OBJECTIVE

1. APPLYING DECISION TREE WITH AVG_WORD_2_VEC VECTORIZATION
    - FINDING THE BEST HYPERPARAMETER USING GRIDSEARCHCV WITH TRAIN DATA AND CROSS-VALIDATION DATA BY PLOTTING THE RESLUTS OF VAROIUS TRAIN DATA AND CROSS VALIDATION DATA
    - USING THE APROPRIATE VALUE OF HYPERPARAMETER ,TESTING ACCURACY ON TEST DATA USING F1-SCORE
    - PLOTTING THE CONFUSION MATRIX TO GET THE PRECISOIN ,RECALL VALUE WITH HELP OF HEATMAP #

In [0]:

```
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 sklearn.tree import DecisionTreeClassifier
from gensim.models import Word2Vec
from tqdm import tqdm
```

In [4]:

```
from google.colab import drive
drive.mount('/content/gdrive')#geeting the content from the google driv
e
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remoun
t, call drive.mount("/content/gdrive", force_remount=True).
```

In [0]:
```
final_processed_data=pd.read_csv("gdrive/My Drive/final_new_data.csv")#loading the preprocessed data  with 100k points into dataframe 
```

In [6]:
```
# 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 [0]:
```
#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 [8]:
```
# 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 r
efrain hes learn whale india droop love new word book introduc silli cl
assic 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 [9]:

```
len(w_t_c_words)
```

Out[9]:

```
11428
```

In [10]:

```
###### 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 [01:37<00:00, 819.42it/s]
```

```
80000
50
```

In [11]:

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

```
(80000, 50)
```

In [12]:

```
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('*******************************************************************')
```

```
hard find item dont buy mani either came stale got way quick classic nonetheless
*****************************************************************
['hard', 'find', 'item', 'dont', 'buy', 'mani', 'either', 'came', 'stale', 'got', 'way', 'quick', 'classic', 'nonetheless']
*******************************************************************
```

In [13]:

```
###### 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 [00:26<00:00, 745.48it/s]
```

```
20000
50
```

In [14]:

```
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 [0]:

```
#using time series split method for cross-validation score
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)
from sklearn.tree import DecisionTreeClassifier
```

In [16]:

```
#biudling the model

dt=DecisionTreeClassifier(criterion='gini', splitter='best',class_weigh
t={1:.5,0:.5})
```

```
tuned_parameters=[{'max_depth':[5,7,10,15,50],'min_samples_split':[5,25
,50,100,500]}]
#applying the model of decision tree and using gridsearchcv to find the
 best hyper parameter
%time
from sklearn.model_selection import GridSearchCV
model = GridSearchCV(dt, tuned_parameters, scoring = 'f1', cv=tscv,n_jo
bs=-1)#building the gridsearchcv model
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.11 µs
```

In [17]:

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

```
CPU times: user 5.3 s, sys: 178 ms, total: 5.48 s
Wall time: 7min 7s
```

Out[17]:

```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=5),
       error_score='raise-deprecating',
       estimator=DecisionTreeClassifier(class_weight={1: 0.5, 0: 0.5}, 
criterion='gini',
            max_depth=None, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
            splitter='best'),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid=[{'max_depth': [5, 7, 10, 15, 50], 'min_samples_spli
t': [5, 25, 50, 100, 500]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1', verbose=0)
```

In [18]:

```
print(model.best_estimator_)#printing the best estimator
```

```
DecisionTreeClassifier(class_weight={1: 0.5, 0: 0.5}, criterion='gini',
            max_depth=5, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
```

```
                min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
                splitter='best')
```

In [19]:

```
print(model.score(x_test_data,y_test))#checking the score on test_Data
```

```
0.9348750304638631
```

In [20]:

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

Out[20]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
0	2.002659	0.007253	0.940868	0.948264	5
1	1.976443	0.006667	0.940868	0.948244	5
2	1.983162	0.006709	0.940883	0.948218	5
3	1.977678	0.006696	0.940776	0.947942	5
4	1.976446	0.006760	0.941019	0.946464	5

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
5	2.653299	0.007029	0.936799	0.954525	7
6	2.650946	0.007054	0.936483	0.954193	7
7	2.643516	0.007065	0.937085	0.953450	7
8	2.640853	0.007006	0.936414	0.952404	7
9	2.597897	0.006915	0.937176	0.947477	7
10	3.554496	0.007523	0.929992	0.967917	10
11	3.566285	0.007451	0.928857	0.965112	10
12	3.570510	0.007753	0.930568	0.962035	10
13	3.537582	0.007217	0.929990	0.958256	10

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
14	3.377554	0.007211	0.935202	0.948921	10
15	5.068437	0.008073	0.919053	0.986317	15
16	6.632008	0.012310	0.918959	0.975908	15
17	10.286272	0.015601	0.922272	0.968651	15
18	10.270528	0.016565	0.924815	0.961958	15
19	9.488561	0.015178	0.933946	0.949421	15
20	13.637567	0.016597	0.904959	0.996557	50
21	13.563862	0.016665	0.909945	0.980089	50
22	18.209814	0.023592	0.916139	0.970819	50

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
23	19.613257	0.021649	0.920456	0.963261	50
24	14.350410	0.016698	0.933064	0.949631	50

25 rows × 22 columns

In [0]:
```
results['mean_train_score']=results['mean_train_score']*100
results['mean_test_score']=results['mean_test_score']*100
```

In [0]:
```
results=results.round(decimals=2)
```

In [0]:
```
results['mean_test_score']=100-results['mean_test_score']
```

PLOTTING THE HEATMAP WITH HYPERPARAMETERS FOR CV_ERROR SCORE

In [0]:
```
test_score_heatmap=results.pivot(        'param_max_depth'        ,'param
_min_samples_split','mean_test_score'    )
```

In [29]:
```
test_score_heatmap
```

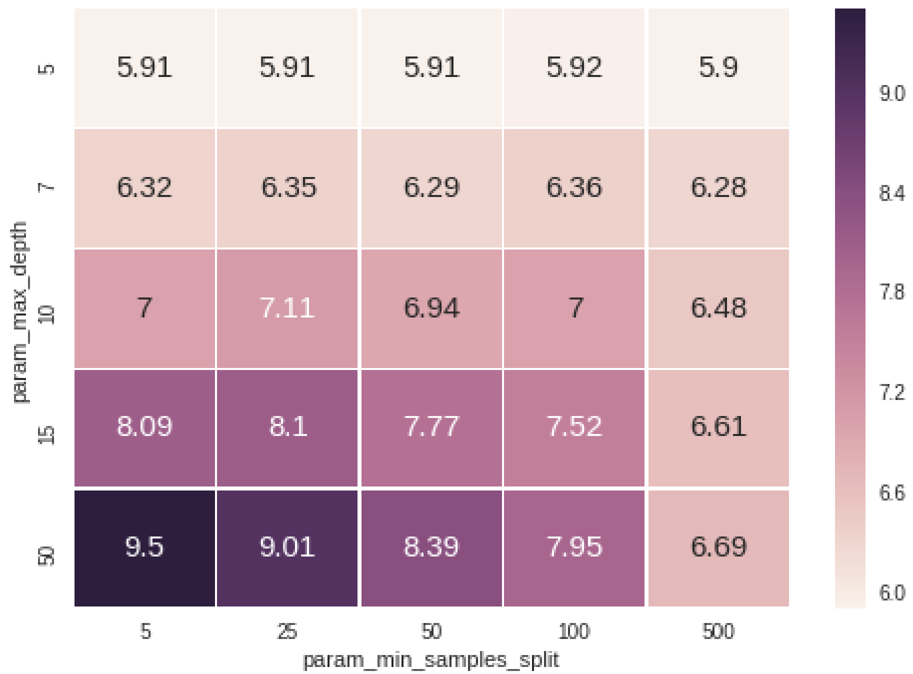
Out[29]:

param_min_samples_split	5	25	50	100	500
param_max_depth					

param_min_samples_split	5	25	50	100	500
param_max_depth					
5	5.91	5.91	5.91	5.92	5.90
7	6.32	6.35	6.29	6.36	6.28
10	7.00	7.11	6.94	7.00	6.48
15	8.09	8.10	7.77	7.52	6.61
50	9.50	9.01	8.39	7.95	6.69

In [30]:
```
import seaborn as sns
sns.heatmap(test_score_heatmap,annot=True,annot_kws={"size": 15}, fmt=
'g',linewidths=.3)
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff36562f860>



In [31]:
```
print(model.best_estimator_)#printing the best estimator
```

```
DecisionTreeClassifier(class_weight={1: 0.5, 0: 0.5}, criterion='gini',
            max_depth=5, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=500,
            min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
            splitter='best')
```

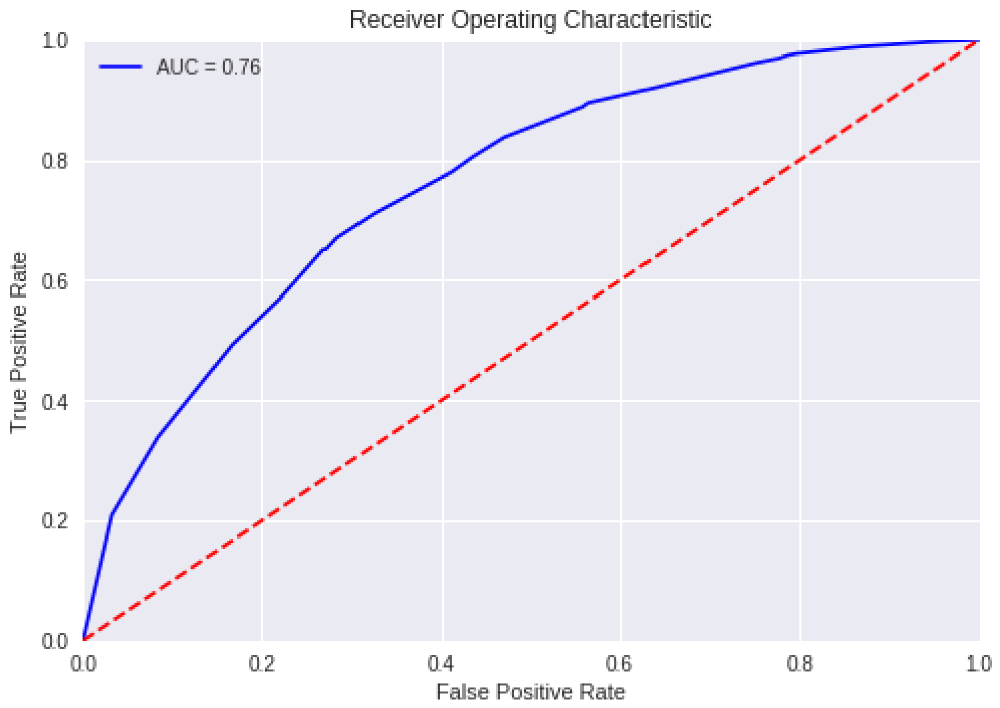
FROM THE ABOVE HEATMAPS RESULTS FOR CV DATA,WE FOUND THAT BEST HYPERPARAMETERS AS MAX_DEPTH=5 AND MIN_SAMPLE_SPLIT=500

PLOTTING THE ROC CURVE FOR GETTING AUC SCORE

In [32]:
```
probs = model.predict_proba(x_test_data)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
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 FROM THE PLOT IS FOUND AS ",roc_auc*100)
```

```
AUC SCORE FROM THE PLOT IS FOUND AS  76.0695896321256
```

TESTING OUR MODEL ON TEST DATA AND CHECKING ITS PRECISION ,RECALL ,F1_FCORE

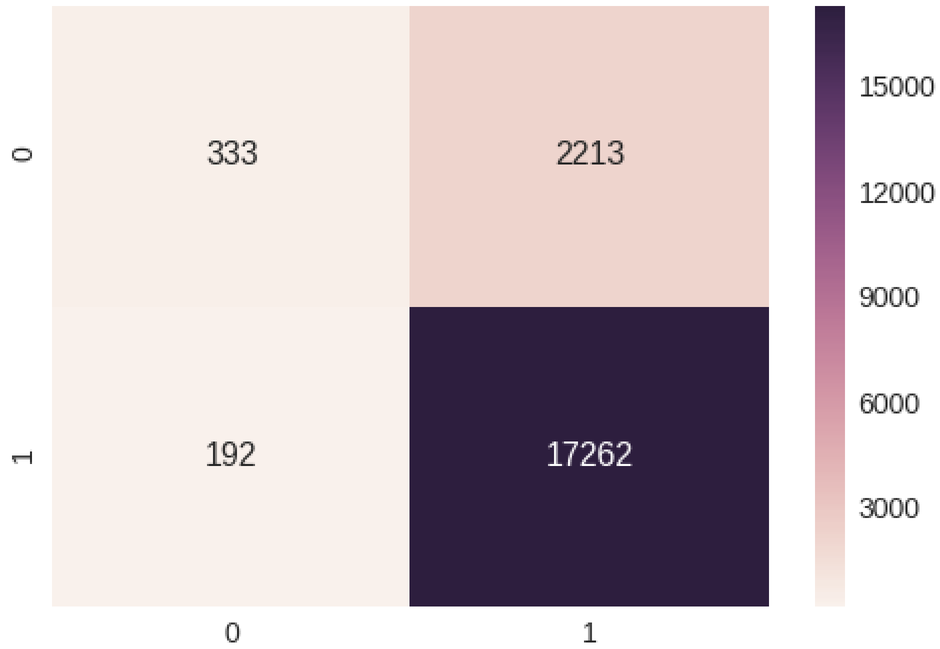
In [38]:
```
#Testing Accuracy on Test data
dt=DecisionTreeClassifier(criterion='gini', splitter='best',class_weigh
t={1:.5,0:.5},min_samples_split=500,max_depth=5)
```

```python
dt.fit(x_train_data,y_train)#fitting the model
import seaborn as sns #importing seaborn as sns
from sklearn.metrics import *#importing varoius metrics from sklearn
#building the model
y_pred = dt.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: 87.975%
Precision on test set: 0.886
Recall on test set: 0.989
F1-Score on test set: 0.935
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff365404940>
```



AVG WORD2 VECTORIZATION FOR DECISION TREE IS COMPLETED

In [0]:
```
#avg_word_2_vertorization_is_completed_for_decision_trees
```