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

from tqdm import tqdm
import os
```

In [2]:
```
# using SQLite Table to read data.
con = sqlite3.connect('database.sqlite')

# 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 data points
# you can change the number to any other number based on your computing power

# filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 50
0000""", con)
# for tsne assignment you can take 5k data points

filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 1500
00""", con)

# Give reviews with Score>3 a positive rating(1), and reviews with a score<3 a negative rating
(0).
def partition(x):
    if x < 3:
        return 0
    return 1

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

Number of data points in our data (150000, 10)

Out[2]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulNe
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1

In [3]:
```
display = pd.read_sql_query("""
SELECT UserId, ProductId, ProfileName, Time, Score, Text, COUNT(*)
FROM Reviews
GROUP BY UserId
HAVING COUNT(*)>1
""", con)
```

In [4]:
```
print(display.shape)
display.head()
```

(80668, 7)

Out[4]:

	UserId	ProductId	ProfileName	Time	Score	Text	COU
0	#oc-R115TNMSPFT9I7	B007Y59HVM	Breyton	1331510400	2	Overall its just OK when considering the price...	2
1	#oc-R11D9D7SHXIJB9	B005HG9ET0	Louis E. Emory "hoppy"	1342396800	5	My wife has recurring extreme muscle spasms, u...	3
2	#oc-R11DNU2NBKQ23Z	B007Y59HVM	Kim Cieszykowski	1348531200	1	This coffee is horrible and unfortunately not ...	2
3	#oc-R11O5J5ZVQE25C	B005HG9ET0	Penguin Chick	1346889600	5	This will be the bottle that you grab from the...	3
4	#oc-R12KPBODL2B5ZD	B007OSBE1U	Christopher P. Presta	1348617600	1	I didnt like this coffee. Instead of telling y...	2

In [5]:
```
display[display['UserId']=='AZY10LLTJ71NX']
```

Out[5]:

	UserId	ProductId	ProfileName	Time	Score	Text
80638	AZY10LLTJ71NX	B006P7E5ZI	undertheshrine "undertheshrine"	1334707200	5	I was recommended to try green tea extract to ...

In [6]:
```
display['COUNT(*)'].sum()
```

Out[6]: 393063

[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 [7]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND UserId="AR5J8UI46CURR"
ORDER BY ProductID
""", con)
display.head()
```

Out[7]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	Helpfuli
0	78445	B000HDL1RQ	AR5J8UI46CURR	Geetha Krishnan	2	2
1	138317	B000HDOPYC	AR5J8UI46CURR	Geetha Krishnan	2	2
2	138277	B000HDOPYM	AR5J8UI46CURR	Geetha Krishnan	2	2
3	73791	B000HDOPZG	AR5J8UI46CURR	Geetha Krishnan	2	2
4	155049	B000PAQ75C	AR5J8UI46CURR	Geetha Krishnan	2	2

As it can be seen above that same user has multiple reviews with same values for HelpfulnessNumerator, HelpfulnessDenominator, Score, Time, Summary and Text and on doing analysis it was found that

ProductId=B000HDOPZG was Loacker Quadratini Vanilla Wafer Cookies, 8.82-Ounce Packages (Pack of 8)

ProductId=B000HDL1RQ was Loacker Quadratini Lemon Wafer Cookies, 8.82-Ounce Packages (Pack of 8) and so on

It was inferred after analysis that reviews with same parameters other than ProductId belonged to the same product just having different flavour or quantity. Hence in order to reduce redundancy it was decided to eliminate the rows having same parameters.

The method used for the same was that we first sort the data according to ProductId and then just keep the first similar product review and delelte the others. for eg. in the above just the review for ProductId=B000HDL1RQ remains. This method ensures that there is only one representative for each product and deduplication without sorting would lead to possibility of different representatives still existing for the same product.

In [8]:
```
#Sorting data according to ProductId in ascending order
sorted_data=filtered_data.sort_values('ProductId', axis=0, ascending=True, inplace=False, kind
='quicksort', na_position='last')
```

In [9]:
```
#Deduplication of entries
final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', in
place=False)
final.shape
```

Out[9]: (126359, 10)

In [10]:
```
#Checking to see how much % of data still remains
(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100
```

Out[10]: 84.23933333333333

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 [11]:
```
display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[11]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulI
0	64422	B000MIDROQ	A161DK06JJMCYF	J. E. Stephens "Jeanne"	3	1
1	44737	B001EQ55RW	A2V0I904FH7ABY	Ram	3	2

In [12]:
```
final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

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

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

(126357, 10)

Out[13]:
```
1    106326
0     20031
Name: Score, dtype: int64
```

[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]:
```
# printing some random reviews
sent_0 = final['Text'].values[0]
print(sent_0)
print("="*50)

sent_1000 = final['Text'].values[1000]
print(sent_1000)
print("="*50)

sent_1500 = final['Text'].values[1500]
print(sent_1500)
print("="*50)

sent_4900 = final['Text'].values[4900]
print(sent_4900)
print("="*50)
```

```
I grew up reading these Sendak books, and watching the Really Rosie movie that incorporates t
hem, and love them. My son loves them too. I do however, miss the hard cover version. The pap
erbacks seem kind of flimsy and it takes two hands to keep the pages open.
==================================================
Its about time Spanish products started getting their due.. The most famous (rightly so) Spanish
cheese, Manchego, is world class, and is really tough to beat.  Try some with some fig cake, or s
ome quince paste, or drizzled with olive oil garnished with rosemary..  Serve with a fino sherry, m
anzanilla, or any number of red wines (depending on age of cheese), and you are guaranteed a
winning combination..  Sliced, melted over a great burger, with a roasted red pepper and a heart
y glass of earthy zinfandel = heavenly joy..  Cube and marinate in Spanish olive oil is also a trea
t... I have had no problems with iGourmet so far, so I can't comment on their customer service -
all of my orders have been shipped quickly (1-3 business days), without issue.  Their Manchego i
s much better than I can get at the local gourmet grocery stores.  They have great specials and
offers, so I have to count myself as a big iGourmet fan.  Regardless, Manchego cheese is so go
od, you'll undoubtedly be back for more.. and more..  and more.
==================================================
I love this stuff.  I nuke a mug a milk until it's very hot, drop in 2 of the triangles, stir until the choc
olate melts, then froth it with my Aerolatte.  Simple and tasty.
==================================================
There's nothing like the scent of real lavender! Just a whiff smells so good. Besides enjoying the
fragrance, I've only used a small amount a few times to make some tea but it is wonderful! Can't
wait to try it in ice cubes, add a pinch on top of a dessert, throw a little into homemade laundry s
oap, or whatever else comes to mind. Would definitely buy more!
==================================================
```

In [15]:
```
# remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_0 = re.sub(r"http\S+", "", sent_0)
sent_1000 = re.sub(r"http\S+", "", sent_1000)
sent_150 = re.sub(r"http\S+", "", sent_1500)
sent_4900 = re.sub(r"http\S+", "", sent_4900)

print(sent_0)
```

```
I grew up reading these Sendak books, and watching the Really Rosie movie that incorporates t
hem, and love them. My son loves them too. I do however, miss the hard cover version. The pap
erbacks seem kind of flimsy and it takes two hands to keep the pages open.
```

In [16]:
```
# https://stackoverflow.com/questions/16206380/python-beautifulsoup-how-to-remove-all-tags-from-an-element
from bs4 import BeautifulSoup

soup = BeautifulSoup(sent_0, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1000, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_1500, 'lxml')
text = soup.get_text()
print(text)
print("="*50)

soup = BeautifulSoup(sent_4900, 'lxml')
text = soup.get_text()
print(text)
```

```
I grew up reading these Sendak books, and watching the Really Rosie movie that incorporates t
hem, and love them. My son loves them too. I do however, miss the hard cover version. The pap
erbacks seem kind of flimsy and it takes two hands to keep the pages open.
==================================================
Its about time Spanish products started getting their due.. The most famous (rightly so) Spanish
cheese, Manchego, is world class, and is really tough to beat.  Try some with some fig cake, or s
ome quince paste, or drizzled with olive oil garnished with rosemary..  Serve with a fino sherry, m
anzanilla, or any number of red wines (depending on age of cheese), and you are guaranteed a
winning combination..  Sliced, melted over a great burger, with a roasted red pepper and a heart
y glass of earthy zinfandel = heavenly joy..  Cube and marinate in Spanish olive oil is also a trea
t... I have had no problems with iGourmet so far, so I can't comment on their customer service -
all of my orders have been shipped quickly (1-3 business days), without issue.  Their Manchego i
s much better than I can get at the local gourmet grocery stores.  They have great specials and
offers, so I have to count myself as a big iGourmet fan.  Regardless, Manchego cheese is so go
od, you'll undoubtedly be back for more.. and more..  and more.
==================================================
I love this stuff.  I nuke a mug a milk until it's very hot, drop in 2 of the triangles, stir until the choc
olate melts, then froth it with my Aerolatte.  Simple and tasty.
==================================================
There's nothing like the scent of real lavender! Just a whiff smells so good. Besides enjoying the
fragrance, I've only used a small amount a few times to make some tea but it is wonderful! Can't
wait to try it in ice cubes, add a pinch on top of a dessert, throw a little into homemade laundry s
oap, or whatever else comes to mind. Would definitely buy more!
```

In [17]:
```
# 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 [18]:
```
sent_1500 = decontracted(sent_1500)
print(sent_1500)
print("="*50)
```

```
I love this stuff.  I nuke a mug a milk until it is very hot, drop in 2 of the triangles, stir until the cho
colate melts, then froth it with my Aerolatte.  Simple and tasty.
==================================================
```

In [19]:
```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

```
I grew up reading these Sendak books, and watching the Really Rosie movie that incorporates t
hem, and love them. My son loves them too. I do however, miss the hard cover version. The pap
erbacks seem kind of flimsy and it takes two hands to keep the pages open.
```

In [20]:
```
#remove spacial character: https://stackoverflow.com/a/5843547/4084039
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

```
I love this stuff I nuke a mug a milk until it is very hot drop in 2 of the triangles stir until the chocol
ate melts then froth it with my Aerolatte Simple and tasty
```

In [21]:
```
# 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 1st step

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

In [22]:
```
# Combining all the above stundents
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 = 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%|[redacted]| 126357/126357 [00:50<00:00, 2504.84it/s]

In [0]:
```
preprocessed_reviews[1500]
```

Out[0]:
```
'wow far two two star reviews one obviously no idea ordering wants crispy cookies hey sorry revi
ews nobody good beyond reminding us look ordering chocolate oatmeal cookies not like combin
ation not order type cookie find combo quite nice really oatmeal sort calms rich chocolate flavor
gives cookie sort coconut type consistency let also remember tastes differ given opinion soft che
wy cookies advertised not crispy cookies blurb would say crispy rather chewy happen like raw co
okie dough however not see taste like raw cookie dough soft however confusion yes stick togeth
er soft cookies tend not individually wrapped would add cost oh yeah chocolate chip cookies ten
d somewhat sweet want something hard crisp suggest nabiso ginger snaps want cookie soft che
wy tastes like combination chocolate oatmeal give try place second order'
```

In [128]:
```
#replacing the reviews with preprocessed reviews
final['Text']=preprocessed_reviews

fd = final.sample(n=100000)

# Sorting based on time
fd['Time'] = pd.to_datetime(fd['Time'], origin='unix', unit='s')
fd= fd.sort_values('Time')
fd.shape
```

Out[128]: (100000, 10)

In [129]:
```
#splitting data into X and Y
Y= fd.Score
X=fd.Text
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=False)# this is for time series split

print('Before Vectorization')
print('Train--->',X_train.shape, y_train.shape)
print('Test---->',X_test.shape, y_test.shape)
```

```
Before Vectorization
Train---> (67000,) (67000,)
Test----> (33000,) (33000,)
```

[4] Featurization

[4.1] BAG OF WORDS

In [130]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(ngram_range=(1,2),min_df=10, max_features=10000)
vectorizer.fit(X_train) # fit has to happen only on train data

# we use the fitted CountVectorizer to convert the text to vector
X_train_bow = vectorizer.transform(X_train)
X_test_bow = vectorizer.transform(X_test)

print("After vectorizations")
print(X_train_bow.shape, y_train.shape)
print(X_test_bow.shape, y_test.shape)

scalar = StandardScaler(with_mean=False)
X_train_bow = scalar.fit_transform(X_train_bow)
X_test_bow= scalar.transform(X_test_bow)
```

```
After vectorizations
(67000, 10000) (67000,)
(33000, 10000) (33000,)
```

[4.2] Bi-Grams and n-Grams.

In [131]:
```
#bi-gram, tri-gram and n-gram

#removing stop words like "not" should be avoided before building n-grams
# count_vect = CountVectorizer(ngram_range=(1,2))
# please do read the CountVectorizer documentation http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

# you can choose these numebrs min_df=10, max_features=5000, of your choice
count_vect = CountVectorizer(ngram_range=(1,2), min_df=10, max_features=5000)
final_bigram_counts = count_vect.fit_transform(preprocessed_reviews)
print("the type of count vectorizer ",type(final_bigram_counts))
print("the shape of out text BOW vectorizer ",final_bigram_counts.get_shape())
print("the number of unique words including both unigrams and bigrams ", final_bigram_counts.get_shape()[1])
```

```
the type of count vectorizer  <class 'scipy.sparse.csr.csr_matrix'>
the shape of out text BOW vectorizer  (126357, 5000)
the number of unique words including both unigrams and bigrams  5000
```

[4.3] TF-IDF

In [132]:
```
#splitting data into X and Y
Y= fd.Score
X=fd.Text
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=False)# this is for t
ime series split

print('Before Vectorization')
print('Train--->',X_train.shape, y_train.shape)
print('Test---->',X_test.shape, y_test.shape)

tf_idf_vect = TfidfVectorizer(ngram_range=(1,2), min_df = 10, max_features=10000)
tf_idf_vect.fit(X_train)

X_train_tfidf = tf_idf_vect.transform(X_train)
X_test_tfidf = tf_idf_vect.transform(X_test)

scalar = StandardScaler(with_mean=False)
X_train_tfidf = scalar.fit_transform(X_train_tfidf)
X_test_tfidf= scalar.transform(X_test_tfidf)
print("After vectorizations")
print(X_train_tfidf.shape, y_train.shape)
print(X_test_tfidf.shape, y_test.shape)
```

```
Before Vectorization
Train---> (67000,) (67000,)
Test----> (33000,) (33000,)
After vectorizations
(67000, 10000) (67000,)
(33000, 10000) (33000,)
```

[4.4] Word2Vec

In [133]:
```
#splitting data into X and Y
Y= fd.Score
X=fd.Text
#splitting data into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=False)
#before vectorization
print('Train Data' ,X_train.shape, y_train.shape)
print('Test Data',X_test.shape, y_test.shape)
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
# this line of code trains your w2v model on the give list of sentances
w2v_model=Word2Vec(list_of_sentance_train,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])
```

```
Train Data (67000,) (67000,)
Test Data (33000,) (33000,)
number of words that occured minimum 5 times  15424
sample words  ['guiltless', 'irritation', 'steamer', 'magnifying', 'sans', 'uh', 'size', 'pray', 'calamari', 's
uggestion', 'discernible', 'sky', 'surge', 'raisin', 'receiver', 'whisper', 'pint', 'fair', 'tracking', 'honeyde
w', 'mugs', 'tastefully', 'association', 'cleaned', 'ortho', 'recognition', 'relief', 'contributing', 'indicato
r', 'suet', 'accepted', 'starch', 'chocoholics', 'grail', 'lesions', 'derivative', 'mei', 'delicacy', 'dropping',
'mood', 'lactaid', 'cheesier', 'bali', 'squirrels', 'retains', 'turbinado', 'uneducated', 'piece', 'unappetizi
ng', 'possibility']
```

[4.4.1] Converting text into vectors using Avg W2V, TFIDF-W2V

[4.4.1.1] Avg W2v

converting training data

In [134]:
```
#splitting data into X and Y
Y= fd.Score
X=fd.Text
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=False)# this is for t
ime series split
from tqdm import tqdm
import numpy as np
i=0
list_of_sentance_train=[]
for sentance in tqdm(X_train):
    list_of_sentance_train.append(sentance.split())
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change th
is to 300 if you use google's w2v
    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_train.append(sent_vec)
sent_vectors_train = np.array(sent_vectors_train)
print(sent_vectors_train.shape)
print(sent_vectors_train[0])
```

```
100%|██████████| 67000/67000 [00:00<00:00, 69139.94it/s]
100%|██████████| 67000/67000 [20:27<00:00, 54.58it/s]

(67000, 50)
[ 0.4199983  -0.08895202 -0.08477899 -0.56357564 -0.32572078 -0.23358604
 -0.09379596 -0.16820438  0.11279046  0.19832524 -0.29224967 -0.56195543
  0.16540707 -0.02921105 -0.38846539  0.02124481  0.20096504 -0.1320331
 -0.05069149  0.12026193 -0.36959341  0.19822839 -0.32310567  0.28105706
 -0.10034016  0.18261783 -0.50143642  0.07751147 -0.59836154 -0.29924685
 -0.59661647  0.19420152  0.39212492  0.13561657  0.23312985  0.02388437
  0.12648212 -0.31596941 -0.15555687 -0.4174979   0.38247245  0.1623963
  0.38501526 -0.34789802 -0.18294796 -0.22034915 -0.07360141 -0.02886957
 -0.02624704 -0.04207289]
```

converting test data

In [135]:
```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
# average Word2Vec
# compute average word2vec for each review.
sent_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change th
is to 300 if you use google's w2v
    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_test.append(sent_vec)
sent_vectors_test = np.array(sent_vectors_test)
print(sent_vectors_test.shape)
print(sent_vectors_test[0])
```

```
100%|██████████| 33000/33000 [09:41<00:00, 56.73it/s]

(33000, 50)
[-0.57494164 -0.63889936 -0.26762456 -0.48066144  0.06375151 -0.08131631
 -0.2271974  -0.31895295  0.47978469  0.53948555  0.02120619 -1.05823928
 -0.27055266  0.08885294  0.46607984 -0.04254009 -0.21140016  0.17351155
 -0.39816678 -0.06121482 -0.18794716 -0.81092872 -0.09019224  0.82531787
 -0.5943539   0.00108349 -0.2525971   0.71909119 -0.21815007 -0.81342914
 -0.13206631  0.38473471  0.85218717  0.4071799   0.32462158  0.52186149
  0.02717864 -0.00432699  0.12636176 -0.02860289  0.69696964 -0.07023695
  0.82638956 -0.44082381 -0.23821987  0.04918594  0.34332362 -0.3217108
  0.50152153  0.2180292 ]
```

[4.4.1.2] TFIDF weighted W2v

In [136]:
```
#splitting data into X and Y
Y= fd.Score
X=fd.Text
# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=False)# this is for t
ime series split
#before vectorization
print('Train Data' ,X_train.shape, y_train.shape)
print('Test Data',X_test.shape, y_test.shape)
model1 = TfidfVectorizer()
tf_idf_train_w2v= model1.fit_transform(X_train)
tf_idf_test_w2v= model1.transform(X_test)
# we are converting a dictionary with word as a key, and the idf as a value
dictionary1 = dict(zip(model1.get_feature_names(), list(model1.idf_)))
print(tf_idf_train_w2v.shape, tf_idf_test_w2v.shape)
```

```
Train Data (67000,) (67000,)
Test Data (33000,) (33000,)
(67000, 48097) (33000, 48097)
```

converting train data

In [137]:
```
i=0
list_of_sentance_train=[]
for sentance in X_train:
    list_of_sentance_train.append(sentance.split())
# TF-IDF weighted Word2Vec
tfidf_feat_train = model1.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_train = []; # the tfidf-w2v for each sentence/review is stored in this list
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_train:
            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 = dictionary1[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_train.append(sent_vec)
    row += 1
tfidf_sent_vectors_train = np.array(tfidf_sent_vectors_train)
```

```
100%|[redacted]| 67000/67000 [1:26:12<00:00, 12.95it/s]
```

converting test data

In [138]:
```
i=0
list_of_sentance_test=[]
for sentance in X_test:
    list_of_sentance_test.append(sentance.split())
# TF-IDF weighted Word2Vec
tfidf_feat_test = model1.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_test = []; # the tfidf-w2v for each sentence/review is stored in this list
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_test:
            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 = dictionary1[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_test.append(sent_vec)
    row += 1
tfidf_sent_vectors_test = np.array(tfidf_sent_vectors_test)
```

```
100%|██████████| 33000/33000 [43:49<00:00, 12.55it/s]
```

In [139]:
```
print(tfidf_sent_vectors_train.shape,tfidf_sent_vectors_test.shape)
```

```
(67000, 50) (33000, 50)
```

[5] Assignment 9: Random Forests

1. **Apply Random Forests & GBDT on these feature sets**

   - SET 1:Review text, preprocessed one converted into vectors using (BOW)
   - SET 2:Review text, preprocessed one converted into vectors using (TFIDF)
   - SET 3:Review text, preprocessed one converted into vectors using (AVG W2v)
   - SET 4:Review text, preprocessed one converted into vectors using (TFIDF W2v)

2. **The hyper paramter tuning (Consider two hyperparameters: n_estimators & max_depth)**

   - Find the best hyper parameter which will give the maximum AUC (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/receiver-operating-characteristic-curve-roc-curve-and-auc-1/) value
   - Find the best hyper paramter using k-fold cross validation or simple cross validation data
   - Use gridsearch cv or randomsearch cv or you can also write your own for loops to do this task of hyperparameter tuning

3. **Feature importance**

   - Get top 20 important features and represent them in a word cloud. Do this for BOW & TFIDF.

4. **Feature engineering**

   - To increase the performance of your model, you can also experiment with with feature engineering like :
     - Taking length of reviews as another feature.
     - Considering some features from review summary as well.

5. **Representation of results**

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure

     with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*

(or)

   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure

     seaborn heat maps (https://seaborn.pydata.org/generated/seaborn.heatmap.html) with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
   - You choose either of the plotting techniques out of 3d plot or heat map
   - Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.

   - Along with plotting ROC curve, you need to print the confusion matrix (https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/confusion-matrix-

tpr-fpr-fnr-tnr-1/) with predicted and original labels of test data points. Please visualize your confusion matrices using seaborn heatmaps.

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)
(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

6. **Conclusion** (https://seaborn.pydata.org/generated/seaborn.heatmap.html)

(https://seaborn.pydata.org/generated/seaborn.heatmap.html)

- You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library (https://seaborn.pydata.org/generated/seaborn.heatmap.html) link (http://zetcode.com/python/prettytable/)

Note: Data Leakage

1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.
2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.
4. For more details please go through this link. (https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf)

[5.1] Applying RF

Function for tuning hyperparameter for RandomForest Classifier

In [140]:
```
def best_RF(Xtrain,ytrain):
    """
    Returns : best depth and best number of models
    ---
    Input : Training dataset
    """
    # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
    import numpy as np
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.metrics import roc_auc_score
    from sklearn.model_selection import GridSearchCV
    import matplotlib.pyplot as plt
    max_depth = [10, 50, 100, 200 ,300, 500]
    n_models = [40,50,70,90]
    parameters = {'max_depth': [10, 50, 100, 200 ,300, 500], 'n_estimators' :[40,50,70,90]}
    clftree = RandomForestClassifier(class_weight='balanced')
    clf = GridSearchCV(clftree,parameters,cv=4, scoring='roc_auc',return_train_score=True)
    clf.fit(X_train_bow,y_train)
    tab = pd.DataFrame(clf.cv_results_)
    import seaborn as sns
    plt.figure(figsize=(15, 8))
    max_scores = tab.groupby(['param_max_depth','param_n_estimators']).max()
    max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
    sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g')
    plt.title('Grid Search CV Score on Train and Test Data')
    plt.show()
    best_parameter= clf.best_params_
    print('Best depth: ', clf.best_estimator_.max_depth)
    print('Best n estimators: ', clf.best_estimator_.n_estimators)
    return clf.best_estimator_.max_depth, clf.best_estimator_.n_estimators
```

Function for fitting the model for best hyperparameter

In [141]:
```
def final_RF( Xtrain ,ytrain ,Xtest ,ytest , param1 , param2):
    """
    Returns : threshold values, False positive rate , True positive rate and the trained model
    ---
    Input : Train dataset, Test Dataset , best parameters
    """
    # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
    from sklearn.metrics import roc_curve, auc
    import matplotlib.pyplot as plt
    from sklearn.ensemble import RandomForestClassifier

    clftre = RandomForestClassifier(class_weight='balanced',n_estimators=param1, max_depth=param2)
    clftre.fit(Xtrain,ytrain)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_fpr, train_tpr, thresholds = roc_curve(y_train, clftre.predict_proba(Xtrain)[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(y_test, clftre.predict_proba(Xtest)[:,1])

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.grid(True)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.show()
    return thresholds ,train_fpr, train_tpr, clftre
```

Function for picking the best threshold that will give the least fpr

In [142]:
```
# we are writing our own function for predict, with defined thresholds
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshold, fpr, tpr):
    """
    Input: Threshold values , false positive rate and true positive rate of final model
    ---
    Return: best threshold value for confusion matrix
    """
    import numpy as np
    t = threshold[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t,3))
    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
```

Function for confusion matrix for best threshold values

In [143]:
```
def conf_matrix(Xtrain,ytrain,Xtest,ytest,best_t,model):
    """
    Input : Train dataset, Test dataset, best threshold values for confusion matrix and final model
    ---
    Return: Confusion matrix of train and test dataset
    """
    #https://stackoverflow.com/questions/19233771/sklearn-plot-confusion-matrix-with-labels
    import seaborn as sns
    %matplotlib inline
    sns.set()

    y_train_pred = model.predict_proba(Xtrain)[:,1]
    y_test_pred = model.predict_proba(Xtest)[:,1]
    from sklearn.metrics import confusion_matrix
    print('Train Data')
    cm = confusion_matrix(y_train, predict_with_best_t(y_train_pred, best_t))
    cl = ['Negative', 'Positive']
    confusion_df = pd.DataFrame(cm, index=cl, columns=cl)
    sns.heatmap(confusion_df, annot=True, cbar = False,fmt = 'd'); #annot=True to annotate cells
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
    print('='*50)
    print('Test Data')
    cm_test= (confusion_matrix(y_test, predict_with_best_t(y_test_pred, best_t)))
    confusion_df_test = pd.DataFrame(cm_test, index=cl, columns=cl)
    sns.heatmap(confusion_df_test, annot=True, fmt = 'd'); #annot=True to annotate cells
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

[5.1.1] Applying Random Forests on BOW, SET 1

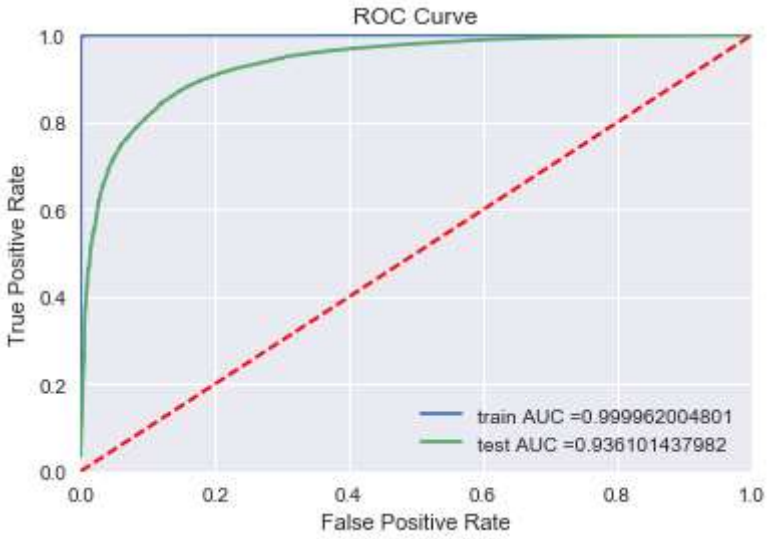
Hyperparameter tuning

In [144]:
```
%%time
best_depth , n_models = best_RF(X_train_bow,y_train)
```

Best depth: 200
Best n estimators: 90
Wall time: 1h 17min 43s

fitting the model with best hyperparameter

In [145]:
```
thr_bow,bow_fpr , bow_tpr, bow_model =  final_RF(X_train_bow,y_train,X_test_bow,y_test,n_models,best_depth)
```

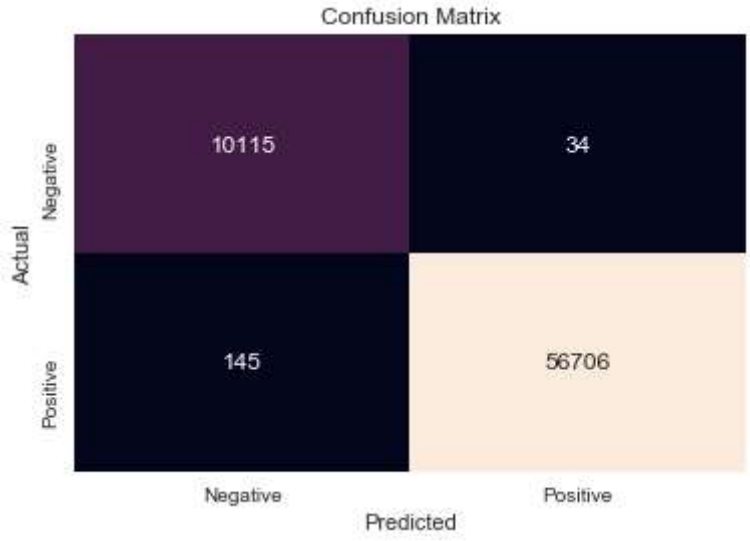


[5.1.2] Wordcloud of top 20 important features from SET 1

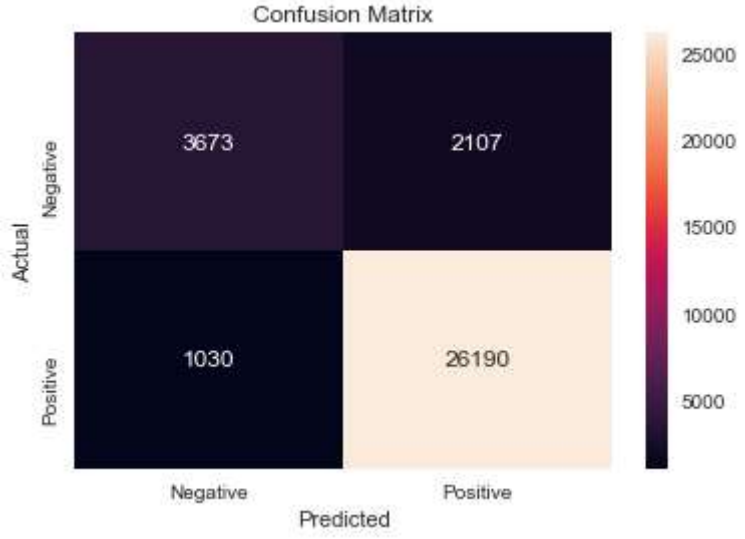
In [146]:
```
ft = vectorizer.get_feature_names()
string=''
fi=bow_model.feature_importances_
features=np.argsort(fi)[::-1]
for i in features[0:20]:
    string+=ft[i]
    string+=' '


from wordcloud import WordCloud
wordcloud = WordCloud(background_color="red").generate(string)
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

In [147]:
```
best_t_bow = find_best_threshold(thr_bow, bow_fpr, bow_tpr)
conf_matrix(X_train_bow,y_train,X_test_bow,y_test,best_t_bow,bow_model)
```

the maximum value of tpr*(1-fpr) 0.995165178798 for threshold 0.615
Train Data



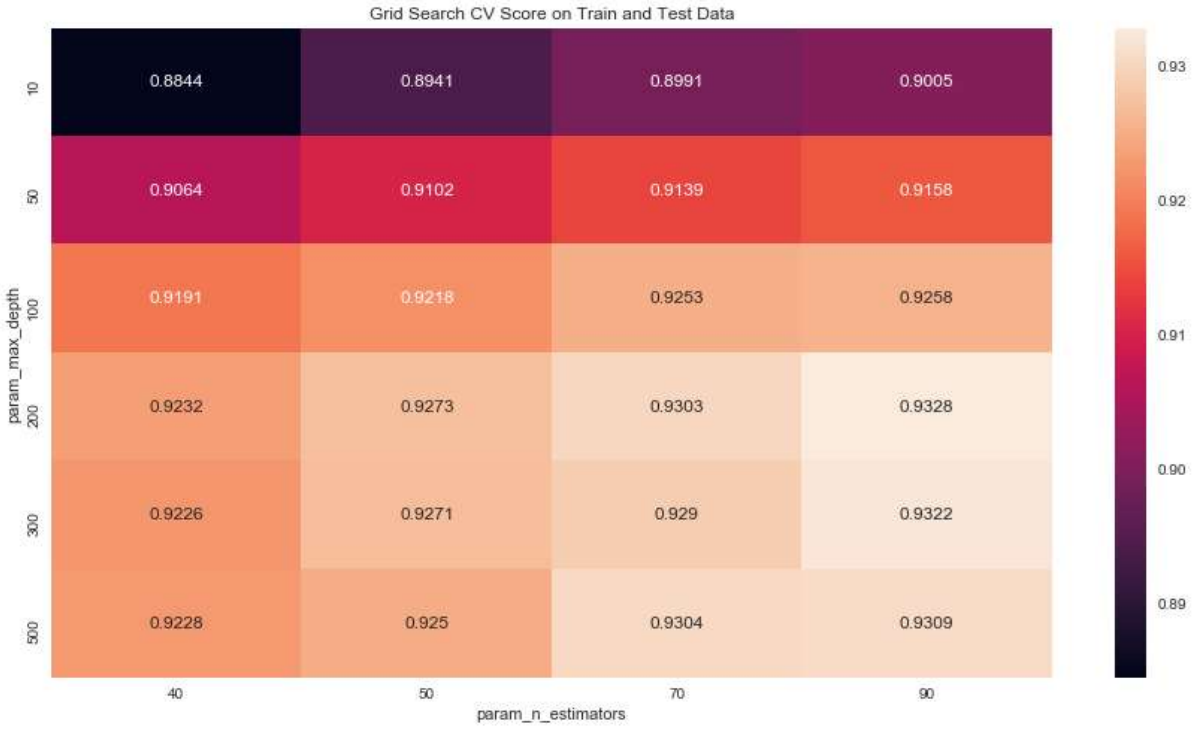
==========
Test Data



[5.1.3] Applying Random Forests on TFIDF, SET 2

Hyperaprameter Tuning

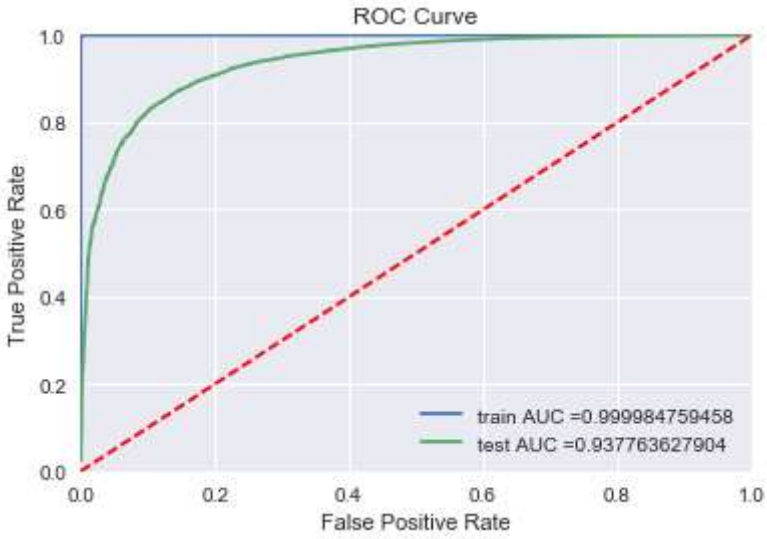
In [148]:
```
%%time
best_depth_tfidf, best_n_estimators = best_RF(X_train_tfidf,y_train)
```



Best depth: 200
Best n estimators: 90
Wall time: 1h 11min 14s

Training the model with best hyperparameter

In [149]:
```
th_tfidf, fpr_tfidf , tpr_tfidf , clftre_tfidf = final_RF(X_train_tfidf,y_train, X_test_tfidf, y_test,best_n
_estimators,best_depth_tfidf)
```



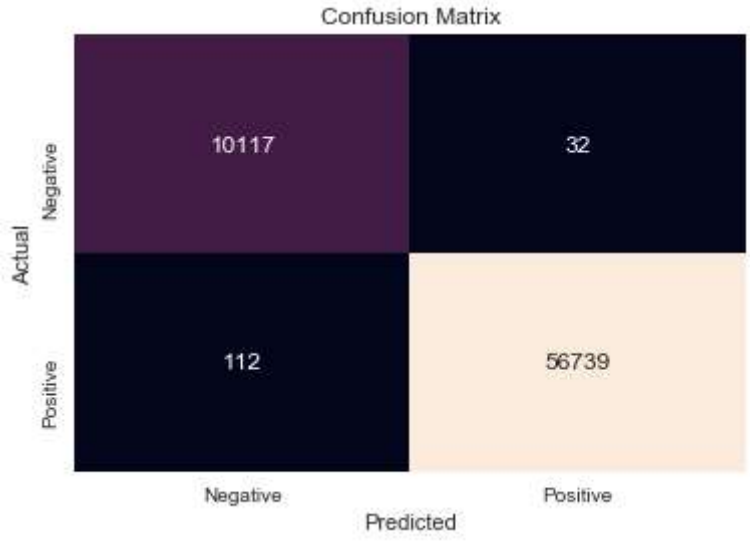
[5.1.4] Wordcloud of top 20 important features from SET 2

In [150]:
```
ft2 = tf_idf_vect.get_feature_names()
string2=''
fi2=clftre_tfidf.feature_importances_
features2=np.argsort(fi2)[::-1]
for i in features2[0:20]:
    string2+=ft[i]
    string2+=' '


from wordcloud import WordCloud
wordcloud = WordCloud(background_color="red").generate(string2)
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

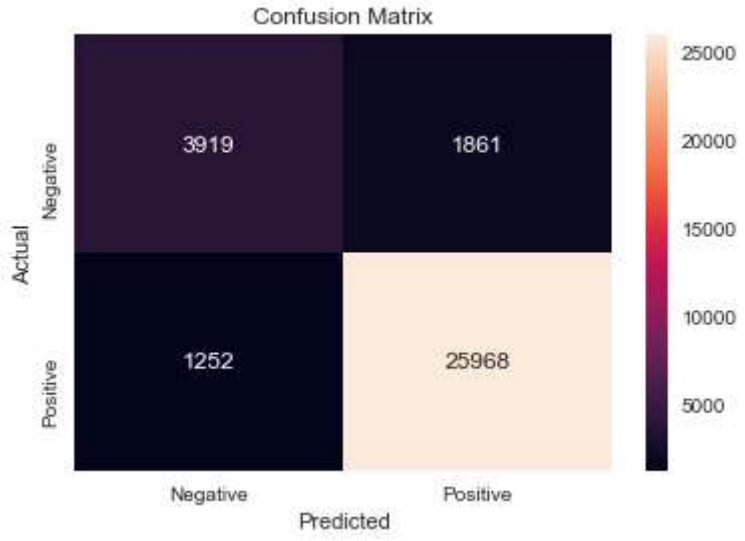
confusion matrix

In [151]:
```
best_t_tfidf = find_best_threshold(th_tfidf, fpr_tfidf, tpr_tfidf)
conf_matrix(X_train_tfidf,y_train,X_test_tfidf,y_test,best_t_tfidf,clftre_tfidf)
```

the maximum value of tpr*(1-fpr) 0.996106457955 for threshold 0.644
Train Data



==========
Test Data



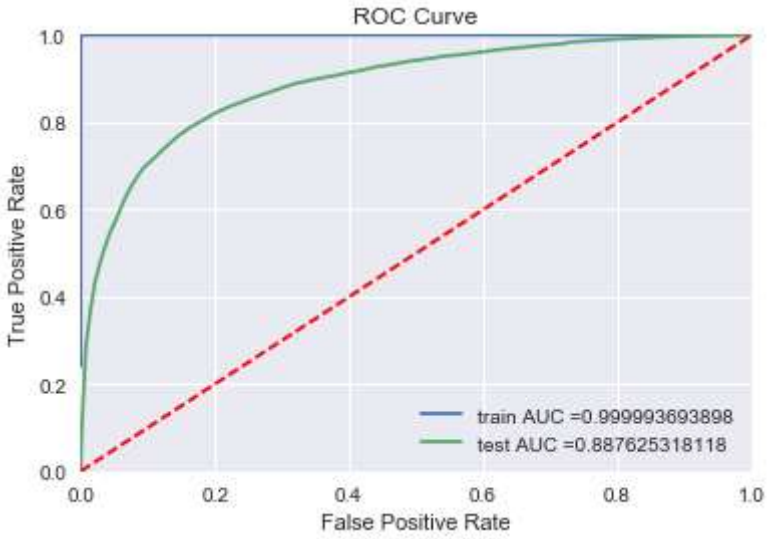
[5.1.5] Applying Random Forests on AVG W2V, SET 3

In [152]:
```
%%time
best_depth_avgw2v, best_n_estimaotr_avgw2v = best_RF(sent_vectors_train,y_train)
```

Best depth: 200
Best n estimators: 90
Wall time: 1h 11min 11s

Tuning the model with best hyperparameter

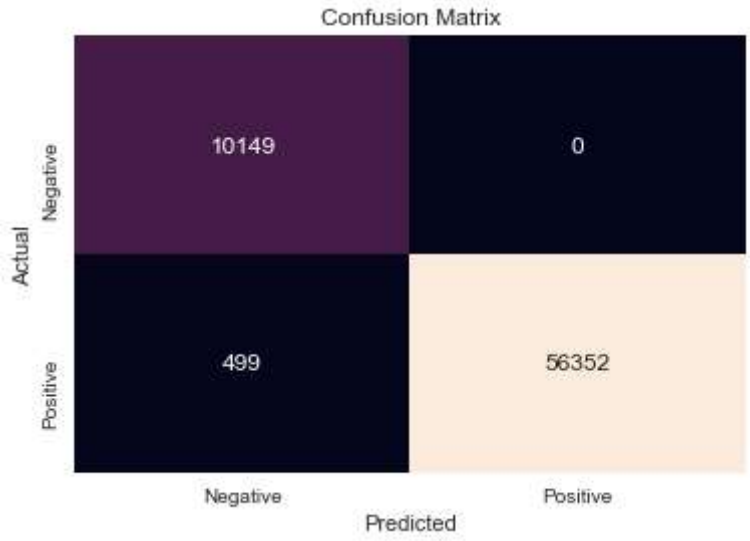
In [153]:
```
th_avgw2v, fpr_avgw2v , tpr_avgw2v , clftre_avgw2v = final_RF(sent_vectors_train,y_train, sent
_vectors_test, y_test,best_n_estimaotr_avgw2v
                                                    ,best_depth_avgw2v)
```



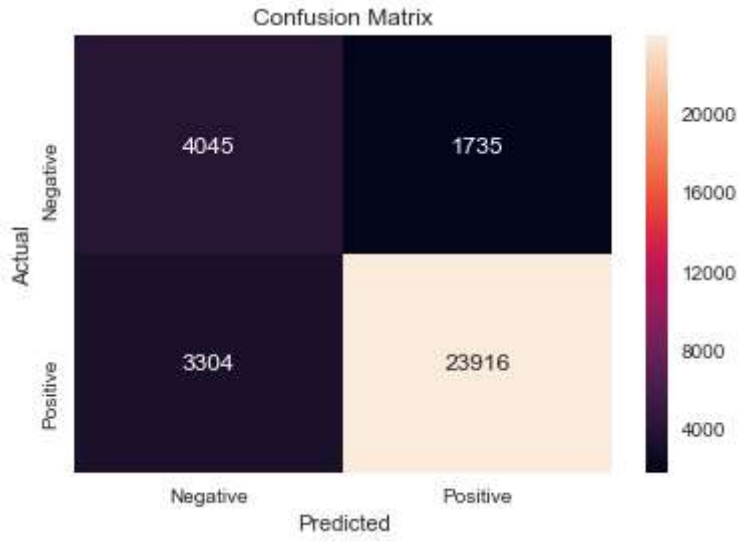
confusion matrix

In [154]:
```
best_t_2_avgw2v = find_best_threshold(th_avgw2v, fpr_avgw2v, tpr_avgw2v)
conf_matrix(sent_vectors_train,y_train,sent_vectors_test,y_test,best_t_2_avgw2v,clftre_avgw2v
)
```

the maximum value of tpr*(1-fpr) 0.997238395103 for threshold 0.757
Train Data



==========
Test Data



[5.1.6] Applying Random Forests on TFIDF W2V, SET 4

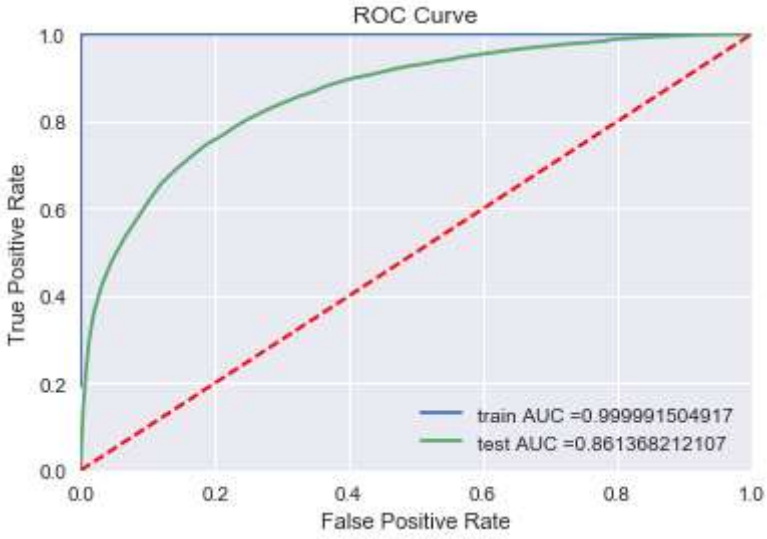
Hyperparameter tuning

In [155]:
```
%%time
best_depth_tfidfavgw2v, best_n_estimator_tfidfavgw2v = best_RF(tfidf_sent_vectors_train,y_train)
```

Best depth: 200
Best n estimators: 90
Wall time: 1h 11min 18s

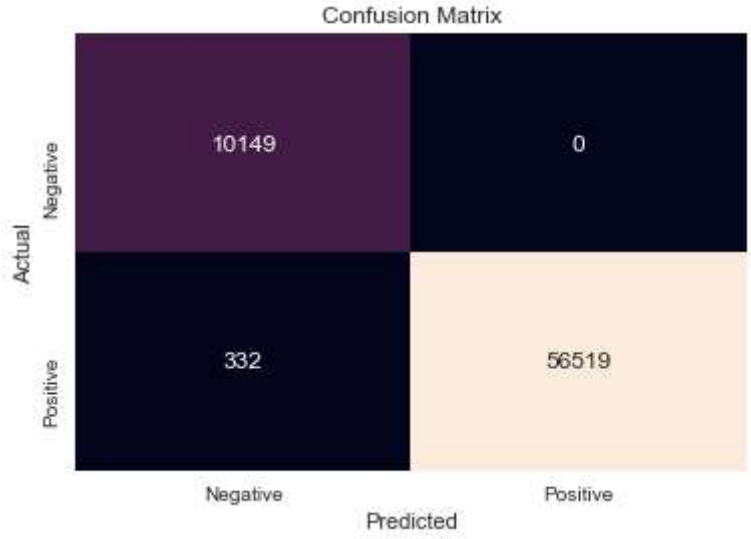
Training the model with best hyperparameter

In [156]:
```
th_tfidfavgw2v, fpr_tfidfavgw2v , tpr_tfidfavgw2v , clftre_tfidfavgw2v = final_RF(tfidf_sent_vectors_train,y_train, tfidf_sent_vectors_test, y_test,best_n_estimator_tfidfavgw2v
                                                                                  ,best_depth_tfidfavgw2v)
```

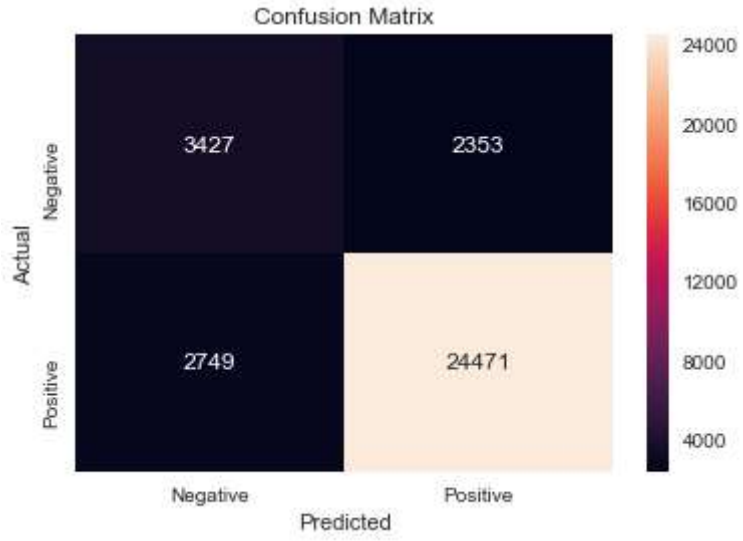


In [157]:
```
best_t_2_tfidfavgw2v = find_best_threshold(th_tfidfavgw2v, fpr_tfidfavgw2v, tpr_tfidfavgw2v)
conf_matrix(tfidf_sent_vectors_train,y_train,tfidf_sent_vectors_test,y_test,best_t_2_tfidfavgw2v,
clftre_tfidfavgw2v)
```

```
the maximum value of tpr*(1-fpr) 0.997238395103 for threshold 0.735
Train Data
```




```
==================================================
Test Data
```



[5.2] Applying GBDT using XGBOOST

Function for hyperparameter tuning of GBDT using XGBOOST

In [158]:
```
def best_GBDT(Xtrain,ytrain):
    # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
    import numpy as np
    from xgboost import XGBClassifier
    from sklearn.metrics import roc_auc_score
    from sklearn.model_selection import GridSearchCV
    import matplotlib.pyplot as plt
    max_depth = [10, 50, 100, 200 ,300, 500]
    n_models = [40,50,70,90]
    parameters = {'max_depth': [10, 50, 100, 200 ,300, 500], 'n_estimators' :[40,50,70,90]}
    clftree = XGBClassifier(booster='gbtree')
    clf = GridSearchCV(clftree,parameters,cv=4, scoring='roc_auc',return_train_score=True)
    clf.fit(X_train_bow,y_train)
    tab = pd.DataFrame(clf.cv_results_)
    import seaborn as sns
    plt.figure(figsize=(15, 8))
    max_scores = tab.groupby(['param_max_depth','param_n_estimators']).max()
    max_scores = max_scores.unstack()[['mean_test_score', 'mean_train_score']]
    sns.heatmap(max_scores.mean_test_score, annot=True, fmt='.4g')
    plt.title('Grid Search CV Score on Train and Test Data')
    plt.show()
    best_parameter= clf.best_params_
    print('Best depth: ', clf.best_estimator_.max_depth)
    print('Best n estimators: ', clf.best_estimator_.n_estimators)
    return clf.best_estimator_.max_depth, clf.best_estimator_.n_estimators
```

Funtion for fitting the model using best hyperparameter

In [159]:
```
def final_GBDT( Xtrain ,ytrain ,Xtest ,ytest , param1 , param2):
    """
    Returns : threshold values, False positive rate , True positive rate and the trained model
    ---
    Input : Train dataset, Test Dataset , best parameters
    """
    # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
    from sklearn.metrics import roc_curve, auc
    import matplotlib.pyplot as plt
    from xgboost import XGBClassifier
    clftre = XGBClassifier(booster='gbtree', n_estimators=param1 , max_depth=param2)
    clftre.fit(Xtrain,ytrain)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
    # not the predicted outputs
    train_fpr, train_tpr, thresholds = roc_curve(y_train, clftre.predict_proba(Xtrain)[:,1])
    test_fpr, test_tpr, thresholds = roc_curve(y_test, clftre.predict_proba(Xtest)[:,1])

    plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
    plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
    plt.legend()
    plt.grid(True)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title("ROC Curve")
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.show()
    return thresholds ,train_fpr, train_tpr, clftre
```

[5.2.1] Applying XGBOOST on BOW, SET 1

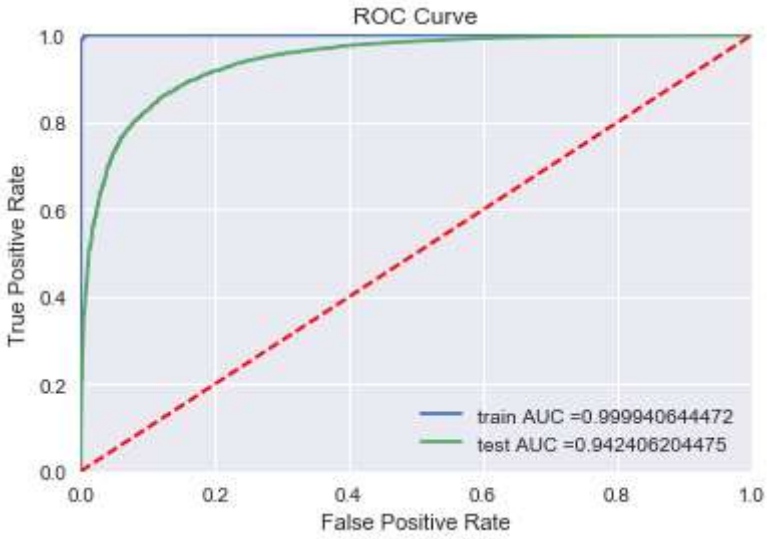
Hyperparameter tuning

In [160]:
```
%%time
best_depth_bow_xg , n_models_bow_xg = best_GBDT(X_train_bow,y_train)
```

Best depth: 100
Best n estimators: 90
Wall time: 7h 17min 24s

fitting the model with best hyperarameter

In [161]:
```
thr_bow_xg,bow_fpr_xg , bow_tpr_xg, bow_model_xg =  final_GBDT(X_train_bow,y_train,X_tes
t_bow,y_test,n_models_bow_xg,best_depth_bow_xg)
```



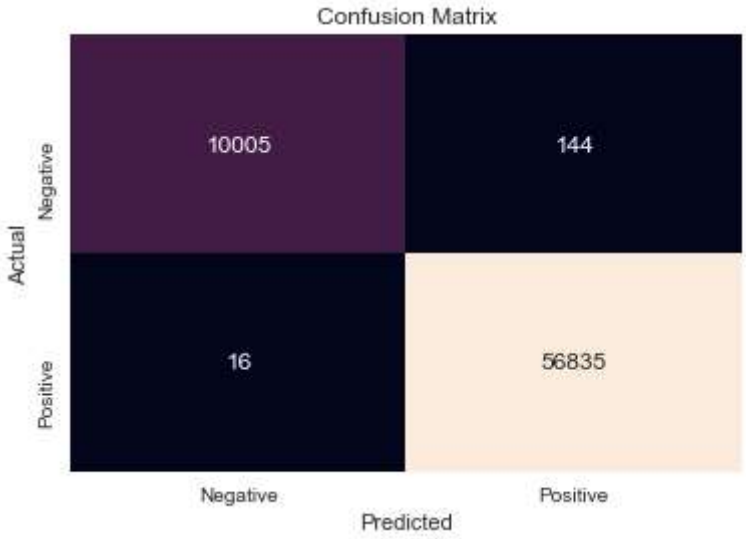
Word Cloud

In [162]:
```
ft3 = vectorizer.get_feature_names()
string3=''
fi3=bow_model_xg.feature_importances_
features3=np.argsort(fi3)[::-1]
for i in features3[0:20]:
    string3+=ft[i]
    string3+=' '


from wordcloud import WordCloud
wordcloud = WordCloud(background_color="red").generate(string3)
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

confusion matrix

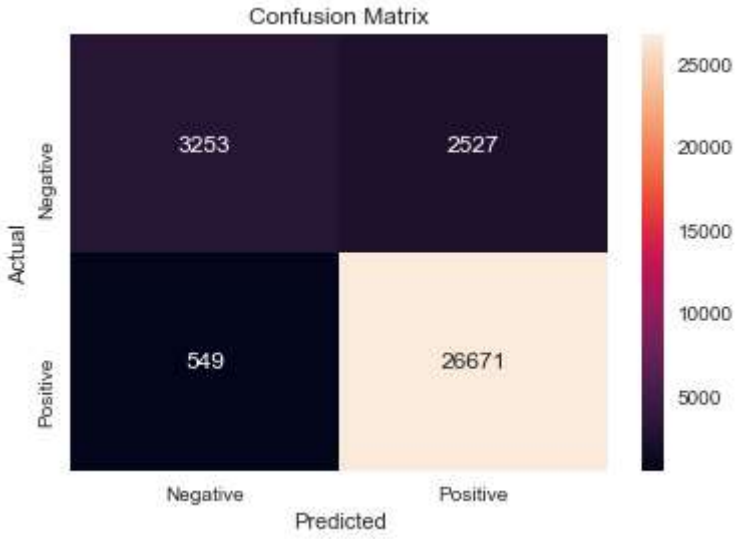
In [170]:
```
#best_t_bow_xg = find_best_threshold(thr_bow_xg, bow_fpr_xg, bow_tpr_xg)
conf_matrix(X_train_bow,y_train,X_test_bow,y_test,0.5,bow_model_xg)
```

Train Data



==========

Test Data



[5.2.2] Applying XGBOOST on TFIDF, SET 2

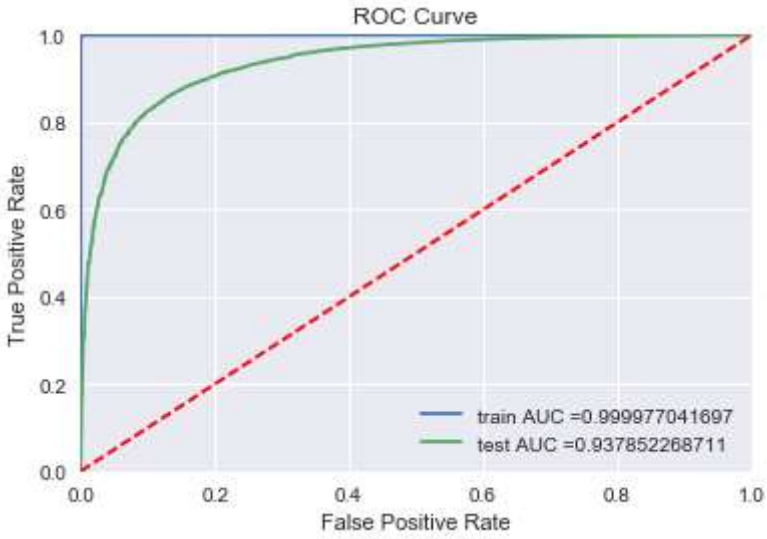
Hyperparameter tuning

In [180]:
```
%%time
best_depth_tfidfxg, best_n_estimatorsxg = best_RF(X_train_tfidf,y_train)
```

Best depth: 200
Best n estimators: 90
Wall time: 1h 20min 21s

fitting model with best hyperparameter

In [181]:
```
th_tfidfxg, fpr_tfidfxg , tpr_tfidfxg , clftre_tfidfxg = final_RF(X_train_tfidf,y_train, X_test_tfidf, y_test,best_n_estimatorsxg,best_depth_tfidfxg)
```



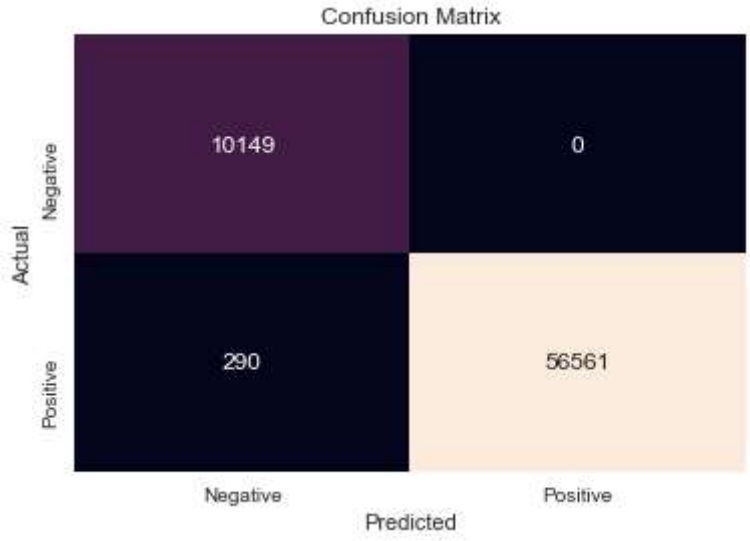
Word Cloud

In [182]:
```
ft4 = tf_idf_vect.get_feature_names()
string4=''
fi4=clftre_tfidf.feature_importances_
features4=np.argsort(fi4)[::-1]
for i in features4[0:20]:
    string4+=ft[i]
    string4+=' '


from wordcloud import WordCloud
wordcloud = WordCloud(background_color="red").generate(string2)
plt.figure(figsize=(10, 8))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

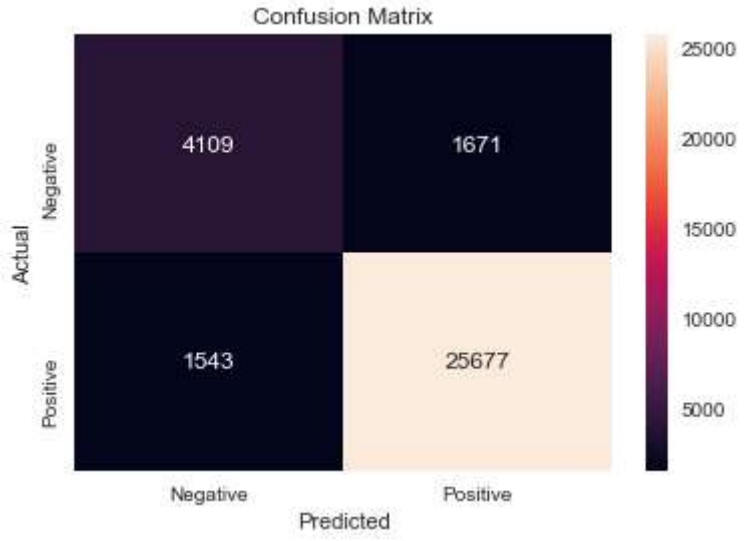
confusion matrix

In [183]:
```
best_t_tfidfxg = find_best_threshold(th_tfidfxg, fpr_tfidfxg, tpr_tfidfxg)
conf_matrix(X_train_tfidf,y_train,X_test_tfidf,y_test,best_t_tfidfxg,clftre_tfidfxg)
```

the maximum value of tpr*(1-fpr) 0.996260050588 for threshold 0.661
Train Data



==========
Test Data



[5.2.3] Applying XGBOOST on AVG W2V, SET 3

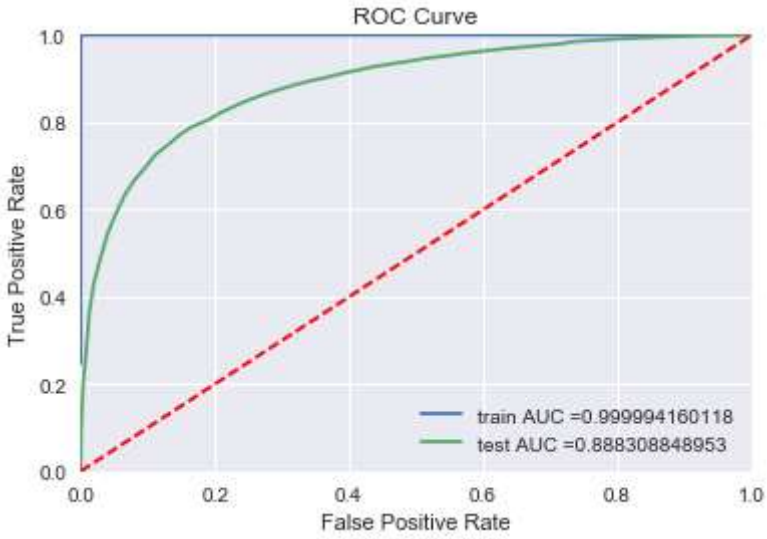
Hyperparameter tuning

In [171]:
```
%%time
best_depth_avgw2vxg, best_n_estimaotr_avgw2vxg = best_RF(sent_vectors_train,y_train)
```

Best depth: 300
Best n estimators: 90
Wall time: 1h 25min 28s

fitting the model with best hyperparameter

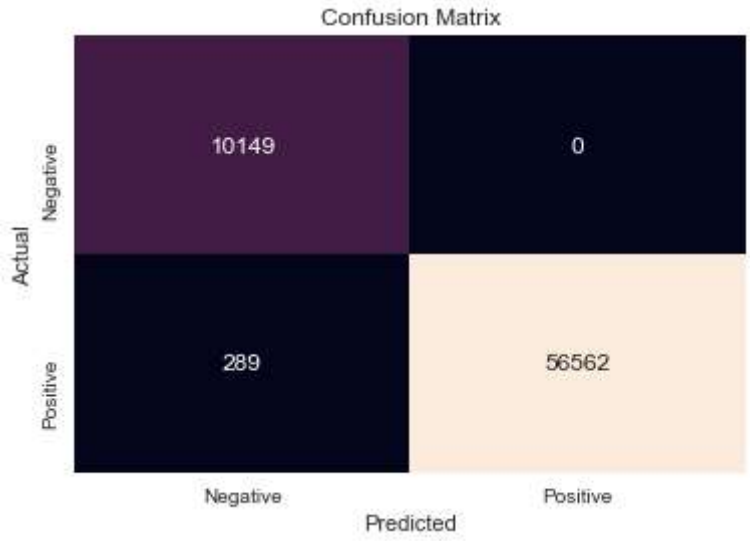
In [172]:
```
th_avgw2vxg, fpr_avgw2vxg , tpr_avgw2vxg , clftre_avgw2vxg = final_RF(sent_vectors_train,y_t
rain, sent_vectors_test, y_test,best_n_estimaotr_avgw2vxg
                                                ,best_depth_avgw2vxg)
```



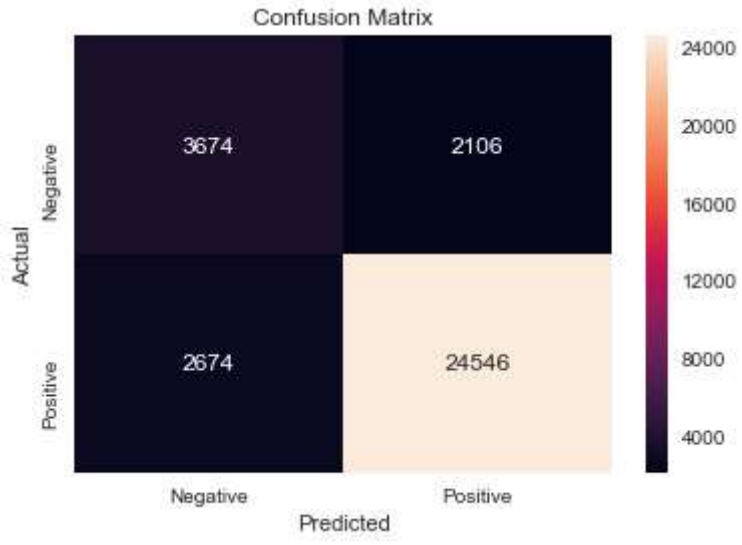
Confusion matrix

In [173]:
```
best_t_2_avgw2vxg = find_best_threshold(th_avgw2vxg, fpr_avgw2vxg, tpr_avgw2vxg)
conf_matrix(sent_vectors_train,y_train,sent_vectors_test,y_test,best_t_2_avgw2vxg,clftre_avgw2vxg)
```

the maximum value of tpr*(1-fpr) 0.997238395103 for threshold 0.733
Train Data



==========
Test Data



[5.2.4] Applying XGBOOST on TFIDF W2V, SET 4

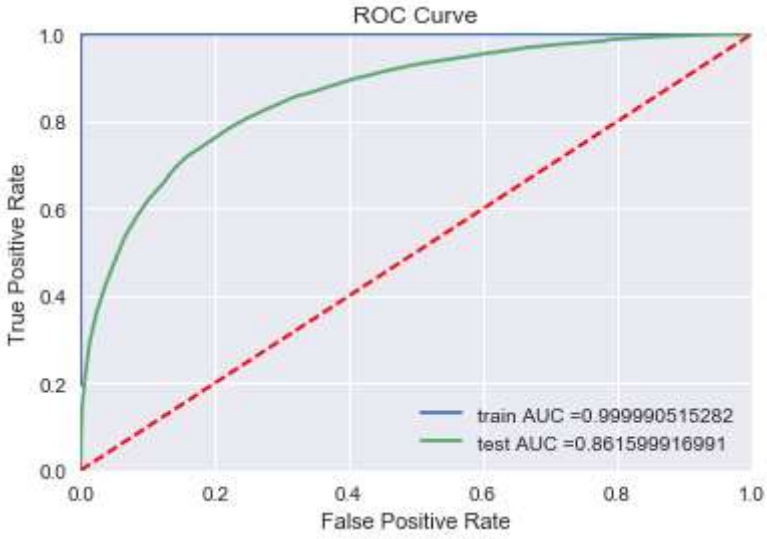
Hyperparameter tuning

In [177]:
```
%%time
best_depth_tfidfavgw2vxg, best_n_estimator_tfidfavgw2vxg = best_RF(tfidf_sent_vectors_train, y_train)
```

Best depth: 200
Best n estimators: 90
Wall time: 1h 19min 26s

fitting the model with best hyperparameter

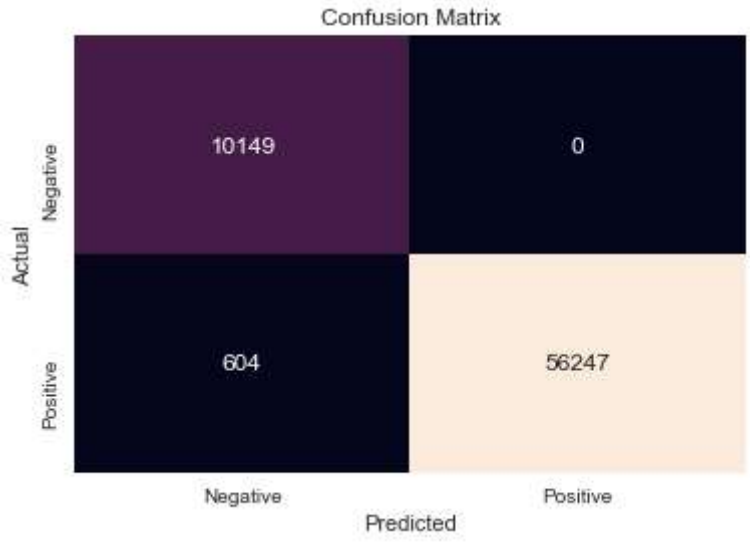
In [178]:
```
th_tfidfavgw2vxg, fpr_tfidfavgw2vxg , tpr_tfidfavgw2vxg , clftre_tfidfavgw2vxg = final_RF(tfidf_sent_vectors_train,y_train, tfidf_sent_vectors_test, y_test,best_n_estimator_tfidfavgw2vxg
                                                                                          ,best_depth_tfidfavgw2vxg)
```



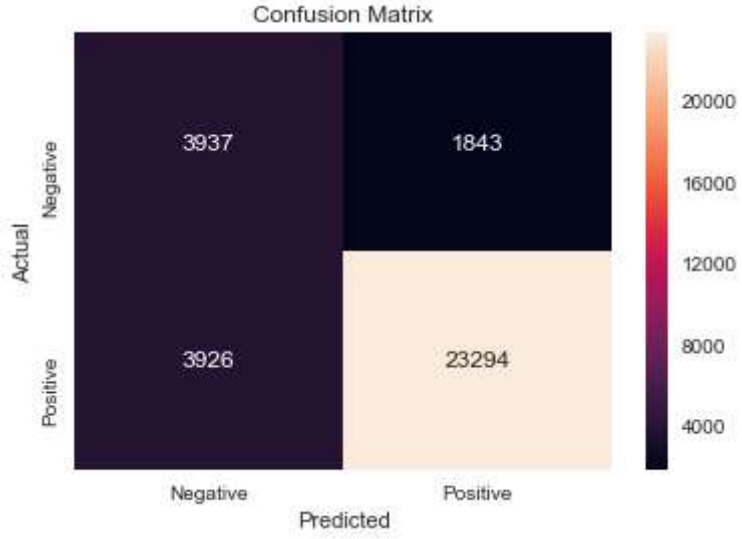
confusion matrix

In [179]:
```
best_t_2_tfidfavgw2vxg = find_best_threshold(th_tfidfavgw2vxg, fpr_tfidfavgw2vxg, tpr_tfidfavgw2vxg)
conf_matrix(tfidf_sent_vectors_train,y_train,tfidf_sent_vectors_test,y_test,best_t_2_tfidfavgw2vxg,clftre_tfidfavgw2vxg)
```

the maximum value of tpr*(1-fpr) 0.997238395103 for threshold 0.789
Train Data



==========
Test Data



[6] Conclusions

In [188]:
```
# Creating table using PrettyTable library
from prettytable import PrettyTable
vectorizer = ['BOW','TFIDF','Avgw2v','TFIDFW2v','BOW','TFIDF','Avgw2v','TFIDFW2v' ]
AUC = [0.93,0.93,0.88,0.86,0.94,0.93,0.88,0.86]
Model = ['Random Forest','Random Forest','Random Forest','Random Forest','GBDT (XGBOOS
T)','GBDT (XGBOOST)','GBDT (XGBOOST)','GBDT (XGBOOST)' ]
max_depth = [200,200,200,200,100,200,300,200]
n_models = [90,90,90,90,90,90,90,90]
sno = [1,2,3,4,5,6,7,8]
# Initializing prettytable
ptable = PrettyTable()
# Adding columns
ptable.add_column("S.NO.",sno)
ptable.add_column("     Vectorizer           ",vectorizer)
ptable.add_column("      Model                      "   ,      Model)
ptable.add_column("  n_estimators    ",n_models)
ptable.add_column("  best_depth     ",max_depth)
ptable.add_column("  Test AUC             ",AUC)
# Printing the Table
print(ptable)
```

```
+-------+----------------------------+-----------------------------------+-------------------+------------------+-----
-------------------+
| S.NO. |      Vectorizer            |      Model                        |   n_estimators    |  best_depth      |
Test AUC             |
+-------+----------------------------+-----------------------------------+-------------------+------------------+-----
-------------------+
|   1   |           BOW              |          Random Forest            |        90         |       200        |
0.93          |
|   2   |          TFIDF             |          Random Forest            |        90         |       200        |
0.93          |
|   3   |          Avgw2v            |          Random Forest            |        90         |       200        |
0.88          |
|   4   |         TFIDFW2v           |          Random Forest            |        90         |       200        |
0.86          |
|   5   |           BOW              |          GBDT (XGBOOST)           |        90         |       100        |
0.94          |
|   6   |          TFIDF             |          GBDT (XGBOOST)           |        90         |       200        |
0.93          |
|   7   |          Avgw2v            |          GBDT (XGBOOST)           |        90         |       300        |
0.88          |
|   8   |         TFIDFW2v           |          GBDT (XGBOOST)           |        90         |       200        |
0.86          |
+-------+----------------------------+-----------------------------------+-------------------+------------------+-----
-------------------+
```

In [189]:
```
!jupyter nbconvert --to html AFFRRF.ipynb
```

```
[NbConvertApp] Converting notebook AFFRRF.ipynb to html
[NbConvertApp] Writing 1849901 bytes to AFFRRF.html
```

In [ ]: