OBJECTIVE

1. APPLYING DECISION TREE WITH TFIDF 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
    - PRINTING THE TOP 30 MOST IMPORTANT FEATURES #

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
```

In [2]:
```
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 [4]:
```
# 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_pr
ocessed_data['CleanedText'].values,final_processed_data['Score'].values
,test_size=0.2,shuffle=False)
```

In [6]:
```
vectorizer=TfidfVectorizer(min_df=2)#building the vertorizer 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)
```

```
(80000, 17204)
```

In [7]:
```
test_tfidf=vectorizer.transform(x_test)#fitting the bow model on test d
ata
print("shape of x_test after bow vectorization ",test_tfidf.shape)
```

```
shape of x_test after bow vectorization  (20000, 17204)
```

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 [9]:
```
#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 3 µs, sys: 0 ns, total: 3 µs
Wall time: 6.44 µs
```

In [10]:
```
%%time
model.fit(train_tfidf, y_train)#fiitting the training data
```

```
CPU times: user 7.53 s, sys: 193 ms, total: 7.72 s
Wall time: 11min 16s
```

Out[10]:
```
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 [11]:

```
print(model.best_estimator_)#printing the best_estimator
```

```
DecisionTreeClassifier(class_weight={1: 0.5, 0: 0.5}, criterion='gini',
            max_depth=10, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=25,
            min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
            splitter='best')
```

In [12]:

```
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[12]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
0	3.717103	0.049752	0.941827	0.946548	5
1	3.880105	0.050026	0.941762	0.946510	5
2	4.041946	0.045297	0.941781	0.946444	5
3	3.664768	0.047388	0.941798	0.946302	5

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
4	3.590582	0.047712	0.942471	0.945038	5
5	5.138468	0.051349	0.941456	0.949036	7
6	5.104313	0.048833	0.941458	0.948932	7
7	4.974762	0.045080	0.941550	0.948790	7
8	4.956825	0.046148	0.941510	0.948440	7
9	5.319208	0.057799	0.941709	0.946268	7
10	7.272699	0.045531	0.942963	0.953337	10
11	7.451539	0.049989	0.943004	0.953036	10
12	11.148507	0.066126	0.942957	0.952550	10

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
13	11.771657	0.073249	0.942868	0.951835	10
14	9.066739	0.064905	0.941934	0.947957	10
15	14.256550	0.061187	0.942603	0.958822	15
16	9.672854	0.045246	0.942587	0.958137	15
17	7.362329	0.033603	0.942368	0.957160	15
18	7.360886	0.042999	0.942360	0.955669	15
19	6.901044	0.032926	0.941300	0.950367	15
20	24.837026	0.044877	0.936484	0.979441	50
21	25.553242	0.044001	0.935630	0.976085	50

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
22	27.006873	0.042103	0.935194	0.973301	50
23	24.532741	0.039900	0.935919	0.969236	50
24	19.418280	0.039892	0.935051	0.959868	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 [22]:

```
test_score_heatmap
```

Out[22]:

param_min_samples_split	5	25	50	100	500
param_max_depth					
5	5.82	5.82	5.82	5.82	5.75
7	5.85	5.85	5.84	5.85	5.83
10	5.70	5.70	5.70	5.71	5.81
15	5.74	5.74	5.76	5.76	5.87
50	6.35	6.44	6.48	6.41	6.49

In [23]:
```
import seaborn as sns
sns.heatmap(test_score_heatmap,annot=True,annot_kws={"size": 15}, fmt=
'g',linewidths=.3)
```

Out[23]: `<matplotlib.axes._subplots.AxesSubplot at 0x7f4ab99861d0>`

In [19]:
```
print(model.best_estimator_)#printing the best_estimator
```

```
DecisionTreeClassifier(class_weight={1: 0.5, 0: 0.5}, criterion='gini',
            max_depth=10, max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=25,
            min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
            splitter='best')
```

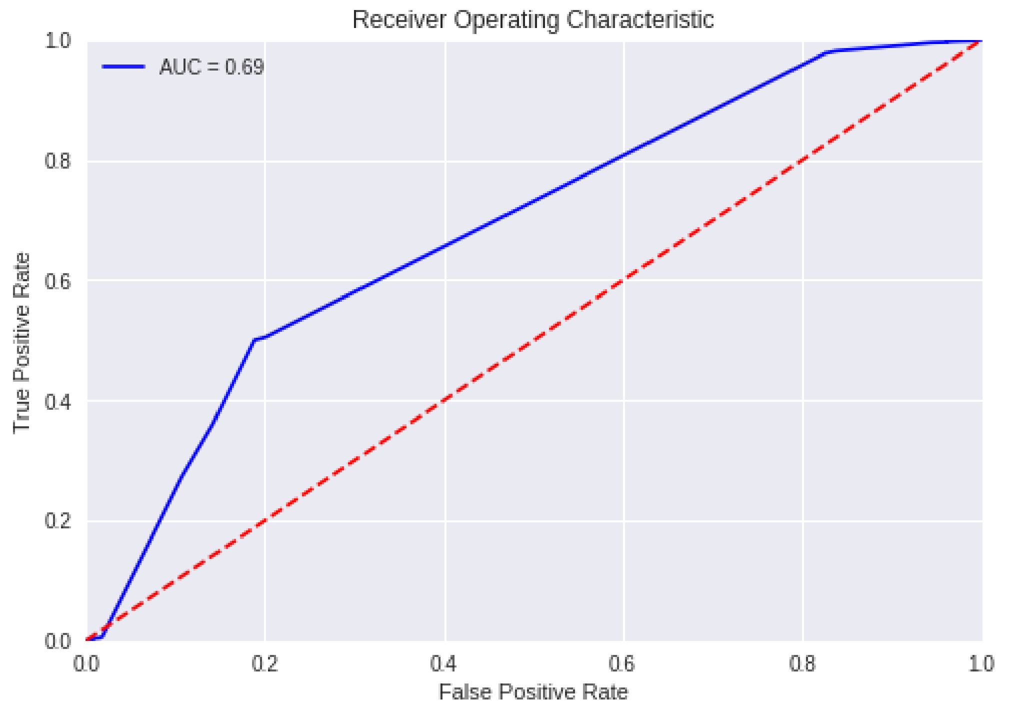
FROM THE ABOVE HEATMAPS RESULTS FOR CV DATA,WE FOUND THAT BEST HYPERPARAMETERS AS MAX_DEPTH=10 AND MIN_SAMPLE_SPLIT=25

PLOTTING THE ROC CURVE FOR GETTING AUC SCORE

In [24]:
```
probs = model.predict_proba(test_tfidf)
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 [25]:
```
print('FROM THE ABOVE CURVE ,AUC SCORE IS FOUND AS',roc_auc*100)
```

```
FROM THE ABOVE CURVE ,AUC SCORE IS FOUND AS 68.58426022265147
```

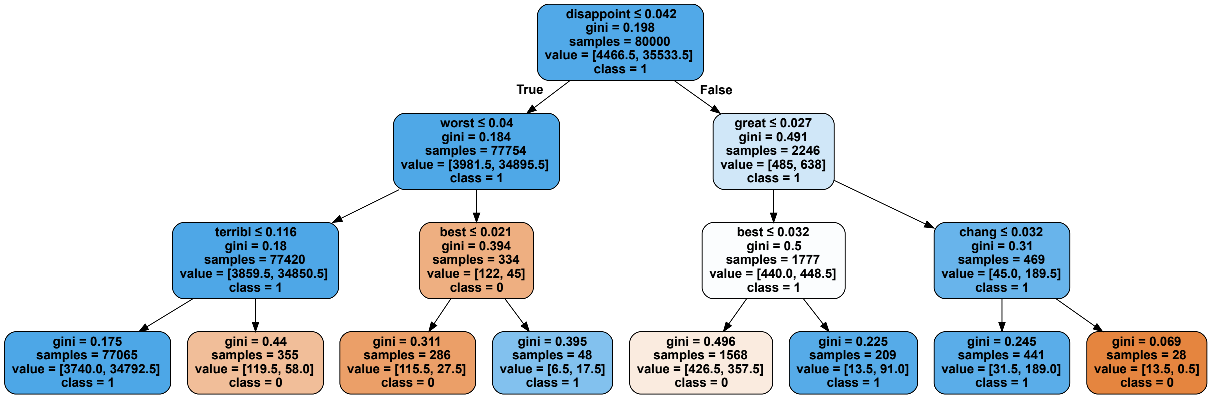
VISUALIZING DECISION TREE WITH GRAPHVIZ,FOR PLOTTING PURPOSE TAKING MAX_DEPTH AS 3

In [38]:
```
dt=DecisionTreeClassifier(criterion='gini', splitter='best',class_weigh
t={1:.5,0:.5},min_samples_split=10,max_depth=3)
dt.fit(train_tfidf,y_train)#fitting the model
```

Out[38]:
```
DecisionTreeClassifier(class_weight={1: 0.5, 0: 0.5}, criterion='gini',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, presort=False, random_state=N
one,
                       splitter='best')
```

In [39]:
```
import graphviz
target=['0','1']
dot_data = tree.export_graphviz(dt,out_file=None,feature_names=vectoriz
er.get_feature_names(),class_names=target,filled=True,rounded=True,spec
ial_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Out[39]:



In [28]:
```
graph.render("gdrive/My Drive/decision_tree_tfidf")
```

Out[28]:
```
'gdrive/My Drive/decision_tree_tfidf.pdf'
```

REPRESENTING TOP IMPORTANT FEATURES USING WORDCLOUD LIBRARY

In [34]:
```
dt=DecisionTreeClassifier(criterion='gini', splitter='best',class_weigh
```

```
t={1:.5,0:.5},min_samples_split=25,max_depth=10)
dt.fit(train_tfidf,y_train)#fitting the model
z=dt.feature_importances_
a=z.argsort()
print('shape of wieght vector is:',a.shape)
top_features=np.take(vectorizer.get_feature_names(),a[17180:])#taking l
ast features as they are of very high importance
```

```
shape of wieght vector is: (17204,)
```

In [35]:

```
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_features))

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

plt.show()
```

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

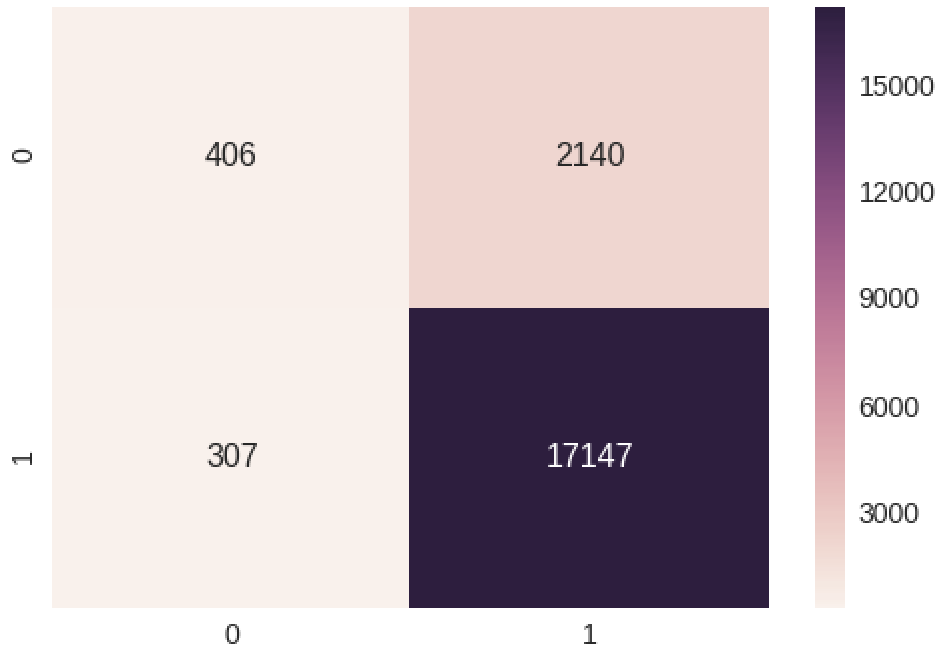
In [36]:

```
#Testing Accuracy on Test data
import seaborn as sns #importing seaborn as sns
from sklearn.metrics import *#importing varoius metrics from sklearn
#building the model
y_pred = dt.predict(test_tfidf)
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: 87.765%
Precision on test set: 0.889
Recall on test set: 0.982
F1-Score on test set: 0.933
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x7f4ab4a89048>



TFIDF VECTORIZATION FOR DECISION TREE IS COMPLETED

In [0]:
```
#TFIDF vertorization is completed for decision_trees
```