OBJECTIVE

1. **APPLYING RANDOM FOREST WITH BOW 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.ensemble import RandomForestClassifier
```

In [20]:

```
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 [22]:
```
# 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=0.3,shuffle=False)
```

In [92]:
```
vectorizer=CountVectorizer(min_df=2)#building the vertorizer with word
 counts equal and more then 2
train_bow=vectorizer.fit_transform(x_train)#fitting the model on traini
ng data
print(train_bow.shape)
```

```
(70000, 16382)
```

In [93]:
```
test_bow=vectorizer.transform(x_test)#fitting the bow model on test dat
a
print("shape of x_test after bow vectorization ",test_bow.shape)
```

```
shape of x_test after bow vectorization  (30000, 16382)
```

In [0]:

```

```

In [94]:

```
#biudling the model
#using time series split method for cross-validation score
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)
rf=RandomForestClassifier(criterion='gini',class_weight={1:.5,0:.5})
tuned_parameters=[{'max_depth':[61,64,68,73,77,80],'n_estimators':[21,3
0,35,40,45,50]}]
#applying the model of decision tree and using gridsearchcv to find the
 best hyper parameter
%time
from sklearn.model_selection import GridSearchCV
model = GridSearchCV(rf, 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: 7.87 µs
```

In [95]:

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

```
CPU times: user 18.7 s, sys: 208 ms, total: 18.9 s
Wall time: 25min 2s
```

Out[95]:

```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=5),
       error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=
{1: 0.5, 0: 0.5},
            criterion='gini', max_depth=None, max_features='auto',
            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,
            n_estimators='warn', n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid=[{'max_depth': [61, 64, 68, 73, 77, 80], 'n_estimator
s': [21, 30, 35, 40, 45, 50]}],
```

```
                 pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
                 scoring='f1', verbose=0)
```

In [96]:
```
print(model.best_estimator_)#printing the best_estimator
```

```
RandomForestClassifier(bootstrap=True, class_weight={1: 0.5, 0: 0.5},
            criterion='gini', max_depth=77, max_features='auto',
            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,
            n_estimators=35, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
```

In [106]:
```
print(model.score(test_bow,y_test))#checking the score on test_Data
```

```
0.9380587825900744
```

In [107]:
```
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[107]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
0	7.693781	0.126196	0.943111	0.970668	61
1	10.929001	0.169017	0.942684	0.969995	61

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
2	12.851338	0.198344	0.942826	0.970563	61
3	14.822200	0.223285	0.942867	0.970194	61
4	16.404032	0.246995	0.942496	0.970630	61
5	18.451207	0.272454	0.942391	0.970166	61
6	8.061914	0.127674	0.942987	0.971677	64
7	11.573845	0.175007	0.942734	0.971318	64
8	13.473842	0.201682	0.942827	0.972150	64

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
9	15.423412	0.228173	0.943077	0.972066	64
10	17.217195	0.250988	0.942758	0.971803	64
11	19.453093	0.278592	0.942536	0.971756	64
12	8.525478	0.132627	0.943196	0.973631	68
13	12.238119	0.175908	0.943114	0.973612	68
14	14.318984	0.208440	0.943126	0.973527	68
15	16.243402	0.234396	0.943139	0.973558	68

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
16	18.247492	0.260157	0.943030	0.973599	68
17	20.666725	0.288669	0.942924	0.973688	68
18	9.186626	0.135550	0.943606	0.975151	73
19	13.311219	0.186319	0.943350	0.975634	73
20	15.180538	0.211716	0.943150	0.975475	73
21	17.508943	0.241048	0.943122	0.976006	73
22	19.411326	0.268782	0.943314	0.975545	73

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
23	21.840919	0.297235	0.943257	0.975653	73
24	9.778775	0.141616	0.943868	0.977652	77
25	13.969068	0.193177	0.943720	0.977494	77
26	16.430060	0.223109	0.944037	0.977108	77
27	18.301833	0.246289	0.943744	0.977534	77
28	20.481514	0.276714	0.943328	0.976954	77
29	22.827447	0.306357	0.943203	0.977366	77

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
30	9.886808	0.141406	0.944000	0.978311	80
31	14.167045	0.194468	0.943674	0.977792	80
32	16.406874	0.221186	0.943401	0.978075	80
33	18.961226	0.253323	0.943534	0.978649	80
34	21.339856	0.281226	0.943406	0.978393	80
35	22.205270	0.291927	0.943865	0.978379	80

36 rows × 22 columns

In [0]:

```
results['mean_test_score']=results['mean_test_score']*100 #multiplying
 mean_test_score by 100
```

In [0]:
```
results=results.round(decimals=2)# rounding off to 2 decimal places
results['cv_error_score']=100-results['mean_test_score']# generating a
 new colum for getting cv_Errorscores and using it in pivottable
```

PLOTTING THE HEATMAP WITH HYPERPARAMETERS FOR CV_ERROR SCORE

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

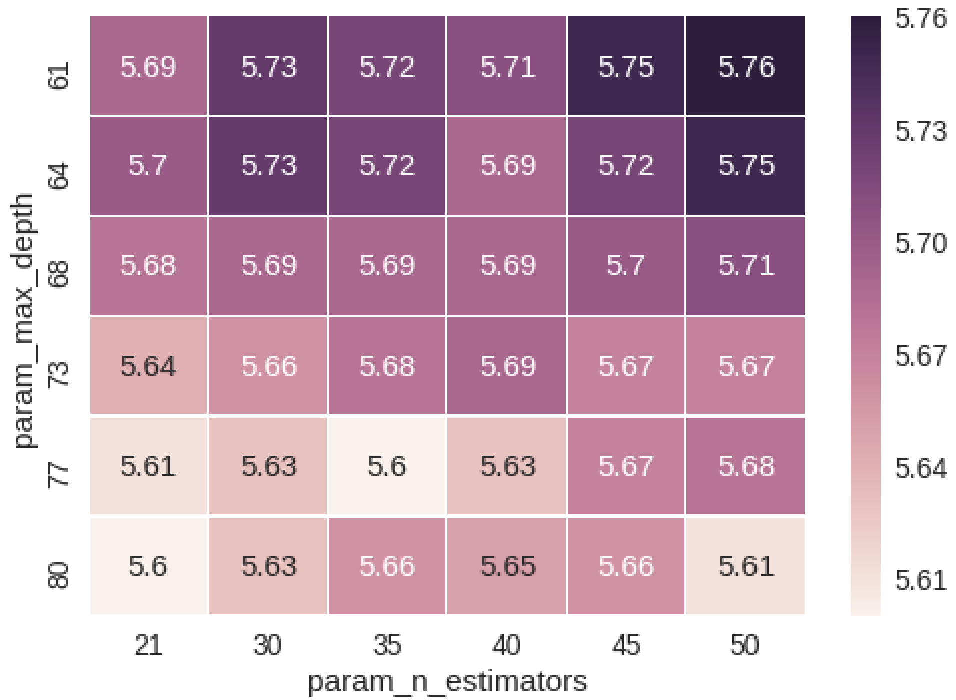
In [112]:
```
test_score_heatmap
```

Out[112]:

param_n_estimators	21	30	35	40	45	50
param_max_depth						
61	5.69	5.73	5.72	5.71	5.75	5.76
64	5.70	5.73	5.72	5.69	5.72	5.75
68	5.68	5.69	5.69	5.69	5.70	5.71
73	5.64	5.66	5.68	5.69	5.67	5.67
77	5.61	5.63	5.60	5.63	5.67	5.68
80	5.60	5.63	5.66	5.65	5.66	5.61

In [113]:
```
import seaborn as sns
sns.heatmap(test_score_heatmap,annot=True,annot_kws={"size": 15}, fmt=
'g',linewidths=.3)
```

Out[113]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7d024dd400>
```



In [114]:
```
print(model.best_estimator_)#printing the best_estimator
```

```
RandomForestClassifier(bootstrap=True, class_weight={1: 0.5, 0: 0.5},
            criterion='gini', max_depth=77, max_features='auto',
            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,
            n_estimators=35, n_jobs=None, oob_score=False,
            random_state=None, verbose=0, warm_start=False)
```

FROM THE ABOVE HEATMAPS RESULTS FOR CV DATA,WE FOUND THAT BEST HYPERPARAMETERS AS MAX_DEPTH=77 AND MIN_SAMPLE_SPLIT=35

PLOTTING THE ROC CURVE FOR GETTING AUC SCORE

In [116]:

```
rf=RandomForestClassifier(criterion='gini',class_weight={1:.5,0:.5},max
_depth=77 ,n_estimators=35)
rf.fit(train_bow,y_train)#fitting the model
probs = rf.predict_proba(test_bow)
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 [117]:
```
print('accuracy from the ROC curve is found as ',roc_auc*100)
```

```
accuracy from the ROC curve is found as  86.5744784708663
```

In [118]:
```
z=rf.feature_importances_
a=z.argsort()
print('shape of wieght vector is:',a.shape)
top_features=np.take(vectorizer.get_feature_names(),a[16360:])#taking l
ast features as they are of very high importance
```

```
shape of wieght vector is: (16382,)
```

In [119]:
```
print(top_features)#printing the top_features
top=list(top_features)
```

```
['product' 'unfortun' 'good' 'wors' 'tast' 'delici' 'bad' 'mayb' 'mone
```

```
y'
 'would' 'receiv' 'refund' 'horribl' 'return' 'worst' 'best' 'threw'
 'terribl' 'love' 'aw' 'great' 'disappoint']
```

REPRESENTING TOP IMPORTANT FEATURES USING WORDCLOUD LIBRARY

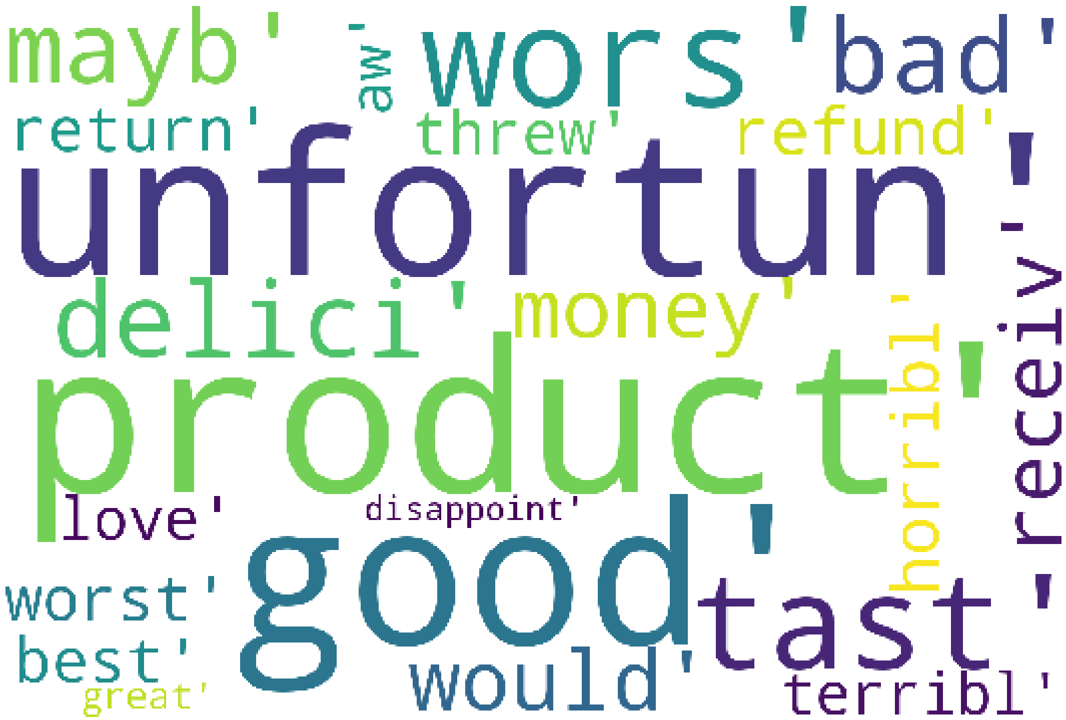
In [120]:

```
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))

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

plt.show()
```



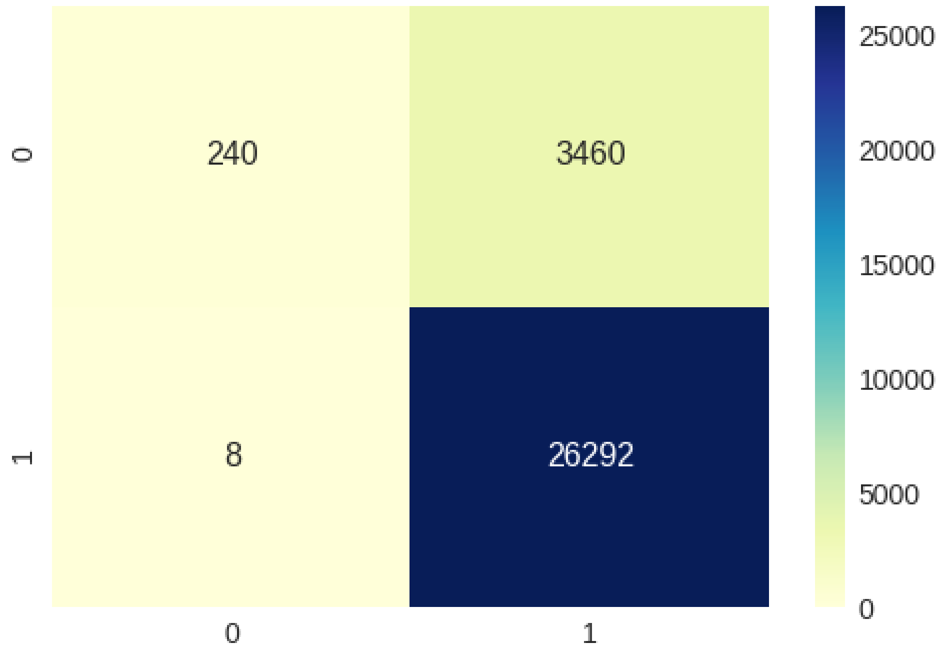
TESTING OUR MODEL ON TEST DATA AND CHECKING ITS PRECISION ,RECALL ,F1_FCORE

In [121]:
```
#Testing Accuracy on Test data
import seaborn as sns #importing seaborn as sns
from sklearn.metrics import *#importing varoious metrics from sklearn
#building the model
y_pred = rf.predict(test_bow)
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',cmap="YlG
nBu")
```

```
Accuracy on test set: 88.440%
Precision on test set: 0.884
Recall on test set: 1.000
F1-Score on test set: 0.938
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[121]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7d023aa550>



BOW VECTORIZATION FOR RANDOM FOREST IS COMPLETED

OBJECTIVE

1. **APPLYING GBDT WITH BOW 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 [79]:
```
from xgboost import XGBClassifier
#biudling the model
#using time series split method for cross-validation score
from sklearn.model_selection import TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)
xg=XGBClassifier(n_jobs=-1)
tuned_parameters=[{'max_depth':[11,15,20,24,27,30],'n_estimators':[21,3
0,35,40,45,50]}]
#applying the model of decision tree and using gridsearchcv to find the
 best hyper parameter
%time
from sklearn.model_selection import GridSearchCV
model = GridSearchCV(xg, 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: 7.87 µs
```

In [81]:
```
%%time
model.fit(train_bow, y_train)#fiitting the training data
```

```
CPU times: user 1min 37s, sys: 379 ms, total: 1min 38s
Wall time: 46min 2s
```

Out[81]:
```
GridSearchCV(cv=TimeSeriesSplit(max_train_size=None, n_splits=5),
       error_score='raise-deprecating',
       estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsam
ple_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=
0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=-1, nthread=None, objective='binary:logistic',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid=[{'max_depth': [11, 15, 20, 24, 27, 30], 'n_estimator
s': [21, 30, 35, 40, 45, 50]}],
```

```
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
             scoring='f1', verbose=0)
```

In [82]:
```
print(model.best_estimator_)#printing the best_estimator
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=
0,
       max_depth=27, min_child_weight=1, missing=None, n_estimators=50,
       n_jobs=-1, nthread=None, objective='binary:logistic',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1)
```

In [83]:
```
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[83]:

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
0	8.378044	0.223508	0.945871	0.957951	11
1	11.744793	0.234820	0.946986	0.960688	11
2	13.630929	0.243391	0.947529	0.962423	11

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
3	15.404321	0.254485	0.948213	0.963805	11
4	17.146619	0.261241	0.948735	0.965392	11
5	19.081864	0.270164	0.949192	0.967105	11
6	11.781351	0.240545	0.947026	0.963850	15
7	16.268544	0.254496	0.948179	0.967263	15
8	18.858186	0.269509	0.948571	0.969465	15
9	21.280204	0.278331	0.949295	0.971127	15

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
10	24.122753	0.291332	0.949752	0.973010	15
11	26.368965	0.303199	0.950301	0.974599	15
12	15.796100	0.258495	0.947925	0.970377	20
13	22.099479	0.278792	0.948880	0.974037	20
14	25.815812	0.296917	0.949754	0.976184	20
15	29.185396	0.311018	0.950857	0.977981	20
16	32.608864	0.330868	0.951227	0.979562	20

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
17	36.093679	0.347879	0.951690	0.980947	20
18	19.114895	0.268711	0.948154	0.973932	24
19	26.965733	0.301868	0.949403	0.978179	24
20	31.717748	0.318615	0.950061	0.979911	24
21	35.838385	0.337649	0.950983	0.981705	24
22	39.751508	0.354716	0.951516	0.983142	24
23	43.892469	0.376672	0.951935	0.984425	24

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
24	21.777656	0.279111	0.948460	0.976372	27
25	30.553274	0.310690	0.949773	0.980333	27
26	35.477301	0.333657	0.950606	0.982152	27
27	40.452885	0.357649	0.951335	0.983875	27
28	45.264877	0.379433	0.952002	0.985435	27
29	49.976577	0.401949	0.952426	0.986562	27
30	24.410512	0.281754	0.948172	0.978369	30

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_max_d
31	33.874339	0.324059	0.949829	0.982440	30
32	39.416312	0.350812	0.950284	0.984381	30
33	44.859274	0.370497	0.951061	0.985891	30
34	50.118966	0.398341	0.951202	0.987295	30
35	51.189381	0.393930	0.951405	0.988397	30

36 rows × 22 columns

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

results=results.round(decimals=2)

results['cv_error_score']=100-results['mean_test_score']
```

PLOTTING THE HEATMAP WITH HYPERPARAMETERS FOR CV_ERROR SCORE

In [85]:
```
test_score_heatmap=results.pivot( 'param_max_depth' ,'param_n_estimator
s','cv_error_score'   )

test_score_heatmap
```

Out[85]:

param_n_estimators	21	30	35	40	45	50
param_max_depth						
11	5.41	5.30	5.25	5.18	5.13	5.08
15	5.30	5.18	5.14	5.07	5.02	4.97
20	5.21	5.11	5.02	4.91	4.88	4.83
24	5.18	5.06	4.99	4.90	4.85	4.81
27	5.15	5.02	4.94	4.87	4.80	4.76
30	5.18	5.02	4.97	4.89	4.88	4.86

In [86]:
```
import seaborn as sns
sns.heatmap(test_score_heatmap,annot=True,annot_kws={"size": 15}, fmt=
'g',linewidths=.3)
```

Out[86]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7d02f90a20>

In [87]:
```
print(model.best_estimator_)#printing the best_estimator
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=
0,
       max_depth=27, min_child_weight=1, missing=None, n_estimators=50,
       n_jobs=-1, nthread=None, objective='binary:logistic',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1)
```

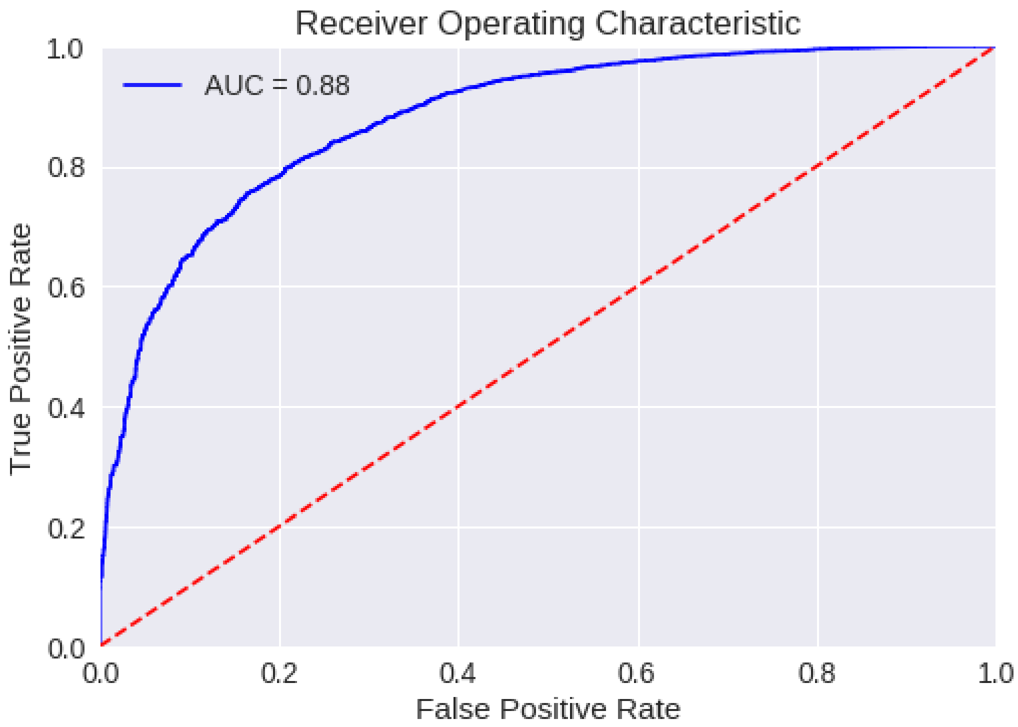
FROM THE ABOVE HEATMAPS RESULTS FOR CV DATA,WE FOUND THAT BEST HYPERPARAMETERS AS MAX_DEPTH=27 AND N_ESTIMATORS=50

PLOTTING THE ROC CURVE FOR GETTING AUC SCORE

In [131]:

```
xg=XGBClassifier(n_jobs=-1,max_depth=27 ,n_estimators=50)
xg.fit(train_bow,y_train)#fitting the model
probs = xg.predict_proba(test_bow)
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 [132]:
```
print('accuracy from the ROC curve is found as ',roc_auc*100)
```

```
accuracy from the ROC curve is found as  87.99880639194328
```

In [125]:
```
z=rf.feature_importances_
a=z.argsort()
print('shape of wieght vector is:',a.shape)
top_features=np.take(vectorizer.get_feature_names(),a[16360:])#taking l
ast features as they are of very high importance
```

```
shape of wieght vector is: (16382,)
```

In [126]:
```
print(top_features)#printing the top_features
top=list(top_features)
```

```
['product' 'unfortun' 'good' 'wors' 'tast' 'delici' 'bad' 'mayb' 'mone
```

```
y'
 'would' 'receiv' 'refund' 'horribl' 'return' 'worst' 'best' 'threw'
 'terribl' 'love' 'aw' 'great' 'disappoint']
```

REPRESENTING TOP IMPORTANT FEATURES USING WORDCLOUD LIBRARY

In [127]:

```
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))

# 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 [133]:

```
y_pred = xg.predict(test_bow)
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',cmap="YlG
nBu")
```

```
Accuracy on test set: 90.103%
Precision on test set: 0.907
Recall on test set: 0.988
F1-Score on test set: 0.946
Confusion Matrix of test set:
 [ [TN  FP]
 [FN TP] ]
```

Out[133]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7d023f4908>