Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

```
import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os

from google.colab import drive
drive.mount('/content/gdrive')
```

```
# using the SQLite Table to read data.
con = sqlite3.connect('gdrive/My Drive/Colab Notebooks/database.sqlite')
#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 5000
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50000'

# Give reviews with Score>3 a positive rating, and reviews with a score<3 a negative rati
def partition(x):
    if x < 3:
        return 0
    return 1
```

```
#changing reviews with score less than 3 to be positive and vice-versa
actualScore = filtered_data['Score']
positiveNegative = actualScore.map(partition)
filtered_data['Score'] = positiveNegative
print("Number of data points in our data", filtered_data.shape)
filtered_data.head(3)
```

```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Summary,Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

```
print(display.shape)
display.head()
```

```
display[display['UserId']=='AZY10LLTJ71NX']
```

```
display['COUNT(*)'].sum()
```

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False
```

```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='f
final.shape
```

```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
#Before starting the next phase of preprocessing lets see the number of entries left
print(final.shape)

#How many positive and negative reviews are present in our dataset?
final['Score'].value_counts()
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.

3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tag
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)
```

```
soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase


sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves',
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', '
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'th
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "tha
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as',
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ove
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any',
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too',
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'might
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wa
            'won', "won't", 'wouldn', "wouldn't"])
```

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopword
    preprocessed_reviews.append(sentance.strip())
```

```
preprocessed_reviews[1500]
```

[3.2] Preprocessing Review Summary

```
## Similartly you can do preprocessing for review summary also.
```

[4] Featurization

[4.1] BAG OF WORDS

```
#BoW
count_vect = CountVectorizer() #in scikit-learn
count_vect.fit(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

final_counts = count_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_counts))
print("the shape of out text BOW vectorizer ",final_counts.get_shape())
print("the number of unique words ", final_counts.get_shape()[1])
```

[4.2] Bi-Grams and n-Grams.

```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/modules
# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("some feature names ", count_vect.get_feature_names()[:10])
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_co
```

[4.3] TF-IDF

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df=10)
tf_idf_vect.fit(preprocessed_reviews)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feature_names()
print('='*50)

final_tf_idf = tf_idf_vect.transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_tf_idf))
print("the shape of out text TFIDF vectorizer ",final_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_tf_idf.ge
print(final_tf_idf[1:10].toarray())
```

[4.4] Word2Vec

```
# Train your own Word2Vec model using your own text corpus
i=0
list_of_sentance=[]
for sentance in preprocessed_reviews:
    list_of_sentance.append(sentance.split())
```

```
# Using Google News Word2Vectors

# in this project we are using a pretrained model by google
# its 3.3G file, once you load this into your memory
# it occupies ~9Gb, so please do this step only if you have >12G of ram
# we will provide a pickle file wich contains a dict ,
# and it contains all our courpus words as keys and  model[word] as values
# To use this code-snippet, download "GoogleNews-vectors-negative300.bin"
# from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
# it's 1.9GB in size.


# http://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/#.W17SRFAzZPY
# you can comment this whole cell
# or change these varible according to your need

is_your_ram_gt_16g=False
want_to_use_google_w2v = False
want_to_train_w2v = True

if want_to_train_w2v:
    # min_count = 5 considers only words that occured atleast 5 times
    w2v_model=Word2Vec(list_of_sentance,min_count=5,size=500, workers=4)
    print(w2v_model.wv.most_similar('great'))
    print('='*50)
    print(w2v_model.wv.most_similar('worst'))

elif want_to_use_google_w2v and is_your_ram_gt_16g:
    if os.path.isfile('GoogleNews-vectors-negative300.bin'):
        w2v_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin'
        print(w2v_model.wv.most_similar('great'))
        print(w2v_model.wv.most_similar('worst'))
    else:
        print("you don't have gogole's word2vec file, keep want_to_train_w2v = True, to
```

```
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

[4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

```
# average Word2Vec
# compute average word2vec for each review.
sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(500) # as word vectors are of zero length 50, you might need to
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
```

```
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    print('count',cnt_words)
    print(len(sent_vectors))
    print(len(sent_vectors[0]))
```

[4.4.1.2] TFIDF weighted W2v

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(preprocessed_reviews)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
```

```
# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(500) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors.append(sent_vec)
    row += 1
```

Double-click (or enter) to edit

[5] Assignment 5: Apply Logistic Regression

1. **Apply Logistic Regression on these feature sets**
    - SET 1:Review text, preprocessed one converted into vectors using (BOW)
    - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
    - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
    - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **Hyper paramter tuning (find best hyper parameters corresponding the algorithm that you choose)**

    - Find the best hyper parameter which will give the maximum AUC value
    - Find the best hyper paramter using k-fold cross validation or simple cross validation data
    - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Pertubation Test**

    - Get the weights W after fit your model with the data X i.e Train data.
    - Add a noise to the X (X' = X + e) and get the new data set X' (if X is a sparse matrix, X.data+=e)
    - Fit the model again on data X' and get the weights W'
    - Add a small eps value(to eliminate the divisible by zero error) to W and W' i.e W=W+10^-6 and W' = W'+10^-6
    - Now find the % change between W and W' (| (W-W') / (W) |)*100)
    - Calculate the 0th, 10th, 20th, 30th, ...100th percentiles, and observe any sudden rise in the values of percentage_change_vector
    - Ex: consider your 99th percentile is 1.3 and your 100th percentiles are 34.6, there is sudden rise from 1.3 to 34.6, now calculate the 99.1, 99.2, 99.3,..., 100th percentile values and get the proper value after which there is sudden rise the values, assume it is 2.5
    - Print the feature names whose % change is more than a threshold x(in our example it's 2.5)

4. **Sparsity**

    - Calculate sparsity on weight vector obtained after using L1 regularization

    NOTE: Do sparsity and multicollinearity for any one of the vectorizers. Bow or tf-idf is recommended.

5. **Feature importance**

    - Get top 10 important features for both positive and negative classes separately.

6. **Feature engineering**

    - To increase the performance of your model, you can also experiment with with feature engineering like :

        - Taking length of reviews as another feature.
        - Considering some features from review summary as well.

7. **Representation of results**

    - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure.
    - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.
    - Along with plotting ROC curve, you need to print the confusion matrix with predicted and original labels of test data points. Please visualize your confusion matrices using

seaborn heatmaps.

8. **Conclusion**
    - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library link

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link.

```
# Please write all the code with proper documentation
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_sp
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
import seaborn as sn
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
# Split the SET1 in Train, CV and test data set

Y = final['Score'].values
X = preprocessed_reviews





X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.3, random_state =
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this


vectorizer = CountVectorizer()
vectorizer.fit(X_train)
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)


scaler = StandardScaler(with_mean=False, with_std=True)
scaler.fit(X_train_bow)
scaler.transform(X_train_bow)
scaler.transform(X_cv_bow)
scaler.transform(X_test_bow)
#preprocessing.scale(X_train_bow)


print(X_train_bow.get_shape())
print(X_cv_bow.get_shape())
print(X_test_bow.get_shape())

# getting AUC ROC curve

# Getting the optimal function with Grid search
parameters = [{'C':[5,1,0.1,0.01,0.05,0.01,0.005,0.001, 0.0005, 0.0001]}]
LR_optimal = GridSearchCV(LogisticRegression(penalty='l2'),parameters,scoring = 'f1',cv=
LR_optimal.fit(X_train_bow, y_train)
print("Best estimator & Score for the LR with GridSearchCV")
print(LR_optimal.best_estimator_)
print(LR_optimal.score(X_train_bow, y_train))

# Getting the optimal function with normal looping
train_auc = []
cv_auc = []
alpha = [5,1,0.5,0.1,0.05,0.01,0.005,0.001, 0.0005, 0.0001]
for i in alpha:
    LR_optimal = LogisticRegression(C=i, class_weight=None, dual=False, fit_intercept=Tru
    LR_optimal.fit(X_train_bow, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs
    y_train_pred =  LR_optimal.predict_log_proba(X_train_bow)[:,1]
    y_cv_pred =  LR_optimal.predict_log_proba(X_cv_bow)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(np.log(alpha), train_auc, label='Train AUC')
plt.plot(np.log(alpha), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("hyperparameter")
plt.ylabel("Error")
plt.title("ERROR PLOTS")
plt.show()



#Checking if sparsity increases with L1 regularizer
clf = LogisticRegression(C=1,penalty = 'l1')
clf.fit(X_train_bow, y_train)
w=clf.coef_
print(np.count_nonzero(w))

clf = LogisticRegression(C=0.1,penalty = 'l1')
clf.fit(X_train_bow, y_train)
w=clf.coef_
print(np.count_nonzero(w))

# Fitting with best hyperparamter
LR_optimal = LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,in
LR_optimal.fit(X_train_bow, y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, LR_optimal.predict_log_proba(X_tra
test_fpr, test_tpr, thresholds = roc_curve(y_test, LR_optimal.predict_log_proba(X_test_bo
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC AUC Plot")
plt.show()

print("Printing weight vector befor noise")
W1 = LR_optimal.coef_
print(W1)

print("="*100)

# Adding some noise
print("Plotting the graph after some adding some noise")
e = np.random.normal()
X_train_bow.data = X_train_bow.data + e
X_cv_bow.data = X_cv_bow.data + e
X_test_bow.data = X_test_bow.data + e

LR_optimal = LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,in
LR_optimal.fit(X_train_bow, y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, LR_optimal.predict_log_proba(X_tra
test_fpr, test_tpr, thresholds = roc_curve(y_test, LR_optimal.predict_log_proba(X_test_bo
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
```

```
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC AUC PLOTS with Noise")
plt.show()

print("Printing weight vector after noise")
W2 = LR_optimal.coef_
print(W2)
#W1 = W1 + e
#W2 = W2 + e
# Finding percentage difference between 2 vectors
percentage_change_vector = np.abs((W2-W1)/W1)*100
print("%change",percentage_change_vector)
change=[]
for i in range(90,101):
   change.append(np.percentile(percentage_change_vector,i))
x1 = list(range(90,101))
y1 = change
plt.title("Multicolliniearlity test")
plt.plot(x1,y1)
plt.show()
```

```
change_max = []
ind = 1;
print("features which changed more than threshold")
feature_names = vectorizer.get_feature_names()
print(percentage_change_vector[0])
for i in range(1,len(percentage_change_vector[0])):

  if(percentage_change_vector[0][i]>2.5):
    change_max.append(feature_names[i])

print(change_max)


from sklearn.metrics import confusion_matrix
print("Train confusion matrix")
print(confusion_matrix(y_train, LR_optimal.predict(X_train_bow)))
print("Test confusion matrix")
print(confusion_matrix(y_test, LR_optimal.predict(X_test_bow)))
sn.heatmap(confusion_matrix(y_test, LR_optimal.predict(X_test_bow)), annot=True)
```

[5.1.1] Top 10 important features of positive class from SET 1

```
feature_names = vectorizer.get_feature_names()
last10 = np.argsort(LR_optimal.coef_[0])[-10:]
print("top words for +Ve class is")
Top_Positive = []
for j in last10:
  Top_Positive.append(feature_names[j])
print(Top_Positive)
```

[5.1.2] Top 10 important features of negative class from SET 1

```
print("top words for -Ve class is")
top10 = np.argsort(LR_optimal.coef_[0])[1:10]
Top_Negative = []
for j in top10:
  Top_Negative.append(feature_names[j])
print(Top_Negative)
```

```
# Pretty table update
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Vectorize Model", "ROC Test score", "Top 10 +ve features", "Top 10 -ve
x.add_row(["BOW",str(auc(test_fpr, test_tpr)), Top_Positive, Top_Negative])
```

[5.2] Applying Logistic Regression on TFIDF, SET 2

```
# Please write all the code with proper documentation
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_sp
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LogisticRegression
import seaborn as sn
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
# Split the SET1 in Train, CV and test data set

Y = final['Score'].values
X = preprocessed_reviews






X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.3, random_state =
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this


vectorizer = TfidfVectorizer(ngram_range=(1,2), min_df=10)
vectorizer.fit(X_train)
X_train_bow = vectorizer.transform(X_train)
X_cv_bow = vectorizer.transform(X_cv)
X_test_bow = vectorizer.transform(X_test)


scaler = StandardScaler(with_mean=False, with_std=True)
scaler.fit(X_train_bow)
scaler.transform(X_train_bow)
scaler.transform(X_cv_bow)
scaler.transform(X_test_bow)
#preprocessing.scale(X_train_bow)


print(X_train_bow.get_shape())
print(X_cv_bow.get_shape())
print(X_test_bow.get_shape())

# getting AUC ROC curve

# Getting the optimal function with Grid search
parameters = [{'C':[5,1,0.1,0.01,0.05,0.01,0.005,0.001, 0.0005, 0.0001]}]
LR_optimal = GridSearchCV(LogisticRegression(penalty='l2'),parameters,scoring = 'f1',cv=
LR_optimal.fit(X_train_bow, y_train)
print("Best estimator & Score for the LR with GridSearchCV")
print(LR_optimal.best_estimator_)
print(LR_optimal.score(X_train_bow, y_train))

# Getting the optimal function with normal looping
train_auc = []
cv_auc = []
alpha = [10,5,1,0.5,0.1,0.05,0.01,0.005,0.001, 0.0005, 0.0001]
for i in alpha:
    LR_optimal = LogisticRegression(C=i, class_weight=None, dual=False, fit_intercept=Tr
    LR_optimal.fit(X_train_bow, y_train)
```

```
        # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
        # not the predicted outputs
        y_train_pred =  LR_optimal.predict_log_proba(X_train_bow)[:,1]
        y_cv_pred =  LR_optimal.predict_log_proba(X_cv_bow)[:,1]

        train_auc.append(roc_auc_score(y_train,y_train_pred))
        cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

    plt.plot(np.log(alpha), train_auc, label='Train AUC')
    plt.plot(np.log(alpha), cv_auc, label='CV AUC')
    plt.legend()
    plt.xlabel("log alpha")
    plt.ylabel("Error")
    plt.title("ERROR PLOTS")
    plt.show()



    #Checking if sparsity increases with L1 regularizer
    clf = LogisticRegression(C=1,penalty = 'l1')
    clf.fit(X_train_bow, y_train)
    w=clf.coef_
    print(np.count_nonzero(w))

    clf = LogisticRegression(C=0.1,penalty = 'l1')
    clf.fit(X_train_bow, y_train)
    w=clf.coef_
    print(np.count_nonzero(w))

    # Fitting with best hyperparamter
    LR_optimal = LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,in
    LR_optimal.fit(X_train_bow, y_train)
    train_fpr, train_tpr, thresholds = roc_curve(y_train, LR_optimal.predict_log_proba(X_trai
    test_fpr, test_tpr, thresholds = roc_curve(y_test, LR_optimal.predict_log_proba(X_test_bo
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title("ROC AUC Plot")
    plt.show()

    print("Printing weight vector befor noise")
    W1 = LR_optimal.coef_
    print(W1)

    print("="*100)

    # Adding some noise
    print("Plotting the graph after some adding some noise")
    e = np.random.normal()
    X_train_bow.data = X_train_bow.data + e
    X_cv_bow.data = X_cv_bow.data + e
    X_test_bow.data = X_test_bow.data + e

    LR_optimal = LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,in
    LR_optimal.fit(X_train_bow, y_train)
    train_fpr, train_tpr, thresholds = roc_curve(y_train, LR_optimal.predict_log_proba(X_trai
    test_fpr, test_tpr, thresholds = roc_curve(y_test, LR_optimal.predict_log_proba(X_test_bo
    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title("ROC AUC PLOTS with Noise")
    plt.show()

    print("Printing weight vector after noise")
    W2 = LR_optimal.coef_
    print(W2)
    #W1 = W1 + e
    #W2 = W2 + e
    # Finding percentage difference between 2 vectors
    percentage_change_vector = np.abs((W2-W1)/W1)*100
```

```
print("%change",percentage_change_vector)
change=[]
for i in range(90,101):
   change.append(np.percentile(percentage_change_vector,i))
x1 = list(range(90,101))
y1 = change
plt.title("Multicolliniearlity test")
plt.plot(x1,y1)
plt.show()
```

[5.2.1] Top 10 important features of positive class from SET 2

```
feature_names = vectorizer.get_feature_names()
last10 = np.argsort(LR_optimal.coef_[0])[-10:]
print("top words for +Ve class is")
Top_Positive = []
for j in last10:
  #print(feature_names[j])
  Top_Positive.append(feature_names[j])
print(Top_Positive)
```

[5.2.2] Top 10 important features of negative class from SET 2

```
print("top words for -Ve class is")
top10 = np.argsort(LR_optimal.coef_[0])[1:10]
Top_Negative = []
for j in top10:
  #print(feature_names[j])
  Top_Negative.append(feature_names[j])
print(Top_Negative)
```

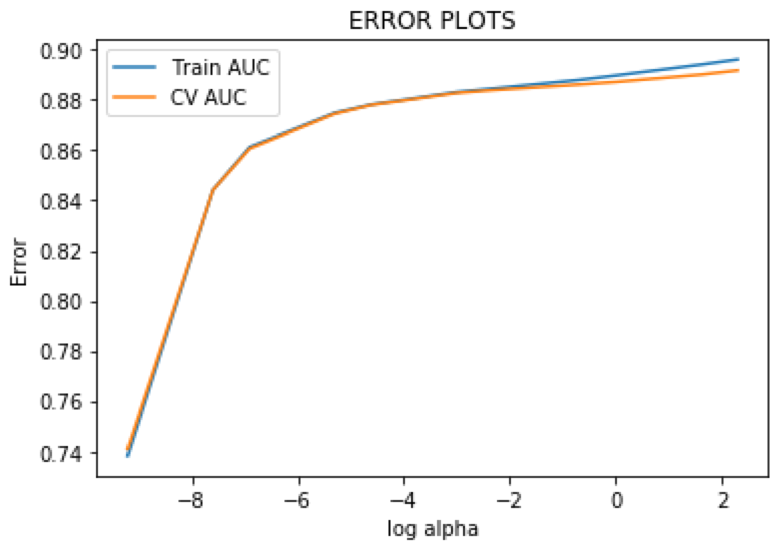
```
x.add_row(["TFIDF",str(auc(test_fpr, test_tpr)), Top_Positive, Top_Negative])
```

[5.2] Applying Logistic Regression on WordToVec, SET 3

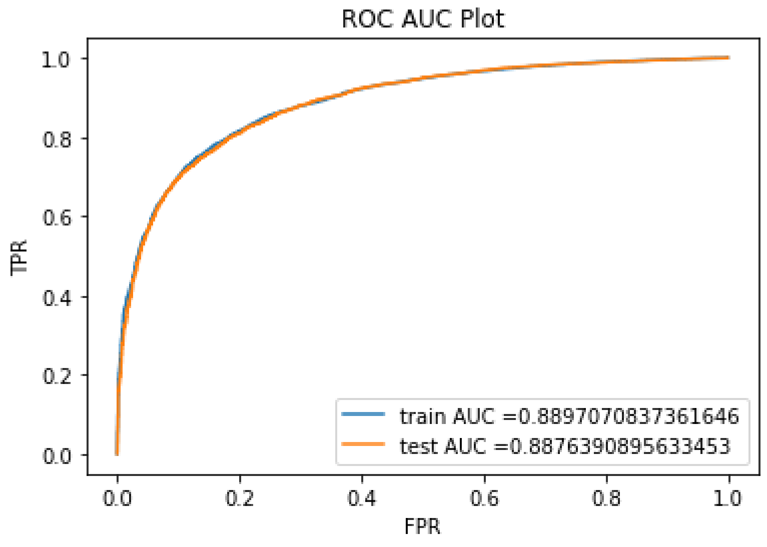
```
Y = final['Score'].values
X = preprocessed_reviews

X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.3, random_state =
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this

# Computing Average Word2Vec for X_train
i=0
list_of_sentance=[]
for sentance in X_train:
    list_of_sentance.append(sentance.split())
print(list_of_sentance[1:10])
w2v_model=Word2Vec(list_of_sentance,min_count=5,size=500, workers=4)
w2v_words = list(w2v_model.wv.vocab)
sent_vectors_X_train = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(500) # as word vectors are of zero length 50, you might need to 
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_X_train.append(sent_vec)
print('count',cnt_words)
print(len(sent_vectors_X_train))
print(len(sent_vectors_X_train[0]))


# Computing Average Word2Vec for X_cv
i=0
list_of_sentance=[]
for sentance in X_cv:
    list_of_sentance.append(sentance.split())
#print(list_of_sentance[1:10])
#w2v_model=Word2Vec(list_of_sentance,min_count=5,size=500, workers=4)
#w2v_words = list(w2v_model.wv.vocab)

sent_vectors_X_cv = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(500) # as word vectors are of zero length 50, you might need to 
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_X_cv.append(sent_vec)

#Computing Average Word2Vec for X_Test
i=0
list_of_sentance=[]
for sentance in X_test:
    list_of_sentance.append(sentance.split())
#print(list_of_sentance[1:10])
#w2v_model=Word2Vec(list_of_sentance,min_count=5,size=500, workers=4)
#w2v_words = list(w2v_model.wv.vocab)
```

```
sent_vectors_X_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(500) # as word vectors are of zero length 50, you might need to c
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words:
            vec = w2v_model.wv[word]
            sent_vec += vec
            cnt_words += 1
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors_X_test.append(sent_vec)

# Getting the optimal function with Grid search
parameters = [{'C':[5,1,0.1,0.01,0.05,0.01,0.005,0.001, 0.0005, 0.0001]}]
LR_optimal = GridSearchCV(LogisticRegression(penalty='l2'),parameters,scoring = 'f1',cv=8
LR_optimal.fit(sent_vectors_X_train, y_train)
print("Best estimator & Score for the LR with GridSearchCV")
print(LR_optimal.best_estimator_)
print(LR_optimal.score(sent_vectors_X_train, y_train))

# Getting the optimal function with normal looping
train_auc = []
cv_auc = []
alpha = [10,5,1,0.5,0.1,0.05,0.01,0.005,0.001, 0.0005, 0.0001]
for i in alpha:
    LR_optimal = LogisticRegression(C=i, class_weight=None, dual=False, fit_intercept=Tru
    LR_optimal.fit(sent_vectors_X_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs
    y_train_pred =  LR_optimal.predict_log_proba(sent_vectors_X_train)[:,1]
    y_cv_pred =  LR_optimal.predict_log_proba(sent_vectors_X_cv)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(np.log(alpha), train_auc, label='Train AUC')
plt.plot(np.log(alpha), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("log alpha")
plt.ylabel("Error")
plt.title("ERROR PLOTS")
plt.show()



#Checking if sparsity increases with L1 regularizer
clf = LogisticRegression(C=1,penalty = 'l1')
clf.fit(sent_vectors_X_train, y_train)
w=clf.coef_
print(np.count_nonzero(w))

clf = LogisticRegression(C=0.1,penalty = 'l1')
clf.fit(sent_vectors_X_train, y_train)
w=clf.coef_
print(np.count_nonzero(w))

# Fitting with best hyperparamter
LR_optimal = LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,in
LR_optimal.fit(sent_vectors_X_train, y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, LR_optimal.predict_log_proba(sent_v
test_fpr, test_tpr, thresholds = roc_curve(y_test, LR_optimal.predict_log_proba(sent_vect
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC AUC Plot")
plt.show()

print("Printing weight vector befor noise")
W1 = LR_optimal.coef_
print(W1)
print("="*100)

# Adding some noise
print("Plotting the graph after some adding some noise")
e = np.random.normal()
sent_vectors_X_train = sent_vectors_X_train + e
sent_vectors_X_cv = sent_vectors_X_cv + e
sent_vectors_X_test = sent_vectors_X_test + e

LR_optimal = LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,i
LR_optimal.fit(sent_vectors_X_train, y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, LR_optimal.predict_log_proba(sent_
test_fpr, test_tpr, thresholds = roc_curve(y_test, LR_optimal.predict_log_proba(sent_vec
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC AUC PLOTS with Noise")
plt.show()

print("Printing weight vector after noise")
W2 = LR_optimal.coef_
print(W2)
#W1 = W1 + e
#W2 = W2 + e
# Finding percentage difference between 2 vectors
percentage_change_vector = np.abs((W2-W1)/W1)*100
print("%change",percentage_change_vector)
change=[]
for i in range(90,101):
   change.append(np.percentile(percentage_change_vector,i))
x1 = list(range(90,101))
y1 = change
plt.title("Multicolliniearlity test")
plt.plot(x1,y1)
plt.show()
```

```
[['since', 'flowers', 'mostly', 'bought', 'either', 'women', 'seems', 'self', 'des
100%|██████████| 21606/21606 [00:34<00:00, 622.51it/s]
  1%|          | 63/10643 [00:00<00:16, 623.43it/s]count 54
21606
500
100%|██████████| 10643/10643 [00:17<00:00, 615.85it/s]
100%|██████████| 13822/13822 [00:22<00:00, 585.38it/s]
Best estimator & Score for the LR with GridSearchCV
LogisticRegression(C=5, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
0.9279166109848991
```



```
66
30
```



```
Printing weight vector befor noise
[[-0.16333303 -0.42002892 -0.22308782  0.67389992  0.02507807 -0.17897671
  -0.08089893 -0.39781039 -0.52855354 -0.24624009 -1.08050206  1.40622915
  -0.01489639 -0.37696482 -0.1914728   0.74954412 -0.11668544 -0.90389554
  -0.60782331 -0.45184668 -0.11917645  0.40299752 -0.36752524  0.09352298
  -0.76112742  0.57764551 -0.36299308 -1.06529776 -1.16854839  0.57198304
  -0.53174241 -0.36345383  0.00435436 -0.26203421  0.60660524 -0.28459231
  -0.92790426  1.3608943  -0.59597148 -0.33996206 -0.37536848 -1.1090406
   1.21329097  0.64642046 -1.85998468 -0.11438628  0.41321391 -0.34682838
   0.96961569  0.25311999 -1.06311587 -0.20948377 -0.57427968 -0.06326077
   0.00368139  0.09598057 -0.48386932 -0.29530823  0.17145528  0.67427605
   1.07532778 -0.6731058   0.96289383  0.18789148 -0.86641962 -0.34570253
   1.05497769  0.03544843  0.61196458  0.06146481  0.39869482 -1.27523965
  -0.24800719 -0.20960539 -0.70333065  0.02574333 -0.36809115 -0.56070301
   0.41205303  0.12462257  0.29507131 -0.25276875 -0.68872465 -0.29997153
   1.25390247  0.21404761  0.25633165 -0.2492133   0.30653573  0.02540081
   0.59842242 -0.75936373  0.26507584 -0.91076146 -0.19639805  0.22241293
  -0.97668472  0.12102477 -0.37286209  0.04766306 -0.22954448 -0.09245316
  -0.32427895 -0.59418198 -1.01224098 -0.02447498 -0.18438656  0.20234708
  -0.63212088  0.93035751  0.07158877  0.4822112   0.95189932 -0.14025406
  -0.19528814  0.25169593 -0.54311221 -0.29917783  0.06143898 -0.91381357
   0.01618615  0.42862278 -0.38912962 -0.04458793 -0.17326163 -0.13245062
   0.4340115  -0.6963422  -0.85980025 -0.43140726 -0.22707763 -1.18909767
  -0.06927797  0.5823302   0.36198908  0.20024005 -0.0955168  -0.12110448
  -0.69073316 -0.44486833 -0.39771413  0.20449391 -0.20113201  0.47353343
  -0.13273598 -0.77209614  0.64993044 -0.45727653  0.06470759 -0.48424399
   0.19578832  0.03484757  0.30194866 -1.41670974 -1.44305051  0.27603314
   0.71864298 -0.67823256 -0.1012126  -0.13881814 -0.4972417  -0.14463435
   0.08024269  0.43952421  0.44216374 -0.27066006 -0.24064447  0.4392595
  -0.57983214  0.42094892 -0.08774673 -0.02669324 -0.33699355 -0.06349418
   0.13677508 -0.45794907  0.01310832 -0.00559776 -0.48944322 -0.49906572
  -0.46241814  0.14678015 -0.76504544 -0.32322963 -0.10019866  0.0991304
   0.41112671 -0.2710305  -0.34683723  0.27037482  0.20818442 -0.47371092
   0.20747935  0.82419771 -0.30335407 -0.04045438  0.02824666  0.02808391
  -0.49080401 -0.5688258   0.47126925  0.2022671   0.38493447  0.08876792
   1.27519906  0.4462924   0.39347745  0.11666918  0.06778209 -1.20613048
  -0.11548783  0.27686452 -0.23384263  1.68897993  0.63988591 -0.49505594
  -0.38987114  0.62323155 -0.38287585  0.40677063 -0.62026361 -0.17896472
  -0.20838209  0.01060772  0.94849051 -0.03373106 -0.73127824 -0.58855072
   0.76478798 -0.34366838 -0.57030195  0.21831335 -0.57520572 -1.39002516
  -0.60259929  0.49002767  0.57120298  0.09318642 -0.46205412  0.55669152
   1.17928122 -0.33986491 -0.29204562  0.74948966  0.16028206  0.28582335
   0.36981395 -0.8892001  -0.20057655 -0.64154985 -0.27670852 -0.52076588
  -0.9197553  -0.18351293 -1.01503698  0.71174126  0.23029802  0.36294879
   0.33523609 -0.58816705 -0.07862355 -0.21811683 -0.58105412  0.32452659
   0.49397692 -0.16916563 -0.48032761  0.08301965 -0.0304365  -0.39192999
  -0.52061577 -0.64304052  0.22531647  0.51760348  0.47412903  0.66924472
   0.05090916  0.36355507 -0.3149382   0.12370478  0.0392704   0.65960042
   0.79761007  0.37522009 -0.1984805   0.35211477 -1.06524505 -0.32969187
  -0.36809532 -0.5815883  -0.46354352  0.66260783  0.06373911  0.86864187
  -0.20783365 -0.22234112 -0.81892577 -0.28472978 -0.06349814 -0.8494289
   0.39353393 -0.66672876  0.05010112  0.00751366  0.32613769 -0.09495854
   0.60343264  0.32322781 -0.40608615  0.17972508 -0.19166209  0.37357427
   1.04699433  0.4233178   0.92660467  0.40194515  0.18911855  0.39318772
  -0.0893274   0.23706237  0.8942019  -0.21428729 -0.387624    0.10798085
  -1.19553495  0.66178257  0.17921375  0.44679731  0.06907752 -0.07189459
   0.02495596  0.10632878  0.0192016   0.33181526  0.23995606 -0.12520186
   0.45290264  0.0228095  -0.28570186 -0.15788082 -0.01509418 -0.60983199
   0.12098668 -0.11709075 -0.67111926 -0.5802262   0.43671069 -0.5309576
   0.37137007 -0.29859356 -0.34795421 -0.01216104 -0.64635312  0.22417183
  -0.33899932  1.01111971 -0.00599007  0.54244936 -0.16147698  0.76552733
  -0.1072882   0.22910574  0.56672359  0.22172949 -0.4133234   0.78421636
   0.00797799 -0.08750519 -0.19315679 -0.20689443 -0.62453538  0.31999977
  -0.30191605  0.88496653  0.29066977  0.26968416  0.02586731 -0.50759263
  -0.24566071  0.26472273  0.54767648  0.70219116 -0.14381433 -1.1976279
   0.65336595  0.80555778 -0.57059744  0.09486038  0.5628525   0.06831728
  -0.31894806 -0.29994224  0.60887653  0.0409763  -0.3976903   0.28505408
  -1.471597   -0.23748488  0.52274683 -1.22280661 -0.82234298  0.01977513
   0.47445535 -1.04206465 -0.57470101 -0.45098467 -0.2136245   0.1147028
  -0.07021623 -0.4189423  -0.0358931  -0.69314721 -0.13522829  0.04384328
   0.60098496  0.4035587   0.50169391  0.66035055  0.23121869 -0.22519256
   0.85182854 -0.3149741  -1.00959009  0.41779218 -0.11987076 -0.56337182
```

[5.2] Applying Logistic Regression on TFIDF Word2Vec, SET 4

```
        0.50031751  0.31821843 -0.02788253 -0.07795483 -0.7221563  -0.1389732
```

```
  # Split the SET1 in Train, CV and test data set

Y = final['Score'].values
X = preprocessed_reviews

X_train, X_test, y_train, y_test = train_test_split(X,Y, test_size = 0.3, random_state =
X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train, test_size=0.33) # this

# Computing TFIDF Average Word2Vec for X_train
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]

i=0
list_of_sentance=[]
for sentance in X_train:
    list_of_sentance.append(sentance.split())

model = TfidfVectorizer()
model.fit(X_train)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
tfidf_feat = model.get_feature_names()

w2v_model=Word2Vec(list_of_sentance,min_count=5,size=500, workers=4)
w2v_words = list(w2v_model.wv.vocab)
tfidf_sent_vectors_X_train = []; # the avg-w2v for each sentence/review is stored in thi
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(500) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#               tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_X_train.append(sent_vec)
    row += 1
print('count',cnt_words)
print(len(tfidf_sent_vectors_X_train))
print(len(tfidf_sent_vectors_X_train[0]))


# Computing Average Word2Vec for X_cv

i=0
list_of_sentance=[]
for sentance in X_cv:
    list_of_sentance.append(sentance.split())

#model = TfidfVectorizer()
#model.fit(X_cv)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
tfidf_feat = model.get_feature_names()

#w2v_model=Word2Vec(list_of_sentance,min_count=5,size=500, workers=4)
#w2v_words = list(w2v_model.wv.vocab)

tfidf_sent_vectors_X_cv = []; # the avg-w2v for each sentence/review is stored in this l
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(500) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
```

```
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_X_cv.append(sent_vec)
    row += 1


# Computing Average Word2Vec for X_Test

i=0
list_of_sentance=[]
for sentance in X_test:
    list_of_sentance.append(sentance.split())

#model = TfidfVectorizer()
#model.fit(X_test)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
tfidf_feat = model.get_feature_names()

#w2v_model=Word2Vec(list_of_sentance,min_count=5,size=500, workers=4)
#w2v_words = list(w2v_model.wv.vocab)

tfidf_sent_vectors_X_test = []; # the avg-w2v for each sentence/review is stored in this
row=0;
for sent in tqdm(list_of_sentance): # for each review/sentence
    sent_vec = np.zeros(500) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/review
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    tfidf_sent_vectors_X_test.append(sent_vec)
    row += 1

scaler = StandardScaler(with_mean=False, with_std=True)
scaler.fit(tfidf_sent_vectors_X_train)
scaler.transform(tfidf_sent_vectors_X_train)
scaler.transform(tfidf_sent_vectors_X_cv)
scaler.transform(tfidf_sent_vectors_X_test)
#preprocessing.scale(X_train_bow)



# getting AUC ROC curve

# Getting the optimal function with Grid search
parameters = [{'C':[5,1,0.1,0.01,0.05,0.01,0.005,0.001, 0.0005, 0.0001]}]
LR_optimal = GridSearchCV(LogisticRegression(penalty='l2'),parameters,scoring = 'f1',cv=
LR_optimal.fit(tfidf_sent_vectors_X_train, y_train)
print("Best estimator & Score for the LR with GridSearchCV")
print(LR_optimal.best_estimator_)
print(LR_optimal.score(tfidf_sent_vectors_X_train, y_train))

# Getting the optimal function with normal looping
train_auc = []
cv_auc = []
alpha = [10,5,1,0.5,0.1,0.05,0.01,0.005,0.001, 0.0005, 0.0001]
```

```
for i in alpha:
    LR_optimal = LogisticRegression(C=i, class_weight=None, dual=False, fit_intercept=Tru
    LR_optimal.fit(tfidf_sent_vectors_X_train, y_train)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of
    # not the predicted outputs
    y_train_pred =  LR_optimal.predict_log_proba(tfidf_sent_vectors_X_train)[:,1]
    y_cv_pred =  LR_optimal.predict_log_proba(tfidf_sent_vectors_X_cv)[:,1]

    train_auc.append(roc_auc_score(y_train,y_train_pred))
    cv_auc.append(roc_auc_score(y_cv, y_cv_pred))

plt.plot(np.log(alpha), train_auc, label='Train AUC')
plt.plot(np.log(alpha), cv_auc, label='CV AUC')
plt.legend()
plt.xlabel("log alpha")
plt.ylabel("Error")
plt.title("ERROR PLOTS")
plt.show()



#Checking if sparsity increases with L1 regularizer
clf = LogisticRegression(C=1,penalty = 'l1')
clf.fit(tfidf_sent_vectors_X_train, y_train)
w=clf.coef_
print(np.count_nonzero(w))

clf = LogisticRegression(C=0.1,penalty = 'l1')
clf.fit(tfidf_sent_vectors_X_train, y_train)
w=clf.coef_
print(np.count_nonzero(w))

# Fitting with best hyperparamter
LR_optimal = LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,in
LR_optimal.fit(tfidf_sent_vectors_X_train, y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, LR_optimal.predict_log_proba(tfidf_
test_fpr, test_tpr, thresholds = roc_curve(y_test, LR_optimal.predict_log_proba(tfidf_sen
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC AUC Plot")
plt.show()

print("Printing weight vector befor noise")
W1 = LR_optimal.coef_
print(W1)

print("="*100)

# Adding some noise
print("Plotting the graph after some adding some noise")
e = np.random.normal()
tfidf_sent_vectors_X_train  = tfidf_sent_vectors_X_train + e
tfidf_sent_vectors_X_cv  = tfidf_sent_vectors_X_cv + e
tfidf_sent_vectors_X_test = tfidf_sent_vectors_X_test + e

LR_optimal = LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True,in
LR_optimal.fit(tfidf_sent_vectors_X_train, y_train)
train_fpr, train_tpr, thresholds = roc_curve(y_train, LR_optimal.predict_log_proba(tfidf_
test_fpr, test_tpr, thresholds = roc_curve(y_test, LR_optimal.predict_log_proba(tfidf_sen
plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC AUC PLOTS with Noise")
plt.show()

print("Printing weight vector after noise")
W2 = LR_optimal.coef_
print(W2)
#W1 = W1 + e
```

```
#W2 = W2 + e
# Finding percentage difference between 2 vectors
percentage_change_vector = np.abs((W2-W1)/W1)*100
print("%change",percentage_change_vector)
change=[]
for i in range(90,101):
   change.append(np.percentile(percentage_change_vector,i))
x1 = list(range(90,101))
y1 = change
plt.title("Multicolliniearlity test")
plt.plot(x1,y1)
plt.show()
x.add_row(["TFIDF_Avg_Word_To_Vec",str(auc(test_fpr, test_tpr)), Top_Positive, Top_Negat
```