**SENTIMENT ANALYSIS FOR MARKETING**

**PROJECT PHASE 4**

**OBJECTIVE:**

To implement the techniques learnt as a part of the course.

**DATA DESCRIPTION:**

A sentiment analysis job about the problems of each major U.S. airline. Twitter data was scraped from February of 2015 and contributors were asked to first classify positive, negative, and neutral tweets, followed by categorizing negative reasons (such as "late flight" or "rude service").

**DATA SET:**

Link to the Kaggle project site: <https://www.kaggle.com/crowdflower/twitter-airline-sentiment>

**STEP – 1:**

**Import the libraries, load dataset, print shape of data, data description:**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

%matplotlib inline

import re, string, unicodedata

from bs4 import BeautifulSoup

!pip install contractions

import nltk

import contractions

nltk.download('wordnet')

nltk.download('punkt')

from nltk.tokenize import word\_tokenize

from nltk.stem import LancasterStemmer, WordNetLemmatizer

from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer

from sklearn import metrics

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import accuracy\_score,confusion\_matrix,classification\_report

from mlxtend.plotting import plot\_confusion\_matrix

from imblearn.over\_sampling import SMOTE

*#from google.colab import drive*

*#drive.mount('/content/drive/')*

import os

for dirname, \_, filenames **in** os.walk('/kaggle/input'):

for filename **in** filenames:

print(os.path.join(dirname, filename))

**LOAD DATASET:**

*#Tweet= pandas.read\_csv("../input/Tweets.csv")*

*#Tweet.head()*

project\_path = '/kaggle/input/twitter-airline-sentiment/'

*# Load the dataset*

tweet\_data = pd.read\_csv(project\_path + 'Tweets.csv',header=0)

tweet\_data.head()

**SHAPE OF DATA:**

*# There are 14640 rows and 15 columns in the tweet data*

Print(tweet\_data.shape)

(14640, 15)

tweet\_data.info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 14640 entries, 0 to 14639

Data columns (total 15 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 tweet\_id 14640 non-null int64

1 airline\_sentiment 14640 non-null object

2 airline\_sentiment\_confidence 14640 non-null float64

3 negativereason 9178 non-null object

4 negativereason\_confidence 10522 non-null float64

5 airline 14640 non-null object

6 airline\_sentiment\_gold 40 non-null object

7 name 14640 non-null object

8 negativereason\_gold 32 non-null object

9 retweet\_count 14640 non-null int64

10 text 14640 non-null object

11 tweet\_coord 1019 non-null object

12 tweet\_created 14640 non-null object

13 tweet\_location 9907 non-null object

14 user\_timezone 9820 non-null object

dtypes: float64(2), int64(2), object(11)

memory usage: 1.7+ MB

tweet\_data.dtypes

tweet\_id int64

airline\_sentiment object

Tweet\_location object

user\_timezone object

dtype: object

**EXPLOROTARY DATA ANALYSIS EDA:**

**Sentiment Analysis for each Airline**

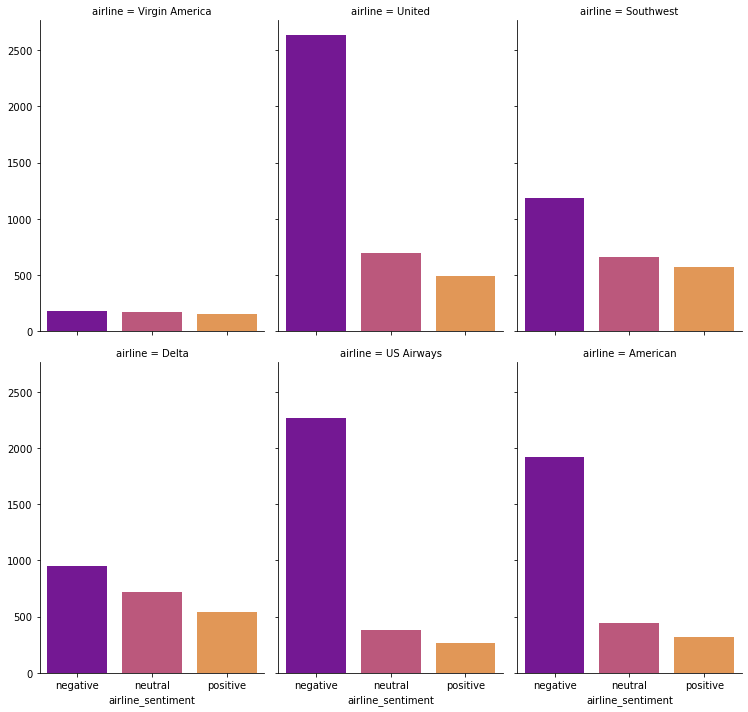
g = sns.FacetGrid(tweet\_data, col="airline", col\_wrap=3, height=5, aspect =0.7)

g = g.map(sns.countplot, "airline\_sentiment",order =tweet\_data.airline\_sentiment.value\_counts().index, palette='plasma')

plt.show()

*# Here we can see that United Airlines, US Airways, American Airlines has the most number of negative review*

*# Virgin America has the least number of negative reviews*

****

**Most Common negative review reasons**

*# Check the most common negative reason*

y = tweet\_data['negativereason']

print(y.value\_counts())

plt.figure(figsize=(25,5))

g = sns.countplot(y)

*# Customer service and late flight seems to be the main reason why customers are giving bad feedback*

Customer Service Issue 2898

Late Flight 1655

Can't Tell 1190

Cancelled Flight 839

Lost Luggage 718

Bad Flight 580

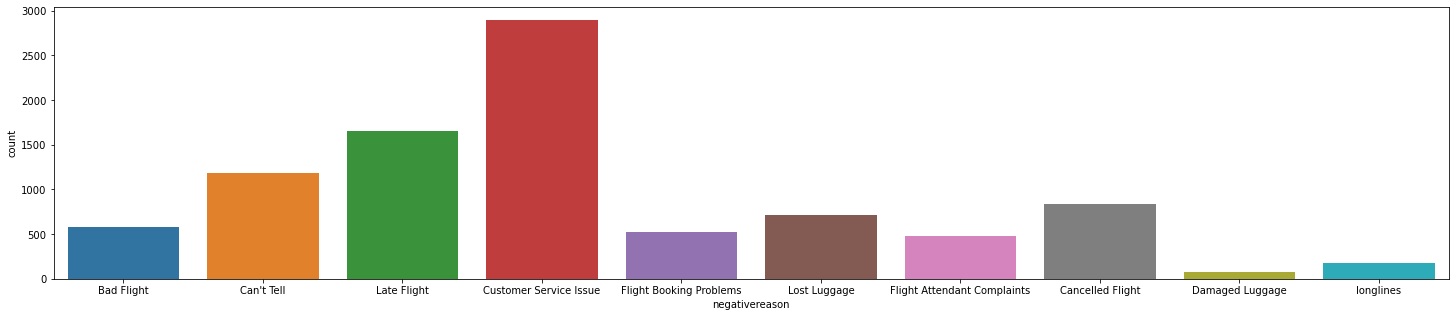
Flight Booking Problems 529

Flight Attendant Complaints 479

longlines 178

Damaged Luggage 74

Name: negativereason, dtype: int64



**STEP – 2:**

**Understand of data-columns:**

a. Drop all other colmns except “text” and “airline\_sentiment”.

b. Check the shape of data.

c. Print first 5 rows of data.

tweet\_data.columns

Index(['tweet\_id', 'airline\_sentiment', 'airline\_sentiment\_confidence',

'negativereason', 'negativereason\_confidence', 'airline',

'airline\_sentiment\_gold', 'name', 'negativereason\_gold',

'retweet\_count', 'text', 'tweet\_coord', 'tweet\_created',

'tweet\_location', 'user\_timezone'],

dtype='object')

**Drop Irrelevant columns**

*# Let us now remove irrelevant columns*

tweet\_data\_relevant = tweet\_data.drop(['tweet\_id', 'airline\_sentiment\_confidence',

'negativereason', 'negativereason\_confidence', 'airline',

'airline\_sentiment\_gold', 'name', 'negativereason\_gold',

'retweet\_count','tweet\_coord', 'tweet\_created',

'tweet\_location', 'user\_timezone'], axis =1)

**Shape of Data**

*# There are 14568 rows and 2 columns (This is result of keeping relevant rows and duplicate data cleanup)*

tweet\_data\_relevant.shape

(14568, 2)

**Printing first 5 rows of data**

tweet\_data\_relevant.head(5)

print(tweet\_data\_relevant.airline\_sentiment.value\_counts())

negative 9140

neutral 3083

positive 2345

Name: airline\_sentiment, dtype: int64

**Is the data balanced**

y = tweet\_data\_relevant['airline\_sentiment']

print(y.value\_counts())

plt.figure(figsize=(20,5))

g = sns.countplot(y)

*# No, Here we can see that the data is not balanced, There are lot of negative sentiments*

negative 9140

neutral 3083

positive 2345

Name: airline\_sentiment, dtype: int64



**STEP 3:**

**Text pre-processing: Data preparation.**

**Html tag removal:**

def perform\_html\_cleanup( raw\_review ):

*# 1. Remove HTML*

review\_text = BeautifulSoup(raw\_review).get\_text()

return review\_text

**Replace Contraction**

def replace\_contractions(raw\_review):

*#Replace contractions in raw\_review*

return contractions.fix(raw\_review)

**Tokenization**

def perform\_tokenization( raw\_review ):

*# 2. Perform Tokenization*

word\_tokens = word\_tokenize(raw\_review) *# Tokenization*

return word\_tokens

**Removal of Numbers**

def remove\_numbers(list\_of\_words):

pattern = '[0-9]'

list = [re.sub(pattern, '', i) for i **in** list\_of\_words]

return list

**Remove special characters and punctuations**

def remove\_special\_character\_punctuation(list\_of\_words):

pattern = '[^A-Za-z0-9]+'

list = [re.sub(pattern, '', i) for i **in** list\_of\_words]

return list

def remove\_punctuation(words):

*"""Remove punctuation from list of tokenized words"""*

new\_words = [] *# Create empty list to store pre-processed words.*

for word **in** words:

new\_word = re.sub(r'[^\w\s]', '', word)

if new\_word != '':

new\_words.append(new\_word) *# Append processed words to new list.*

return new\_words

**Conversion to Lower case**

def to\_lowercase(words):

*"""Convert all characters to lowercase from list of tokenized words"""*

new\_words = [] *# Create empty list to store pre-processed words.*

for word **in** words:

new\_word = word.lower() *# Converting to lowercase*

new\_words.append(new\_word) *# Append processed words to new list.*

return new\_words

**Remove empty String**

def remove\_empty\_string(words):

return list(filter(None, words))

**Stemming**

def stem\_words(words):

*"""Stem words in list of tokenized words"""*

stemmer = LancasterStemmer()

stems = [] *# Create empty list to store pre-processed words.*

for word **in** words:

stem = stemmer.stem(word)

stems.append(stem) *# Append processed words to new list.*

return stems

**Lemmatization**

def lemmatize\_verbs(words):

*"""Lemmatize verbs in list of tokenized words"""*

lemmatizer = WordNetLemmatizer()

lemmas = [] *# Create empty list to store pre-processed words.*

for word **in** words:

lemma = lemmatizer.lemmatize(word, pos='v')

lemmas.append(lemma) *# Append processed words to new list.*

return lemmas

**Complete Pre-preocessing**

def perform\_cleanup(raw\_review):

clean\_review = perform\_html\_cleanup(raw\_review)

clean\_review = replace\_contractions(clean\_review)

clean\_review = perform\_tokenization(clean\_review)

clean\_review = remove\_numbers(clean\_review)

clean\_review = remove\_special\_character\_punctuation(clean\_review)

clean\_review = remove\_punctuation(clean\_review)

clean\_review = to\_lowercase(clean\_review)

clean\_review = remove\_empty\_string(clean\_review)

*#clean\_review = stem\_words(clean\_review)*

clean\_review = lemmatize\_verbs(clean\_review)

return clean\_review

print(tweet\_data\_relevant.head())

clean\_reviews = []

for i, row **in** tweet\_data\_relevant.iterrows():

words = tweet\_data\_relevant.at[i, 'text']

words = perform\_cleanup(words)

tweet\_data\_relevant.at[i,'text'] = " ".join( words )

clean\_reviews.append( tweet\_data\_relevant.at[i, 'text'] )

tweet\_data\_relevant.head()

**Print first 5 rows of data after pre-processing**

tweet\_data\_relevant.head(5)

**STEP – 4**

**Vectorization: )**

a. Use CountVectorizer.

b. Use TfidfVectorizer

**CountVectorizer**

print ("Creating the bag of words...**\n**")

*# Initialize the "CountVectorizer" object, which is scikit-learn's*

*# bag of words tool.*

count\_vectorizer = CountVectorizer(analyzer = "word", \

tokenizer = None, \

preprocessor = None, \

stop\_words = None, \

max\_features = 5000)

*# fit\_transform() does two functions: First, it fits the model*

*# and learns the vocabulary; second, it transforms our training data*

*# into feature vectors. The input to fit\_transform should be a list of*

*# strings.*

count\_vectorizer\_data\_features = count\_vectorizer.fit\_transform(clean\_reviews)

*# Numpy arrays are easy to work with, so convert the result to an*

*# array*

count\_vectorizer\_data\_features = count\_vectorizer\_data\_features.toarray()

print (count\_vectorizer\_data\_features.shape)

print(count\_vectorizer\_data\_features)

(14568, 5000)

[[0 0 0 ... 0 0 0]

[0 0 0 ... 0 0 0]

[0 0 0 ... 0 0 0]

...

[0 0 0 ... 0 0 0]

[0 0 0 ... 0 0 0]

[0 0 0 ... 0 0 0]]

**Stop words**

count\_vectorizer\_stop\_words = count\_vectorizer.get\_stop\_words()

print (count\_vectorizer\_stop\_words)

*# There are no stop words since we are doing sentiment analysis*

**Sum up the counts of each vocabulary word**

*# Sum up the counts of each vocabulary word*

dist = np.sum(count\_vectorizer\_data\_features, axis=0)

*# For each, print the vocabulary word and the number of times it*

*# appears in the training set*

for tag, count **in** zip(count\_vectorizer\_vocab, dist):

print (count, tag)

**STEP - 5**

**Fit and evaluate model using both type of vectorization.**

**RandomForest Classifier on CountVectorizer**

tweet\_data\_relevant.head()

y = tweet\_data\_relevant['airline\_sentiment']

print(y.value\_counts())

plt.figure(figsize=(20,5))

g = sns.countplot(y)

*# No, Here we can see that the data is not balanced, There are lot of negative sentiments*

negative 9140

neutral 3083

positive 2345

Name: airline\_sentiment, dtype: int64



**Dividing Data to Train and Test**

x = count\_vectorizer\_data\_features *# Predictor feature columns*

y = tweet\_data\_relevant['airline\_sentiment'] *# Predicted class*

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=1) *# 1 is just any random seed number*

print(x\_train.shape)

print(y\_train.shape)

print(x\_test.shape)

print(y\_test.shape)

(10197, 5000)

(10197,)

(4371, 5000)

(4371,)

**Applying SMOTE since the data is not balanced**

smt = SMOTE(random\_state=0)

X\_train\_SMOTE, y\_train\_SMOTE = smt.fit\_sample(x\_train, y\_train)

print(X\_train\_SMOTE.shape)

print(y\_train\_SMOTE.shape)

(19266, 5000)

(19266,)

y\_train\_SMOTE

0 negative

1 negative

2 positive

3 negative

4 negative

...

19261 positive

19262 positive

19263 positive

19264 positive

19265 positive

Name: airline\_sentiment, Length: 19266, dtype: object

**Checking if data is balanced after applying SMOTE**

after\_smote\_airline\_sentiment=pd.DataFrame(y\_train\_SMOTE, columns=['airline\_sentiment'])

y = after\_smote\_airline\_sentiment['airline\_sentiment']

print(y.value\_counts())

plt.figure(figsize=(20,5))

g = sns.countplot(y)

*# Here we can see that after applying smote, the data is balanced*

positive 6422

neutral 6422

negative 6422

Name: airline\_sentiment, dtype: int64



**Dividing Test data to Test and Validation Data**

*# Dividing the test data into test and validation set in 50-50 ratio*

x\_validation, x\_test\_main, y\_validation, y\_test\_main = train\_test\_split(x\_test, y\_test, test\_size=0.50, random\_state=1)

print(x\_validation.shape)

print(x\_test\_main.shape)

print(y\_validation.shape)

print(y\_test\_main.shape)

*# There are 2185 samples for validation and 2186 samples for testing*

(2185, 5000)

(2186, 5000)

(2185,)

(2186,)

**Initialize RandomForestClassifier**

*# Initialize a Random Forest classifier with 100 trees*

randomforestclassifier = RandomForestClassifier(bootstrap=True, ccp\_alpha=0.0, class\_weight=None,

criterion='gini', max\_depth=None, max\_features='auto',

max\_leaf\_nodes=None, max\_samples=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=100,

n\_jobs=None, oob\_score=False, random\_state=1, verbose=0,

warm\_start=False)

*# Fit the forest to the training set, using the bag of words as*

*# features and the sentiment labels as the response variable*

print ("Training the random forest...")

randomforestclassifier = randomforestclassifier.fit( X\_train\_SMOTE, y\_train\_SMOTE)

randomforestclassifier.score(X\_train\_SMOTE, y\_train\_SMOTE)

0.9391674452403197

**Evaluate score by cross-validation**

print (np.mean(cross\_val\_score(randomforestclassifier,X\_train\_SMOTE, y\_train\_SMOTE,cv=10)))

0.8023050588429925

*# Make class predictions for the Validation set*

y\_validation\_predict= randomforestclassifier.predict(x\_validation)

**Training and Validation Accuracy**

print("Trainig accuracy",randomforestclassifier.score(X\_train\_SMOTE,y\_train\_SMOTE))

print()

print("Validation accuracy",randomforestclassifier.score(x\_validation, y\_validation))

print()

Trainig accuracy 0.9391674452403197

Validation accuracy 0.765675057208238

**Classification Report** (Validation Set)

print(metrics.classification\_report(y\_validation,y\_validation\_predict))

precision recall f1-score support

negative 0.82 0.89 0.85 1331

neutral 0.61 0.52 0.56 467

positive 0.73 0.62 0.67 387

accuracy 0.77 2185

macro avg 0.72 0.68 0.69 2185

weighted avg 0.76 0.77 0.76 2185

**Confusion Matrix** (Validation Set)

cm=confusion\_matrix(y\_validation\_predict , y\_validation)

plt.figure()

plot\_confusion\_matrix(cm,figsize=(12,8), hide\_ticks=True,cmap=plt.cm.Reds)

plt.xticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16,color='black')

plt.yticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16)

plt.show()



**Random Forest Classifier on TfidfVectorizer**

**Dividing Data into train and Test**

x\_tf\_idf = tfidf\_vectorizer\_data\_features *# Predictor feature columns*

y\_tf\_idf = tweet\_data\_relevant['airline\_sentiment'] *# Predicted class*

x\_train\_tf\_idf, x\_test\_tf\_idf, y\_train\_tf\_idf, y\_test\_tf\_idf = train\_test\_split(x\_tf\_idf, y\_tf\_idf, test\_size=0.3, random\_state=1) *# 1 is just any random seed number*

print(x\_train\_tf\_idf.shape)

print(y\_train\_tf\_idf.shape)

print(x\_test\_tf\_idf.shape)

print(y\_test\_tf\_idf.shape)

(10197, 5000)

(10197,)

(4371, 5000)

(4371,)

y = tweet\_data\_relevant['airline\_sentiment']

print(y.value\_counts())

plt.figure(figsize=(20,5))

g = sns.countplot(y)

*# No, Here we can see that the data is not balanced, There are lot of negative sentiments*

negative 9140

neutral 3083

positive 2345

Name: airline\_sentiment, dtype: int64



**Applying SMOTE since the data is not balanced**

tf\_idf\_smt = SMOTE(random\_state=0)

X\_train\_tf\_idf\_SMOTE, y\_train\_tf\_idf\_SMOTE = smt.fit\_sample(x\_train\_tf\_idf, y\_train\_tf\_idf)

print(X\_train\_tf\_idf\_SMOTE.shape)

print(y\_train\_tf\_idf\_SMOTE.shape)

(19266, 5000)

(19266,)

after\_smote\_airline\_sentiment\_tf\_idf=pd.DataFrame(y\_train\_tf\_idf\_SMOTE, columns=['airline\_sentiment'])

y = after\_smote\_airline\_sentiment\_tf\_idf['airline\_sentiment']

print(y.value\_counts())

plt.figure(figsize=(20,5))

g = sns.countplot(y)

*# Here we can see that after smote , the data is balanced*

positive 6422

neutral 6422

negative 6422

Name: airline\_sentiment, dtype: int64



*# Dividing the test data into test and validation set in 50-50 ratio*

x\_validation\_tf\_idf, x\_test\_main\_tf\_idf, y\_validation\_tf\_idf, y\_test\_main\_tf\_idf = train\_test\_split(x\_test\_tf\_idf, y\_test\_tf\_idf, test\_size=0.50, random\_state=1)

print(x\_validation\_tf\_idf.shape)

print(x\_test\_main\_tf\_idf.shape)

print(y\_validation\_tf\_idf.shape)

print(y\_test\_main\_tf\_idf.shape)

*# There are 2185 samples for validation and 2186 samples for testing*

(2185, 5000)

(2186, 5000)

(2185,)

(2186,)

**Training and Validation Accuracy**

print("Trainig accuracy",randomforestclassifier\_tf\_idf.score(X\_train\_tf\_idf\_SMOTE,y\_train\_tf\_idf\_SMOTE))

print()

print("Validation accuracy",randomforestclassifier\_tf\_idf.score(x\_validation\_tf\_idf, y\_validation\_tf\_idf))

print()

**Classification Report** (Test Set)

*# Make class predictions for the test set*

y\_test\_predict\_tf\_idf= randomforestclassifier\_tf\_idf.predict(x\_test\_main\_tf\_idf)

print(metrics.classification\_report(y\_test\_main\_tf\_idf,y\_test\_predict\_tf\_idf))

precision recall f1-score support

negative 0.82 0.92 0.86 1387

neutral 0.64 0.48 0.55 465

positive 0.72 0.59 0.65 334

accuracy 0.78 2186

macro avg 0.73 0.66 0.69 2186

weighted avg 0.76 0.78 0.76 2186

**Confusion Matrix** (Test Set)

cm=confusion\_matrix(y\_validation\_predict\_tf\_idf , y\_validation\_tf\_idf)

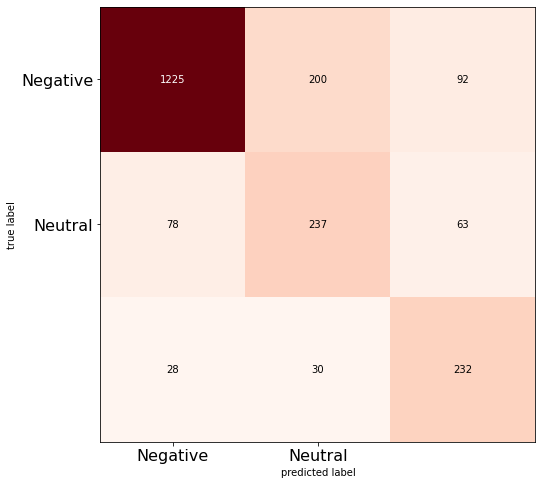
plt.figure()

plot\_confusion\_matrix(cm,figsize=(12,8), hide\_ticks=True,cmap=plt.cm.Reds)

plt.xticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16,color='black')

plt.yticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16)

plt.show()



**Test Accuracy**

print("Test accuracy",randomforestclassifier\_tf\_idf.score(x\_test\_main\_tf\_idf, y\_test\_main\_tf\_idf))

print()

**Classification Report** (Test Set)

*# Make class predictions for the test set*

y\_test\_predict\_tf\_idf= randomforestclassifier\_tf\_idf.predict(x\_test\_main\_tf\_idf)

print(metrics.classification\_report(y\_test\_main\_tf\_idf,y\_test\_predict\_tf\_idf))

**Confusion Matrix** (Test Set)

cm=confusion\_matrix(y\_test\_predict\_tf\_idf , y\_test\_main\_tf\_idf)

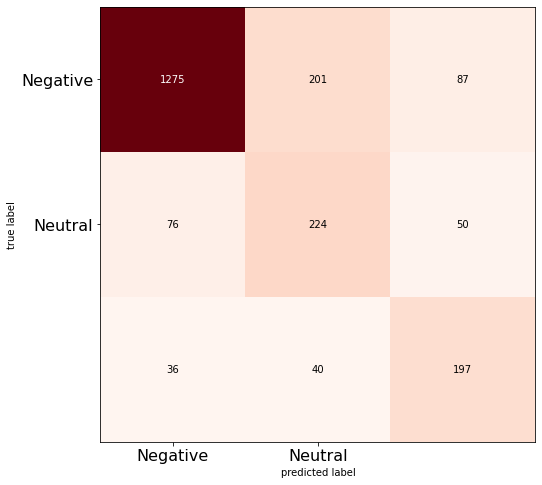
plt.figure()

plot\_confusion\_matrix(cm,figsize=(12,8), hide\_ticks=True,cmap=plt.cm.Reds)

plt.xticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16,color='black')

plt.yticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16)

plt.show()



**STEP 6**

**Pre-processing steps in NLP**

*# Pre-processing steps in NLP*

*# 1. HTML tag cleanup*

*# - It returns all the text in a document or beneath a tag, as a single Unicode string:*

*# 2. Contraction*

*# - Contractions are shortened version of words or syllables.*

*# - In case of English contractions are often created by removing one of the vowels from the word.*

*# - Examples would be, do not to don’t and I would to I’d. Converting each contraction to its expanded, original form helps with text standardization.*

*# 3. Tokenization*

*# - Tokenization is a step which splits longer strings of text into smaller pieces, or tokens.*

*# - Larger chunks of text can be tokenized into sentences, sentences can be tokenized into words, etc.*

*# - Further processing is generally performed after a piece of text has been appropriately tokenized.*

*# - Tokenization is also referred to as text segmentation or lexical analysis.*

*# - Sometimes segmentation is used to refer to the breakdown of a large chunk of text into pieces larger than words (e.g. paragraphs or sentences),*

*# while tokenization is reserved for the breakdown process which results exclusively in words.*

*# 4. Removing numbers*

*# - Remove numbers from list of tokenized words*

*# 5. Remove special characters*

*# - Remove special characters from list of tokenized words*

*# 6. Remove punctuation*

*# - Remove punctuation from list of tokenized words*

*# 7. Convert text to lower case*

*# - converting all text to the same case*

*# 8. Remove empty strings*

*# - Remove empty string from list of tokenized words*

*# 9. Stemming*

*# - Converting the words into their base word or stem word ( Ex - tastefully, tasty, these words are converted to stem word called 'tasti').*

*# This reduces the vector dimension because we dont consider all similar words*

*# 10.Lemmatization*

*# - Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language.*

*# In Lemmatization root word is called Lemma. A lemma (plural lemmas or lemmata) is the canonical form, dictionary form, or citation form of a set of words.*

**Techniques for Encoding**

*# Steps to do after text pre-processing:*

*#Techniques for Encoding - These are the popular techniques that are used for encoding:*

*# o Bag of words (CountVectorization)*

*# In BoW we construct a dictionary that contains set of all unique words from our text review dataset.*

*# The frequency of the word is counted here. If there are d unique words in our dictionary then for every sentence or review the vector will be of length d*

*# and count of word from review is stored at its particular location in vector. The vector will be highly sparse in such case.*

*# o Tf-idf (TfIdfVectorization) (Term Frequency - Inverse Document Frequency)*

*# Term Frequency - Inverse Document Frequency it makes sure that less importance is given to most frequent words and also considers less frequent words.*

*# Term Frequency is number of times a particular word(W) occurs in a review divided by totall number of words (Wr) in review. The term frequency value ranges from 0 to 1.*

*# Inverse Document Frequency is calculated as log(Total Number of Docs(N) / Number of Docs which contains particular word(n)). Here Docs referred as Reviews.*

*# TF-IDF is TF \* IDF that is (W/Wr)\*LOG(N/n)*

**Performance of Classification Model**

**RandomForestClassifier on CountVectorizer**

print("Trainig accuracy",randomforestclassifier.score(X\_train\_SMOTE,y\_train\_SMOTE))

print()

print("Testing accuracy",randomforestclassifier.score(x\_test\_main, y\_test\_main))

print()

y\_test\_predict= randomforestclassifier.predict(x\_test\_main)

print(metrics.classification\_report(y\_test\_main,y\_test\_predict))

cm=confusion\_matrix(y\_test\_predict , y\_test\_main)

plt.figure()

plot\_confusion\_matrix(cm,figsize=(12,8), hide\_ticks=True,cmap=plt.cm.Reds)

plt.xticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16,color='black')

plt.yticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16)

plt.show()

precision recall f1-score support

negative 0.81 0.90 0.85 1387

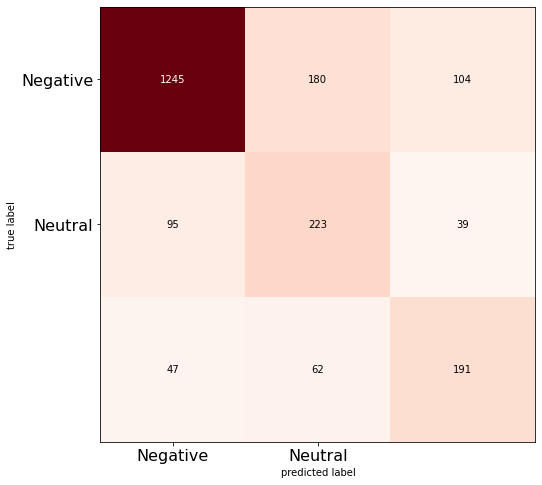
neutral 0.62 0.48 0.54 465

positive 0.64 0.57 0.60 334

accuracy 0.76 2186

macro avg 0.69 0.65 0.67 2186

weighted avg 0.75 0.76 0.75 2186



**RandomForestClassifier on TfidfVectorizer**

print("Trainig accuracy",randomforestclassifier\_tf\_idf.score(X\_train\_tf\_idf\_SMOTE,y\_train\_tf\_idf\_SMOTE))

print()

print("Test accuracy",randomforestclassifier\_tf\_idf.score(x\_test\_main\_tf\_idf, y\_test\_main\_tf\_idf))

print()

*# Make class predictions for the test set*

y\_test\_predict\_tf\_idf= randomforestclassifier\_tf\_idf.predict(x\_test\_main\_tf\_idf)

print(metrics.classification\_report(y\_test\_main\_tf\_idf,y\_test\_predict\_tf\_idf))

cm=confusion\_matrix(y\_test\_predict\_tf\_idf , y\_test\_main\_tf\_idf)

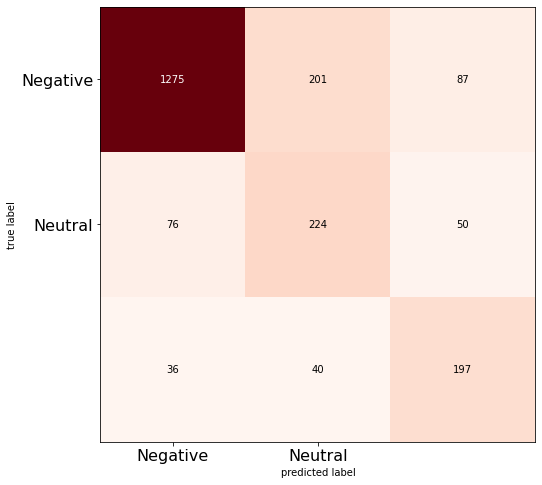
plt.figure()

plot\_confusion\_matrix(cm,figsize=(12,8), hide\_ticks=True,cmap=plt.cm.Reds)

plt.xticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16,color='black')

plt.yticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16)

plt.show()



Accuracy=[]

Model=[]

Accuracy.append(randomforestclassifier.score(x\_test\_main, y\_test\_main))

Accuracy.append(randomforestclassifier\_tf\_idf.score(x\_test\_main\_tf\_idf, y\_test\_main\_tf\_idf))

Model.append("RandomForestClassifier on CountVectorizer")

Model.append("RandomForestClassifier on TfidfVectorizer")

index=[0,1]

plt.bar(index,Accuracy,color='rgbyk')

plt.xticks(index,Model,rotation=45)

plt.ylabel('Accuracy')

plt.xlabel('Model')

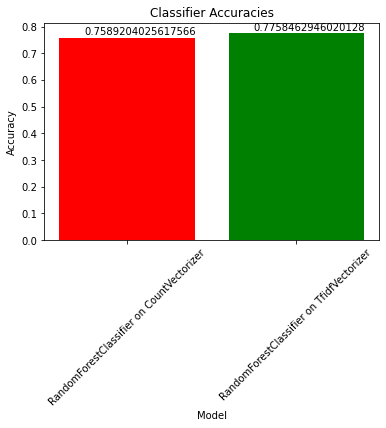
plt.title('Classifier Accuracies')

xlocs, xlabs = plt.xticks()

for i, v **in** enumerate(Accuracy):

plt.text(xlocs[i] - 0.25, v + 0.01, str(v))

*# The RandomForestClassfier on TfidfVectorizer is having better accuracy*



CONCLUSION:

Upon evaluating all the models, we can conclude the following details i.e. Accuracy: As far as the accuracy of the model is concerned, Logistic Regression performs better than SVM, which in turn performs better than Bernoulli Naive Bayes.