SUPPORT VECTOR MACHINE 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".

Import all Required Libraries

```
%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 c
  import pandas.util.testing as tm
```

Pull the dataset from Google Drive & mount

```
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
```

Connect to Database and execute the query

```
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	

Data Cleaning & Preparation

```
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)
```

```
{'itself', 'against', 'here', "didn't", 'myself', 'under', "you're", 'most', 'no', 'all', "should've", 'will', '
```

```
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 [06:07<00:00, 991.13it/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_100K=final_data[0:100000]
amazon_polarity_labels=final_data_100K['Score'].values
final_data_100K.head(2)
```

	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

```
final_data_40K = final_data[0:30000]
amazon_polarity_labels_40K = final_data_40K['Score'].values
```

Split the dataset in Train , Test & Cross Validation

```
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 LINEAR SVM (SGDCLASSIF FIND THE BEST ALPHA

```
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, 29463) (64000,)
(16000, 29463) (16000,)
(20000, 29463) (20000,)
====================================================================================================
```

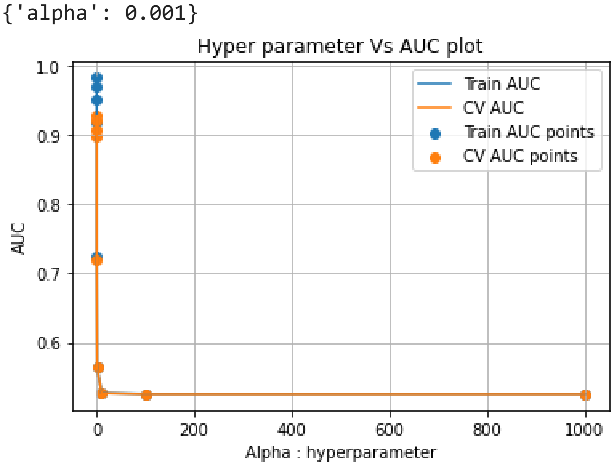
```
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier

def SVM_Linear_SVM(x_training_data,y_training_data,regularization):
  grid_params = { 'alpha' : [10**x for x in range(-4,5)]
               }
  LinearSVM = SGDClassifier(loss='hinge',penalty=regularization,random_state=None, class_weight=None)
  clf=GridSearchCV(LinearSVM,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_alpha'])
```

```
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_alpha =  results['param_alpha']
#print(type(alpha))
#print(alpha)
#log_c=np.log10(list(results["param_C"]))
print(clf.best_score_)
print(clf.best_params_)
plt.plot(best_alpha, train_auc, label='Train AUC')
plt.plot(best_alpha, cv_auc, label='CV AUC')
plt.scatter(best_alpha, train_auc, label='Train AUC points')
plt.scatter(best_alpha, cv_auc, label='CV AUC points')
plt.legend()
plt.xlabel("Alpha : hyperparameter")
plt.ylabel("AUC")
plt.title("Hyper parameter Vs AUC plot")
plt.grid()
plt.show()
return results,clf,LinearSVM
```

Call Linear SVM using L2 Regularization

```
results,best_alpha,linear_svm = SVM_Linear_SVM(X_Train_data_bow,Y_Train,'l2')
results.head()
```

```
0.9266877065638912
{'alpha': 0.001}
```



	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_
0	1.228909	0.056521	0.016896	0.001494	1e-05	{'alpha': 1e-05}	0.905648	
1	0.641934	0.049558	0.016620	0.000650	0.0001	{'alpha': 0.0001}	0.923277	
2	0.377793	0.019492	0.016994	0.001798	0.001	{'alpha': 0.001}	0.930186	
3	0.297809	0.009442	0.016568	0.000745	0.01	{'alpha': 0.01}	0.907816	
4	0.297083	0.006493	0.017016	0.001693	0.1	{'alpha': 0.1}	0.704151	

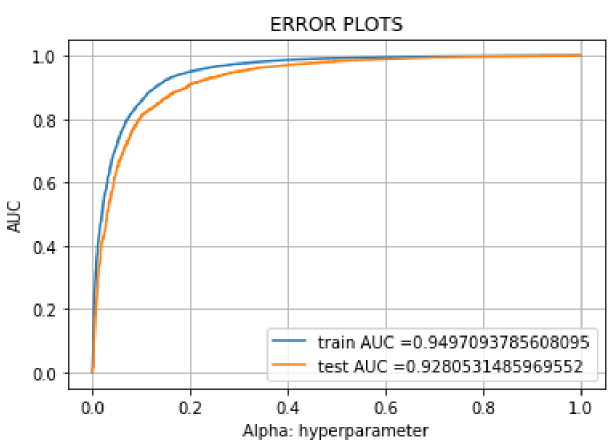
Got Best Alpha = 0.001 and AUC SCORE = 0.9266 ON TRAINING DATA

```
def Find_best_Alpha(best_alpha) :
  best_Alpha = best_alpha.best_params_
  best_alpha=best_Alpha.get("alpha")
  print(best_alpha)
  return best_alpha
```

```
best_alpha = Find_best_Alpha(best_alpha)
```

```
0.001
```

```
from sklearn.metrics import roc_curve, auc
from sklearn.calibration import CalibratedClassifierCV

linear_svm = SGDClassifier(loss='hinge',penalty='l2',random_state=None, class_weight=None,alpha=best_alpha)
clf=linear_svm.fit(X_Train_data_bow,Y_Train)
calibrated_model=CalibratedClassifierCV(clf,cv='prefit')
model_m=calibrated_model.fit(X_Train_data_bow,Y_Train)
pred_test_data=linear_svm.predict(X_Test_data_bow)
y_train_predicted_prob = model_m.predict_proba(X_Train_data_bow)[:,1]
y_test_predicted_prob=model_m.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("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```



ON TEST DATA WE GOT AUC AS 0.9280 FOR ALPHA = 0.001

```
feature_names = count_vector.get_feature_names()
print(feature_names)
coefs_with_fns = sorted(zip(linear_svm.coef_[0], feature_names))
top_features = zip(coefs_with_fns[:20], coefs_with_fns[:-(20 + 1):-1])
list(top_features)
```

```
['aaa', 'aaaaaaaaagghh', 'aaaaah', 'aaaaahhhhhhhhhhhhhhhh', 'aaah', 'aaahhhhhh', 'aachen', 'aacur', 'aadp', 'aaf
[((-1.058942800585861, 'worst'), (0.5322610283239771, 'delici')),
 ((-0.7799799974810454, 'disappoint'), (0.4731209140657581, 'excel')),
 ((-0.7665897829320187, 'terribl'), (0.4641941043664035, 'amaz')),
 ((-0.7152606271607351, 'horribl'), (0.45303559224220846, 'best')),
 ((-0.7074496686738002, 'threw'), (0.4374136752683411, 'perfect')),
 ((-0.698522858974446, 'aw'), (0.4240234607193107, 'great')),
 ((-0.6873643468502516, 'return'), (0.3648833464610924, 'addict')),
 ((-0.6159498692554216, 'tasteless'), (0.3570723879741582, 'nice')),
 ((-0.6025596547063876, 'bland'), (0.3548406855493194, 'awesom')),
 ((-0.5858218865201033, 'stale'), (0.3314078100885153, 'satisfi')),
 ((-0.5032488968010768, 'unfortun'), (0.32582855402641864, 'wonder')),
 ((-0.49990134316381896, 'disgust'), (0.32359685160158047, 'yummi')),
 ((-0.4932062358893016, 'poor'), (0.31578589311464605, 'smooth')),
 ((-0.482047723765109, 'money'), (0.3113224882649689, 'beat')),
 ((-0.4697733604285002, 'stuck'), (0.3057432322028735, 'happi')),
 ((-0.4574989970918883, 'yuck'), (0.30239567856561533, 'hook')),
 ((-0.4563831458794691, 'gross'), (0.30239567856561467, 'tasti')),
 ((-0.4552672946670491, 'nasti'), (0.29904812492835753, 'rich')),
 ((-0.4362978240559228, 'ined'), (0.2990481249283572, 'uniqu')),
 ((-0.4362978240559222, 'weak'), (0.2945847200786804, 'fantast'))]
```

TOP 20 POSITIVE & NEGATIVE FEATURES

```
from sklearn.metrics import roc_auc_score

roc_auc_score(Y_Test,y_test_predicted_prob)
```

```
0.9280531485969552
```

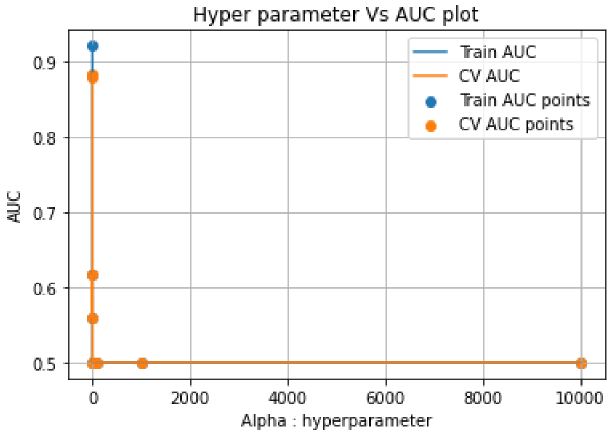
```
from sklearn.metrics import classification_report,confusion_matrix

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

```
              precision    recall  f1-score   support

    negative       0.82      0.44      0.57      2457
    positive       0.93      0.99      0.96     17543

    accuracy                           0.92     20000
   macro avg       0.87      0.71      0.76     20000
weighted avg       0.91      0.92      0.91     20000

[[ 1069  1388]
 [  237 17306]]
```

SVM on Amazon Fine Food Review using L1 Regularization

```
results,best_alpha,linear_svm = SVM_Linear_SVM(X_Train_data_bow,Y_Train,'l1')
results.head()
```

```
0.8839167307373945
{'alpha': 0.0001}
```



	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_
0	4.734316	0.663837	0.019478	0.000709	0.0001	{'alpha': 0.0001}	0.893648	
1	0.740307	0.322096	0.019451	0.000378	0.001	{'alpha': 0.001}	0.881980	
2	0.369847	0.158847	0.020437	0.002277	0.01	{'alpha': 0.01}	0.592772	
3	0.740072	0.042864	0.019065	0.000328	0.1	{'alpha': 0.1}	0.596985	
4	0.413893	0.038416	0.018740	0.000523	1	{'alpha': 1}	0.500000	

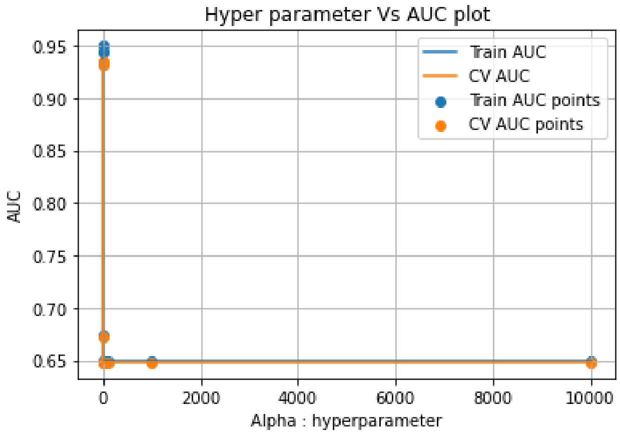
TF-IDF Vectorization Technique on SUPPORT VECTOR MACHINE

```
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, 6775) (64000,)
(16000, 6775) (16000,)
(20000, 6775) (20000,)
====================================================================================================
```

```
results,best_alpha,linear_svm = SVM_Linear_SVM(X_Train_data_tfidf,Y_Train,'l2')
results.head()
```

```
0.9353531913390751
{'alpha': 0.0001}
```



	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_
0	0.200958	0.006349	0.016475	0.001642	0.0001	{'alpha': 0.0001}	0.939161	
1	0.167422	0.002496	0.017399	0.002993	0.001	{'alpha': 0.001}	0.934442	
2	0.151980	0.004264	0.015957	0.000469	0.01	{'alpha': 0.01}	0.934696	
3	0.360621	0.023838	0.015771	0.000320	0.1	{'alpha': 0.1}	0.654612	
4	0.256873	0.003282	0.016171	0.001593	1	{'alpha': 1}	0.629172	

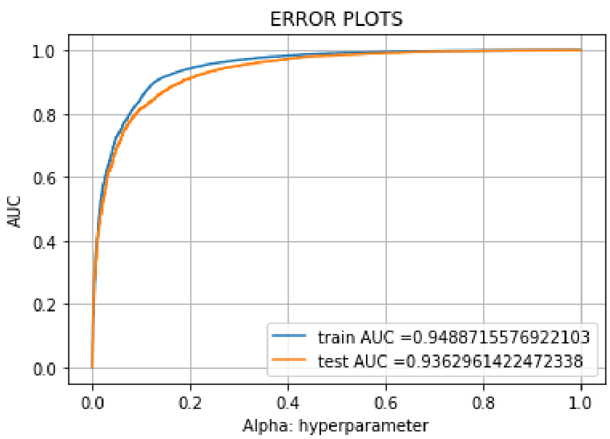
BEST ALPHA = 0.0001 & AUC SCORE = 0.9353 ON TRAINING DATA

```
best_alpha = Find_best_Alpha(best_alpha)
```

```
0.0001
```

```
linear_svm = SGDClassifier(loss='hinge',penalty='l2',random_state=None, class_weight=None,alpha=best_alpha)
clf=linear_svm.fit(X_Train_data_tfidf,Y_Train)
calibrated_model=CalibratedClassifierCV(clf,cv='prefit')
model_m=calibrated_model.fit(X_Train_data_tfidf,Y_Train)
pred_test_data=linear_svm.predict(X_Test_data_tfidf)
y_train_predicted_prob = model_m.predict_proba(X_Train_data_tfidf)[:,1]
y_test_predicted_prob=model_m.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("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
feature_names = tfidf_vector.get_feature_names()
print(feature_names)
coefs_with_fns = sorted(zip(linear_svm.coef_[0], feature_names))
top_features = zip(coefs_with_fns[:20], coefs_with_fns[:-(20 + 1):-1])
list(top_features)
```

```
['abandon', 'abc', 'abdomin', 'abil', 'abl', 'abroad', 'absenc', 'absent', 'absolut', 'absolutley', 'absorb', 'a
[((-3.5252305591227597, 'worst'), (3.1691492148895763, 'great')),
 ((-3.4005030003873884, 'disappoint'), (2.6431849273213213, 'best')),
 ((-3.011837811348974, 'return'), (2.346785762435066, 'love')),
 ((-2.8599641269962075, 'horribl'), (2.2155651378079484, 'delici')),
 ((-2.8490643379944345, 'terribl'), (1.973408817473574, 'good')),
 ((-2.6997285069022863, 'aw'), (1.8981549152660266, 'excel')),
 ((-2.551410076986622, 'threw'), (1.8505124817553666, 'perfect')),
 ((-2.2322630718087475, 'wast'), (1.8235615796747322, 'nice')),
 ((-2.222731395767679, 'stale'), (1.5899723194087558, 'wonder')),
 ((-2.1713121701828735, 'money'), (1.5875181466524377, 'amaz')),
 ((-2.1643257959150217, 'bland'), (1.554797524815107, 'favorit')),
 ((-2.0874293899363803, 'tasteless'), (1.3285533753518113, 'find')),
 ((-1.8424766374606494, 'refund'), (1.2952990206015922, 'awesom')),
 ((-1.8317515775148232, 'disgust'), (1.2663691707134266, 'tasti')),
 ((-1.8199214620059139, 'poor'), (1.2245791948877016, 'addict')),
 ((-1.7322759294298344, 'stuck'), (1.193569113054718, 'smooth')),
 ((-1.7126635922215447, 'weak'), (1.1820805784546944, 'satisfi')),
 ((-1.6849467359259969, 'unfortun'), (1.148515450544414, 'fast')),
 ((-1.5951771543881406, 'gross'), (1.1385650966912313, 'easi')),
 ((-1.564336208325889, 'bad'), (1.1284937586979071, 'yummi'))]
```

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

```
0.9362961422472338
```

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

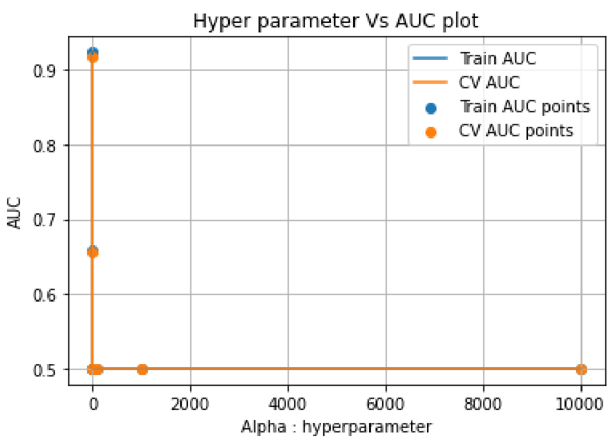
```
              precision    recall  f1-score   support

    negative       0.90      0.33      0.48      2457
    positive       0.91      0.99      0.95     17543

    accuracy                           0.91     20000
   macro avg       0.91      0.66      0.72     20000
weighted avg       0.91      0.91      0.89     20000

[[  804  1653]
 [   90 17453]]
```

AND FOR BEST ALPHA = 0.0001 , AUC SCORE = 0.9326

SVM on Amazon Fine Food Review using L1 Regularization

```
results,best_alpha,linear_svm = SVM_Linear_SVM(X_Train_data_tfidf,Y_Train,'l1')
results.head()
```

```
0.9180460072517856
{'alpha': 0.0001}
```



	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_
0	0.282197	0.004999	0.016205	0.000478	0.0001	{'alpha': 0.0001}	0.921746	
1	0.201957	0.009141	0.016437	0.001334	0.001	{'alpha': 0.001}	0.690765	
2	0.192060	0.007254	0.015663	0.000876	0.01	{'alpha': 0.01}	0.500000	
3	1.452912	0.292737	0.014967	0.000299	0.1	{'alpha': 0.1}	0.500000	
4	0.337768	0.006949	0.016103	0.003331	1	{'alpha': 1}	0.500000	

Avg Word2Vec Vectorization Technique on SUPPORT VECTOR MACHINE 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, 78995.77it/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, 152319.91it/s]
100%|██████████| 16000/16000 [00:00<00:00, 189355.89it/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]))
```

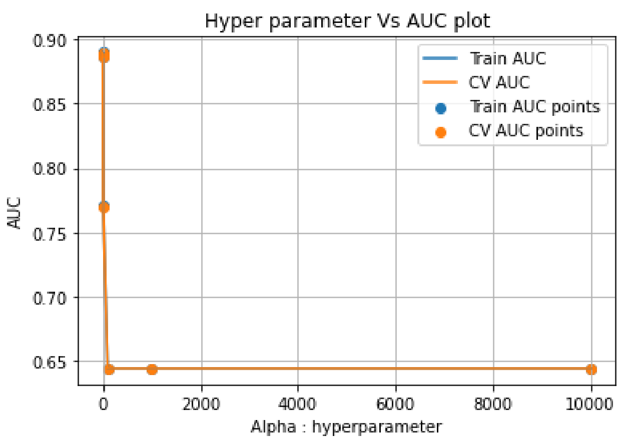
```
100%|██████████| 20000/20000 [00:16<00:00, 1200.95it/s]20000
50
```

```
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 [00:12<00:00, 1257.27it/s]16000
50
```

```
results,best_alpha,linear_svm = SVM_Linear_SVM(train_vectors,Y_Train,'l2')
results.head()
```

```
0.8893952681617303
{'alpha': 0.001}
```



	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_
0	0.464100	0.050165	0.035754	0.003152	0.0001	{'alpha': 0.0001}	0.887549	
1	0.268217	0.012332	0.034349	0.000384	0.001	{'alpha': 0.001}	0.889850	
2	0.218326	0.003064	0.034603	0.000560	0.01	{'alpha': 0.01}	0.890220	
3	0.218255	0.016672	0.035085	0.000583	0.1	{'alpha': 0.1}	0.888572	
4	0.207437	0.010266	0.035847	0.000986	1	{'alpha': 1}	0.880606	

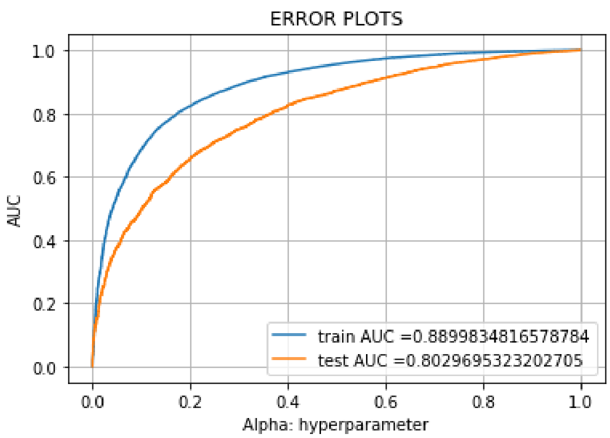
BEST ALPHA = 0.001 WITH AUC = 0.8890 ON TRAINING DATA

```
best_alpha = Find_best_Alpha(best_alpha)
```

```
0.001
```

```
linear_svm = SGDClassifier(loss='hinge',penalty='l2',random_state=None, class_weight=None,alpha=best_alpha)
clf=linear_svm.fit(train_vectors,Y_Train)
```

```
calibrated_model=CalibratedClassifierCV(clf,cv='prefit')
model_m=calibrated_model.fit(train_vectors,Y_Train)
pred_test_data=linear_svm.predict(test_vectors)
y_train_predicted_prob = model_m.predict_proba(train_vectors)[:,1]
y_test_predicted_prob=model_m.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("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```

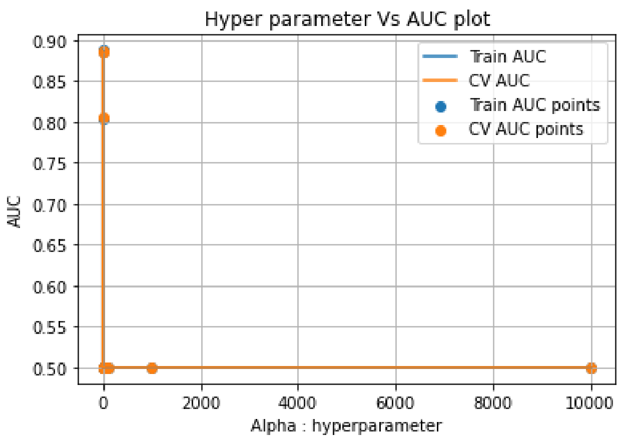


TEST AUC = 0.80326 FOR ALPHA = 0.001

SVM on Amazon Fine Food REview using L1 Regularization

```
results,best_alpha,linear_svm = SVM_Linear_SVM(train_vectors,Y_Train,'l1')
results.head()
```

```
0.8868313361702558
{'alpha': 0.001}
```



	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_
0	0.615362	0.034282	0.029011	0.000364	0.0001	{'alpha': 0.0001}	0.888781	
1	0.338329	0.021439	0.029690	0.001517	0.001	{'alpha': 0.001}	0.887724	
2	0.306859	0.007375	0.029253	0.000761	0.01	{'alpha': 0.01}	0.830911	
3	0.272682	0.004794	0.028153	0.000324	0.1	{'alpha': 0.1}	0.500000	
4	0.279753	0.009730	0.029353	0.001307	1	{'alpha': 1}	0.500000	

TF-IDF Word2Vec Vectorization Technique for SUPPORT VECTOR MACHINE on Amazon Review

```
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
```

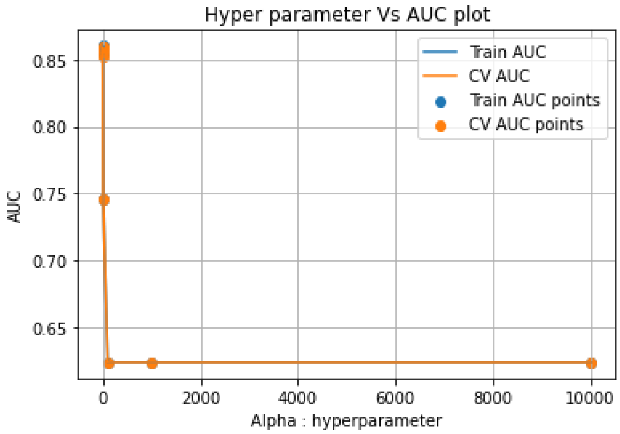
```
100%|██████████| 64000/64000 [14:28<00:00, 73.65it/s]
```

```
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 [04:23<00:00, 75.82it/s]
```

```
results,best_alpha,linear_svm = SVM_Linear_SVM(tfidf_sent_vectors_train,Y_Train,'l2')
results.head()
```

```
0.8593906637211086
{'alpha': 0.001}
```



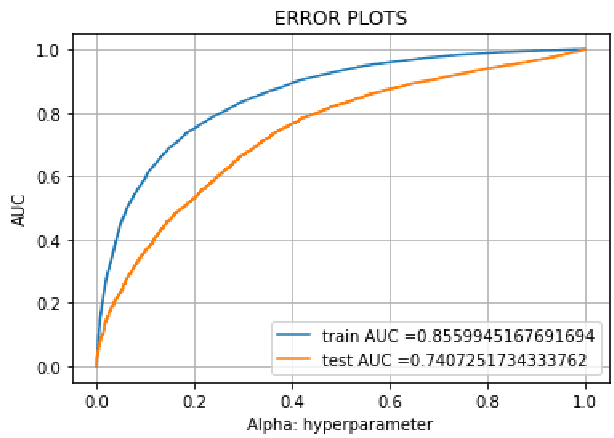
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_
0	0.593249	0.028793	0.035492	0.001220	0.0001	{'alpha': 0.0001}	0.848499	
1	0.310280	0.012341	0.035850	0.002070	0.001	{'alpha': 0.001}	0.856023	
2	0.234241	0.006350	0.035853	0.001414	0.01	{'alpha': 0.01}	0.855193	
3	0.230646	0.004971	0.034734	0.001037	0.1	{'alpha': 0.1}	0.842829	
4	0.212810	0.006187	0.035471	0.001092	1	{'alpha': 1}	0.854022	

BEST ALPHA = 0.001 WITH AUC = 0.8593 ON TRAINING DATA

```
best_alpha = Find_best_Alpha(best_alpha)
```

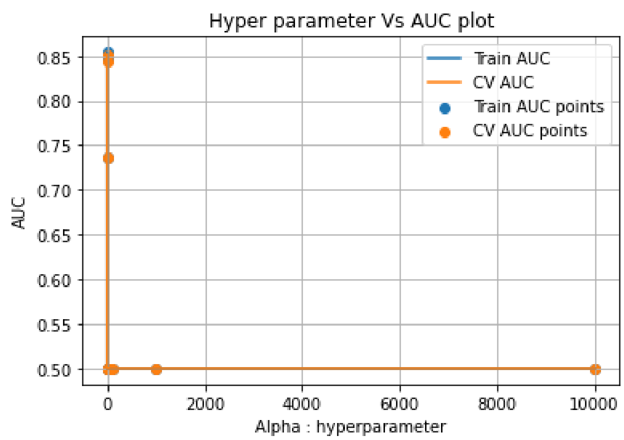
```
0.001
```

```
linear_svm = SGDClassifier(loss='hinge',penalty='l2',random_state=None, class_weight=None,alpha=best_alpha)
clf=linear_svm.fit(tfidf_sent_vectors_train,Y_Train)
calibrated_model=CalibratedClassifierCV(clf,cv='prefit')
model_m=calibrated_model.fit(tfidf_sent_vectors_train,Y_Train)
pred_test_data=linear_svm.predict(tfidf_sent_vectors_test)
y_train_predicted_prob = model_m.predict_proba(tfidf_sent_vectors_train)[:,1]
y_test_predicted_prob=model_m.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("Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
results,best_alpha,linear_svm = SVM_Linear_SVM(tfidf_sent_vectors_train,Y_Train,'l1')
results.head()
```

```
0.8519893373085686
{'alpha': 0.0001}
```



	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_score	split1_
0	0.788327	0.064601	0.029334	0.000754	0.0001	{'alpha': 0.0001}	0.848334	
1	0.390466	0.020528	0.029376	0.001168	0.001	{'alpha': 0.001}	0.843245	
2	0.308332	0.004531	0.029242	0.000428	0.01	{'alpha': 0.01}	0.618617	
3	0.272443	0.007234	0.027860	0.000470	0.1	{'alpha': 0.1}	0.500000	
4	0.271363	0.005789	0.028100	0.000459	1	{'alpha': 1}	0.500000	

IMPLEMENT SUPPORT VECTOR MACHINE USING RBF KERNEL

SPLIT THE DATA INTO TRAIN , TEST & CV

```
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_RBF,Y_1,Y_Test_RBF = train_test_split(final_data_40K,amazon_polarity_labels_40K,test_size=0.2,random_st
X_Train_RBF,X_CV_RBF,Y_Train_RBF,Y_CV_RBF = train_test_split(X_1,Y_1,test_size=0.2)
```

```
print(X_Train_RBF.shape, Y_Train_RBF.shape)
print(X_CV_RBF.shape, Y_CV_RBF.shape)
print(X_Test_RBF.shape, Y_Test_RBF.shape)

print("="*100)


count_vector=CountVectorizer(min_df=10,max_features=500)
X_Train_data_bow_rbf=(count_vector.fit_transform(X_Train_RBF['Cleaned_text'].values))
X_Test_data_bow_rbf=(count_vector.transform(X_Test_RBF['Cleaned_text'].values))
X_CV_data_bow_rbf=(count_vector.transform(X_CV_RBF['Cleaned_text'].values))

print("After vectorizations")
print(X_Train_data_bow_rbf.shape, Y_Train_RBF.shape)
print(X_CV_data_bow_rbf.shape, Y_CV_RBF.shape)
print(X_Test_data_bow_rbf.shape, Y_Test_RBF.shape)
print("="*100)
```

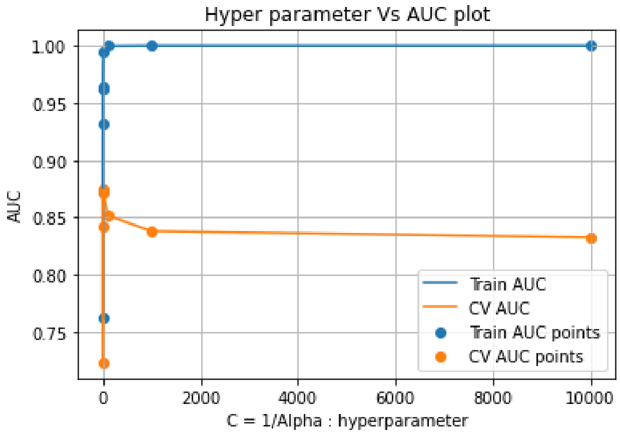
```
(19200, 12) (19200,)
(4800, 12) (4800,)
(6000, 12) (6000,)
====================================================================================================
After vectorizations
(19200, 500) (19200,)
(4800, 500) (4800,)
(6000, 500) (6000,)
====================================================================================================
```

```
from sklearn.model_selection import GridSearchCV
from scipy.stats import randint as sp_randint
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC

def RBF_SVM(x_training_data,y_training_data):
  grid_params = { 'C' : [10**x for x in range(-4,5)]
               }
  rbf_SVM = SVC(kernel='rbf',random_state=None, class_weight=None)
  clf=GridSearchCV(rbf_SVM,grid_params,scoring='roc_auc',return_train_score=True,cv=3)
  clf.fit(x_training_data,y_training_data)
  results = pd.DataFrame.from_dict(clf.cv_results_)
  results = results.sort_values(['param_C'])
  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_c =  results['param_C']
  print(clf.best_score_)
  print(clf.best_params_)
  plt.plot(best_c, train_auc, label='Train AUC')
  plt.plot(best_c, cv_auc, label='CV AUC')
  plt.scatter(best_c, train_auc, label='Train AUC points')
  plt.scatter(best_c, cv_auc, label='CV AUC points')
  plt.legend()
  plt.xlabel("C = 1/Alpha : hyperparameter")
  plt.ylabel("AUC")
  plt.title("Hyper parameter Vs AUC plot")
  plt.grid()
```

```
plt.show()
return results,clf,rbf_SVM
```

```
results,best_one_by_alpha,rbf_svm = RBF_SVM(X_Train_data_bow_rbf,Y_Train_RBF)
results.head()
```

```
0.8744640078013962
{'C': 1}
```



	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_tes
0	8.881859	0.075935	3.833624	0.016110	0.0001	{'C': 0.0001}	0.726542	0
1	11.182248	0.074443	4.521126	0.398858	0.001	{'C': 0.001}	0.841201	0
2	16.277469	0.170417	6.572980	0.049068	0.01	{'C': 0.01}	0.872069	0
3	16.762876	0.292377	7.022699	0.063776	0.1	{'C': 0.1}	0.872636	0
4	16.579718	0.298460	7.122815	0.083631	1	{'C': 1}	0.873039	0

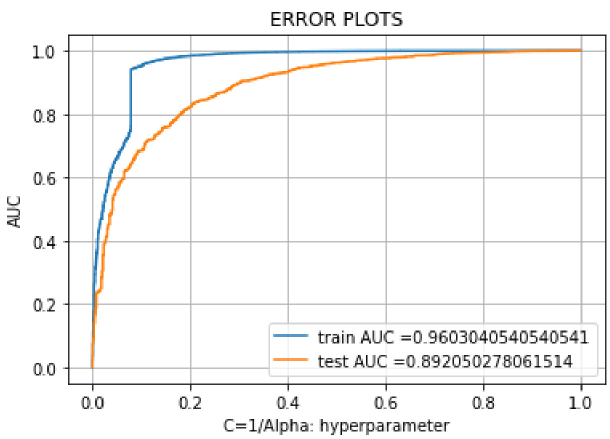
```
def Find_best_C(best_c) :
  best_C = best_c.best_params_
  best_C=best_C.get("C")
  print(best_C)
  return best_C
```

```
best_c = Find_best_C(best_one_by_alpha)
```

```
1
```

```
rbf_svm = SVC(C=best_c, kernel='rbf',random_state=None, class_weight=None,probability=True)
rbf_svm.fit(X_Train_data_bow_rbf,Y_Train_RBF)
  #calibrated_model=CalibratedClassifierCV(clf,cv='prefit')
  #model_m=calibrated_model.fit(tfidf_sent_vectors_train,Y_Train)
pred_test_data=rbf_svm.predict(X_Test_data_bow_rbf)
y_train_predicted_prob = rbf_svm.predict_proba(X_Train_data_bow_rbf)[:,1]
y_test_predicted_prob=rbf_svm.predict_proba(X_Test_data_bow_rbf)[:,1]
train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train_RBF,y_train_predicted_prob,pos_label='positive')
test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test_RBF, 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("C=1/Alpha: 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_RBF,y_test_predicted_prob)
```

```
0.892050278061514
```

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

```
              precision    recall  f1-score   support

    negative       0.87      0.14      0.24       660
    positive       0.90      1.00      0.95      5340

    accuracy                           0.90      6000
   macro avg       0.89      0.57      0.60      6000
weighted avg       0.90      0.90      0.87      6000

[[  93  567]
 [  14 5326]]
```

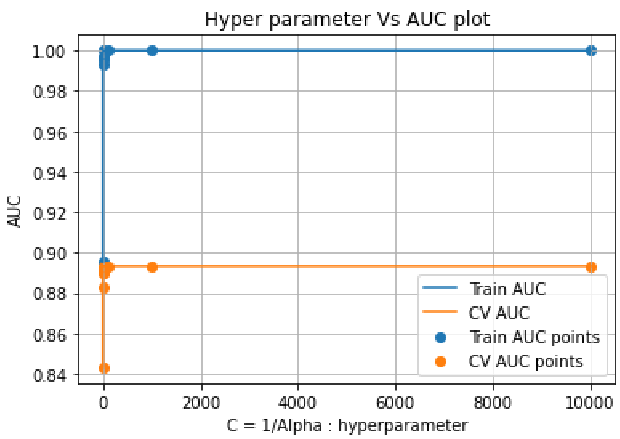
SUPPORT VECTOR MACHINE TFIDF VECTORIZATION TECHNIQUE FOR RBF KI

```
from sklearn.feature_extraction.text import TfidfVectorizer

print(X_Train_RBF.shape, Y_Train_RBF.shape)
print(X_CV_RBF.shape, Y_CV_RBF.shape)
print(X_Test_RBF.shape, Y_Test_RBF.shape)

print("="*100)


tfidf_vector=TfidfVectorizer(min_df=10,max_features=500)
X_Train_data_tfidf=(tfidf_vector.fit_transform(X_Train_RBF['Cleaned_text'].values))
X_Test_data_tfidf=(tfidf_vector.transform(X_Test_RBF['Cleaned_text'].values))
X_CV_data_tfidf=(tfidf_vector.transform(X_CV_RBF['Cleaned_text'].values))

print("After vectorizations")
print(X_Train_data_tfidf.shape, Y_Train_RBF.shape)
print(X_CV_data_tfidf.shape, Y_CV_RBF.shape)
print(X_Test_data_tfidf.shape, Y_Test_RBF.shape)
print("="*100)
```

```
(19200, 12) (19200,)
(4800, 12) (4800,)
(6000, 12) (6000,)
====================================================================================================
After vectorizations
(19200, 500) (19200,)
(4800, 500) (4800,)
(6000, 500) (6000,)
====================================================================================================
```

```
results,best_one_by_alpha,rbf_svm = RBF_SVM(X_Train_data_tfidf,Y_Train_RBF)
results.head()
```

```
0.8933775480843108
{'C': 10}
```



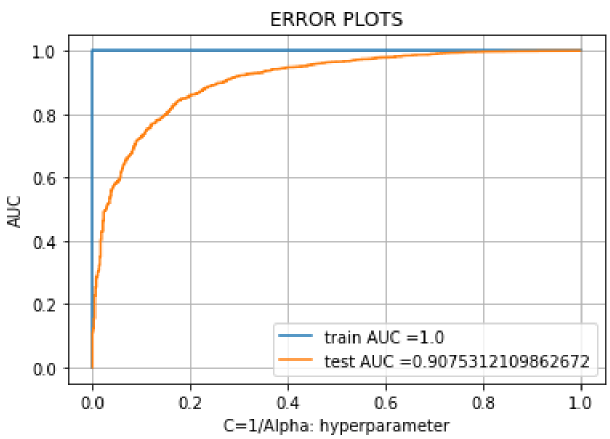
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_tes
0	8.389556	0.094299	3.762124	0.025662	0.0001	{'C': 0.0001}	0.848933	0
1	11.170771	0.033705	4.209851	0.006669	0.001	{'C': 0.001}	0.882891	0
2	15.728408	0.163814	5.994792	0.052254	0.01	{'C': 0.01}	0.888252	0
3	15.862320	0.210091	6.360304	0.062101	0.1	{'C': 0.1}	0.888043	0
4	15.695362	0.228676	6.734292	0.308750	1	{'C': 1}	0.889058	0

```
best_c = Find_best_C(best_one_by_alpha)
```

```
10
```

```
rbf_svm = SVC(C=best_c, kernel='rbf',random_state=None, class_weight=None,probability=True)
rbf_svm.fit(X_Train_data_tfidf,Y_Train_RBF)
  #calibrated_model=CalibratedClassifierCV(clf,cv='prefit')
  #model_m=calibrated_model.fit(tfidf_sent_vectors_train,Y_Train)
pred_test_data=rbf_svm.predict(X_Test_data_tfidf)
y_train_predicted_prob = rbf_svm.predict_proba(X_Train_data_tfidf)[:,1]
y_test_predicted_prob=rbf_svm.predict_proba(X_Test_data_tfidf)[:,1]
train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train_RBF,y_train_predicted_prob,pos_label='positive')
test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test_RBF, 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("C=1/Alpha: hyperparameter")
plt.ylabel("AUC")
```

```
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
roc_auc_score(Y_Test_RBF,y_test_predicted_prob)
```

```
0.9075312109862672
```

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

```
              precision    recall  f1-score   support

    negative       0.73      0.37      0.49       660
    positive       0.93      0.98      0.95      5340

    accuracy                           0.92      6000
   macro avg       0.83      0.67      0.72      6000
weighted avg       0.90      0.92      0.90      6000

[[ 241  419]
 [  88 5252]]
```

AVERAGE WORD2VEC VECTORIZATION TECHNIQUE USING RFB KERNEL

```
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_RBF['Cleaned_text'].values):
    list_of_sent_train_avgw2v.append(sent_train_avgw2v.split())
```

```
100%|██████████| 19200/19200 [00:00<00:00, 170831.10it/s]
```

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

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

```
100%|██████████| 6000/6000 [00:00<00:00, 198876.43it/s]
100%|██████████| 4800/4800 [00:00<00:00, 165923.50it/s]
```

```
w2v_model_train = Word2Vec(list_of_sent_train_avgw2v,min_count=5,size=50,workers=4)
w2v_words_svmrbf_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_svmrbf_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_svmrbf_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_svmrbf_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]))
```

```
19200
50
```

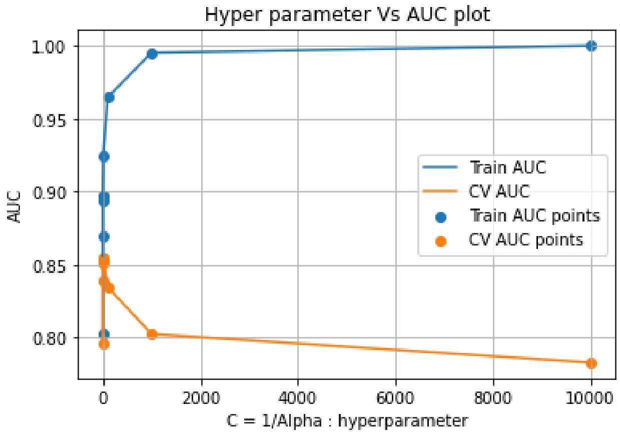
```
test_vectors=[];
for sent in list_of_sent_test_avgw2v:
    sent_vec=np.zeros(50)
    cnt_words=0;
    for word in sent:
        if word in w2v_words_svmrbf_test:
            vec=w2v_model_test.wv[word]
            sent_vec+=vec
            cnt_words+=1
    if cnt_words !=0:
        sent_vec/=cnt_words
    test_vectors.append(sent_vec)
print(len(test_vectors))
print(len(test_vectors[0]))
```

```
6000
50
```

```
cv_vectors=[];
for sent in list_of_sent_cv_avgw2v:
    sent_vec=np.zeros(50)
    cnt_words=0;
    for word in sent:
        if word in w2v_words_svmrbf_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]))
```

```
results,best_one_by_alpha,rbf_svm = RBF_SVM(train_vectors,Y_Train_RBF)
results.head()
```

```
0.854387844023114
{'C': 10}
```

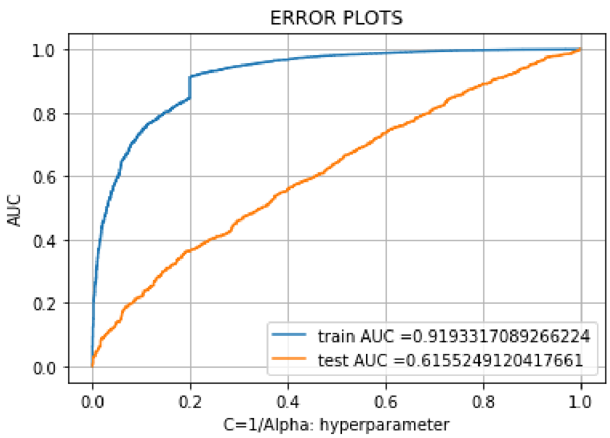


	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test
0	4.219129	0.043246	1.662449	0.004553	0.0001	{'C': 0.0001}	0.798200	0
1	4.590138	0.026659	1.666605	0.004292	0.001	{'C': 0.001}	0.844461	0
2	5.761482	0.085106	1.824312	0.001734	0.01	{'C': 0.01}	0.851311	0
3	5.973886	0.087304	2.019669	0.227633	0.1	{'C': 0.1}	0.853838	0
4	5.691197	0.070469	1.847777	0.007446	1	{'C': 1}	0.854072	0

```
best_c = Find_best_C(best_one_by_alpha)
```

```
10
```

```
rbf_svm = SVC(C=best_c, kernel='rbf',random_state=None, class_weight=None,probability=True)
rbf_svm.fit(train_vectors,Y_Train_RBF)
  #calibrated_model=CalibratedClassifierCV(clf,cv='prefit')
  #model_m=calibrated_model.fit(tfidf_sent_vectors_train,Y_Train)
pred_test_data=rbf_svm.predict(test_vectors)
y_train_predicted_prob = rbf_svm.predict_proba(train_vectors)[:,1]
y_test_predicted_prob=rbf_svm.predict_proba(test_vectors)[:,1]
train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train_RBF,y_train_predicted_prob,pos_label='positive')
test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test_RBF, 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("C=1/Alpha: 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_RBF,y_test_predicted_prob)
```

```
0.6155249120417661
```

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

```
              precision    recall  f1-score   support

    negative       0.14      0.00      0.00       660
    positive       0.89      1.00      0.94      5340

    accuracy                           0.89      6000
   macro avg       0.52      0.50      0.47      6000
weighted avg       0.81      0.89      0.84      6000

[[   1  659]
 [   6 5334]]
```

TF-IDF Average Word2Vec using RBF Kernel

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

```
X_Test_Avgw2v=model_Avgw2v.transform(X_Test_RBF['Cleaned_text'].values)
X_CV_Avgw2v=model_Avgw2v.transform(X_CV_RBF['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_svmrbf_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
```

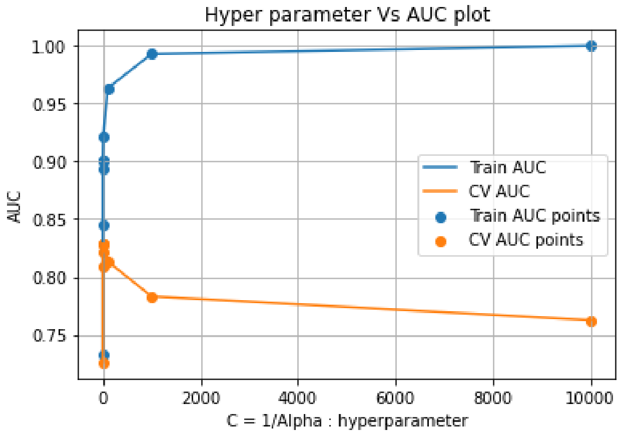
```
100%|██████████| 19200/19200 [02:15<00:00, 141.21it/s]
```

```
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_svmrbf_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%|██████████| 6000/6000 [00:40<00:00, 148.81it/s]
```

```
results,best_one_by_alpha,rbf_svm = RBF_SVM(tfidf_sent_vectors_train,Y_Train_RBF)
results.head()
```

```
0.8291443099232637
{'C': 10}
```

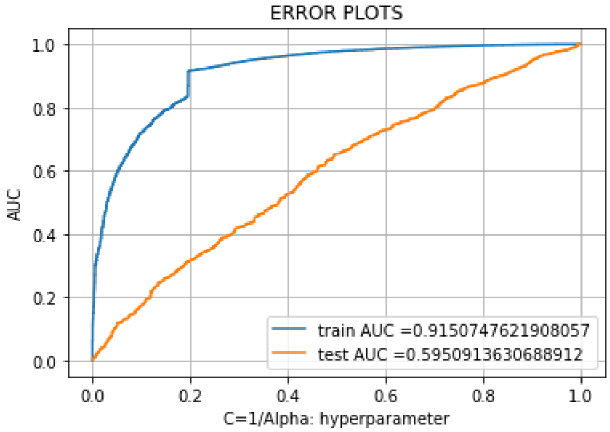


	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_test_score	split1_test
0	4.131723	0.030522	1.669118	0.010235	0.0001	{'C': 0.0001}	0.724419	0
1	4.467314	0.044934	1.671763	0.007991	0.001	{'C': 0.001}	0.784976	0
2	5.640827	0.061905	1.883818	0.011674	0.01	{'C': 0.01}	0.820578	0
3	6.628933	0.044802	1.990721	0.026210	0.1	{'C': 0.1}	0.830384	0
4	6.555834	0.098492	1.970353	0.011784	1	{'C': 1}	0.831317	0

```
best_c = Find_best_C(best_one_by_alpha)
```

```
10
```

```
rbf_svm = SVC(C=best_c, kernel='rbf',random_state=None, class_weight=None,probability=True)
rbf_svm.fit(tfidf_sent_vectors_train,Y_Train_RBF)
  #calibrated_model=CalibratedClassifierCV(clf,cv='prefit')
  #model_m=calibrated_model.fit(tfidf_sent_vectors_train,Y_Train)
pred_test_data=rbf_svm.predict(tfidf_sent_vectors_test)
y_train_predicted_prob = rbf_svm.predict_proba(tfidf_sent_vectors_train)[:,1]
y_test_predicted_prob=rbf_svm.predict_proba(tfidf_sent_vectors_test)[:,1]
train_fpr, train_tpr, train_thresholds=roc_curve(Y_Train_RBF,y_train_predicted_prob,pos_label='positive')
test_fpr, test_tpr, test_thresholds = roc_curve(Y_Test_RBF, 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("C=1/Alpha: hyperparameter")
plt.ylabel("AUC")
plt.title("ERROR PLOTS")
plt.grid()
plt.show()
```




```
roc_auc_score(Y_Test_RBF,y_test_predicted_prob)
```

```
0.5950913630688912
```

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

```
              precision    recall  f1-score   support

    negative       0.16      0.01      0.01       660
    positive       0.89      0.99      0.94      5340

    accuracy                           0.89      6000
   macro avg       0.52      0.50      0.48      6000
weighted avg       0.81      0.89      0.84      6000

[[   5  655]
 [  27 5313]]
```

PRETTY TABLE

```
pip install -U PTable
```

```
Collecting PTable
  Downloading https://files.pythonhosted.org/packages/ab/b3/b54301811173ca94119eb474634f120a49cd370f257d1aae5a4a
Building wheels for collected packages: PTable
  Building wheel for PTable (setup.py) ... done
  Created wheel for PTable: filename=PTable-0.9.2-cp36-none-any.whl size=22908 sha256=34e773e9cf72ff9c2b6259e9be
  Stored in directory: /root/.cache/pip/wheels/22/cc/2e/55980bfe86393df3e9896146a01f6802978d09d7ebcba5ea56
Successfully built PTable
Installing collected packages: PTable
Successfully installed PTable-0.9.2
```

```
from prettytable import PrettyTable

x= PrettyTable()
x.field_names = ["Vectorizer" , "Hyperparameter(Alpha/C)", "AUC","Regularization","Kernel"]
x.add_row(["Bag Of Words",0.001,0.9266,'L2',"Linear SVM"])
x.add_row(["Bag Of Words",0.0001,0.8836,'L1',"Linear SVM"])
x.add_row(["Tf-Idf",0.0001,0.9353,"L2","Linear SVM"])
x.add_row(["Tf-Idf",0.0001,0.9180,"L1","Linear SVM"])
x.add_row(["Avg Word2Vec",0.001,0.8893,"L2","Linear SVM"])
x.add_row(["Avg Word2Vec",0.001,0.8868,"L1","Linear SVM"])
x.add_row(["TF-IDF Word2Vec",0.001,0.8593,"L2","Linear SVM"])
x.add_row(["TF-IDF Word2Vec",0.0001,0.8519,"L1","Linear SVM"])
```

```
x.add_row(["Bag Of Words ",1,0.8744,"N/A","RBF SVM"])
x.add_row(["TF-IDF ",10,0.8933,"N/A","RBF SVM"])
x.add_row(["Avg Word2Vec ",10,0.8543,"N/A","RBF SVM"])
x.add_row(["TF-IDF Word2Vec ",10,0.82914,"N/A","RBF SVM"])
print(x)
```

```
+-----------------+-----------------------+---------+----------------+------------+
|    Vectorizer   | Hyperparameter(Alpha/C) |   AUC   | Regularization |   Kernel   |
+-----------------+-----------------------+---------+----------------+------------+
|   Bag Of Words  |         0.001          |  0.9266 |       L2       | Linear SVM |
|   Bag Of Words  |         0.0001         |  0.8836 |       L1       | Linear SVM |
|      Tf-Idf     |         0.0001         |  0.9353 |       L2       | Linear SVM |
|      Tf-Idf     |         0.0001         |  0.918  |       L1       | Linear SVM |
|   Avg Word2Vec  |         0.001          |  0.8893 |       L2       | Linear SVM |
|   Avg Word2Vec  |         0.001          |  0.8868 |       L1       | Linear SVM |
| TF-IDF Word2Vec |         0.001          |  0.8593 |       L2       | Linear SVM |
| TF-IDF Word2Vec |         0.0001         |  0.8519 |       L1       | Linear SVM |
|  Bag Of Words   |           1            |  0.8744 |      N/A       |  RBF SVM   |
|     TF-IDF      |           10           |  0.8933 |      N/A       |  RBF SVM   |
|  Avg Word2Vec   |           10           |  0.8543 |      N/A       |  RBF SVM   |
| TF-IDF Word2Vec |           10           | 0.82914 |      N/A       |  RBF SVM   |
+-----------------+-----------------------+---------+----------------+------------+
```

CONCLUSION : IT HAS BEEN OBSERVED THAT

1) FOR RBF KERNEL , AS C INCREASES , THE AUC SCORE DECREASES ON TEST DATA

2) FOR LINEAR SVM , THERE IS SLIGHTLY DECREASE IN AUC SCORE

3)* FOR LINEAR SVM :THE BEST ALPHA = 0.0001 AND AUC SCORE = 0.9353 USING TF-IDF VECTORIZ TECHNIQUE USING L2 REGULARIZATION*

4) FOR RBF SVM , BEST C = 10 AND AUC SCORE = 0.8933 USING TF-IDF VECTORIZATION TECHNIQUI FOLD CROSS VALIDATION

```
1
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.