Amazon Fine Food Reviews Analysis

Data Source: https://www.kaggle.com/snap/amazon-fine-food-reviews (https://www.kaggle.com/snap/amazon-fine-food-reviews)

EDA: https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/ (https://nycdatascience.com/blog/student-works/amazon-fine-foods-visualization/)

The Amazon Fine Food Reviews dataset consists of reviews of fine foods from Amazon.

Number of reviews: 568,454
Number of users: 256,059
Number of products: 74,258
Timespan: Oct 1999 - Oct 2012
Number of Attributes/Columns in data: 10

Attribute Information:

1. Id
2. ProductId - unique identifier for the product
3. UserId - unqiue identifier for the user
4. ProfileName
5. HelpfulnessNumerator - number of users who found the review helpful
6. HelpfulnessDenominator - number of users who indicated whether they found the review helpful or not
7. Score - rating between 1 and 5
8. Time - timestamp for the review
9. Summary - brief summary of the review
10. Text - text of the review

Objective:

Given a review, determine whether the review is positive (rating of 4 or 5) or negative (rating of 1 or 2).

[Q] How to determine if a review is positive or negative?

[Ans] We could use Score/Rating. A rating of 4 or 5 can be cosnidered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered nuetral and such reviews are ignored from our analysis. This is an approximate and proxy way of determining the polarity (positivity/negativity) of a review.

[1]. Reading Data

[1.1] Loading the data

The dataset is available in two forms

1. .csv file
2. SQLite Database

In order to load the data, We have used the SQLITE dataset as it is easier to query the data and visualise the data efficiently.

Here as we only want to get the global sentiment of the recommendations (positive or negative), we will purposefully ignore all Scores equal to 3. If the score is above 3, then the recommendation wil be set to "positive". Otherwise, it will be set to "negative".

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")


import sqlite3
import pandas as pd
import numpy as np
import nltk
import string
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import confusion_matrix
from sklearn import metrics
from sklearn.metrics import roc_curve, auc
from nltk.stem.porter import PorterStemmer

import re
# Tutorial about Python regular expressions: https://pymotw.com/2/re/
import string
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from nltk.stem.wordnet import WordNetLemmatizer

from sklearn.tree import DecisionTreeClassifier

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
import pickle

from tqdm import tqdm
import os
```

In [2]:

```
# using the SQLite Table to read data.
con = sqlite3.connect(r'C:\Sandy\privy\AI\Data Sets\Amazon Food rev dataset\c

#filtering only positive and negative reviews i.e.
# not taking into consideration those reviews with Score=3
# SELECT * FROM Reviews WHERE Score != 3 LIMIT 500000, will give top 500000 c
# you can change the number to any other number based on your computing power

#Took 3000 points from each Category i.e from Positive reviews and Negative R
#Negative Data
Neg_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score < 3 LIMIT
#Positive Data
Pos_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score > 3 LIMIT

Neg_data.head()
preprocessed_data =pd.concat([Neg_data,Pos_data])
print("Total Sample Points : ",preprocessed_data.shape)
#filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3
print("\n Sample Points : ")
preprocessed_data.head()
```

```
Total Sample Points :  (40000, 10)

 Sample Points : 
```

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDe
0	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
1	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
2	13	B0009XLVG0	A327PCT23YH90	LT	1	
3	17	B001GVISJM	A3KLWF6WQ5BNYO	Erica Neathery	0	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDe
4	27	B001GVISJM	A3RXAU2N8KV45G	lady21	0	

In [3]:
```
preprocessed_data=preprocessed_data[preprocessed_data['HelpfulnessNumerator']
```

In [4]:
```
preprocessed_data.shape
```

Out[4]: (40000, 10)

[2] Exploratory Data Analysis

[2.1] Data Cleaning: Deduplication

It is observed (as shown in the table below) that the reviews data had many duplicate entries. Hence it was necessary to remove duplicates in order to get unbiased results for the analysis of the data. Following is an example:

In [5]:
```
#Sorting data according to ProductId in ascending order
sorted_data=preprocessed_data.sort_values('ProductId', axis=0, ascending=True
```

In [6]:
```
# Give reviews with Score>3 a positive rating, and reviews with a score<3 a r
def partition(x):
    if x < 3:
        return 0
    return 1

#changing reviews with score less than 3 to be positive and vice-versa
actualScore = preprocessed_data['Score']
positiveNegative = actualScore.map(partition)
preprocessed_data['Score'] = positiveNegative
print("Number of data points in our dataset", preprocessed_data.shape)
preprocessed_data.head(3)
```

```
Number of data points in our dataset (40000, 10)
```

Out[6]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenon
0	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
1	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
2	13	B0009XLVG0	A327PCT23YH90	LT	1	

In [7]:
```
#Deduplication of entries
final_data=preprocessed_data.drop_duplicates(subset={"UserId","ProfileName",'
final_data.shape
```

Out[7]: (36569, 10)

In [8]:
```
#Checking to see how much % of data still remains

(final_data['Id'].size*1.0)/(preprocessed_data['Id'].size*1.0)*100
```

Out[8]: 91.4225

Observation:- It was also seen that in two rows given below the value of HelpfulnessNumerator is greater than HelpfulnessDenominator which is not practically possible hence these two rows too are removed from calcualtions

In [9]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDer
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	

In [10]:
```
final=final_data[final_data.HelpfulnessNumerator<=final_data.HelpfulnessDenom
```

In [11]:
```
final=final.sort_values('Time',axis=0, ascending=True , inplace=False, kind='
```

In [12]:
```
#Before starting the next phase of preprocessing lets see the number of entri
print(final.shape)

#How many positive and negative reviews are present in our dataset?
print(final['Score'].value_counts())
```

```
(36569, 10)
1    19250
0    17319
Name: Score, dtype: int64
```

In [13]:
```
#Before starting the next phase of preprocessing lets see the number of entri
print(final.shape)

y=final[['Score']]
print(len(y))
```

```
(36569, 10)
36569
```

[3] Preprocessing

[3.1]. Preprocessing Review Text

Now that we have finished deduplication our data requires some preprocessing before we go on further with analysis and making the prediction model.

Hence in the Preprocessing phase we do the following in the order below:-

1. Begin by removing the html tags
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters)
5. Convert the word to lowercase
6. Remove Stopwords
7. Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming)

After which we collect the words used to describe positive and negative reviews

In [14]:
```
# https://stackoverflow.com/a/47091490/4084039
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
```

In [15]:
```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1s

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours',
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'i
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', '
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have',
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'beca
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on',
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', '
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'th
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "shoul
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn',
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", '
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shou
            'won', "won't", 'wouldn', "wouldn't"])
```

In [16]:
```
# Combining all the above stundents
from tqdm import tqdm
from bs4 import BeautifulSoup
preprocessed_reviews_text = []
# tqdm is for printing the status bar
for sentance in tqdm(final['Text'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'html.parser').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
    preprocessed_reviews_text.append(sentance.strip())
```

```
100%|██████████████████████████████████████████████████████████
███| 36569/36569 [00:25<00:00, 1439.36it/s]
```

In [17]:
```
preprocessed_reviews_text[1]
```

Out[17]:
```
'received shipment could hardly wait try product love slickers call instead
stickers removed easily daughter designed signs printed reverse use car win
dows printed beautifully print shop program going lot fun product windows e
verywhere surfaces like tv screens computer monitors'
```

[3.2] Preprocessing Review Summary

In [18]:

```
from tqdm import tqdm
from bs4 import BeautifulSoup
preprocessed_reviews_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, 'html.parser').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
    preprocessed_reviews_summary.append(sentance.strip())
```

```
100%|██████████████████████████████████████████████████
████| 36569/36569 [00:22<00:00, 1626.73it/s]
```

Concatenating Summary and text reviews

In [19]:

```
preprocessed_reviews=[]
for i in tqdm(range(len(preprocessed_reviews_text))):
    preprocessed_reviews.append(preprocessed_reviews_text[i]+' '+preprocessec
```

```
100%|██████████████████████████████████████████████████
██| 36569/36569 [00:00<00:00, 501225.45it/s]
```

In [20]:

```
preprocessed_reviews[0]
```

Out[20]:

```
'really good idea final product outstanding use decals car window everybody
asks bought decals made two thumbs great product'
```

In [21]:

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(preprocessed_reviews,y,test_si
```

In [22]:
```
print("train x shape",len(x_train))
print("Test x Shape  ",len(x_test),"\n")
print(y_test['Score'].value_counts(),"\n")
print(y_train['Score'].value_counts(),"\n")
print("Train y  Shape  ",len(y_train),"\n")
print("Test Y  Shape  ",len(y_test))
```

```
train x shape 25598
Test x Shape   10971

0    5841
1    5130
Name: Score, dtype: int64

1    14120
0    11478
Name: Score, dtype: int64

Train y  Shape   25598

Test Y  Shape   10971
```

[4] Featurization

[4.1] BAG OF WORDS

In [23]:

```
#BoW
count_vect = CountVectorizer(ngram_range=(1,2),min_df=10,analyzer='word',max_
count_vect.fit(x_train)
print("some feature names ", count_vect.get_feature_names()[:10])
print('='*50)

x_train_BOW = count_vect.transform(x_train)
print("the type of count vectorizer ",type(x_train_BOW))
print("the shape of out text BOW vectorizer ",x_train_BOW.get_shape())
print("the number of unique words ", x_train_BOW.get_shape()[1])
print("=="*50)


x_test_BOW=count_vect.transform(x_test)
print("the type of count vectorizer ",type(x_test_BOW))
print("the shape of out text BOW vectorizer ",x_test_BOW.get_shape())
print("the number of unique words ", x_test_BOW.get_shape()[1])
```

```
some feature names  ['ability', 'able', 'able eat', 'able find', 'able ge
t', 'absolute', 'absolute favorite', 'absolutely', 'absolutely delicious',
'absolutely love']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (25598, 5000)
the number of unique words  5000
===========================================================================
=========================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (10971, 5000)
the number of unique words  5000
```

In [24]:

```
def find_best_threshold(threshold, fpr, tpr):
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very h
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshol
    return t

def predict_with_best_t(proba, threshold):
    predictions = []
    for i in proba:
        if i>=threshold:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

[4.2] TF-IDF

In [25]:

```
from sklearn.model_selection import train_test_split

x_train_tf_idf_sent,x_test_tf_idf_sent,y_train_tf_idf,y_test_tf_idf=train_tes
```

In [26]:

```
tf_idf_vect = TfidfVectorizer(ngram_range=(1,2),min_df=10,analyzer='word',max
tf_idf_vect.fit(x_train_tf_idf_sent)
print("some sample features(unique words in the corpus)",tf_idf_vect.get_feat
print('='*50)

x_train_tf_idf = tf_idf_vect.transform(x_train_tf_idf_sent)
print("the type of count vectorizer ",type(x_train_tf_idf))
print("the shape of out text TFIDF vectorizer ",x_train_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", x_tr
print('='*50)


x_test_tf_idf = tf_idf_vect.transform(x_test_tf_idf_sent)
print("the type of count vectorizer ",type(x_test_tf_idf))
print("the shape of out text TFIDF vectorizer ",x_test_tf_idf.get_shape())
print("the number of unique words including both unigrams and bigrams ", x_te
```

```
some sample features(unique words in the corpus) ['ability', 'able', 'able
find', 'able get', 'able use', 'absolute', 'absolutely', 'absolutely delici
ous', 'absolutely love', 'absolutely loves']
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (25598, 5000)
the number of unique words including both unigrams and bigrams  5000
==================================================
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text TFIDF vectorizer  (10971, 5000)
the number of unique words including both unigrams and bigrams  5000
```

Random Forest

[5.1] Applying Random Forest on BOW, SET 1

In [27]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc,roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier as RFC

#parametres
depth=[i*2 for i in range(1,5)]
n_estimators=[i*5 for i in range(1,2)]
p_grid_DT = {'max_depth': depth,'n_estimators':n_estimators}

#Random Forest Model
RF_clasifier=RFC(criterion='gini',min_samples_split=50,max_features='auto')

#Grid Search Fitment
clf=GridSearchCV(RF_clasifier,scoring='roc_auc',iid=True,param_grid=p_grid_DT
clf.fit(x_train_BOW,y_train)

#train Auc
train_auc= clf.cv_results_['mean_train_score']
train_auc_std_temp= clf.cv_results_['std_train_score']

#CV Auc
cv_auc= clf.cv_results_['mean_test_score']
cv_auc_std_temp= clf.cv_results_['std_test_score']
```

In [28]:

```
#optimal parametres
params_BOW=clf.best_params_
auc_BOW=clf.best_score_
opt_depth_BOW=clf.best_params_['max_depth']
opt_n_estimators_BOW=clf.best_params_['n_estimators']
print("best params :",clf.best_params_)
print("best auc :",auc_BOW)
```

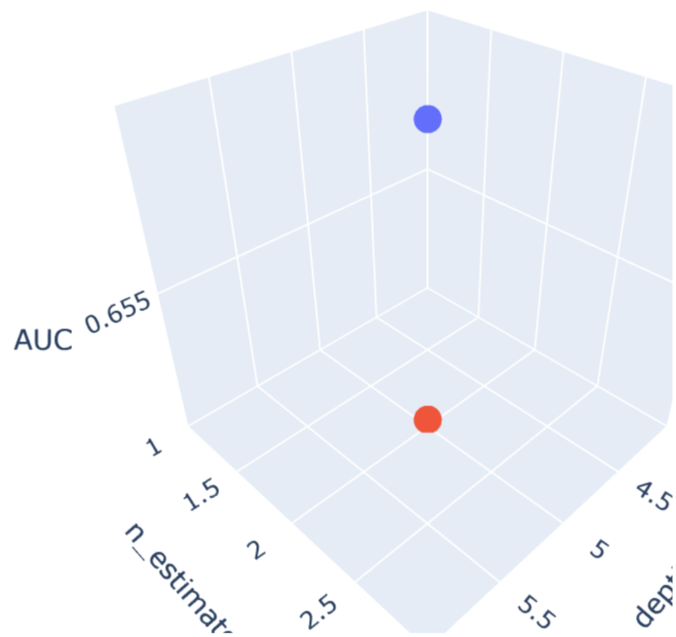
```
best params : {'max_depth': 8, 'n_estimators': 5}
best auc : 0.7943932162663988
```

In [29]:

```
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
import numpy as np
```

In [30]:

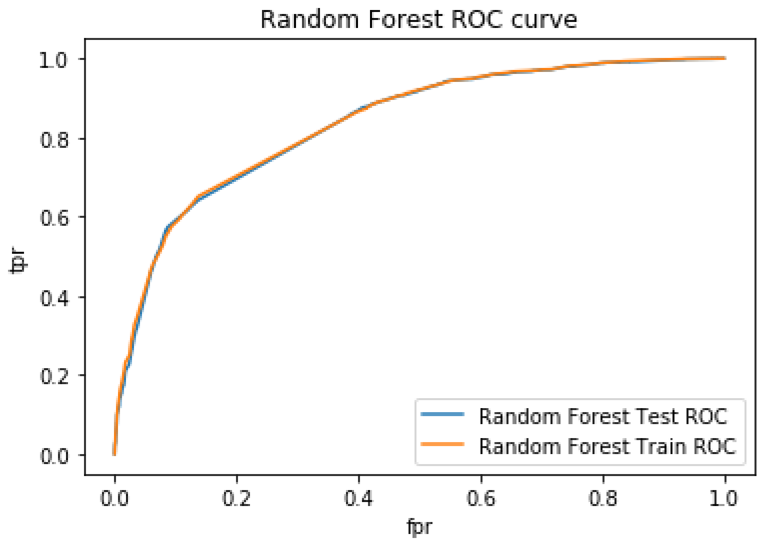
```
t1=go.Scatter3d(x=n_estimators,y=depth,z=train_auc,name = 'train')
t2=go.Scatter3d(x=n_estimators,y=depth,z=cv_auc,name = 'CV')
data=[t1,t2]
layout=go.Layout(scene=dict(xaxis=dict(title='depth'),
                            yaxis=dict(title='n_estimators'),
                            zaxis=dict(title='AUC')))
fig=go.Figure(data,layout)
offline.iplot(fig,filename='RF_BOW_3D_PLOT')
```



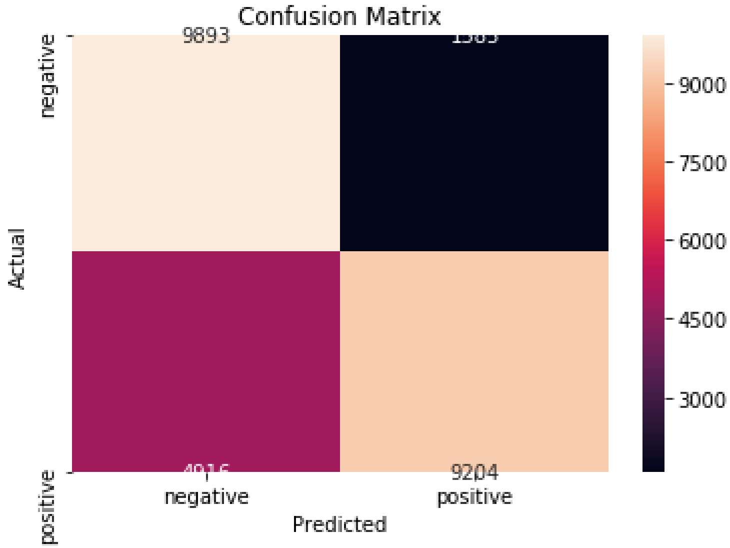
In [31]:
```
clf_bow=RFC(n_estimators=opt_n_estimators_BOW,criterion='gini',max_depth=opt_
clf_bow.fit(x_train_BOW,y_train)


#predict probabilities
y_Test_pred_proba = clf_bow.predict_proba(x_test_BOW)[:,1]
y_Train_pred_proba = clf_bow.predict_proba(x_train_BOW)[:,1]

#code for AUC
fpr_Test, tpr_Test, thresholds_Test = roc_curve(y_test, y_Test_pred_proba)
fpr_Train, tpr_Train, thresholds_train = roc_curve(y_train, y_Train_pred_prob
print("Test data AUC of Random Forest with BOW Implentation  : ",roc_auc_scor
AUC_BOW=roc_auc_score(y_test,y_Test_pred_proba)
print("Train data AUC of Random Forest with BOW Implentation  : ",roc_auc_sco


#generate plot
plt.plot(fpr_Test,tpr_Test, label='Random Forest Test ROC')
plt.plot(fpr_Train,tpr_Train, label='Random Forest Train ROC')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.legend()
plt.title('Random Forest ROC curve')
plt.show()


#confusion matrix of train data
best_t = find_best_threshold(thresholds_train, fpr_Train, tpr_Train)
print("Train confusion matrix")
conf_matrix=confusion_matrix(y_train, predict_with_best_t(y_Train_pred_proba,
print(confusion_matrix(y_train, predict_with_best_t(y_Train_pred_proba, best_

class_label = ['negative', 'positive']
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

#confusion matrix of test data
best_t = find_best_threshold(thresholds_Test, fpr_Test, tpr_Test)
print("Test confusion matrix")
conf_matrix=confusion_matrix(y_test, predict_with_best_t(y_Test_pred_proba, b
print(confusion_matrix(y_test, predict_with_best_t(y_Test_pred_proba, best_t)
class_label = ['negative', 'positive']
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

Accuracy_BOW=clf_bow.score(x_test_BOW, y_test)
```

```
print('Accuracy of Random Forest  when Max_Depth={} and n_estimators= {} is {
```

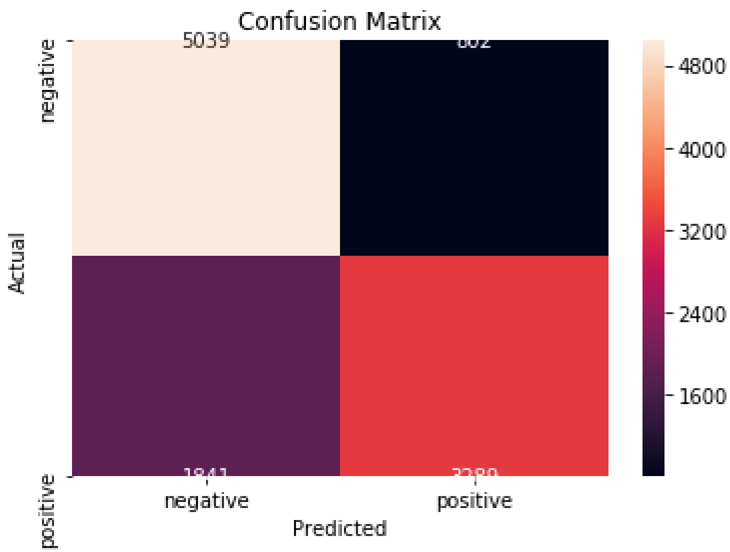
```
Test data AUC of Random Forest with BOW Implentation  :  0.8356597995016075
Train data AUC of Random Forest with BOW Implentation  :  0.838122526676233
```




```
the maximum value of tpr*(1-fpr) 0.561828417166576 for threshold 0.557
Train confusion matrix
[[9893 1585]
 [4916 9204]]
```




```
the maximum value of tpr*(1-fpr) 0.553100002569722 for threshold 0.557
Test confusion matrix
[[5039  802]
 [1841 3289]]
```



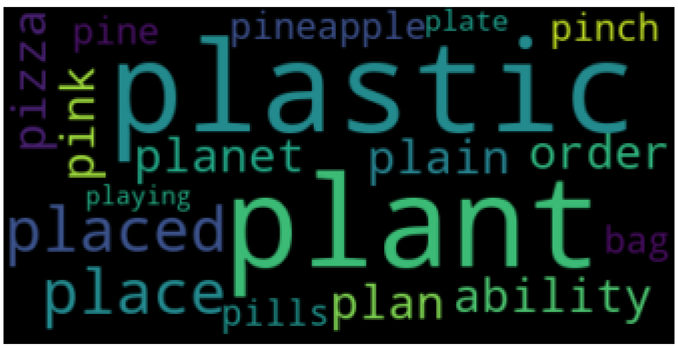

```
Accuracy of Random Forest  when Max_Depth=8 and n_estimators= 5 is 0.689089
4175553732
```

[5.1.1] Top 10 important features of positive class from SET 1

In [32]:
```
pos_class_prob_sorted = clf_bow.feature_importances_.argsort()

print(np.take(count_vect.get_feature_names(), pos_class_prob_sorted[:20]))
pos_rev_words=np.take(count_vect.get_feature_names(), pos_class_prob_sorted[:
```

```
['ability' 'plastic' 'plants' 'plant' 'planet' 'plan' 'plain' 'places'
 'placed order' 'placed' 'place' 'pizza' 'pink' 'pineapple' 'pine' 'pinch'
 'plastic bag' 'pills' 'plate' 'playing']
```

In [33]:
```
from  wordcloud  import WordCloud
temp=''
for i in pos_rev_words:
    temp=temp+' '+i
pos_rev=WordCloud().generate(temp)
plt.imshow(pos_rev,interpolation='bilinear')
plt.axis("off")
plt.show()
```



[5.1.2] Top 10 important features of negative class from SET 1

In [34]:
```
neg_class_prob_sorted= clf_bow.feature_importances_.argsort()
#print(neg_class_prob_sorted[:1])
len_features=len(neg_class_prob_sorted)
#print(len_features)
print(np.take(count_vect.get_feature_names(), neg_class_prob_sorted[len_featu
neg_rev_words=np.take(count_vect.get_feature_names(), neg_class_prob_sorted[]
```

```
['box' 'good' 'treat' 'poor' 'disappointing' 'threw' 'not' 'not worth'
 'perfect' 'loves' 'awful' 'disappointed' 'nice' 'delicious' 'not good'
 'horrible' 'away' 'thought' 'not buy' 'great']
```

In [35]:
```
from  wordcloud  import WordCloud
temp=''
for i in neg_rev_words:
    temp=temp+' '+i
pos_rev=WordCloud().generate(temp)
plt.imshow(pos_rev,interpolation='bilinear')
plt.axis("off")
plt.show()
```

[5.2] Applying Random Forest on TFIDF, SET 2

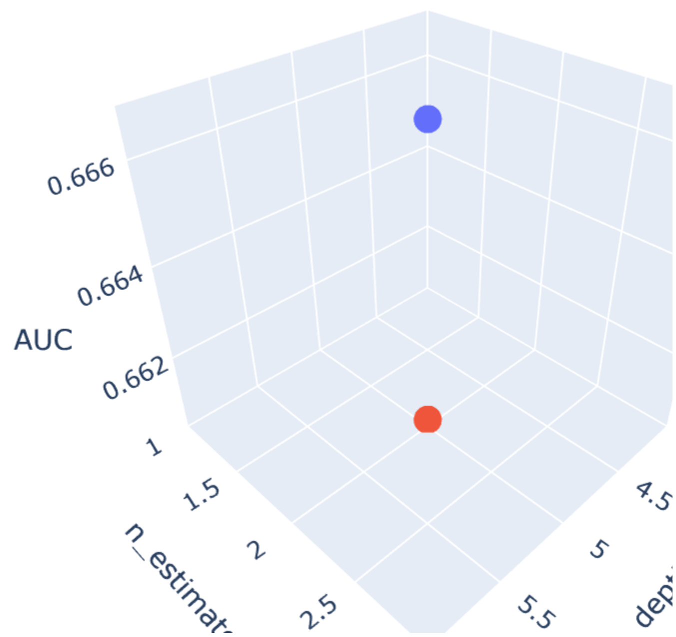
In [36]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc,roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier as RFC


#parametres
#depth=[i*2 for i in range(1,50)]
#n_estimators=[i*5 for i in range(1,20)]
p_grid_DT = {'max_depth': depth,'n_estimators':n_estimators}

#Random Forest Train Model
RF_clasifier=RFC(criterion='gini',min_samples_split=50,max_features='auto')

#Grid Search Fitment Model
clf=GridSearchCV(RF_clasifier,scoring='roc_auc',iid=True,param_grid=p_grid_DT
clf.fit(x_train_tf_idf,y_train_tf_idf)

#Train Auc
train_auc= clf.cv_results_['mean_train_score']
train_auc_std_temp= clf.cv_results_['std_train_score']

#Test Auc
cv_auc= clf.cv_results_['mean_test_score']
cv_auc_std_temp= clf.cv_results_['std_test_score']

#Optimal Parametres
params_tf_idf=clf.best_params_
auc_tf_idf=clf.best_score_
opt_depth_tf_idf=clf.best_params_ ['max_depth']
opt_n_estimators_tf_idf=clf.best_params_['n_estimators']
print("best params :",clf.best_params_)
print("best Auc :",auc_tf_idf)
```

```
best params : {'max_depth': 8, 'n_estimators': 5}
best Auc : 0.8164482337662151
```

In [37]:

```
t1=go.Scatter3d(x=n_estimators,y=depth,z=train_auc,name = 'train')
t2=go.Scatter3d(x=n_estimators,y=depth,z=cv_auc,name = 'CV')
data=[t1,t2]
layout=go.Layout(scene=dict(xaxis=dict(title='depth'),
                            yaxis=dict(title='n_estimators'),
                            zaxis=dict(title='AUC')))
fig=go.Figure(data,layout)
offline.iplot(fig,filename='RF_TF_IDF_3D_PLOT')
```

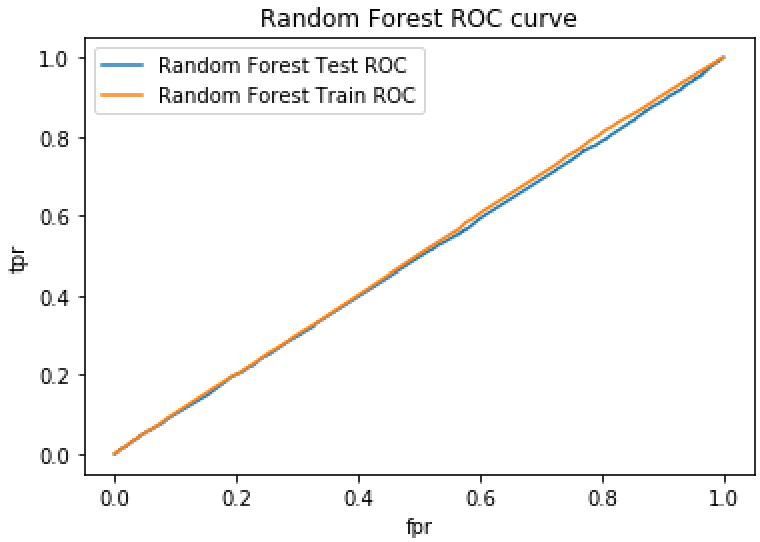


In [38]:

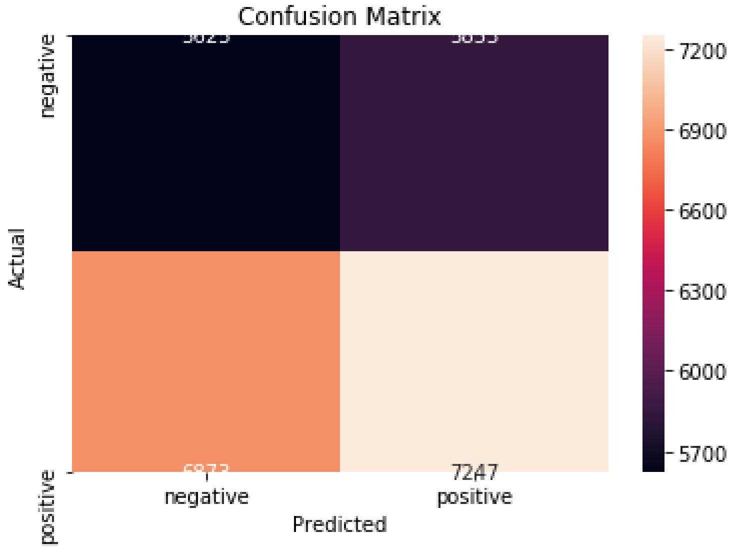
```
clf_tf_idf=RFC(n_estimators=opt_n_estimators_tf_idf,criterion='gini',max_dept
clf_tf_idf.fit(x_train_tf_idf,y_train_tf_idf)


#predict probabilities
y_Test_pred_proba = clf_tf_idf.predict_proba(x_test_tf_idf)[:,1]
y_Train_pred_proba = clf_tf_idf.predict_proba(x_train_tf_idf)[:,1]

#code for AUC
fpr_Test, tpr_Test, thresholds_Test = roc_curve(y_test, y_Test_pred_proba)
fpr_Train, tpr_Train, thresholds_train = roc_curve(y_train, y_Train_pred_prob
print("Test data AUC of Random Forest with Tf_idf Implentation  : ",roc_auc_s
AUC_tf_idf=roc_auc_score(y_test,y_Test_pred_proba)
print("Train data AUC of Random Forest with Tf_idf Implentation  : ",roc_auc_


#generate plot
plt.plot(fpr_Test,tpr_Test, label='Random Forest Test ROC')
plt.plot(fpr_Train,tpr_Train, label='Random Forest Train ROC')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.legend()
plt.title('Random Forest ROC curve')
plt.show()


#confusion matrix of train data
best_t = find_best_threshold(thresholds_train, fpr_Train, tpr_Train)
print("Train confusion matrix")
conf_matrix=confusion_matrix(y_train, predict_with_best_t(y_Train_pred_proba,
print(confusion_matrix(y_train, predict_with_best_t(y_Train_pred_proba, best_

class_label = ['negative', 'positive']
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

#confusion matrix of test data
best_t = find_best_threshold(thresholds_Test, fpr_Test, tpr_Test)
print("Test confusion matrix")
conf_matrix=confusion_matrix(y_test, predict_with_best_t(y_Test_pred_proba, b
print(confusion_matrix(y_test, predict_with_best_t(y_Test_pred_proba, best_t)
class_label = ['negative', 'positive']
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

Accuracy_tf_idf=clf_tf_idf.score(x_test_tf_idf, y_test_tf_idf)
```

```
print('Accuracy of Random Forest when Max_Depth={} and n_estimators = {} is {
```

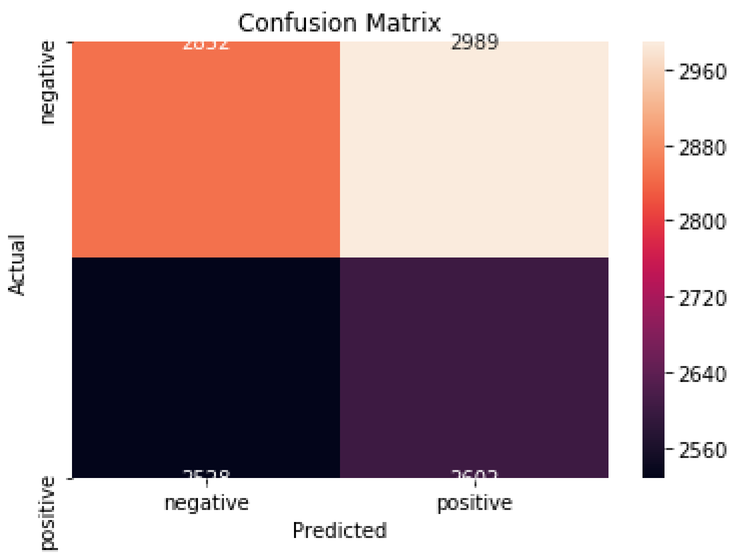
```
Test data AUC of Random Forest with Tf_idf Implentation  :  0.4949368632637
5397
Train data AUC of Random Forest with Tf_idf Implentation  :  0.503270575018
0047
```




```
the maximum value of tpr*(1-fpr) 0.2515242548005373 for threshold 0.553
Train confusion matrix
[[5625 5853]
 [6873 7247]]
```




```
the maximum value of tpr*(1-fpr) 0.2476579319477525 for threshold 0.553
Test confusion matrix
[[2852 2989]
 [2528 2602]]
```




```
Accuracy of Random Forest when Max_Depth=8 and n_estimators = 5 is 0.7440
```

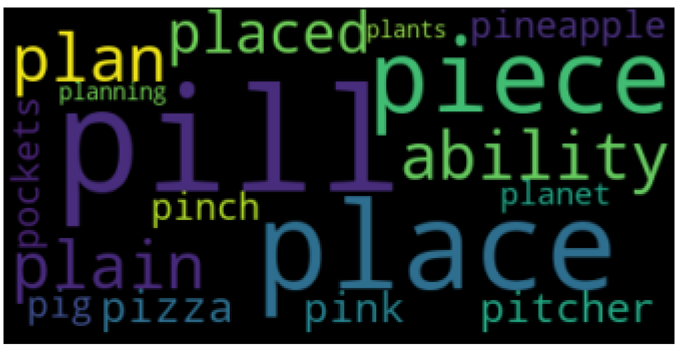
```
525020508614
```

[5.2.1] Top 10 important features of positive class from SET 2

In [39]:
```
pos_feature_sorted = clf_tf_idf.feature_importances_.argsort()
#print(len(pos_feature_sorted))
print(np.take(tf_idf_vect.get_feature_names(), pos_feature_sorted[:20]))
pos_rev_words=np.take(tf_idf_vect.get_feature_names(), pos_feature_sorted[:2(
```

```
['ability' 'plan' 'plain' 'places' 'placed' 'place' 'pizza' 'pitcher'
 'pink' 'pineapple' 'pinch' 'pills' 'pill pockets' 'pill' 'pig' 'pieces'
 'planet' 'piece' 'planning' 'plants']
```

In [40]:
```
from  wordcloud  import WordCloud
temp=''
for i in pos_rev_words:
    temp=temp+' '+i
pos_rev=WordCloud().generate(temp)
plt.imshow(pos_rev,interpolation='bilinear')
plt.axis("off")
plt.show()
```



[5.2.2] Top 10 important features of negative class from SET 2

In [41]:
```
neg_class_prob_sorted = clf_tf_idf.feature_importances_.argsort()
len_features=len(neg_class_prob_sorted)
print(np.take(tf_idf_vect.get_feature_names(), neg_class_prob_sorted[len_feat
neg_rev_words=np.take(tf_idf_vect.get_feature_names(), neg_class_prob_sorted[
```

```
['chips' 'not like' 'product not' 'date' 'bland' 'not recommend' 'gross'
 'recommend' 'not even' 'threw' 'thought' 'terrible' 'treat' 'best'
 'waste money' 'not good' 'money' 'disappointed' 'great' 'not']
```

In [42]:

```
from  wordcloud  import WordCloud
temp=''
for i in neg_rev_words:
    temp=temp+' '+i
pos_rev=WordCloud().generate(temp)
plt.imshow(pos_rev,interpolation='bilinear')
plt.axis("off")
plt.show()
```

Avg Weighted W2V on Random Forest

In [43]:

```
i=0
list_of_sentance=[]
for sentance in tqdm(preprocessed_reviews):
    list_of_sentance.append(sentance.split())
print(list_of_sentance[1:2])
print('\n')
print(type(preprocessed_reviews))
```

```
100%|██████████████████████████████████████████████████████████████
█| 36569/36569 [00:00<00:00, 109551.02it/s]

[['received', 'shipment', 'could', 'hardly', 'wait', 'try', 'product', 'lov
e', 'slickers', 'call', 'instead', 'stickers', 'removed', 'easily', 'daught
er', 'designed', 'signs', 'printed', 'reverse', 'use', 'car', 'windows', 'p
rinted', 'beautifully', 'print', 'shop', 'program', 'going', 'lot', 'fun',
'product', 'windows', 'everywhere', 'surfaces', 'like', 'tv', 'screens', 'c
omputer', 'monitors', 'wow', 'make', 'islickers']]


<class 'list'>
```

In [44]:

```
print(list_of_sentance[0:1])
X_train_Avg_W2V_sent, X_test_Avg_W2V_sent, y_train, y_test = train_test_split
#X_train_Avg_W2V_sent, CV_Avg_W2V_sent, y_train, y_CV = train_test_split(X_1,
#print(X_train_Avg_W2V_sent[0:1])
```

```
[['really', 'good', 'idea', 'final', 'product', 'outstanding', 'use', 'deca
ls', 'car', 'window', 'everybody', 'asks', 'bought', 'decals', 'made', 'tw
o', 'thumbs', 'great', 'product']]
```

In [45]:

```
w2v_model=Word2Vec(X_train_Avg_W2V_sent,min_count=5,size=50, workers=4)
w2v_words = list(w2v_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  10437
sample words  ['absolutely', 'worst', 'thing', 'ever', 'tasted', 'entire',
'life', 'low', 'carb', 'long', 'time', 'understand', 'even', 'though', 'som
ething', 'advertised', 'country', 'biscuit', 'wont', 'consistancy', 'tast
e', 'regular', 'would', 'however', 'nothing', 'resembling', 'tasting', 'lik
e', 'definitely', 'not', 'recommend', 'product', 'every', 'last', 'bit', 'w
ent', 'trash', 'along', 'side', 'money', 'spent', 'purchase', 'dixie', 'cou
nters', 'mix', 'sure', 'reviews', 'came', 'freeze', 'dried']
```

In [46]:

```
# average Word2Vec
# compute average word2vec for each review.
from tqdm import  tqdm
X_train_Avg_W2V = []; # the avg-w2v for each sentence/review is stored in thi
for sent in tqdm(X_train_Avg_W2V_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero's with length 50, y
    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
    X_train_Avg_W2V.append(sent_vec)
print(len(X_train_Avg_W2V))
print(len(X_train_Avg_W2V[0]))
print(X_train_Avg_W2V[0:1])
```

```
100%|██████████████████████████████████████████████████████████████████████
█████| 25598/25598 [01:39<00:00, 257.35it/s]

25598
50
[array([-0.02975664, -0.43181064, -0.12610799,  0.19262551,  0.14883617,
        0.39310726,  0.42155959, -0.19430732,  0.28795124,  0.19552924,
        0.5526067 , -0.2336948 , -0.3180427 , -0.12945631, -0.50018308,
       -0.27680368,  0.22176353, -0.38532788,  0.01383564, -0.31525447,
       -0.50833173, -0.3125778 , -0.02663645, -0.03753659,  0.5384718 ,
       -0.34501125, -0.42755425,  0.39521397, -0.287917  , -0.31389219,
       -0.63431548,  0.52366676, -0.17427099,  0.78092532,  0.10174117,
       -0.35784332,  0.38479561, -0.0604132 ,  0.62368624,  0.25766425,
       -0.65587499, -0.48903547,  0.16913006, -0.59864579,  0.03928852,
        0.29542336,  0.34880483, -0.21836809,  0.10367581,  0.18046329])]
```

In [47]:

```
# average Word2Vec
# compute average word2vec for each review.
from tqdm import  tqdm
X_test_Avg_W2V = []; # the avg-w2v for each sentence/review is stored in this
for sent in tqdm(X_test_Avg_W2V_sent): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero's with length 50, y
    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
    X_test_Avg_W2V.append(sent_vec)
print(len(X_test_Avg_W2V))
print(len(X_test_Avg_W2V[0]))
print(X_test_Avg_W2V[0:1])
```

```
100%|██████████████████████████████████████████████████████████████████████
█████| 10971/10971 [00:45<00:00, 239.38it/s]

10971
50
[array([-0.21321692, -0.12780262,  0.07911558, -0.0226503 ,  0.85735976,
       -0.21990722,  0.29757709, -0.4207674 ,  0.50099765,  0.25549408,
       -0.11059337, -0.13907942, -0.39898334, -0.16903394, -0.45022868,
        0.19580003,  0.41365274, -0.51678363, -0.29433669,  0.2091103 ,
        0.02071852, -0.15394098, -0.33263889, -0.22740563,  0.1622838 ,
        0.17195116, -0.51529909, -0.2780106 , -0.15605859, -0.56426012,
       -0.20846825,  0.46248827, -0.80319923,  0.57038608,  0.60678756,
       -0.55014215,  0.37643203, -0.11128338,  0.50464831,  0.28508775,
       -0.67120012, -0.81271588, -0.48590884, -0.59615332, -0.25016286,
        0.32580751, -0.37069052, -0.29238422, -0.62449254,  0.10201115])]
```

In [48]:

```
from sklearn.preprocessing import StandardScaler


SS=StandardScaler(with_mean=False).fit(X_train_Avg_W2V)
x_train_Avg_W2V=SS.transform(X_train_Avg_W2V)
#X_CV_Avg_W2V= SS.transform(CV_Avg_W2V)
x_test_Avg_W2V=SS.transform(X_test_Avg_W2V)
print(X_train_Avg_W2V[0:1])
print("\n",len(X_train_Avg_W2V))
```

```
[array([-0.02975664, -0.43181064, -0.12610799,  0.19262551,  0.14883617,
        0.39310726,  0.42155959, -0.19430732,  0.28795124,  0.19552924,
        0.5526067 , -0.2336948 , -0.3180427 , -0.12945631, -0.50018308,
       -0.27680368,  0.22176353, -0.38532788,  0.01383564, -0.31525447,
       -0.50833173, -0.3125778 , -0.02663645, -0.03753659,  0.5384718 ,
       -0.34501125, -0.42755425,  0.39521397, -0.287917  , -0.31389219,
       -0.63431548,  0.52366676, -0.17427099,  0.78092532,  0.10174117,
       -0.35784332,  0.38479561, -0.0604132 ,  0.62368624,  0.25766425,
       -0.65587499, -0.48903547,  0.16913006, -0.59864579,  0.03928852,
        0.29542336,  0.34880483, -0.21836809,  0.10367581,  0.18046329])]

 25598
```

In [49]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc,roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier as RFC

#parametres
#depth=[i*2 for i in range(1,50)]
#n_estimators=[i*5 for i in range(1,20)]
p_grid_DT = {'max_depth': depth,'n_estimators':n_estimators}

#Random Forest Model
RF_clasifier=RFC(criterion='gini',min_samples_split=50,max_features='auto')

#Grid Search Fitment Model
clf=GridSearchCV(RF_clasifier,scoring='roc_auc',iid=True,param_grid=p_grid_DT
clf.fit(x_train_Avg_W2V,y_train)

#Train Auc
train_auc= clf.cv_results_['mean_train_score']
train_auc_std_temp= clf.cv_results_['std_train_score']

#CV Auc
cv_auc= clf.cv_results_['mean_test_score']
cv_auc_std_temp= clf.cv_results_['std_test_score']

#Optimal Parametres
params_Avg_W2V=clf.best_params_
auc_Avg_W2V=clf.best_score_
opt_depth_Avg_W2V=clf.best_params_ ['max_depth']
opt_n_estimators_Avg_W2V=clf.best_params_['n_estimators']
print("best params :",clf.best_params_)
print("best auc :",auc_Avg_W2V)
```

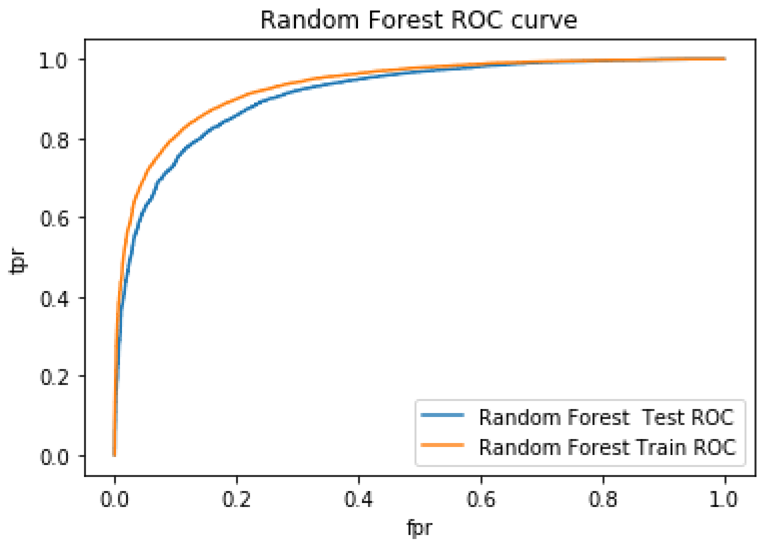
```
best params : {'max_depth': 8, 'n_estimators': 5}
best auc : 0.90661667319736
```

In [50]:

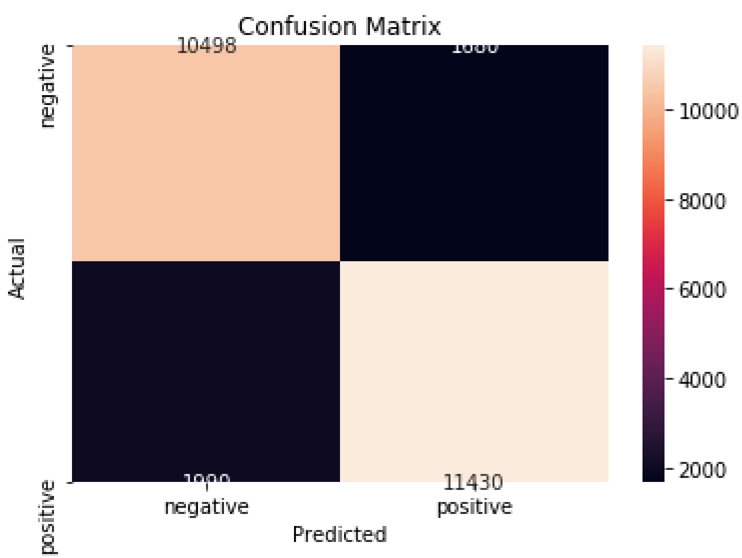
```
clf_Avg_W2V=RFC(n_estimators=opt_n_estimators_Avg_W2V,criterion='gini',max_de
clf_Avg_W2V.fit(x_train_Avg_W2V,y_train)


#predict probabilities
y_Test_pred_proba = clf_Avg_W2V.predict_proba(x_test_Avg_W2V)[:,1]
y_Train_pred_proba = clf_Avg_W2V.predict_proba(x_train_Avg_W2V)[:,1]

#code for AUC
fpr_Test, tpr_Test, thresholds_Test = roc_curve(y_test, y_Test_pred_proba)
fpr_Train, tpr_Train, thresholds_train = roc_curve(y_train, y_Train_pred_prob
print("Test data AUC of Random Forest  with Avg_W2V Implentation  : ",roc_auc
AUC_Avg_W2V=roc_auc_score(y_test,y_Test_pred_proba)
print("Train data AUC of Random Forest  with Avg_W2V Implentation  : ",roc_au


#generate plot
plt.plot(fpr_Test,tpr_Test, label='Random Forest  Test ROC')
plt.plot(fpr_Train,tpr_Train, label='Random Forest Train ROC')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.legend()
plt.title('Random Forest ROC curve')
plt.show()


#confusion matrix of train data
best_t = find_best_threshold(thresholds_train, fpr_Train, tpr_Train)
print("Train confusion matrix")
conf_matrix=confusion_matrix(y_train, predict_with_best_t(y_Train_pred_proba,
print(confusion_matrix(y_train, predict_with_best_t(y_Train_pred_proba, best_

class_label = ['negative', 'positive']
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

#confusion matrix of test data
best_t = find_best_threshold(thresholds_Test, fpr_Test, tpr_Test)
print("Test confusion matrix")
conf_matrix=confusion_matrix(y_test, predict_with_best_t(y_Test_pred_proba, b
print(confusion_matrix(y_test, predict_with_best_t(y_Test_pred_proba, best_t)
class_label = ['negative', 'positive']
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

Accuracy_Avg_W2V=clf_Avg_W2V.score(x_test_Avg_W2V, y_test)
```

```
print('Accuracy of Random Forest when Max_Depth={} and opt_n_estimators= {} i
```

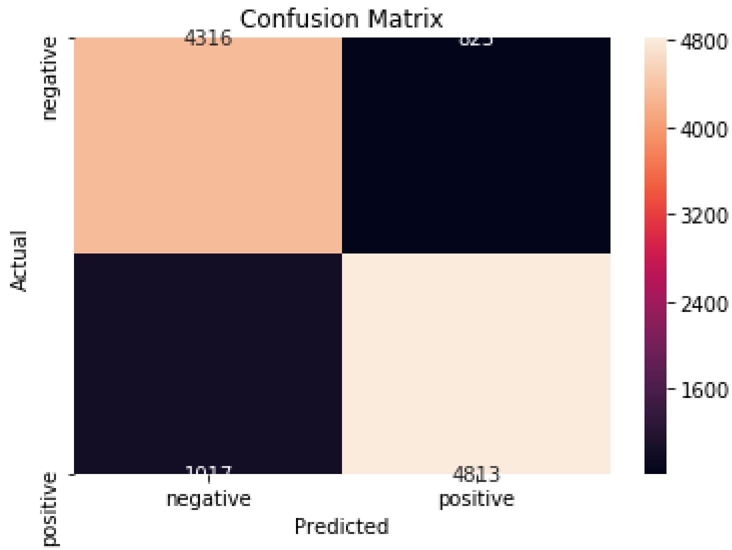
```
Test data AUC of Random Forest  with Avg_W2V Implentation  :  0.91171622342
56404
Train data AUC of Random Forest  with Avg_W2V Implentation  :  0.9326338858
595024
```




```
the maximum value of tpr*(1-fpr) 0.7342167926869175 for threshold 0.518
Train confusion matrix
[[10498  1680]
 [ 1990 11430]]
```



```
the maximum value of tpr*(1-fpr) 0.6930764449388312 for threshold 0.525
Test confusion matrix
[[4316  825]
 [1017 4813]]
```




```
Accuracy of Random Forest when Max_Depth=8 and opt_n_estimators= 5 is 0.831
3736213654179
```

TF_IDF W2V on Random Forest

In [51]:
```
X_train, X_test, y_train, y_test = train_test_split(preprocessed_reviews, y,
#X_train, CV, y_train, y_CV = train_test_split(X_1, y_t, test_size=0.3)# Plea
```

In [52]:
```
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())
print(list_of_sentance_train[1:2])
```

```
[['done', 'research', 'buying', 'product', 'dog', 'seemed', 'little', 'appe
tite', 'food', 'skin', 'problems', 'loose', 'stools', 'unfortunately', 'no
t', 'want', 'believe', 'anything', 'food', 'finally', 'decided', 'look', 'm
onths', 'trying', 'get', 'eat', 'regularly', 'found', 'problems', 'not', 'u
ncommon', 'food', 'since', 'switched', 'blue', 'buffalo', 'food', 'skin',
'problems', 'gotten', 'better', 'become', 'energetic', 'stools', 'no', 'lon
ger', 'loose', 'happily', 'eats', 'done', 'research', 'buying']]
```

In [53]:

```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
print(list_of_sentance_test[1:2])
```

```
[['bought', 'packets', 'planted', 'one', 'took', 'couple', 'days', 'sprout
s', 'cats', 'love', 'cats', 'love']]
```

In [54]:

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
model.fit(X_train)
tf_idf_matrix = model.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_)))
```

In [55]:

```
# 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

X_train_tf_idf_W2V = []; # the tfidf-w2v for each sentence/review is stored i
row=0;
for sent in tqdm(list_of_sentance_train): # 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_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
    X_train_tf_idf_W2V.append(sent_vec)
    row += 1
```

```
100%|
| 25598/25598 [16:25<00:00, 25.97it/s]
```

In [56]:

```
# 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

X_test_tf_idf_W2V = []; # the tfidf-w2v for each sentence/review is stored ir
row=0;
for sent in tqdm(list_of_sentance_test): # 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_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
    X_test_tf_idf_W2V.append(sent_vec)
    row += 1
```

```
100%|██████████████████████████████████████████████████
██████| 10971/10971 [07:15<00:00, 25.22it/s]
```

In [57]:

```
from sklearn.preprocessing import StandardScaler

SS=StandardScaler(with_mean=False).fit(X_train_tf_idf_W2V)
x_train_tf_idf_W2V = SS.transform(X_train_tf_idf_W2V)
#X_CV_tf_idf_W2V= SS.transform(CV_tf_idf_W2V)
x_test_tf_idf_W2V=SS.transform(X_test_tf_idf_W2V)
```

In [58]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_curve, auc,roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier as RFC

#parametres
#depth=[i*2 for i in range(1,5)]
#n_estimators=[i*5 for i in range(1,20)]
p_grid_DT = {'max_depth': depth,'n_estimators':n_estimators}

#model
RF_clasifier=RFC(criterion='gini',min_samples_split=50,max_features='auto')

#gridsearch fitment
clf=GridSearchCV(RF_clasifier,scoring='roc_auc',iid=True,param_grid=p_grid_DT
clf.fit(x_train_tf_idf_W2V,y_train)

#train auc
train_auc= clf.cv_results_['mean_train_score']
train_auc_std_temp= clf.cv_results_['std_train_score']

#CV Auc
cv_auc= clf.cv_results_['mean_test_score']
cv_auc_std_temp= clf.cv_results_['std_test_score']

#Optimal parametres
params_tf_idf_W2V=clf.best_params_
auc_tf_idf_W2V=clf.best_score_
opt_depth_tf_idf_W2V=clf.best_params_ ['max_depth']
opt_n_estimators_tf_idf_W2V=clf.best_params_['n_estimators']

print("best params :",clf.best_params_)
print("best auc :",auc_tf_idf_W2V)
```

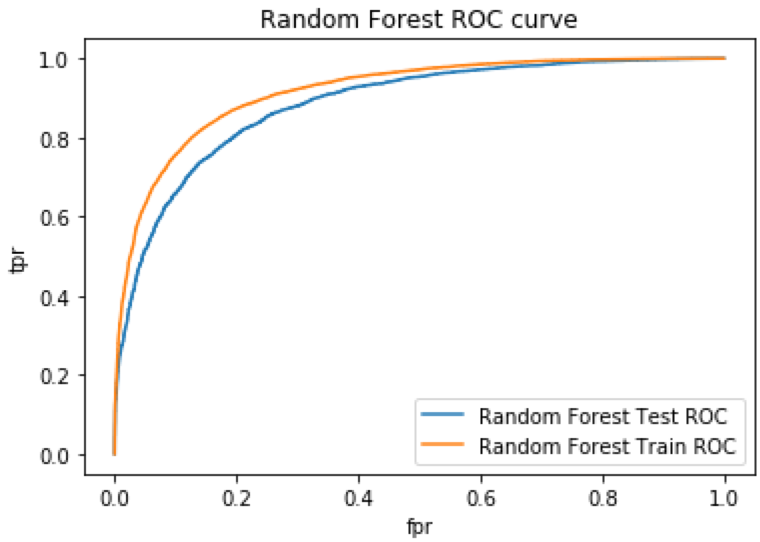
```
best params : {'max_depth': 8, 'n_estimators': 5}
best auc : 0.8842936032057034
```

In [59]:

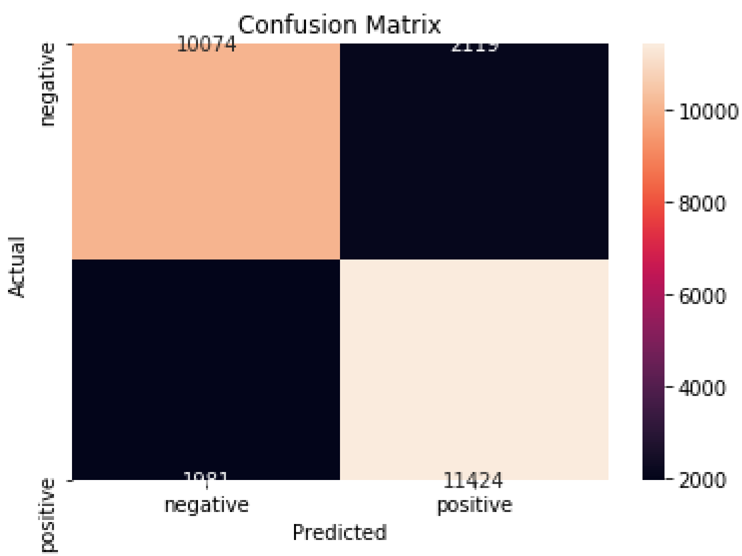
```
clf_tf_idf_W2V=RFC(n_estimators=opt_n_estimators_tf_idf_W2V,criterion='gini',
clf_tf_idf.fit(x_train_tf_idf_W2V,y_train)


#predict probabilities
y_Test_pred_proba = clf_tf_idf.predict_proba(x_test_tf_idf_W2V)[:,1]
y_Train_pred_proba = clf_tf_idf.predict_proba(x_train_tf_idf_W2V)[:,1]

#code for AUC
fpr_Test, tpr_Test, thresholds_Test = roc_curve(y_test, y_Test_pred_proba)
fpr_Train, tpr_Train, thresholds_train = roc_curve(y_train, y_Train_pred_prob
print("Test data AUC of random Forest  with Tf_idf_W2V Implentation  : ",roc_
AUC_tf_idf_W2V=roc_auc_score(y_test,y_Test_pred_proba)
print("Train data AUC of Random Forest with Tf_idf_W2V Implentation  : ",roc_


#generate plot
plt.plot(fpr_Test,tpr_Test, label='Random Forest Test ROC')
plt.plot(fpr_Train,tpr_Train, label='Random Forest Train ROC')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.legend()
plt.title('Random Forest ROC curve')
plt.show()


#confusion matrix of train data
best_t = find_best_threshold(thresholds_train, fpr_Train, tpr_Train)
print("Train confusion matrix")
conf_matrix=confusion_matrix(y_train, predict_with_best_t(y_Train_pred_proba,
print(confusion_matrix(y_train, predict_with_best_t(y_Train_pred_proba, best_

class_label = ['negative', 'positive']
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

#confusion matrix of test data
best_t = find_best_threshold(thresholds_Test, fpr_Test, tpr_Test)
print("Test confusion matrix")
conf_matrix=confusion_matrix(y_test, predict_with_best_t(y_Test_pred_proba, b
print(confusion_matrix(y_test, predict_with_best_t(y_Test_pred_proba, best_t)
class_label = ['negative', 'positive']
df_conf_matrix = pd.DataFrame(
    conf_matrix, index=class_label, columns=class_label)
sns.heatmap(df_conf_matrix, annot=True, fmt='d')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

Accuracy_tf_idf_W2V=clf_tf_idf.score(x_test_tf_idf_W2V, y_test)
```

```
print('Accuracy of Random Forest when Max_Depth={} and opt_n_estimators= {} i
```

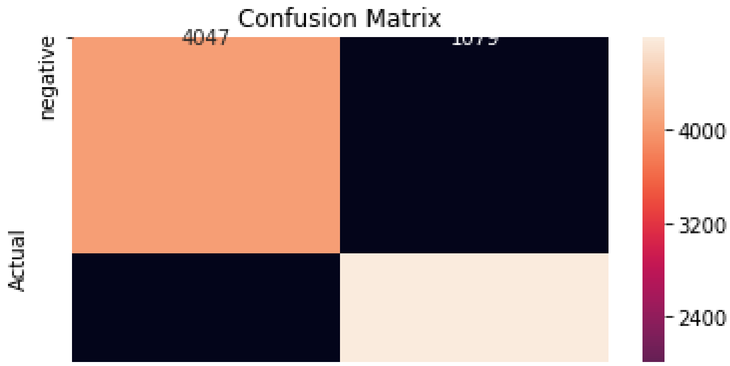
```
Test data AUC of random Forest  with Tf_idf_W2V Implentation  :  0.88402341
40714724
Train data AUC of Random Forest with Tf_idf_W2V Implentation  :  0.91685109
37464103
```




```
the maximum value of tpr*(1-fpr) 0.7041136259536835 for threshold 0.511
Train confusion matrix
[[10074  2119]
 [ 1981 11424]]
```




```
the maximum value of tpr*(1-fpr) 0.6471370396712846 for threshold 0.509
Test confusion matrix
[[4047 1079]
 [1054 4791]]
```




```
Accuracy of Random Forest when Max_Depth=8 and opt_n_estimators= 5 is 0.804
3934007838848
```

In [60]:

```
from prettytable import PrettyTable

x=PrettyTable()
x.field_names = ["Vectorizer", "Max depth","N_estimators", "AUC","Accuracy"]
x.add_row(["BOW", opt_depth_BOW,opt_n_estimators_BOW, auc_BOW,Accuracy_BOW])
x.add_row(["TF-IDF", opt_depth_tf_idf,opt_n_estimators_tf_idf,auc_tf_idf,Accu
x.add_row(["Avg-W2V", opt_depth_Avg_W2V,opt_n_estimators_Avg_W2V, auc_Avg_W2V
x.add_row(["TF-IDF_W2V", opt_depth_tf_idf_W2V,opt_n_estimators_tf_idf_W2V ,au
print(x)
```

```
+------------+-----------+--------------+--------------------+-------------
-------+
| Vectorizer | Max depth | N_estimators |        AUC         |       Accurac
y      |
+------------+-----------+--------------+--------------------+-------------
-------+
|    BOW     |     8     |      5       | 0.7943932162663988 | 0.6890894175
553732 |
|   TF-IDF   |     8     |      5       | 0.8164482337662151 | 0.7440525020
508614 |
|  Avg-W2V   |     8     |      5       |  0.90661667319736  | 0.8313736213
654179 |
| TF-IDF_W2V |     8     |      5       | 0.8842936032057034 | 0.8043934007
838848 |
+------------+-----------+--------------+--------------------+-------------
-------+
```

[6] Conclusions

- From above table we can observe that for max_depth 8 only we got the high accuracy

- For Avg_W2V and TF_IDF_W2V vectorizers we got max Accuracy
- Latency is less compared to K-NN
- For Avg_W2V vectorizer with max_depth : 8 and n_estimators : 5 we got the optimal solution i.e high Accuracy and high Auc

In [ ]:

```

```