

LP1 Assignment DA R4

Twitter Data Analysis

Date - 24th September, 2020.

Assignment Number - DA R4

Title

Twitter Data Analysis

Problem Definition

Use Twitter data for sentiment analysis. The dataset is 3MB in size and has 31,962 tweets. Identify the tweets which are hate tweets and which are not

Learning Objectives

- Learn Classification Algorithms
- Learn to summarize the properties in the training dataset.
- Learn to split the dataset into training and test datasets.
- Learn to develop a predictive classification model

Learning Outcomes

I will be able to develop a classification model for sentiment analysis of tweets on twitter.

Software Packages and Hardware Apparatus Used

- Operating System : 64-bit Ubuntu 18.04
- Programming Language : Python 3
- Jupyter Notebook Environment : Google Colaboratory
- Python Libraries : Kaggle, Kaggle CLI, Sklearn, Pandas, Matplotlib, NLTK, Keras, CLTK, Stanza (StanfordNLP)

Programmer's Perspective

Let S be the system set:

$S = \{s; e; X; Y; F_{me}; DD; NDD; F_c; S_c\}$
where Dataset is loaded into the dataframe

s=start state

e=end state
predicted state of tweets (0 - Not a Hate Tweet, 1 - Hate Tweet)

X=set of inputs
 $X = \{X_1\}$
where X_1 = Twitter Dataset (31962 records, 2 columns)

Y=set of outputs
 $Y = \{Y_1, Y_2, Y_3\}$

1. Y_1 = Predicted Values
2. Y_2 = Confusion Matrix
3. Y_3 = Accuracy Score

F_{me} is the set of main functions
 $F_e = \{f_0\}$

- f_0 = Main Display Function

F_f is the set of friend functions
 $F_f = \{f_1, f_2, f_3, f_4, f_5\}$ where

1. f_1 = function to download dataset using kaggle API
2. f_2 = function to load dataset into pandas dataframe
3. f_3 = function to clean tweets
4. f_4 = function to split dataset into test and train data
5. f_5 = function to train the model

DD = Deterministic Data
Twitter Dataset

NDD = Non-deterministic data (Eg - Null Values in Dataset)
No null value detected

F_c = failure case

Concepts related Theory

Sentiment Analysis

Sentiment analysis (also known as opinion mining or emotion AI) refers to the use of natural language processing, text analysis, computational linguistics, and biometrics to systematically identify, extract, quantify, and study affective states and subjective information. Sentiment analysis is widely applied to voice of the customer materials such as reviews and survey responses, online and social media, and healthcare materials for applications that range from marketing to customer service to clinical medicine.

Classification

In statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Examples are assigning a given email to the "spam" or "non-spam" class, and assigning a diagnosis to a given patient based on observed characteristics of the patient (sex, blood pressure, presence or absence of certain symptoms, etc.). Classification is an example of pattern recognition. In the terminology of machine learning, classification is considered an instance of supervised learning, i.e., learning where a training set of correctly identified observations is available.

Support Vector Machines

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

Computing the (soft-margin) SVM classifier amounts to minimizing an expression of the form

$$\left[\frac{1}{n} \sum_{i=1}^n \max(0, 1 - y_i(w \cdot x_i - b)) \right] + \lambda \|w\|^2. \quad (2)$$

LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or IDSs (intrusion detection systems).

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

The compact forms of the equations for the forward pass of an LSTM unit with a forget gate are:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ \sigma_h(c_t)$$

Natural Language Processing

Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural-language generation.

Tokenization

Tokenization is the process of demarcating and possibly classifying sections of a string of input characters. The resulting tokens are then passed on to some other form of processing. The process can be considered a sub-task of parsing input.

Stemming

In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word

Lemmatization

In computational linguistics, lemmatization is the algorithmic process of determining the lemma of a word based on its intended meaning. Unlike stemming, lemmatization depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document. As a result, developing efficient lemmatization algorithms is an open area of research.

Stopwords

Stopwords are the words in any language which does not add much meaning to a sentence. They can safely be ignored without sacrificing the meaning of the sentence. For some search engines, these are some of the most common, short function words, such as the, is, at, which, and on.

TFIDF

In information retrieval, tf-idf or TFIDF, short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in searches of information retrieval, text mining, and user modeling. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general. tf-idf is one of the most popular term-weighting schemes today. A survey conducted in 2015 showed that 83% of text-based recommender systems in digital libraries use tf-idf.

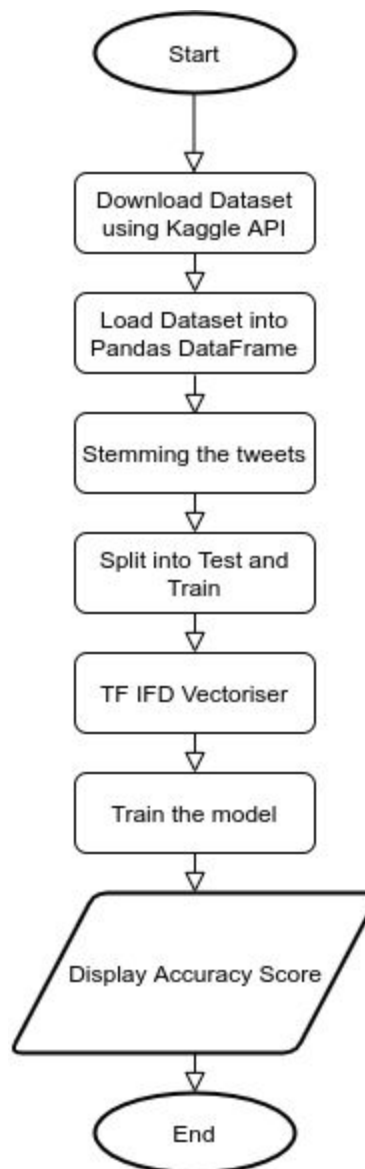
NLTK

The Natural Language Toolkit, or more commonly NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) for English written in the Python programming language.

Stanza

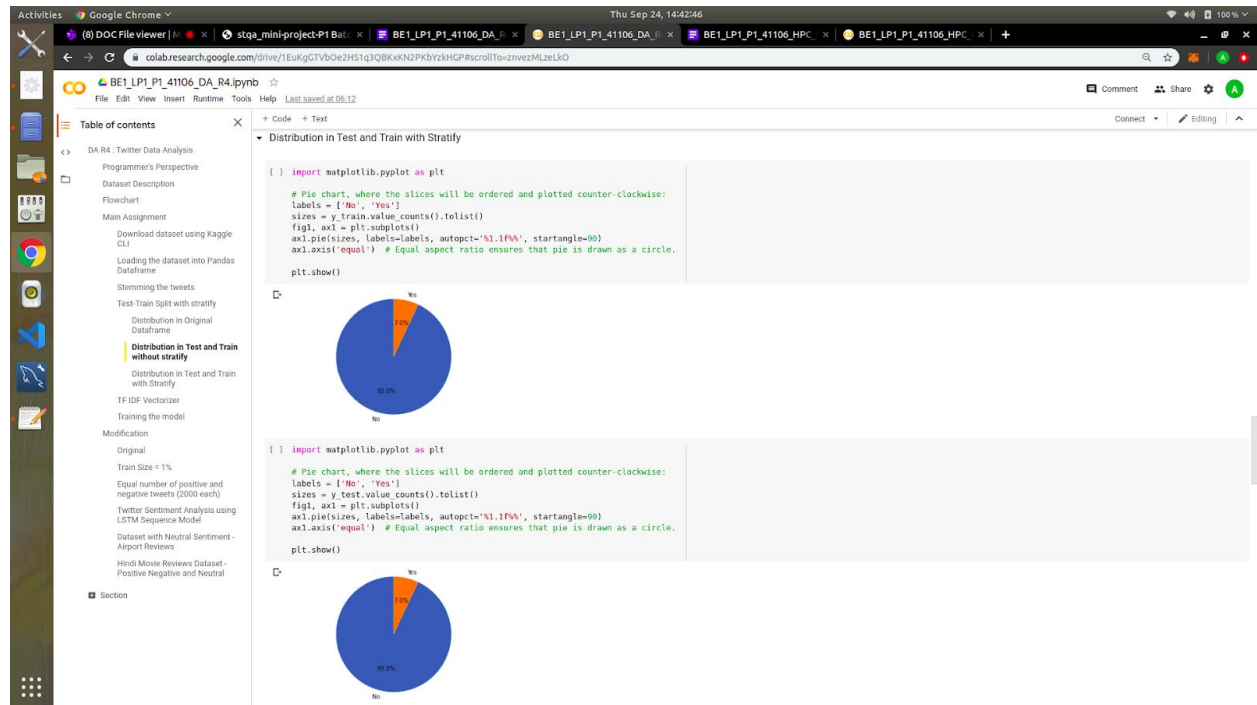
Stanza is a Python NLP Package for Many Human Languages developed by Stanford University.

Flowchart

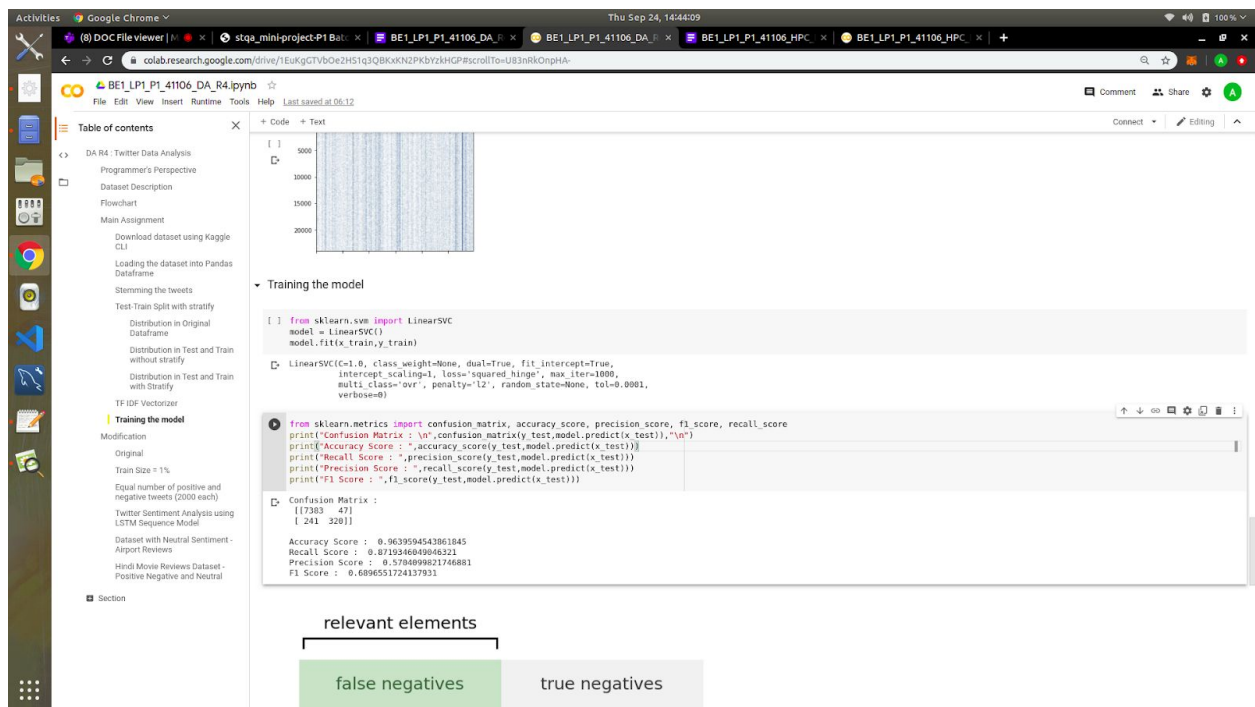


Output Screenshots

Distribution of Tweets in Test and Train Dataset



LinearSVC on preprocessed data



The screenshot displays a Google Colab notebook titled "BE1_LP1_P1_41106_DA.ipynb". The notebook is open to a section titled "Training the model". The code cell shows the following steps:

```
[ ] from sklearn.svm import LinearSVC
model = LinearSVC()
model.fit(x_train, y_train)

[ ] LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
              intercept_scaling=1, loss='squared_hinge', max_iter=1000,
              multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
              verbose=0)

[ ] from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, f1_score, recall_score
print("Confusion Matrix : \n", confusion_matrix(y_test, model.predict(x_test)), "\n")
print("Accuracy Score : ", accuracy_score(y_test, model.predict(x_test)))
print("Recall Score : ", precision_score(y_test, model.predict(x_test)))
print("Precision Score : ", recall_score(y_test, model.predict(x_test)))
print("F1 Score : ", f1_score(y_test, model.predict(x_test)))
```

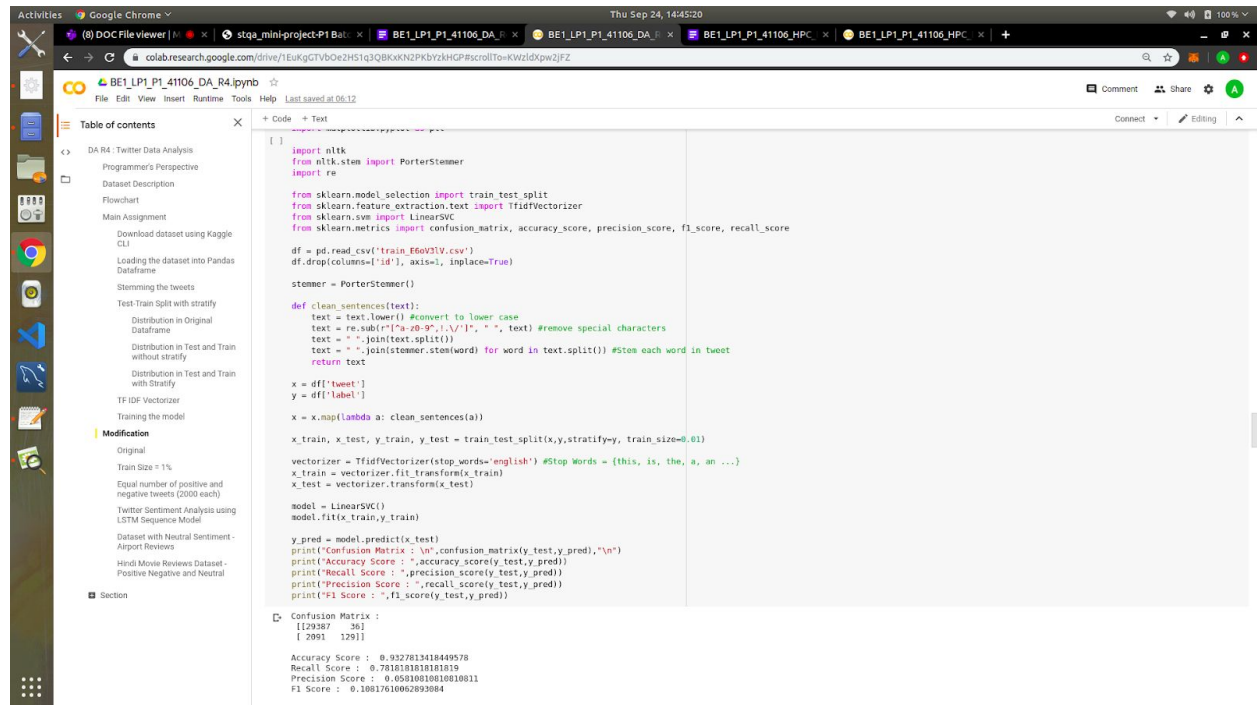
The output of the code cell shows the following results:

```
Confusion Matrix :
[[ 7383   47]
 [  241  328]]

Accuracy Score : 0.963504543861845
Recall Score : 0.871246689896321
Precision Score : 0.5704899821746881
F1 Score : 0.6896551724137931
```

Below the code cell, there is a diagram illustrating the components of the confusion matrix. A box labeled "relevant elements" contains two sub-boxes: "false negatives" and "true negatives".

Modification 1 - Train Size = 1%



```
[ ]
import nltk
from nltk.stem import PorterStemmer
import re

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, f1_score, recall_score

df = pd.read_csv('train_E6oV3V.csv')
df.drop(columns=['id'], axis=1, inplace=True)

stemmer = PorterStemmer()

def clean_sentences(text):
    text = text.lower() #convert to lower case
    text = re.sub(r'[a-z0-9!./\']+', ' ', text) #remove special characters
    text = " ".join(text.split())
    text = " ".join(stemmer.stem(word) for word in text.split()) #Stem each word in tweet
    return text

x = df['tweet']
y = df['label']

x = x.map(lambda a: clean_sentences(a))

x_train, x_test, y_train, y_test = train_test_split(x, y, stratify=y, train_size=0.01)

vectorizer = TfidfVectorizer(stop_words='english') #Stop Words = {this, is, the, a, an ...}
x_train = vectorizer.fit_transform(x_train)
x_test = vectorizer.transform(x_test)

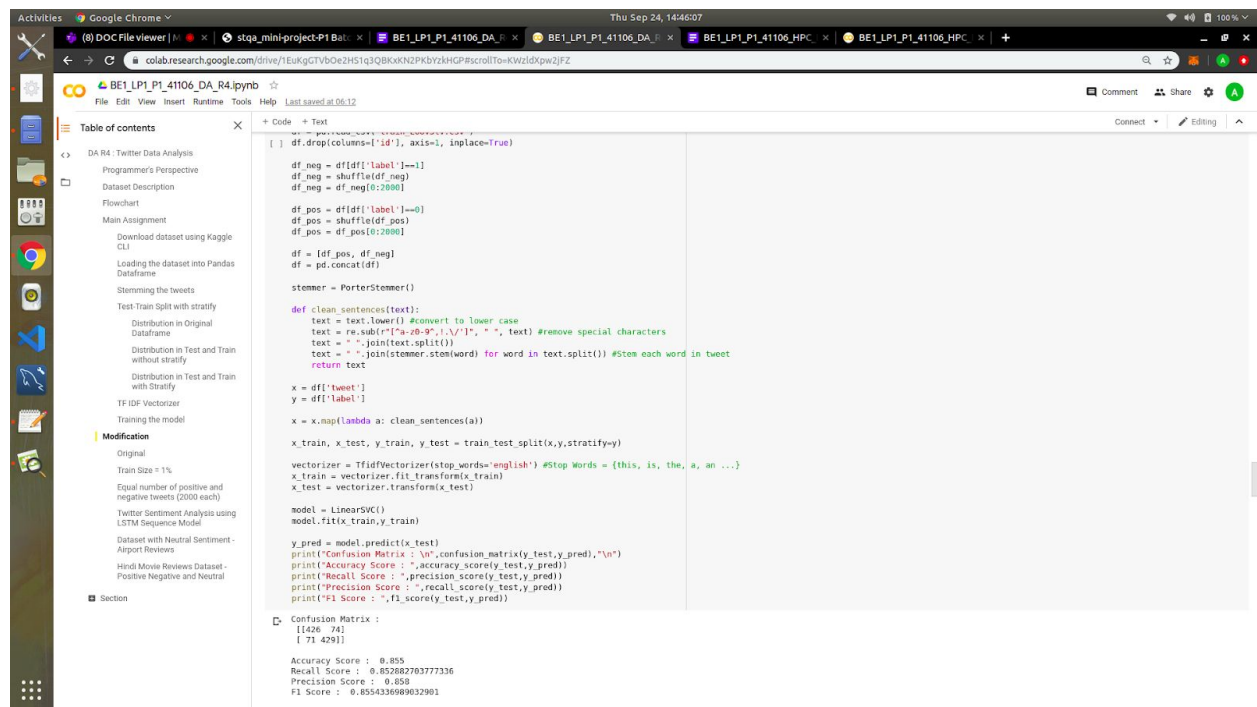
model = LinearSVC()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)
print("Confusion Matrix : \n", confusion_matrix(y_test, y_pred), "\n")
print("Accuracy Score : ", accuracy_score(y_test, y_pred))
print("Recall Score : ", precision_score(y_test, y_pred))
print("Precision Score : ", recall_score(y_test, y_pred))
print("F1 Score : ", f1_score(y_test, y_pred))

Confusion Matrix :
[[29367  361]
 [ 2991 129]]

Accuracy Score : 0.9327813418449578
Recall Score : 0.7818181818181819
Precision Score : 0.6581081081081081
F1 Score : 0.718610862993884
```

Modification 2 - Equal Number of Positive and Negative Tweets



```
[ ] df.drop(columns=['id'], axis=1, inplace=True)

df_neg = df[df['label']==-1]
df_pos = df[df['label']==0]

df_neg = df_neg.sample(n=2000)
df_pos = df_pos.sample(n=2000)

df = [df_pos, df_neg]
df = pd.concat(df)

stemmer = PorterStemmer()

def clean_sentences(text):
    text = text.lower() #convert to lower case
    text = re.sub(r'([a-z0-9]+/[^\s]+)', ' ', text) #remove special characters
    text = ' '.join(text.split())
    text = ' '.join(stemmer.stem(word) for word in text.split()) #stem each word in tweet
    return text

x = df['tweet']
y = df['label']

x = x.map(lambda a: clean_sentences(a))

x_train, x_test, y_train, y_test = train_test_split(x, y, stratify=y)

vectorizer = TfidfVectorizer(stop_words='english') #Stop Words = {this, is, the, a, an ...}
x_train = vectorizer.fit_transform(x_train)
x_test = vectorizer.transform(x_test)

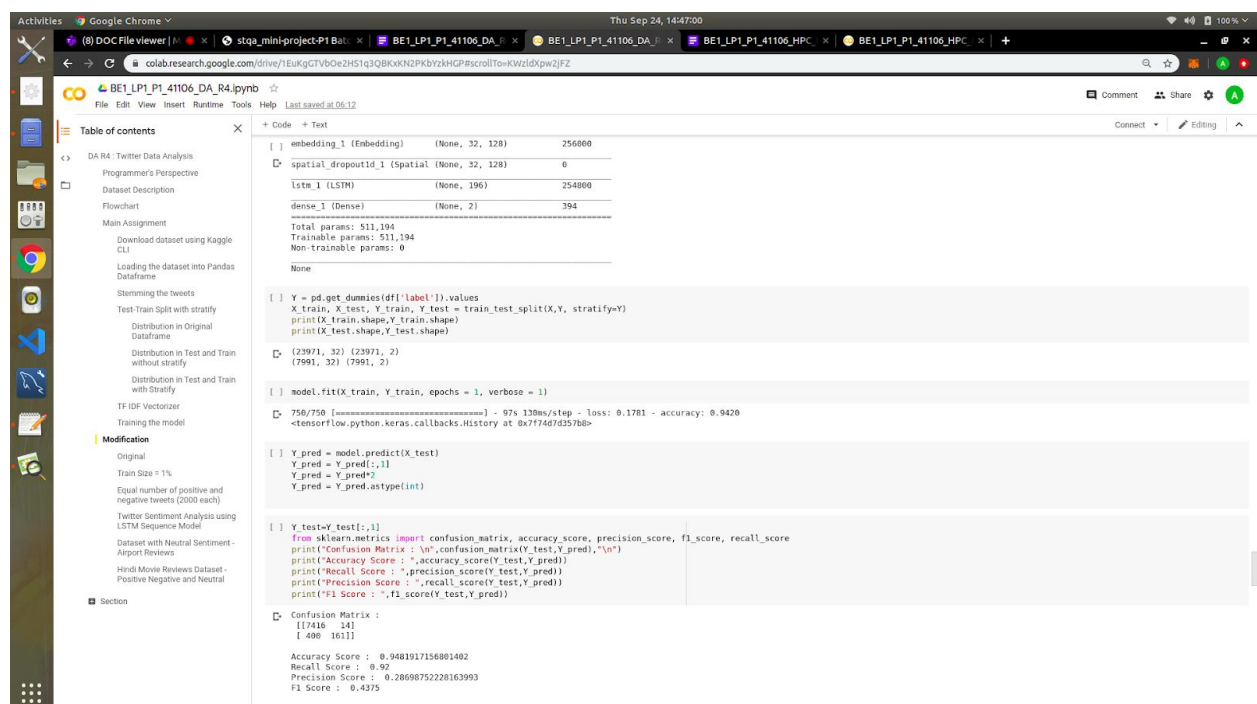
model = LinearSVC()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)
print("Confusion Matrix : \n", confusion_matrix(y_test, y_pred), "\n")
print("Accuracy Score : ", accuracy_score(y_test, y_pred))
print("Recall Score : ", recall_score(y_test, y_pred))
print("Precision Score : ", precision_score(y_test, y_pred))
print("F1 Score : ", f1_score(y_test, y_pred))

Confusion Matrix :
[[426  74]
 [ 71 429]]

Accuracy Score : 0.855
Recall Score : 0.8528270377336
Precision Score : 0.850
F1 Score : 0.854333089032901
```

Modification 3 - Modification using LSTM (Sequence Model)



```
[ ] embedding_1 (Embedding) (None, 32, 128) 256000
[ ] spatial_dropout1d_1 (Spatial) (None, 32, 128) 0
[ ] lstm_1 (LSTM) (None, 196) 254800
[ ] dense_1 (Dense) (None, 2) 394
-----
Total params: 511,194
Trainable params: 511,194
Non-trainable params: 0
None

[ ] Y = pd.get_dummies(df['label']).values
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, stratify=Y)
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)

[ ] model.fit(X_train, Y_train, epochs = 1, verbose = 1)

[ ] Y_pred = model.predict(X_test)
Y_pred = Y_pred[:,1]
Y_pred = Y_pred*2
Y_pred = Y_pred.astype(int)

[ ] Y_test=Y_test[:,1]
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, f1_score, recall_score
print("Confusion Matrix : \n", confusion_matrix(Y_test, Y_pred), "\n")
print("Accuracy Score : ", accuracy_score(Y_test, Y_pred))
print("Recall Score : ", recall_score(Y_test, Y_pred))
print("Precision Score : ", precision_score(Y_test, Y_pred))
print("F1 Score : ", f1_score(Y_test, Y_pred))

Confusion Matrix :
[[1416  14]
 [ 400 161]]

Accuracy Score : 0.9481917156801402
Recall Score : 0.92
Precision Score : 0.28698752220163993
F1 Score : 0.4375
```

Modification 4 - Dataset with Neutral Sentiment

The screenshot shows a Google Colab environment with a Jupyter Notebook open. The notebook has several tabs at the top, including 'DOC File viewer', 'colab.research.google.com/drive/EuqGTVbDeH5tG3QBkxNzPKvbyr4MGp#scrollTo=EDGDCvxZom1', and three tabs for 'BE1_LP1_41106_DA...'. The active tab is 'BE1_LP1_41106_DA_R4.ipynb', which was last saved at 06:12.

The left sidebar contains a 'Table of contents' panel with sections like 'DA R4 : Twitter Data Analysis', 'Programmer's Perspective', 'Dataset Description', 'Flowchart', and 'Main Assignment'. Under 'Main Assignment', there are links to 'Download dataset using Kaggle CLI', 'Loading the dataset into Pandas DataFrame', 'Stemming the tweets', 'Test Train Split with stratify', 'Distribution in Original DataFrame', 'Distribution in Test and Train without stratify', 'Distribution in Test and Train with Stratify', 'TF-IDF Vectorizer', and 'Training the model'. Below this is a 'Modification' section with 'Original', 'Train Size = 1%', 'Equal number of positive and negative tweets (2000 each)', 'Twitter Sentiment Analysis using LSTM Sequence Model', and 'Dataset with Neutral Sentiment - Airport Reviews'.

The main area of the notebook displays the following Python code:

```
x = df['text']  
y = df['airline_sentiment']  
  
stemmer = PorterStemmer()  
  
def clean_sentences(text):  
    text = text.lower() #convert to lower case  
    text = re.sub(r"[^a-z0-9,\./]", " ", text) #remove special characters  
    text = " ".join(text.split())  
    text = " ".join(stemmer.stemword for word in text.split()) #Stem each word in tweet  
    return text  
  
x = x.map(lambda a: clean_sentences(a))  
  
le = LabelEncoder()  
y = le.fit_transform(y)  
  
X_train, x_test, y_train, y_test = train_test_split(x,y,stratify=y)  
  
vectorizer = TfidfVectorizer(stop_words='english') #Stop Words = {this, is, the, a, an ...}  
x_train = vectorizer.fit_transform(x_train)  
x_test = vectorizer.transform(x_test)  
  
model = LinearSVC()  
model.fit(x_train,y_train)  
  
y_pred = model.predict(x_test)  
print("Confusion Matrix : \n",confusion_matrix(y_test,y_pred),"\n")  
print("Accuracy Score : ",accuracy_score(y_test,y_pred))  
print("F1 Score : ",f1_score(y_test,y_pred,average='weighted'))
```

The output of the code is shown below the code cells:

```
Building wheel for kaggle-cli (setup.py) ... done  
Building wheel for pyperclip (setup.py) ... done  
downloading twitter-airline-sentiment.zip to /content  
100% 2.55M/2.55M [08:00<00:00, 75.7kB/s]  
Inflating: Tweets.csv  
Inflating: database.sqlite  
Confusion Matrix :  
[[2078 152 64]  
 [282 415 78]  
 [109 75 407]]  
  
Accuracy Score : 0.7923497267759563  
F1 Score : 0.786228909396048
```

Modification 5 - Hindi Dataset

Google Chrome

Thur Sep 24, 14:49:17

(8) DOC File viewer | stqp_mini-project-P1 Ba... | BE1_LP1_P1_41106_DA | BE1_LP1_P1_41106_DA | BE1_LP1_P1_41106_HPC | BE1_LP1_P1_41106_HPC | +

colab.research.google.com/drive/1EukGCTVbOe2H5tq3Q8KxN2PYbYwHCPscrollTo=1wF9EwCOW

BE1_LP1_P1_41106_DA_R4.ipynb

File Edit View Insert Runtime Tools Help

Table of contents

DA_R4: Twitter Data Analysis

Programmer's Perspective

Dataset Description

Flowchart

Main Assignment

Download dataset using Kaggle CLI

Loading the dataset into Pandas Dataframe

Stemming the tweets

Test Train Split with stratify

Distribution in Original Dataframe

Distribution in Text and Train without stratify

Distribution in Text and Train with Stratify

TF-IDF Vectorizer

Training the model

Modification

Original

Train Size = 1%

Equal number of positive and negative tweets (2000 each)

Twitter Sentiment Analysis using LSTM Sequence Model

Dataset with Neutral Sentiment - Airport Reviews

Hindi Movie Reviews Dataset - Positive Negative and Neutral

Section

Code

```

hindi_text = re.sub('[^\w\s]','',hindi_text)
hindi_text = " ".join(hindi_text.split())

hi_doc = hi_stanza(hindi_text)

text_lemma = []
for sentence in hi_doc.sentences:
    for word in sentence.words:
        text_lemma.append(word.lemma)
text_lemma = " ".join(word for word in text_lemma)
return text_lemma

X_train_cleaned = X_train.map(lambda x: clean_review(x))
X_test_cleaned = X_test.map(lambda x: clean_review(x))

X_train_cleaned.to_csv('X_train_cleaned.csv', index=False)
X_test_cleaned.to_csv('X_test_cleaned.csv', index=False)

!ncapture
pip install nltk
from nltk.stop.classical_hindi_stops import STOPS_LIST as stop_words_hindi
stop_words_hindi

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(stop_words=stop_words_hindi) #stop Words = (this, is, the, a, an ...)
X_train_Vectorized = vectorizer.fit_transform(X_train_cleaned)
X_test_Vectorized = vectorizer.transform(X_test_cleaned)

from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score

model = LinearSVC()
model.fit(X_train_Vectorized, Y_train)

Y_pred = model.predict(X_test_Vectorized)
print("Confusion Matrix : \n", confusion_matrix(Y_test, Y_pred), "\n")
print("Accuracy Score : ", accuracy_score(Y_test, Y_pred))
print("F1 Score : ", f1_score(Y_test, Y_pred, average="weighted"))

Confusion Matrix :
[[27 14 12]
 [26 27 18]
 [13 12 37]]

Accuracy Score : 0.5055555555555555
F1 Score : 0.5035504070502846

```

Conclusion

I have successfully developed a classification model for sentiment analysis of Twitter dataset.