OBJECTIVE

1. APPLYING SVM WITH TFIDF 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 [22]:

```
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")
```

In [23]:

```
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 [24]:

```
# 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 [25]:
```
#spliiting the data into train and test data
x_train,x_test,y_train,y_test=model_selection.train_test_split(final_pr
ocessed_data['CleanedText'].values,final_processed_data['Score'].values
,test_size=0.2,shuffle=False)
```

In [26]:
```
vectorizer=TfidfVectorizer(min_df=5)#building the vertorizer with word
 counts equal and more then 5
train_tfidf=vectorizer.fit_transform(x_train)#fitting the model on trai
ning data
print(train_tfidf.shape)
```

```
(80000, 10917)
```

In [27]:
```
from sklearn.preprocessing import StandardScaler #standarizing the trai
ning  data
x_train_data=StandardScaler( with_mean=False).fit_transform(train_tfidf
)
print(x_train_data.shape)
```

```
(80000, 10917)
```

In [28]:
```
test_tfidf=vectorizer.transform(x_test)#fitting the bow model on test d
ata
print("shape of x_test after tfidf vectorization ",test_tfidf.shape)
x_test_data=StandardScaler( with_mean=False).fit_transform(test_tfidf)#
```

```
standarizing the test data
print("shape of x_test after standardization ",x_test_data.shape)
```

```
shape of x_test after tfidf vectorization  (20000, 10917)
shape of x_test after standardization  (20000, 10917)
```

In [29]:
```
#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 [30]:
```
#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 ESTIMATOR ARE',model.best_estimator_)#printing the best_est
imator
print('AUC_SCORE FOR TEST DATA IS  ',model.score(x_test_data, y_test))
```

```
Wall time: 0 ns
BEST ESTIMATOR ARE SGDClassifier(alpha=0.1, 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)
AUC_SCORE FOR TEST DATA IS   0.895712721155
```

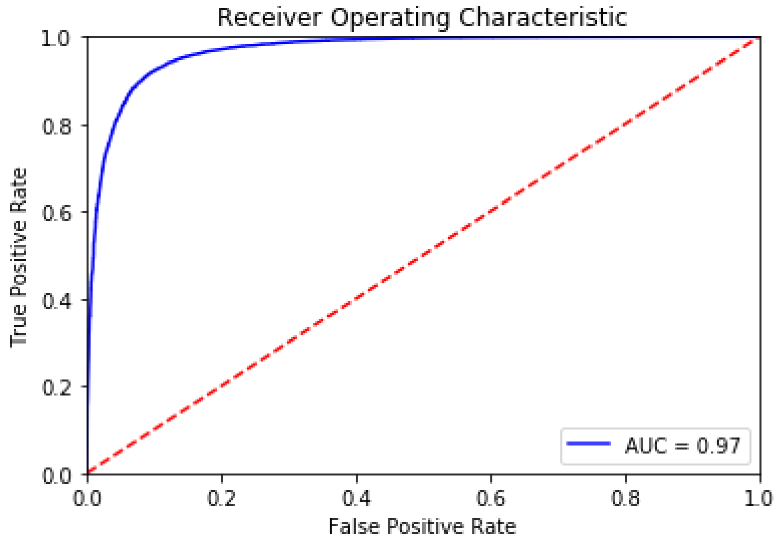
PLOTTING AUC_SCORE FOR TRAIN DATA

In [31]:

```
sgd=SGDClassifier(loss='log',class_weight={1:0.5,0:0.5},n_jobs=-1,alpha
=0.1,penalty='l2')
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_train_data)[:, 1]
```

In [32]:

```
y_pred_train=model.predict_proba(x_train_data)
fpr, tpr, threshold = metrics.roc_curve(y_train, 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 [18]:
```
print("AUC_SCORE FOR TRAIN DATA IS",roc_auc*100)
```

```
AUC_SCORE FOR TRAIN DATA IS 96.8082395135
```

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_alpha
0	0.217220	0.003818	0.934271	0.983597	0.0001
1	0.128190	0.003553	0.934151	0.988675	0.0001

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha
2	0.224778	0.003738	0.933115	0.966405	0.001
3	0.129829	0.003741	0.935490	0.989279	0.001
4	0.187672	0.003636	0.934111	0.940754	0.01
5	0.129374	0.003852	0.940891	0.990868	0.01
6	0.204964	0.003587	0.939195	0.938931	0.1
7	0.128476	0.003690	0.950613	0.984166	0.1
8	0.204750	0.003510	0.941227	0.941558	1

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha
9	0.132657	0.003898	0.943775	0.958495	1
10	0.211364	0.003663	0.941242	0.941578	10
11	0.148843	0.003535	0.941195	0.943069	10
12	0.203402	0.003696	0.753750	0.753222	100
13	0.157282	0.003559	0.941189	0.942489	100
14	0.201995	0.003641	0.846817	0.847370	1000
15	0.152189	0.003675	0.941196	0.942387	1000

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha
16	0.204081	0.003541	0.658483	0.658764	10000
17	0.147448	0.003502	0.935909	0.941572	10000

18 rows × 32 columns

In [18]:

```
results['mean_test_score']=results['mean_test_score']*100
results['mean_test_score']
```

Out[18]:

```
0     93.427100
1     93.415144
2     93.311543
3     93.549007
4     93.411092
5     94.089095
6     93.919495
7     95.061258
8     94.122695
9     94.377517
10    94.124239
11    94.119458
12    75.374970
13    94.118856
14    84.681730
15    94.119638
16    65.848330
17    93.590946
Name: mean_test_score, dtype: float64
```

In [19]:
```
results['mean_test_score']=100-results['mean_test_score']
results['mean_cv_error']=results['mean_test_score'].round(decimals=2)
results.head()
```

Out[19]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha	p
0	0.217220	0.003818	6.572900	0.983597	0.0001	l'
1	0.128190	0.003553	6.584856	0.988675	0.0001	l:
2	0.224778	0.003738	6.688457	0.966405	0.001	l'
3	0.129829	0.003741	6.450993	0.989279	0.001	l:
4	0.187672	0.003636	6.588908	0.940754	0.01	l'

5 rows × 33 columns

PLOTTING THE HEATMAP WITH

HYPERPARAMETERS FOR CV_ERROR SCORE

In [0]:
```
test_score_heatmap=results.pivot(        'param_alpha'    ,'param_penalt
y','mean_cv_error'       )
```

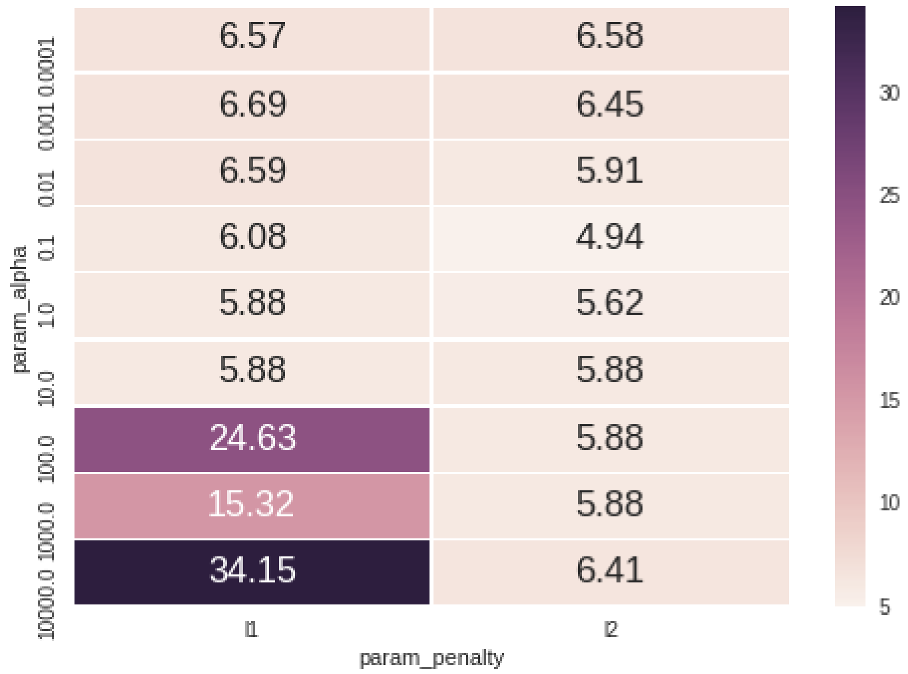
In [21]:
```
test_score_heatmap
```

Out[21]:

param_penalty	l1	l2
param_alpha		
0.0001	6.57	6.58
0.0010	6.69	6.45
0.0100	6.59	5.91
0.1000	6.08	4.94
1.0000	5.88	5.62
10.0000	5.88	5.88
100.0000	24.63	5.88
1000.0000	15.32	5.88
10000.0000	34.15	6.41

In [22]:
```
import seaborn as sns
sns.heatmap(test_score_heatmap,annot=True,annot_kws={"size": 18}, fmt=
'g',linewidths=.5)
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8d6ff33550>
```



FROM HEATMAP THE BEST HYPERPARAMETER VALUES ARE FOUND TO BE PENALTY='L2' AND 'PARAM_ALPHA'=0.1

BUILDING MODEL FOR SGD WITH CALIBRATED CLASSIFIER CV

In [19]:
```
sgd=SGDClassifier(loss='log',class_weight={1:0.5,0:0.5},n_jobs=-1,alpha
=0.1,penalty='l2')
sgd.fit(x_train_data,y_train)
```

Out[19]:
```
SGDClassifier(alpha=0.1, 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 [20]:

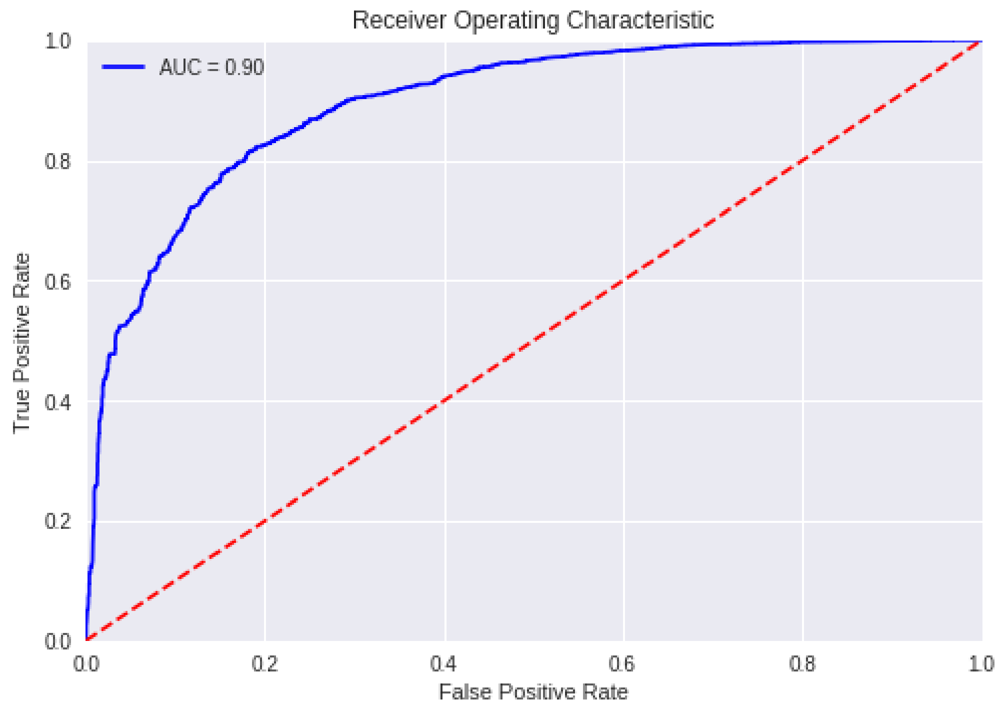
```
from sklearn.metrics import brier_score_loss
prob_pos_clf = sgd.predict_proba(x_test_data)[:, 1]

# 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: 0.074
With isotonic calibration: 0.070
With sigmoid calibration: 0.070
```

ISOTONIC CALIBRATION IS HAVING BEST VALUE FOR CALIBRATED CLASSIFIER CV

PLOTTING THE ROC CURVE FOR TEST_dATA

In [25]:
```
fpr, tpr, threshold = metrics.roc_curve(y_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 [26]:
```
print("FROM ABOVE PLOT,AUC_SCORE IS FOUND AS ",roc_auc*100)
```

```
FROM ABOVE PLOT,AUC_SCORE IS FOUND AS  89.5685503387155
```

REPRESENTING TOP IMPORTANT FEATURES USING WORDCLOUD LIBRARY

In [27]:
```
z=sgd.coef_[0]#getting the wieght of the vector
print(z)#printing the wieght of the vector
```

```
[ 0.00168642  0.00163335 -0.00478886 ...  0.00215207  0.00811962
  0.00373967]
```

In [28]:
```
a=z.argsort()
```

```
print('shape of wieght vector is:',a.shape)
top_30_positive=np.take(vectorizer.get_feature_names(),a[10887:])
top_30_negative=np.take(vectorizer.get_feature_names(),a[:30])
```

```
shape of wieght vector is: (10917,)
```

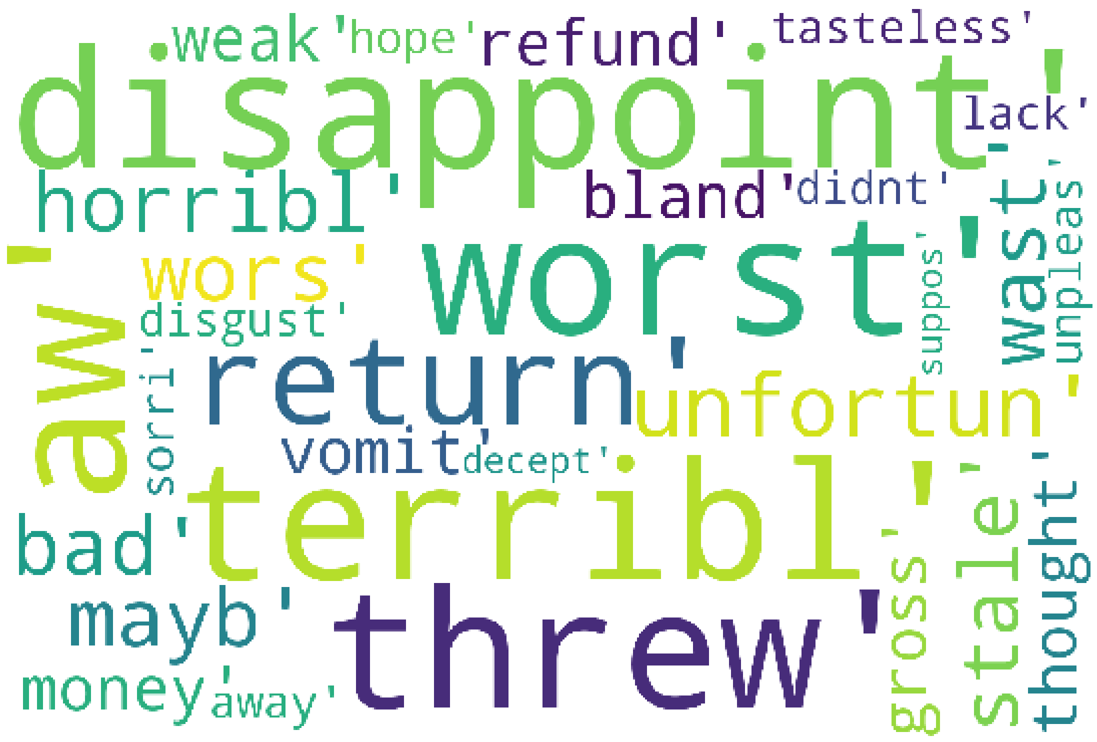
In [29]:
```
print(top_30_positive)#TOP 30 POSITIVE WORDS
print(top_30_negative)#TOP 30 NEGATIVE WORDS
```

```
['price' 'ive' 'alway' 'flavor' 'treat' 'time' 'enjoy' 'year' 'right'
 'keep' 'quick' 'tasti' 'easi' 'thank' 'high' 'tea' 'snack' 'make'
 'wonder' 'use' 'find' 'nice' 'perfect' 'excel' 'favorit' 'delici' 'bes
t'
 'good' 'love' 'great']
['disappoint' 'worst' 'terribl' 'aw' 'threw' 'return' 'unfortun' 'horri
bl'
 'mayb' 'stale' 'bad' 'wors' 'wast' 'money' 'refund' 'thought' 'bland'
 'gross' 'vomit' 'weak' 'disgust' 'tasteless' 'didnt' 'unpleas' 'lack'
 'sorri' 'hope' 'away' 'decept' 'suppos']
```

In [30]:
```
from wordcloud import WordCloud #here we are printing the top features
 using wordcloud library
import matplotlib.pyplot as plt
wordcloud = WordCloud(width = 1500, height = 1000,
                background_color ='white',

                min_font_size = 10).generate(str(top_30_positive))

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```

In [31]:

```
from wordcloud import WordCloud #here we are printing the top features
 using wordcloud library
import matplotlib.pyplot as plt
wordcloud = WordCloud(width = 1500, height = 1000,
                background_color ='white',
                min_font_size = 10).generate(str(top_30_negative))

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



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

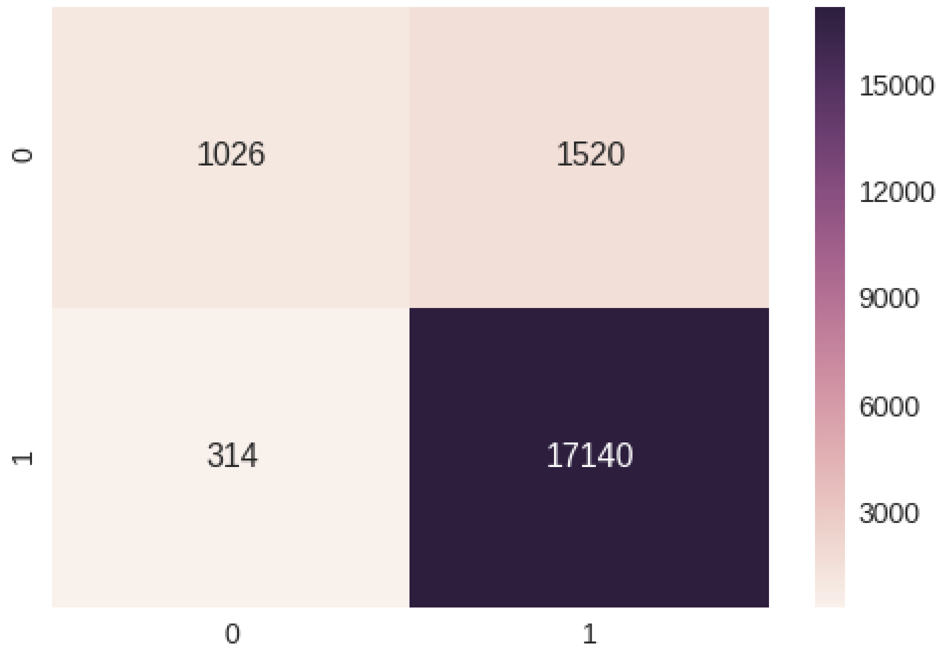
In [32]:

```
#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')
```

```
Accuracy on test set: 90.830%
Precision on test set: 0.919
Recall on test set: 0.982
F1-Score on test set: 0.949
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[32]: <matplotlib.axes._subplots.AxesSubplot at 0x7f8d6fcb1b38>



RBF KERNEL WITH TFIDF VECTORIZATION

OBJECTIVE

1. APPLYING SVM WITH RBF KERNEL WITH TFIDF 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 40K POINTS ONLY

In [2]:
```
final_data=pd.read_csv('final_data.csv',encoding='latin-1')# IMPORT THE DATA FILE
final_data.head()
```

Out[2]:

	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 [3]:
```
#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 [4]:
```
vectorizer=TfidfVectorizer(min_df=10,max_features=1000)#building the ve
rtorizer with word counts equal and more then 2
train_tfidf=vectorizer.fit_transform(x_train)#fitting the model on trai
ning data
print(train_tfidf.shape)
```

```
(21000, 1000)
```

In [5]:
```
from sklearn.preprocessing import StandardScaler #standarizing the trai
ning  data
x_train_data=StandardScaler( with_mean=False).fit_transform(train_tfidf
)
print(x_train_data.shape)
```

```
(21000, 1000)
```

In [6]:
```
test_tfidf=vectorizer.transform(x_test)#fitting the bow model on test d
ata
print("shape of x_test after tfidf vectorization ",test_tfidf.shape)
x_test_data=StandardScaler( with_mean=False).fit_transform(test_tfidf)#
standarizing the test data
print("shape of x_test after standardization ",x_test_data.shape)
```

```
shape of x_test after tfidf vectorization  (9000, 1000)
shape of x_test after standardization  (9000, 1000)
```

In [7]:
```
#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=[10**-2,10**-1,1,10**1,10**2]#range of hyperparameter
gamma_values=[10**-2,10**-1,1,10**1,10**2]#range of hyperparameter
```

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

In [8]:
```
#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 4 µs, sys: 1 µs, total: 5 µs
Wall time: 8.11 µs
```

In [9]:
```
%%time
model.fit(x_train_data, y_train)#fiitting the training data
```

```
CPU times: user 33min 20s, sys: 1.16 s, total: 33min 21s
Wall time: 2h 5min 54s
```

Out[9]:
```
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=[{'gamma': [0.01, 0.1, 1, 10, 100], 'C': [0.01, 0.1,
1, 10, 100]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)
```

In [10]:
```
model.best_estimator_
```

Out[10]:
```
SVC(C=1, cache_size=200, class_weight='balanced', coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=0.01, 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 [11]:

```
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[11]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	para
0	726.061317	46.209252	0.802086	0.798360	0.01	0.01
1	760.969195	46.820340	0.802086	0.798360	0.01	0.1
2	714.459650	44.630063	0.802086	0.798360	0.01	1
3	718.333226	44.904255	0.802086	0.798360	0.01	10
4	678.264367	42.247382	0.802086	0.798360	0.01	100
5	675.817528	43.863024	0.821742	0.834432	0.1	0.01
6	630.776486	33.962349	0.821635	0.834395	0.1	0.1

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	para
7	507.994252	34.274826	0.821563	0.834320	0.1	1
8	454.288352	28.604413	0.821563	0.834320	0.1	10
9	433.648696	29.951571	0.821563	0.834320	0.1	100
10	801.141237	22.675756	0.849541	1.000000	1	0.01
11	761.008114	22.951434	0.849115	1.000000	1	0.1
12	764.523544	22.107811	0.848999	1.000000	1	1
13	744.609939	21.263023	0.848999	1.000000	1	10
14	678.193262	20.364434	0.848999	1.000000	1	100
15	880.993728	21.503904	0.849541	1.000000	10	0.01
16	964.960491	21.441882	0.849115	1.000000	10	0.1

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	para
17	885.318333	19.688480	0.848999	1.000000	10	1
18	872.843041	19.029072	0.848999	1.000000	10	10
19	867.598465	19.356147	0.848999	1.000000	10	100
20	764.865655	20.518105	0.849541	1.000000	100	0.01
21	875.215546	15.824762	0.849115	1.000000	100	0.1
22	825.066551	15.685777	0.848999	1.000000	100	1
23	737.179261	15.661377	0.848999	1.000000	100	10
24	668.455223	15.519956	0.848999	1.000000	100	100

In [12]:
```
results['mean_test_score']=results['mean_test_score']*100
results['mean_test_score']
```

Out[12]: 0   80.208631

```
1     80.208631
2     80.208631
3     80.208631
4     80.208631
5     82.174224
6     82.163467
7     82.156300
8     82.156300
9     82.156300
10    84.954066
11    84.911542
12    84.899941
13    84.899941
14    84.899941
15    84.954066
16    84.911542
17    84.899941
18    84.899941
19    84.899941
20    84.954066
21    84.911542
22    84.899941
23    84.899941
24    84.899941
Name: mean_test_score, dtype: float64
```

In [13]:
```
results['mean_test_score']=100-results['mean_test_score']
results['mean_cv_error']=results['mean_test_score'].round(decimals=2)
results.head()
```

Out[13]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	parar
0	726.061317	46.209252	19.791369	0.79836	0.01	0.01

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	parar
1	760.969195	46.820340	19.791369	0.79836	0.01	0.1
2	714.459650	44.630063	19.791369	0.79836	0.01	1
3	718.333226	44.904255	19.791369	0.79836	0.01	10
4	678.264367	42.247382	19.791369	0.79836	0.01	100

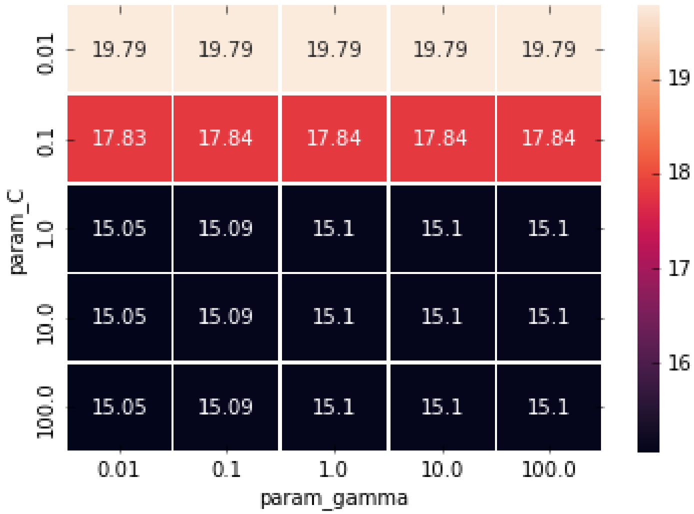
In [14]:
```
test_score_heatmap=results.pivot(       'param_C'        ,'param_gamma',
'mean_cv_error' )
```

In [15]:
```
test_score_heatmap
```

Out[15]:

param_gamma	0.01	0.1	1.0	10.0	100.0
param_C					
0.01	19.79	19.79	19.79	19.79	19.79
0.10	17.83	17.84	17.84	17.84	17.84
1.00	15.05	15.09	15.10	15.10	15.10
10.00	15.05	15.09	15.10	15.10	15.10
100.00	15.05	15.09	15.10	15.10	15.10

In [19]:
```
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()
```



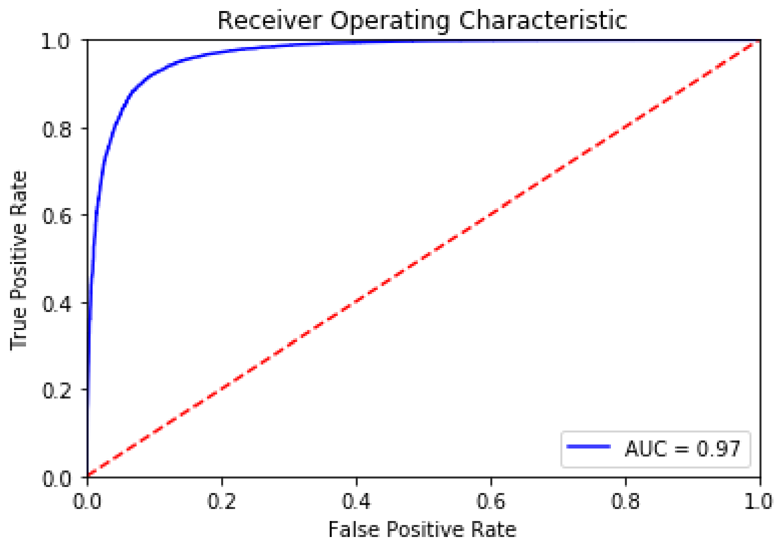
FROM HERE BEST HPYERPARAMETERS ARE GAMMA =0.01 AND C=1

In [8]:
```
svc=SVC(class_weight='balanced',probability=True,C=1,gamma=0.01)
```

In [9]:
```
#fitting the model
svc.fit(x_train_data,y_train)
probs = svc.predict_proba(x_test_data)#predicting the model
```

PLOTTING THE AUC_SCORE FOR TRAIN DATA

In [33]:
```
y_pred_train=model.predict_proba(x_train_data)
fpr, tpr, threshold = metrics.roc_curve(y_train, 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()
```



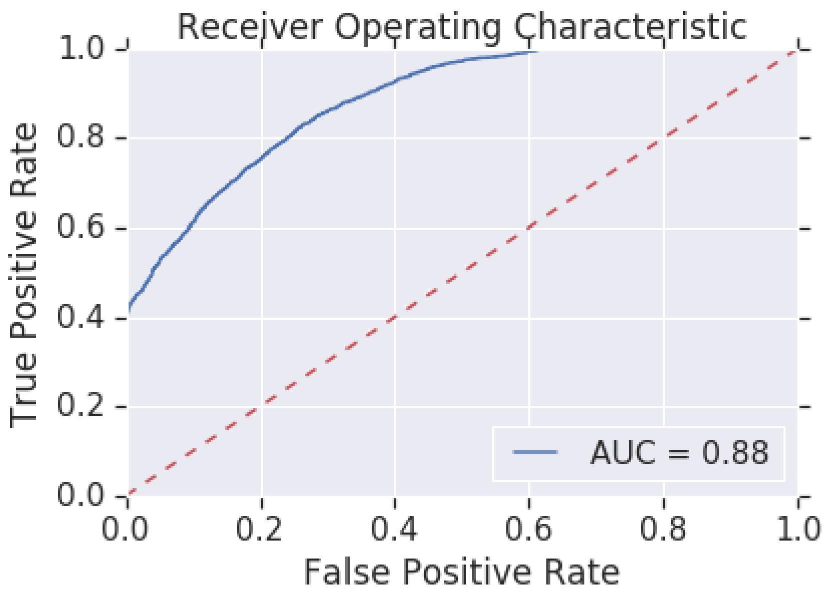
In [34]:
```
print('AUC_SCORE FOR TRAIN DATA IS',roc_auc*100)
```

```
AUC_SCORE FOR TRAIN DATA IS 96.8089099958
```

PLOTTING THE AUC_SCORE FOR TEST DATA

In [15]:

```
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()
```



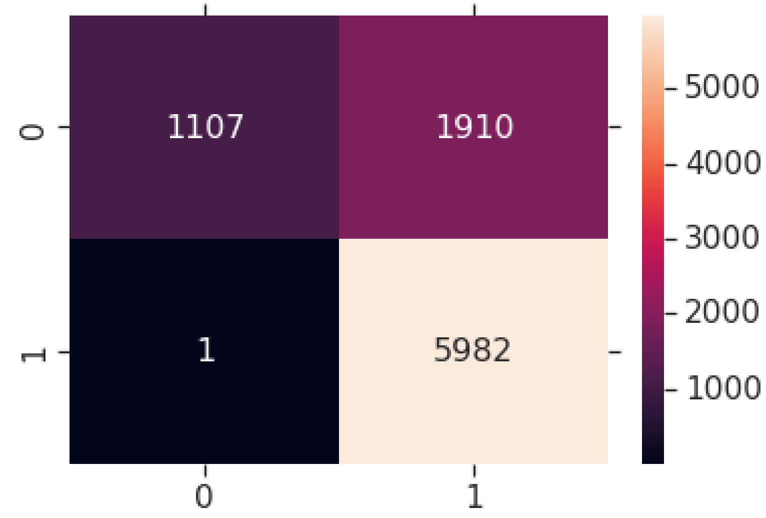
USING BEST HYPERPARAMETER VALUE ON

TEST DATA AND PLOTTING THE CONFUSION MATRIX WITH HEATMAP

In [11]:

```
#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.pyplot as plt
plt.show()
```

```
Accuracy on test set: 78.767%
Precision on test set: 0.758
Recall on test set: 1.000
F1-Score on test set: 0.862
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



TFIDF VECTORIZATION WITH SUPPORT VECTOR MACHINE WITH LINEAR KERNEL AND RBF KERNEL IS DONE