

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 - unique 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 considered as a positive review. A rating of 1 or 2 can be considered as negative one. A review of rating 3 is considered neutral 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 [57]: %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
import numpy as np
import matplotlib.pyplot as plt
from io import StringIO
from sklearn.tree import export_graphviz
from sklearn.model_selection import train_test_split as tts
from sklearn import tree
from sklearn import metrics
from IPython.display import Image
import pydotplus
```

```
In [4]: # 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 500000""", con)  
# for tsne assignment you can take 5k data points  
  
filtered_data = pd.read_sql_query(""" SELECT * FROM Reviews WHERE Score != 3 LIMIT 450000""", 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 (450000, 10)

Out[4]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator
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 [5]:

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

In [6]: `print(display.shape)
display.head()`

(80668, 7)

Out[6]:

	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 [7]: `display[display['UserId']=='AZY10LLTJ71NX']`

Out[7]:

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



```
In [8]: display['COUNT(*)'].sum()
```

```
Out[8]: 393063
```

[2] Exploratory Data Analysis

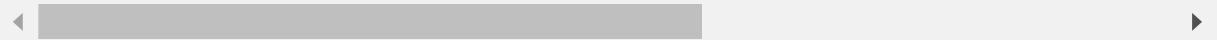
[2.1] Data Cleaning: Deduplication

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

Out[9]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator
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 delete 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 [10]: *#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 [11]: *#Deduplication of entries*
`final=sorted_data.drop_duplicates(subset={"UserId","ProfileName","Time","Text"}, keep='first', inplace=False)
final.shape`

Out[11]: (319788, 10)

In [12]: *#Checking to see how much % of data still remains*
`(final['Id'].size*1.0)/(filtered_data['Id'].size*1.0)*100`

Out[12]: 71.06400000000001

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 calculations

```
In [13]: display= pd.read_sql_query("""
SELECT *
FROM Reviews
WHERE Score != 3 AND Id=44737 OR Id=64422
ORDER BY ProductID
""", con)

display.head()
```

Out[13]:

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

◀ ▶

```
In [14]: final=final[final.HelpfulnessNumerator<=final.HelpfulnessDenominator]
```

```
In [15]: #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()
```

```
(319786, 10)
```

```
Out[15]: 1    269419
0    50367
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 observed to be better than Porter Stemming)

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

In [16]: # 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)
```

This is a great little book to read aloud- it has a nice rhythm as well as good repetition that little ones like, in the lines about "chicken soup with rice". The child gets to go through the months of the year and go to wonderful places like Bombay and down the Nile, all the while, eating, well, you know what they get to eat. Some kids will have Maurice Sendak's version of ice skating or how to treat roses in their heads for a long time and they won't even know where it came from. Surprise! It came from this little witty book. :-)

I've purchased both the Expressione Espresso (classic) and the 100% Arabica. My vote is definitely with the 100% Arabica. The flavor has more bite and flavor (much more like European coffee than American).

We bought a small bag of Canidae to give it a try, as we are always trying new foods for our 3 dogs. I normally serve 2 kinds of food for dinner and whenever we put Canidae in the dish, the dogs would always eat the other food first. I'm glad I only bought a small bag. For the price, the food should be more appealing. But our dogs just didn't like it and we were glad when it was all gone. I will never buy again!

Sans Sucre "Mousse Mix - Chocolate Cheesecake" is a fantastic diet dessert. Being a diabetic I am always looking for good tasting, low carb food and this dessert is high on my list.

The texture is very good and the flavor is not "diet" at all.
Highly Recommended!

In [17]: # 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)
```

This is a great little book to read aloud- it has a nice rhythm as well as good repetition that little ones like, in the lines about "chicken soup with rice". The child gets to go through the months of the year and go to wonderful places like Bombay and down the Nile, all the while, eating, well, you know what they get to eat. Some kids will have Maurice Sendak's version of ice skating or how to treat roses in their heads for a long time and they won't even know where it came from. Surprise! It came from this little witty book. :-)

```
In [18]: # 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)
```

This is a great little book to read aloud- it has a nice rhythm as well as good repetition that little ones like, in the lines about "chicken soup with rice". The child gets to go through the months of the year and go to wonderful places like Bombay and down the Nile, all the while, eating, well, you know what they get to eat. Some kids will have Maurice Sendak's version of ice skating or how to treat roses in their heads for a long time and they won't even know where it came from. Surprise! It came from this little witty book. :-)

I've purchased both the Expressione Espresso (classic) and the 100% Arabica. My vote is definitely with the 100% Arabica. The flavor has more bite and flavor (much more like European coffee than American).

We bought a small bag of Canidae to give it a try, as we are always trying new foods for our 3 dogs. I normally serve 2 kinds of food for dinner and whenever we put Canidae in the dish, the dogs would always eat the other food first. I'm glad I only bought a small bag. For the price, the food should be more appealing. But our dogs just didn't like it and we were glad when it was all gone. I will never buy again!

Sans Sucre "Mousse Mix - Chocolate Cheesecake" is a fantastic diet dessert. Being a diabetic I am always looking for good tasting, low carb food and this dessert is high on my list. The texture is very good and the flavor is not "diet" at all. Highly Recommended!

In [19]: *# 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 [20]: `sent_1500 = decontracted(sent_1500)`

```
print(sent_1500)
print("="*50)
```

We bought a small bag of Canidae to give it a try, as we are always trying new foods for our 3 dogs. I normally serve 2 kinds of food for dinner and whenever we put Canidae in the dish, the dogs would always eat the other food first. I am glad I only bought a small bag. For the price, the food should be more appealing. But our dogs just did not like it and we were glad when it was all gone. I will never buy again!

In [21]: *#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039*

```
sent_0 = re.sub("\S*\d\S*", "", sent_0).strip()
print(sent_0)
```

This is a great little book to read aloud- it has a nice rhythm as well as good repetition that little ones like, in the lines about "chicken soup with rice". The child gets to go through the months of the year and go to wonderful places like Bombay and down the Nile, all the while, eating, well, you know what they get to eat. Some kids will have Maurice Sendak's version of ice skating or how to treat roses in their heads for a long time and they won't even know where it came from. Surprise! It came from this little witty book. :-)

In [22]: *#remove spacial character: https://stackoverflow.com/a/5843547/4084039*

```
sent_1500 = re.sub('[^A-Za-z0-9]+', ' ', sent_1500)
print(sent_1500)
```

We bought a small bag of Canidae to give it a try as we are always trying new foods for our 3 dogs I normally serve 2 kinds of food for dinner and whenever we put Canidae in the dish the dogs would always eat the other food first I am glad I only bought a small bag For the price the food should be more appealing But our dogs just did not like it and we were glad when it was all gone I will never buy again

```
In [23]: # 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 removed in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you'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", 'how', '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 [24]: # Combining all the above students
from tqdm import tqdm
preprocessed_reviews = []
# tqdm is for printing the status bar
for sentence in tqdm(final['Text'].values):
    sentence = re.sub(r"http\S+", "", sentence)
    sentence = BeautifulSoup(sentence, 'lxml').get_text()
    sentence = decontracted(sentence)
    sentence = re.sub("\S*\d\S*", "", sentence).strip()
    sentence = re.sub('[^A-Za-z]+', ' ', sentence)
    # https://gist.github.com/sebleier/554280
    sentence = ''.join(e.lower() for e in sentence.split() if e.lower() not in stopwords)
    preprocessed_reviews.append(sentence.strip())
```

100% [██████████] 319786/319786 [02:50<00:00, 1872.69it/s]

```
In [25]: preprocessed_reviews[1500]
```

```
Out[25]: 'bought small bag canidae give try always trying new foods dogs normally serve kinds food dinne
r whenever put canidae dish dogs would always eat food first glad bought small bag price food a
ppealing dogs not like glad gone never buy'
```

```
In [94]: #replacing the reviews with preprocessed reviews
final['Text']=preprocessed_reviews

positive = final[final["Score"] == 1].sample(n = 50000)
negative = final[final["Score"] == 0].sample(n = 50000)
final_posinegi = pd.concat([positive, negative])
final_posinegi.shape
```

Out[94]: (100000, 10)

```
In [95]: fd = final_posinegi
# Sorting based on time
fd['Time'] = pd.to_datetime(fd['Time'], origin='unix', unit='s')
fd= fd.sort_values('Time')
fd.shape
```

Out[95]: (100000, 10)

```
In [96]: #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 [99]: 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)
```

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

[4.2] Bi-Grams and n-Grams.

```
In [100]: #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
#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 numbers 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(X_train)
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 (67000, 5000)
the number of unique words including both unigrams and bigrams 5000

[4.3] TF-IDF

```
In [101]: #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)

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)

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 [102]: #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 sentences
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 occurred 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 occurred minimum 5 times 16187
 sample words ['dim', 'incorporated', 'tobacman', 'handle', 'kelloggs', 'wally', 'polysorbate', 'restful',
 'knew', 'overwhelms', 'catering', 'tried', 'kirsch', 'slick', 'saying', 'numbers', 'gloves', 'soldiers', 'leath
 ery', 'hurry', 'flaxseed', 'drew', 'reportedly', 'compostable', 'ms', 'saturday', 'rick', 'hash', 'sep', 'inge
 sts', 'entr', 'lunchbox', 'balls', 'sipping', 'truffettes', 'micklegal', 'porcini', 'optional', 'passable', 'stress
 ful', 'intelligent', 'dragging', 'tuscany', 'vegies', 'separation', 'perplexed', 'hey', 'branching', 'volvic',
 'adventure']

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

[4.4.1.1] Avg W2v

converting train data

```
In [103]: #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
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 this 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, 81280.45it/s]
100% [██████████] 67000/67000 [17:53<00:00, 62.42it/s]

(67000, 50)
[-0.1803195 -0.13940734 -0.2849912 0.27537448 -0.24381202 0.34372241
0.22689834 -0.51288711 0.06082354 -0.58063919 0.57698228 0.37321008
0.16578258 -0.13367881 0.20810957 0.06665729 0.11591636 -0.08841441
0.05655464 -0.12282168 -0.34626053 -0.11077388 0.13928449 0.32836773
0.1164139 -0.15587625 -0.09838883 0.28216509 0.43204927 -0.08650029
0.59855421 -0.22013457 -0.08082424 -0.06847419 0.52872963 -0.33214108
0.43698825 -0.15248731 0.43095022 -0.32605791 -0.20527792 -0.40423975
-0.01743077 0.24407261 0.33959067 0.01299589 0.3002753 0.31177462
-0.04067031 0.31891449]

converting test data

```
In [104]: 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 this
    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 [09:31<00:00, 57.74it/s]

(33000, 50)
[-0.15775323 -0.52375099 -1.05260785 0.07637636 0.29775889 0.42357553
 0.23755251 0.10277213 0.04255446 -0.61672753 -0.61765669 -0.1254461
 0.81928839 0.35288902 0.73848755 -0.28928202 0.2410572 -0.08628419
 -0.48963884 0.12831096 -0.08677552 0.27199203 -0.37136227 -0.49031368
 0.98964234 -0.62725621 0.07255439 -0.10144516 1.09731896 0.05396022
 0.31800065 -0.16079359 -0.35464524 -0.15055434 0.11788343 -0.26226764
 0.0334972 -0.33685697 -0.1365845 0.2089626 -0.33025685 -0.84765875
 0.24425624 0.70683078 -0.35575626 -0.55581838 0.22254658 -0.04813054
 -0.05167773 0.51898974]

[4.4.1.2] TFIDF weighted W2v

```
In [105]: #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
#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, 49825) (33000, 49825)

converting train data

```
In [106]: 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 corpus
            # sent.count(word) = tf values 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% [██████████] 67000/67000 [1:13:16<00:00, 15.24it/s]

converting test data

```
In [107]: 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 corpus
#             sent.count(word) = tf values 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 [25:27<00:00, 21.60it/s]

```
In [108]: print(tfidf_sent_vectors_train.shape,tfidf_sent_vectors_test.shape)
```

(67000, 50) (33000, 50)

[5] Assignment 8: Decision Trees

1. Apply Decision Trees 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 parameter tuning (best `depth` in range [4,6, 8, 9,10,12,14,17] , and the best `min_samples_split` in range [2,10,20,30,40,50])

- 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 parameter 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. Graphviz

- Visualize your decision tree with Graphviz. It helps you to understand how a decision is being made, given a new vector.
- Since feature names are not obtained from word2vec related models, visualize only BOW & TFIDF decision trees using Graphviz
- Make sure to print the words in each node of the decision tree instead of printing its index.
- Just for visualization purpose, limit max_depth to 2 or 3 and either embed the generated images of graphviz in your notebook, or directly upload them as .png files.

4. Feature importance

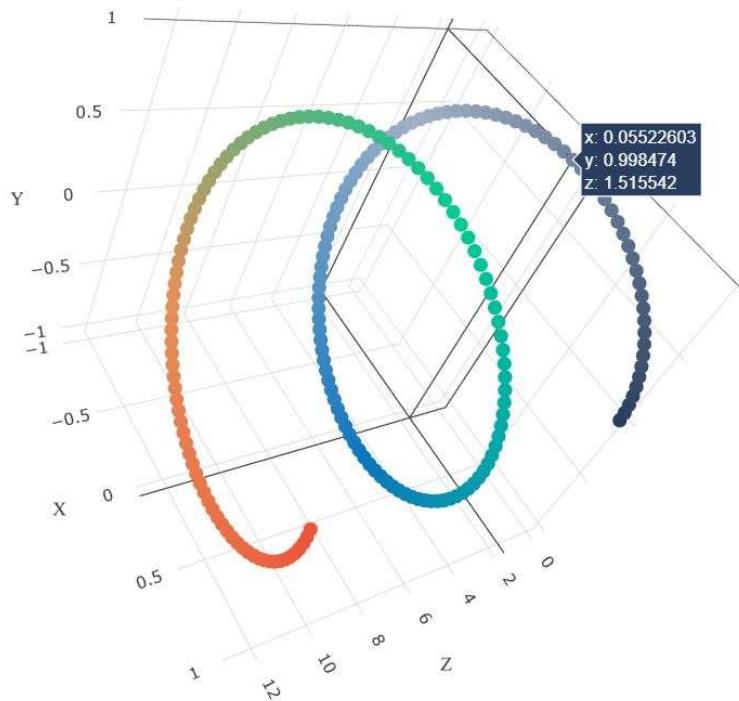
- Find the top 20 important features from both feature sets Set 1 and Set 2 using `feature_importances_` method of Decision Tree Classifier (<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>) and print their corresponding feature names

5. 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.

6. Representation of results

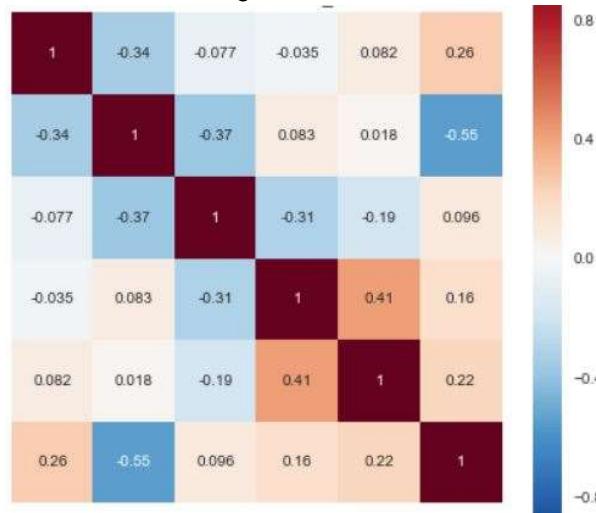
- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure



with X-axis as **min_sample_split**, 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](#)

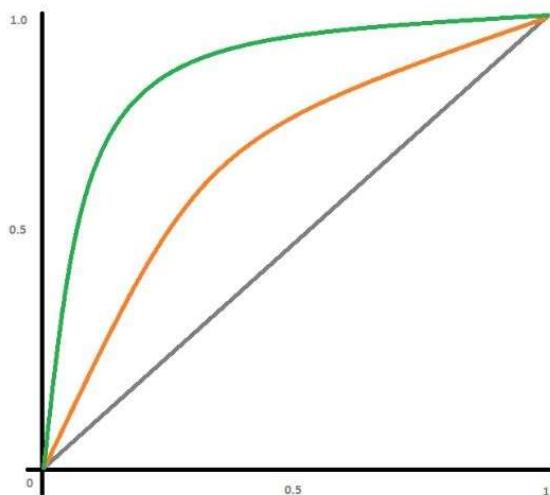
or

- 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 **min_sample_split**, 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>)

7. 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-leakage, make sure to split your data first and then vectorize it.
3. While vectorizing your data, apply the method `fit_transform()` on your 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). (<https://soundcloud.com/applied-ai-course/leakage-bow-and-tfidf>)

Applying Decision Trees

Function for finding the best depth and sample split in DecisionTree Classifier

```
In [109]: def dtc(Xtrain,ytrain):
    """
    Returns : best parameters of DecisionTree Classifier
    ---
    Input : Dataset
    """
    # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
    import numpy as np
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.metrics import roc_auc_score
    from sklearn.model_selection import GridSearchCV
    import matplotlib.pyplot as plt
    max_depth = [ 5, 10, 50, 100, 500]
    min_samples_split = [10, 50, 100, 200, 300 ]
    parameters = {'max_depth': [5, 10, 50, 100, 500], 'min_samples_split' :[10, 50, 100,200, 300]}
    clftree = DecisionTreeClassifier()
    clf = GridSearchCV(clftree,parameters,cv=4, scoring='roc_auc',return_train_score=True)
    clf.fit(Xtrain,ytrain)
    tab = pd.DataFrame(clf.cv_results_)
    import seaborn as sns
    plt.figure(figsize=(15, 8))
    max_scores = tab.groupby(['param_max_depth','param_min_samples_split']).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 samples split: ', clf.best_estimator_.min_samples_split)
    return clf.best_estimator_.max_depth, clf.best_estimator_.min_samples_split
```

Training the model with best hyperparameter

```
In [110]: def final_tree( 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.me
    metrics.roc_curve
    from sklearn.metrics import roc_curve, auc
    import matplotlib.pyplot as plt
    from sklearn.tree import DecisionTreeClassifier

    clftre = DecisionTreeClassifier(class_weight='balanced',min_samples_split=param1, max_de
    th=param2)
    clftre.fit(Xtrain,ytrain)
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the po
    sitive 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 [111]: # 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 [112]: 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] Applying Decision Trees on BOW, SET 1

Hyperparameter tuning

In [113]: **%%time**

```
best_depth_bow, best_sample_split_bow = dtc(X_train_bow,y_train)
```



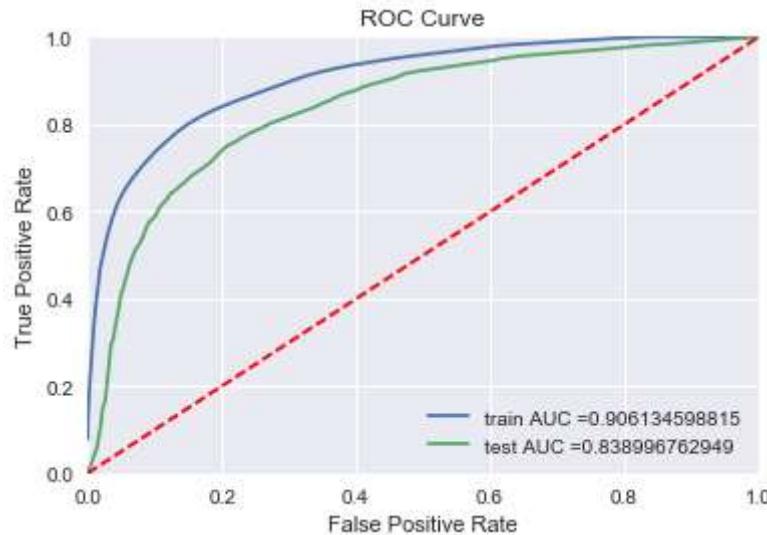
Best depth: 50

Best samples split: 300

Wall time: 25min 27s

Training the model with best hyperparameter

In [114]: `th_bow, fpr_bow , tpr_bow , clftre_bow = final_tree(X_train_bow,y_train, X_test_bow, y_test,best_sample_split_bow,best_depth_bow)`



[5.1.1] Top 20 important features from SET 1

```
In [115]: ft_1 = clftre_bow.feature_importances_
bow_ft = vectorizer.get_feature_names()
temp = sorted(zip(ft_1,bow_ft))
final_feature= pd.DataFrame(temp[:-21:-1])
final_feature
```

Out[115]:

	0	1
0	0.151516	not
1	0.096684	great
2	0.049114	best
3	0.039924	delicious
4	0.028564	disappointed
5	0.027592	love
6	0.027484	good
7	0.026047	perfect
8	0.019773	loves
9	0.017005	excellent
10	0.015468	bad
11	0.015125	favorite
12	0.014501	thought
13	0.014376	money
14	0.011662	nice
15	0.011643	wonderful
16	0.011053	easy
17	0.009792	not good
18	0.009748	awful
19	0.009597	worst

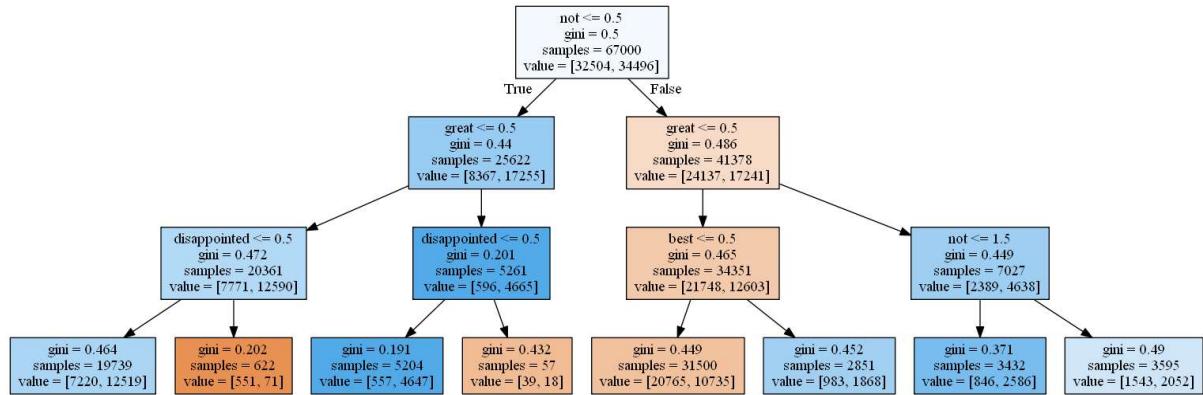
[5.1.2] Graphviz visualization of Decision Tree on BOW, SET 1

```
In [116]: model_bow=tree.DecisionTreeClassifier(max_depth=3)
model_bow.fit(X_train_bow, y_train)
# Create DOT data for visualizing the tree
dot_data_bow = tree.export_graphviz(model_bow, out_file=None, feature_names=bow_ft, filled=True)

# Draw graph
graph_bow = pydotplus.graph_from_dot_data(dot_data_bow)

# Show graph
Image(graph_bow.create_png())
```

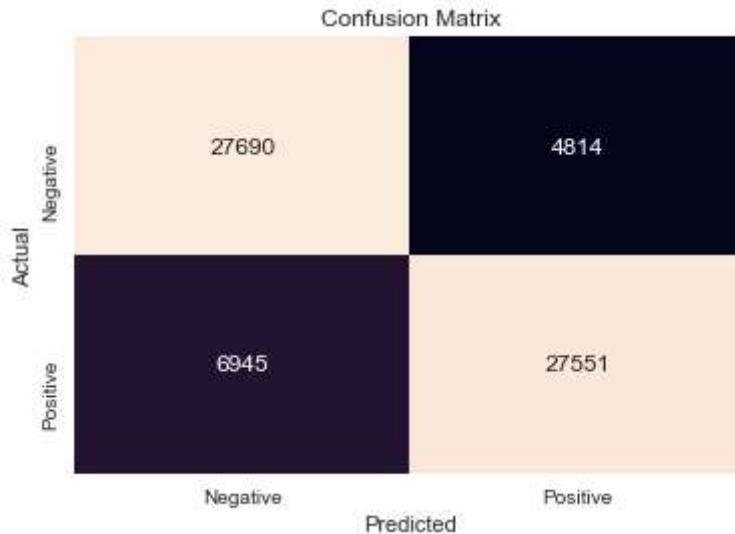
Out[116]:



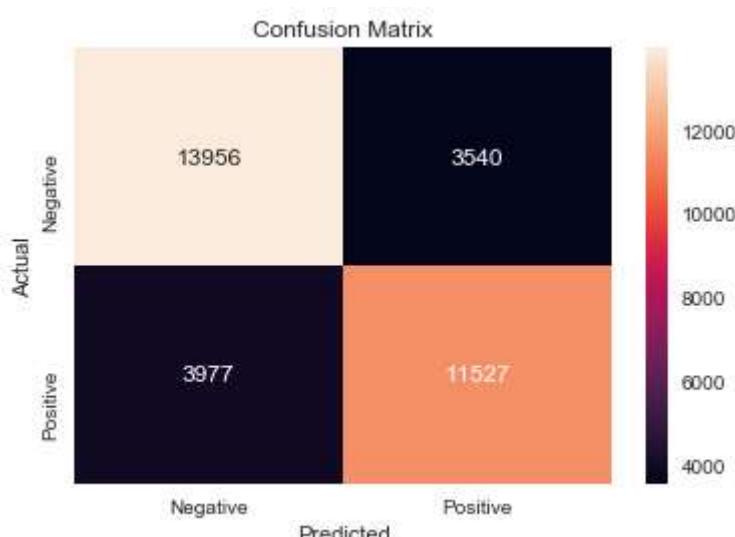
Confusion Matrix

```
In [117]: best_t_bow = find_best_threshold(th_bow, fpr_bow, tpr_bow)
conf_matrix(X_train_bow,y_train,X_test_bow,y_test,best_t_bow,clftre_bow)
```

the maximum value of $tpr * (1 - fpr)$ 0.680457570771 for threshold 0.501
 Train Data



=====
 Test Data



[5.2] Applying Decision Trees on TFIDF, SET 2

Hyperparameter tuning

In [118]: %%time

```
best_depth_tfidf, best_sample_split_tfidf = dtc(X_train_tfidf,y_train)
```



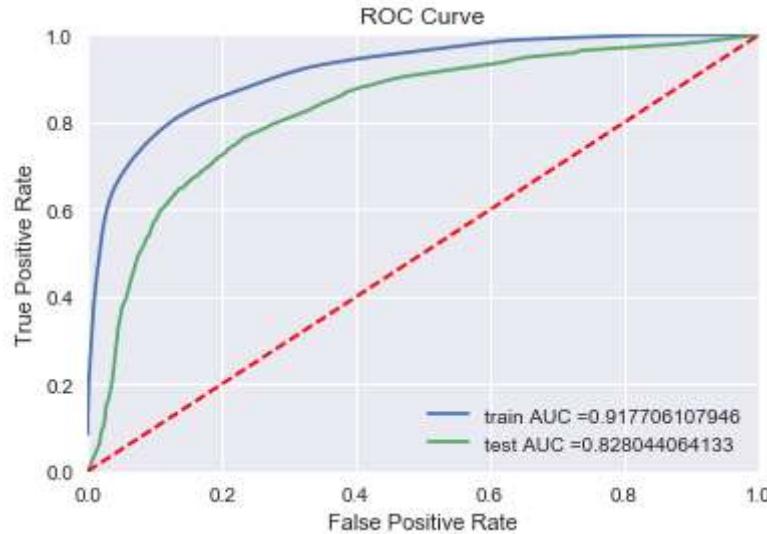
Best depth: 50

Best samples split: 300

Wall time: 30min 45s

Training the model with best hyperparameter

In [119]: th_tfidf, fpr_tfidf, tpr_tfidf, clftre_tfidf = final_tree(X_train_tfidf,y_train, X_test_tfidf, y_test,best_sample_split_tfidf,best_depth_tfidf)



[5.2.1] Top 20 important features from SET 2

```
In [120]: ft_2 = clftre_tfidf.feature_importances_
tfidf_ft = tf_idf_vect.get_feature_names()
temp2 = sorted(zip(ft_2,tfidf_ft))
final_feature_tfidf= pd.DataFrame(temp2[:-21:-1])
final_feature_tfidf
```

Out[120]:

	0	1
0	0.129622	not
1	0.093118	great
2	0.046307	best
3	0.036889	delicious
4	0.028373	disappointed
5	0.027465	love
6	0.025208	perfect
7	0.024827	good
8	0.019255	loves
9	0.018097	excellent
10	0.016318	favorite
11	0.015756	thought
12	0.014148	bad
13	0.013247	money
14	0.011555	nice
15	0.011539	easy
16	0.010906	wonderful
17	0.009490	awful
18	0.008555	not good
19	0.008130	worst

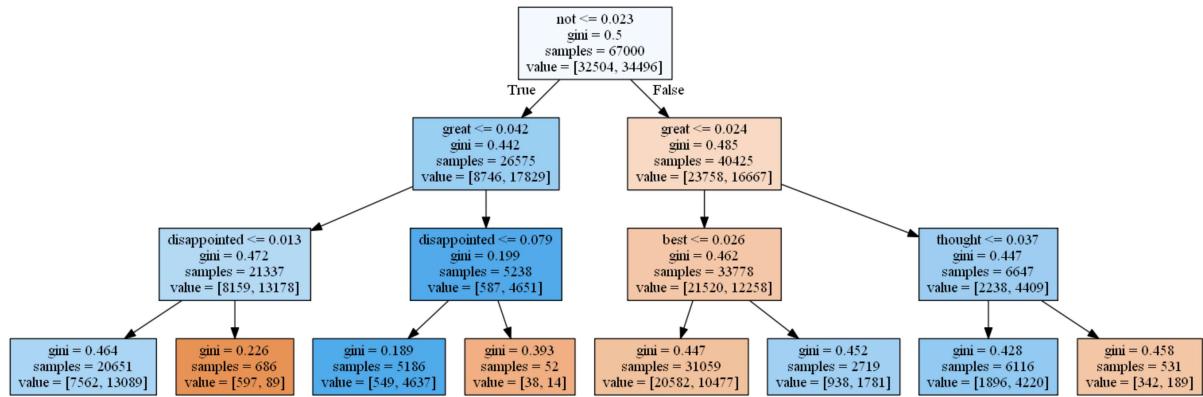
[5.2.2] Graphviz visualization of Decision Tree on TFIDF, SET 2

```
In [121]: model_tfidf=tree.DecisionTreeClassifier(max_depth=3)
model_tfidf.fit(X_train_tfidf, y_train)
# Create DOT data for visualizing the tree
dot_data_tfidf = tree.export_graphviz(model_tfidf, out_file=None, feature_names=tfidf_ft, filled = True)

# Draw graph
graph_tfidf = pydotplus.graph_from_dot_data(dot_data_tfidf)

# Show graph
Image(graph_tfidf.create_png())
```

Out[121]:

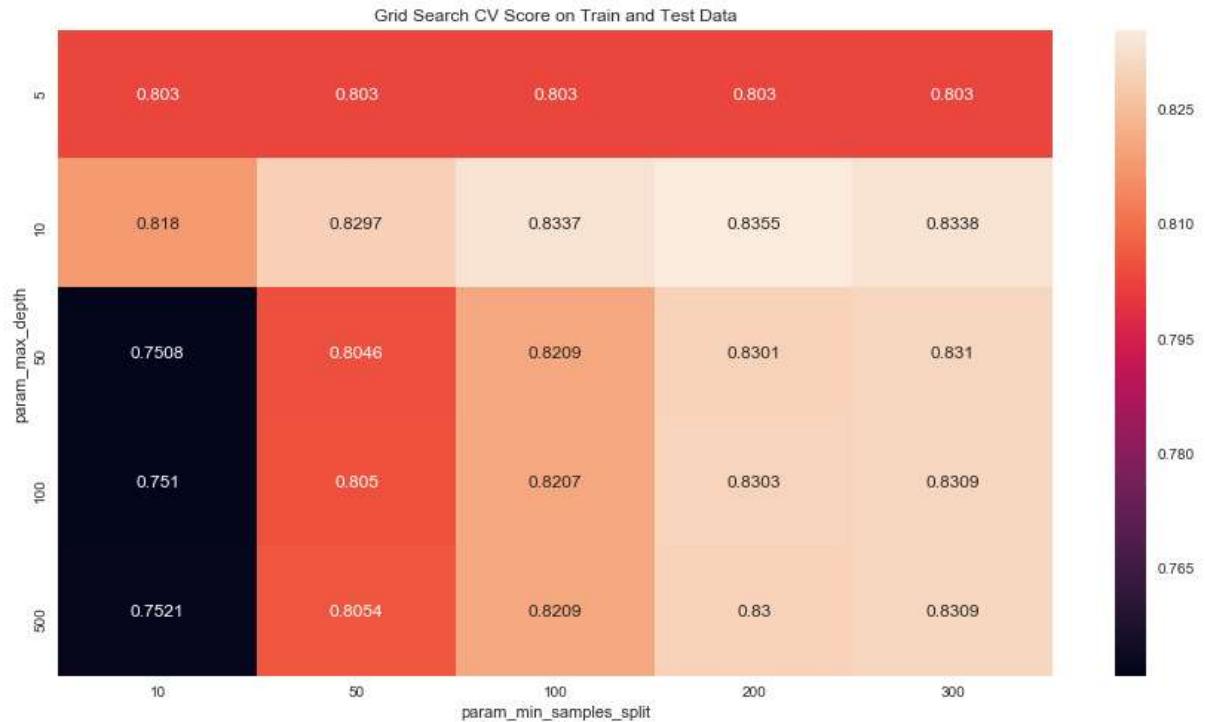


[5.3] Applying Decision Trees on AVG W2V, SET 3

Hyperparameter Tuning

In [122]: **%%time**

```
best_depth_avgw2v, best_sample_split_avgw2v = dtc(sent_vectors_train,y_train)
```



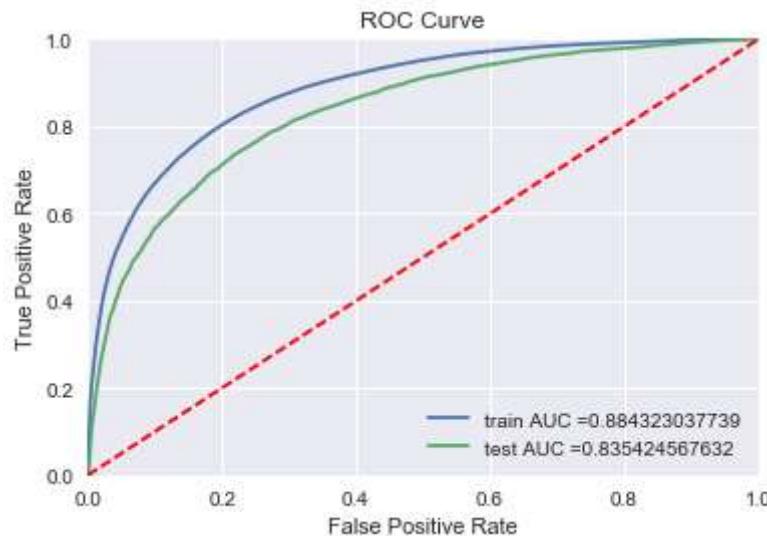
Best depth: 10

Best samples split: 200

Wall time: 5min 21s

Training the model with best hyperparameter

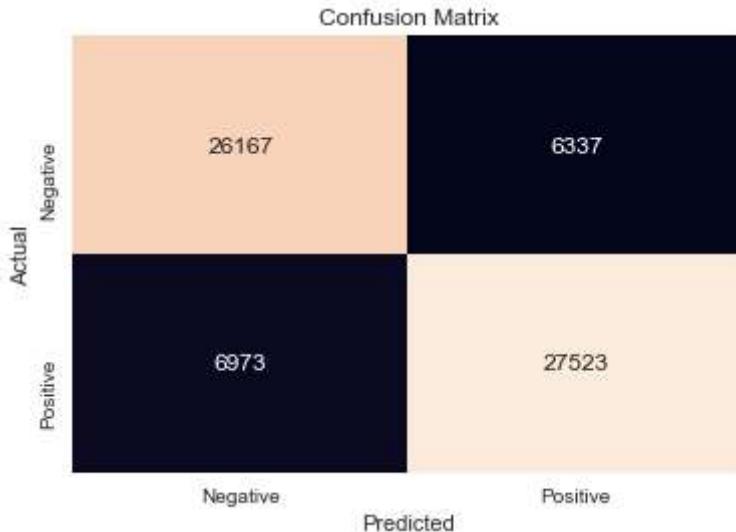
In [123]: **th_avgw2v, fpr_avgw2v , tpr_avgw2v , clftre_avgw2v = final_tree(sent_vectors_train,y_train, sent_vectors_test, y_test,best_sample_split_avgw2v ,best_depth_avgw2v)**



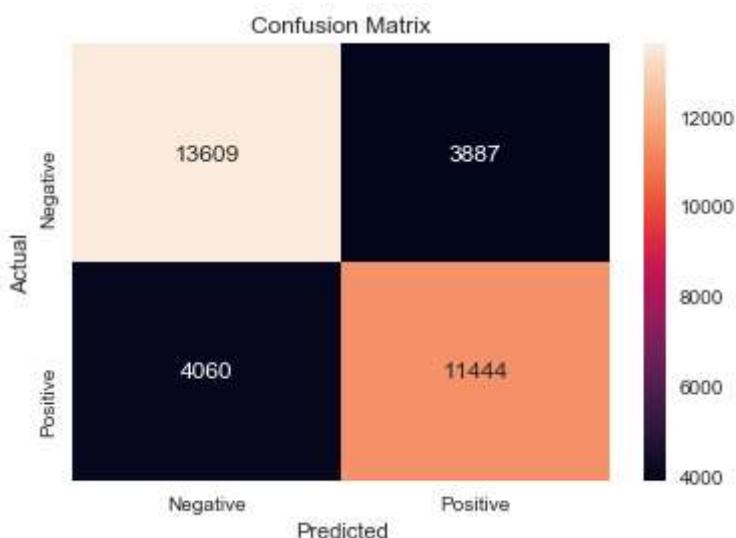
confusion matrix

```
In [124]: 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 $tpr * (1 - fpr)$ 0.642309219891 for threshold 0.497
 Train Data



=====
 Test Data

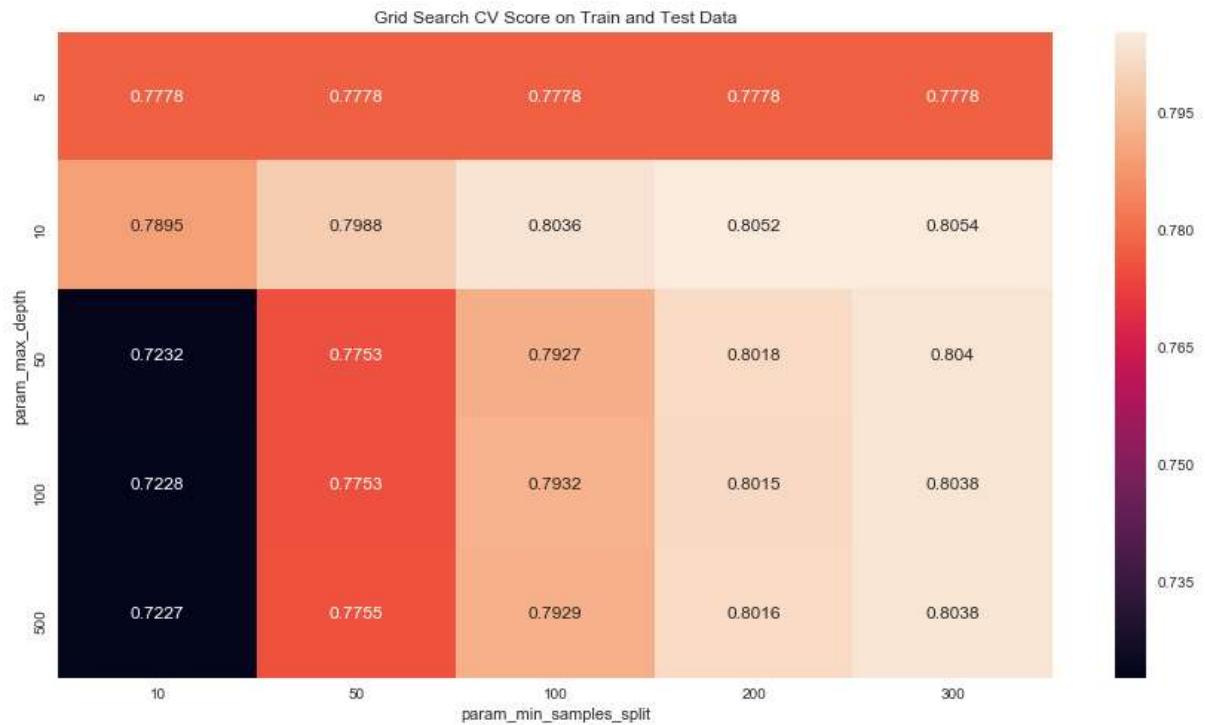


[5.4] Applying Decision Trees on TFIDF W2V, SET 4

Hyperparameter Tuning

In [125]: %%time

```
best_depth_tfidfavgw2v, best_sample_split_tfidfavgw2v = dtc(tfidf_sent_vectors_train,y_train)
```



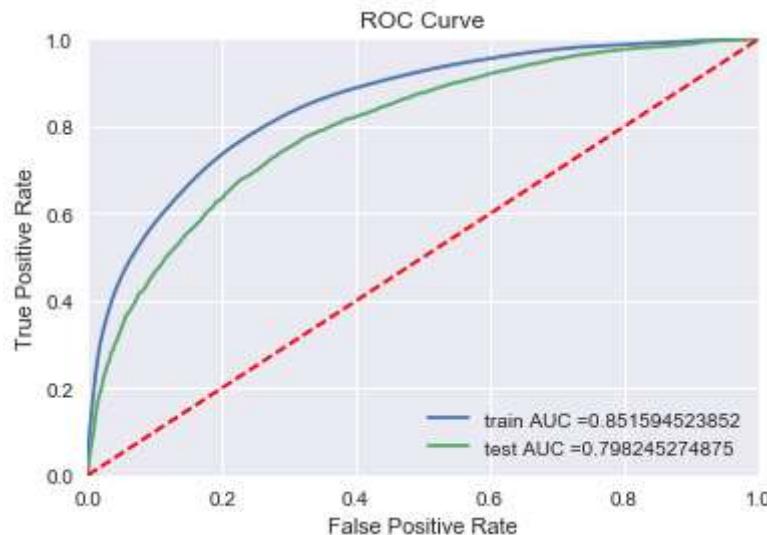
Best depth: 10

Best samples split: 300

Wall time: 5min 6s

Training the model with best hyperparameter

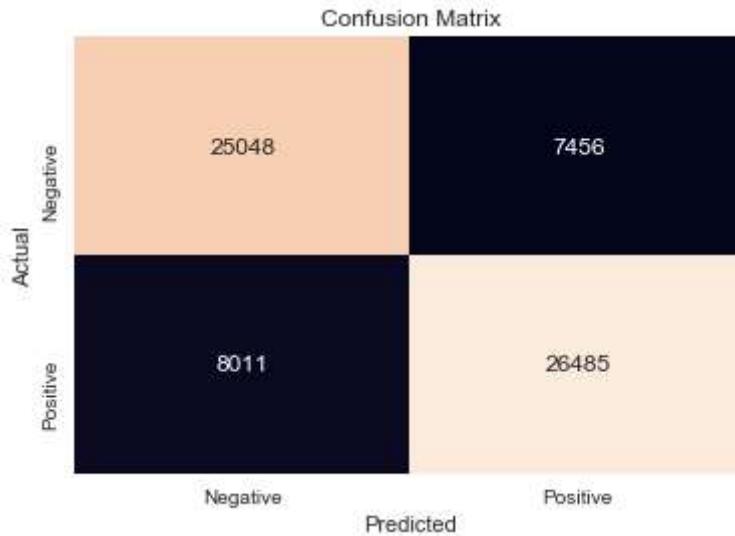
In [126]: th_tfidfavgw2v, fpr_tfidfavgw2v , tpr_tfidfavgw2v , clftre_tfidfavgw2v = final_tree(tfidf_sent_vectors_train,y_train, tfidf_sent_vectors_test, y_test,best_sample_split_tfidfavgw2v ,best_depth_tfidfavgw2v)



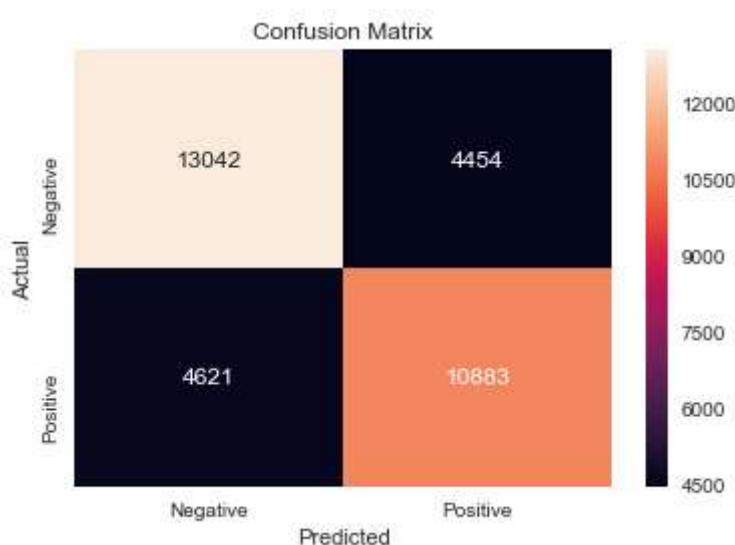
confusion matrix

```
In [127]: 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 $tpr^*(1-fpr)$ 0.59165777142 for threshold 0.503
Train Data



=====
Test Data



[6] Conclusions

```
In [135]: # Creating table using PrettyTable library
from prettytable import PrettyTable
vectorizer = ['BOW','TFIDF','Avgw2v','TFIDFW2v']
AUC = [0.83,0.82,0.83,0.79]
Model = ['DecisionTree Classifier','DecisionTree Classifier','DecisionTree Classifier','DecisionTree Classifier']
hyperparameter1= [50,50,10,10]
hyperparameter2= [300,300,200,300]
sno = [1,2,3,4]
# 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(" Best depth ",hyperparameter1)
ptable.add_column(" Minimum samples split ",hyperparameter2)
ptable.add_column(" Test AUC ",AUC)
# Printing the Table
print(ptable)
```

S.NO.	Vectorizer	Model	Best depth	Minimum samples split	Test AUC
AUC					
1	BOW	DecisionTree Classifier	50	300	0.83
2	TFIDF	DecisionTree Classifier	50	300	0.82
3	Avgw2v	DecisionTree Classifier	10	200	0.83
4	TFIDFW2v	DecisionTree Classifier	10	300	0.79

```
In [136]: !jupyter nbconvert --to html AFFRDT.ipynb
```

[NbConvertApp] Converting notebook AFFRDT.ipynb to html
[NbConvertApp] Writing 1121392 bytes to AFFRDT.html

```
In [ ]:
```