OBJECTIVE

1. APPLYING SVM WITH AVG WORD2VEC VECTORIZATION

1. FINDING THE BEST HYPERPARAMETER USING GRIDSEARCHCV WITH TRAIN DATA AND CROSS-VALIDATION DATA BY PLOTTING THE RESLUTS OF CROSS VALIDATION DATA UISNG HEATMAP
2. PLOTTING OF ROC CURVE TO CHECK FOR THE AUC_SCORE
3. USING THE APROPRIATE VALUE OF HYPERPARAMETER ,TESTING ACCURACY ON TEST DATA USING AUC_SCORE
4. PLOTTING THE CONFUSION MATRIX TO GET THE PRECISOIN ,RECALL VALUE WITH HELP OF HEATMAP
5. PRINTING THE TOP 30 MOST IMPORTANT FEATURES #

In [3]:
```
from sklearn.model_selection import train_test_split          #importin
g the necessary libraries
from sklearn.model_selection import RandomizedSearchCV
from sklearn.datasets import *
from sklearn import naive_bayes
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import pandas as pd
from sklearn import *
import warnings
warnings.filterwarnings("ignore")
from gensim.models import Word2Vec
from tqdm import tqdm
```

In [4]:
```
final_processed_data=pd.read_csv("C:/Users/Mayank/Desktop/final_new_dat
a.csv")#loading the preprocessed data  with 100k points into dataframe
```

In [5]:
```
# getting the counts of 0 and 1 in "SCORE" column to know whether it is
```

```
 unbalanced data or not
count_of_1=0
count_of_0=0
for i in final_processed_data['Score']:
   if i==1:
    count_of_1+=1
   else:
    count_of_0+=1
print(count_of_1)
print(count_of_0)
#it is an imbalanced dataset
```

```
88521
11479
```

In [6]:
```
#spliiting the data into train and test data
x_train,x_test,y_train,y_test=model_selection.train_test_split(final_processed_data['CleanedText'].values,final_processed_data['Score'].values,test_size=0.2,shuffle=False)
```

In [7]:
```
# Training my own Word2Vec model using your own text corpus
list_of_sent=[]
for sent in x_train:
 list_of_sent.append(sent.split())#splitting of sentences into words AND appending them to list
print(x_train[0])
print("*****************************************************************")
print(list_of_sent[0])
word_to_vector=Word2Vec(list_of_sent,min_count=5,size=50,workers=2)#constructing my our word to vector
w_t_c_words=list(word_to_vector.wv.vocab)
print("*********************************************************************")
print("sample words ", w_t_c_words[0:50])
```

```
witti littl book make son laugh loud recit car drive along alway sing refrain hes learn whale india droop love new word book introduc silli classic book will bet son still abl recit memori colleg
```

```
*****************************************************************
['witti', 'littl', 'book', 'make', 'son', 'laugh', 'loud', 'recit', 'ca
r', 'drive', 'along', 'alway', 'sing', 'refrain', 'hes', 'learn', 'whal
e', 'india', 'droop', 'love', 'new', 'word', 'book', 'introduc', 'sill
i', 'classic', 'book', 'will', 'bet', 'son', 'still', 'abl', 'recit',
'memori', 'colleg']
***********************************************************************
sample words  ['witti', 'littl', 'book', 'make', 'son', 'laugh', 'lou
d', 'car', 'drive', 'along', 'alway', 'sing', 'refrain', 'hes', 'lear
n', 'india', 'droop', 'love', 'new', 'word', 'introduc', 'silli', 'clas
sic', 'will', 'bet', 'still', 'abl', 'memori', 'colleg', 'rememb', 'se
e', 'show', 'air', 'televis', 'year', 'ago', 'child', 'sister', 'late
r', 'bought', 'day', 'thirti', 'someth', 'use', 'seri', 'song', 'studen
t', 'teach', 'preschool', 'turn']
```

In [8]:

```python
###### NOW STARTING AVERAGE WORD TO VEC FOR TRAIN DATA#################
##################################################
train_sent_vectors = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sent in tqdm(list_of_sent): # for each review/sentence
 sent_vec = np.zeros(50) # as word vectors are of zero length
 cnt_words =0; # num of words with a valid vector in the sentence/revie
w
 for word in sent: # for each word in a review/sentence
   if word in w_t_c_words:
     vec = word_to_vector.wv[word]
     sent_vec += vec
     cnt_words += 1
 if cnt_words != 0:
  sent_vec /= cnt_words
 train_sent_vectors.append(sent_vec)
print(len(train_sent_vectors))
print(len(train_sent_vectors[0]))
```

```
100%|██████████████████████████████████████████████████████████████
█████████| 80000/80000 [05:05<00:00, 261.94it/s]
```

```
80000
50
```

In [9]:
```
from sklearn.preprocessing import StandardScaler #standarizing the trai
ning  data
x_train_data=StandardScaler( with_mean=False).fit_transform(train_sent_
vectors)
print(x_train_data.shape)
```

```
(80000, 50)
```

In [10]:
```
list_of_sent=[]
for sent in x_test:
 list_of_sent.append(sent.split())#splitting of sentences into words AN
D appending them to list
print(x_test[0])
print("*****************************************************************
*")
print(list_of_sent[0])
print('*****************************************************************
***')
```

```
hard find item dont buy mani either came stale got way quick classic no
netheless
*****************************************************************
['hard', 'find', 'item', 'dont', 'buy', 'mani', 'either', 'came', 'stal
e', 'got', 'way', 'quick', 'classic', 'nonetheless']
*******************************************************************
```

In [11]:
```
###### NOW STARTING AVERAGE WORD TO VEC FOR TEST DATA##################
################################################
sent_vectors = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in tqdm(list_of_sent): # for each review/sentence
 sent_vec = np.zeros(50) # as word vectors are of zero length
 cnt_words =0; # num of words with a valid vector in the sentence/revie
w
 for word in sent: # for each word in a review/sentence
   if word in w_t_c_words:
     vec = word_to_vector.wv[word]
     sent_vec += vec
     cnt_words += 1
```

```
    if cnt_words != 0:
        sent_vec /= cnt_words
    sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████████████████████████████████████
██████████| 20000/20000 [01:07<00:00, 297.36it/s]
```

```
20000
50
```

In [12]:

```
from sklearn.preprocessing import StandardScaler #standarizing the trai
ning  data
x_test_data=StandardScaler( with_mean=False).fit_transform(sent_vectors
)
print(x_test_data.shape)
```

```
(20000, 50)
```

In [14]:

```
#using time series split method for cross-validation score
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=10)
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
data=[10**-4,10**-3,10**-2,10**-1,10**0,10**1,10**2,10**3,10**4]#range
 of hyperparameter


sgd=SGDClassifier(loss='log',class_weight={1:0.5,0:0.5},n_jobs=-1)
tuned_para=[{'alpha':data,'penalty':['l1','l2']}]
```

In [14]:

```
#applying the model of support vector machine and using gridsearchcv to
 find the best hyper parameter
%time
from sklearn.model_selection import GridSearchCV
model = GridSearchCV(sgd, tuned_para, scoring = 'roc_auc', cv=tscv,n_jo
bs=-1)#building the gridsearchcv model
model.fit(x_train_data, y_train)#fiitting the training data
```

```
print('BEST ESTIMATORS FOR MODEL ARE ',model.best_estimator_)#printing
 the best_estimator
print('AUC_SCORE OF TEST DATA IS',model.score(x_test_data, y_test))
```

```
Wall time: 0 ns
BEST ESTIMATORS FOR MODEL ARE  SGDClassifier(alpha=0.01, average=False,
class_weight={1: 0.5, 0: 0.5},
       epsilon=0.1, eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='hinge', max_iter=None, n_iter=Non
e,
       n_jobs=-1, penalty='l2', power_t=0.5, random_state=None,
       shuffle=True, tol=None, verbose=0, warm_start=False)
AUC_SCORE OF TEST DATA IS 0.883125713187
```

In [20]:

```
results=pd.DataFrame(model.cv_results_)# getting varoius cv_scores and
 train_scores various values of alpha given as parameter and storing it
 in a dataframe
results#printing the dataframe
```

Out[20]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha
0	0.484598	0.020583	0.935504	0.940772	0.0001
1	0.317703	0.043863	0.941739	0.944645	0.0001
2	0.539336	0.042289	0.945506	0.947192	0.001

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha
3	0.331282	0.021286	0.945987	0.949872	0.001
4	0.468636	0.036624	0.941242	0.941594	0.01
5	0.364310	0.030142	0.943711	0.945327	0.01
6	0.481707	0.032230	0.941242	0.941594	0.1
7	0.392222	0.036737	0.941242	0.941594	0.1
8	0.455280	0.032383	0.941242	0.941594	1
9	0.360499	0.028852	0.941242	0.941594	1

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha
10	0.633698	0.031558	0.941242	0.941594	10
11	0.389046	0.032003	0.941242	0.941594	10
12	0.594911	0.026392	0.847434	0.847301	100
13	0.298296	0.036590	0.868463	0.862732	100
14	0.570670	0.038366	0.941242	0.941594	1000
15	0.315481	0.030964	0.941242	0.941594	1000
16	0.639877	0.020887	0.752562	0.753096	10000

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha
17	0.340530	0.027133	0.941242	0.941594	10000

18 rows × 32 columns

In [21]:
```
results['mean_test_score']=results['mean_test_score']*100
results['mean_test_score']
```

Out[21]:
```
0     93.550441
1     94.173856
2     94.550553
3     94.598665
4     94.124239
5     94.371058
6     94.124239
7     94.124239
8     94.124239
9     94.124239
10    94.124239
11    94.124239
12    84.743398
13    86.846289
14    94.124239
15    94.124239
16    75.256239
17    94.124239
Name: mean_test_score, dtype: float64
```

In [22]:
```
results['mean_test_score']=100-results['mean_test_score']
results['mean_cv_error']=results['mean_test_score'].round(decimals=2)
results.head()
```

Out[22]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha	p

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha	p
0	0.484598	0.020583	6.449559	0.940772	0.0001	l'
1	0.317703	0.043863	5.826144	0.944645	0.0001	l:
2	0.539336	0.042289	5.449447	0.947192	0.001	l'
3	0.331282	0.021286	5.401335	0.949872	0.001	l:
4	0.468636	0.036624	5.875761	0.941594	0.01	l'

5 rows × 33 columns

PLOTTING THE HEATMAP WITH HYPERPARAMETERS FOR CV_ERROR SCORE

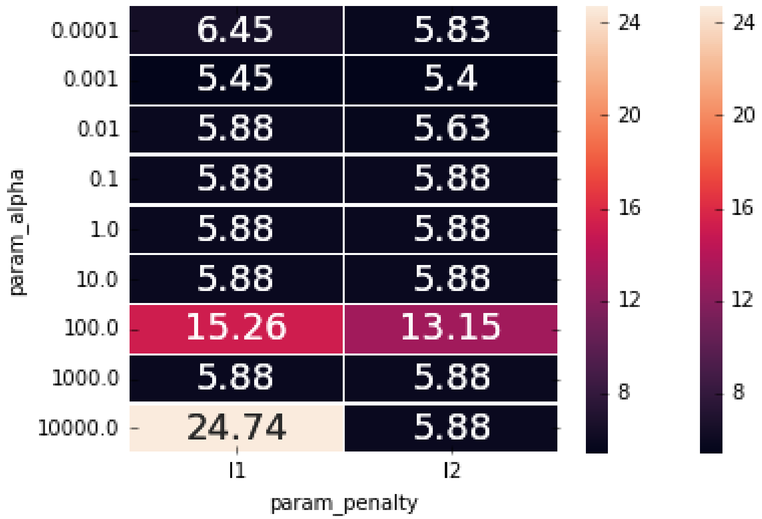
In [23]:
```
test_score_heatmap=results.pivot(        'param_alpha'    ,'param_penalt
y','mean_cv_error'       )
```

In [24]:
```
test_score_heatmap
```

Out[24]:

param_penalty	l1	l2
param_alpha		
0.0001	6.45	5.83
0.0010	5.45	5.40
0.0100	5.88	5.63
0.1000	5.88	5.88
1.0000	5.88	5.88
10.0000	5.88	5.88
100.0000	15.26	13.15
1000.0000	5.88	5.88
10000.0000	24.74	5.88

In [26]:
```
import seaborn as sns
sns.heatmap(test_score_heatmap,annot=True,annot_kws={"size": 18}, fmt=
'g',linewidths=.5)
import matplotlib.pylab as plt
plt.show()
```



FROM HEATMAP THE BEST HYPERPARAMETER VALUES ARE FOUND TO BE PENALTY='L2' AND 'PARAM_ALPHA'=0.001

BUILDING MODEL FOR SGD WITH CALIBRATED CLASSIFIER CV

In [16]:
```
sgd=SGDClassifier(loss='log',class_weight={1:0.5,0:0.5},n_jobs=-1,alpha
=0.001,penalty='l2')
sgd.fit(x_train_data,y_train)
```

Out[16]:
```
SGDClassifier(alpha=0.001, average=False, class_weight={1: 0.5, 0: 0.
5},
       epsilon=0.1, eta0=0.0, fit_intercept=True, l1_ratio=0.15,
       learning_rate='optimal', loss='log', max_iter=None, n_iter=None,
```

```
                 n_jobs=-1, penalty='l2', power_t=0.5, random_state=None,
                 shuffle=True, tol=None, verbose=0, warm_start=False)
```

In [17]:
```python
from sklearn.metrics import brier_score_loss
prob_pos_clf = sgd.predict_proba(x_test_data)[:, 1]

# Gaussian Naive-Bayes with isotonic calibration
from sklearn.calibration import   CalibratedClassifierCV
clf_isotonic = CalibratedClassifierCV(sgd, cv=5, method='isotonic')
clf_isotonic.fit(x_train_data, y_train)
prob_pos_isotonic = clf_isotonic.predict_proba(x_test_data)[:, 1]

# Gaussian Naive-Bayes with sigmoid calibration
clf_sigmoid = CalibratedClassifierCV(sgd, cv=5, method='sigmoid')
clf_sigmoid.fit(x_train_data, y_train)
prob_pos_sigmoid = clf_sigmoid.predict_proba(x_test_data)[:, 1]

print("Brier scores: (the smaller the better)")

clf_score = brier_score_loss(y_test, prob_pos_clf)
print("No calibration: %1.3f" % clf_score)


clf_isotonic_score = brier_score_loss(y_test, prob_pos_isotonic)
print("With isotonic calibration: %1.3f" % clf_isotonic_score)

clf_sigmoid_score = brier_score_loss(y_test, prob_pos_sigmoid)
print("With sigmoid calibration: %1.3f" % clf_sigmoid_score)
```

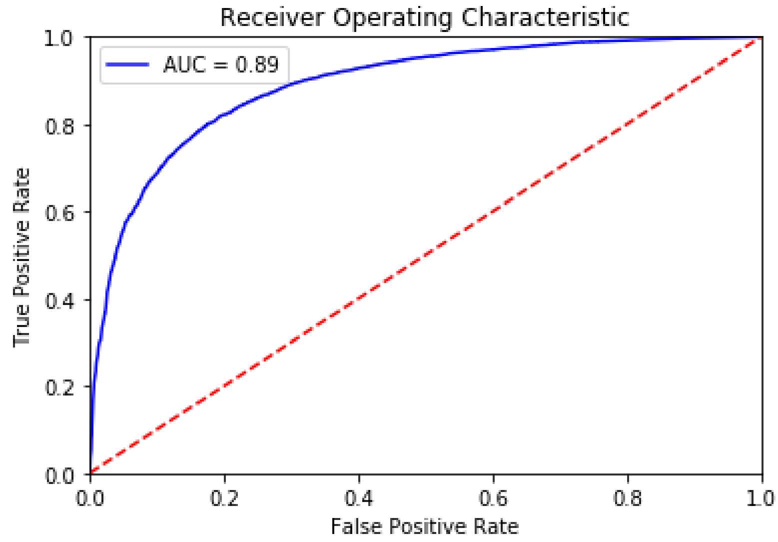
```
Brier scores: (the smaller the better)
No calibration: 0.077
With isotonic calibration: 0.077
With sigmoid calibration: 0.077
```

ISOTONIC CALIBRATION IS HAVING BEST VALUE FOR CALIBRATED CLASSIFIER CV

PLOTTING THE ROC CURVE FOR TRAIN_DATA

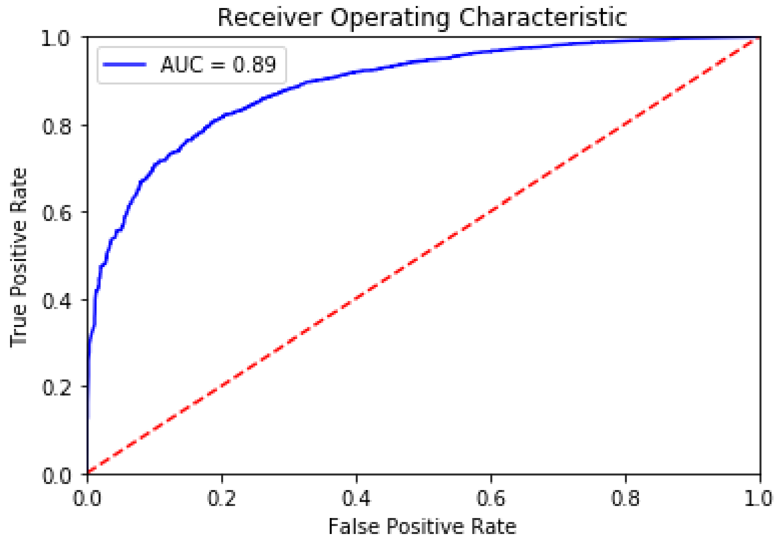
In [18]:
```
clf_isotonic = CalibratedClassifierCV(sgd, cv=5, method='isotonic')
clf_isotonic.fit(x_train_data, y_train)
train_prob_pos_isotonic = clf_isotonic.predict_proba(x_train_data)[:, 1
]
```

In [19]:
```
fpr, tpr, threshold = metrics.roc_curve(y_train, train_prob_pos_isotoni
c)
roc_auc = metrics.auc(fpr, tpr)


import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'best')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



PLOTTING THE ROC CURVE FOR TEST_DATA

In [20]:
```
test_prob_pos_isotonic = clf_isotonic.predict_proba(x_test_data)[:, 1]
fpr, tpr, threshold = metrics.roc_curve(y_test, test_prob_pos_isotonic)
roc_auc = metrics.auc(fpr, tpr)

#
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'best')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [21]:
```
print("FROM ABOVE PLOT,AUC_SCORE IS FOUND AS ",roc_auc*100)
```

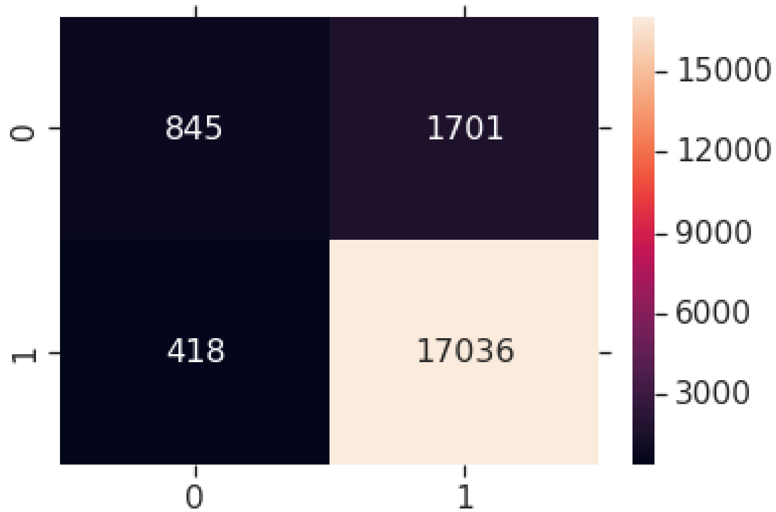
```
FROM ABOVE PLOT,AUC_SCORE IS FOUND AS  88.9106477257
```

USING BEST HYPERPARAMETER VALUE ON TEST DATA AND PLOTTING THE CONFUSION MATRIX WITH HEATMAP

In [34]:
```
#Testing Accuracy on Test data
import seaborn as sns #importing seaborn as sns
from sklearn.metrics import *#importing varoius metrics from sklearn
y_pred=clf_isotonic.predict(x_test_data)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*1
00))#printing accuracy
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
#printing precision score
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred))) #prin
ting recall
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
```

```
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2
)) #generating the heatmap for confusion matrix
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
import matplotlib.pylab as plt
plt.show()
```

```
Accuracy on test set: 89.405%
Precision on test set: 0.909
Recall on test set: 0.976
F1-Score on test set: 0.941
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



RBF KERNEL WITH AVG WORD2VEC VECTORIZATION

OBJECTIVE

1. APPLYING SVM WITH RBF KERNEL WITH AVG WORD2VEC VECTORIZATION
1. FINDING THE BEST HYPERPARAMETER USING GRIDSEARCHCV WITH TRAIN DATA AND CROSS-VALIDATION DATA BY PLOTTING THE RESLUTS OF CROSS VALIDATION DATA UISNG HEATMAP
2. PLOTTING OF ROC CURVE TO CHECK FOR THE AUC_SCORE
3. USING THE APROPRIATE VALUE OF HYPERPARAMETER ,TESTING ACCURACY ON TEST DATA USING F1-SCORE
4. PLOTTING THE CONFUSION MATRIX TO GET THE PRECISOIN ,RECALL VALUE WITH HELP OF HEATMAP

RBF KERNEL IS COMPUTATIONALLY EXPENSIVE SO USING FIRST 30K POINTS ONLY

In [3]:
```
final_data=pd.read_csv('final_data.csv',encoding='latin-1')# IMPORT THE DATA FILE
final_data.head()
```

Out[3]:

	Unnamed: 0	Score	CleanedText
0	0	1	realli like emerald nut buy smoke almond cashe...
1	1	1	crispi chewi intens flavor wow great ive love ...
2	2	1	great product fresh tast school teacher use po...
3	3	1	purchas along espresso ive mix two equal amoun...
4	4	1	yummi stuff surpris quick cook like mccann buy...

In [4]:
```
final_data.shape#PRINTING THE SHAPE OF FILE
```

Out[4]: (30000, 3)

In [5]:
```
#spliiting the data into train and test data
```

```
x_train,x_test,y_train,y_test=model_selection.train_test_split(final_da
ta['CleanedText'].values,final_data['Score'].values,test_size=0.30,shuf
fle=False)
```

In [6]:
```
# Training my own Word2Vec model using your own text corpus
list_of_sent=[]
for sent in x_train:
 list_of_sent.append(sent.split())#splitting of sentences into words AN
D appending them to list
print(x_train[0])
print("*****************************************************************
*")
print(list_of_sent[0])
word_to_vector=Word2Vec(list_of_sent,min_count=5,size=50,workers=2)#con
structing my our word to vector
w_t_c_words=list(word_to_vector.wv.vocab)
print("*****************************************************************
*******")
```

```
realli like emerald nut buy smoke almond cashew cocoa roast almond pres
erv much like emerald nut fresh much oil high qualiti snack instead sal
ti one sweet doesnt come sugar though sweeten light sucralos sweeten so
ld brand name splenda note product doesnt contain chocol though describ
dark chocol flavor cocoa roast surfac nut almond coat chocol good part
dont make mess theyr better choic someon tri avoid sweet sure bought ex
pect get chocol disappoint like enough ill get especi need someth cut c
rave chocol candi enough chocol flavor one gram sugar per serv
*****************************************************************
['realli', 'like', 'emerald', 'nut', 'buy', 'smoke', 'almond', 'cashe
w', 'cocoa', 'roast', 'almond', 'preserv', 'much', 'like', 'emerald',
'nut', 'fresh', 'much', 'oil', 'high', 'qualiti', 'snack', 'instead',
'salti', 'one', 'sweet', 'doesnt', 'come', 'sugar', 'though', 'sweete
n', 'light', 'sucralos', 'sweeten', 'sold', 'brand', 'name', 'splenda',
'note', 'product', 'doesnt', 'contain', 'chocol', 'though', 'describ',
'dark', 'chocol', 'flavor', 'cocoa', 'roast', 'surfac', 'nut', 'almon
d', 'coat', 'chocol', 'good', 'part', 'dont', 'make', 'mess', 'theyr',
'better', 'choic', 'someon', 'tri', 'avoid', 'sweet', 'sure', 'bought',
'expect', 'get', 'chocol', 'disappoint', 'like', 'enough', 'ill', 'ge
t', 'especi', 'need', 'someth', 'cut', 'crave', 'chocol', 'candi', 'eno
```

```
ugh', 'chocol', 'flavor', 'one', 'gram', 'sugar', 'per', 'serv']
***********************************************************************
```

In [7]:
```
###### NOW STARTING AVERAGE WORD TO VEC FOR TRAIN DATA#################
################################################
train_sent_vectors = []; # the avg-w2v for each sentence/review is stor
ed in this list
for sent in tqdm(list_of_sent): # for each review/sentence
 sent_vec = np.zeros(50) # as word vectors are of zero length
 cnt_words =0; # num of words with a valid vector in the sentence/revie
w
 for word in sent: # for each word in a review/sentence
   if word in w_t_c_words:
     vec = word_to_vector.wv[word]
     sent_vec += vec
     cnt_words += 1
 if cnt_words != 0:
  sent_vec /= cnt_words
 train_sent_vectors.append(sent_vec)
print(len(train_sent_vectors))
print(len(train_sent_vectors[0]))
```

```
100%|██████████| 21000/21000 [01:14<00:00, 283.00it/s]
```

```
21000
50
```

In [9]:
```
from sklearn.preprocessing import StandardScaler #standarizing the trai
ning  data
x_train_data=StandardScaler( with_mean=False).fit_transform(train_sent_
vectors)
print(x_train_data.shape)
```

```
(21000, 50)
```

In [10]:
```
list_of_sent=[]
for sent in x_test:
 list_of_sent.append(sent.split())#splitting of sentences into words AN
```

```
D appending them to list
print(x_test[0])
print("****************************************************************
*")
print(list_of_sent[0])
print('****************************************************************
***')
```

```
big famili hit bigger fruit tast found sweet make great snack even dess
ert
*****************************************************************
['big', 'famili', 'hit', 'bigger', 'fruit', 'tast', 'found', 'sweet',
'make', 'great', 'snack', 'even', 'dessert']
*******************************************************************
```

In [11]:

```
###### NOW STARTING AVERAGE WORD TO VEC FOR TEST DATA##################
###############################################
sent_vectors = []; # the avg-w2v for each sentence/review is stored in
 this list
for sent in tqdm(list_of_sent): # for each review/sentence
 sent_vec = np.zeros(50) # as word vectors are of zero length
 cnt_words =0; # num of words with a valid vector in the sentence/revie
w
 for word in sent: # for each word in a review/sentence
   if word in w_t_c_words:
     vec = word_to_vector.wv[word]
     sent_vec += vec
     cnt_words += 1
 if cnt_words != 0:
  sent_vec /= cnt_words
 sent_vectors.append(sent_vec)
print(len(sent_vectors))
print(len(sent_vectors[0]))
```

```
100%|██████████| 9000/9000 [00:31<00:00, 282.94it/s]
```

```
9000
50
```

In [12]:
```
from sklearn.preprocessing import StandardScaler #standarizing the trai
ning  data
x_test_data=StandardScaler( with_mean=False).fit_transform(sent_vectors
)
print(x_test_data.shape)
```

```
(9000, 50)
```

In [22]:
```
#using time series split method for cross-validation score
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=2)
from sklearn.svm import SVC
from sklearn.calibration import CalibratedClassifierCV
c_values=[0.001,0.01,0.1,1,5,10,100]#range of hyperparameter
gamma_values=[0.001,0.01,0.1,1,5,10,100]#range of hyperparameter

svc=SVC(class_weight='balanced',probability=True)
tuned_para=[{'C':c_values,'gamma':gamma_values}]
```

In [14]:
```
#applying the model of support vector machine and using gridsearchcv to
 find the best hyper parameter
%time
from sklearn.model_selection import GridSearchCV
model = GridSearchCV(svc, tuned_para, scoring = 'f1', cv=tscv,n_jobs=-1
)#building the gridsearchcv model
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 6.2 µs
```

In [15]:
```
%%time
model.fit(x_train_data, y_train)#fiitting the training data
```

```
CPU times: user 13min 13s, sys: 918 ms, total: 13min 14s
Wall time: 1h 11min 49s
```

Out[15]:
```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=2),
       error_score='raise-deprecating',
       estimator=SVC(C=1.0, cache_size=200, class_weight='balanced', co
ef0=0.0,
```

```
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=True, random_state=None,
  shrinking=True, tol=0.001, verbose=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid=[{'C': [0.001, 0.01, 0.1, 1, 5, 10, 100], 'gamma':
[0.001, 0.01, 0.1, 1, 5, 10, 100]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)
```

In [16]:
```
model.best_estimator_#checking the best estimator
```

Out[16]:
```
SVC(C=10, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf',
  max_iter=-1, probability=True, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

BUILDING THE HEATMAP FOR CV_ERROR SCORE FOR HYPERPARAMETERS

In [17]:
```
results=pd.DataFrame(model.cv_results_)# getting varoius cv_scores and
 train_scores various values of alpha given as parameter and storing it
 in a dataframe
results#printing the dataframe
```

Out[17]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	para
0	197.648184	10.131180	0.802086	0.798360	0.001	0.00
1	165.438953	9.544565	0.811272	0.807801	0.001	0.01

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	para
2	237.728254	10.377414	0.802086	0.798360	0.001	0.1
3	243.532411	9.945897	0.802086	0.798360	0.001	1
4	193.668029	9.947155	0.802086	0.798360	0.001	5
5	175.502086	9.581793	0.802086	0.798360	0.001	10
6	148.011728	7.960449	0.802086	0.798360	0.001	100
7	157.158270	8.891936	0.772807	0.768199	0.01	0.00
8	130.200086	7.456697	0.829821	0.824365	0.01	0.01

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	para
9	154.453660	9.169340	0.813678	0.808912	0.01	0.1
10	225.879915	9.360167	0.802086	0.798360	0.01	1
11	222.594407	9.790440	0.802086	0.798360	0.01	5
12	245.046898	9.481926	0.802086	0.798360	0.01	10
13	169.181980	7.755779	0.802086	0.798360	0.01	100
14	108.204936	6.550000	0.828290	0.823173	0.1	0.00
15	87.981490	5.343685	0.850993	0.850838	0.1	0.01

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	para
16	148.364382	7.732078	0.861503	0.877600	0.1	0.1
17	188.745009	9.518725	0.821671	0.834432	0.1	1
18	194.440600	9.624862	0.821635	0.834395	0.1	5
19	193.634870	9.317103	0.821635	0.834357	0.1	10
20	164.106144	7.779089	0.821563	0.834320	0.1	100
21	82.629875	4.809428	0.844002	0.843793	1	0.00
22	75.335731	4.206840	0.867249	0.876953	1	0.01
23	167.699641	6.168305	0.897020	0.978558	1	0.1
24	390.728909	7.365018	0.849731	1.000000	1	1

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	para
25	375.612578	7.681297	0.849155	1.000000	1	5
26	411.215178	7.429545	0.849076	1.000000	1	10
27	326.965480	6.117492	0.848999	1.000000	1	100
28	76.941535	4.325538	0.850732	0.853180	5	0.00
29	72.609496	3.858884	0.879103	0.905571	5	0.01
30	207.011395	6.001206	0.898804	0.998951	5	0.1
31	481.868241	7.357225	0.849731	1.000000	5	1
32	537.670162	7.629465	0.849155	1.000000	5	5
33	464.285086	7.340664	0.849076	1.000000	5	10

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	para
34	445.449799	6.167587	0.848999	1.000000	5	100
35	76.290897	4.166864	0.855737	0.859382	10	0.00
36	74.306315	3.776842	0.883809	0.919265	10	0.01
37	216.059892	6.037575	0.899499	1.000000	10	0.1
38	507.007309	7.403592	0.849731	1.000000	10	1
39	513.312575	7.663222	0.849155	1.000000	10	5
40	497.708723	6.724783	0.849076	1.000000	10	10
41	428.170595	5.045459	0.848999	1.000000	10	100
42	83.167659	3.902312	0.864380	0.877149	100	0.00

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	para
43	125.538173	3.485628	0.886143	0.965904	100	0.01
44	218.786061	5.972927	0.898917	1.000000	100	0.1
45	410.852446	6.484249	0.849731	1.000000	100	1
46	421.140705	6.802036	0.849155	1.000000	100	5
47	406.586368	6.433972	0.849076	1.000000	100	10
48	287.551197	5.040411	0.848999	1.000000	100	100

In [19]:

```
results['mean_test_score']=results['mean_test_score']*100#multiplying m
ean_test_score by 100
results['mean_test_score']
results['mean_test_score']=100-results['mean_test_score']#substracting
 from 100 to get a cv_error score
results['mean_cv_error']=results['mean_test_score'].round(decimals=2)#
 rounding cv_error score upto 2 decimal points
results.head()
```

Out[19]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	parar

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	parar
0	197.648184	10.131180	19.791369	0.798360	0.001	0.001
1	165.438953	9.544565	18.872808	0.807801	0.001	0.01
2	237.728254	10.377414	19.791369	0.798360	0.001	0.1
3	243.532411	9.945897	19.791369	0.798360	0.001	1
4	193.668029	9.947155	19.791369	0.798360	0.001	5

In [20]:

```
test_score_heatmap=results.pivot(      'param_C'       ,'param_gamma',
'mean_cv_error' )#converting  into pivot table
```

In [21]:

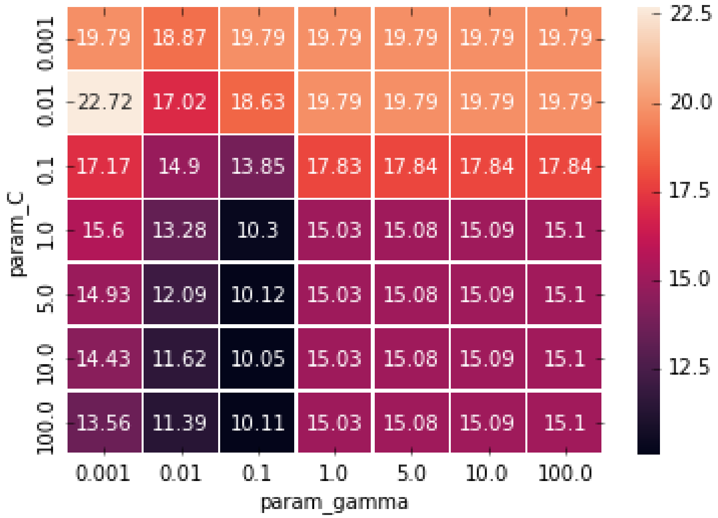
```
test_score_heatmap#printing the pivot table
```

Out[21]:

param_gamma	0.001	0.01	0.1	1.0	5.0	10.0	100.0
param_C							

param_gamma	0.001	0.01	0.1	1.0	5.0	10.0	100.0
param_C							
0.001	19.79	18.87	19.79	19.79	19.79	19.79	19.79
0.010	22.72	17.02	18.63	19.79	19.79	19.79	19.79
0.100	17.17	14.90	13.85	17.83	17.84	17.84	17.84
1.000	15.60	13.28	10.30	15.03	15.08	15.09	15.10
5.000	14.93	12.09	10.12	15.03	15.08	15.09	15.10
10.000	14.43	11.62	10.05	15.03	15.08	15.09	15.10
100.000	13.56	11.39	10.11	15.03	15.08	15.09	15.10

In [27]:
```
import seaborn as sns
sns.heatmap(test_score_heatmap,annot=True,annot_kws={"size": 10}, fmt=
'g',linewidths=.5)
import matplotlib.pylab as plt
plt.show()#printing the heatmap with cv_error
```



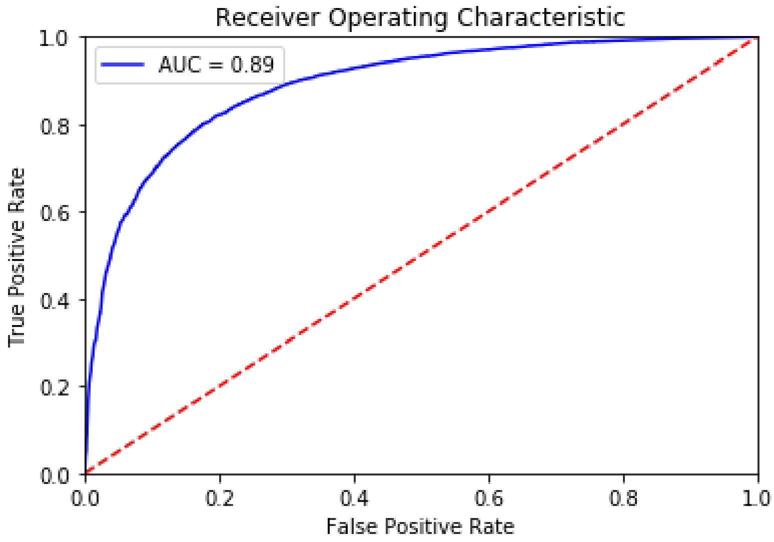
FROM HERE BEST HPYERPARAMETERS ARE GAMMA =0.1 AND C=10

In [23]:
```
# building the model with value of hyperparameters values
svc=SVC(class_weight='balanced',probability=True,C=10,gamma=0.1)
```

PLOTTING THE ROC CURVE FOR TRAIN_DATA

In [ ]:
```
#fitting the model
svc.fit(x_train_data,y_train)
probs = svc.predict_proba(x_train_data)#predicting the model
y_pred_train = probs[:,1]
```

In [22]:
```
fpr, tpr, threshold = metrics.roc_curve(y_train, y_pred_train)
roc_auc = metrics.auc(fpr, tpr)


import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'best')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

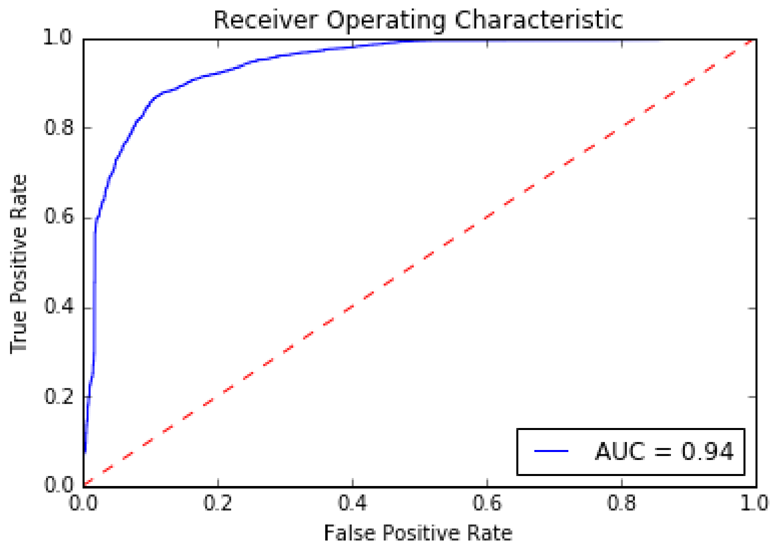


PLOTTING THE ROC CURVE FOR TEST_DATA

In [ ]:
```
#fitting the model
svc.fit(x_train_data,y_train)
probs = svc.predict_proba(x_test_data)#predicting the model
y_pred = probs[:,1]
```

In [37]:
```
#plotting the curve for finding the auc_score
fpr, tpr, threshold = metrics.roc_curve(y_test,y_pred)
roc_auc = metrics.auc(fpr, tpr)

import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'best')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
```

```
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [44]:
```
print("Best auc_score  from above curve is founs to be ",roc_auc*100)
```

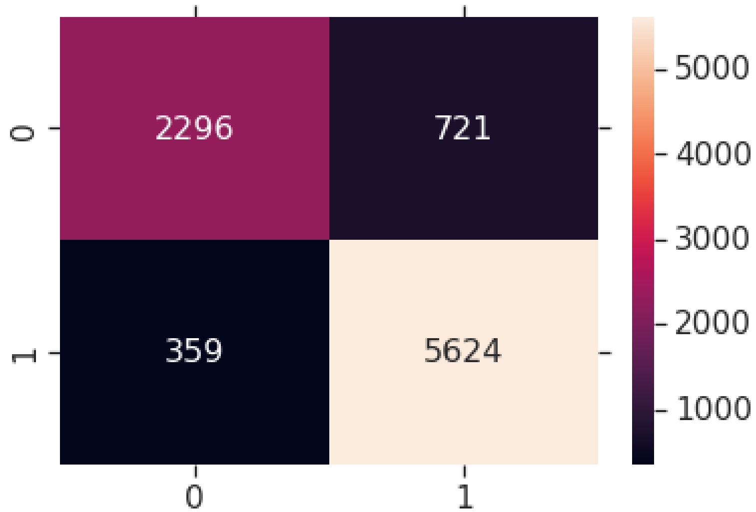
```
Best auc_score  from above curve is founs to be  94.13072703895155
```

USING BEST HYPERPARAMETER VALUE ON TEST DATA AND PLOTTING THE CONFUSION MATRIX WITH HEATMAP

In [42]:
```
#Testing Accuracy on Test data
import seaborn as sns #importing seaborn as sns
from sklearn.metrics import *#importing varoius metrics from sklearn
y_pred=svc.predict(x_test_data)
print("Accuracy on test set: %0.3f%%"%(accuracy_score(y_test, y_pred)*1
00))#printing accuracy
print("Precision on test set: %0.3f"%(precision_score(y_test, y_pred)))
```

```
#printing precision score
print("Recall on test set: %0.3f"%(recall_score(y_test, y_pred))) #prin
ting recall
print("F1-Score on test set: %0.3f"%(f1_score(y_test, y_pred)))
print("Confusion Matrix of test set:\n [ [TN  FP]\n [FN TP] ]\n")
df_cm = pd.DataFrame(confusion_matrix(y_test, y_pred), range(2),range(2
)) #generating the heatmap for confusion matrix
sns.set(font_scale=1.4)#for label size
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
import matplotlib.pylab as plt
plt.show()
```

```
Accuracy on test set: 88.000%
Precision on test set: 0.886
Recall on test set: 0.940
F1-Score on test set: 0.912
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



AVG WORD2VEC VECTORIZATION WITH

SUPPORT VECTOR MACHINE WITH LINEAR KERNEL AND RBF KERNEL IS DONE