DECISION TREE ALGORITHM on Amazon Fine Food Reviews

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).

Loading,Cleaning & Preprocessing 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 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 3. If the score id above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

```
%matplotlib inline
import warnings
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
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

warnings.filterwarnings("ignore")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testing is d
  import pandas.util.testing as tm
```

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pf

Enter your authorization code:
··········
Mounted at /content/drive
```

```
con = sqlite3.connect("/content/drive/My Drive/Colab Notebooks/database.sqlite")

filtered_data=pd.read_sql_query("""SELECT * FROM Reviews WHERE Score != 3""",con);
filtered_data.head(3)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Scor
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	

```
filtered_data.shape

def partition(x):
  if x < 3 :
    return 'negative'
  return 'positive'

actualScore=filtered_data['Score']
positive_negative=actualScore.map(partition)
filtered_data['Score']=positive_negative
print("Number of datapoints",filtered_data.shape)
filtered_data.head(3)
```

```
Number of datapoints (525814, 10)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Sc
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	posi
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	nega
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	posi

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

```
print(display.shape)
display.head(3)
```

```
(80668, 7)
```

	UserId	ProductId	ProfileName	Time	Score	
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering t
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle sp
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortuna

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

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2	5	1199

```
sorted_data=filtered_data.sort_values('ProductId',axis=0,ascending=True,inplace=False,kind='quicksort',na_positior

final_data=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"},keep='first',inplace=False)
final_data.shape
```

```
(364173, 10)
```

```
(final_data['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

```
69.25890143662969
```

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

display.head()
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1	5	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2	4	1

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

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

```
stopping_words = set(stopwords.words('english'))
print(stopping_words)
```

```
{'before', 'ourselves', "couldn't", 'didn', 'the', 'some', "aren't", 'itself', 'can', 'that', 'them', 'did', 'sh
```

```
def clean_html(text):
    clean_r = re.compile('<,*?>')
    clean_text = re.sub(clean_r,'',text)
    return clean_text

def Clean_punc(text):
    clean_sentence = re.sub(r'[?|!|\'|"|#]',r' ',text)
    clean_data = re.sub(r'[.|,|)|(|\|/)]',r' ',clean_sentence)
    return clean_data
```

```
from tqdm import tqdm
import os
import pdb
import pickle

from tqdm import tqdm
import os
import pdb
import pickle

stem_no = nltk.stem.SnowballStemmer('english')

if not os.path.isfile('final_data.sqlite'):
    final_string=[]
```

```
all_positive_words=[]
all_negative_words=[]
for i,sentence in enumerate(tqdm(final_data['Text'].values)):
    filtered_sentence=[]
    sent_without_html_tags=clean_html(sentence)
    #pdb.set_trace()
    for w in sent_without_html_tags.split():
        for cleaned_words in Clean_punc(w).split():
            if ((cleaned_words.isalpha()) & (len(cleaned_words) > 2)):
                if(cleaned_words.lower() not in stopping_words) :
                    stemming=(stem_no.stem(cleaned_words.lower())).encode('utf8')
                    filtered_sentence.append(stemming)
                    if(final_data['Score'].values)[i]=='positive':
                        all_positive_words.append(stemming)
                    if(final_data['Score'].values)[i]=='negative':
                        all_negative_words.append(stemming)
    str1 = b" ".join(filtered_sentence)
    final_string.append(str1)

final_data['Cleaned_text']=final_string
final_data['Cleaned_text']=final_data['Cleaned_text'].str.decode("utf-8")

conn = sqlite3.connect('final_data.sqlite')
cursor=conn.cursor
conn.text_factory = str
final_data.to_sql('Reviews',conn,schema=None,if_exists='replace',index=True,index_label=None,chunksize=None,dt
conn.close()


with open('positive_words.pkl','wb') as f :
    pickle.dump(all_positive_words,f)
with open('negative_words.pkl','wb') as f :
    pickle.dump(all_negative_words,f)
```

```
100%|██████████| 364171/364171 [06:05<00:00, 996.62it/s]
```

```
final_data['total_words'] = [len(x.split()) for x in final_data['Cleaned_text'].tolist()]
```

```
final_data.sort_values(by=['Time'], inplace=True, ascending=True)
```

```
final_data.head(3)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	0	positive
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2	2	positive
417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0	0	positive

```
final_data['Score'].value_counts()
```

```
positive    307061
negative     57110
Name: Score, dtype: int64
```

```
count_positive,count_negative=final_data['Score'].value_counts()
```

```
count_positive
```

```
307061
```

```
final_data_positive_class=final_data[final_data['Score']=='positive']
final_data_negative_class=final_data[final_data['Score']=='negative']
```

```
final_data_negative_class.shape
```

```
(57110, 12)
```

RANDOM DOWN SAMPLING

Note : In Down Sampling , there will be loss of information. Since we are removing the random records majority class

```
final_data_positive=final_data_positive_class.sample(count_negative)

fina_data_after_Sampling=pd.concat([final_data_positive,final_data_negative_class], axis=0)
```

```
fina_data_after_Sampling['Score'].value_counts()
```

```
negative    57110
positive    57110
Name: Score, dtype: int64
```

```
fina_data_after_Sampling.shape
```

```
(114220, 12)
```

```
final_data_100K=fina_data_after_Sampling[0:100000]
amazon_polarity_labels=final_data_100K['Score'].values
final_data_100K.head(2)
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Scor
372042	402328	B0051S7P54	A29CNJP06GO2N1	Sandy	1	1	positiv
436815	472353	B001JG537O	AWX2C6PPAMFX0	Melissa	1	1	positiv

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from collections import Counter
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

X_1,X_Test,Y_1,Y_Test = train_test_split(final_data_100K,amazon_polarity_labels,test_size=0.2,random_state=0)
X_Train,X_CV,Y_Train,Y_CV = train_test_split(X_1,Y_1,test_size=0.2)
```

APPLY BAG OF WORDS VECTORIZATION TECHNIQUE USING DECISION TREE CLASSIFIE THE BEST DEPTH

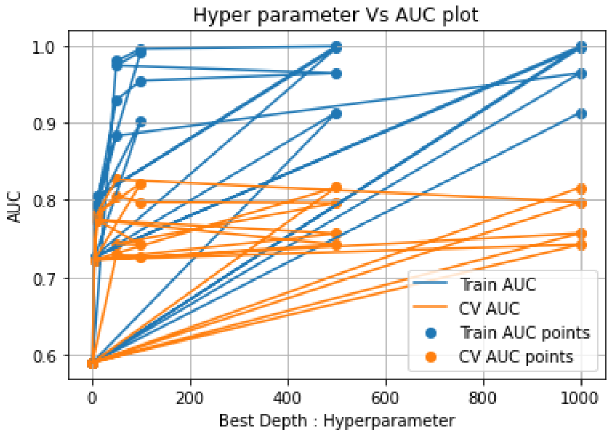
```
print(X_Train.shape, Y_Train.shape)
print(X_CV.shape, Y_CV.shape)
print(X_Test.shape, Y_Test.shape)

print("="*100)


count_vector=CountVectorizer(min_df=1)
X_Train_data_bow=(count_vector.fit_transform(X_Train['Cleaned_text'].values))
X_Test_data_bow=(count_vector.transform(X_Test['Cleaned_text'].values))
X_CV_data_bow=(count_vector.transform(X_CV['Cleaned_text'].values))

print("After vectorizations")
print(X_Train_data_bow.shape, Y_Train.shape)
print(X_CV_data_bow.shape, Y_CV.shape)
print(X_Test_data_bow.shape, Y_Test.shape)
print("="*100)
```

```
(64000, 12) (64000,)
(16000, 12) (16000,)
(20000, 12) (20000,)
====================================================================================================
After vectorizations
(64000, 30582) (64000,)
(16000, 30582) (16000,)
(20000, 30582) (20000,)
====================================================================================================
```

```
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

def Decision_tree_Classifier(x_training_data,y_training_data):
  grid_params = { 'max_depth' : [1,5,10,50,100,500,1000],
                  'min_samples_split' : [5,10,100,500]
                }
  Classifier_DT = DecisionTreeClassifier(random_state=None, class_weight ='balanced')
  clf=GridSearchCV(Classifier_DT,grid_params,scoring='roc_auc',return_train_score=True,cv=10)
  clf.fit(x_training_data,y_training_data)
  results = pd.DataFrame.from_dict(clf.cv_results_)
  results = results.sort_values(['param_max_depth'])
  results = results.sort_values(['param_min_samples_split'])
  train_auc= results['mean_train_score']
  train_auc_std= results['std_train_score']
  cv_auc = results['mean_test_score']
  cv_auc_std= results['std_test_score']
  best_depth =  results['param_max_depth']
  min_sample_split =  results['param_min_samples_split']
  #log_alpha=np.log10(list(results["param_alpha"]))
  print(clf.best_score_)
  print(clf.best_params_)
  plt.plot(best_depth, train_auc, label='Train AUC')
  plt.plot(best_depth, cv_auc, label='CV AUC')
  plt.scatter(best_depth, train_auc, label='Train AUC points')
  plt.scatter(best_depth, cv_auc, label='CV AUC points')
  plt.legend()
  plt.xlabel("Best Depth : Hyperparameter")
  plt.ylabel("AUC")
  plt.title("Hyper parameter Vs AUC plot")
  plt.grid()
  plt.show()
  return results,clf,min_sample_split,Classifier_DT
```

```
print ('-------------BEST DEPTH USING DIFFERENT RANGE OF SAMPLE SPLIT')
result,best_depth,min_sample_split,decision_tree=Decision_tree_Classifier(X_Train_data_bow,Y_Train)
```

```
-------------BEST DEPTH USING DIFFERENT RANGE OF SAMPLE SPLIT
0.8269227163421584
{'max_depth': 50, 'min_samples_split': 500}
```




```
pip install graphviz
```

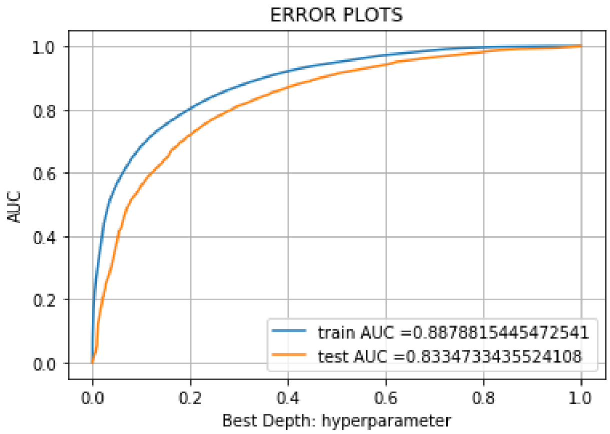
```
Requirement already satisfied: graphviz in /usr/local/lib/python3.6/dist-packages (0.10.1)
```

```
def Find_best_Depth(best_depth) :
  best_depth = best_depth.best_params_
  best_depth=best_depth.get("max_depth")
  print(best_depth)
  return best_depth
```

```
best_depth = Find_best_Depth(best_depth)
```

```
50
```

```
from sklearn.metrics import roc_curve, auc

decision_tree= DecisionTreeClassifier(max_depth=50,random_state=None, class_weight ='balanced',min_samples_split=5
clf=decision_tree.fit(X_Train_data_bow,Y_Train)
pred_test_data=decision_tree.predict(X_Test_data_bow)
y_train_predicted_prob = decision_tree.predict_proba(X_Train_data_bow)[:,1]
y_test_predicted_prob=decision_tree.predict_proba(X_Test_data_bow)[:,1]
train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train,y_train_predicted_prob,pos_label='positive')
test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test, y_test_predicted_prob,pos_label='positive')
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("Best Depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



FIND THE TOP 10 POSITIVE & NEGATIVE FEATURE IMPORTANCE

```
feature_names = count_vector.get_feature_names()
coef =decision_tree.feature_importances_
coefs_with_fns = sorted(zip(coef, feature_names))
top_features = zip(coefs_with_fns[:20], coefs_with_fns[:-(20 + 1):-1])
list(top_features)
```

```
[((0.0, 'aaa'), (0.1236684175187976, 'great')),
 ((0.0, 'aaaaaah'), (0.07609134152081516, 'disappoint')),
 ((0.0, 'aaaaaahhhhhyaaaaaa'), (0.07222343378029503, 'love')),
 ((0.0, 'aaaahhhhhhhhhhh'), (0.05697057925290443, 'best')),
 ((0.0, 'aaaand'), (0.04649185534002008, 'delici')),
 ((0.0, 'aaah'), (0.03780892096838088, 'bad')),
 ((0.0, 'aaahhhhhh'), (0.029834880209780194, 'perfect')),
 ((0.0, 'aachen'), (0.025822645675856973, 'good')),
 ((0.0, 'aad'), (0.022588207511681016, 'thought')),
 ((0.0, 'aadult'), (0.021714440275189956, 'favorit')),
 ((0.0, 'aaf'), (0.0206032805560982, 'excel')),
 ((0.0, 'aafco'), (0.0190078941097605, 'return')),
 ((0.0, 'aah'), (0.018835473774696902, 'money')),
 ((0.0, 'aappubl'), (0.016752904967661568, 'tast')),
 ((0.0, 'aar'), (0.014240257776803395, 'nice')),
 ((0.0, 'aarp'), (0.010728983483504234, 'review')),
 ((0.0, 'aarrgh'), (0.010648035010274169, 'tasti')),
 ((0.0, 'aback'), (0.010434062444733945, 'horribl')),
 ((0.0, 'abalon'), (0.009822693535309353, 'worst')),
 ((0.0, 'abandon'), (0.009342407470265334, 'amaz'))]
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report,confusion_matrix

roc_auc_score(Y_Test,y_test_predicted_prob)
```

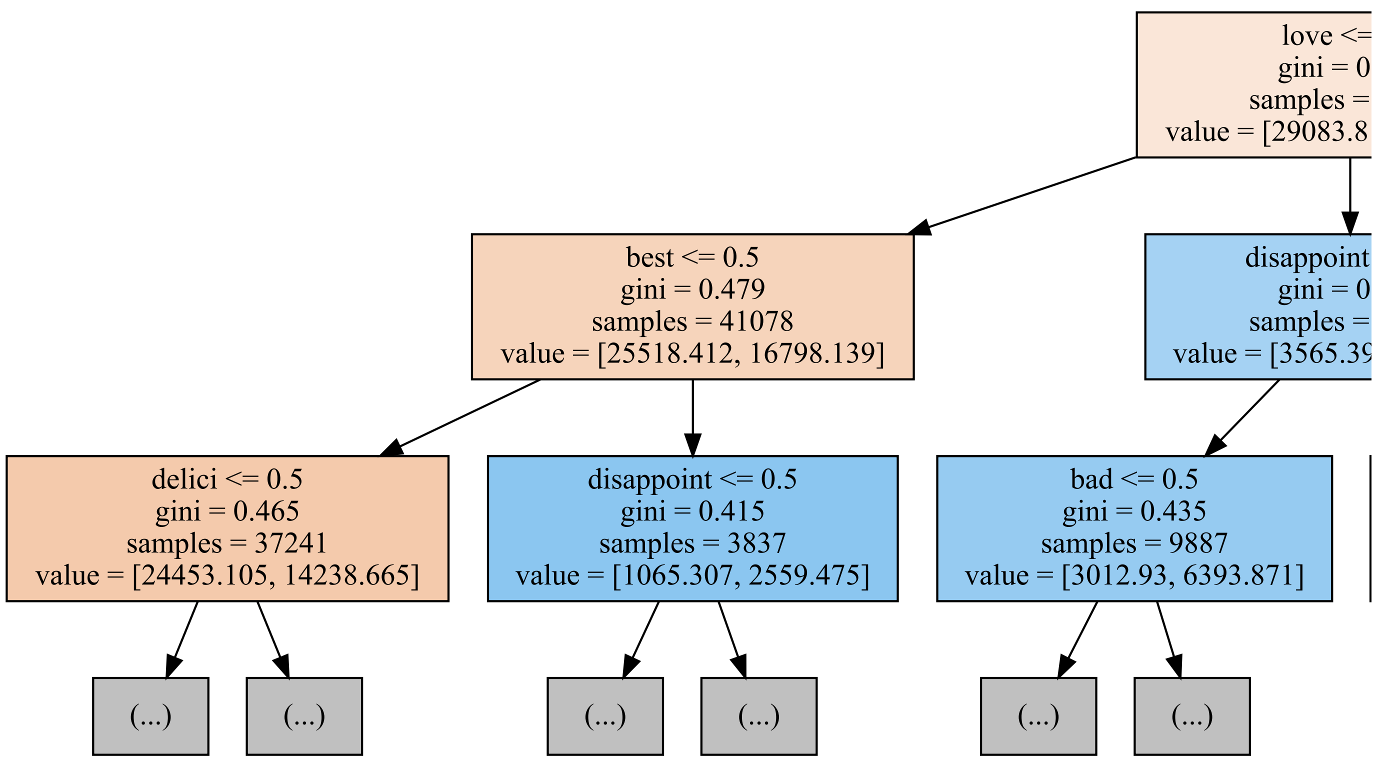
```
0.8334733435524108
```

```
print(classification_report(Y_Test,pred_test_data))
print(confusion_matrix(Y_Test,pred_test_data))
```

```
              precision    recall  f1-score   support

    negative       0.70      0.77      0.73      8582
    positive       0.81      0.75      0.78     11418

    accuracy                           0.76     20000
   macro avg       0.76      0.76      0.76     20000
weighted avg       0.76      0.76      0.76     20000

[[6640 1942]
 [2868 8550]]
```

VISUALIZE THE DECISION TREE USING GRAPHVIZ

```
import graphviz
from sklearn.tree import export_graphviz

export_graphviz(decision_tree, out_file="my_bow_tree.dot", feature_names = count_vector.get_feature_names(), max_c
with open("my_bow_tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```



TF-IDF VECTORIZATION TECHNIQUE USING DECISION TREE

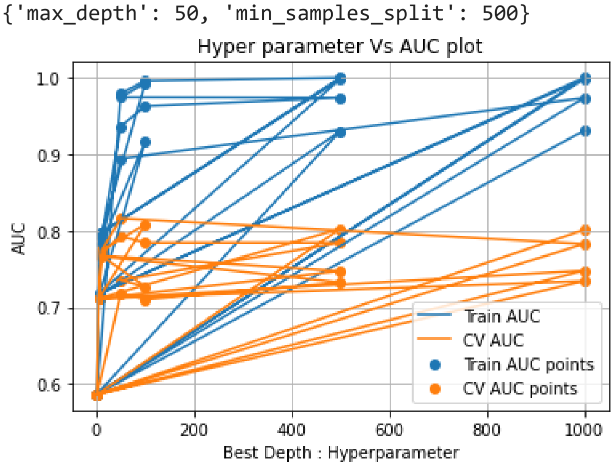
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
print(X_Train.shape, Y_Train.shape)
print(X_CV.shape, Y_CV.shape)
print(X_Test.shape, Y_Test.shape)

print("="*100)


tfidf_vector=TfidfVectorizer(min_df=10)
X_Train_data_tfidf=(tfidf_vector.fit_transform(X_Train['Cleaned_text'].values))
X_Test_data_tfidf=(tfidf_vector.transform(X_Test['Cleaned_text'].values))
X_CV_data_tfidf=(tfidf_vector.transform(X_CV['Cleaned_text'].values))

print("After vectorizations")
print(X_Train_data_tfidf.shape, Y_Train.shape)
print(X_CV_data_tfidf.shape, Y_CV.shape)
print(X_Test_data_tfidf.shape, Y_Test.shape)
print("="*100)
```

```
(64000, 12) (64000,)
(16000, 12) (16000,)
(20000, 12) (20000,)
====================================================================================================
After vectorizations
(64000, 6982) (64000,)
(16000, 6982) (16000,)
(20000, 6982) (20000,)
====================================================================================================
```

```
print ('-------------BEST DEPTH USING DIFFERENT RANGE OF SAMPLE SPLIT')
result,best_depth,min_sample_split,decision_tree_tfidf=Decision_tree_Classifier(X_Train_data_tfidf,Y_Train)
```

```
-------------BEST DEPTH USING DIFFERENT RANGE OF SAMPLE SPLIT
0.8161551568177459
{'max_depth': 50, 'min_samples_split': 500}
```

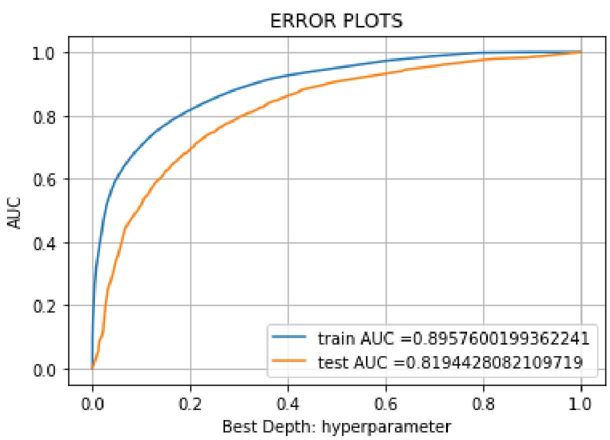



```
best_depth = Find_best_Depth(best_depth)
```

```
50
```

```
from sklearn.metrics import roc_curve, auc

decision_tree= DecisionTreeClassifier(max_depth=50,random_state=None, class_weight ='balanced',min_samples_split=5
clf=decision_tree.fit(X_Train_data_tfidf,Y_Train)
pred_test_data=decision_tree.predict(X_Test_data_tfidf)
y_train_predicted_prob = decision_tree.predict_proba(X_Train_data_tfidf)[:,1]
y_test_predicted_prob=decision_tree.predict_proba(X_Test_data_tfidf)[:,1]
train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train,y_train_predicted_prob,pos_label='positive')
test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test, y_test_predicted_prob,pos_label='positive')
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("Best Depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
feature_names = tfidf_vector.get_feature_names()
coef =decision_tree.feature_importances_
coefs_with_fns = sorted(zip(coef, feature_names))
top_features = zip(coefs_with_fns[:20], coefs_with_fns[:-(20 + 1):-1])
list(top_features)
```

```
[((0.0, 'abandon'), (0.119583149703108, 'great')),
 ((0.0, 'abdomin'), (0.07876461867874446, 'love')),
 ((0.0, 'abil'), (0.06551048114796722, 'disappoint')),
 ((0.0, 'abnorm'), (0.05513699661609732, 'best')),
 ((0.0, 'abomin'), (0.043514076712564805, 'delici')),
 ((0.0, 'abroad'), (0.031624697865599585, 'perfect')),
 ((0.0, 'absenc'), (0.02855670121807223, 'good')),
 ((0.0, 'absent'), (0.024225330947117193, 'bad')),
 ((0.0, 'absolut'), (0.02338306543496268, 'favorit')),
 ((0.0, 'absorb'), (0.020743958280810784, 'excel')),
 ((0.0, 'absorpt'), (0.018155354606437885, 'return')),
 ((0.0, 'absurd'), (0.016899149378417147, 'thought')),
 ((0.0, 'abund'), (0.016727685635741073, 'nice')),
 ((0.0, 'abus'), (0.013437025603691977, 'wast')),
 ((0.0, 'acai'), (0.013367511071551532, 'tast')),
 ((0.0, 'accent'), (0.012387272541635018, 'easi')),
 ((0.0, 'access'), (0.01150590456613557, 'aw')),
 ((0.0, 'accessori'), (0.010232869023075354, 'keep')),
 ((0.0, 'accid'), (0.009755070269865338, 'horribl')),
 ((0.0, 'accident'), (0.009202447902923704, 'amaz'))]
```

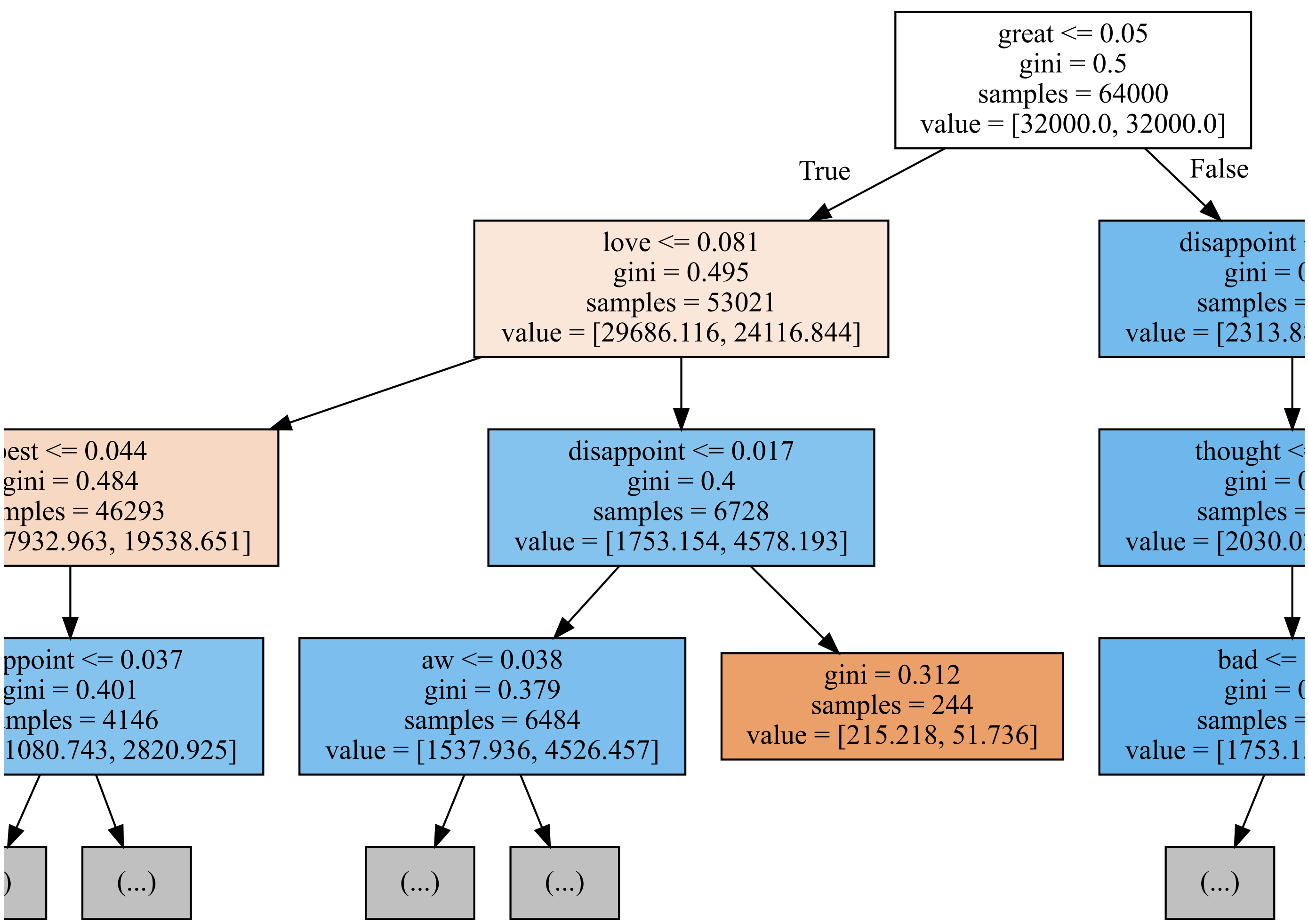
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report,confusion_matrix

roc_auc_score(Y_Test,y_test_predicted_prob)
```

```
0.8194428082109719
```

```
print(classification_report(Y_Test,pred_test_data))
print(confusion_matrix(Y_Test,pred_test_data))
```

```
              precision    recall  f1-score   support

    negative       0.69      0.76      0.72      8582
    positive       0.81      0.74      0.77     11418

    accuracy                           0.75     20000
   macro avg       0.75      0.75      0.75     20000
weighted avg       0.76      0.75      0.75     20000

[[6563 2019]
 [2986 8432]]
```

```
import graphviz
from sklearn.tree import export_graphviz

export_graphviz(decision_tree, out_file="my_tfidf_tree.dot", feature_names = tfidf_vector.get_feature_names(), max
with open("my_tfidf_tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```



Avg Word2Vec Vectorization Technique on Decision Tree Algorithm

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

list_of_sent_train_avgw2v=[]
```

```
list_of_sent_test_avgw2v=[]
list_of_sent_cv_avgw2v=[]
for sent_train_avgw2v in tqdm(X_Train['Cleaned_text'].values):
    list_of_sent_train_avgw2v.append(sent_train_avgw2v.split())
```

```
100%|██████████| 64000/64000 [00:00<00:00, 83969.58it/s]
```

```
for sent_test_avgw2v in tqdm(X_Test['Cleaned_text'].values):
    list_of_sent_test_avgw2v.append(sent_test_avgw2v.split())

for sent_cv_avgw2v in tqdm(X_CV['Cleaned_text'].values):
    list_of_sent_cv_avgw2v.append(sent_cv_avgw2v.split())
```

```
100%|██████████| 20000/20000 [00:00<00:00, 156933.10it/s]
100%|██████████| 16000/16000 [00:00<00:00, 176717.61it/s]
```

```
w2v_model_train = Word2Vec(list_of_sent_train_avgw2v,min_count=5,size=50,workers=4)
w2v_words_svm_train=list(w2v_model_train.wv.vocab)
```

```
w2v_model_test = Word2Vec(list_of_sent_test_avgw2v,min_count=5,size=50,workers=4)
w2v_words_svm_test=list(w2v_model_test.wv.vocab)
```

```
w2v_model_cv = Word2Vec(list_of_sent_cv_avgw2v,min_count=5,size=50,workers=4)
w2v_words_svm_cv=list(w2v_model_cv.wv.vocab)
```

```
train_vectors=[];
for sent in list_of_sent_train_avgw2v:
    sent_vec=np.zeros(50)
    cnt_words=0;
    for word in sent:
        if word in w2v_words_svm_train:
            vec=w2v_model_train.wv[word]
            sent_vec+=vec
            cnt_words+=1
    if cnt_words !=0:
        sent_vec/=cnt_words
    train_vectors.append(sent_vec)
print(len(train_vectors))
print(len(train_vectors[0]))
```

```
64000
50
```

```
test_vectors=[];
for sent in tqdm(list_of_sent_test_avgw2v):
    sent_vec=np.zeros(50)
    cnt_words=0;
    for word in sent:
        if word in w2v_words_svm_test:
            vec=w2v_model_test.wv[word]
            sent_vec+=vec
            cnt_words+=1
    if cnt_words !=0:
        sent_vec/=cnt_words
    test_vectors.append(sent_vec)
print(len(test_vectors))
print(len(test_vectors[0]))
```

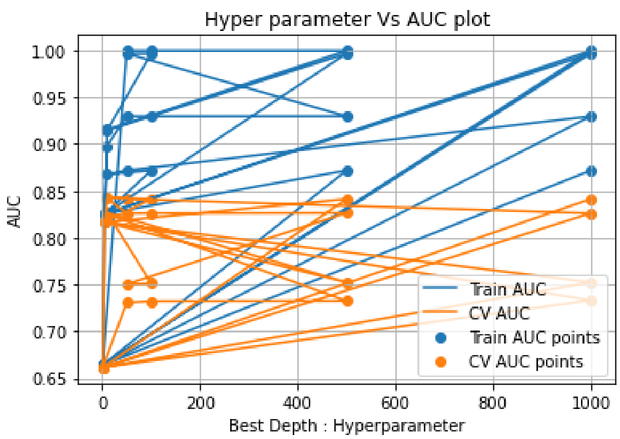
```
100%|██████████| 20000/20000 [00:16<00:00, 1183.73it/s]20000
50
```

```
cv_vectors=[];
for sent in tqdm(list_of_sent_cv_avgw2v):
    sent_vec=np.zeros(50)
    cnt_words=0;
    for word in sent:
        if word in w2v_words_svm_cv:
            vec=w2v_model_cv.wv[word]
            sent_vec+=vec
            cnt_words+=1
    if cnt_words !=0:
        sent_vec/=cnt_words
    cv_vectors.append(sent_vec)
print(len(cv_vectors))
print(len(cv_vectors[0]))
```

```
100%|██████████| 16000/16000 [00:13<00:00, 1203.18it/s]16000
50
```

```
print ('-------------BEST DEPTH USING DIFFERENT RANGE OF SAMPLE SPLIT')
result,best_depth,min_sample_split,decision_tree=Decision_tree_Classifier(train_vectors,Y_Train)
```

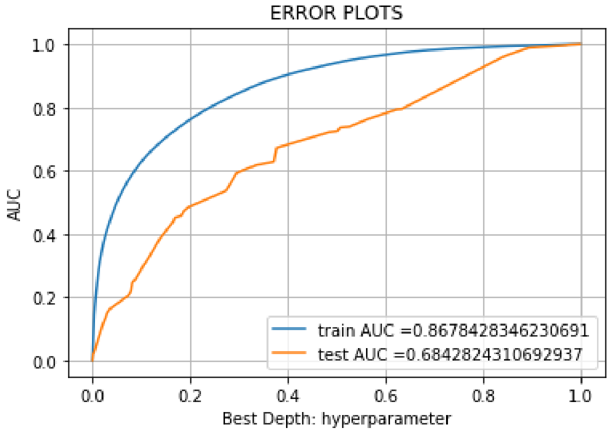
```
-------------BEST DEPTH USING DIFFERENT RANGE OF SAMPLE SPLIT
0.8429747830570099
{'max_depth': 10, 'min_samples_split': 500}
```




```
best_depth = Find_best_Depth(best_depth)
```

```
10
```

```
from sklearn.metrics import roc_curve, auc

decision_tree= DecisionTreeClassifier(max_depth=10,random_state=None, class_weight ='balanced',min_samples_split=5
clf=decision_tree.fit(train_vectors,Y_Train)
pred_test_data=decision_tree.predict(test_vectors)
y_train_predicted_prob = decision_tree.predict_proba(train_vectors)[:,1]
y_test_predicted_prob=decision_tree.predict_proba(test_vectors)[:,1]
train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train,y_train_predicted_prob,pos_label='positive')
test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test, y_test_predicted_prob,pos_label='positive')
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("Best Depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import classification_report,confusion_matrix

roc_auc_score(Y_Test,y_test_predicted_prob)
```

```
0.6842824310692937
```

```
print(classification_report(Y_Test,pred_test_data))
print(confusion_matrix(Y_Test,pred_test_data))
```

```
              precision    recall  f1-score   support

    negative       0.50      0.89      0.64      8582
    positive       0.79      0.32      0.46     11418

    accuracy                           0.56     20000
   macro avg       0.64      0.60      0.55     20000
weighted avg       0.66      0.56      0.53     20000

[[7611  971]
 [7761 3657]]
```

TF-IDF Word2Vec Vectorization Technique for DECISION TREE on Amazon Fine Food Rev

```
model_Avgw2v = TfidfVectorizer()
X_Train_Avgw2v=model_Avgw2v.fit_transform(X_Train['Cleaned_text'].values)
```

```
X_Test_Avgw2v=model_Avgw2v.transform(X_Test['Cleaned_text'].values)
X_CV_Avgw2v=model_Avgw2v.transform(X_CV['Cleaned_text'].values)
```

```
dictionary = dict(zip(model_Avgw2v.get_feature_names(), list(model_Avgw2v.idf_)))
```

```
tfidf_feature=model_Avgw2v.get_feature_names()

tfidf_sent_vectors_train=[];
#final_tf_idf = [];
row=0;

for sent in tqdm(list_of_sent_train_avgw2v):
    sent_vec=np.zeros(50)
    weight_sum=0;
    for word in sent :
        if word in w2v_words_svm_train and word in tfidf_feature :
            vec=w2v_model_train.wv[word]
```

```
            #tf_idf=final_tf_idf[row,tfidf_feature.index(word)]
            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_train.append(sent_vec)
    row+=1
```

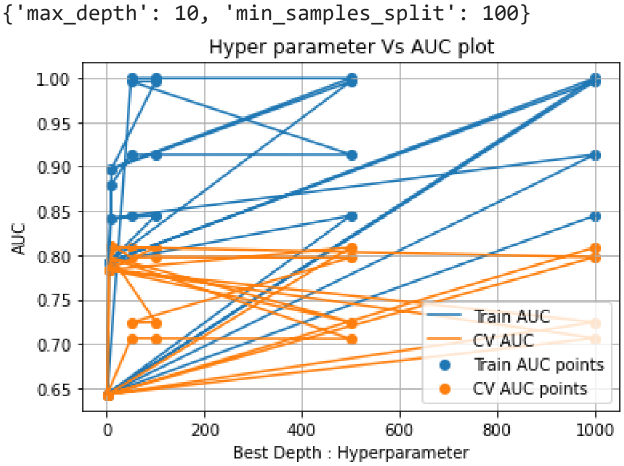
```
100%|██████████| 64000/64000 [14:31<00:00, 73.43it/s]
```

```
tfidf_sent_vectors_test=[];
#final_tf_idf = [];
row=0;

for sent in tqdm(list_of_sent_test_avgw2v):
    sent_vec=np.zeros(50)
    weight_sum=0;
    for word in sent :
        if word in w2v_words_svm_test and word in tfidf_feature :
            vec=w2v_model_test.wv[word]
            #tf_idf=final_tf_idf[row,tfidf_feature.index(word)]
            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_test.append(sent_vec)
    row+=1
```

```
100%|██████████| 20000/20000 [04:19<00:00, 76.92it/s]
```

```
print ('-------------BEST DEPTH USING DIFFERENT RANGE OF SAMPLE SPLIT')
result,best_depth,min_sample_split,decision_tree=Decision_tree_Classifier(tfidf_sent_vectors_train,Y_Train)
```

```
-------------BEST DEPTH USING DIFFERENT RANGE OF SAMPLE SPLIT
0.811198148361221
{'max_depth': 10, 'min_samples_split': 100}
```

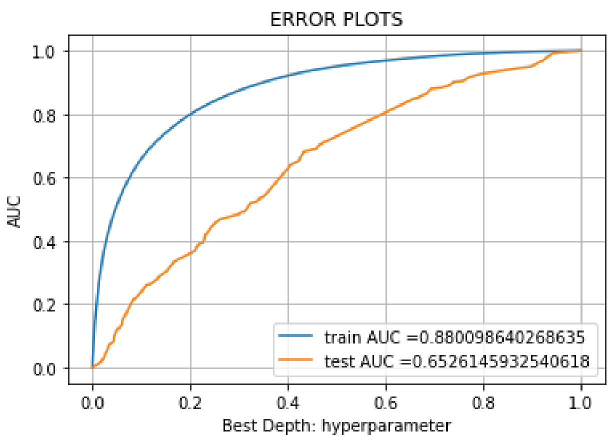



```
best_depth = Find_best_Depth(best_depth)
```

```
10
```

```
from sklearn.metrics import roc_curve, auc

decision_tree= DecisionTreeClassifier(max_depth=10,random_state=None, class_weight ='balanced',min_samples_split=1
```

```
clf=decision_tree.fit(tfidf_sent_vectors_train,Y_Train)
pred_test_data=decision_tree.predict(tfidf_sent_vectors_test)
y_train_predicted_prob = decision_tree.predict_proba(tfidf_sent_vectors_train)[:,1]
y_test_predicted_prob=decision_tree.predict_proba(tfidf_sent_vectors_test)[:,1]
train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train,y_train_predicted_prob,pos_label='positive')
test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test, y_test_predicted_prob,pos_label='positive')
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("Best Depth: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
roc_auc_score(Y_Test,y_test_predicted_prob)
```

```
0.6526145932540618
```

```
print(classification_report(Y_Test,pred_test_data))
print(confusion_matrix(Y_Test,pred_test_data))
```

```
              precision    recall  f1-score   support

    negative       0.50      0.77      0.60      8582
    positive       0.71      0.42      0.53     11418

    accuracy                           0.57     20000
   macro avg       0.60      0.59      0.57     20000
weighted avg       0.62      0.57      0.56     20000

[[6583 1999]
 [6600 4818]]
```

PRETTY TABLE

```
pip install -U PTable
```

```
Collecting PTable
```

```
from prettytable import PrettyTable

x= PrettyTable()
x.field_names = ["Vectorizer" , "Hyperparameter(Best Depth)", "Minimum Sample Split","AUC"]
x.add_row(["Bag Of Words",50,500,0.8296])
x.add_row(["Tf-Idf",50,500,0.8161])
x.add_row(["Avg Word2Vec",10,500,0.8429])
x.add_row(["Tf-Idf Word2Vec",10,100,0.8111])
print(x)
```

```
+-----------------+----------------------------+----------------------+--------+
|    Vectorizer   | Hyperparameter(Best Depth) | Minimum Sample Split |  AUC   |
+-----------------+----------------------------+----------------------+--------+
|   Bag Of Words  |             50             |         500          | 0.8296 |
|      Tf-Idf     |             50             |         500          | 0.8161 |
|   Avg Word2Vec  |             10             |         500          | 0.8429 |
| Tf-Idf Word2Vec |             10             |         100          | 0.8111 |
+-----------------+----------------------------+----------------------+--------+
```

Observation :

1) The AUC Score for Avg Word2Vec is 0.8429 with Hyperparameter = 10 & Minimum Sample Split = 500

2) To Balanced the Dataset , i have used Undersampling technique.

3) THe Important Features does not show up the top 10 negative features

```

```