In [1]:
```
import pyforest
```

In [2]:
```
lazy_imports()
```

Out[2]:
```
['import glob',
 'import datetime as dt',
 'import os',
 'import pickle',
 'import spacy',
 'from sklearn.model_selection import train_test_split',
 'import sys',
 'import numpy as np',
 'import matplotlib.pyplot as plt',
 'from sklearn.ensemble import RandomForestClassifier',
 'import nltk',
 'from sklearn.preprocessing import OneHotEncoder',
 'import statistics',
 'from dask import dataframe as dd',
 'import plotly as py',
 'import bokeh',
 'import plotly.graph_objs as go',
 'import gensim',
 'from sklearn.manifold import TSNE',
 'from sklearn.ensemble import RandomForestRegressor',
 'import seaborn as sns',
 'import plotly.express as px',
 'from sklearn.ensemble import GradientBoostingClassifier',
 'import pandas as pd',
 'from openpyxl import load_workbook',
 'import re',
 'from sklearn.feature_extraction.text import TfidfVectorizer',
 'import altair as alt',
 'from pyspark import SparkContext',
 'import dash',
 'import matplotlib as mpl',
 'from pathlib import Path',
 'import sklearn',
 'import tensorflow as tf',
 'from sklearn.ensemble import GradientBoostingRegressor',
 'import keras',
 'import tqdm',
 'import pydot',
 'from sklearn import svm']
```

In [3]:
```
import sqlite3
```

In [4]:
```
con=sqlite3.connect("/home/sushil/Downloads/Datasets/amazon-fine-food-reviews/database
```

In [5]:
```
database=pd.read_sql_query("select * from reviews where Score<>3 limit 20000",con)
```

In [6]:
```
database.Score.value_counts()
```

Out[6]:
```
5    13745
4     3110
1     1953
2     1192
Name: Score, dtype: int64
```

In [7]:
```
def partition(x):
    if x>3:
        return 1
    return 0
```

In [8]:
```
database=database[database.HelpfulnessNumerator<=database.HelpfulnessDenominator]
```

In [9]:
```
database
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5
...	...	...	...	...	...	...	..
19995	21784	B000KV61FC	A3FVKI0UH9DO2A	S. Malosh	1	1	5
19996	21785	B000KV61FC	A3ACVJEAM4L2LQ	Elb	1	1	4
19997	21786	B000KV61FC	AHHWZ4723VGOL	Sherry Lynn	1	1	1
19998	21787	B000KV61FC	A2O4CZ102I8Q2K	jus42day	1	1	4
19999	21788	B000KV61FC	A3I4GCI6XTX1BB	Eric C. Vizinas "Q"	1	1	5

20000 rows × 10 columns

In [10]:
```
database.Score=database.Score.map(partition)
```

In [11]:
```
database.Score.value_counts()
```

Out[11]:
```
1    16855
0     3145
Name: Score, dtype: int64
```

Datasets is Severely imbalance

In [12]:
```
df=database.sort_values("ProductId")
```

In [13]:
```
df=df.drop_duplicates(["ProductId","UserId","Time","Text"])
```

In [15]:
```
df.shape
```

Out[15]:
```
(19963, 10)
```

In [16]:
```
type(df)
```

Out[16]:
```
pandas.core.frame.DataFrame
```

Process Text Summary and Text Data Cleaning

In [17]:
```
def preprocess_text(sentence):
    def remove_html(sentence):
        html_tag_re_obj = re.compile('<.*>?')
        return re.sub(html_tag_re_obj, ' ', sentence)

    def remove_punctuations(sentence):
        cleaned_sentence = re.sub(r'[^a-zA-Z]', r' ', sentence)
        return cleaned_sentence

    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

    from bs4 import BeautifulSoup
    import re
    #from nltk.corpus import stopwords
    import nltk
    from tqdm import tqdm
    from nltk.stem import SnowballStemmer
    from nltk.corpus import stopwords
    stopwords = stopwords.words('english')
    stemmer = SnowballStemmer('english')
    stopwords = set(stopwords)
    stopwords.remove('not')

    cleaned_corpus = []
    for doc in sentence:
        cleaned_doc_1 = remove_html(doc)
        cleaned_doc_2 = remove_punctuations(doc)
        cleaned_doc_2 = decontracted(cleaned_doc_2)
        cleaned_corpus.append(cleaned_doc_2)
    count = 0

    filtered_corpus = list(map(lambda doc: ' '.join(list(filter(lambda word: True if
                                                                , doc.split()))),clea
    process_text = list(map(lambda doc: ' '.join(list(map(stemmer.stem, doc.split()))

    return process_text
```

```
In [18]: %%time
         df.Text=preprocess_text(df.Text)
         df.Summary=preprocess_text(df.Summary)

         CPU times: user 18.6 s, sys: 252 ms, total: 18.9 s
         Wall time: 25.1 s
```

```
In [19]: df["Text"]

Out[19]: 2547     we use victor fli bait season can beat great p...
         2546     whi product avail br http www amazon com victo...
         1146     this realli good idea final product outstand i...
         1145     i receiv shipment could hard wait tri product ...
         8696     i use brand year if feel clog ate massiv meal ...
                                         ...
         11590    some product work need done disc i wast sever ...
         11588    i love coffe usual i would not recommend maxwe...
         11589    this cappuccino good tast almost good starbuck...
         1362     this coffe suppos premium tast wateri thin not...
         5259     purchas product local store ny kid love it qui...
         Name: Text, Length: 19963, dtype: object
```

```
In [20]: df.Summary

Out[20]: 2547                                      fli begon
         2546                                    thirti buck
         1146                                  great product
         1145                               wow make slicker
         8696                 the best cleans tea i ever had
                                     ...
         11590                              cappucino disc
         11588            coff shop qualiti capuccino home
         11589                              veri veri good
         1362     weak coffe not good premium product price
         5259                                         delici
         Name: Summary, Length: 19963, dtype: object
```

```
In [21]: %%time
         text_length=[]
         for i in df.Text:
             text_length.append(len(i.split()))

         CPU times: user 91.4 ms, sys: 3.88 ms, total: 95.3 ms
         Wall time: 94.1 ms
```

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

Out[22]: 19963
```

```
In [23]: text_length[0]

Out[23]: 10
```

```
In [25]: type(df.Text)

Out[25]: pandas.core.series.Series
```

```
In [26]: df.shape

Out[26]: (19963, 10)
```

```
In [27]: index=list(df.index)
         print(type(index))

         <class 'list'>
```

In [28]:

```
df_text_length=pd.Series(data=text_length,name="text_length",index=index)
```

In [29]:

```
df_text_length
```

Out[29]:

```
2547      10
2546      32
1146      19
1145      47
8696      40
        ... 
11590     25
11588    116
11589     79
1362      35
5259      22
Name: text_length, Length: 19963, dtype: int64
```

In [30]:
```
df_new=pd.concat([df,df_text_length],axis=1)
df_new
```

Out[30]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Sco
2547	2775	B00002NCJC	A13RRPGE79XFFH	reader48	0	0	
2546	2774	B00002NCJC	A196AJHU9EASJN	Alex Chaffee	0	0	
1146	1245	B00002Z754	A29Z5PI9BW2PU3	Robbie	7	7	
1145	1244	B00002Z754	A3B8RCEI0FXFI6	B G Chase	10	10	
8696	9527	B00005V3DC	A8KY7S48EW7LW	A. Daly "AD"	0	0	
...	...	...	...	...	...	...	
11590	12621	B009KP6HBM	A3UCO959VA9MV	Maryann Wardach	0	0	
11588	12619	B009KP6HBM	A20NB4UBW4WDKG	Gerardo "GD"	1	1	
11589	12620	B009KP6HBM	A3D9NUCR4RXDPY	Kathleen San Martino	1	1	
1362	1478	B009UOFU20	AJVB004EB0MVK	D. Christofferson	0	0	
5259	5703	B009WSNWC4	AMP7K1O84DH1T	ESTY	0	0	

19963 rows × 11 columns

In [31]:
```
a=df_new.text_length.isnull()
```

```
In [32]: a.value_counts()
```

```
Out[32]: False    19963
         Name: text_length, dtype: int64
```

```
In [33]: df_new=df_new.sort_values("Time")
```

```
In [34]: x_train,x_test,y_train,y_test=train_test_split(df_new.Text,df_new.Score,test_size=0.3
```

```
In [35]: print(x_train.shape)
         print(y_train.shape)
```

```
(13375,)
(13375,)
```

```
In [36]: from sklearn.feature_extraction.text import CountVectorizer
         from sklearn.feature_extraction.text import TfidfVectorizer
```

```
In [37]: from sklearn.model_selection import GridSearchCV
```

```
In [38]: count_vec=CountVectorizer(min_df=10)
```

```
In [39]: count_vec.fit(x_train)
```

```
Out[39]: CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                         dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
                         lowercase=True, max_df=1.0, max_features=None, 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)
```

```
In [40]: bow_x_train=count_vec.transform(x_train)
         bow_x_test=count_vec.transform(x_test)
```

```
In [43]: bow_features=count_vec.get_feature_names
         bow_vocab=count_vec.vocabulary
```

```
In [41]: print(bow_x_train.shape)
         print(bow_x_test.shape)
```

```
(13375, 3424)
(6588, 3424)
```

```
In [42]: bow_x_train=bow_x_train.toarray()
         bow_x_test=bow_x_test.toarray()
```

```
In [52]: rm=RandomForestClassifier(n_jobs=-1,class_weight="balanced")
```

```
In [53]: parameter={"n_estimators":[100,200,300,400,500]}
```

```
In [54]: clf_bow=GridSearchCV(estimator=rm,param_grid=parameter,cv=4,scoring="roc_auc",return_t
```

```
In [55]: from sklearn.metrics import roc_auc_score,roc_curve,classification_report,confusion_ma
In [56]: %%time
         clf_bow.fit(bow_x_train,y_train)
```

```
CPU times: user 10min 42s, sys: 11.1 s, total: 10min 53s
Wall time: 24min 7s
```

```
Out[56]: GridSearchCV(cv=4, error_score=nan,
                      estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                                       class_weight='balanced',
                                                       criterion='gini', max_depth=None,
                                                       max_features='auto',
                                                       max_leaf_nodes=None,
                                                       max_samples=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=100, n_jobs=-1,
                                                       oob_score=False,
                                                       random_state=None, verbose=0,
                                                       warm_start=False),
                      iid='deprecated', n_jobs=None,
                      param_grid={'n_estimators': [100, 200, 300, 400, 500]},
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
                      scoring='roc_auc', verbose=0)
```

```
In [57]: print(clf_bow.best_params_)
```

```
{'n_estimators': 500}
```

```
In [58]: print(clf_bow.best_score_)
```

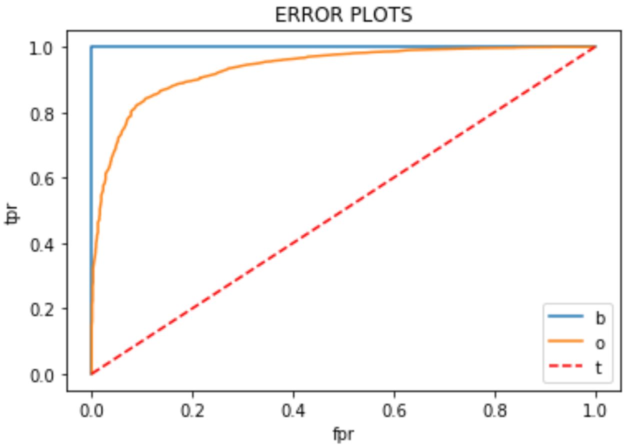
```
0.9221189236360187
```

In [69]:

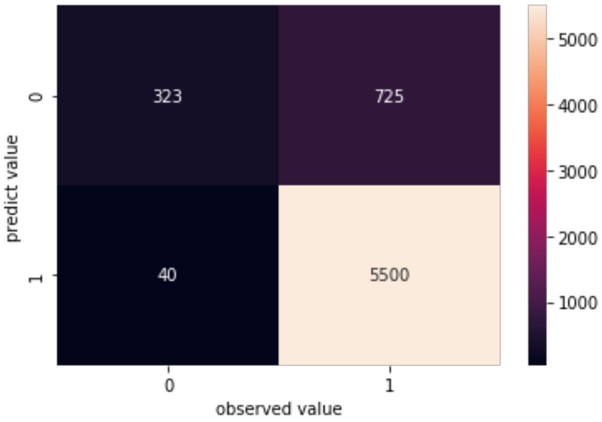
```
def plot_model(title,clf,x_train,y_train,x_test,y_test):

    print(title)
    print("="*100)

    y_train_pred=clf.predict_proba(x_train)
    train_fpr,train_tpr,train_thres=roc_curve(y_train,y_train_pred[:,1])
    train_auc_score=auc(train_fpr,train_tpr)
    print("train_auc_score:",train_auc_score)
    y_test_pred=clf.predict_proba(x_test)
    test_fpr,test_tpr,test_thres=roc_curve(y_test,y_test_pred[:,1])
    test_auc_score=auc(test_fpr,test_tpr)
    print("test_auc_score",test_auc_score)
    plt.plot(train_fpr,train_tpr,label="train auc =%0.2f"+str(train_auc_score))
    plt.plot(test_fpr,test_tpr,label="test auc = %0.2f"+str(test_auc_score))
    plt.plot([0,1],[0,1],"r--")
    plt.xlabel("fpr")
    plt.ylabel("tpr")
    plt.legend("bottom right")
    plt.title("ERROR PLOTS")
    plt.show()

    y_pred_test=clf.predict(x_test)
    accuracy=accuracy_score(y_test,y_pred_test)
    print("the accuracy score of our model on test:",accuracy)

    class_report=classification_report(y_test,y_pred_test)
    print("the classifiction reports \n",class_report)

    confuse_mat=confusion_matrix(y_test,y_pred_test)
    print("the confusion matrix :\n",confuse_mat)

    sns.heatmap(confuse_mat,annot=True,fmt="g")
    plt.xlabel("observed value")
    plt.ylabel("predict value")
    plt.show()
```

In [70]:
```
plot_model("Random Forest on Bag of words",clf_bow,bow_x_train,y_train,bow_x_test,y_te
```

```
Random Forest on Bag of words
====================================================================================
================
train_auc_score: 1.0
test_auc_score 0.9345570555570866
```



```
the accuracy score of our model on test: 0.8838797814207651
the classifiction reports
               precision    recall  f1-score   support

           0       0.89      0.31      0.46      1048
           1       0.88      0.99      0.93      5540

    accuracy                           0.88      6588
   macro avg       0.89      0.65      0.70      6588
weighted avg       0.88      0.88      0.86      6588

the confusion matrix :
 [[ 323  725]
 [  40 5500]]
```



In [71]:
```
tf_vec=TfidfVectorizer(min_df=10)
```

In [72]:
```
tf_vec.fit(x_train)
```

Out[72]:
```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=None,
                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)
```

In [73]:
```
tfidf_x_train=tf_vec.transform(x_train).toarray()
tfidf_x_test=tf_vec.transform(x_test).toarray()
```

In [74]:
```
tfidf_features_names=tf_vec.get_feature_names()
```

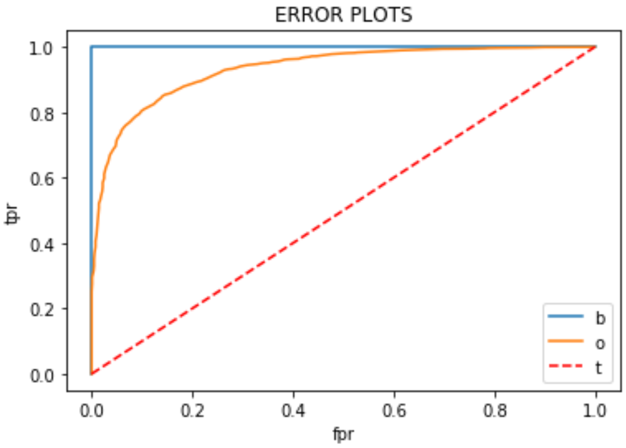
In [75]:
```
tfidf_vocab=tf_vec.vocabulary_
```

In [76]:
```
%%time
clf_tfidf=GridSearchCV(estimator=rm,param_grid=parameter,cv=4,scoring="roc_auc",return
clf_tfidf.fit(tfidf_x_train,y_train)
print(clf_tfidf.best_params_)
print(clf_tfidf.best_score_)
```

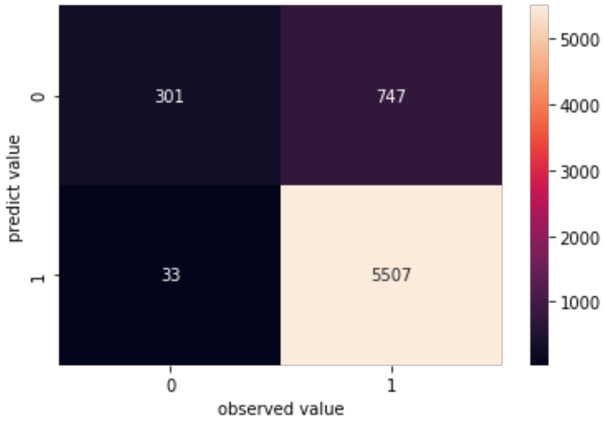
```
{'n_estimators': 300}
0.9241887548388995
CPU times: user 7min 45s, sys: 14.3 s, total: 7min 59s
Wall time: 26min 36s
```

In [77]:
```
plot_model("Random forest with tfidf",clf_tfidf,tfidf_x_train,y_train,tfidf_x_test,y_t
```

```
Random forest with tfidf
====================================================================================
================
train_auc_score: 1.0
test_auc_score 0.9327563073552512
```



```
the accuracy score of our model on test: 0.8816029143897997
the classifiction reports
               precision    recall  f1-score   support

           0       0.90      0.29      0.44      1048
           1       0.88      0.99      0.93      5540

    accuracy                           0.88      6588
   macro avg       0.89      0.64      0.68      6588
weighted avg       0.88      0.88      0.85      6588

the confusion matrix :
 [[ 301  747]
 [  33 5507]]
```



In [79]:
```
from gensim.models import Word2Vec
```

In [146]:
```
list_of_sentance_train=[]
for sentance in x_train:
    list_of_sentance_train.append(sentance.split())
```

In [147]:
```
print(list_of_sentance_train[1])
```

```
['glutino', 'gluten', 'free', 'cracker', 'probabl', 'closest', 'thing', 'get', 'ritz
', 'club', 'like', 'cracker', 'these', 'cracker', 'crisp', 'without', 'feel', 'stale
', 'nice', 'crunch', 'i', 'miss', 'regular', 'cracker', 'the', 'flavor', 'decent', '
i', 'never', 'ate', 'cracker', 'alon', 'i', 'still', 'i', 'judg', 'tast', 'well', 'p
air', 'dip', 'spread', 'chees', 'the', 'flavor', 'subtl', 'enough', 'put', 'virtual
', 'anyth', 'top', 'be', 'warn', 'get', 'stale', 'quick', 'keep', 'zip', 'lock', 'ba
g', 'even', 'box', 'the', 'price', 'cours', 'anoth', 'downsid', 'like', 'gluten', 'f
ree', 'food', 'i', 'develop', 'avers', 'price', 'gluten', 'free', 'product']
```

In [158]:
```
w2v_models=Word2Vec(list_of_sentance_train,size=100,min_count=10,workers=4)
```

In [159]:
```
w2v_models.vocabulary
```

Out[159]:
```
<gensim.models.word2vec.Word2VecVocab at 0x7f3c581a07d0>
```

In [160]:
```
w2v_words=list(w2v_models.wv.vocab)
print(w2v_words[:50])
```

```
['robust', 'aromat', 'time', 'pop', 'contain', 'top', 'last', 'drop', 'pot', 'br', '
great', 'flavor', 'black', 'mix', 'cream', 'sugar', 'glutino', 'gluten', 'free', 'cr
acker', 'probabl', 'closest', 'thing', 'get', 'ritz', 'club', 'like', 'these', 'cris
p', 'without', 'feel', 'stale', 'nice', 'crunch', 'i', 'miss', 'regular', 'the', 'de
cent', 'never', 'ate', 'alon', 'still', 'judg', 'tast', 'well', 'pair', 'dip', 'spre
ad', 'chees']
```

In [151]:
```
w2v_models.wv["robust"]
```

Out[151]:
```
array([ 6.51459172e-02, -7.78796613e-01,  2.80000001e-01, -1.15873307e-01,
        2.27104917e-01, -2.63447672e-01,  5.04275877e-03, -3.60053539e-01,
       -1.91085145e-01, -7.23941207e-01,  1.03078105e-01, -2.45274186e-01,
       -2.60765910e-01, -8.56026649e-01,  2.03870103e-01, -2.66229630e-01,
        3.71845365e-01,  6.93805158e-01,  2.10332870e-02,  5.75901031e-01,
        1.00094378e+00,  1.15692258e-01, -5.58268130e-01, -7.22508907e-01,
       -8.95051509e-02, -4.06676948e-01,  5.56860030e-01, -2.79707164e-02,
       -5.93563676e-01,  3.42972249e-01, -3.46264452e-01, -2.49021590e-01,
       -9.85664546e-01, -6.12769246e-01,  5.50739467e-01, -5.50841451e-01,
       -5.80104709e-01,  1.66340202e-01, -3.29534203e-01,  8.75152588e-01,
       -5.43673038e-01, -1.09755814e+00, -6.01606965e-01,  3.54879647e-01,
       -9.77566764e-02, -1.35977857e-03, -3.69163692e-01, -2.13634953e-01,
        1.79683864e-01, -3.60152453e-01,  4.48719949e-01, -1.09393764e+00,
       -2.32582822e-01, -3.18637043e-01,  7.39820957e-01, -8.12138379e-01,
       -4.40511376e-01,  1.78786069e-01,  4.65160161e-01,  5.63497901e-01,
       -1.10800803e-01,  1.33539056e-02,  5.83385348e-01,  8.72598946e-01,
       -9.72747803e-01, -4.75852460e-01,  2.35692393e-02, -9.43671286e-01,
       -4.77544785e-01, -4.81497735e-01,  2.42418051e-01,  2.17397541e-01,
       -3.04513387e-02,  4.01639372e-01, -8.15329850e-01,  4.81980950e-01,
       -3.94194067e-01, -1.30939376e+00,  4.24000621e-02,  4.90813315e-01,
       -1.35115814e+00,  1.40824854e+00, -3.54114711e-01,  5.31737983e-01,
       -3.55638117e-01, -2.38751635e-01, -6.91885173e-01,  1.78309411e-01,
        1.71968591e+00, -4.46066111e-01,  5.94441712e-01, -5.32339036e-01,
       -1.45108953e-01,  2.74965495e-01,  8.53092849e-01,  7.79579818e-01,
        2.42316425e-01,  1.04311742e-02,  9.84088480e-01,  3.21566433e-01],
      dtype=float32)
```

In [161]:
```
def avg_words2vec(list_of_sentance,w2v_model,w2v_words):
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in list_of_sentance: # for each review/sentence
        sent_vec = np.zeros(100) # as word vectors are of zero length 50, you might ne
        cnt_words =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:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    sent_vectors_train = np.array(sent_vectors)
    return sent_vectors,time
```

In [153]:
```
list_of_sentance_test=[]
for sentance in x_test:
    list_of_sentance_test.append(sentance.split())
```

In [154]:
```
w2v_models_test=Word2Vec(list_of_sentance_test,size=100,min_count=10,workers=4)
```

In [155]:
```
w2v_words_test=list(w2v_models_test.wv.vocab)
```

In [169]:
```
def avg_w2v(x_train):
    i=0
    list_of_sentance=[]
    for sentance in x_train:
        list_of_sentance.append(sentance.split())

    w2v_model=Word2Vec(list_of_sentance,size=100,min_count=10,workers=4)
    w2v_words=list(w2v_model.wv.vocab)
    # average Word2Vec
    # compute average word2vec for each review.
    sent_vectors = []; # the avg-w2v for each sentence/review is stored in this list
    for sent in list_of_sentance: # for each review/sentence
        sent_vec = np.zeros(100) # as word vectors are of zero length 50, you might ne
        cnt_words =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:
                vec = w2v_model.wv[word]
                sent_vec += vec
                cnt_words += 1
        if cnt_words != 0:
            sent_vec /= cnt_words
        sent_vectors.append(sent_vec)
    sent_vectors = np.array(sent_vectors)
    return sent_vectors
```

In [170]:
```
avg_x_train=avg_w2v(x_train)
avg_x_test=avg_w2v(x_test)
```

In [171]:
```
avg_x_train.shape
```

Out[171]:
```
(13375, 100)
```

In [173]:
```
avg_x_test.shape
```

Out[173]:
```
(6588, 100)
```

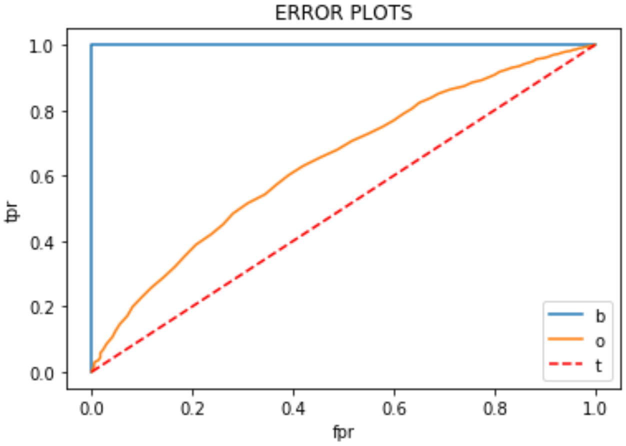
In [174]:

```
%%time
clf_avg=GridSearchCV(estimator=rm,param_grid=parameter,cv=4,scoring="roc_auc",return_t
clf_avg.fit(avg_x_train,y_train)
print(clf_tfidf.best_params_)
print(clf_tfidf.best_score_)
```

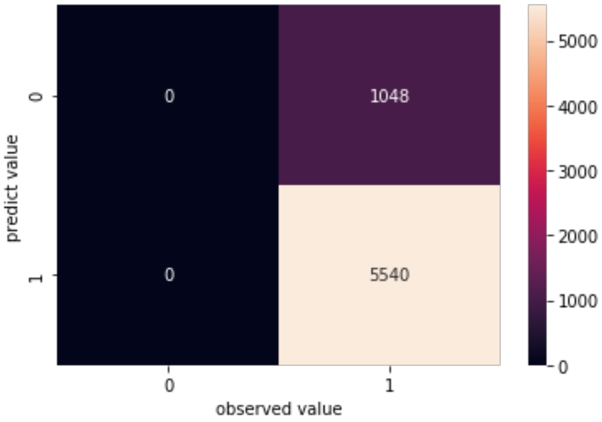
```
{'n_estimators': 300}
0.9241887548388995
CPU times: user 2min 20s, sys: 2.86 s, total: 2min 23s
Wall time: 4min 55s
```

In [181]:
```
plot_model("RandomForest with Avg-words2vec",clf_avg,avg_x_train,y_train,avg_x_test,y
```

```
RandomForest with Avg-words2vec
====================================================================================
================
train_auc_score: 1.0
test_auc_score 0.6421158059360101
```



```
the accuracy score of our model on test: 0.840922890103218
the classifiction reports
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1048
           1       0.84      1.00      0.91      5540

    accuracy                           0.84      6588
   macro avg       0.42      0.50      0.46      6588
weighted avg       0.71      0.84      0.77      6588

the confusion matrix :
 [[   0 1048]
 [   0 5540]]

/home/sushil/anaconda3/lib/python3.7/site-packages/sklearn/metrics/_classification.p
y:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```



In [187]:
```
def weighted_tfidf(x_train):
    model = TfidfVectorizer()
    model.fit(x_train)
    i=0
    list_of_sentance=[]
    for sentance in x_train:
        list_of_sentance.append(sentance.split())

    w2v_model=Word2Vec(list_of_sentance,size=100,min_count=10,workers=4)
    w2v_words=list(w2v_model.wv.vocab)
    # 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_)))
    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 = t

    tfidf_sent_vectors = []; # the tfidf-w2v for each sentence/review is stored in th
    row=0;
    for sent in list_of_sentance: # 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 w2v_words and word in tfidf_feat:
                vec = w2v_model.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
```

In [198]:
```
w_x_train=weighted_tfidf(x_train)
w_x_test=weighted_tfidf(x_test)
```

In [199]:
```
w_x_train=np.array(w_x_train)
```

In [200]:
```
w_x_test=np.array(w_x_test)
```

In [201]:
```
print(w_x_test.shape)
print(w_x_train.shape)
```

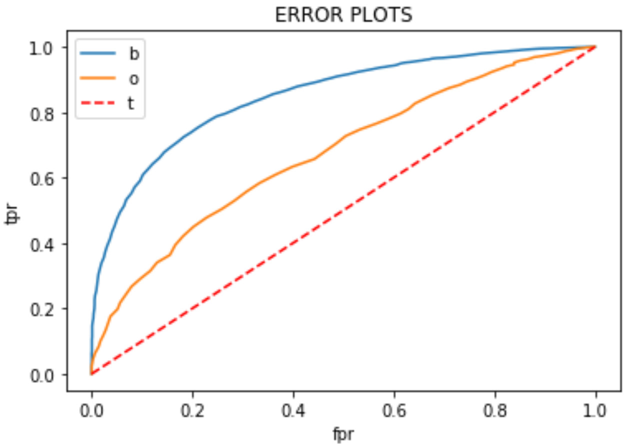
```
(6588, 100)
(13375, 100)
```

In [189]:
```
%%time
clf_w=GridSearchCV(estimator=rm,param_grid=parameter,cv=4,scoring="roc_auc",return_tra
clf_w.fit(w_x_train,y_train)
print(clf_tfidf.best_params_)
print(clf_tfidf.best_score_)
```

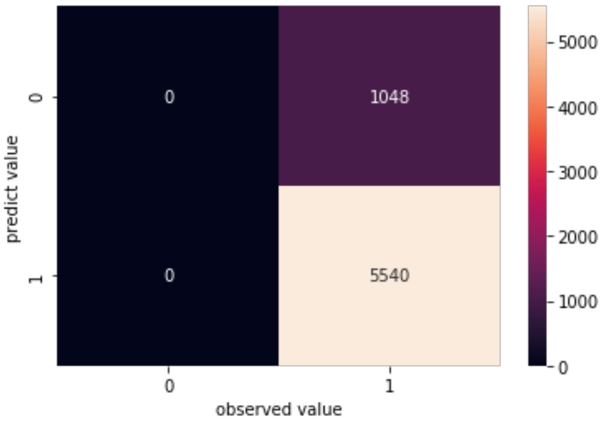
```
{'n_estimators': 300}
0.9241887548388995
CPU times: user 1min 58s, sys: 2.55 s, total: 2min 1s
Wall time: 4min 56s
```

```
In [202]: plot_model("RandomForest with WeightedTFIDF",clf_w_w,x_train,y_train,w_x_test,y_test)
```

```
RandomForest with WeightedTFIDF
====================================================================================
================
train_auc_score: 0.8501717179667327
test_auc_score 0.6744074151900129
```



```
the accuracy score of our model on test: 0.840922890103218
the classificition reports
               precision    recall  f1-score   support

           0       0.00      0.00      0.00      1048
           1       0.84      1.00      0.91      5540

    accuracy                           0.84      6588
   macro avg       0.42      0.50      0.46      6588
weighted avg       0.71      0.84      0.77      6588


/home/sushil/anaconda3/lib/python3.7/site-packages/sklearn/metrics/_classification.p
y:1272: UndefinedMetricWarning: Precision and F-score are ill-defined and being set
to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

the confusion matrix :
 [[   0 1048]
 [   0 5540]]
```



```
In [205]: from prettytable import PrettyTable
```

In [217]:

```
ptable1=PrettyTable()
```

In [218]:

```
train_auc_score=[1,1,1,0.85]
test_auc_score=[0.93,0.92,0.64,0.67]
n_neighbors=[500,300,300,300]
names=["bag_of_words","tfidf","Avg-w2v","weighted-tfidf"]
```

In [219]:

```
ptable1.add_column("Random-Forest",names)
ptable1.add_column("n_neighbors",n_neighbors)
ptable1.add_column("TRAIN AUC",train_auc_score)
ptable1.add_column("TEST AUC",test_auc_score)
print(ptable1)
```

```
+----------------+-------------+-----------+----------+
| Random-Forest  | n_neighbors | TRAIN AUC | TEST AUC |
+----------------+-------------+-----------+----------+
|  bag_of_words  |     500     |     1     |   0.93   |
|     tfidf      |     300     |     1     |   0.92   |
|    Avg-w2v     |     300     |     1     |   0.64   |
| weighted-tfidf |     300     |    0.85   |   0.67   |
+----------------+-------------+-----------+----------+
```

In [ ]: