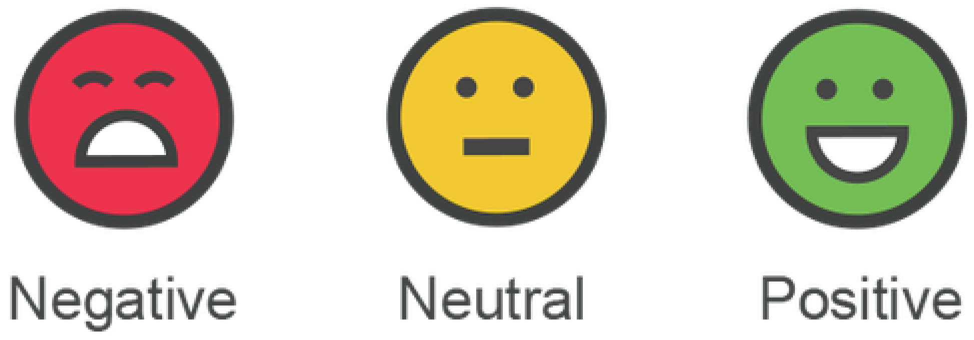
Analysis of data **16**

Implementation **18**

Results **18**

Project Management **19**

References **20**



Project Name & Description:

Sentiment Analyzer: In this project, we are going to build an application that would take in a textual input from user and provide the sentiment analysis on the given input text. The output would provide a detailed report on the sentiment identified in the given text. By analyzing these large volumes of customer feedback, sentiment analysis helps turn dissatisfied customers into promoters for the business and determine the functional and non-functional requirements of the products, as well as its price and performance.

Participants & Roles:

Yaswanth Bandaru: yaswanthbandaru@my.unt.edu
Lakshmi Vandana Nunna : LakshmiVandanaNunna@my.unt.edu
Tanuja Polineni : tanujapolineni@my.unt.edu

Workflow:

Communication:

- Weekly zoom meetings with the team.
- Email correspondence.
- Microsoft team communications

Google drive:

- We will use google drive for our shared documents and files.

Github Repository:

- Our code and data will be stored here.

- Used for version control.

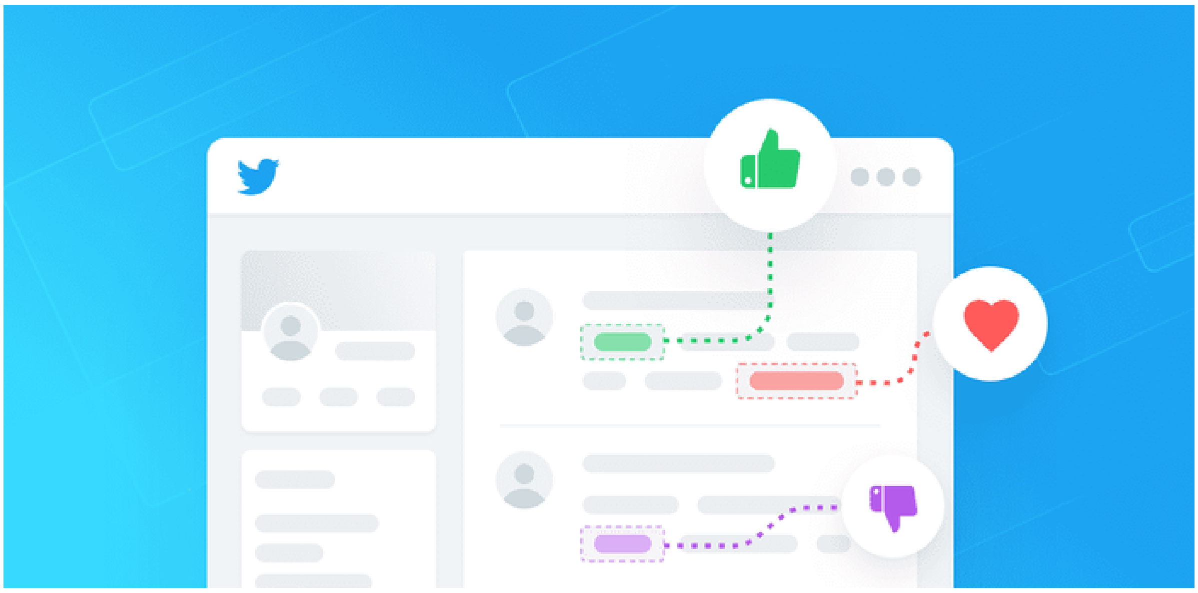
Introduction:

Goal and Objectives:

Motivation:

Sentiment analysis plays a crucial role in understanding the sentiment of people from different groups. For political purposes, it is important to know what percentage of the majority supports the government or how many oppose their services and policies. On the other hand, it helps companies to better understand their customers in the business world. By using such a resource, businesses can offer products and services in accordance with customer expectations and achieve success.

The theory of sentiment analysis is applicable to social media, and the biggest platform where it is and must be, applied in order to interpret people's feelings and opinions. (Theiler, 2019). Then, we need to learn how it operates, how it is applied, and why it is crucial for business organizations, among other things. There is a lot of textual content on social media platforms that when processed and analyzed could have great potential. This is because there are endless applications and benefits that result from this analysis. For instance, this content could help us predict a future event (ex: rise/fall of a stock price), predict the number of people attending an event, check the public opinion on a government policy etc.



Significance:

Our application helps in identifying the sentiment of a given text that helps the end user to understand the underlying motive/sentiment of the text without having to read through the text. This helps the user to analyze a lot of textual content without having to manually read through it and perceive the sentiments. We can apply sentiment analysis to automatically know the movie reviews, customer feedback, opinions. We use Natural Language Toolkit (NLTK), a commonly used NLP library in Python, to analyze textual data (Bonthu, 2021)

Milestones:

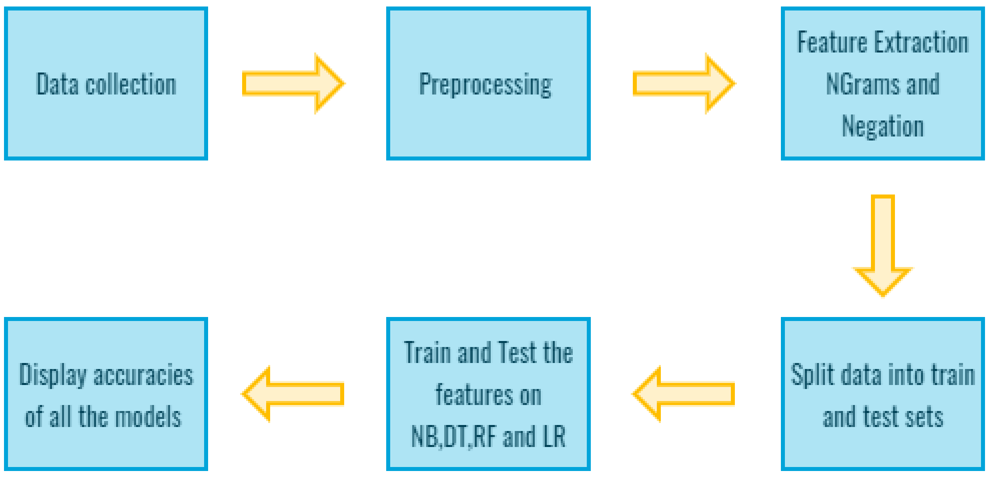
1. Requirement Analysis
2. System Design
3. Implementation
4. Testing

The application development is divided into 3 phases.

1. Collection and preprocessing of data.
2. Train the data on different classification models, perfume hyper parameter tuning and cross validation.
3. Develop GUI to interact with backend model and predict the sentiment of given text.

Objectives:

The main objective of this application is to preprocess, extract features on the given user input and feed it to the backend model that gives optimal sentiment analysis results on the given input text. So, the major focus is on cleaning and preprocessing the text while also choosing the right model that has greater AUC/prediction capability. The secondary objective is to provide the user with an interface so that he/she can interact with it to get predictions on the input text of their desire.


Data collection
Preprocessing
Feature Extraction NGrams and Negation
Split data into train and test sets
Train and Test the features on NB,DT,RF and LR
Display accuracies of all the models

NB - Naive Bayes ; DT- Decision Tree ; RF - Random Forest ; LR - Logistic Regression

Background

Sentiment Analysis:
Sentiment Analysis is a text classification that determines the opinion and subjectivity of the readers. Sentiment classification techniques are classified into three categories: machine learning approach, lexicon-based approach, and hybrid approach.

Machine Learning Approach:
Here ML algorithms with linguistic features are applied, which are supervised and unsupervised learning methods.
Supervised Learning methods are based on the existence of labeled data with different kinds of classifiers in literature, out of which below are the popular ones.

1. **Probabilistic Classifiers:**

These use mixture models (Walaa Medhat, 2014) for classification where the input is a probability distribution over a set of classes and output predicts an observation. The Probabilistic Classifiers are further divided into

a) **Naive Bayes Classifier**

Bayes theorem is used to predict the probability that a given feature set belongs to a particular label. Here, BOW - Bag Of Words can be used as feature extraction. This classifier is used more often for its good performance.

$$P(\text{label} \mid \text{features}) = \frac{P(\text{label}) * P(\text{features} \mid \text{label})}{P(\text{features})}$$

b) **Bayesian Network**

Bayesian Network is a joint probability distribution of random variables that deal with conditional dependencies using the directed acyclic graph.

c) **Maximum entropy classifier**

In this classifier, the labeled features are converted to vectors using encoding and used to calculate each feature's weights. These classifiers are mainly used to detect parallel sentences between language pairs.
The probability of each label is computed as

$$P(\text{fs} \mid \text{label}) = \frac{\text{dotprod(weights, encode(fs, label))}}{\text{Sum(dotprod(weights, encode(fs, l)) forlinlables)}}$$

2. **Linear Classifiers:**

In linear classification, the output is a hyperplane that separates between classes. It makes a decision based on the linear relationship of the feature vectors. The most popular classifiers are SVM and NN.

a) **Support Vector Machines Classifiers (SVM)**

SVM analyzes data for classification and regression analysis. This classifier can be used as a sentiment polarity classifier mainly in Market Intelligence by emphasizing opinion subjectivity and expresser credibility.

b) **Neural Network (NN)**

The neural network is a network of neurons where each neuron in text classification would be the word frequencies associated with weights. For non-linear data, we use Multilayer Neural Networks. Recent experiments say that ANN provides better results than SVM, with an exception for unbalanced data. DNN is used for more precise results.

(Walaa Medhat, 2014)

Lexicon Based Approach:
This approach mainly uses parts of speech and WordNet for analyzing the text inputs. There are Dictionary-based approaches and a Corpus-based approach. Domain and context-specific opinion words are identified using the corpus approach. But the corpus approach alone cannot be more effective than the Dictionary approach.

Hybrid Approaches:
These methods combine machine learning and lexical resources.

Model

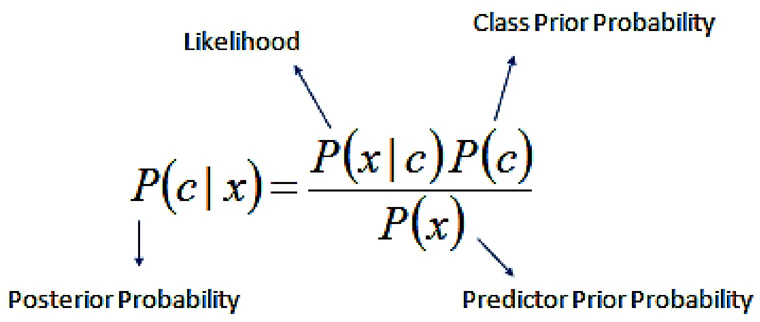
Architecture:

As described above, we train our extracted features on Naive Bayes, Decision Tree, Random Forest and Logistic regression classifiers. So, we do not have a customized model architecture.

So, here we present the architectures/fundamental designs of these classifiers.

Naive Bayes Classifier

This classifier works on the basic conditional probability presented by Bayes. Based on the predictors/input variables provided, we calculate likelihoods(probabilities) for predicted prior probability, class prior probability and the probability of the value of an input variable given a particular class. The posterior probability calculated would finally result in the optimal prediction of the class given an input.

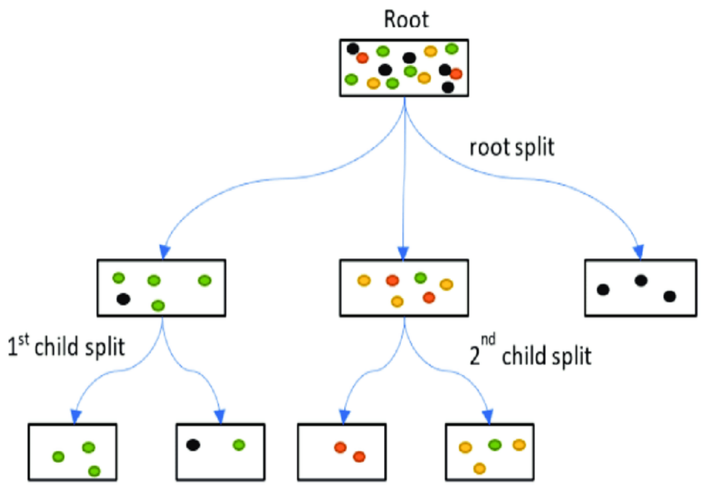


$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

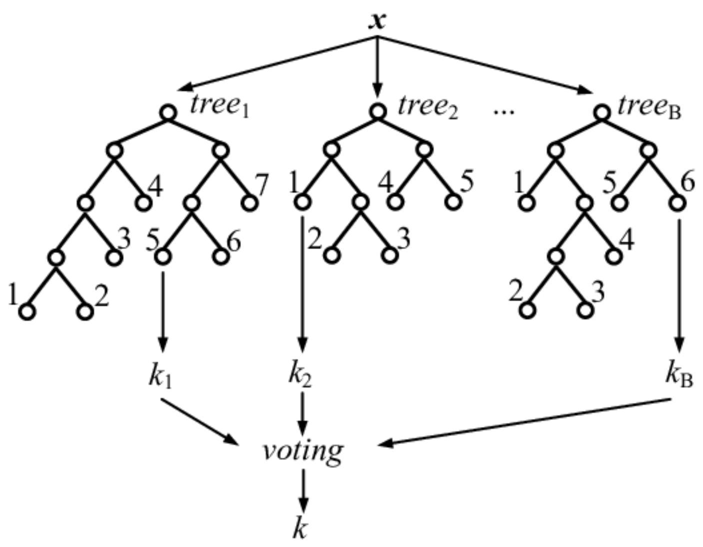
Decision Tree Classifier

This classifier works on the concept of a tree structure wherein each input feature is a decision node while the branches represent the decision rules. The leaf nodes represent the final outcome. This helps in grouping and organizing the features. Gini impurity or Entropy and Information gain are the metrics used for decision tree.



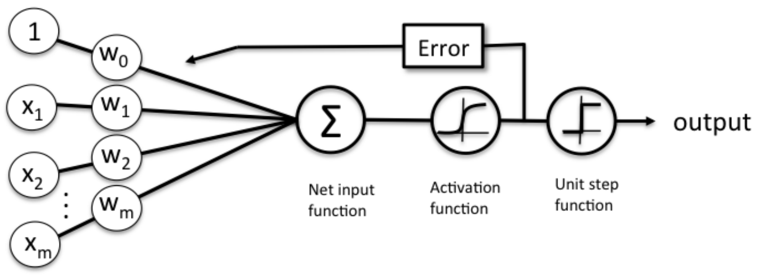
Random Forest Classifier

This classifier is an amalgamation of multiple Decision Trees. These tree are generated based on randomly selected data entries to make sure that the process avoids overfitting.The predictions from all the trees are considered and finally the optimal prediction is given as output based on majority result from the decision trees.



Logistic Regression Classifier

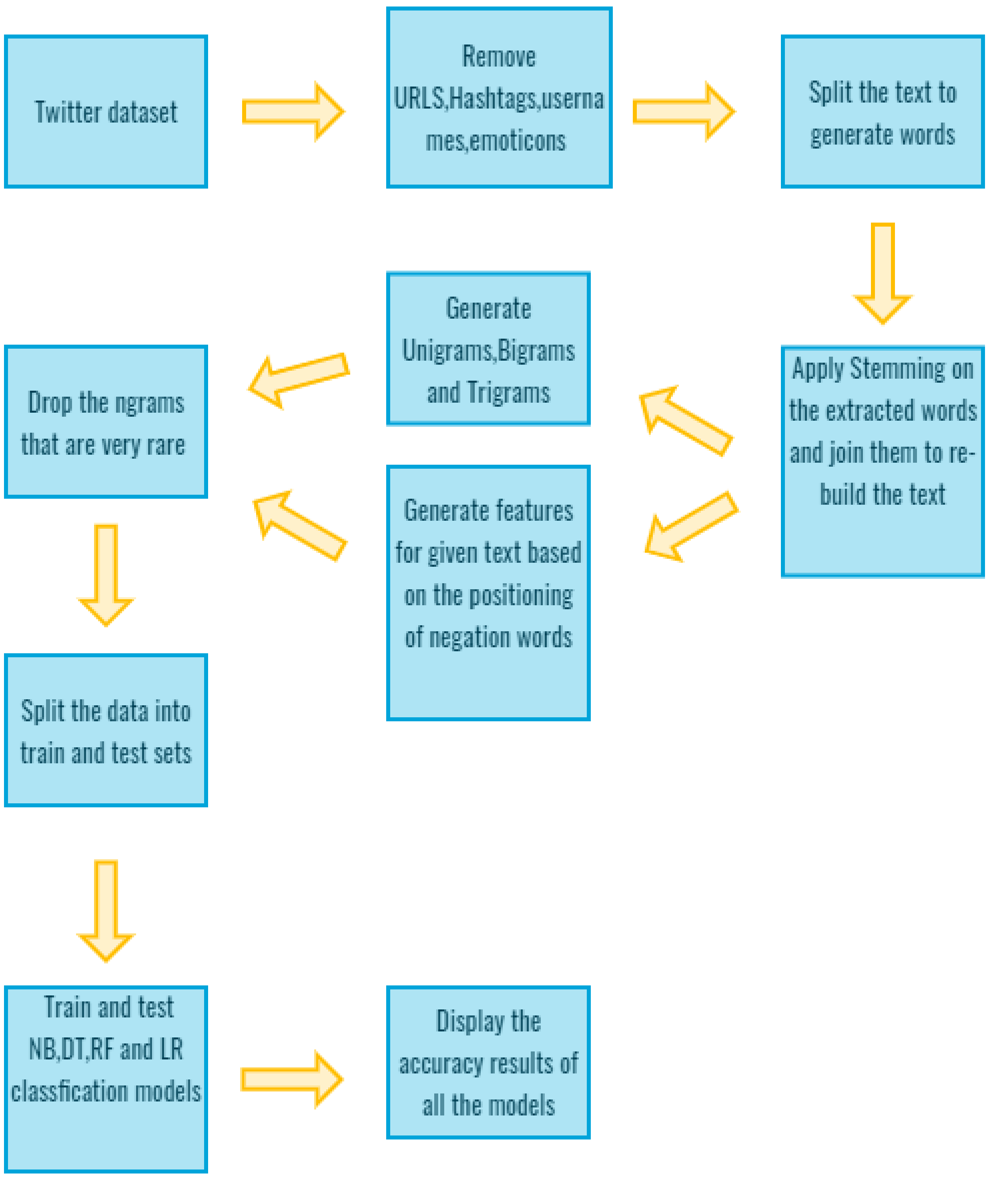
This classifier works based on a sigmoid function to interpret the tendency of given input data towards a specific class. Since the task is classification but not regression, the data would lie in two different sections as opposed to a straight line. It is impossible to fit the categorical data in a straight line while it could be well fit in a step shape plot. To address this, sigmoid would generate a value between 0 and 1 for any given input real value. For a decision boundary of 0.5, any value greater than or equal to 0.5 is considered as one class while values lesser than 0.5 are considered as another class.



Schematic of a logistic regression classifier.

Workflow:

Below workflow diagram illustrates the process step by step. First, we retrieve the twitter dataset. Second, clean up the data by removing URLS, hashtags, usernames and emoticons because these are irrelevant for us to infer the sentiment of the text. Third, we split the cleaned text into words. Fourth, we apply stemming on the generated words and join them to rebuild the text. Fifth, we generate unigrams, bigrams and trigrams from the processed text; we also generate features based on the position of the negation words. Sixth, we drop the ngrams that are very rare so that the data is enriched with more relevant features. Seventh, we split the data into train and test sets.Eight, we train and test the Naive bayes, Decision tree, Random forest and Logistic regression models based on this data. Finally, we display the accuracies of all the models.


Twitter dataset
Remove URLS,Hashtags,usernames,emoticons
Split the text to generate words
Apply Stemming on the extracted words and join them to re-build the text
Generate Unigrams,Bigrams and Trigrams
Generate features for given text based on the positioning of negation words
Drop the ngrams that are very rare
Split the data into train and test sets
Train and test NB,DT,RF and LR classfication models
Display the accuracy results of all the models

Dataset

Dataset: Our dataset has 1,600,000 tweets, extracted from Twitter API. We annotated the tweets as negative for 0, neutral for 2, positive for 4 for detecting sentiments. Dataset has the following fields – target, ids, date, flag, user, text.

The training data was created automatically instead of human tweets. Emoticons are classified into positive and negative. Twitter Search API is used.

citation: Go, A., Bhayani, R. and Huang, L., 2009. Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 1(2009), p.12.

Now, each tweet in the dataset is in raw format that includes URLs, Hashtags, Handles, emoticons, and words.

Preprocessing data:

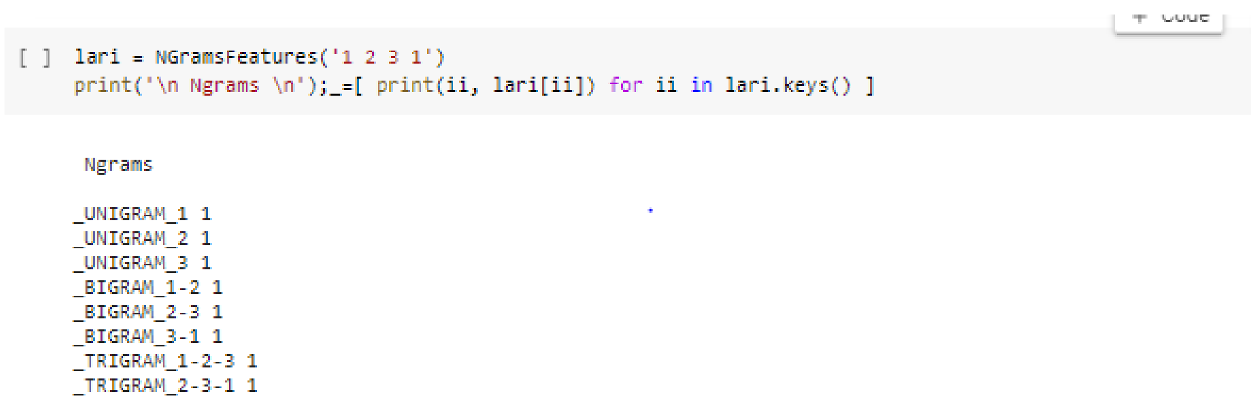
- Remove hashtags, handles, URLs and emoticons.
- Remove punctuations
- Remove repeating character like - Happyyyyyyyyyy Happy
- Apply Stemming
- Count the number of unigrams, bigrams and trigrams and negating words in the text
- Generate a new dataset set using the newly features generated in above step. (Daityari, 2019)

Features

Once the preprocessing is done, we parse the words in each tweet to generate the list of unigrams, bigrams and trigrams. Merging all these lists would eventually give us a superset - this becomes the feature list. The presence and absence of each of these n-gram words would be the values for each tweet/data instance. Since typically twitter sentiment analysis happens on a corpus of tweets that belong to a specific hashtag or a specific topic, we believe that when we pull up this n-gram list, the entire corpus would have a limited no of n-grams since the same combination of words are repeated given that they all are about the same topic.

Along with these features, we also check for negation words to understand the underlying emotion. For instance, “This is a good policy” and “This is not a good policy” convey different sentiments. So, screening the corpus for negating words like no,never,not,wont etc would make sure that the inferences are pulled optimally. We apply regular expressions to the text to identify these negations. Then, calculate the left negativity and right negativity of each word in the text. These left and right negative impact probability on each word is treated as a feature in itself. These features are also merged into the list of N-gram features discussed above. This final list of features is fed to the classification models to predict the neg/pos sentiment on a given text.

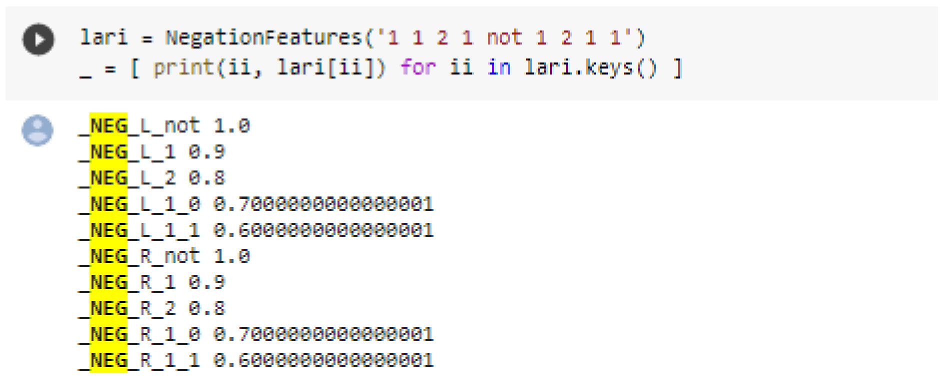
To analyse the way we are generating the features, you can see in the below demonstration that the N gram features generated for a text - ‘1 2 3 1’ are as below:



```
[ ]  lari = NGramsFeatures('1 2 3 1')
     print('\n Ngrams \n');_=[ print(ii, lari[ii]) for ii in lari.keys() ]
```

```
 Ngrams

_UNIGRAM_1 1
_UNIGRAM_2 1
_UNIGRAM_3 1
_BIGRAM_1-2 1
_BIGRAM_2-3 1
_BIGRAM_3-1 1
_TRIGRAM_1-2-3 1
_TRIGRAM_2-3-1 1
```

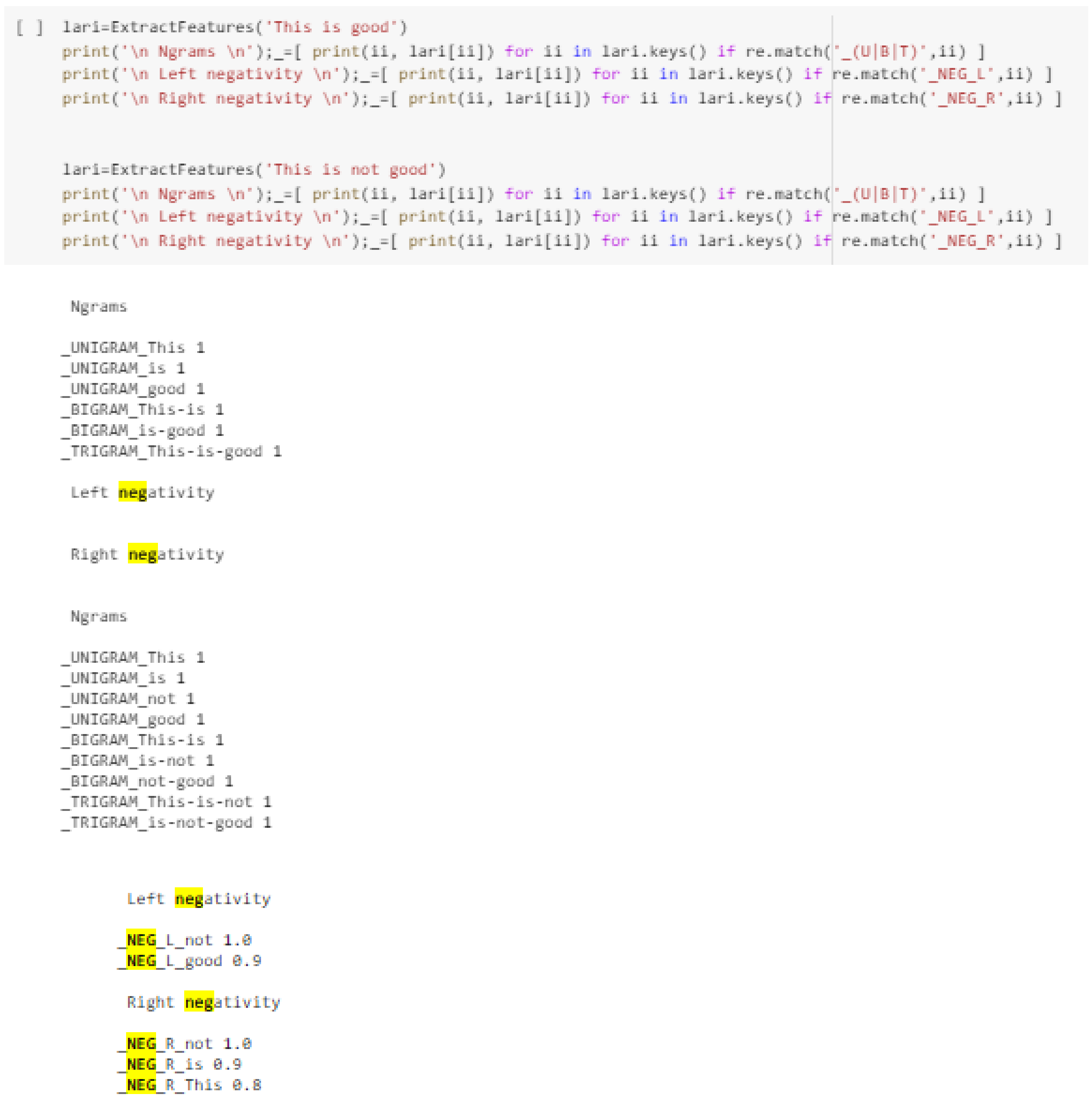
The Negation features extracted from a given text of ' 1 1 2 1 not 1 2 1 1' are as below. You observe that the word right after not has the highest negation value.



```
lari = NegationFeatures('1 1 2 1 not 1 2 1 1')
_ = [ print(ii, lari[ii]) for ii in lari.keys() ]
```

```
_NEG_L_not 1.0
_NEG_L_1 0.9
_NEG_L_2 0.8
_NEG_L_1_0 0.7000000000000001
_NEG_L_1_1 0.6000000000000001
_NEG_R_not 1.0
_NEG_R_1 0.9
_NEG_R_2 0.8
_NEG_R_1_0 0.7000000000000001
_NEG_R_1_1 0.6000000000000001
```

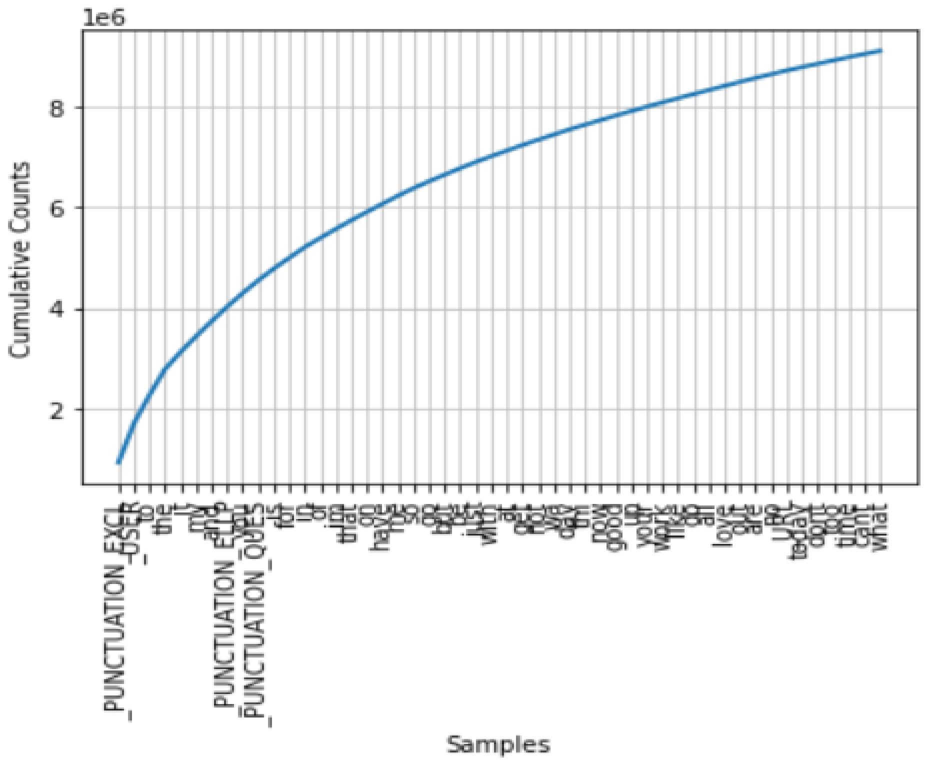
Below is the result of the ExtractFeatures function that would generate ngram, left negation and right negation features of a given text:



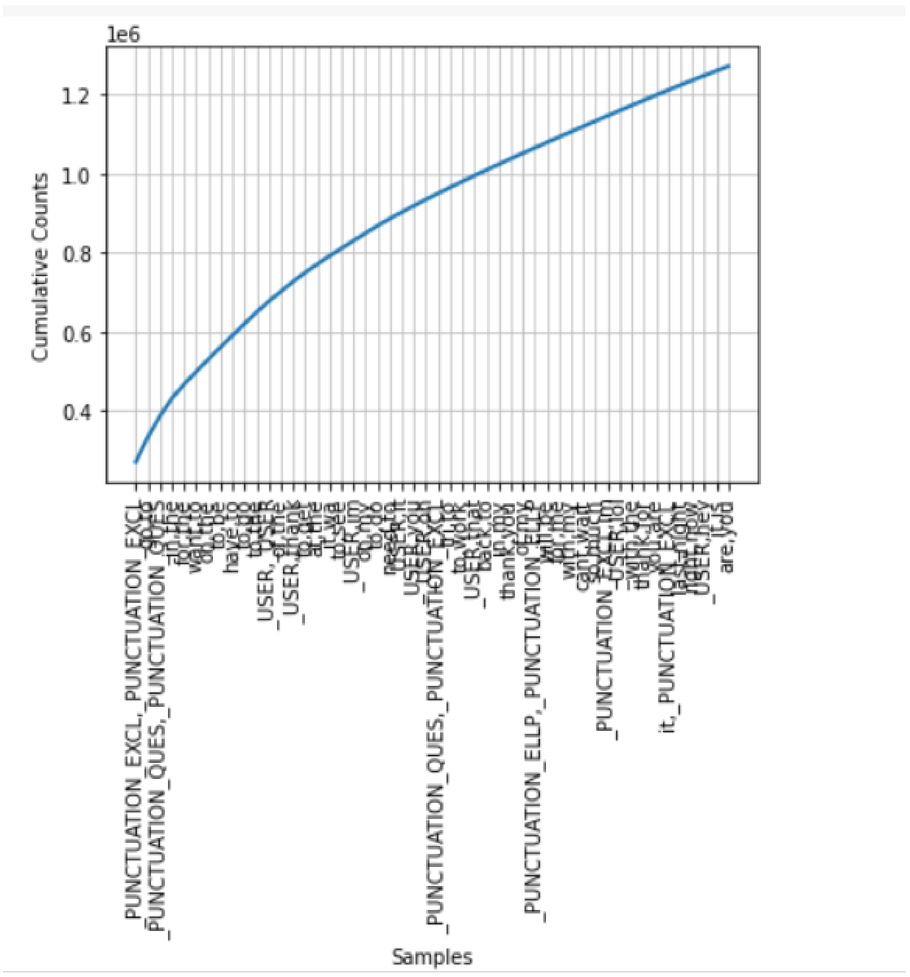
```
[ ]  lari=ExtractFeatures('This is good')
     print('\n Ngrams \n');_=[ print(ii, lari[ii]) for ii in lari.keys() if re.match('_(U|B|T)',ii) ]
     print('\n Left negativity \n');_=[ print(ii, lari[ii]) for ii in lari.keys() if re.match('_NEG_L',ii) ]
     print('\n Right negativity \n');_=[ print(ii, lari[ii]) for ii in lari.keys() if re.match('_NEG_R',ii) ]


     lari=ExtractFeatures('This is not good')
     print('\n Ngrams \n');_=[ print(ii, lari[ii]) for ii in lari.keys() if re.match('_(U|B|T)',ii) ]
     print('\n Left negativity \n');_=[ print(ii, lari[ii]) for ii in lari.keys() if re.match('_NEG_L',ii) ]
     print('\n Right negativity \n');_=[ print(ii, lari[ii]) for ii in lari.keys() if re.match('_NEG_R',ii) ]
```

```
 Ngrams

_UNIGRAM_This 1
_UNIGRAM_is 1
_UNIGRAM_good 1
_BIGRAM_This-is 1
_BIGRAM_is-good 1
_TRIGRAM_This-is-good 1

 Left negativity


 Right negativity


 Ngrams

_UNIGRAM_This 1
_UNIGRAM_is 1
_UNIGRAM_not 1
_UNIGRAM_good 1
_BIGRAM_This-is 1
_BIGRAM_is-not 1
_BIGRAM_not-good 1
_TRIGRAM_This-is-not 1
_TRIGRAM_is-not-good 1


 Left negativity

_NEG_L_not 1.0
_NEG_L_good 0.9

 Right negativity

_NEG_R_not 1.0
_NEG_R_is 0.9
_NEG_R_This 0.8
```

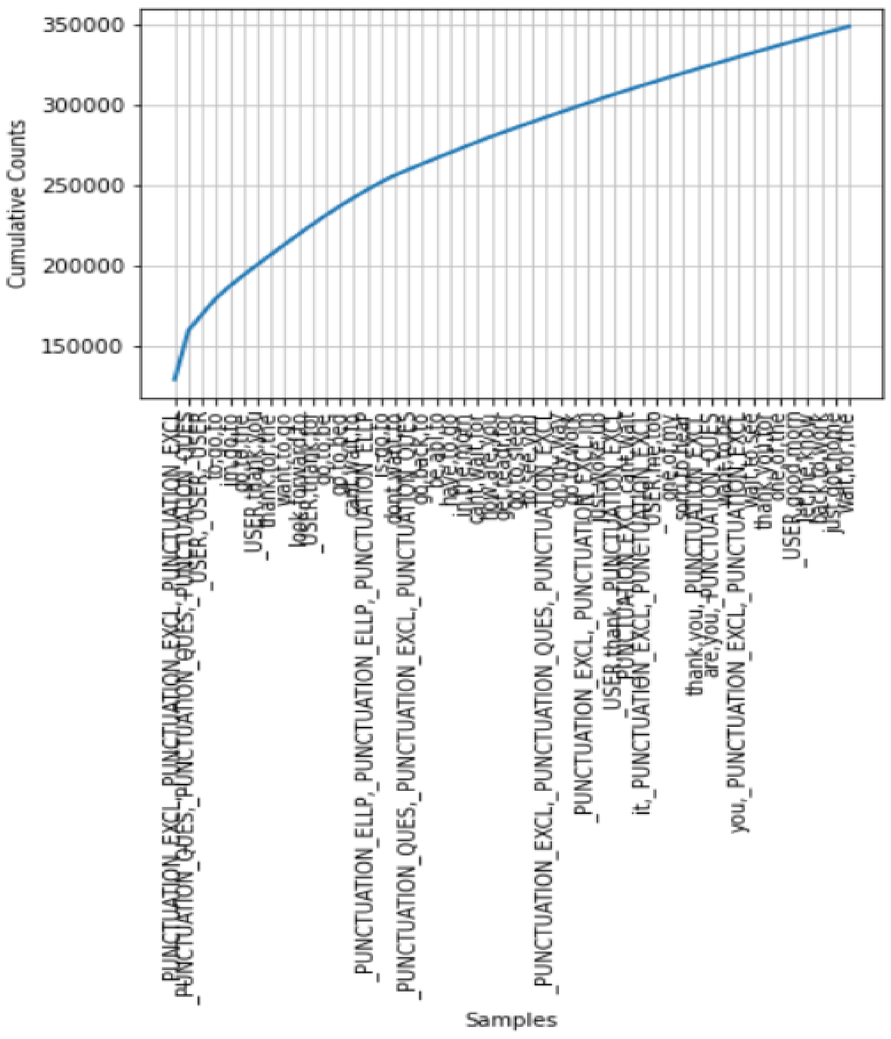
Unigram distribution



Bigram Distribution



Trigram Distribution

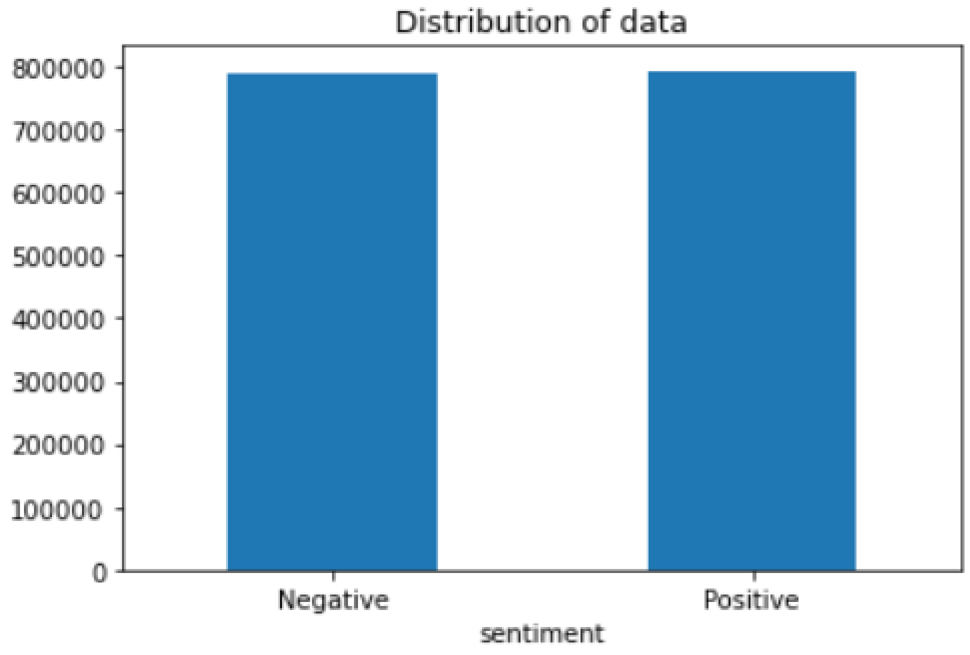


Analysis of data

Data Preprocessing:

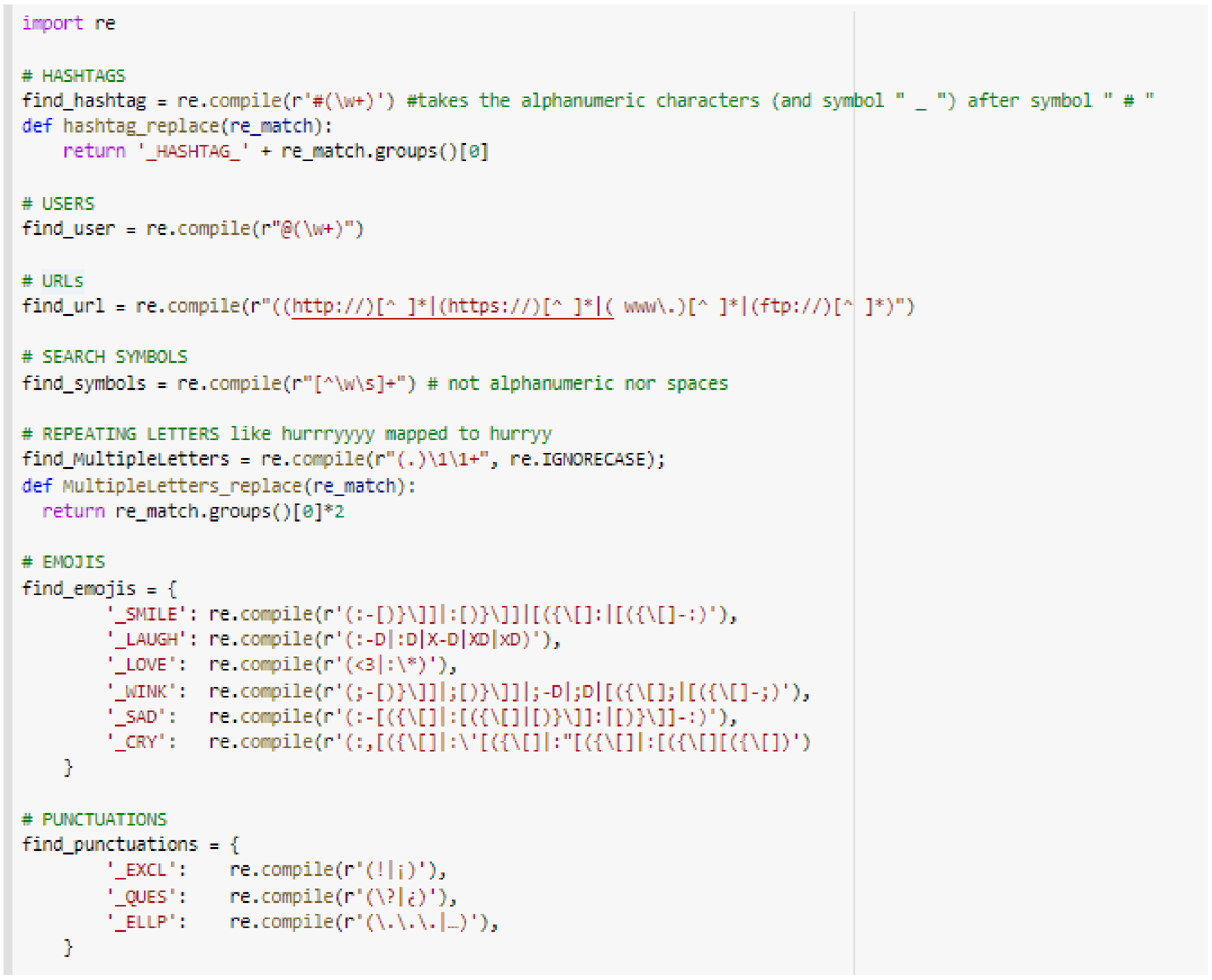
Data distribution

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1583691 entries, 0 to 1599999
Data columns (total 2 columns):
 #   Column     Non-Null Count    Dtype 
---  ------     --------------    ----- 
 0   sentiment  1583691 non-null  int64 
 1   text       1583691 non-null  object
dtypes: int64(1), object(1)
memory usage: 36.2+ MB
None
[Text(0, 0, 'Negative'), Text(1, 0, 'Positive')]
```

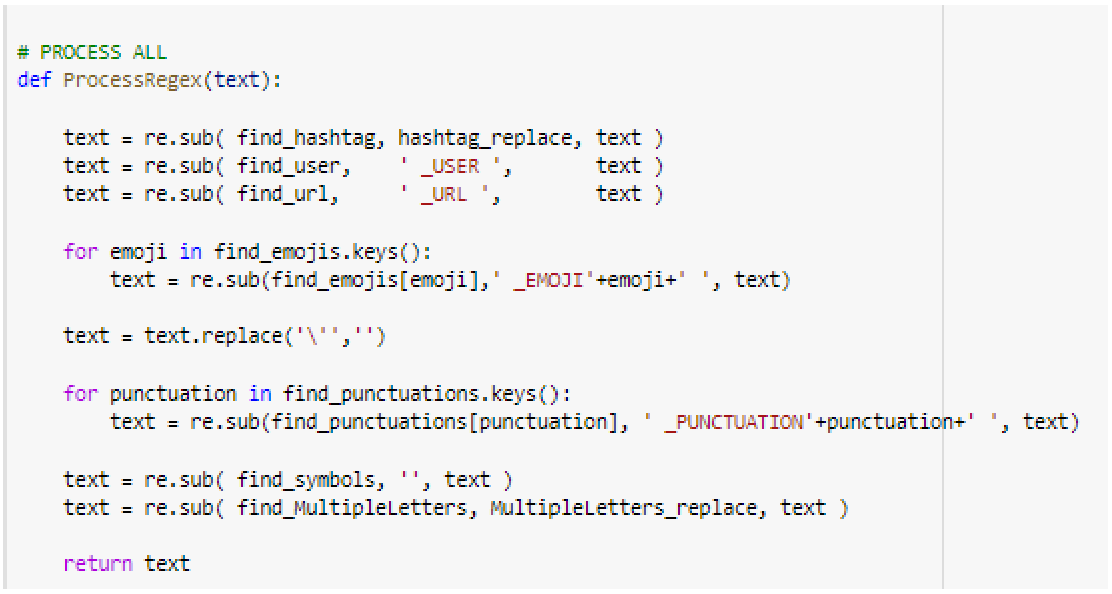


Data cleaning:

Remove URLS,usernames,hashtags and emoticons:

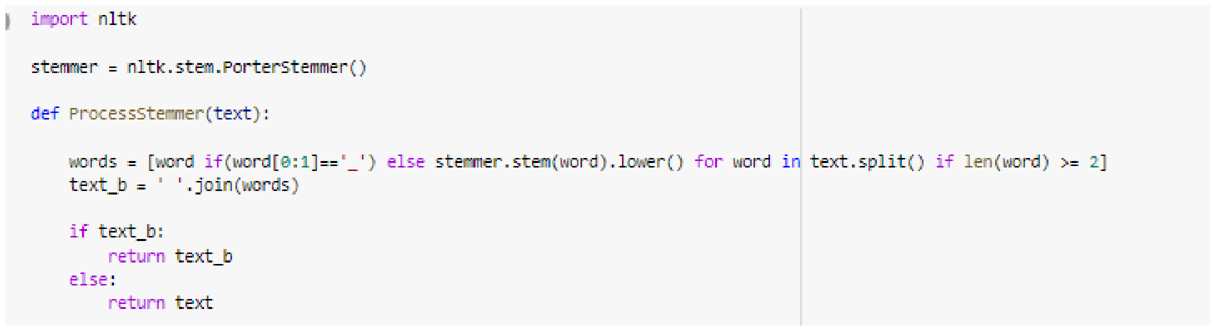


```
import re

# HASHTAGS
find_hashtag = re.compile(r'#(\w+)') #takes the alphanumeric characters (and symbol " _ ") after symbol " # "
def hashtag_replace(re_match):
    return '_HASHTAG_' + re_match.groups()[0]

# USERS
find_user = re.compile(r"@(\w+)")

# URLs
find_url = re.compile(r"((http://)[^ ]*|(https://)[^ ]*|( www\.)[^ ]*|(ftp://)[^ ]*)")

# SEARCH SYMBOLS
find_symbols = re.compile(r"[^\w\s]+") # not alphanumeric nor spaces

# REPEATING LETTERS like hurrryyyy mapped to hurryy
find_MultipleLetters = re.compile(r"(.)\1\1+", re.IGNORECASE);
def MultipleLetters_replace(re_match):
  return re_match.groups()[0]*2

# EMOJIS
find_emojis = {
        '_SMILE': re.compile(r'(:-[)}\]]|:[)}\]]|[({\[]:|[({\[]-:)'),
        '_LAUGH': re.compile(r'(:-D|:D|X-D|XD|xD)'),
        '_LOVE':  re.compile(r'(<3|:\*)'),
        '_WINK':  re.compile(r'(;-[)}\]]|;[)}\]]|;-D|;D|[({\[];|[({\[]-;)'),
        '_SAD':   re.compile(r'(:-[({\[]|:[({\[]|[)}\]]:|[)}\]]-:)'),
        '_CRY':   re.compile(r'(:,[({\[]|:\'[({\[]|:"[({\[]|:[({\[][({\[])')
    }

# PUNCTUATIONS
find_punctuations = {
        '_EXCL':    re.compile(r'(!|¡)'),
        '_QUES':    re.compile(r'(\?|¿)'),
        '_ELLP':    re.compile(r'(\.\.\.|…)'),
    }
```



```
# PROCESS ALL
def ProcessRegex(text):

    text = re.sub( find_hashtag, hashtag_replace, text )
    text = re.sub( find_user,    ' _USER ',       text )
    text = re.sub( find_url,     ' _URL ',        text )

    for emoji in find_emojis.keys():
        text = re.sub(find_emojis[emoji],' _EMOJI'+emoji+' ', text)

    text = text.replace('\'','')

    for punctuation in find_punctuations.keys():
        text = re.sub(find_punctuations[punctuation], ' _PUNCTUATION'+punctuation+' ', text)

    text = re.sub( find_symbols, '', text )
    text = re.sub( find_MultipleLetters, MultipleLetters_replace, text )

    return text
```

Stemming:



```
import nltk

stemmer = nltk.stem.PorterStemmer()

def ProcessStemmer(text):

    words = [word if(word[0:1]=='_') else stemmer.stem(word).lower() for word in text.split() if len(word) >= 2]
    text_b = ' '.join(words)

    if text_b:
        return text_b
    else:
        return text
```

Implementation

Implementation steps are as follows:

- Retrieve the Twitter dataset.
- Clean up the data by removing URLS, hashtags, usernames and emoticons because these are irrelevant for us to infer the sentiment of the text.
- We split the cleaned text into words.
- We apply stemming to the generated words and join them to rebuild the text.
- We generate unigrams, bigrams and trigrams from the processed text; we also generate features based on the position of the negation words.
- We drop the Ngrams that are very rare so that the data is enriched with more relevant features.
- We split the data into train and test sets.
- We train and test the Naive Bayes, Decision tree, Random forest and Logistic regression models based on this data.
- Finally, we display the accuracy of all the models.

GitHub Link: https://github.com/yaswanth-bandaru/NLP_Sentiment_Analysis.git

Results

Below are the accuracy results of all the models with respect to the features that they were trained on.

```
####################
####################

classif NaiveBayes
features Ngrams
Training
...
Testing
Accuracy: 0.7685097193588412
Confusion Matrix
  |      0      1 |
--+---------------+
0 |<135480> 22579 |
1 |  50743<107937>|
--+---------------+
(row = reference; col = test)


####################
####################

classif NaiveBayes
features Negation
Training
...
Testing
Accuracy: 0.7565692889097964
Confusion Matrix
  |      0      1 |
--+---------------+
0 |<140733> 17326 |
1 |  59778 <98902>|
--+---------------+
(row = reference; col = test)



####################
####################

classif NaiveBayes
features
Training
...
Testing
Accuracy: 0.7685097193588412
Confusion Matrix
  |      0      1 |
--+---------------+
0 |<135480> 22579 |
1 |  50743<107937>|
--+---------------+
(row = reference; col = test)
```

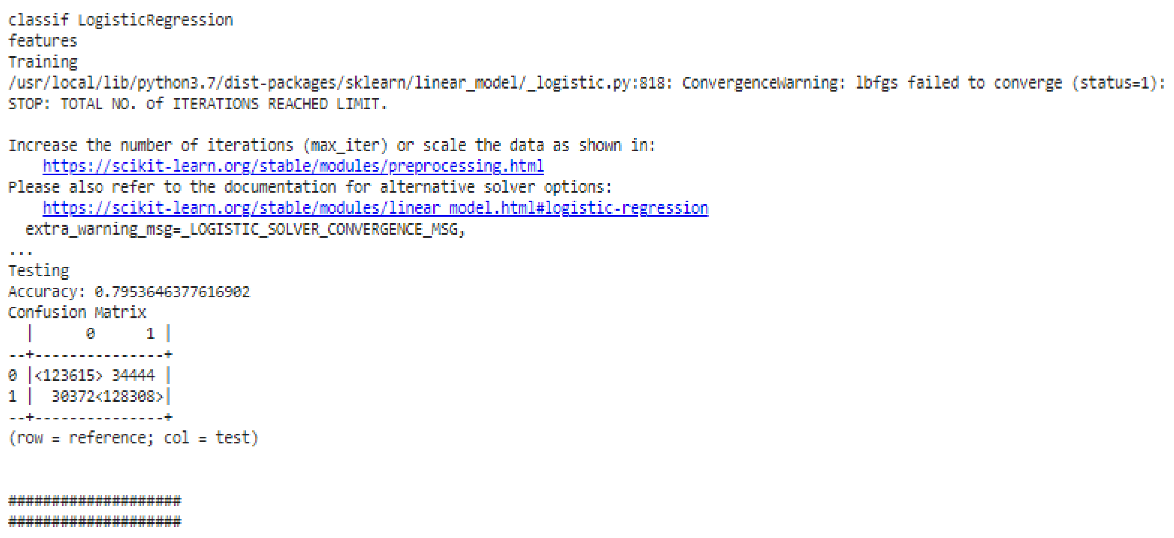
```
####################
####################

classif DecisionTree
features Ngrams
Training
...
Testing
Accuracy: 0.7329504734181771
Confusion Matrix
  |      0      1 |
--+---------------+
0 |<110814> 47245 |
1 |  37340<121340>|
--+---------------+
(row = reference; col = test)
```

```
####################
####################

classif DecisionTree
features Negation
Training
...
Testing
Accuracy: 0.7346806045355956
Confusion Matrix
  |      0      1 |
--+---------------+
0 |<110934> 47125 |
1 |  36912<121768>|
--+---------------+
(row = reference; col = test)
```

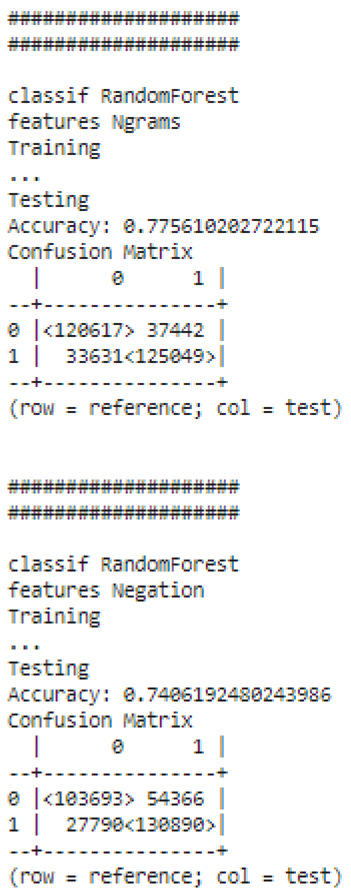
```
####################
####################

classif DecisionTree
features
Training
...
Testing
Accuracy: 0.7323758678280856
Confusion Matrix
  |      0      1 |
--+---------------+
0 |<110662> 47397 |
1 |  37370<121310>|
--+---------------+
(row = reference; col = test)
```

```
classif LogisticRegression
features Ngrams
Training
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
...
Testing
Accuracy: 0.7953646377616902
Confusion Matrix
  |      0      1 |
--+---------------+
0 |<123615> 34444 |
1 |  30372<128308>|
--+---------------+
(row = reference; col = test)
```

```
####################
####################

classif LogisticRegression
features Negation
Training
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
...
Testing
Accuracy: 0.8067936060920822
Confusion Matrix
  |      0      1 |
--+---------------+
0 |<125438> 32621 |
1 |  28575<130105>|
--+---------------+
(row = reference; col = test)
```

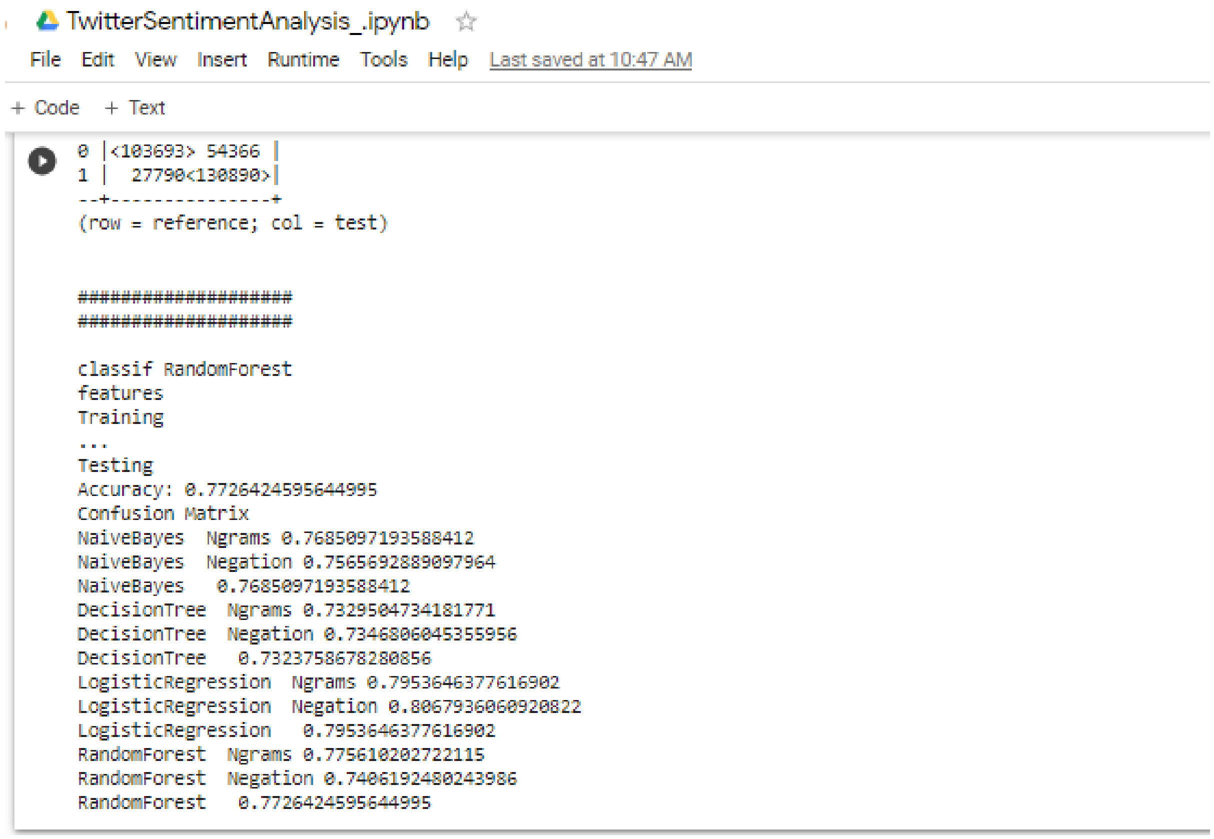


```
classif LogisticRegression
features
Training
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
...
Testing
Accuracy: 0.7953646377616902
Confusion Matrix
  |      0      1 |
--+---------------+
0 |<123615> 34444 |
1 |  30372<128308>|
--+---------------+
(row = reference; col = test)


####################
####################
```



```
####################
####################

classif RandomForest
features Ngrams
Training
...
Testing
Accuracy: 0.775610202722115
Confusion Matrix
  |      0      1 |
--+---------------+
0 |<120617> 37442 |
1 |  33631<125049>|
--+---------------+
(row = reference; col = test)


####################
####################

classif RandomForest
features Negation
Training
...
Testing
Accuracy: 0.7406192480243986
Confusion Matrix
  |      0      1 |
--+---------------+
0 |<103693> 54366 |
1 |  27790<130890>|
--+---------------+
(row = reference; col = test)
```

It is observed that Logistic regression gave best results when trained on Negation features.


TwitterSentimentAnalysis_.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:47 AM
+ Code + Text

```
0 |<103693> 54366 |
1 |  27790<130890>|
--+---------------+
(row = reference; col = test)


####################
####################

classif RandomForest
features
Training
...
Testing
Accuracy: 0.7726424595644995
Confusion Matrix
NaiveBayes  Ngrams 0.7685097193588412
NaiveBayes  Negation 0.7565692889097964
NaiveBayes   0.7685097193588412
DecisionTree  Ngrams 0.7329504734181771
DecisionTree  Negation 0.7346806045355956
DecisionTree   0.7323758678280856
LogisticRegression  Ngrams 0.7953646377616902
LogisticRegression  Negation 0.8067936060920822
LogisticRegression   0.7953646377616902
RandomForest  Ngrams 0.775610202722115
RandomForest  Negation 0.7406192480243986
RandomForest   0.7726424595644995
```

Project Management

Implementation Status Report:

- **Work Completed:** Dataset collection, Preprocessing, Feature Extraction and Modelling.
  - **Description:** Data has been collected from Kaggle and then preprocessed, such as removing punctuation, URLs, user names, symbols, emojis, repeated characters and performed stemming. We then extracted unigrams, bigrams, trigrams, and negation features. Finally, using different combinations of features we trained Naive Bayes, Decision Tree, Random Forest and Logistic Regression.
  - **Responsibility:** Coding and documentation have been done together. Tanuja has handled data collection, analysis of data and preprocessing, Vandana has dealt with feature extraction techniques such as unigrams, bigrams, trigrams, negation features and Yaswanth has dealt with modelling.
  - **Contribution:** The team members contributed equally.
  - **Issues/Concerns:** As the number of input features is large, it takes a lot of time to train because of which we had to limit the number of iterations.

References

(2020, Jan). Retrieved from Geeks For Geeks: https://www.geeksforgeeks.org/sentiment-detector-gui-using-tkinter-python/

Bonthu, H. (2021, Jan). Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2021/06/rule-based-sentiment-analysis-in-python/

Daityari, S. (2019, September). Retrieved from Digital Ocean: https://www.digitalocean.com/community/tutorials/how-to-perform-sentiment-analysis-in-python-3-using-the-natural-language-toolkit-nltk

Go, A. (2013, 01 01). *For Academics*. Retrieved from Sentiment140: http://help.sentiment140.com/for-students

Theiler, S. (2019, Dec). Retrieved from Analytics Vidhya: https://medium.com/analytics-vidhya/building-a-twitter-sentiment-analysis-app-using-streamlit-d16e9f5591f8

Walaa Medhat, A. H. (2014). Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 1093-1113. Retrieved from Science Direct: https://www.sciencedirect.com/science/article/pii/S2090447914000550?via%3Dihub