



SENTIMENT ANALYSIS OF TWEETS

Submitted By GroupX:

Chandan Goel

Gopikrishnan Srinivasan

Hitesh Yadav

Sanjeev Singh

Shubhendu Bhaskar

Table of Contents

SENTIMENT ANALYSIS OF TWEETS	2
Preface	2
Why Sentiment Analysis of Tweets?	2
1. Overview	3
2. How to use the software?	4
a) Software Requirements:	4
b) Code execution:	5
3. How is Software implemented?	5
a) Importing required Dependencies	6
b) Reading and Loading the Kaggle Dataset	8
c) Data Curation and Pre-processing	8
d) Extract tweets (features) and the corresponding targets (labels)	10
e) Split dataset into train and test datasets	11
f) Create, train, and transform the Model	11
g) Model Evaluation	12
4. Model Performance Evaluation	13
5. Description of Team Contribution	17
6. Related Work	18
7. References	18

SENTIMENT ANALYSIS OF TWEETS

Preface

With quick and easy access to the internet through cell phones, people are not hesitant anymore to express themselves over social media on a global scale. Users are expressing themselves be it the presidential election or review of a single dollar product.

This is one of the major reasons that Twitter has become the gargantuan barn of opinions and emotions therefore, product owners, political parties, government agencies, etc. track the viewpoints of the intended audiences via this social media giant. Consequently, there comes a need that requires examination and understanding of the sentiment associated with the tweets that depict the emotional inclination of the individuals. This paves the path to sentiment analysis.

Why Sentiment Analysis of Tweets?

We are currently witnessing the technological boom in all sphere humans have stepped on. In the last two decades, the tech around us has not only shown the advantages of its power but also has drawn several disadvantages towards humankind. For instance, the intent of Twitter creation was to connect its users and allow them to share their thoughts with their followers and others through the use of hashtags ([cited](#)) but not long ago, we discovered the dark side of this application. A few of such impacts are:

- Mental Health: Depression, Anxiety, Insomnia, etc.
- Cyberbullying
- Fear of Missing Out (FOMO)
- Unrealistic Expectations
- Negative Body Image
- General Addiction
- Political Polarization

Therefore, with both positive and negative impacts, the Twitter application aligns very well with our objective of classifying a piece of text into positive or negative emotions. This kind of analysis

Sentiment Analysis of Tweets

can be further extended by individuals to understand and weigh opinions and make judgments accordingly.


1. Overview

The application (tool) developed allows user to enter a text (tweet) on the screen and computationally determines the sentiment of the tweet for the intended audience.


This tool can be used by different stake holders in overviewing the sentiments for a specific objective and understanding the generic opinion trend. For instance, a government may extend this application and integrate tweets (or text from any other application) to assess the public opinion about the policy they want to bring to the congress for its citizen's welfare.

Example of the application output:

Positive tweet sentiment analysis:

 **Sentiment Analysis of Tweets**

The Black Friday event has become a global opportunity for marketing and companies' strategies aimed at increasing sale




Submit

Reset


Positive

Our model has analyzed your tweet message and resulted in a Positive sentiment.



Sentiment Analysis of Tweets

Negative tweet sentiment analysis:

 **Sentiment Analysis of Tweets**

Suicide has been one of the leading causes of deaths in the United States. One major cause of suicide is psychiatric stressors


1+

Submit

Reset


Negative

Our model has analyzed your tweet message and resulted in a Negative sentiment.



2. How to use the software?

The user interface of the application is quite straightforward. The user is required to enter the tweet in the provided text box. Click the “Submit” button to analyze the tweet and produce the corresponding analysis output.

 **Twitter Sentiment Analysis**

Enter your tweet here...

Submit

Reset

a) Software Requirements:

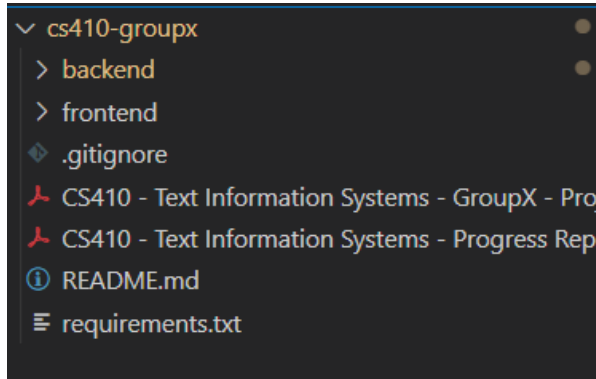
- Visual Studio Code (IDE)
- Python v3.10.6

Sentiment Analysis of Tweets

- iii. Node.js v19
- iv. Npm v8.19
- v. Data Source at [Kaggle](#) (expected format)

b) Code execution:

- i. Download the code from the [link](#). As shown below root directory has two folders “backend” and “frontend”



- ii. “backend” folder contains the code of sentiment analysis engine while “frontend” holds UI code
- iii. Install the required dependencies using “*pip install -r requirements.txt*” command. File “requirements.txt” lists the dependencies along with the version.
- iv. Go to the directory backend “*cd backend*”
- v. Run the modules of Natural Language Toolkit: “*python nltk_modules.py*”
- vi. Run the server using “*python server.py*”
- vii. Go back one level from the current “backend” directory using “*cd..*” and then change the current directory to “frontend”: “*cd frontend*”
- viii. Install npm: “*npm install*”
- ix. Run the app: “*npm start*”
- x. Fire up a browser and hit the url <http://localhost:3000>

3. How is Software implemented?

As already mentioned in the aforesaid section, Python has been chosen as the coding language in this project. The UI is developed in ReactJS.

Following are the steps for building the twitter sentiment analysis:

Sentiment Analysis of Tweets

a) Importing required Dependencies

Below are the screenshots of the different dependencies used in Twitter Sentiment implementation.

```
cs410-groupx > backend > data_preprocessing.py > ...  
  
1  # utilities  
2  import re  
3  import numpy as np  
4  import pandas as pd  
5  import string  
6  
7  # nltk  
8  import nltk  
9  from nltk.stem import WordNetLemmatizer  
10 from nltk.corpus import stopwords  
11 from nltk.tokenize import RegexpTokenizer  
12
```

```
cs410-groupx > backend > get_sentiments.py > ...  
  
1  import pickle  
2  import pandas as pd  
3  from data_preprocessing import *  
4
```

```
cs410-groupx > backend > get_tweets.py > ...  
  
1  import configparser  
2  import pandas as pd  
3  import tweepy  
4  from datetime import datetime
```

Sentiment Analysis of Tweets

```
cs410-groupx > backend > sentiment_analysis.py > ...

1  # utilities
2  import re
3  import numpy as np
4  import pandas as pd
5  import string
6
7  # nltk
8  import nltk
9  from nltk.stem import WordNetLemmatizer
10 from nltk.corpus import stopwords
11 from nltk.tokenize import RegexpTokenizer
12
13 # sklearn
14 from sklearn.svm import LinearSVC
15 from sklearn.naive_bayes import BernoulliNB
16 from sklearn.linear_model import LogisticRegression
17 from sklearn.model_selection import train_test_split
18 from sklearn.feature_extraction.text import TfidfVectorizer
19 from sklearn.metrics import confusion_matrix, classification_report
20
21 # Pickle
22 import pickle
```

```
cs410-groupx > backend > server.py > ...

1  from flask import Flask, jsonify, request
2  from flask_cors import CORS
3  from get_sentiments import *
4
```


Sentiment Analysis of Tweets

```
① README.md × train_and_create_model.py ×
cs410-groupx > backend > train_and_create_model.py > ...

1  # import data cleaning
2  from data_preprocessing import *
3
4  # sklearn
5  from sklearn.linear_model import LogisticRegression
6  from sklearn.model_selection import train_test_split
7  from sklearn.feature_extraction.text import TfidfVectorizer
8
9  # pickle
10 import pickle
```

b) Reading and Loading the Kaggle Dataset

The source data can be found at [kaggle](#):

This link describes context and content of data points, such as the polarity of the target (0 = negative, 2 = neutral, 4 = positive), date of the tweet, user, text of tweet etc. In our project we will be focusing only on *negative* and positive *polarities*.

```
① README.md train_and_create_model.py ×
cs410-groupx > backend > train_and_create_model.py > ...

13 DATASET_COLUMNS=['target','ids','date','flag','user','text']
14 DATASET_ENCODING = "ISO-8859-1"
15 df = pd.read_csv('data/training.1600000.processed.noemoticon.csv', encoding=DATASET_ENCODING, names=DATASET_COLUMNS)
16
```

c) Data Curation and Pre-processing

In this analysis the only columns chosen are “target” and “text”; the other columns like user, ids, date etc. are irrelevant in sentiment prediction therefore, these columns in the datasets are ignored.

In order to make things simple we replaced the target code of “positive” tweets from “4” to “1”. Negative tweets are still represented by 0.

Sentiment Analysis of Tweets

```
17 # Select required column from the dataset
18 data = df[['text', 'target']]
19 data['target'] = data['target'].replace(4,1)
```

Post this replacement the data is cleaned before training the model.

To ensure proper data processing below steps are followed:

- i. Cleaning the urls – used regular expression
- ii. Removing the punctuations – used translation table and str.translate
- iii. Remove repeating characters – used regular expression
- iv. Converted entire text in lower case
- v. Cleaned stopwords - used natural language toolkit (nltk)
- vi. Remove numbers – used regular expression
- vii. Tokenize each word in the remaining text – used RegexpTokenizer
- viii. Stemming – used natural language toolkit (nltk.WordNetLemmatizer)
- ix. Lemmatization - used natural language toolkit (nltk.WordNetLemmatize)

Sentiment Analysis of Tweets

```
cs410-groupx > backend > data_preprocessing.py > clean_data
12
13 STOPWORDS = set(stopwords.words('english'))
14 st = nltk.PorterStemmer()
15 lm = nltk.WordNetLemmatizer()
16
17 # Remove URLs
18 def cleaning_urls(data):
19     return re.sub('((www.[^s]+)|(https?://[^\s]+))', ' ', data)
20
21 # Remove punctuation
22 # english_punctuations = string.punctuation
23 # punctuations_list = english_punctuations
24
25 def cleaning_punctuations(text):
26     translator = str.maketrans('', '', string.punctuation)
27     return text.translate(translator)
28
29 # Remove repeating characters
30 def cleaning_repeating_char(text):
31     return re.sub(r'(.)1+', r'1', text)
32
33 # Remove stop words
34 def cleaning_stopwords(text):
35     return " ".join([word for word in str(text).split() if word not in STOPWORDS])
36
37 # Remove numbers
38 def cleaning_numbers(data):
39     return re.sub('[0-9]+', '', data)
40
41 # Stemming
42 def stemming_on_text(data):
43     text = [st.stem(word) for word in data]
44     return data
45
46 # Lemmatize
47 def lemmatizer_on_text(data):
48     text = [lm.lemmatize(word) for word in data]
49     return data
```

d) Extract tweets (features) and the corresponding targets (labels)

From the cleaned dataset, fetch the tweet and the corresponding polarity (target)

```
36 # Split dataset features and labels
37 X = dataset['text']
38 y = dataset['target']
```

e) Split dataset into train and test datasets

Next, we split the clean data into training and test set.

The data is split using the test size of 0.05 and a random state. X represents the “text” and Y represents the target (positive or negative). Therefore, we see X_Train, X_Test, Y_Train and Y_Test.

Test and train datasets are assigned rows randomly to ensure they are a random sample of the original dataset and represent observation samples. This is achieved by using the random state variable which is an integer value.

For this project, we have split the dataset so that 95% is used to train the model and 5% is used to evaluate the model. The split percentage was chosen arbitrarily.

```
41 # This is to check the model
42 X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.05, random_state =26105111)
```

f) Create, train, and transform the Model

Utilize Term Frequency - Inverse Document Frequency Vectorizer(TfidfVectorizer) to remold the dataset. This will not only consider the word occurrences in a single tweet but in the entire range of tweets. This method will assign lower weights to common words while giving higher weights to rare words but appearing multiple times across tweets. Used sklearn.feature_extraction.text.TfidfVectorizer to transform the X_Train and X_Test (“text”) into Vectorizer.

```
vectorizer = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
```

(1, 2) means unigrams and bigrams values that are used. Max_features allow to build vocabulary of top 500000 features ordered by term frequency across all tweets.

The vectorizer transforms text into vectors so that they can be used as input to the estimator.

Post pre-modelling, we are using Logistic Regression algorithm on the datasets to build the model that helps to classify the tweets as positive or negative. Logistic regression will predict the binary outcome as 0 – negative tweet and 1-positive tweet.

Sentiment Analysis of Tweets

This is done with Inverse of regularization strength of 2, max iterations of 1000 and then fit the vectorized X_Train against Y_Train. Using regularization strength of 2, we are applying penalty to increase parameter value magnitudes to reduce overfitting. The tf idf fit helps in normalizing the counts of occurrence of words across tweets. Both the outputs of the vectorizer and the train model is dumped into individual /pickle file. Here, pickle files allow us to keep track of tweets serialized so later references will not be serialized again thus achieving faster execution time.

```
cs410-groupx > backend > train_and_create_model.py > ...  
40  
41 # This is to check the model  
42 X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.05, random_state =26105111)  
43  
44 vectorizer = TfidfVectorizer(ngram_range=(1,2), max_features=500000)  
45  
46 vectorizer.fit(X_train, y_train)  
47 X_train_vectorized = vectorizer.transform(X_train)  
48 X_test_vectorized = vectorizer.transform(X_test)  
49  
50 lrModel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)  
51 lrModel.fit(X_train_vectorized, y_train)  
52  
53 vector_filename = 'textVetorizer.pickle'  
54 model_filename = 'lrModel_trained.pickle'  
55  
56 pickle.dump(vectorizer, open(vector_filename, 'wb'))  
57 pickle.dump(lrModel, open(model_filename, 'wb'))
```

g) Model Evaluation

Once model training is completed, the next step is to evaluate the performance of the model. A classification report is built for evaluation measures.

```
116 def model_Evaluate(model):
117     # Predict values for Test dataset
118     y_pred = model.predict(X_test_vectorized)
119     # Print the evaluation metrics for the dataset.
120     print(classification_report(y_test, y_pred))
121
122     # bnbmodel = BernoulliNB()
123     # bnbmodel.fit(X_train_vectorized, y_train)
124     # model_Evaluate(bnbmodel)
125     # y_pred1 = bnbmodel.predict(X_test_vectorized)
126
127     lrModel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
128     lrModel.fit(X_train_vectorized, y_train)
129
130     model_filename = 'lrModel_trained.sav'
131     pickle.dump(lrModel, open(model_filename, 'wb'))
132
133     sampleTweet = ["I like machine learning."]
134     inputData = pd.DataFrame([sampleTweet, columns=['text']])
```

The trained model is consequently used for making prediction of the input tweet.

```
prediction = model.predict(predictForVectorized)

print("Prediction for tweet '{}': {}".format(tweet, prediction))

return prediction
```

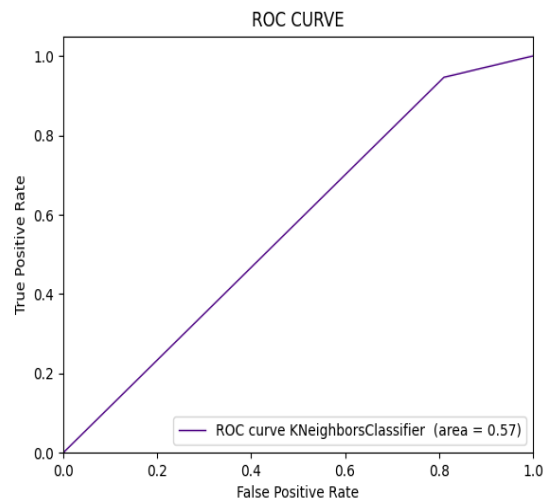
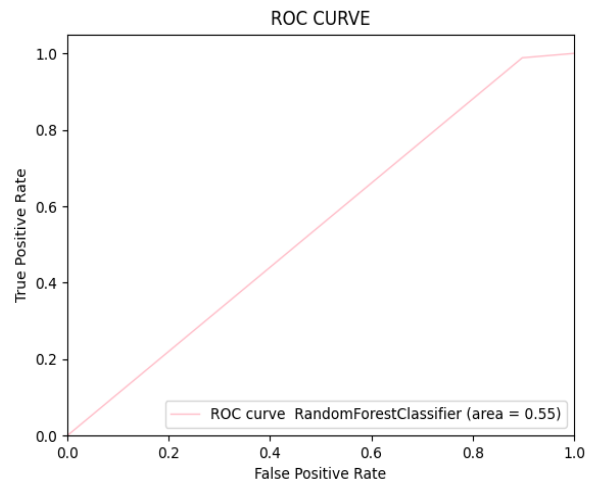
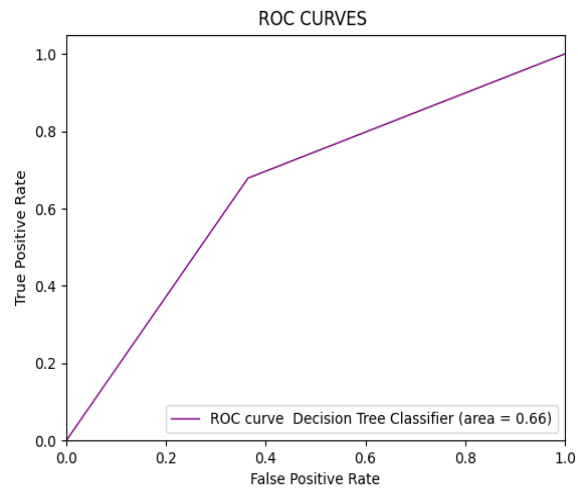
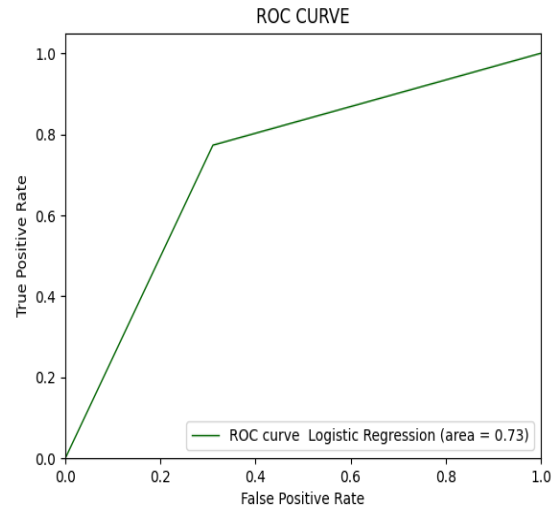
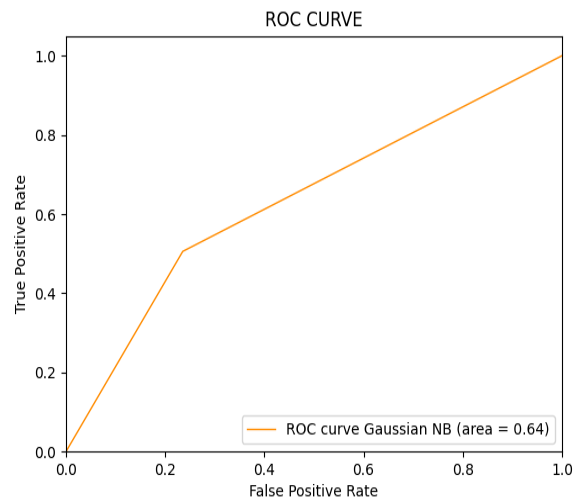
4. Model Performance Evaluation

To compare different models based on their performance, ROC curves for 5 different models were generated. Based on the ROC plots shown below, it was confirmed that for Twitter Sentiment Analysis, Logistic Regression model is the best suited model as compared to the other models.

Following were the models chosen for comparison:

- Gaussian Naïve Bayes
- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- K-Neighbors Classifier

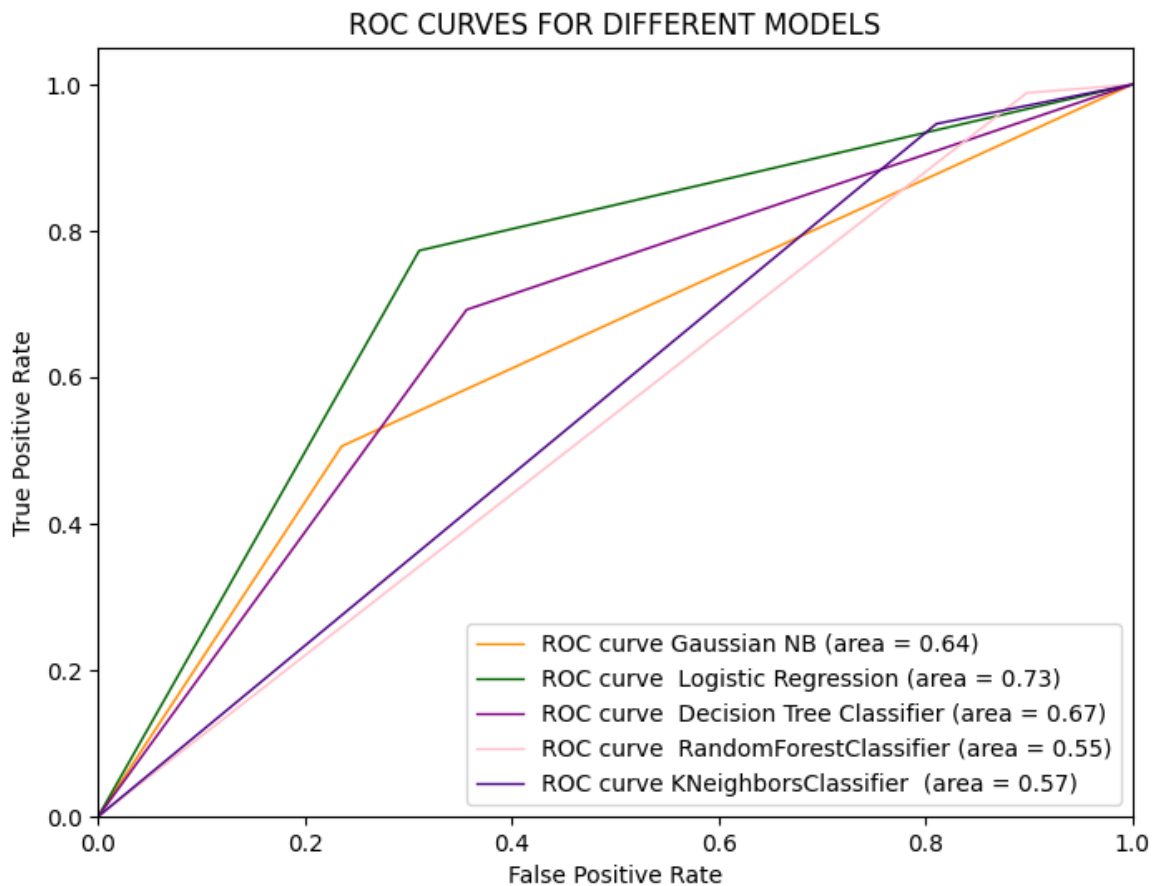
Sentiment Analysis of Tweets



Sentiment Analysis of Tweets

The order of ROC AUC Score for the models:

Logistic Regression (0.73) > Decision Tree Classifier (0.67) > Gaussian NB (0.64) > K-Neighbors Classifier (0.57) > Random Forest Classifier (0.55)



For this project, the Logistic Regression model has a chance of performing sentiment analysis with 73% chance.

Code Implementation: ModelComparison.py file

Sentiment Analysis of Tweets

```
ModelComparison.py U
cs410-groupx > backend > ModelComparison.py > ...
1 import pandas
2 import matplotlib.pyplot as plt
3 from data_preprocessing import *
4 from sklearn import model_selection
5 from sklearn.linear_model import LogisticRegression
6 from sklearn.model_selection import train_test_split
7 from sklearn.tree import DecisionTreeClassifier
8 from sklearn.naive_bayes import GaussianNB
9 from sklearn.metrics import roc_curve, auc
10 from sklearn.feature_extraction.text import TfidfVectorizer
11 from sklearn.ensemble import RandomForestClassifier
12 from sklearn.neighbors import KNeighborsClassifier
13
14 # we are only using a single dataset at this link so we can hardcode some stuff for training.
15 DATASET_COLUMNS=['target','ids','date','flag','user','text']
16 DATASET_ENCODING = "ISO-8859-1"
17 df = pd.read_csv('C:/Users/User/Documents/tis/FinalProject/cs410-groupx/backend/training.1600000.processed.noemoticon.csv', encoding=
18
19 # Select required column from the dataset
20 data = df[['text', 'target']]
21 data['target'] = data['target'].replace(4,1)
22
23
24 # Reduce dataset
25 data_pos = data[data['target'] == 1]
26 data_neg = data[data['target'] == 0]
27 data_pos = data_pos.iloc[:int(21000)]
28 data_neg = data_neg.iloc[:int(21000)]
29 dataset = pd.concat([data_pos, data_neg])
30
31 # The same function should be called for prediction when taking data from UI
32 dataset = clean_data(dataset)
33 # print(dataset['text'].head())
34
35 # Split dataset features and labels
36 X = dataset['text']
37 y = dataset['target']
38
39 #models
40 X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.05, random_state =26105111)
41 vectorizer = TfidfVectorizer(ngram_range=(1,2), max_features=10000)
42
43 vectorizer.fit(X_train, y_train)
44 X_train_vectorized = vectorizer.transform(X_train)
45 X_test_vectorized = vectorizer.transform(X_test)
46
47 #GaussianNB
48 GNB = GaussianNB()
49 GNB.fit(X_train_vectorized.toarray(),y_train)
50 y_pred1 =GNB.predict(X_test_vectorized.toarray())
51
52 #Logistic Regression
53 LR = LogisticRegression()
54 LR.fit(X_train_vectorized.toarray(), y_train)
55
56 y_pred2 = LR.predict(X_test_vectorized.toarray())
57
58 # Decision Tree Classifier
59 DTC = DecisionTreeClassifier()
60 DTC.fit(X_train_vectorized.toarray(), y_train)
```

Sentiment Analysis of Tweets

```
61 y_pred3 = DTC.predict(X_test_vectorized.toarray())
62
63 #Random Forest Classifier
64 RFC = RandomForestClassifier(random_state=42, n_jobs=-1, max_depth=5, n_estimators=100, oob_score=True)
65 RFC.fit(X_train_vectorized.toarray(), y_train)
66 y_pred4 = RFC.predict(X_test_vectorized.toarray())
67
68 #K Neighbors Classifier
69 KN = KNeighborsClassifier(n_neighbors =3)
70 KN.fit(X_train_vectorized.toarray(), y_train)
71 y_pred5 = KN.predict(X_test_vectorized)
72
73 fp1, tp1, thresholds1 = roc_curve(y_test, y_pred1)
74 roc_auc1= auc(fp1, tp1)
75 fp2, tp2, thresholds2 = roc_curve(y_test, y_pred2)
76 roc_auc2= auc(fp2, tp2)
77 fp3, tp3, thresholds3 = roc_curve(y_test, y_pred3)
78 roc_auc3= auc(fp3, tp3)
79 fp4, tp4, thresholds4 = roc_curve(y_test, y_pred4)
80 roc_auc4= auc(fp4, tp4)
81 fp5, tp5, thresholds5 = roc_curve(y_test, y_pred5)
82 roc_auc5= auc(fp5, tp5)
83
84 plt.figure()
85 plt.plot(fp1, tp1, color='darkorange', lw=1, label='ROC curve Gaussian NB (area = %0.2f)' % roc_auc1)
86 plt.plot(fp2, tp2, color='darkgreen', lw=1, label='ROC curve Logistic Regression (area = %0.2f)' % roc_auc2)
87 plt.plot(fp3, tp3, color='purple', lw=1, label='ROC curve Decision Tree Classifier (area = %0.2f)' % roc_auc3)
88 plt.plot(fp4, tp4, color='pink', lw=1, label='ROC curve RandomForestClassifier (area = %0.2f)' % roc_auc4)
89 plt.plot(fp5, tp5, color='indigo', lw=1, label='ROC curve KNeighborsClassifier (area = %0.2f)' % roc_auc5)
90 plt.xlim([0.0, 1.0])
91 plt.ylim([0.0, 1.05])
92
93 plt.xlabel('False Positive Rate')
94 plt.ylabel('True Positive Rate')
95 plt.title('ROC CURVES FOR DIFFERENT MODELS')
96 plt.legend(loc="lower right")
97 plt.show()
```

5. Description of Team Contribution

Team Member	Project Contribution
Chandan Goel	Project Proposal and Progress Report Documentation
Gopikrishnan Srinivasan	Initial data analysis, User Interface creation using ReactJS, Integration with the backend ML model using Flask (Python)
Hitesh Yadav	Data curation, machine learning modeling and prediction code
Sanjeev Singh	Presentation PowerPoint
Shubhendu Bhaskar	Initial Data Analysis, UI research and front end POC built in Dash, ROC-AUC models comparison programming and Final Project Report

6. Related Work

Different related work was referenced during the project.

Projects from our UIUC university:

https://mediaspace.illinois.edu/media/t/1_b5o4qhuo/112201961

NLP user case for beginners:

<https://www.analyticsvidhya.com/blog/2021/06/twitter-sentiment-analysis-a-nlp-use-case-for-beginners/>

7. References

https://scikit-learn.org/stable/model_persistence.html

<https://www.skillfinder.com.au/course/what-is-the-main-purpose-of-twitter>