OBJECTIVE

1. APPLYING SVM WITH TFIDF 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 [2]:

```
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 [3]:

```
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 [4]:

```
# getting the counts of 0 and 1 in "SCORE" column to know whether it is
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
 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 [5]:

```
#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 [ ]:

```
# 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=100,workers=2)#co
nstructing my our word to vector
w_t_c_words=list(word_to_vector.wv.vocab)
print("****************************************************************
*******")
print("sample words ", w_t_c_words[0:20])
```

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

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
*********************************************************************
['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']
```

In [ ]:

```
###### NOW STARTING TFIDF WORD TO VEC FOR TRAIN DATA##################
##############################################
#NOW STARTING TF-IDF WEIGHTED WORD-TO-VEC
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(x_train)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
train_tfidf_sent_vectors =[]# the tfidf-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(100) # as word vectors are of zero length
  weight_sum =0; # num of words with a valid vector in the sentence/rev
iew
  for word in sent: # for each word in a review/sentence
   if word in w_t_c_words:
     vec = word_to_vector.wv[word]
     tf_idf = dictionary[word]*(sent.count(word)/len(sent))# dictionary
[word] = idf value of word in whole courpus
     sent_vec += (vec * tf_idf)# sent.count(word) = tf valeus of word i
n this review
     weight_sum += tf_idf
  if weight_sum != 0:
   sent_vec /= weight_sum
   train_tfidf_sent_vectors.append(sent_vec)
```

```
 47%|
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
| 37898/80000 [02:51<03:45, 186.83it/s]
```

In [ ]:

```
from sklearn.preprocessing import StandardScaler #standarizing the training  data
x_train_data=StandardScaler( with_mean=False).fit_transform(train_tfidf_sent_vectors)
print(x_train_data.shape)
```

In [ ]:

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

In [ ]:

```
###### NOW STARTING TFIDF WORD TO VEC FOR TEST DATA##############################################################
#NOW STARTING TF-IDF WEIGHTED WORD-TO-VEC
model = TfidfVectorizer()
model.fit_transform(x_train)
model.transform(x_test)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
test_tfidf_sent_vectors =[]# the tfidf-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(100) # as word vectors are of zero length
  weight_sum =0; # num of words with a valid vector in the sentence/review
  for word in sent: # for each word in a review/sentence
   if word in w_t_c_words:
     vec = word_to_vector.wv[word]
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
        tf_idf = dictionary[word]*(sent.count(word)/len(sent))# dictionary
[word] = idf value of word in whole courpus
        sent_vec += (vec * tf_idf)# sent.count(word) = tf valeus of word i
n this review
        weight_sum += tf_idf
    if weight_sum != 0:
      sent_vec /= weight_sum
      test_tfidf_sent_vectors.append(sent_vec)
```

In [ ]:

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

In [13]:

```
#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))
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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='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 OF TEST DATA IS 0.866824824512
```

In [14]:

```
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[14]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha
0	0.390841	0.007605	0.935846	0.938038	0.0001
1	0.256358	0.007321	0.937721	0.940642	0.0001
2	0.378334	0.008695	0.943315	0.946198	0.001
3	0.255560	0.005737	0.942203	0.945842	0.001

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha
4	0.394182	0.006039	0.941242	0.941594	0.01
5	0.253765	0.007308	0.943502	0.946110	0.01
6	0.384208	0.006834	0.941242	0.941594	0.1
7	0.259687	0.008779	0.941317	0.941594	0.1
8	0.360240	0.007955	0.941242	0.941594	1
9	0.258992	0.007090	0.941242	0.941594	1
10	0.564400	0.005683	0.941242	0.941594	10

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha
11	0.305009	0.008994	0.941233	0.941392	10
12	0.561000	0.006147	0.941242	0.941594	100
13	0.311735	0.007591	0.837677	0.838774	100
14	0.557370	0.006901	0.941242	0.941594	1000
15	0.315099	0.008764	0.941242	0.941594	1000
16	0.549220	0.007016	0.846963	0.847903	10000
17	0.304729	0.007485	0.658485	0.659720	10000

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

18 rows × 32 columns

In [15]:
```
results['mean_test_score']=results['mean_test_score']*100
results['mean_test_score']
```

Out[15]:
```
0     93.584643
1     93.772139
2     94.331540
3     94.220349
4     94.124239
5     94.350178
6     94.124239
7     94.131700
8     94.124239
9     94.124239
10    94.124239
11    94.123293
12    94.124239
13    83.767744
14    94.124239
15    94.124239
16    84.696311
17    65.848527
Name: mean_test_score, dtype: float64
```

In [16]:
```
results['mean_test_score']=100-results['mean_test_score']
results['mean_cv_error']=results['mean_test_score'].round(decimals=2)#
 creating a new row for cv_error
results.head()
```

Out[16]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha	p
0	0.390841	0.007605	6.415357	0.938038	0.0001	l'

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_alpha	p
1	0.256358	0.007321	6.227861	0.940642	0.0001	l:
2	0.378334	0.008695	5.668460	0.946198	0.001	l'
3	0.255560	0.005737	5.779651	0.945842	0.001	l:
4	0.394182	0.006039	5.875761	0.941594	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 [18]:

```
test_score_heatmap# shape of data for heatmap
# here each value of matrix represent the cv_error corresponding to val
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
ues of hyperparameters
```

Out[18]:

param_penalty	l1	l2
param_alpha		
0.0001	6.42	6.23
0.0010	5.67	5.78
0.0100	5.88	5.65
0.1000	5.88	5.87
1.0000	5.88	5.88
10.0000	5.88	5.88
100.0000	5.88	16.23
1000.0000	5.88	5.88
10000.0000	15.30	34.15

In [19]:

```
# plotting of cv error with hyperparamter values
import seaborn as sns
sns.heatmap(test_score_heatmap,annot=True,annot_kws={"size": 18}, fmt=
'g',linewidths=.5)
```

Out[19]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05e0e82080>
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

FROM HEATMAP THE BEST HYPERPARAMETER VALUES ARE FOUND TO BE PENALTY='L2' AND 'PARAM_ALPHA'=0.01

BUILDING MODEL FOR SGD WITH CALIBRATED CLASSIFIER CV

In [15]:
```
sgd=SGDClassifier(loss='log',class_weight={1:0.5,0:0.5},n_jobs=-1,alpha
=0.01,penalty='l2')
sgd.fit(x_train_data,y_train)#building the sgd model
```

Out[15]:
```
SGDClassifier(alpha=0.01, average=False, class_weight={1: 0.5, 0: 0.5},
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
              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 [16]:

```
# checking which mode of calibrated classifier is best and then using i
t
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: 0.083
With isotonic calibration: 0.082
With sigmoid calibration: 0.083
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

ISOTONIC CALIBRATION IS HAVING BEST VALUE FOR CALIBRATED CLASSIFIER CV

PLOTTING THE ROC CURVE FOR TRAIN_DATA

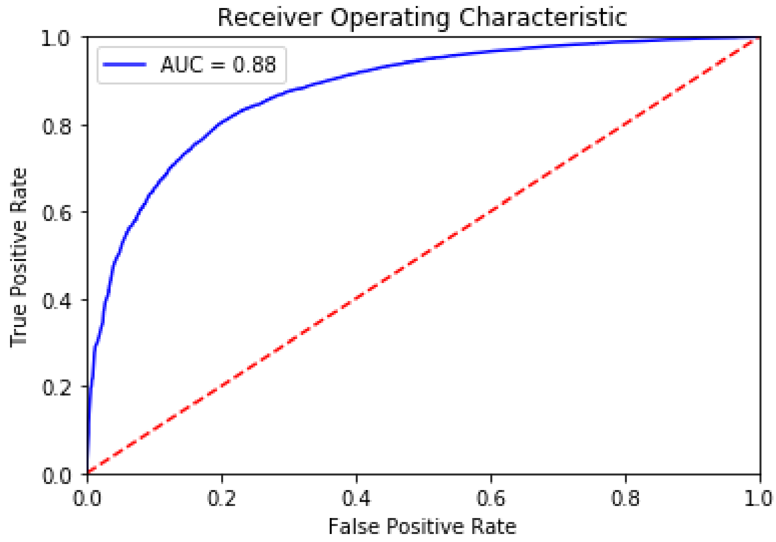
In [18]:

```
from sklearn.calibration import   CalibratedClassifierCV
clf_isotonic = CalibratedClassifierCV(sgd, cv=5, method='isotonic')
clf_isotonic.fit(x_train_data, y_train)
prob_pos_isotonic_train = clf_isotonic.predict_proba(x_train_data)[:, 1
]

fpr, tpr, threshold = metrics.roc_curve(y_train, prob_pos_isotonic_trai
n)
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()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

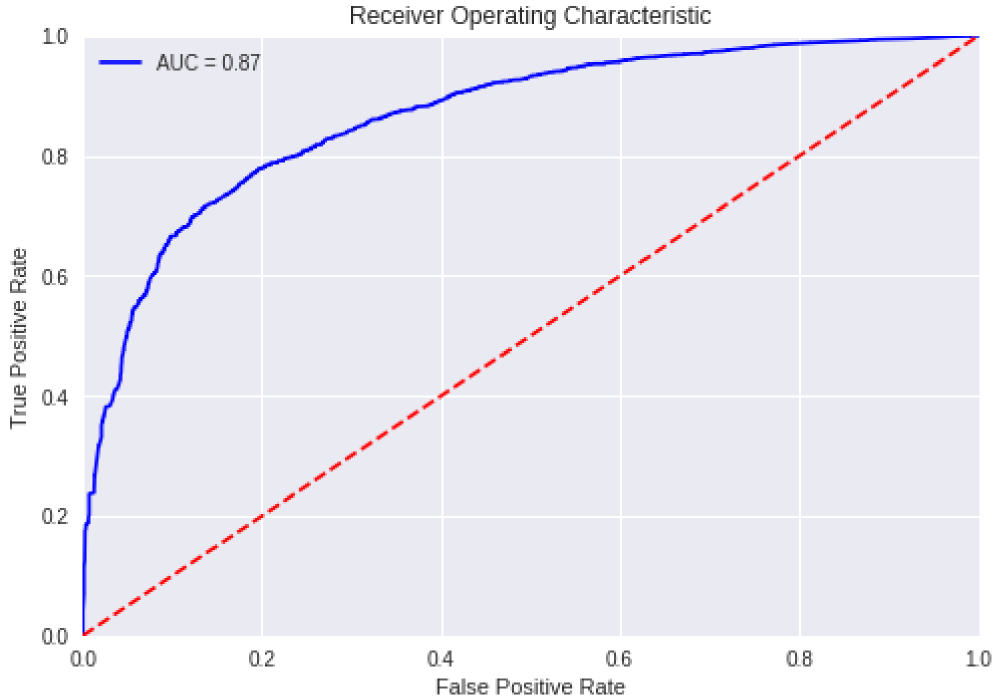


PLOTTING THE ROC CURVE FOR TEST DATA

In [22]:
```
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()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



In [23]:
```
print("FROM ABOVE PLOT,AUC_SCORE IS FOUND AS ",roc_auc*100)
```

```
FROM ABOVE PLOT,AUC_SCORE IS FOUND AS  86.67474805956105
```

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

In [24]:
```
#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
```

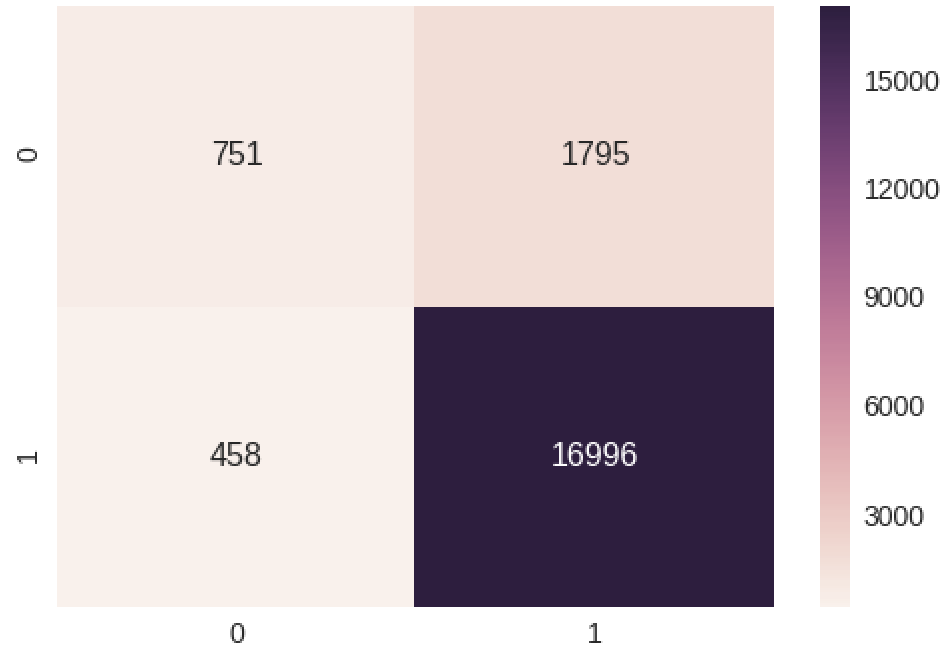
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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: 88.735%
Precision on test set: 0.904
Recall on test set: 0.974
F1-Score on test set: 0.938
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f05e1496f28>
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



RBF KERNEL WITH TFIDF AVG WORD2VEC VECTORIZATION

OBJECTIVE

1. **APPLYING SVM WITH RBF KERNEL WITH TFIDF 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**

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

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])
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
print("*****************************************************************
*")
print(list_of_sent[0])
word_to_vector=Word2Vec(list_of_sent,min_count=5,size=100,workers=2)#co
nstructing my our word to vector
w_t_c_words=list(word_to_vector.wv.vocab)
print("*****************************************************************
*******")
print("sample words ", w_t_c_words[0:20])
```

```
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']
************************************************************************
sample words  ['ecstat', 'rectifi', 'tendenc', 'heritag', 'transpir',
'cruis', 'salt', 'million', 'hunger', 'resembl', 'gluten', 'optim', 'so
metim', 'anonym', 'hydrat', 'fond', 'sumatran', 'sicker', 'meager', 'ki
ndey']
```

In [7]:

```
###### NOW STARTING TFIDF WORD TO VEC FOR TRAIN DATA###################
##############################################
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
#NOW STARTING TF-IDF WEIGHTED WORD-TO-VEC
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(x_train)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
train_tfidf_sent_vectors =[]# the tfidf-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(100) # as word vectors are of zero length
  weight_sum =0; # num of words with a valid vector in the sentence/rev
iew
  for word in sent: # for each word in a review/sentence
   if word in w_t_c_words:
     vec = word_to_vector.wv[word]
     tf_idf = dictionary[word]*(sent.count(word)/len(sent))# dictionary
[word] = idf value of word in whole courpus
     sent_vec += (vec * tf_idf)# sent.count(word) = tf valeus of word i
n this review
     weight_sum += tf_idf
  if weight_sum != 0:
   sent_vec /= weight_sum
   train_tfidf_sent_vectors.append(sent_vec)
```

```
100%|██████████| 21000/21000 [01:23<00:00, 252.54it/s]
```

In [8]:

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

```
(21000, 100)
```

In [9]:

```
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
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
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 [10]:

```
###### NOW STARTING TFIDF WORD TO VEC FOR TEST DATA###################
#############################################
#NOW STARTING TF-IDF WEIGHTED WORD-TO-VEC
model = TfidfVectorizer()
model.fit_transform(x_train)
model.transform(x_test)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))
test_tfidf_sent_vectors =[]# the tfidf-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(100) # as word vectors are of zero length
  weight_sum =0; # num of words with a valid vector in the sentence/rev
iew
  for word in sent: # for each word in a review/sentence
   if word in w_t_c_words:
     vec = word_to_vector.wv[word]
     tf_idf = dictionary[word]*(sent.count(word)/len(sent))# dictionary
[word] = idf value of word in whole courpus
     sent_vec += (vec * tf_idf)# sent.count(word) = tf valeus of word i
n this review
     weight_sum += tf_idf
  if weight_sum != 0:
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
    sent_vec /= weight_sum
    test_tfidf_sent_vectors.append(sent_vec)
```

```
100%|██████████| 9000/9000 [00:36<00:00, 246.74it/s]
```

In [11]:

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

```
(9000, 100)
```

In [14]:

```
#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 [15]:

```
#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 [16]:

```
%%time
model.fit(x_train_data, y_train)#fiitting the training data
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
CPU times: user 8min 26s, sys: 865 ms, total: 8min 27s
Wall time: 1h 55min 11s
```

Out[16]:

```
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': [0.001, 0.01, 0.1, 1, 5, 10, 100], 'C':
[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 [17]:

```
model.best_estimator_
```

Out[17]:

```
SVC(C=10, 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 [18]:

```
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.head()#printing the dataframe
```

Out[18]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	parar
0	285.671830	16.275460	0.802086	0.798360	0.001	0.001

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	parar
1	283.611784	17.385531	0.802155	0.798446	0.001	0.01
2	303.652002	16.206489	0.802086	0.798360	0.001	0.1
3	342.324184	16.511182	0.802086	0.798360	0.001	1
4	331.226701	16.413724	0.802086	0.798360	0.001	5

In [19]:

```
results['mean_test_score']=results['mean_test_score']*100#multiplying m
ean_test_score by 100
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)#
 building a new column cv_error and 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
0	285.671830	16.275460	19.791369	0.798360	0.001	0.001
1	283.611784	17.385531	19.784501	0.798446	0.001	0.01

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C	parai
2	303.652002	16.206489	19.791369	0.798360	0.001	0.1
3	342.324184	16.511182	19.791369	0.798360	0.001	1
4	331.226701	16.413724	19.791369	0.798360	0.001	5

In [20]:

```
test_score_heatmap=results.pivot('param_C','param_gamma','mean_cv_erro
r')
```

In [21]:

```
test_score_heatmap
```

Out[21]:

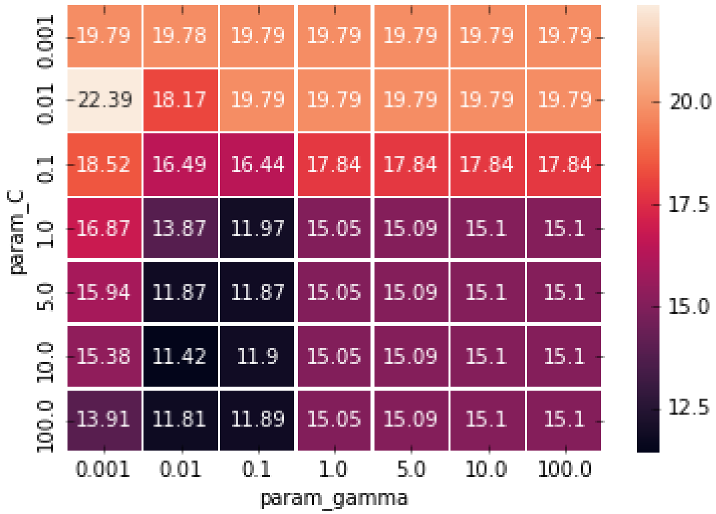
param_gamma	0.001	0.01	0.1	1.0	5.0	10.0	100.0
param_C							
0.001	19.79	19.78	19.79	19.79	19.79	19.79	19.79
0.010	22.39	18.17	19.79	19.79	19.79	19.79	19.79
0.100	18.52	16.49	16.44	17.84	17.84	17.84	17.84
1.000	16.87	13.87	11.97	15.05	15.09	15.10	15.10
5.000	15.94	11.87	11.87	15.05	15.09	15.10	15.10
10.000	15.38	11.42	11.90	15.05	15.09	15.10	15.10
100.000	13.91	11.81	11.89	15.05	15.09	15.10	15.10

In [24]:

```
import seaborn as sns
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
sns.heatmap(test_score_heatmap,annot=True,annot_kws={"size": 10}, fmt=
'g',linewidths=.5)
import matplotlib.pylab as plt
plt.show()
#plotting the heatmap
#here each value of cell contains the cv_error with given value of hype
rparameters
```



In [25]:
```
model.best_estimator_
```

Out[25]:
```
SVC(C=10, 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)
```

FROM HERE BEST HPYERPARAMETERS ARE GAMMA =0.01 AND C=10

In [26]:
```
#building the svc model
svc=SVC(class_weight='balanced',probability=True,C=10,gamma=0.01)
```

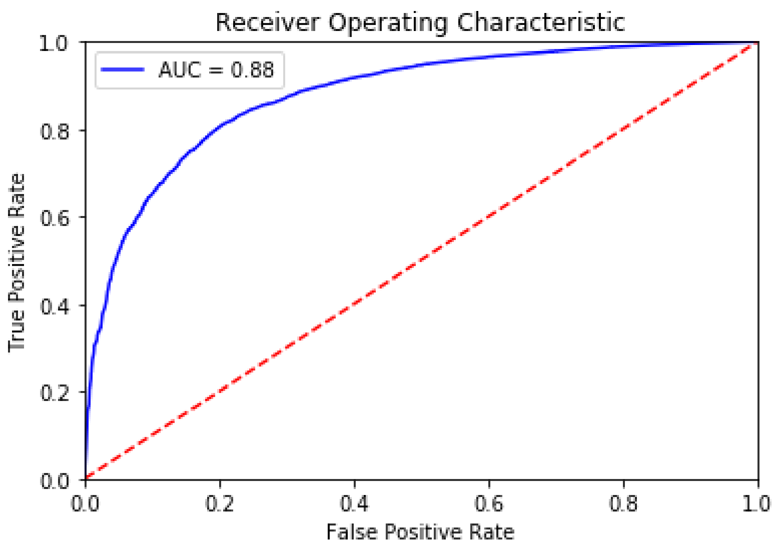
Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

PLOTTING THE ROC CURVE FOR GETTING THE TRAIN_DATA

In [19]:

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

#
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()
```



Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

PLOTTING THE ROC CURVE FOR GETTING THE TEST_DATA

In [ ]:

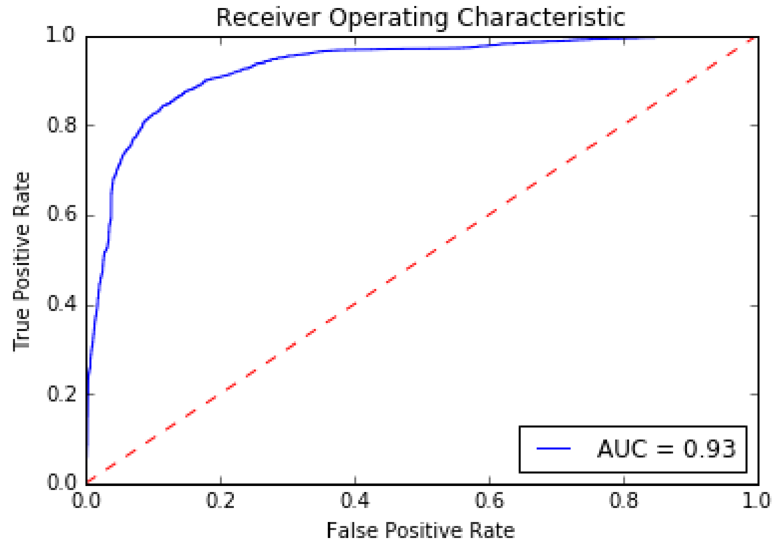
```
#fitting the model
svc.fit(x_train_data,y_train)
probs = svc.predict_proba(x_test_data)#predicting the model
y_pred = probs[:,1]
```

In [28]:

```
#plotting the curve for getting the auc_value
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()
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD



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

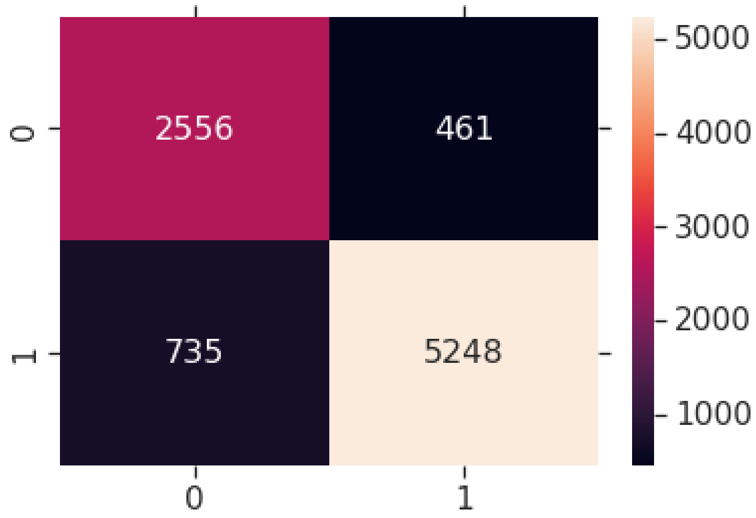
In [32]:

```
#Testing Accuracy on Test data
import seaborn as sns #importing seaborn as sns
from sklearn.metrics import *#importing varoius metrics from sklearn
y_pred=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
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

```
sns.heatmap(df_cm, annot=True,annot_kws={"size": 16}, fmt='g')
import matplotlib.pyplot as plt
plt.show()
```

```
Accuracy on test set: 86.711%
Precision on test set: 0.919
Recall on test set: 0.877
F1-Score on test set: 0.898
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```



TFIDF WORD2VEC VECTORIZATION WITH SUPPORT VECTOR MACHINE WITH LINEAR KERNEL AND RBF KERNEL IS DONE

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

TABULATING VAROIUS VECTORIZATION RESULTS WITH DIFFERENT PARAMETERS

In [1]:
```python
from tabulate import tabulate
```

In [5]:
```python
print("PERFORMANCE EVALUATION for SVM WITH LINEAR KERNEL FOR ALL VECTORIZATIONS")
table = [["BOW",'L2','0.1','95.2%','82.6%'],["TF-IDF",'L2','0.1','95.6%','89.1%'], ["AVG_WORD_2_VEC",'L2','0.001','94.1%','89.8%'],["TFIDF_AVG_WORD_2_VEC",'L2','0.01','93.8%','86.6%']]
headers=['VECTORIZATION','PENALTY','PARAM_ALPHA','F1_SCORE','ACCURACY']
print (tabulate(table, headers, tablefmt="fancy_grid"))
```

PERFORMANCE EVALUATION for SVM WITH LINEAR KERNEL FOR ALL VECTORIZATIONS

VECTORIZATION	PENALTY	PARAM_ALPHA	F1_SCORE	ACCURACY
BOW	L2	0.1	95.2%	82.6%
TF-IDF	L2	0.1	95.6%	89.1%
AVG_WORD_2_VEC	L2	0.001	94.1%	89.8%
TFIDF_AVG_WORD_2_VEC	L2	0.01	93.8%	86.6%

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

In [6]:
```python
print("PERFORMANCE EVALUATION for SVM WITH RBF KERNEL FOR ALL VECTORIZATIONS")
table = [["BOW",'.01','1','87.3%','94.2%'],["TF-IDF",'0.01','1','86.2%','88.1%'], ["AVG_WORD_2_VEC",'0.1','10','91.2%','94.3%'],["TFIDF_AVG_WORD_2_VEC",'0.01','10','89.8%','93.6%']]
headers=['VECTORIZATION','PARAM_GAMMA','PARAM_C','F1_SCORE','ACCURACY']
print (tabulate(table, headers, tablefmt="fancy_grid"))
```

PERFORMANCE EVALUATION for SVM WITH RBF KERNEL FOR ALL VECTORIZATIONS

VECTORIZATION	PARAM_GAMMA	PARAM_C	F1_SCORE	ACCURACY
BOW	0.01	1	87.3%	94.2%
TF-IDF	0.01	1	86.2%	88.1%
AVG_WORD_2_VEC	0.1	10	91.2%	94.3%
TFIDF_AVG_WORD_2_VEC	0.01	10	89.8%	93.6%

SVM ASSIGNMENT WITH LINEAR KERNEL AND RBF KERNEL IS DONE

In [9]:
```python
print('############### SVM ASSIGNMENT WITH LINEAR KERNEL AND RBF KERNEL IS DONE for ALL VECTORIZATIONS######################')
```

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD

################ SVM ASSIGNMENT WITH LINEAR KERNEL AND RBF KERNEL IS DON
E for ALL VECTORIZATIONS######################

Create PDF in your applications with the Pdfcrowd HTML to PDF API PDFCROWD