```
from google.colab import drive
```

```
drive.mount("/content/drive/")
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mo
```

```
pip install pyforest
```

```
Requirement already satisfied: pyforest in /usr/local/lib/python3.6/dist-packages (1.0.
```

```
import pyforest
```

```
lazy_imports()
```

```
['import os',
 'from sklearn.ensemble import RandomForestClassifier',
 'import nltk',
 'from pathlib import Path',
 'from sklearn.ensemble import RandomForestRegressor',
 'import pandas as pd',
 'from openpyxl import load_workbook',
 'import matplotlib as mpl',
 'from sklearn.preprocessing import OneHotEncoder',
 'import keras',
 'import tqdm',
 'import pydot',
 'import spacy',
 'import sys',
 'from sklearn.feature_extraction.text import TfidfVectorizer',
 'from sklearn.ensemble import GradientBoostingClassifier',
 'import statistics',
 'import tensorflow as tf',
 'import plotly.express as px',
 'import plotly as py',
 'import pickle',
 'import re',
 'import glob',
 'from pyspark import SparkContext',
 'import numpy as np',
 'import dash',
 'import sklearn',
 'from sklearn import svm',
 'from sklearn.ensemble import GradientBoostingRegressor',
 'from dask import dataframe as dd',
 'from sklearn.manifold import TSNE',
 'import datetime as dt',
 'import seaborn as sns',
 'import gensim',
 'import matplotlib.pyplot as plt',
 'import bokeh',
 'import altair as alt',
 'from sklearn.model_selection import train_test_split',
 'import plotly.graph_objs as go']
```

```
import sqlite3
```

```
con=sqlite3.connect("/content/drive/My Drive/colab folder/Datasets/amazon-fine-food-reviews,
```

```
database=pd.read_sql_query("select * from reviews where score<>3",con)
```

```
database.Score.value_counts()
```

```
5    363122
4     80655
1     52268
2     29769
Name: Score, dtype: int64
```

```
def partition(x):
  if x>3:
    return 1
  return 0
```

```
sorted_database=database.Score.map(partition)
```

```
database.Score=sorted_database
```

```
database.Score.value_counts()
```

```
1    443777
0     82037
Name: Score, dtype: int64
```

```
sort=database.sort_values("ProductId")
```

```
sort
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpful
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	
...	...	...	...	...	...	
176791	191721	B009UOFTUI	AJVB004EB0MVK	D. Christofferson	0	
1362	1478	B009UOFU20	AJVB004EB0MVK	D. Christofferson	0	
303285	328482	B009UUS05I	ARL20DSHGVM1Y	Jamie	0	

```
final=sort.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inp
final.shape
```

```
(364173, 10)
```

```
final.shape
```

```
(364173, 10)
```

```
sort.shape
```

```
(525814, 10)
```

```
final.Score.value_counts()
```

```
1    307063
0     57110
Name: Score, dtype: int64
```

PREPROCESSING OR TEXT SUMMARIZATION

```
from bs4 import BeautifulSoup
```

```
import re
```

```
def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

```
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'y
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they'
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'l
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'u
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'd
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'v
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'do
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn'
            'won', "won't", 'wouldn', "wouldn't"])
```

```
%%time
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentance.strip())
```

```
100%|██████████| 364173/364173 [02:13<00:00, 2731.14it/s]CPU times: user 2min 8s, sys:
Wall time: 2min 13s
```

```
 string = " geeks for geeks "

# prints the string without stripping
print(string)

# prints the string by removing leading and trailing whitespaces
print(string.strip())
```

```
  geeks for geeks
 geeks for geeks
```

```
# prints the string by removing geeks
print(string.strip(' geeks'))
```

```
for
```

```
from tqdm import tqdm
preprocessed_summary = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Summary'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_summary.append(sentance.strip())
```

```
  6%|▋         | 22339/364173 [00:05<01:17, 4436.65it/s]/usr/local/lib/python3.6/dist-p
  ' Beautiful Soup.' % markup)
100%|██████████| 364173/364173 [01:22<00:00, 4439.45it/s]
```

```
final["Cleaned_text"]=preprocessed_reviews
final["Cleaned_summary"]=preprocessed_summary
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
```

```
final
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu
138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0	
138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1	
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1	
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1	
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3	
...	...	...	...	...	...	
178145	193174	B009RSR8HO	A4P6AN2L435PV	romarc	0	
173675	188389	B009SF0TN6	A1L0GWGRK4BYPT	Bety Robinson	0	
204727	221795	B009SR4OQ2	A32A6X5KCP7ARG	sicamar	1	
5259	5703	B009WSNWC4	AMP7K1O84DH1T	ESTY	0	
302474	327601	B009WVB40S	A3ME78KVX31T21	K'la	0	

364173 rows × 12 columns

```
final["Combined_Text"] = final["Cleaned_text"].values + ' ' + final['Cleaned_summary'].valu
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  """Entry point for launching an IPython kernel.
```

```
final
```

138688	150506	0006641040	A2IW4PEEKO2R0U	Tracy	1
138689	150507	0006641040	A1S4A3IQ2MU7V4	sally sue "sally sue"	1
138690	150508	0006641040	AZGXZ2UUK6X	Catherine Hallberg " (Kate)"	1
138691	150509	0006641040	A3CMRKGE0P909G	Teresa	3
...	...	...	...	...	...
178145	193174	B009RSR8HO	A4P6AN2L435PV	romarc	0
173675	188389	B009SF0TN6	A1L0GWGRK4BYPT	Bety Robinson	0
204727	221795	B009SR4OQ2	A32A6X5KCP7ARG	sicamar	1
5259	5703	B009WSNWC4	AMP7K1O84DH1T	ESTY	0
302474	327601	B009WVB40S	A3ME78KVX31T21	K'la	0

364173 rows × 13 columns

```
final.Combined_Text[138706]
```

```
'witty little book makes son laugh loud recite car driving along always sing refrain le
```

```
final.Cleaned_summary[138706]
```

```
'every book educational'
```

```
final.Cleaned_text[138706]
```

```
'witty little book makes son laugh loud recite car driving along always sing refrain le
```

```
time_sorted_data=final.sort_values("Time")
```

```
final_data=time_sorted_data.take(np.random.permutation(len(final)))[:100000]
print(final_data.shape)
```

```
(100000, 13)
```

```
final_data
```

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfu
310032	335719	B001VNGMMK	A3BZ51CB5PJ3VL	Jeffrey Smith	1	
33460	36400	B004CLCEDE	A3S7PRF6YODY9N	luvallmykids	1	
278521	301805	B0016BU7GO	A3KXIEF51W42PW	Cindy Luymes	3	
460187	497661	B000HDK0D2	ABHP4BWJBX9NT	Tim Ly	5	
515514	557343	B0014C2JFC	A14S1X4IG0IJB	K. Strain "sMilesToGo"	1	
...	...	...	...	...	...	
377779	408499	B004CQVCCS	A20BODDLOJMVQB	George T. Chambers III "George Chambers"	0	
171898	186486	B001L4JH5I	AUV1H02P9GAAA	Starr Messer "Starr"	0	
356707	385834	B000TV4W2C	A347PIFND2R6QW	K. Henderson	1	
477111	515946	B000JSQKS4	A27EMCN9BTYJHF	RoseMarie Cowham	0	
126220	136949	B002AQP5FW	A2843500EKO5YB	Megan Gorg	49	

100000 rows × 13 columns

```
time_sorted_data
```

138706	150524	0006641040	ACITT7DI6IDDL	shari zychinski	0
138683	150501	0006641040	AJ46FKXOVC7NR	Nicholas A Mesiano	2
417839	451856	B00004CXX9	AIUWLEQ1ADEG5	Elizabeth Medina	0
346055	374359	B00004CI84	A344SMIA5JECGM	Vincent P. Ross	1
417838	451855	B00004CXX9	AJH6LUC1UT1ON	The Phantom of the Opera	0
...	...	...	...	...	...
511105	552637	B0012EYELE	AWQOTHBNJBSVB	Gregory E. Grant "GG"	0
282454	306008	B0058CGLH6	A3QRR5YN6ALFPG	james a riche	0
311138	336872	B0012KB4U2	AGQBI6601XH2R	DaniC	0
524273	566798	B001PQTYN2	A3OTHWG8LLCLMU	PACKERS FAN "Gordon Boone"	1
355171	384161	B000EVWQZW	A2PCNXBSKCABG5	Whit	0

364173 rows × 13 columns

```
X_features = final_data['Combined_Text'].values
```

```
y_target = final_data['Score']

x_train, x_test, y_train, y_test = train_test_split(X_features, y_target, test_size=0.25, ra
```

```
x_train.shape
```

```
(75000,)
```

```
x_test.shape
```

```
(25000,)
```

Featurization

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
count_vec=CountVectorizer(min_df=10,max_features=50000,dtype="float")
```

```
count_vec.fit(x_train)
```

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype='float', encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=50000, min_df=10,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

```
bow_x_train=count_vec.transform(x_train)
bow_x_test=count_vec.transform(x_test)
```

```
print(bow_x_train.shape)
print(bow_x_test.shape)
```

```
(75000, 11191)
(25000, 11191)
```

```
bow_feature=count_vec.get_feature_names()
```

```
"""bow_x_train=bow_x_train.toarray()
bow_x_test=bow_x_test.toarray()"""
```

```
'bow_x_train=bow_x_train.toarray()\nbow_x_test=bow_x_test.toarray()'
```

```
bow_vocab=count_vec.vocabulary_
```

```
bow_vocab["not"]
#no of times words has been repeated in review corpus
```

```
6601
```

```
from sklearn.model_selection import TimeSeriesSplit
import xgboost
from xgboost import XGBClassifier
```

```
from sklearn.metrics import roc_auc_score,roc_curve,accuracy_score,confusion_matrix,classif
```

```
from sklearn.model_selection import RandomizedSearchCV
```

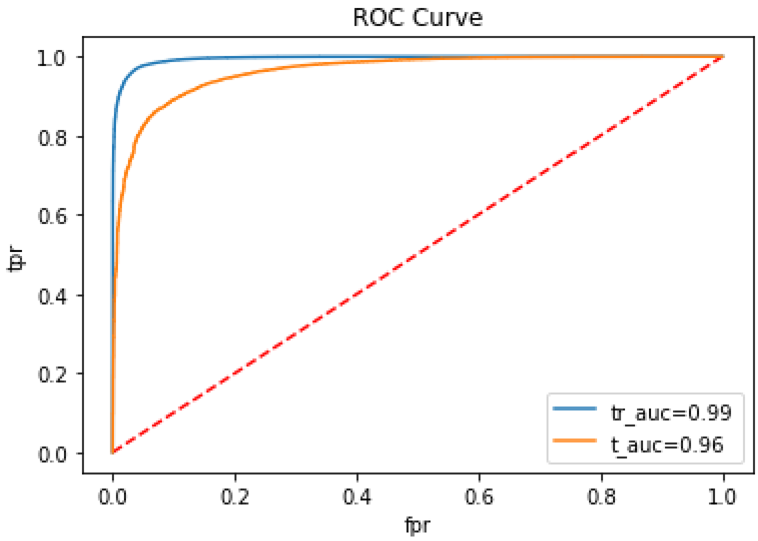
```
%%time
parameter={"n_estimators":[100,200,300,400,500],"max_depth":[3,4,5,6,7,8,9,10]}
cv_timeseries=TimeSeriesSplit(n_splits=5).split(bow_x_train)
xgb=XGBClassifier(n_jobs=-1)
xgb_bow=RandomizedSearchCV(xgb,param_distributions=parameter,scoring="roc_auc",cv=cv_timese
xgb_bow.fit(bow_x_train,y_train)

print("Best parameter obtained from RandomSearch CV: \n", xgb_bow.best_params_)
print("Best Score : ", xgb_bow.best_score_)
```

```
Best parameter obtained from RandomSearch CV:
 {'n_estimators': 500, 'max_depth': 9}
Best Score :  0.9558639370509707
CPU times: user 1h 13min 27s, sys: 4.57 s, total: 1h 13min 32s
Wall time: 19min 12s
```

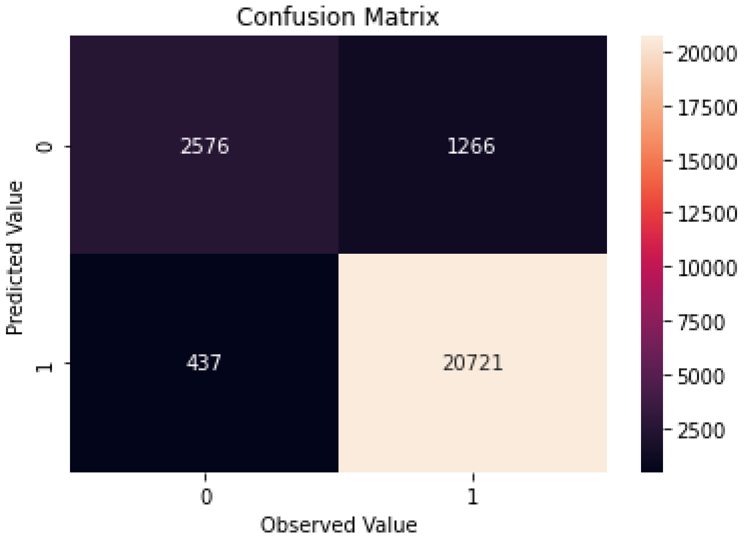
```
def plot_model(title,clf,x_train,x_test,y_train,y_test):
  print(title)
  print("="*50)

  y_train_pred=clf.predict_proba(x_train)
  tr_fpr,tr_tpr,tr_thre=roc_curve(y_train,y_train_pred[:,1])
  tr_auc=auc(tr_fpr,tr_tpr)

  y_test_pred=clf.predict_proba(x_test)
  t_fpr,t_tpr,t_thres=roc_curve(y_test,y_test_pred[:,1])
  t_auc=auc(t_fpr,t_tpr)

  plt.plot([0,1],[0,1],"r--")
  plt.plot(tr_fpr,tr_tpr,label="tr_auc=%0.2f"%tr_auc)
  plt.plot(t_fpr,t_tpr,label="t_auc=%0.2f"%t_auc)
  plt.xlabel("fpr")
  plt.ylabel("tpr")
  plt.legend(loc="lower right")
  plt.title("ROC Curve")
  plt.show()

  y_pred=clf.predict(x_test)
  accuracy=accuracy_score(y_test,y_pred)
  confuse_mat=confusion_matrix(y_test,y_pred)
  cls_report=classification_report(y_test,y_pred)
  print("The Classification Report:\n",cls_report)

  sns.heatmap(confuse_mat,annot=True,fmt="g")
  plt.xlabel("Observed Value")
  plt.ylabel("Predicted Value")
  plt.title("Confusion Matrix")
  plt.show()
```

```
plot_model("XGBOOST WITH BAG OF WORDS",xgb_bow,bow_x_train,bow_x_test,y_train,y_test)
```

```
XGBOOST WITH BAG OF WORDS
==================================================
```



```
The Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.67      0.75      3842
           1       0.94      0.98      0.96     21158

    accuracy                           0.93     25000
   macro avg       0.90      0.82      0.86     25000
weighted avg       0.93      0.93      0.93     25000
```



```
!pip install chart_studio
```

```
    Collecting chart_studio
      Downloading https://files.pythonhosted.org/packages/ca/ce/330794a6b6ca4b9182c38fc69dc
         |████████████████████████████████| 71kB 2.1MB/s
```

```
import chart_studio.plotly as py
import plotly.graph_objs as go
```

```
    Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from char
```

```
import plotly.offline as offline
```

```
    Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-p
```

```
def configure_plotly_browser_state():
  import IPython
  display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></script>
        <script>
          requirejs.config({
            paths: {
              base: '/static/base',
              plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
            },
          });
        </script>
        '''))


configure_plotly_browser_state()
n_estimators=[100,200,300,400,500];max_depth=[3,4,5,6,7,8,9,10];

train_auc=xgb_bow.cv_results_["mean_train_score"]
cv_auc=xgb_bow.cv_results_["mean_test_score"]

x1=n_estimators
y1=max_depth
z1=train_auc

x2=n_estimators
y2=max_depth
z2=cv_auc

offline.init_notebook_mode(connected=False)

trace1=go.Scatter3d(x=x1,y=y1,z=z1,name="train_auc")
trace2=go.Scatter3d(x=x2,y=y2,z=z2,name="cv_auc")

data=[trace1,trace2]

layout=go.Layout(scene=dict(xaxis = dict(title='n_estimators'),yaxis = dict(title='max_dept

fig = go.Figure(data=data, layout=layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

```
pip install wordcloud
```

```
Requirement already satisfied: wordcloud in /usr/local/lib/python3.6/dist-packages (1.5
Requirement already satisfied: pillow in /usr/local/lib/python3.6/dist-packages (from w
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.6/dist-packages (
```

```
from wordcloud import WordCloud
from IPython.core.display import HTML
```

```
bow_i_features=xgb_bow.best_estimator_.feature_importances_
```

```
ERROR! Session/line number was not unique in database. History logging moved to new ses
```

```
bow_i_features
```

```
array([0., 0., 0., ..., 0., 0., 0.], dtype=float32)
```

```
-bow_i_features
```

```
array([-0., -0., -0., ..., -0., -0., -0.], dtype=float32)
```

```
top_features_index=(-bow_i_features).argsort()
```

```
(bow_i_features).argsort()
```

```
array([    0,  7198,  7199, ..., 11060,  5795,  2606])
```

```
top_features_index
```

```
array([ 2606,  5795, 11060, ...,  3995,  3974, 11190])
```

```
top_feature_bow=np.take(bow_feature,top_features_index[:20])
print(top_feature_bow)
```

```
['delicious' 'loves' 'worst' 'yummy' 'easy' 'horrible' 'perfect'
 'wonderful' 'excellent' 'awful' 'highly' 'great' 'yuck' 'terrible'
 'awesome' 'waste' 'beware' 'favorite' 'best' 'threw']
```

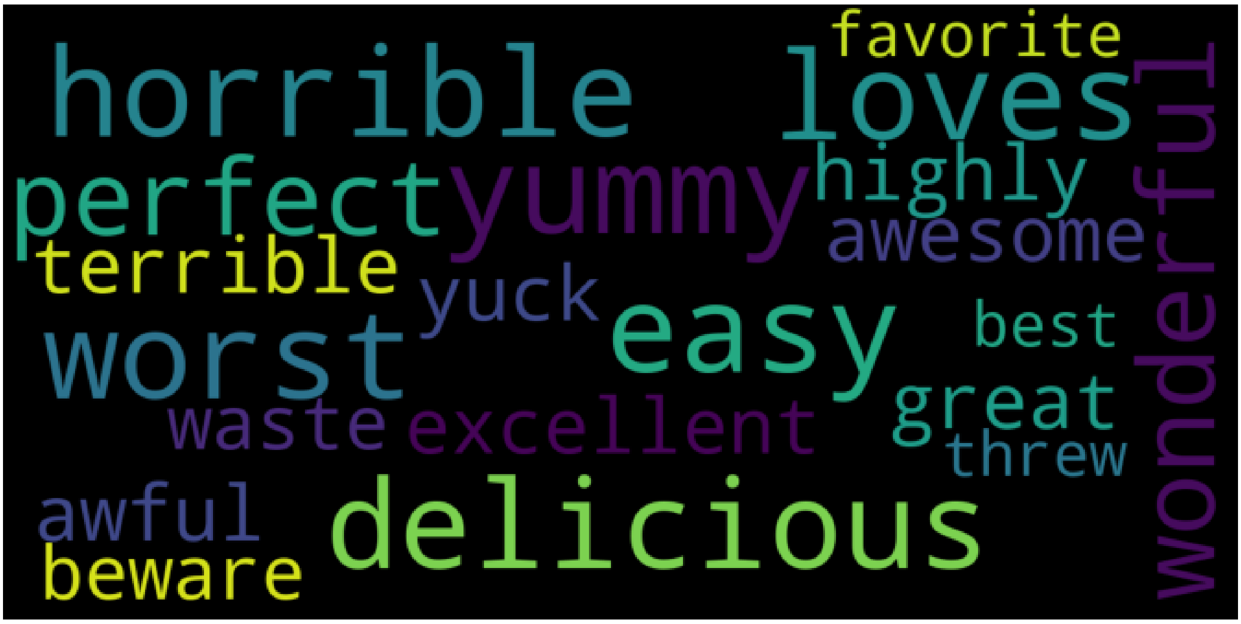
```
text=" "
for i in top_feature_bow:
  text=text+" "+i
```

```
text
```

```
'  delicious loves worst yummy easy horrible perfect wonderful excellent awful highly g
```

```
def show_wordcloud(data, title = None):
    wordcloud = WordCloud(
        background_color='black',
        stopwords=stopwords,
        max_words=200,
        max_font_size=40,
        scale=3,
        random_state=1 # chosen at random by flipping a coin; it was heads
    ).generate(str(data))

    fig = plt.figure(1, figsize=(12, 12))
    plt.axis('off')
    if title:
        fig.suptitle(title, fontsize=20)
        fig.subplots_adjust(top=2.3)

    plt.imshow(wordcloud)
    plt.show()
```

```
wc_bow=show_wordcloud(text,title="bow_top_features")
```



bow_top_features

```
tf_vec=TfidfVectorizer(min_df=10,max_features=50000,dtype="float")
```

```
tf_vec.fit(x_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/feature_extraction/text.py:1817: UserWar

Only (<class 'numpy.float64'>, <class 'numpy.float32'>, <class 'numpy.float16'>) 'dtype

TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype='float', encoding='utf-8', input='content',
                lowercase=True, max_df=1.0, max_features=50000, min_df=10,
                ngram_range=(1, 1), norm='l2', preprocessor=None,
                smooth_idf=True, stop_words=None, strip_accents=None,
                sublinear_tf=False, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, use_idf=True, vocabulary=None)
```

```
tf_feature=tf_vec.get_feature_names()
```

```
tf_x_train=tf_vec.transform(x_train)
tf_x_test=tf_vec.transform(x_test)
print(tf_x_train.shape)
print(tf_x_test.shape)
```

```
(75000, 11191)
(25000, 11191)
```

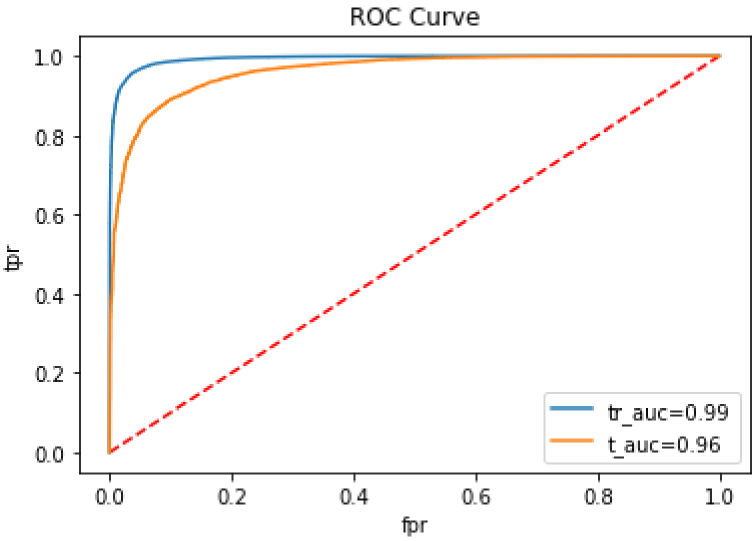
```
%%time
parameter={"n_estimators":[100,200,300,400,500],"max_depth":[3,4,5,6,7,8,9,10]}
cv_timeseries=TimeSeriesSplit(n_splits=5).split(tf_x_train)
xgb=XGBClassifier(n_jobs=-1)
```

```
xgb_tfidf=RandomizedSearchCV(xgb,param_distributions=parameter,scoring="roc_auc",cv=cv_time
xgb_tfidf.fit(tf_x_train,y_train)

print("Best parameter obtained from RandomSearch CV: \n", xgb_tfidf.best_params_)
print("Best Score : ", xgb_tfidf.best_score_)
```

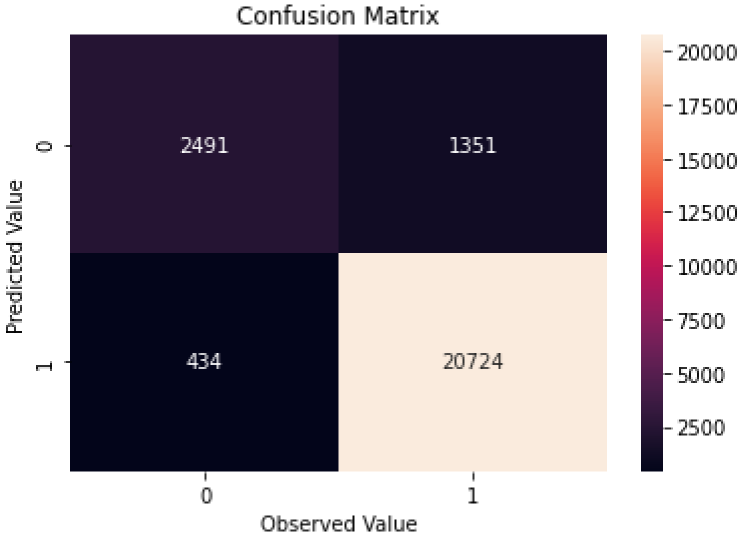
```
Best parameter obtained from RandomSearch CV:
 {'n_estimators': 500, 'max_depth': 7}
Best Score :  0.9557485656814665
CPU times: user 2h 37min 33s, sys: 7.88 s, total: 2h 37min 41s
Wall time: 40min 17s
```

```
plot_model("XGBOOST with TFIDF",xgb_tfidf,tf_x_train,tf_x_test,y_train,y_test)
```

```
XGBOOST with TFIDF
==================================================
```



```
The Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.65      0.74      3842
           1       0.94      0.98      0.96     21158

    accuracy                           0.93     25000
   macro avg       0.90      0.81      0.85     25000
weighted avg       0.93      0.93      0.92     25000
```



```
def plot_3d(n_estimators,max_depth,tr_auc,te_auc):
  configure_plotly_browser_state()
  x1=n_estimators
  y1=max_depth
  z1=tr_auc

  x2=n_estimators
  y2=max_depth
  z2=te_auc

  offline.init_notebook_mode(connected=False)

  trace1=go.Scatter3d(x=x1,y=y1,z=z1,name="train_auc")
  trace2=go.Scatter3d(x=x2,y=y2,z=z2,name="cv_auc")
  data=[trace1,trace2]

  layout=go.Layout(scene=dict(xaxis = dict(title='n_estimators'),yaxis = dict(title='max_dep
  #layout=go.layout(scene=dict(xaxis=dict(title="n_estimators"),yaxis=dict(title="max_depth
  fig = go.Figure(data=data, layout=layout)
  #fig=go.Figure(data=data,layout=layout)
  offline.iplot(fig, filename='3d-scatter-colorscale')
  #offline.iplot(fig,filename="3d-scatter-colorspace")
```

```
feature=xgb_tfidf.best_estimator_.feature_importances_
```

```
top_features_index_tf=(-feature).argsort()
```

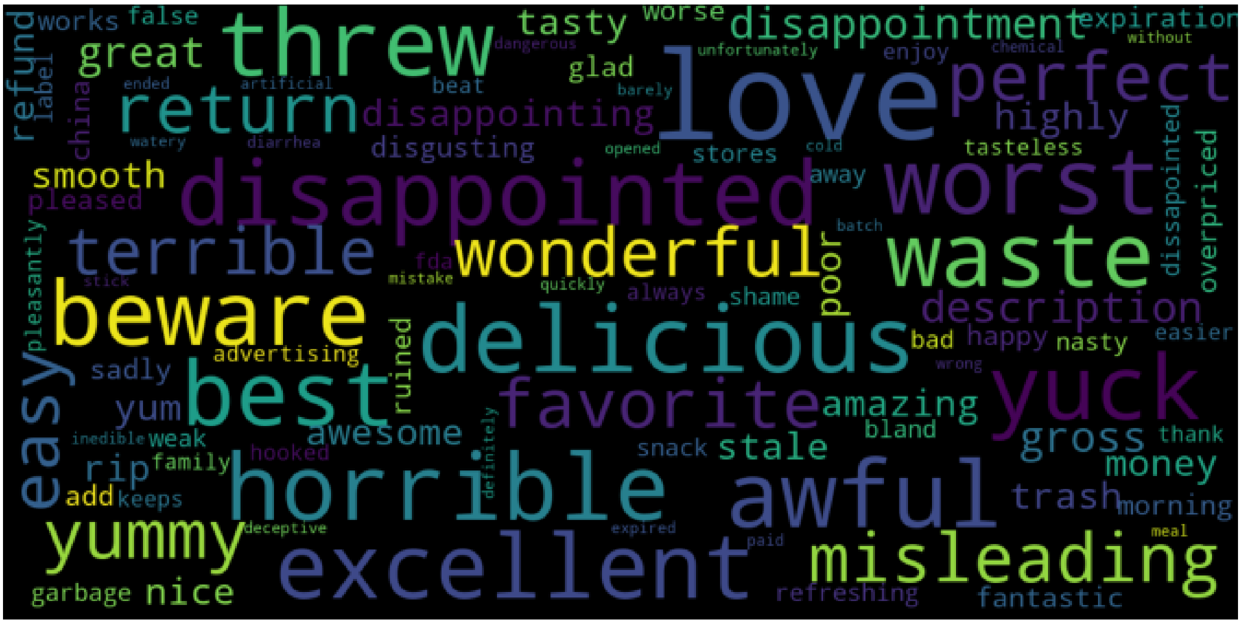
```
top_features_index_tf
```

```
array([ 2606, 11060,  5795, ...,  3928,  3930, 11190])
```

```
top_feature_tfidf=np.take(tf_feature,top_features_index_tf[:100])
print(top_feature_tfidf)
```

```
['delicious' 'worst' 'loves' 'horrible' 'threw' 'awful' 'disappointed'
 'yuck' 'beware' 'excellent' 'best' 'waste' 'misleading' 'favorite'
 'perfect' 'wonderful' 'return' 'easy' 'yummy' 'terrible' 'disappointment'
 'description' 'great' 'gross' 'disappointing' 'money' 'highly' 'tasty'
 'yum' 'rip' 'love' 'refund' 'nice' 'trash' 'poor' 'smooth' 'awesome'
 'stale' 'amazing' 'disgusting' 'glad' 'pleased' 'china' 'worse' 'happy'
 'morning' 'expiration' 'overpriced' 'fantastic' 'works' 'sadly' 'add'
 'ruined' 'refreshing' 'stores' 'weak' 'garbage' 'shame' 'bland' 'snack'
 'label' 'bad' 'away' 'false' 'nasty' 'dissapointed' 'advertising'
 'hooked' 'enjoy' 'thank' 'keeps' 'pleasantly' 'fda' 'tasteless' 'beat'
 'always' 'easier' 'family' 'unfortunately' 'diarrhea' 'quickly' 'expired'
 'ended' 'deceptive' 'wrong' 'artificial' 'mistake' 'watery' 'meal' 'paid'
 'batch' 'barely' 'without' 'dangerous' 'chemical' 'inedible' 'opened'
 'cold' 'definitely' 'stick']
```

```
text_tf=" "
for i in top_feature_tfidf:
  text_tf=text_tf+" "+i
wc_tf=show_wordcloud(text_tf,title="tfidf_top_features")
```



tfidf_top_features

```
tr_auc=xgb_tfidf.cv_results_["mean_train_score"]
ts_auc=xgb_tfidf.cv_results_["mean_test_score"]
n_estimators=[100,200,300,400,500]
max_depth=[3,4,5,6,7,8,9,10]
plot_3d(n_estimators,max_depth,tr_auc,ts_auc)


from gensim.models import Word2Vec


def avg_words(x_train):
  i=0
  list_of_sentance=[]
  for sentance in x_train:
    list_of_sentance.append(sentance.split())

  w2v_models=Word2Vec(list_of_sentance,size=50,min_count=10,workers=8)
  w2v_words=list(w2v_models.wv.vocab)

  sent_vectors = []

  for sent in list_of_sentance:
      sent_vec = np.zeros(50)
      cnt_words =0;
      for word in sent: #
          if word in w2v_words:
              vec = w2v_models.wv[word]
              sent_vec += vec
              cnt_words += 1
      if cnt_words != 0:
          sent_vec /= cnt_words
      sent_vectors.append(sent_vec)
  return sent_vectors
```

```
j=["my name is sushil","my last name is chauhan"]
i=0
list_of_sentance=[]
for sentance in j:
  list_of_sentance.append(sentance.split())
list_of_sentance
```

```
[['my', 'name', 'is', 'sushil'], ['my', 'last', 'name', 'is', 'chauhan']]
```

```
%%time
avg_x_train=avg_words(x_train)
avg_x_test=avg_words(x_test)
```

```
Buffered data was truncated after reaching the output size limit.
```

```
len(avg_x_train)
```

```
75000
```

```
a_x_train=np.array(avg_x_train)
a_x_test=np.array(avg_x_test)
```

```
a_x_train.shape
```

```
(75000, 50)
```

```
a_x_test.shape
```

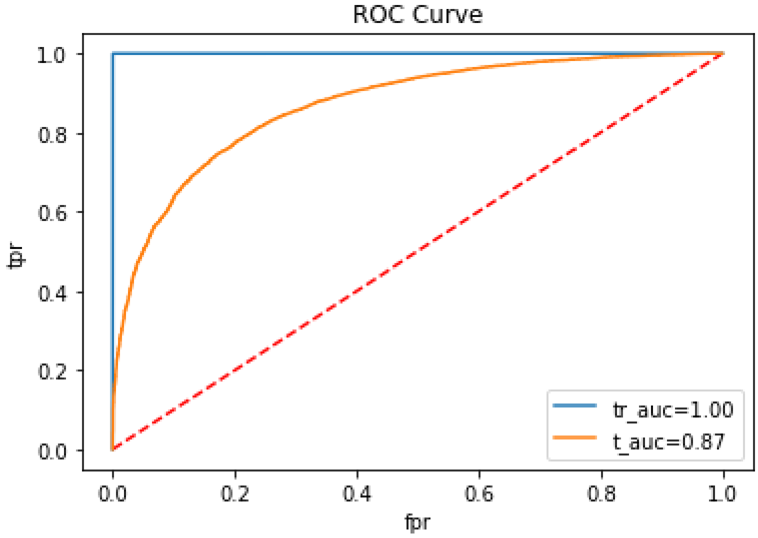
```
(25000, 50)
```

```
%%time
parameter={"n_estimators":[100,200,300,400,500],"max_depth":[3,4,5,6,7,8,9,10]}
cv_timeseries=TimeSeriesSplit(n_splits=5).split(a_x_train)
xgb=XGBClassifier(n_jobs=-1)
xgb_avg=RandomizedSearchCV(xgb,param_distributions=parameter,scoring="roc_auc",cv=cv_timese
xgb_avg.fit(a_x_train,y_train)

print("Best parameter obtained from RandomSearch CV: \n", xgb_avg.best_params_)
print("Best Score : ", xgb_avg.best_score_)
```

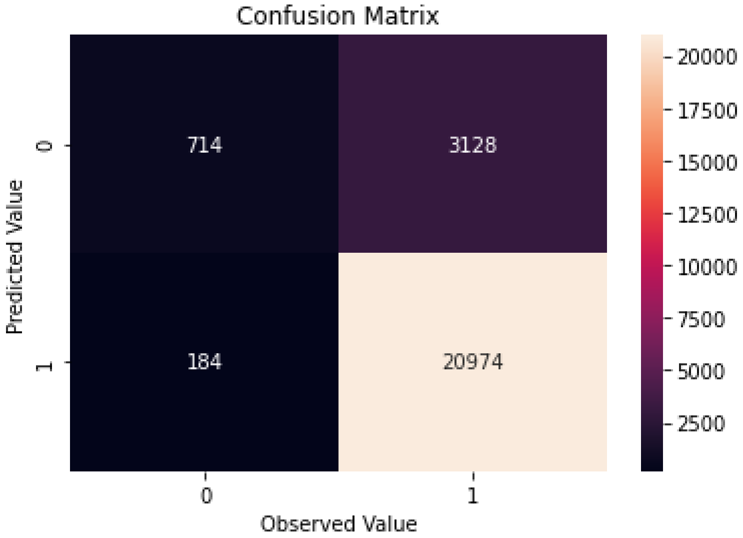
```
Best parameter obtained from RandomSearch CV:
 {'n_estimators': 400, 'max_depth': 8}
Best Score :  0.9414421779936418
CPU times: user 2h 14min 31s, sys: 6.06 s, total: 2h 14min 38s
Wall time: 34min 10s
```

```
plot_model("XGBOOST WITH AVG_W2V",xgb_avg,a_x_train,a_x_test,y_train,y_test)
```

```
XGBOOST WITH AVG_W2V
==================================================
```



```
The Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.19      0.30      3842
           1       0.87      0.99      0.93     21158

    accuracy                           0.87     25000
   macro avg       0.83      0.59      0.61     25000
weighted avg       0.86      0.87      0.83     25000
```



```
tr_auc=xgb_avg.cv_results_["mean_train_score"]
te_auc=xgb_avg.cv_results_["mean_test_score"]
plot_3d(n_estimators,max_depth,tr_auc,te_auc)
```

```
def tfidf(x_train):
    # S = ["abc def pqr", "def def def abc", "pqr pqr def"]
  model = TfidfVectorizer()
  list_of_sentance=[]
  for sentance in x_train:
    list_of_sentance.append(sentance.split())

  w2v_models=Word2Vec(list_of_sentance,size=50,min_count=10,workers=8)
  w2v_words=list(w2v_models.wv.vocab)

  tf_idf_matrix = model.fit_transform(x_train)
  # 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_)))
  # TF-IDF weighted Word2Vec
  tfidf_feat = model.get_feature_names() # tfidf words/col-names
  # final_tf_idf is the sparse matrix with row= sentence, col=word and cell_val = tfidf

  tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in this list
  row=0;
  for sent in list_of_sentance: # for each review/sentence
      sent_vec = np.zeros(50) # 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 w2v_words and word in tfidf_feat:
              vec = w2v_models.wv[word]
              #tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
              # to reduce the computation we are
              # dictionary[word] = idf value of word in whole courpus
              # sent.count(word) = tf valeus of word in this review
              tf_idf = dictionary[word]*(sent.count(word)/len(sent))
              sent_vec += (vec * tf_idf)
              weight_sum += tf_idf
      if weight_sum != 0:
          sent_vec /= weight_sum
      tfidf_sent_vectors.append(sent_vec)
      row += 1
  return tfidf_sent_vectors
```

```
%%time
wtf_x_train=tfidf(x_train)
wtf_x_test=tfidf(x_test)
```

```
Buffered data was truncated after reaching the output size limit.
```

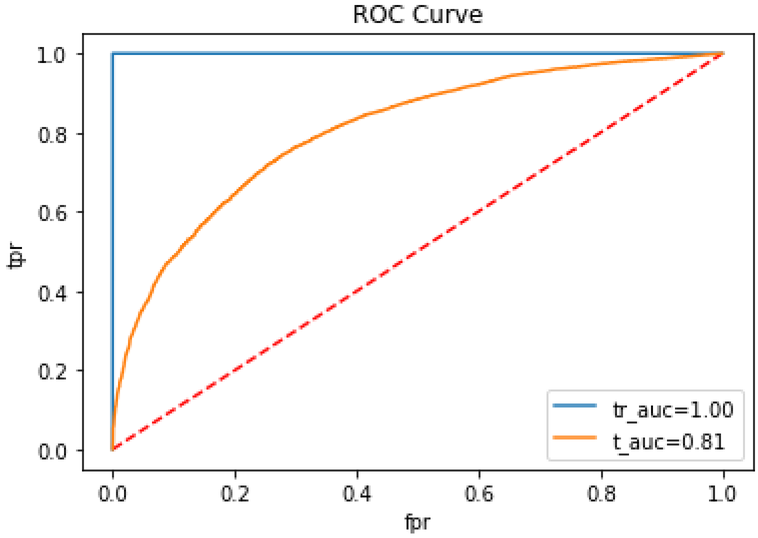
```
w_x_train=np.array(wtf_x_train)
w_x_test=np.array(wtf_x_test)
```

```
%%time
parameter={"n_estimators":[100,200,300,400,500],"max_depth":[3,4,5,6,7,8,9,10]}
cv_timeseries=TimeSeriesSplit(n_splits=5).split(w_x_train)
xgb=XGBClassifier(n_jobs=-1)
xgb_w=RandomizedSearchCV(xgb,param_distributions=parameter,scoring="roc_auc",cv=cv_timeseri
xgb_w.fit(w_x_train,y_train)

print("Best parameter obtained from RandomSearch CV: \n", xgb_w.best_params_)
print("Best Score : ", xgb_w.best_score_)
```

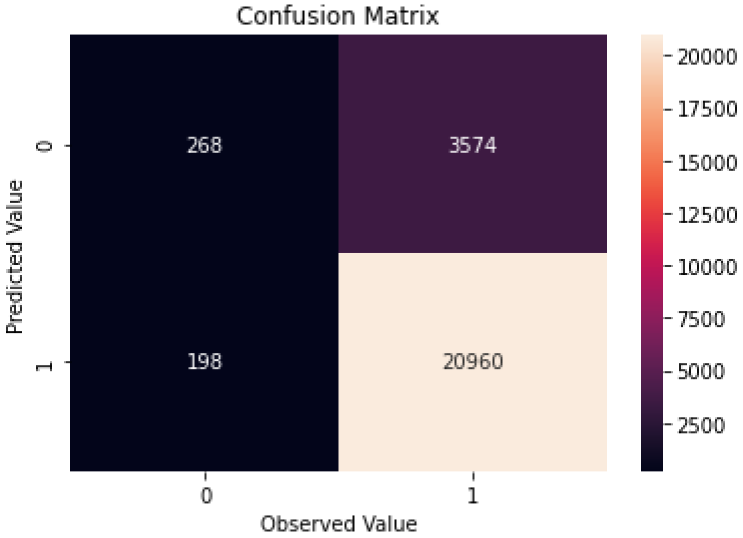
```
Best parameter obtained from RandomSearch CV: 
 {'n_estimators': 400, 'max_depth': 10}
Best Score :  0.9225804036534406
CPU times: user 2h 47min 25s, sys: 7.93 s, total: 2h 47min 33s
Wall time: 42min 20s
```

```
plot_model("XGBOOST with WEIGHTED TFIDF",xgb_w,w_x_train,w_x_test,y_train,y_test)
```

```
XGBOOST with WEIGHTED TFIDF
==================================================
```



```
The Classification Report:
               precision    recall  f1-score   support

           0       0.58      0.07      0.12      3842
           1       0.85      0.99      0.92     21158

    accuracy                           0.85     25000
   macro avg       0.71      0.53      0.52     25000
weighted avg       0.81      0.85      0.80     25000
```



```
tr_auc=xgb_w.cv_results_["mean_train_score"]
ts_auc=xgb_w.cv_results_["mean_test_score"]
plot_3d(n_estimators,max_depth,tr_auc,ts_auc)
```

```
from prettytable import PrettyTable


ptable=PrettyTable()


ptable.add_column("XGBOOST",["BAG OF WORDS","TFIDF","AVERAGE-W2V","WEIGHTED-TFIDF"])
ptable.add_column("n_estimators",[500,500,400,400])
ptable.add_column("max_depth",[9,7,8,10])
ptable.add_column("train_auc",[0.99,0.99,1,1])
ptable.add_column("test_auc",[0.96,0.96,0.87,0.81])
ptable.add_column("Times in min",[19,40,34,42])
print(ptable)
```

```
+----------------+--------------+-----------+-----------+----------+--------------+
|    XGBOOST     | n_estimators | max_depth | train_auc | test_auc | Times in min |
+----------------+--------------+-----------+-----------+----------+--------------+
|  BAG OF WORDS  |     500      |     9     |    0.99   |   0.96   |      19      |
|     TFIDF      |     500      |     7     |    0.99   |   0.96   |      40      |
|  AVERAGE-W2V   |     400      |     8     |     1     |   0.87   |      34      |
| WEIGHTED-TFIDF |     400      |     10    |     1     |   0.81   |      42      |
+----------------+--------------+-----------+-----------+----------+--------------+
```

Conclusion:-